

GÉRARD BERRY

L'Hyperpuissance de l'informatique

*Algorithmes, données,
machines, réseaux*

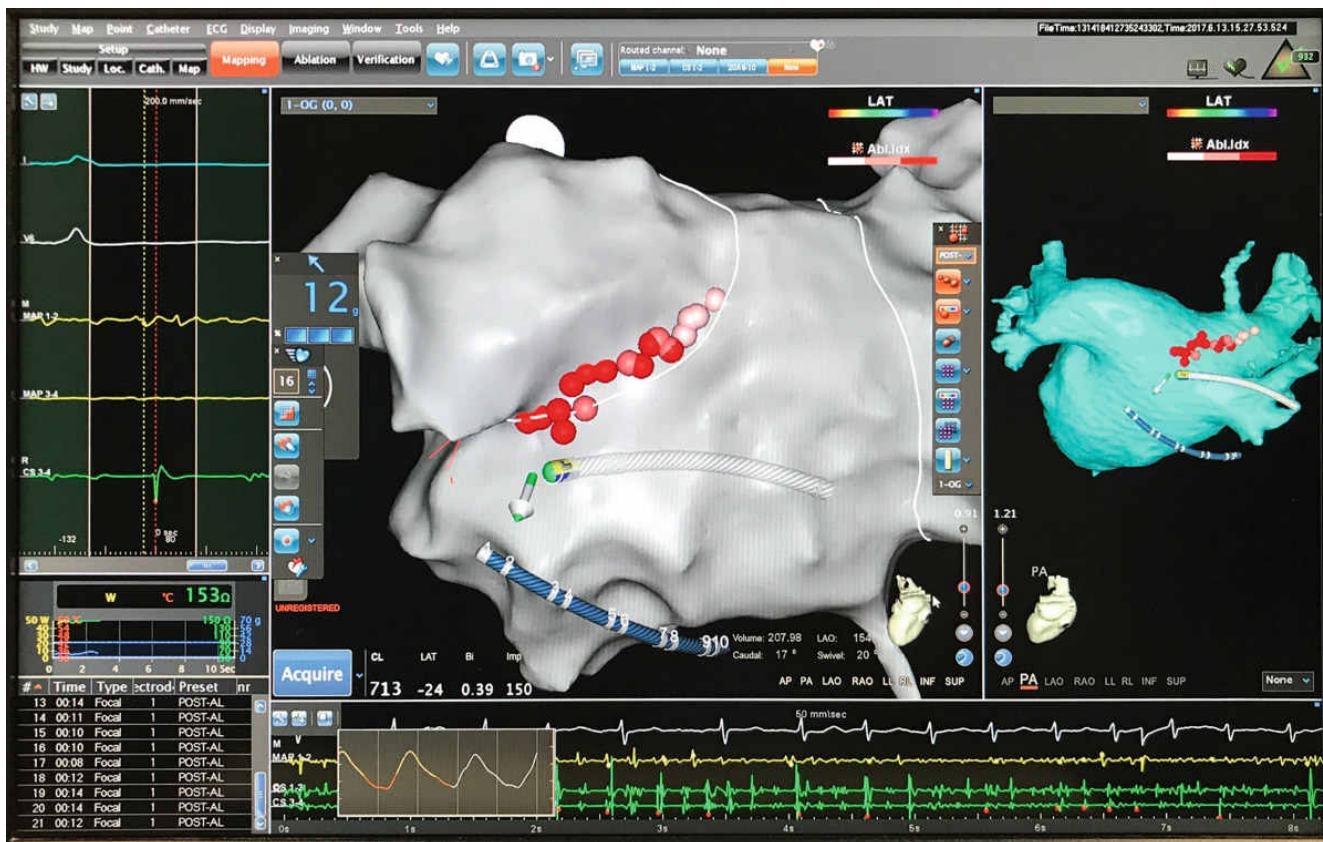


Gérard Berry

L'hyperpuissance de l'informatique

Algorithmes, données,
machines, réseaux





© ODILE JACOB, OCTOBRE 2017
15, RUE SOUFFLOT, 75005 PARIS

www.odilejacob.fr

ISBN : 978-2-7381-3954-2

Le code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5 et 3 a, d'une part, que les « copies ou reproductions strictement réservées à l'usage du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4). Cette représentation ou reproduction donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

Ce document numérique a été réalisé par [Nord Compo](#).

*Je dédie ce livre à Paul Caspi, Philippe Flajolet,
Gilles Kahn et Robin Milner,
grands chercheurs disparus,
mais aussi amis et hommes généreux
avec qui j'ai beaucoup partagé,
dans le travail et ailleurs.*

Avant-propos

Le contenu de ce livre est largement issu des cours que j'ai donnés au Collège de France, d'abord dans le cadre de la chaire annuelle d'Innovation technologique Liliane Bettencourt en 2007-2008 ([Berry, 2008](#)), ensuite dans celui de la chaire annuelle Informatique et sciences numériques créée en partenariat avec l'Inria en 2009-2010 ([Berry, 2009](#)), et enfin sur la chaire de plein exercice « Algorithmes, machines et langages » que j'occupe depuis 2012-2013 ([Berry, 2013](#)). Il a aussi profité des séminaires associés aux cours qu'y ont donnés les collègues que j'y ai invités, des cours d'autres professeurs, et d'un grand nombre de conférences que j'ai données ou auxquelles j'ai assisté dans des milieux des plus variés, allant des grandes instances publiques, scientifiques ou industrielles de tous types jusqu'à des associations locales de curieux en passant par des classes de lycée.

Beaucoup des sujets que je développerai ici ont aussi eu pour moi comme source le Collège de France, à travers les cours des autres professeurs que j'y ai suivis. C'est en particulier le cas pour les instances successives de la chaire annuelle Informatique et sciences numériques, dont les titulaires successifs m'ont beaucoup appris. Je me suis aussi inspiré de ce qu'ont enseigné d'autres professeurs comme Françoise Combes en astronomie, José Sahel pour la vision et bien d'autres. J'ai aussi bien sûr bénéficié de bien d'autres liens avec de nombreux grands scientifiques qui exercent en France ou ailleurs.

C'est une chance extraordinaire que de travailler dans une telle institution, une chance maintenant partageable en vidéo si ce n'est en chair et en os : tous les cours et

séminaires que je mentionnerai sont visibles et téléchargeables sur le site Web du Collège de France, avec leurs supports. J'y ferai de nombreuses références en fournissant des liens « http » directement cliquables sur la version électronique de ce livre.

J'ajoute que beaucoup des idées développées ici et des façons d'en parler sont nées en bonne partie lors d'un enseignement donné à l'école Montessori Les Pouces verts à Mouans-Sartoux, d'abord chez les 9-12 ans puis chez les 6-9 ans. Expliquer le monde aux enfants de cet âge est un exercice difficile. Curieux, ouverts et exigeants, ces enfants et leurs éducateurs ont largement modifié mes façons de comprendre, de faire et de communiquer, pour le mieux j'espère.

J'ai bien sûr aussi bénéficié de dialogues constants avec mes collègues informaticiens ou non, que je remercie vivement. Sans pouvoir les citer tous, je remercie particulièrement Serge Abiteboul, Martin Abadi, Nicholas Ayache, François Baccelli, Francis Besse, Jean-Daniel Boissonnat, François Bourdoncle, Françoise Combes, Gilles Dowek, Patrick Flandrin, Cédric Fournet, Georges Gonthier, Nicolas Halbwachs, Gérard Huet, Michael Kishinevsky, Xavier Leroy, Jean-Jacques Lévy, Jean-Christophe Madre, Philippe Manoury, Laurent Massoulié, Maurice Nivat, Marc Pouzet, Manuel Serrano, Laurent Thénier et Jean Vuillemin. Je voudrais aussi offrir un merci spécial à Frédéric Guichard, qui m'a initié à la photographie algorithmique, et Clément Narteau, qui a fait de même pour la physique des dunes. Enfin, je remercie tout particulièrement mes amis de la Société informatique de France, de l'Inria et de La Main à la pâte qui prennent à bras-le-corps les questions de la diffusion et de l'enseignement de l'informatique.

NOTE. Dans ce livre, j'essaie de couvrir des sujets que je connais un peu mais dont je ne suis pas spécialiste. Il est donc possible que j'aie laissé des erreurs, des contresens, ou des manques vraiment flagrants. Si vous en trouvez, signalez-les-moi s'il vous plaît à l'adresse suivante :

gerard.berry@college-de-france.fr.

CHAPITRE 1

Introduction

La construction d'un nouveau schéma mental

« Rien ne sert de penser, il faut réfléchir avant. »

Pierre DAC.

En janvier 2008, lors de ma première leçon inaugurale au Collège de France intitulée « Pourquoi et comment le monde devient numérique » ([Berry, 2008](#)), je disais que l'informatique serait désormais partout et que son expansion dans tous les secteurs de la société et de la science ne faisait que commencer. C'était précisément l'époque où la société commençait à percevoir confusément ses impacts, surtout à travers la constatation des expansions vertigineuses d'Internet et des téléphones portables. Beaucoup d'acteurs constataient déjà que les communications entre hommes et la structure du travail allaient être profondément modifiées, mais peu s'intéressaient encore à d'autres aspects plus cachés comme l'inéluctable informatisation massive des objets, dont je soulignais la très grande importance à terme, ou les problèmes qu'allaient poser la collecte des données personnelles et la sécurité informatique. On en parlait, mais en se demandant si c'était du lard ou du cochon, comme on dit à la campagne. Je mentionnais aussi une situation étonnante pour moi, qui reste encore largement actuelle : la plupart des acteurs politiques, industriels, médicaux, administratifs et juridiques que je rencontrais semblaient en permanence surpris par des évolutions tout à fait prévues, organisées et annoncées par la recherche et l'industrie informatiques. N'est-il pas surprenant d'être surpris en

permanence par du prévisible explicitement organisé ? Mais, heureusement, la curiosité commençait à apparaître.

Dix ans après, je continue à constater la force de l'informatisation du monde, qui va constamment en s'amplifiant. Mais je constate hélas la persistance de la surprise permanente jointe à certaines peurs, certaines raisonnables et d'autres moins. C'est dû je pense à une compréhension insuffisante par le public général et les décideurs en particulier de la nature même de l'informatique. Or cette compréhension est nécessaire pour bien apprécier les raisons et la dynamique de l'irruption de l'informatique dans la vie de tous les jours, et en particulier dans beaucoup d'activités et de métiers autrefois non touchés. Certes, les choses ont progressé, au moins en surface : les politiques s'y intéressent et essaient enfin d'agir, les médias en parlent bien plus qu'auparavant, et la science et la technique informatiques font partie depuis la rentrée 2016 du programme de l'enseignement général français. Mais, si l'on trouve beaucoup d'articles et d'ouvrages consacrés aux impacts et dangers du « numérique », adjectif maintenant promu en nom commun d'une façon que j'analyserai plus tard, la plupart se contentent d'analyser les effets sans vraiment parler de leurs causes, c'est-à-dire de l'informatique en tant qu'activité propre. Pour progresser dans la compréhension du nouveau monde qui se construit à grande allure, il me semble au contraire indispensable d'analyser bien mieux ces causes : quels sont les sujets d'étude de l'informatique, ses façons de penser et ses façons de faire ? D'où vient son étonnante puissance, et pourquoi gagne-t-elle le monde aussi vite et aussi profondément ? Comment interagit-elle avec les autres activités humaines et avec les systèmes physiques ou mécaniques traditionnels ?

Répondre à ces questions est précisément mon but dans ce livre. J'essaierai de le faire non pas de façon théorique, mais en analysant des exemples pris dans des domaines variés, et en n'entrant dans des détails un peu techniques qu'en de rares occasions. Je ne considérerai mon but comme atteint que si le lecteur parvient à se construire un nouveau *schéma mental* mieux adapté à l'avenir, au lieu d'essayer de faire entrer l'informatique dans les schémas mentaux hérités des siècles précédents – xx^e siècle compris. Pour simplifier la discussion, et même si sa science et sa technologie se sont développées bien avant, je considérerai que l'impact généralisé de l'informatique n'a vraiment commencé qu'autour de l'an 2000 ; ce n'est pas faux quand on regarde son évolution et le nombre d'activités concernées.

Le schéma mental de l'informatique

Mon leitmotiv sera qu'on ne peut pas comprendre les bouleversements provoqués par l'informatique en restant dans les schémas mentaux traditionnels issus des sciences et techniques directement liées au monde physique. En effet, elle diffère profondément de toutes les activités scientifiques et techniques précédentes du fait même de ses objets d'étude et de ses méthodes d'action : l'informatique *calcule sur l'information* à l'aide d'*algorithmes*, de *programmes* et de *machines*, essentiellement des ordinateurs de toutes sortes. L'information est codée dans des *données* numériques, l'algorithme est le mécanisme conceptuel de calcul systématique, le programme constitue l'écriture précise de l'algorithme dans des langages appropriés, et la machine est l'objet matériel capable de faire les calculs nécessaires pour transformer les programmes en actions. Un point absolument essentiel est que la nature physique des objets matériels qui servent à stocker les données et faire les calculs est indifférente : la même information peut être stockée sur n'importe quel support, et n'importe quel ordinateur peut faire les mêmes calculs que n'importe quel autre. Aucun de ces objets conceptuels et pratiques ne ressemble donc à ceux des siècles précédents, tous pour la plupart reliés bien plus intimement au monde physique. Mais l'informatique n'est pas qu'une histoire d'hommes et de pensée. Elle est très souvent en contact direct avec le monde physique, par exemple pour le mesurer et le contrôler, et tout autant avec le monde du vivant à travers les nouvelles avancées qu'elle permet en médecine et en biologie. La dématérialisation n'empêche pas le contact avec la matière ; au contraire, elle peut l'enrichir.

Structure du livre

Dans la première partie, aux [chapitres 2 et 3](#), j'expliquerai la nature de ce nouveau schéma mental et sa mise en œuvre pratique. Dans la deuxième partie, des [chapitres 4 à 8](#), j'étudierai de façon approfondie cinq exemples pris dans des domaines variés de la vie courante, autrefois très indépendants les uns des autres : télécommunications, Internet, photographie et cartographie numériques, médecine, sciences naturelles et mathématiques. Cette énumération n'a aucunement l'intention d'être exhaustive, mais elle touche aux limites de mon champ personnel de compétences et suffira je pense à illustrer mon propos. J'insisterai sur le fait que l'informatique est *la même partout*. Pour elle, il n'y a pas de différence fondamentale entre un système de gestion bancaire, un appareil photo numérique, une expérience de physique, un dispositif médical ou un réseau social. Même si leurs détails sont bien sûr différents, tous procèdent du même type de pensée et d'action et demandent désormais une formation générale commune.

Comme je n'ai pas du tout l'intention de présenter une vision bêtement idyllique de l'informatique, je consacrerai la troisième partie à deux de ses dangers intrinsèques, qui sont majeurs en pratique mais restent souvent méconnus ou sous-estimés du public. Le [chapitre 9](#) sera consacré aux *bugs* qui cassent nos systèmes et aux *trous de sécurité* qui permettent aux intrus d'y pénétrer. Le [chapitre 10](#) présentera les méthodes modernes pour les contrôler ou les éviter, qui forment un pan majeur de la grande science informatique à laquelle j'ai consacré ma carrière. La dernière partie, constituée du seul [chapitre 11](#), présentera quelques projections personnelles (et

modestes) sur les développements et impacts futurs. Enfin, des annexes préciseront dans le [chapitre 12](#) quelques points scientifiques et techniques pour le lecteur curieux.

Sauf en ce qui concerne l'enseignement, je serai peu disert sur des questions que traitent la plupart des articles ou livres actuels destinés au grand public, c'est-à-dire liées aux impacts sociaux de l'informatique à travers Internet, aux réseaux sociaux, à la propriété des données, à l'éthique des algorithmes, etc. Ces ouvrages discutent certes de questions sociétales et éthiques intéressantes, mais le plus souvent sans vraiment entrer dans les causes. Une exception notable est le beau livre *Le Temps des algorithmes* écrit par mes collègues et amis informaticiens Serge Abiteboul et Gilles Dowek ([Abiteboul & Dowek, 2017](#)), dont je recommande chaudement la lecture. Je n'aurais pas grand-chose à ajouter à son contenu. Et je recommande aussi la lecture du blog « Binaire¹ » du journal *Le Monde*, coordonné par Serge Abiteboul avec la Société informatique de France (SIF).

Informatique ou numérique ?

Avant d'entrer dans la chair des chapitres, je souhaite continuer une discussion un peu générale dans cette introduction.

On a vu se produire récemment deux glissements de vocabulaires dans les médias, les discours politiques et l'enseignement : « informatique » est devenu « numérique » et « programmation » est devenu « codage », les deux anciens mots ayant pratiquement disparu. C'est surtout sensible depuis qu'on parle de plus en plus du sujet, en reconnaissant maintenant qu'il faut « y aller » : il faut enseigner « le numérique » (maintenant devenu un substantif), et les enfants doivent apprendre « le code ». D'où vient ce glissement ? Mon idée personnelle, peut-être un peu brutale, est la suivante : changer les mots a permis à ceux qui le souhaitent de se construire des formes de compétence, d'adhésion ou de rejet sans avoir à entrer dans le cœur du sujet, donc en gardant l'intention de ne se renseigner vraiment ni sur l'informatique, ni sur la programmation. Un bon exemple est donné par les très nombreux hommes politiques et commentateurs de la politique qui associent systématiquement numérique avec Internet, les réseaux sociaux et maintenant l'intelligence artificielle. Ce sont certainement des sujets très visibles et très importants, surtout depuis que les réseaux sociaux jouent un rôle essentiel dans les discussions politiques et la transmission des rumeurs et fausses nouvelles de tous types. Mais l'informatique est beaucoup plus que cela, comme nous allons le voir tout au long du livre. Puisque mon but est précisément de montrer à l'inverse qu'on ne peut pas comprendre le monde numérique dans sa totalité sans comprendre suffisamment ce qu'est son cœur informatique, je ferai très attention aux mots pour que l'arbre ne cache pas la forêt.

Le mot « numérique » a pour moi un sens précis que je souhaite conserver. Je l'utilisais comme adjectif dans le titre « Pourquoi et comment le monde devient numérique » de ma leçon inaugurale de 2008 au Collège de France, où j'expliquais comment la numérisation du monde et l'utilisation des algorithmes, programmes et ordinateurs conduisaient à une modification profonde du monde. Ce mot me permettait aussi d'inclure d'autres communautés scientifiques et techniques. Les mathématiciens appliqués qui montrent comment résoudre des équations complexes à l'aide de grands calculs numériques sont appelés depuis toujours des « numériciens ». La science qui invente les algorithmes de contrôle des avions, satellites, voitures ou robots est appelée l'automatique ; si ces algorithmes étaient autrefois réalisés par des machines analogiques, ils le sont désormais par numérisation des données et calcul à l'aide de circuits et logiciels informatiques qui permettent beaucoup plus de souplesse et évitent complètement l'accumulation du bruit qui limitait les machines analogiques. Le traitement du signal, fondamental dans des domaines aussi divers que la musique et l'analyse des images, des tremblements de terre ou des signaux émis par le cerveau, est un domaine en soi. Tous ces domaines et bien d'autres sont devenus numériques car ils s'appuient sur l'informatique pour leurs réalisations. De même, l'expression « économie numérique », maintenant d'usage constant, est justifiée par les faits, alors que l'expression « économie informatique » ne serait pas appropriée.

Je continuerai donc à m'appuyer sur les définitions que mes collègues et moi-même avons utilisées lors de la rédaction du rapport de l'Académie des sciences *L'Enseignement de l'informatique en France. Il est urgent de ne plus attendre* ([Académie des sciences, 2013](#)), que j'ai eu l'honneur de coordonner :

1. Le mot « informatique » désignera spécifiquement la science et la technique du traitement de l'information, et, par extension, l'industrie directement dédiée à ces sujets.

2. L'adjectif « numérique » peut être accolé à toute activité fondée sur la numérisation et le traitement de l'information : photographie numérique, son numérique, édition numérique, sciences numériques, art numérique, etc.

On parle ainsi de « monde numérique » pour exprimer le passage d'un nombre toujours croissant d'activités à la numérisation de l'information et d'« économie numérique » pour toutes les activités économiques liées au

monde numérique, le raccourci « le numérique » rassemblant toutes les activités auxquelles on peut accoler l'adjectif numérique. Puisque toute information numérisée ne peut être traitée que grâce à l'informatique, celle-ci est le moteur conceptuel et technique du monde numérique. Par rapport à l'anglais, notre acception du mot « informatique » recouvre *computer science*, *information technology* et ce que l'on entend souvent par *informatics*, alors que l'adjectif « numérique » correspond à *digital*, par exemple dans la correspondance entre « monde numérique » et *digital age*.

Pour ce qui est de « programmation » et « codage », je garderai aussi le mot initial, bien que les informaticiens utilisent volontiers le mot « code » : *montre-moi ton code* est une expression très répandue. On dit cependant « langage de programmation » et pas « langage de codage », et « code » est ambigu car il a d'autres sens assez différents, comme « le code secret Enigma » pour le chiffrement allemand pendant la Seconde Guerre mondiale. Mais les informaticiens aussi changent les mots. Maintenant, on ne programme plus, on « développe », et les programmeurs sont devenus des développeurs. À suivre...

Enfin, je me tiendrai loin de la récente et étonnante formule à la mode « le logiciel des hommes politiques ». Quiconque sait vraiment ce qu'est un logiciel espère que les hommes politiques ont bien plus que du logiciel dans la tête, et que leur taux de bugs est moins élevé que celui de beaucoup de logiciels...

La pensée algorithmique

La communauté des informaticiens est unanime : le cœur du nouveau schéma mental réside dans une nouvelle façon de penser, la *pensée algorithmique*, qui est quasi indépendante du domaine d'application, et qui conduit à de nouvelles formes d'action dont les implications sont nombreuses et profondes. Cette pensée met en avant les notions d'information et d'algorithme, sans pour autant être éthérée puisque ses applications sont la plupart du temps dans la vie matérielle. Le leitmotiv sur lequel mon discours sera construit est simple :

Avec l'information et par le calcul sur elle, on peut faire assez simplement des choses extraordinaires qu'il est très difficile, voire impossible, de faire avec les moyens physiques traditionnels.

Ce n'est pas la première fois que cette maxime s'applique : avec le télégraphe et le téléphone, les télécommunications ont déjà montré le pouvoir de la transmission rapide de l'information entre les hommes. Mais mes exemples montreront je l'espère que l'informatique va beaucoup plus loin, car elle ne se contente pas de mettre en communication les hommes entre eux : elle interagit aussi de façon uniforme avec une très grande quantité de processus physiques traités autrefois indépendamment les uns des autres. Certains exemples exhiberont même ce que j'appelle des *inversions mentales* : placer la pensée algorithmique avant la pensée physique conduit à penser et

à agir à l'envers de ce qu'on faisait avant. Voyons-en quelques exemples tout de suite, parmi lesquels certains seront davantage détaillés dans la suite.

PREMIERS EXEMPLES D'IMPACTS DE LA PENSÉE ALGORITHMIQUE

Sur les plans industriel et sociétal, on voit de plus en plus de métiers bien établis se faire concurrencer, voire laminer par de nouveaux entrants, qui comprennent qu'il est plus important de posséder et de canaliser l'information que de fabriquer ou posséder les objets physiques comme dans les activités traditionnelles. Les exemples récents crèvent les yeux dans les services, domaine où la prééminence prise par l'information sur la matière se généralise.

Les taxis ont été traditionnellement organisés autour des biens matériels que sont les voitures et, pour améliorer leur chiffre d'affaires, ils ont organisé des centres d'appel et maintenant des sites Web ou applications de téléphone pour trouver les clients. Mais le point central reste pour eux la voiture, dont on paye aussi (parfois cher) le trajet d'arrivée au rendez-vous quand on la réserve à distance. À l'opposé, les entreprises de VTC considèrent que la valeur la plus importante est celle de l'information, pas celle de la voiture. L'information est légère et permet de mettre en avant le client, alors que la voiture qui coûte cher peut être sous-contractée. L'essentiel est de savoir à tout moment et avec précision qui veut aller d'un point à un autre et où se trouve chacune des voitures, puis d'en informer en temps réel à la fois les clients et les chauffeurs. Ces informations permettent à ceux qui les possèdent de simplifier et d'automatiser l'allocation des voitures aux clients, sans besoin d'intermédiaire humain. Elles simplifient aussi la facturation, le montant de la facture étant même connu dès que la course est acceptée. Elles permettent enfin aux clients de noter les chauffeurs et réciproquement, ce qui est impensable pour les taxis (et pose d'autres questions), de comprendre les comportements de chaque utilisateur pour pouvoir lui personnaliser les offres, puis d'analyser les comportements collectifs par agrégation statistique. Pour réussir cela, il faut que les sites Web et les applications de téléphone soient couplés au GPS et surtout très agréables à utiliser, afin de rendre l'interaction bien plus simple que la gestion lourde d'appels téléphoniques ou de sites

Web classiques. Les taxis pourront bien sûr faire la même chose, mais cela nécessitera pour eux un changement de culture complet – et rien n’est plus difficile qu’un changement de culture.

Pour la même raison, les sites de réservations d’hôtels ou d’appartements prennent de plus en plus de pouvoir sur les agences de voyages traditionnelles en se contentant de recueillir et d’organiser l’information à très grande échelle, et en particulier en affichant les retours et recommandations des utilisateurs qui finissent par diriger le marché eux-mêmes en temps réel (sujet pas simple non plus car il n’est pas difficile de fausser ces retours).

Mais il y a un gros ticket d’entrée : dans toutes ces activités, il est facile d’essayer mais très difficile de réussir. L’informatique est *intrinsèquement difficile*, au moins si on essaie de construire des applications qui marchent *vraiment* avec de *vraiment* bonnes ergonomies. Cela demande en premier d’avoir le bon schéma mental.

LA CONCEPTION ET L’INFORMATISATION DES OBJETS INDUSTRIELS

Une évolution d’un autre ordre est à l’œuvre dans l’industrie classique : pour la plupart des objets industriels, la conception assistée par ordinateur (CAO) jointe à la modélisation et à la simulation numérique remplace les techniques classiques de conception par prototypes physiques. Et, dans des cas de plus en plus nombreux, les objets peuvent être fabriqués directement par des imprimantes 3D (qui peuvent maintenant imprimer même des maisons !).

À l’intérieur même des avions, trains et voitures, de plus en plus de fonctions autrefois purement mécaniques sont informatisées pour plus d’efficacité et de souplesse. Prenons le cas de la voiture. Qui sait que la réduction de consommation par rapport à l’époque de la pure mécanique est due essentiellement à la conception numérique des moteurs et à la gestion informatique fine de l’injection et de l’allumage ? Qui sait que le freinage impeccable sur tous les terrains est pour une large part une question d’informatique ? De plus, maintenant que toutes les voitures sont mécaniquement efficaces et peu distinctes, les constructeurs et vendeurs portent massivement l’accent marketing sur les services informatisés qu’ils peuvent

rendre à leurs conducteurs et passagers, soit directement dans la voiture, soit à travers des connexions Internet passant par les réseaux sans fil. Enfin, les clients se renseignent autant sur le Web par les sites de test et de comparaison qu'en venant au garage discuter avec un vendeur. Pourtant, nous allons voir que l'informatique automobile est loin d'être optimale, voire maîtrisée, en particulier parce que les constructeurs ont souhaité rester le plus longtemps possible dans leur schéma mental mécanique traditionnel et sous-traiter l'informatisation de leurs voitures.

De même, immeubles, autobus, trains, avions, vélos, prothèses médicales et même villes entières sont de plus en plus informatisés et connectés pour leur propre fonctionnement, conduisant à ce qu'on appelle l'*Internet des objets* (*Internet of things* ou IOT en anglais). Par exemple, pour gérer correctement les feux rouges d'une ville ou les réseaux électriques hétérogènes du futur, la collecte et l'exploitation d'informations deviendront au moins aussi importantes que la construction des infrastructures. Ce nouvel Internet va vite devenir bien plus grand que le nôtre, à condition toutefois que les questions de sûreté (bon fonctionnement dans toutes les circonstances) et de sécurité informatique (résistance aux intrusions et au vol de données) soient prises beaucoup plus au sérieux que maintenant. Pourquoi ne le sont-elles pas encore ? Encore une fois parce que beaucoup d'acteurs traditionnels ne comprennent pas que les attaques informatiques sont très différentes des attaques physiques. Toujours ce satané schéma mental ! Un joli exemple de fonctionnement étrange est fourni par l'affichage en temps réel des horaires de bus dans Paris, dont je me sers souvent. Il est fait de deux façons : directement par affichage sur les arrêts d'autobus, ou par envoi sur les téléphones portables. Les résultats sont souvent différents, le téléphone étant généralement gagnant. Étonnant, non ?

LA DÉMATÉRIALISATION

Les exemples précédents concernaient le couplage entre matière et information. D'autres domaines vont encore plus loin en passant à la *dématérialisation* totale, qui est en pleine expansion pour la photo, le son et la vidéo. La photo numérique se prend, s'envoie et se regarde avec des téléphones portables, le nombre d'images prises ayant été multiplié par un facteur incroyable en très peu de temps. La conception de

ces photophones et plus généralement des appareils photo numériques est complètement différente de celle des appareils argentiques traditionnels, comme nous le verrons au [chapitre 6](#). La musique s'écoute aussi sur son téléphone ou sur son ordinateur lié à sa chaîne hi-fi à partir de serveurs situés n'importe où dans le monde ; elle se diffusera bientôt à l'aide d'excellentes enceintes sans fil connectées à n'importe quel appareil par des protocoles informatiques. Les films se projettent dans des cinémas qui les téléchargent en format numérique, ou chez soi en se connectant par Internet à des sites dédiés.

Il est très important de garder des vinyles, des bandes magnétiques, des compact-discs, des bobines de films et des DVD pour les montrer dans des musées aux enfants nés dans ce millénaire.

LES SCIENCES ET LA MÉDECINE

Enfin, les sciences et la médecine aussi sont touchées. Les chercheurs des sciences naturelles passent désormais une grande partie de leur vie devant un ordinateur et voient leur métier se transformer par l'appareillage informatisé, l'analyse des données massives, la modélisation et la simulation numériques. Cela souligne aussi les liens forts qui font souvent se rejoindre informatique et mathématiques, qui doivent absolument coopérer dans ces domaines. Les radiologues interventionnels réalisent des opérations mini-invasives du cœur, du cerveau et d'autres organes en en faisant entrer leurs instruments par de toutes petites incisions puis en les en guidant par l'imagerie médicale en direct, voire sous réalité virtuelle, etc. Nous parlerons de tout cela plus longuement aux [chapitres 7 et 8](#).

Les dangers de l'immobilité

Malheureusement, la France et d'autres pays européens ont longtemps fait l'impasse sur l'informatique et opté pour l'immobilité, sauf dans des secteurs particuliers dont fort heureusement la recherche et l'enseignement supérieur. Il me paraît utile d'analyser cette situation pour qu'elle ne se reproduise pas : ceux qui ont fait exploser l'informatique sont ceux qui avaient décidé il y a longtemps de ne pas être immobiles, en particulier aux États-Unis et en Asie. Comme je ne connais vraiment que la France, je ne parlerai que peu du reste de l'Europe, dont je ne pense pas qu'il ait évolué très différemment.

UNE MODE QUI VA PASSER

Dans les années 1970-1980, j'ai souvent entendu ces deux phrases : « L'informatique n'est qu'un outil », sous-entendu pas une science, et « c'est une mode qui va passer² ». Au conseil d'enseignement d'une grande école célèbre, j'ai même entendu : « Si ce n'est pas une science, il ne faut pas l'enseigner. » Cette phrase a eu la vie dure, l'informatique n'entrant dans les grandes écoles que bien plus tard qu'à l'université. Les choses ont heureusement bien changé, et l'informatique est maintenant une matière majeure dans beaucoup d'entre elles.

LE RETARD INDUSTRIEL

Mais cette façon de voir a eu des impacts délétères sur notre industrie, car ce sont les grandes écoles qui dominent le paysage industriel français. Avec des actions volontaristes et soutenues dans le temps, le pays a considérablement investi et réussi scientifiquement et techniquement dans la production et la distribution d'électricité, les transports routiers, ferroviaires et aériens ou le spatial. Mais ça n'a pas été le cas dans l'industrie informatique, jamais considérée au même niveau d'importance, même lorsqu'elle a explosé aux États-Unis. L'effet est bien visible maintenant. En termes de capitalisation, cinq des dix plus grandes entreprises mondiales tous secteurs confondus sont à ce jour des entreprises informatiques, dont les trois premières : 1. Apple ; 2. Alphabet, maison mère de Google ; 3. Microsoft ; 6. Facebook ; 9. Amazon. De son côté, la France ne compte qu'une entreprise informatique dans les cent premières mondiales, Dassault Systèmes, qui occupe une position dominante dans la conception assistée par ordinateur des grands systèmes. Les sociétés de services ont relativement mieux réussi, mais elles ne font quasiment pas de développement et ne jouent donc pas dans la même cour.

L'informatique détruit des emplois comme l'ont fait les autres révolutions industrielles, c'est clair et ce n'est pas simple à vivre et à gérer. Mais on réalise encore peu à quel point elle est aussi une grande source d'emplois à travers le monde. Je ne citerai qu'un exemple. La société de conseil informatique TCS (Tata Consultancy Services) est la plus grande en Inde, suivie de près par InfoSys. Quand je l'ai connue au début des années 1990, elle avait quelques milliers d'employés ; en l'an 2000, 65 000, puis 135 000 en 2004. En 2012-2013, TCS a recruté presque 70 000 personnes, soit plus que la production annuelle d'ingénieurs en France. Fin 2015, son nombre d'employés était de 378 000, dans 45 pays, avec un âge moyen de 28 ans. Ces nombres sont je pense édifiants. Mais le pays est grand et la formation à l'informatique y est considérée comme primordiale, ce qui n'est pas le cas ici.

LA RECONNAISSANCE TARDIVE DES PIONNIERS

En comparaison avec la reconnaissance immédiate des découvertes des physiciens, biologistes et médecins, immédiatement relayées par les scientifiques et les médias, la reconnaissance publique de celles des pionniers de l'informatique a été

bien tardive. Elle n'a réellement commencé chez nous qu'en 2012, année du centenaire de la naissance d'Alan Turing, immense figure du xx^e siècle universellement considérée comme ayant créé les fondements de l'informatique en 1936³. Puis, en 2016, on a fêté le centenaire de la naissance de Claude Shannon, qui a créé la *théorie de la communication* en 1954, fondement des télécommunications modernes, qui allait ensuite se transformer en une *théorie de l'information* bien plus vaste et dont les impacts augmentent de jour en jour. Shannon était resté inconnu du public jusqu'en 2016.

Les choses vont donc beaucoup mieux ces dernières années, mais il reste un grand chemin à faire : si les prix Nobel et les médailles Fields et prix Abel en mathématiques sont immédiatement relayés par les médias, ce n'est toujours pas le cas du prix Turing, que bien peu connaissent. J'en nommerai beaucoup de récipiendaires dans la suite.

L'ENSEIGNEMENT À LA TRAÎNE

Une autre conséquence délétère se voit dans notre système d'enseignement général, celui qui forme les adultes de demain. Des cours optionnels de programmation des ordinateurs avaient été introduits en 1982, supprimés et réintroduits à deux reprises, puis définitivement supprimés en 1997, précisément à l'époque de l'explosion d'Internet et du téléphone portable – quelle prescience ! Ensuite, on a longtemps professé que, puisque l'informatique n'était qu'un outil, il suffisait d'enseigner les usages, c'est-à-dire comment utiliser un ordinateur et quelques logiciels de bureautique. Comprendre ce qu'il y avait derrière n'avait pas d'intérêt. Cette opinion a eu comme conséquence brutale de mettre les élèves dans la position servile de simples utilisateurs de ce qui est fait et conçu ailleurs, au lieu de leur permettre de faire partie des créateurs du monde de demain. Elle n'a pu être fondée que sur un aveuglement obstiné : pour qui sort un peu de son périmètre immédiat, les transformations du monde étaient déjà bien visibles. Comme le professait Maria Montessori au milieu du xx^e siècle, « n'élevons pas nos enfants pour le monde d'aujourd'hui. Ce monde aura changé lorsqu'ils seront grands. Aussi doit-on en priorité aider l'enfant à cultiver ses facultés de création et d'adaptation ».

Une autre mauvaise idée a beaucoup traîné, hélas avec succès : donner des tablettes aux enfants en admirant leur ingéniosité naturelle à s'en servir, comme de beaucoup d'autres choses d'ailleurs, ou encore des ordinateurs portables aux collégiens sans aucun suivi pédagogique. Beaucoup de parents ont encore l'impression que leurs petits génies apprennent l'informatique comme ça, et beaucoup de professeurs semblent également penser que ce n'est du coup pas utile de leur enseigner. Or c'est simplement faux, très faux, et maintenant bien dénoncé par les psychologues : la tablette ou le téléphone peuvent devenir addictifs et ne conduisent pas à une vraie compréhension, l'enseignement aux petits devant absolument être multisensoriel et fondé sur la relation avec les autres. Parents et enseignants ne sont-ils pas en train de compenser avec leurs petits leur propre ignorance du sujet ? Dans tous les cas, l'affirmation de Maria Montessori est tout à fait pertinente pour l'informatique : à cause d'elle, les enfants de tous les âges auront un schéma mental tout à fait différent de celui de leurs parents, ce qui est bien sûr déjà arrivé dans le passé. La génération de mes parents était friande de l'expression : « Il leur faudrait une bonne guerre », mais ce schéma mental ne m'a jamais convaincu.

Après des efforts tout aussi obstinés des informaticiens et d'associations de professeurs⁴ dans les dix dernières années conduisant au rapport de l'Académie des sciences dont j'ai déjà parlé ([Académie des sciences, 2013](#)), les choses ont commencé à vraiment changer en 2016 avec la mise en place d'un véritable enseignement d'informatique allant de la primaire au baccalauréat⁵ dont les programmes ont été définis de manière réellement collaborative par des enseignants, des chercheurs et le CSP (Conseil supérieur des programmes). Cette mise en place, certes psychologiquement marquante, reste cependant très timide, d'autant plus que la formation des enseignants devra apparemment se faire avec quasiment aucun moyen et qu'il n'est pas prévu que les enseignants formés reçoivent les mêmes diplômes que pour les autres matières. Garder cette position condamnerait de façon certaine l'informatique à rester une matière au mieux secondaire.

La question est simple : pouvons-nous accepter qu'un élève arrive au baccalauréat sans formation réelle à l'informatique, alors qu'il sera en contact direct et permanent avec elle dans ses vies personnelles et professionnelles ? La réponse doit être non, et les moyens d'y parvenir doivent être alloués. Rendez-vous au [chapitre 12](#) pour des propositions.

Heureusement, les esprits changent

En quelques années, les choses ont heureusement bien changé, et l'informatique – pardon le numérique – est devenue un sujet de premier plan dont tout le monde parle, même si nous avons abandonné l'idée de faire partie des très grands du domaine industriel. Internet s'installe partout, bien qu'il faille encore beaucoup de travail pour que les campagnes soient équipées ; les chercheurs s'intéressent de plus en plus aux applications dans des domaines toujours plus nombreux ; les start-up naissent et grandissent, maintenant bien mises en avant et aidées ; les journaux deviennent numériques, les sites Web fleurissent de partout et l'enseignement bouge. Comme il n'est jamais sûr dans ce domaine que les positions acquises et assises le resteront toujours, beaucoup de nouvelles possibilités s'ouvrent.

UNE RECHERCHE DE POINTE AU PLAN MONDIAL ET ASSOCIÉE AUX START-UP

Nous pourrions pour cela nous appuyer sur notre recherche scientifique, qui est de très haut rang mondial depuis toujours, ainsi que sur notre enseignement supérieur. L'informatique est entrée tôt dans l'enseignement et la recherche universitaire, dès les années 1960, mais plus tardivement dans les grandes écoles qui se sont montrées beaucoup plus conservatrices. Mais la différence a largement disparu. La création de l'IRIA (Institut de recherche en informatique et automatique, devenu ensuite Inria) en 1967 par le grand mathématicien Jacques-Louis Lions a joué un rôle fondamental

dans l'expansion d'une recherche de très haute qualité internationale allant du fondamental à l'appliqué. L'expansion continue du domaine au CNRS a tout autant contribué à solidifier le paysage, culminant avec la création de l'INS2I (Institut des sciences de l'information et de leurs interactions) en 2009. De plus les acteurs des différentes universités et instituts ont appris à travailler beaucoup plus en commun. Tout cela fournit un champ de compétences sur lequel notre développement pourra s'appuyer.

LA PARITÉ EST HÉLAS TRÈS, TRÈS LOIN. POURQUOI DONC ?

Il reste cependant un problème majeur à résoudre : la très faible présence des femmes dans tout ce qui est lié à l'informatique. Le domaine est vraiment très loin de la parité, que ce soit dans la recherche, l'industrie ou l'enseignement. Je n'ai évidemment qu'une analyse partielle du sujet et pas de solution, mais je souhaite rapporter quelques expériences. J'ai par exemple entendu en 2015 deux jeunes femmes qui venaient de passer leur baccalauréat me dire que les conseillers pédagogiques de leurs lycées leur avaient clairement déconseillé d'aller vers l'informatique « car ce n'est pas un métier pour les femmes », alors même que les deux étaient vraiment tentées. Il y a deux drames conjoints : le premier est qu'on le leur fasse croire, le deuxième qu'elles le croient. L'une étudie le droit, mais a bien l'intention de revenir vers l'informatique ; je lui ai répondu qu'acquérir une double compétence était une excellente idée, car le droit de l'informatique est un sujet naissant, passionnant, et pas près de devenir obsolète. L'autre s'est dirigée vers la biologie, sujet tout aussi fascinant où les femmes sont à l'inverse largement majoritaires, mais où les débouchés sont hélas bien plus aléatoires. J'ai interrogé depuis d'autres jeunes chercheuses d'autres disciplines. J'en ai retiré l'impression que même celles qui étaient intéressées ne s'étaient pas senties assez bonnes en maths pour aller vers ce genre de sujet, ce qui est idiot mais rarement démenti par leur environnement, et qu'on leur conseillait plutôt les sciences de la vie. Un autre souci bien plus compréhensible est qu'elles ne se sentaient pas attirées par un métier massivement masculin – voire elles étaient repoussées par l'image du *geek* mâle, si

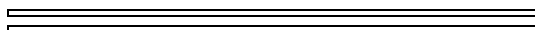
bien publicisée et pourtant très minoritaire dans les métiers de l'informatique. La même chose se produit apparemment dans presque tous les pays occidentaux.

Il faut tordre le cou à ce genre d'idioties, et le plus vite sera le mieux. Notre communauté se bouge beaucoup pour changer cet état de choses, mais sans grand succès jusqu'à présent. Toutes les idées et actions sont bienvenues.

Chères lectrices et lecteurs, il est temps maintenant de passer au cœur du sujet.

PREMIÈRE PARTIE

Du matériel à l'immatériel



CHAPITRE 2

Les grands principes

Avant de parler d'information et d'informatique, il me paraît utile de rappeler brièvement quelle était la pensée scientifique et technique dominante au xx^e siècle. Cela nous permettra de comprendre pourquoi l'informatique n'en est vraiment pas une conséquence directe, ce qui est probablement une des raisons pour lesquelles elle n'a pas été facilement acceptée par les communautés scientifiques de l'époque.

Le triangle du xx^e siècle

Nos sens nous donnent immédiatement accès à deux aspects de la nature : *l'espace* et *le temps*, que nous pouvons qualifier et quantifier assez directement de façon plus ou moins précise suivant les besoins. Nous évoluons dans l'espace, et tel arbre est perçu comme plus ou moins loin. Le soleil se lève tous les matins, et les plantes, les animaux et les hommes vieillissent de façon bien visible. L'espace et le temps font l'objet de spéculations religieuses, philosophiques puis scientifiques depuis la nuit des temps. Au xx^e siècle, les physiciens nous ont expliqué qu'espace et temps sont bien plus subtils qu'il n'y paraît, et que plus le temps passe moins ils comprennent quelle est sa nature. Mais, sauf dans le [chapitre 8](#) où nous parlerons de simulation des phénomènes physiques, nous en resterons à nos impressions macroscopiques et ne déborderons pas de la physique newtonienne – celle qu'on apprend à l'école, avec un temps linéaire et un espace à trois dimensions.

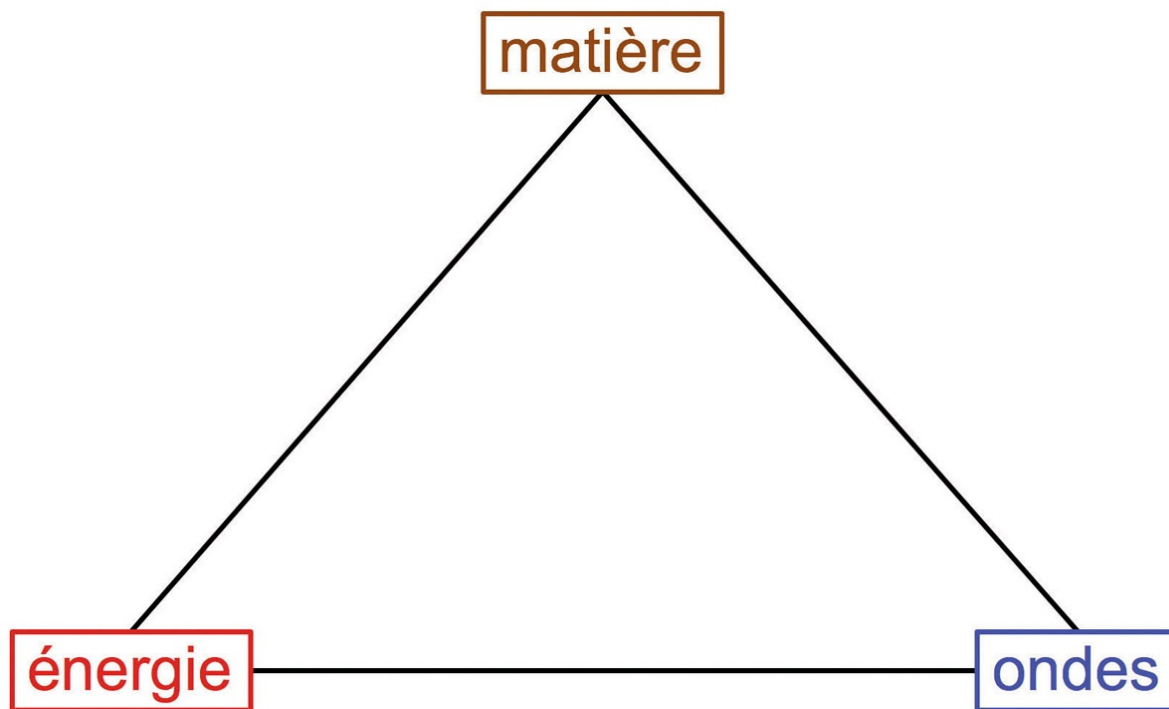


Figure 1. Le triangle du XX^e siècle.

La pensée scientifique et technique s'est petit à petit structurée autour du triangle matière-énergie-ondes, illustré par la figure 1, en commençant par le couple matière-énergie. La matière est un aspect de la nature que nous percevons très directement : un objet solide a une forme bien définie, est lourd ou léger, régulier ou non ; un liquide mouille et prend la forme des récipients, alors qu'un gaz ne se voit pas mais fait du vent et peut sentir bon, mauvais, ou pas du tout. L'énergie reste directement perceptible, bien que plus complexe : on sent bien que monter une côte est plus fatigant que la descendre, mais dire de combien n'est pas simple. Pour les Grecs, cette perception duale conduisait à penser que notre univers était fait de quatre éléments : la terre, l'eau et l'air pour la matière, et le feu pour l'énergie. Cette vision sommaire s'est évidemment perfectionnée petit à petit, et l'explosion des sciences du XVII^e au XIX^e siècle est précisément due à une meilleure compréhension des relations entre matière et énergie. C'est ce qui a conduit aux moteurs, cœurs de la grande révolution de l'industrie et des transports aux XVIII^e et XIX^e siècles, ainsi qu'aux débuts de la chimie.

Vers le milieu du XIX^e siècle, la compréhension de plus en plus fine de l'électricité, de l'électromagnétisme, de l'acoustique et des vagues sur la mer a conduit à la mise en avant des *ondes*. Cela a provoqué les grandes révolutions de la communication à distance avec le télégraphe électrique (vers 1840), le téléphone (vers 1870), l'éclairage électrique (1879) puis la radio (vers 1890) et la télévision (vers 1925). Ces nouveautés ont conduit à des changements très profonds dans l'organisation de la vie de tous les jours et celle de la société. Qu'on se rappelle simplement que les messages entre l'Angleterre et les Indes ou l'Amérique mettaient désormais quelques secondes au lieu de quelques mois ([Standage, 2014](#)), que la nuit n'a plus été l'opposé absolu du jour, et que les ombres sont devenues fixes au lieu de bouger constamment à cause de l'instabilité des flammes¹.

Au XX^e siècle, la pensée scientifique et technique est restée largement ancrée sur ce triangle matière-énergie-ondes. Les sciences physiques, qui ont fondé les progrès technologiques majeurs de ce siècle, ont profité d'une modélisation mathématique efficace, alors que les sciences de la vie sont restées plutôt descriptives.

INFORMATION ET ALGORITHMES : DU TRIANGLE AU TÉTRAÈDRE

À partir du premier tiers du XX^e siècle, deux notions bien différentes ont été lentement théorisées puis rapidement mises en pratique à la fin du siècle : celles d'*information* et de *calcul algorithmique*, qui fondent les sciences et techniques informatiques auxquelles cet ouvrage est consacré. Cela a conduit à étendre le triangle en un tétraèdre illustré par la figure 2, avec les conséquences considérables que nous observons maintenant. Dans la figure, j'ai volontairement placé l'information au centre, car c'est son exploitation systématique à l'aide de l'ordinateur qui a modifié en profondeur les façons de faire traditionnelles et permis de réaliser des choses tout à fait nouvelles et insoupçonnables au XX^e siècle.

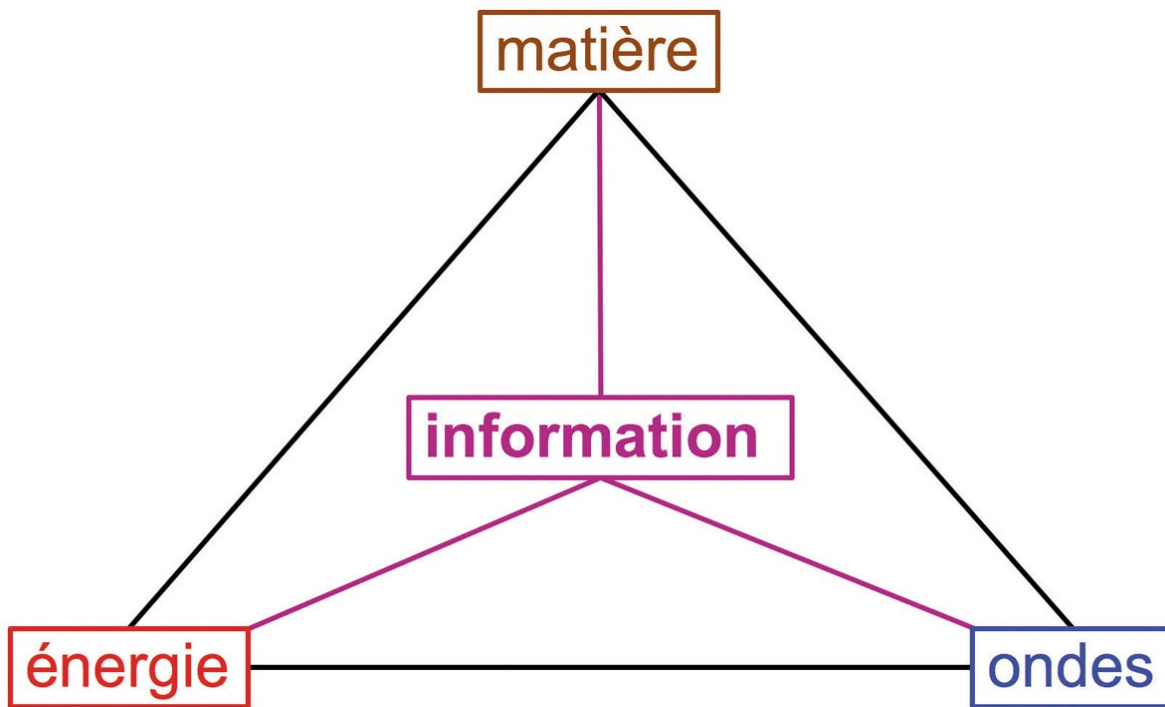


Figure 2. Le tétraèdre du ^{xxi}^e siècle.

L'ancien triangle n'a pas disparu pour autant : l'ordinateur lui-même est bien sûr fait avec de la matière, de l'énergie et des ondes. Mais, comme l'a montré Alan Turing dès son premier article fondateur de 1936 ([Turing, 1936](#)), c'est une *machine universelle* : ses mêmes composants physiques peuvent être utilisés identiquement pour toutes les tâches liées à l'information et au calcul, quels que soient leurs objectifs. De ce point de vue, même s'ils n'ont pas les mêmes performances, tous les ordinateurs se valent, quelle que soit la façon dont ils sont construits.

La naissance de la science du calcul et de l'information

L'information était depuis longtemps une notion sous-jacente dans toutes les sciences et techniques : les mathématiciens et les physiciens modélisaient les phénomènes naturels les plus variés par des équations et des calculs écrits sur papier ou sur des tableaux, alors que les techniciens concevaient leurs machines avec des plans et des calculs en s'aidant de tables numériques, abaquages, règles à calcul ou machines électromécaniques. En cela, ils ne faisaient jamais que manipuler et se communiquer de l'information. Mais cette information sous-jacente, considérée comme intimement liée aux échanges entre humains, n'était pas en elle-même un sujet d'études théoriques. De leur côté, les savants de la physique statistique voyaient bien que pour comprendre l'entropie il fallait quantifier les notions informationnelles d'ordre et de désordre, mais ils restaient fondamentalement dans le cadre triangulaire classique. En biologie, depuis Mendel et les débuts de la génétique, on parlait de caractères héréditaires sans encore connaître leur support physique, donc implicitement en termes d'information. Quand les chromosomes portés par l'ADN ont été mieux compris, les biologistes ont pu se reconcentrer sur la matière, du moins jusqu'à l'informatisation récente des calculs sur le génome et la naissance de la biologie computationnelle dont nous reparlerons au [chapitre 8](#).

LA CONTRIBUTION ESSENTIELLE DES LOGICIENS

C'est par les logiciens qu'est arrivée la vraie révolution conceptuelle. Depuis l'Antiquité, la vieille logique d'Aristote était surtout discutée par les philosophes, qui s'interrogeaient sur la portée de certaines implications ou syllogismes : la phrase « le chat est un oiseau implique que Lyon est capitale de la France » est certes logiquement vraie, mais jusqu'à quel point a-t-elle un sens ? Mais rien d'essentiel n'est sorti de ces discussions, du moins du point de vue de la logique en tant que science.

La première vraie avancée a été la transformation par George Boole du raisonnement logique de base en calcul algébrique sur *vrai, faux, et, ou, non*. Boole a fait vraiment entrer la logique dans la science, et sa théorie est toujours celle qui fonde la conception des circuits électroniques, tout en restant importante en programmation des logiciels. Ce qui est moins connu, c'est que le calcul booléen reste un des grands mystères des mathématiques et de l'informatique, et qu'il passionne les informaticiens car tout progrès dans les algorithmes booléens est utile pour de très nombreuses applications. Nous en reparlerons au [chapitre 10](#) et dans les annexes du [chapitre 12](#).

La deuxième avancée a été la tentative avortée de fondation des mathématiques par Georg Cantor à l'aide de sa théorie intuitive des ensembles, qui ne fut corrigée qu'au début du ^{xx}e siècle par une meilleure compréhension de la notion de *théorie axiomatique*, c'est-à-dire de théorie mathématique fondée sur des axiomes et des règles de déduction. C'est en 1928 que David Hilbert posa son fameux *Entscheidungsproblem* dont le but était de relier logique et calcul : étant donné une théorie axiomatique quelconque, par exemple celle des nombres entiers usuels, existe-t-il une « méthode de calcul effective » pour démontrer toutes ses propositions vraies, c'est-à-dire valides dans le modèle que définit la théorie ? En 1931 Kurt Gödel commença par montrer que les démonstrations pouvaient se réduire à des manipulations numériques effectives, puis établit dans son fameux théorème d'incomplétude que la définition même de la notion de démonstration interdisait de prouver par simple déduction logique la vérité de toutes les formules vraies dans toute théorie axiomatique non triviale, dont la toute simple arithmétique des nombres entiers. Ce résultat a provoqué un choc violent chez les mathématiciens et les philosophes, car il montrait que les notions de vérité et de démonstrabilité ne pouvaient pas être confondues, à moins bien sûr que les mathématiques soient contradictoires, c'est-à-dire que toute proposition y soit simultanément vraie et fausse.

Je ne préciserai pas davantage ce résultat, qui n'est pas central pour mon propos et a été expliqué et commenté dans de nombreux ouvrages.

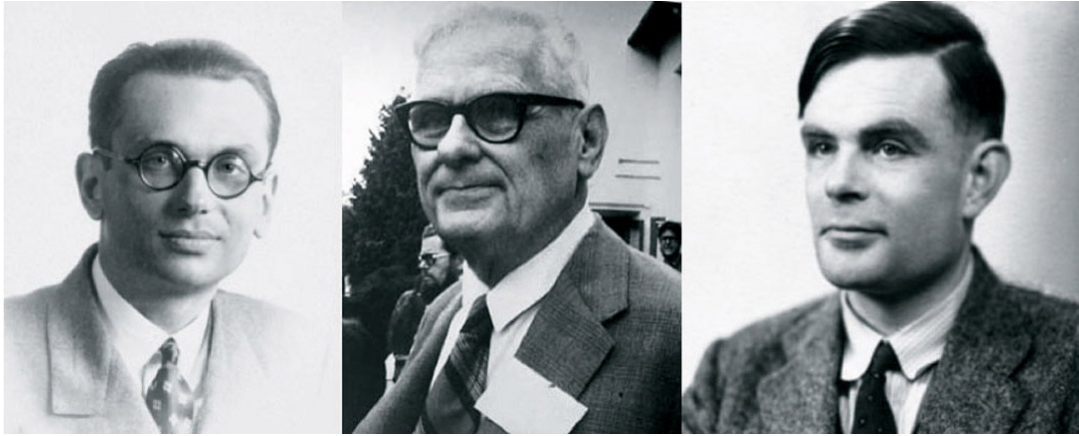


Figure 3. Kurt Gödel, Alonzo Church et Alan Turing.

Mais la notion de « calcul effectif » restait floue. Un peu plus tard, en 1936, a eu lieu l'étape décisive pour notre propos : Alonzo Church et Alan Turing démontrèrent par deux méthodes différentes que la réponse à l'*Entscheidungsproblem* était négative. Ils s'appuyaient chacun sur un formalisme spécifique : Church sur son λ -calcul, langage logique minimal tout en étant extrêmement puissant, et Turing sur sa fameuse machine abstraite simplissime et bien popularisée en 2012 lors du centenaire de sa naissance.

Illustrée par la figure 4, la machine de Turing possède six composants : un alphabet fini constitué de symboles, ici des chiffres ; une bande de mémoire potentiellement infinie, divisée en cases pouvant contenir chacune soit un blanc, soit un symbole ; une tête de lecture/écriture capable de lire un symbole écrit sur la bande et de le remplacer par un autre ; un mécanisme qui peut déplacer la bande d'une case à droite ou à gauche ; un ensemble fini d'états, ici A, B et C ; enfin un mécanisme de contrôle composé d'un nombre fini d'instructions très simples, ici écrites dans la boîte du bas. Une instruction spécifie une paire état/symbole, une action qui peut être soit un mouvement, soit l'écriture d'un nouveau symbole par la tête de lecture/écriture, et un état suivant. Elle peut s'appliquer quand la machine est dans l'état spécifié et lit le symbole spécifié sous sa tête. Dans ce cas, elle effectue son action puis met la

machine dans son nouvel état. La machine démarre avec un nombre fini de symboles non blancs sur la bande et exécute ses instructions en séquence. Elle s'arrête si elle atteint un état qui n'a pas de transition pour le symbole courant, ici l'état C qui n'a pas de transition du tout. Il est possible qu'une machine puisse appliquer plusieurs transitions partant d'un même état et d'un même symbole, comme ici pour B:7. Dans ce cas, la machine est dite *non déterministe*, et elle peut prendre n'importe quelle transition valide.

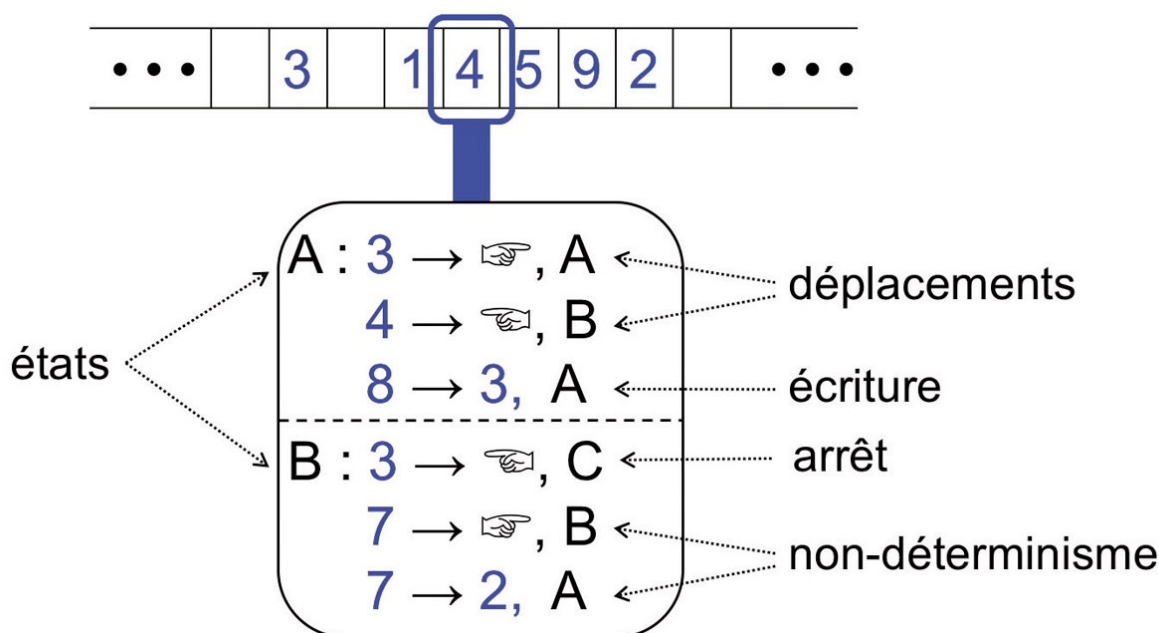


Figure 4. Une machine de Turing.

Il n'est pas très difficile (mais très ennuyeux) de voir que, bien qu'extrêmement rudimentaires, les machines de Turing peuvent faire tous les calculs faisables par n'importe quelle autre machine connue, même si ce n'est pas de façon efficace, et que les machines non déterministes ne sont pas plus puissantes que les machines déterministes.

Le premier grand résultat de Turing est l'existence d'une machine *universelle* : il existe une machine U telle que tout calcul d'une machine spécifique M sur des données spécifiques D peut être réalisé par la machine universelle U en écrivant

d'abord sur sa bande de calcul un *programme* décrivant complètement la machine M , puis les données D . Au passage, Turing inventait la notion cruciale de *programme enregistré* dont nous reparlerons beaucoup dans la suite.

Le second résultat, le plus important, est qu'il n'est pas possible de construire une machine détectant en temps fini si une machine M va s'arrêter sur une donnée D . On dit que l'arrêt de la machine de Turing est un problème algorithmiquement *indécidable*.

Ce résultat fondamental établit une barrière infranchissable aux possibilités des algorithmes se terminant toujours. Il n'est pas difficile à montrer en bricolant un peu la machine universelle U , ce que je ne ferai pas ici. Et il fournit une réponse négative à l'*Entscheidungsproblem*, sur laquelle je ne m'étendrai pas non plus.

Par la suite, Church et Turing montrèrent l'équivalence de leurs formalismes, et Church émit sa thèse que tout formalisme à venir de calcul effectif serait équivalent aux leurs. De nombreux formalismes équivalents mais de présentation et d'usage différents ont été ensuite inventés, mais la thèse de Church n'a jamais été démentie.

DE LA LOGIQUE À L'INFORMATIQUE MODERNE

Mais qu'ont ces élucubrations mathématiques quelque peu ésotériques à voir avec les ordinateurs que nous utilisons tous les jours ? La réponse est simple : l'informatique moderne est très directement issue de ces découvertes fondamentales. D'une part, les théories de Church et Turing ont permis de définir de façon parfaitement propre la notion jusque-là plutôt vague de calcul effectif, et, d'autre part, elles ont mis en place les concepts linguistiques et machinistes qui sont toujours ceux de l'informatique moderne. Le λ -calcul est resté le cœur des meilleurs langages de programmation, et la notion de programme enregistré fonde toujours la dualité matériel-logiciel. C'est cette dualité qui a vraiment permis l'essor de l'informatique, en modifiant les poids respectifs de ses composants lors de ses évolutions successives.

En effet, dans les années 1950, on considérait le logiciel comme « ce qui fait marcher la machine ». La machine matérielle était noble, les logiciels intellectuels étaient spécifiquement développés pour chaque type de machine de chaque constructeur, et ils étaient vus comme tellement peu nobles qu'ils étaient souvent

donnés gratuitement avec la machine. Cela n'a changé que dans l'étape cruciale de l'*unbundling* faite par IBM en 1969. Comme IBM s'était mis à fabriquer une gamme d'ordinateurs compatibles entre eux au niveau des logiciels, les fameux IBM 360, il a décidé de vendre les logiciels séparément. Puis, petit à petit, le logiciel a pris le pas sur le matériel. Maintenant, quand on achète un ordinateur, on choisit davantage le système d'exploitation que le constructeur, et on peut souvent utiliser les mêmes applications logicielles sur des ordinateurs différents. Les navigateurs du Web proposent une unification quasi totale de toutes les plateformes matérielles vis-à-vis de l'accès à Internet, et les applications logicielles sont beaucoup, beaucoup plus nombreuses que les plateformes matérielles.

L'HISTOIRE DES ALGORITHMES

Nous appelons les procédés de calcul effectif des *algorithmes*. Ce mot, qui commence seulement à être connu du public, n'était pas utilisé par Turing dans son article de 1936. Il est en fait ancien, ne vient pas du grec, et ne s'écrit pas avec un « y » comme je le lis encore souvent². Il apparaît en France vers 1230 comme dérivé du latin *algorithmus*, avec le même sens que maintenant. Et le mot latin vient lui-même du nom du grand savant persan Al Khwārizmī³, qui vécut au début du IX^e siècle et nous apporta entre autres les chiffres arabes de la numérisation décimale (en fait importés d'Inde), ainsi que les premières méthodes de calcul algébrique pour résoudre des équations. Le mot *algèbre* est d'ailleurs lui-même issu du titre d'un de ses traités. Mais comment diable enseigne-t-on encore l'arithmétique décimale et l'algèbre sans jamais mentionner son nom ?



Figure 5. Al Khwārizmī, grand-père de l'informatique (source M. Nivat).

Il faut toujours rappeler que la notion d'algorithme n'est pas réservée à l'informatique et encore moins aux ordinateurs. Ils sont aussi omniprésents dans le vivant : les cellules, les organismes et le système nerveux en sont pleins. La pensée algorithmique s'y applique tout autant, comme nous le verrons au [chapitre 8](#).

LA FORMALISATION DE L'INFORMATION

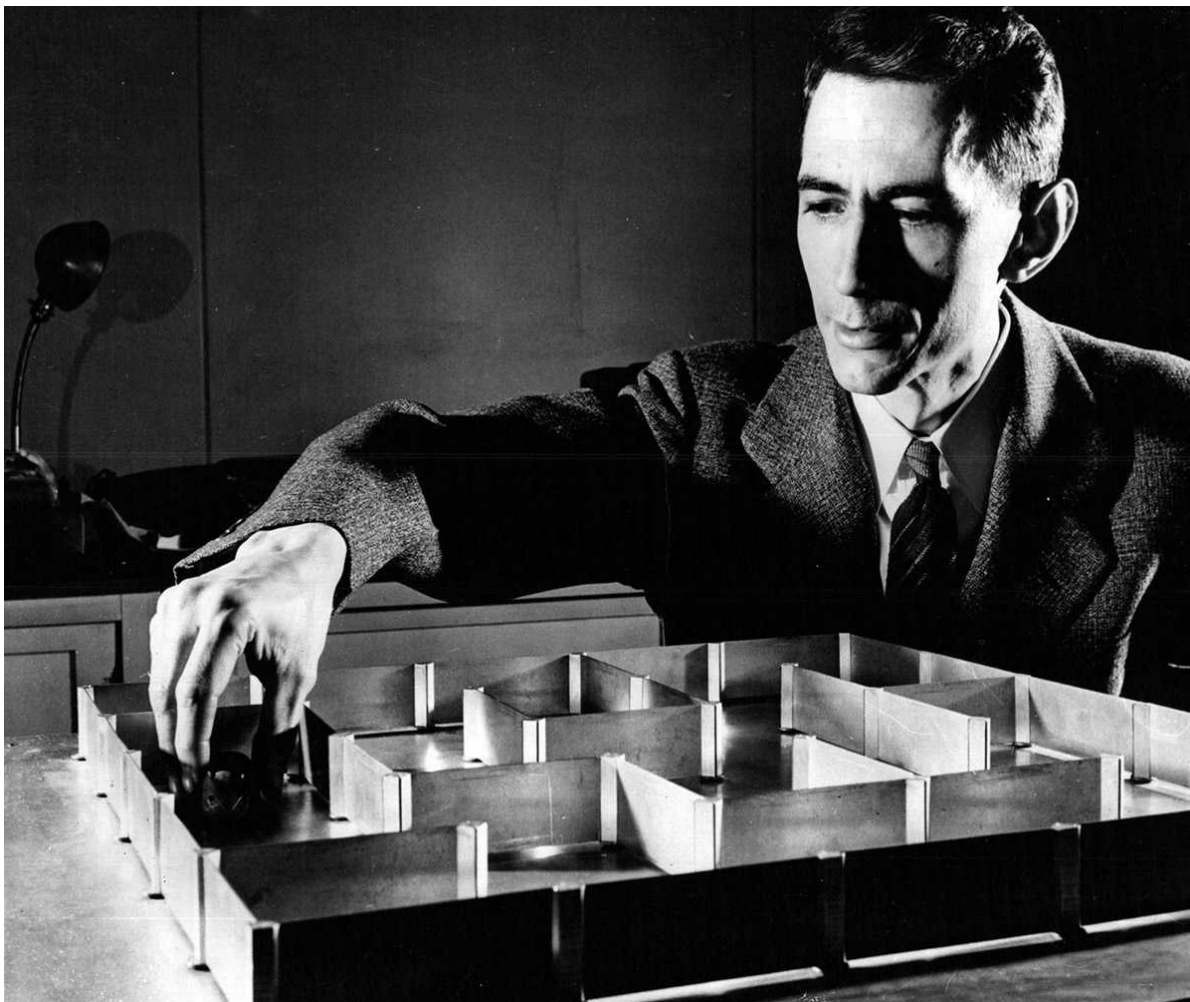


Figure 6. Claude Shannon, créateur de la théorie de l'information.

Claude Shannon (1916-2001), créateur de la théorie de l'information, en février 1959 devant une de ses inventions : une souris à mémoire traversant des labyrinthes (The electrical « mouse » with a « super memory » : The latest scientific invention of Dr Claude E. Shannon of the Bell Telephone Laboratories, New York). © Keystone/Zuma/Leemage.

La notion mathématique d'information n'a été vraiment théorisée qu'au début des années 1950, dans le travail visionnaire de Claude Shannon sur la théorie de la communication, renommée plus tard *théorie de l'information*. C'est grâce à cette théorie profondément novatrice que l'ADSL a permis de faire passer nos communications numériques en vingt ans de quelques kilobits par seconde à des mégabits par seconde sur les mêmes vieux fils de téléphones, que nos communications sans fil évoluent vers le gigabit par seconde, et que nous savons

numériser efficacement la musique, les photos et les films – entre autres applications. Nous y reviendrons dans les chapitres suivants.

LA NAISSANCE DU MOT « INFORMATIQUE »

Enfin, le mot français *informatique* lui-même n'a été créé qu'en 1962 pour parler de traitement automatique de l'information, après l'introduction en 1957 du mot *Informatik* par Karl Steinbuch dans son traité *Informatik : Automatische Informationsverarbeitung*. Il n'a pas pu être utilisé aux États-Unis, car il a été déposé avec le nom de la société Informatics Inc., maintenant disparu. Il a évolué vers un sens plutôt ambigu, car il désigne aussi bien la science et la technique informatique que le rayon de supermarché où on peut acheter son ordinateur, et car les ordinateurs travaillent sur des données et pas de l'information comme nous le verrons au chapitre suivant. Mais c'est quand même un mot bien né, qui est de plus en plus souvent repris en anglais par ceux qui trouvent l'expression *computer science* trop étroite car ne faisant pas mention des aspects techniques fondamentaux en pratique : association Informatics Europe, Informatics Forum à l'Université d'Édimbourg, etc. Je l'utiliserai ici dans le sens très large de traitement automatisé de l'information, effectué ou non par un ordinateur électronique. Dans cette acception, le traitement du signal et la théorie du contrôle sont vus comme à l'intersection des mathématiques appliquées et de l'informatique. Je pense que l'évolution actuelle du sujet et l'intégration continue de ces sciences dans les applications renforcent cette position.

De la calculatrice mécanique à l'ordinateur

Revenons maintenant aux aspects pratiques, en montrant pourquoi l'ordinateur diffère profondément des machines à calculer précédentes. Observons par exemple la superbe calculatrice mécanique de poche Curta de la figure 7, inventée par Curt Herzstark pendant la Seconde Guerre mondiale, et supplantée seulement par l'introduction des calculettes électroniques en 1972 : l'introduction des données se fait par les curseurs latéraux, et le calcul se fait en tournant la manivelle, ce qui met en mouvement des centaines d'engrenages dans un doux chuintement⁴.



Figure 7. La calculatrice de poche Curta.

La Curta et ses collègues électromécaniques sont des machines fantastiques, mais la supériorité de l'ordinateur sur elles s'exprime simplement par deux principes, qui forment selon moi le centre conceptuel de l'informatique :

1. C'est l'ordinateur lui-même qui tourne la manivelle, et c'est son programme enregistré qui décide comment la tourner.

En lui ayant fait avaler la manivelle, on a affranchi l'ordinateur du besoin de l'homme pour faire les calculs, à condition bien sûr de l'avoir préalablement programmé pour la tâche à effectuer et de l'avoir nourri par des données adéquates.

2. Dans un ordinateur, il n'y a pas de différence conceptuelle entre le programme et les données ; l'ordinateur peut donc calculer aussi sur des

programmes.

Ce second principe est vraiment la clef de la distinction entre *matériels* et *logiciels* (programmes). Calculer en prenant des programmes comme données donne à une machine matérielle apparemment figée une souplesse quasi infinie. Cela a d'abord permis de se sortir élégamment d'un problème majeur du début de l'informatique : celui de la très grande difficulté de la programmation dans les « langages machine » des premiers ordinateurs, rustiques, peu intuitifs, très ennuyeux, et de plus différents pour des machines différentes. Ces langages initiaux ont pu être rapidement remplacés par des langages de bien plus haut niveau, beaucoup mieux adaptés à notre pensée et valables pour toutes les machines, qu'il a fallu rendre capables de traduire elles-mêmes les programmes de haut niveau en langage machine à l'aide de programmes spécifiques appelés *compilateurs* pour les exécuter ensuite. Nous en reparlerons aux [chapitres 3 et 10](#), ainsi que dans l'[annexe 12.4](#).

Calculer sur les programmes permet ensuite de construire les logiciels de façon *modulaire*, c'est-à-dire en assemblant des programmes écrits séparément, soit à la main, soit comme éléments de bibliothèques logicielles. La modularité change complètement la programmation en complétant l'écriture des instructions élémentaires par l'appel de programmes réutilisables, dont on ne connaît même plus le texte mais seulement l'interface et la fonctionnalité par une API (*application programming interface*) ; elle est indispensable pour passer à l'échelle des millions de lignes de code. Prendre les programmes comme données a aussi permis la construction des *systèmes d'exploitation*, qui sont des programmes gérant l'exécution des programmes utilisateurs sur une machine donnée. Cela a enfin permis de télécharger des programmes en se connectant aux réseaux d'ordinateurs pour les exécuter ensuite sur place, ce qui est devenu la façon standard de travailler depuis la généralisation d'Internet, ou, à l'inverse, d'envoyer ses données et programmes sur un ordinateur distant comme on le fait en « calcul dans le nuage » (*cloud computing*). Que le lecteur ait une pensée émue pour Turing quand il télécharge une nouvelle version d'un logiciel et la voit s'installer automatiquement !

Le calcul sur les programmes est enfin à la source de ce qu'on appelle le *bootstrapping*, beau mot anglais signifiant « s'élever en l'air en tirant sur ses sangles de chaussure ». *Booter* sa machine, c'est lui faire exécuter des programmes qui vont

mettre progressivement en place les programmes des systèmes d'exploitation adaptés à sa configuration matérielle locale, qui vont à leur tour permettre d'exécuter les programmes d'application qu'on importe ou qu'on écrit soi-même. Le *bootstrapping* est une opération d'autant plus délicate que les systèmes modernes sont de très grande taille, faisant appel à des dizaines de milliers de sous-programmes. Nous y reviendrons à de nombreuses reprises.

L'INFORMATIQUE, UNE SCIENCE DE CONSTRUCTION

Comme les textes et les équations mathématiques, les données et programmes de l'informatique ne sont pas des objets matériels car ils ne sont pas liés à un support physique privilégié. De même qu'une équation s'écrit identiquement sur une feuille de papier ou sur un tableau, une donnée numérique ou un programme peuvent être stockés à l'identique sur une grande variété de supports matériels. Une conséquence importante est que l'informatique n'est pas une science naturelle, mais une *science de construction*. Elle ne s'attache pas à comprendre la nature qui nous a été donnée sans que nous n'y puissions rien, mais au contraire à construire une espèce de « nature synthétique », certes limitée mais bien utile pour nous. C'est pour cela que les transformations dues à l'informatique sont aussi rapides. Si toute expérience de physique quantique ou de biologie demande un équipement parfois considérable, des précautions de chat et beaucoup de temps, il n'a fallu que quelques mois pour construire les premiers moteurs de recherche sur le Web ; en très peu de temps, ces moteurs informatiques ont changé notre vie à un point tel que nous n'arrivons plus à nous souvenir de comment nous faisons avant, ni à l'expliquer à nos enfants qui ne pourront en entendre parler qu'en cours d'histoire. Mais la matière étant toujours nécessaire en fin de compte, n'oublions pas de remercier chaudement les physiciens qui nous permettent de loger quelques milliards de transistors sur un petit centimètre carré de silicium, de fabriquer de superbes écrans plats, et de transmettre une quantité fabuleuse d'informations sur de simples fibres optiques, dans lesquelles ce sont des photons qui font passer votre courrier qu'on continue pourtant à appeler « électronique » !

Comment l'information unifie et transforme tout

La figure 8 illustre quatre types d'objets bien connus au xx^e siècle. En haut à gauche, un livre, constitué de feuilles de papier reliées sous une couverture, sur lesquelles sont imprimés à l'encre des caractères. En haut à droite, un disque vinyle, percé d'un trou au milieu, orné d'une étiquette imprimée, et porteur d'un sillon continu où sont gravées les ondulations qui portent physiquement la musique ; celle-ci sera jouée à l'aide des procédés mécaniques des gramophones ou électromécaniques des électrophones. En bas à gauche, une photographie papier tirée par un procédé chimique à partir d'un négatif argentique exposé par l'appareil photo. En bas à droite, les engrenages, comes et ressorts d'un dispositif mécanique, tels que ceux de la transmission et de la suspension d'une voiture. Tous ces objets étaient profondément différents, conçus par des ingénieurs différents et fabriqués par des usines différentes. À chaque objet son métier.

Textes, dessins
Formules



Figure 8. Le milieu du xx^e siècle.



01100110111101
10010011101100



Figure 9. La fin du xx^e siècle.

La figure 9 montre la profonde transformation apportée par l'informatique à la fin du xx^e siècle : textes, images, son et forces sont maintenant complètement unifiés en étant tous *numérisés*, c'est-à-dire traduits en nombres stockés dans des mémoires parfaitement interchangeables, puis travaillés par les mêmes ordinateurs (les mécanismes de numérisation seront précisés au [chapitre 3](#)). Mais tous ces objets traditionnels peuvent être reconstruits à partir des nombres qui les encodent, parfois en beaucoup mieux. Expliquons pourquoi et comment.

LIVRES, PHOTOS ET SONS

Pour le livre, les technologies numériques ont d'abord remplacé les anciennes, sans d'ailleurs changer la forme ni l'aspect des livres physiques. Mais elles ont aussi permis de nouveaux usages complémentaires. Par exemple, les livres numériques sur liseuses ou tablettes sont bien pratiques pour les voyageurs, même s'ils sont loin de remplacer totalement les livres papiers comme annoncé en fanfare par leurs promoteurs lors de leur sortie.

Pour la photographie, c'est différent. Si la photographie d'art sur papier argentique n'a pas vraiment changé d'aspect, sa technologie de production est devenue largement numérique, y compris d'ailleurs pour les prises de vue argentiques afin de pouvoir utiliser tous les outils numériques de postproduction. Mais un changement de très grande ampleur s'est produit récemment dans la photographie d'amateur et de loisirs : le téléphone portable est devenu le principal moyen de prendre et de regarder les photos⁵, et le nombre de photos prises et diffusées a tellement augmenté qu'il atteint plusieurs milliards par jour. Et, comme nous le montrerons au [chapitre 4](#), les appareils photo numériques modernes et même les téléphones, bourrés d'informatique de haut niveau, ont des possibilités tout à fait impensables en argentique et fort utiles aux photographes (dont je suis).

Les transformations sont du même type en musique. Dès 1982, la musique a été gravée numériquement sur des compact-discs, objets bien plus petits que les vinyles et que l'écoute n'utilise pas ; les miens, que j'ai soignés, montrent une durée de vie très

supérieure à ce qui était annoncé. Ensuite, on a pu graver soi-même ces disques avec son ordinateur, puis enregistrer les sons avec de tout petits enregistreurs numériques. *Exit* les magnétophones. Quand les disques durs sont devenus assez gros, on a pu prendre le chemin exactement inverse : transférer la musique de ses CD vers son ordinateur, et maintenant vers son téléphone. Enfin, plus récemment, l'expansion du *streaming*, c'est-à-dire de la diffusion en continu de la musique depuis des serveurs situés n'importe où dans le monde, a permis de s'affranchir de tout support physique personnel et d'avoir accès en haute qualité à l'ensemble du catalogue mondial⁶. Et beaucoup des enregistrements modernes ont été traités par des procédés algorithmiques, par exemple pour retrouver au mieux le son d'origine en défaisant les distorsions des 78-tours. La même chose s'est produite pour la vidéo et le cinéma, avec l'arrivée du DVD, puis du BluRay, ce dernier ayant été détrôné pratiquement depuis sa naissance par le téléchargement et le streaming. La restauration numérique des vieux films permet enfin de les revoir dans de bonnes conditions et de leur donner une durée de vie potentiellement infinie. Enfin, les salles de cinéma aussi deviennent numériques, avec la transmission des films par réseau. Mais la pellicule garde sa valeur pour le tournage.

Ainsi, en une vingtaine d'années, les choses ont tellement changé que les supports physiques ont pratiquement disparu pour être remplacés par des stockages numériques dans le nuage (*cloud*) des réseaux et ordinateurs. Les bouleversements de la pratique des utilisateurs, de la structure des industries, et des aspects financiers de ces industries ont été fantastiques et ne sont pas terminés. Les acteurs de l'industrie traditionnelle des médias ont été balayés ou secoués très fort, et de nouvelles jeunes pousses ont grandi à des vitesses exceptionnelles pour parfois prendre leur place. La tourmente est loin d'être terminée, et je ne prédirai certainement pas comment l'avenir sera organisé.

Certains regrettent le passé et proclament que la photo argentique et la musique sur vinyle étaient « quand même mieux » que leurs nouvelles formes. Je respecte tout à fait le côté psychologique de cette nostalgie, mais j'affirme qu'il est facile de voir pourquoi c'est une impression vraiment fautive en termes de qualité : vinyles et films argentiques ont des distorsions qui sont effectivement fort agréables aux oreilles et aux yeux, mais qui restent des distorsions. Il est tout à fait possible de reproduire ces distorsions sur un son numérisé par des algorithmes adéquats, et même de rajouter des

clics aléatoires ou répétitifs à une musique ou des rayures à une photo pour faire comme s'ils étaient usés. De toute façon, depuis longtemps, la plupart des vinyles sont réalisés à partir d'enregistrements numériques et les tirages photos à partir de scans numériques post-traités par logiciels, le tout étant conservé sur supports numériques recopiables à l'infini pour éviter l'inéluctable dégradation chimique des bandes magnétiques, négatifs et diapositives. Et je m'étonnerai toujours de lire des tests de comparaison de la qualité de la restitution sonore de câbles numériques hors de prix. S'il y avait une différence, aucun ordinateur ne pourrait fonctionner...

DE LA MÉCANIQUE AUX SYSTÈMES CYBERPHYSIQUES

Une transformation encore plus importante mais moins connue du public est celle de la mécanique au sens large. La numérisation des pressions, forces, et autres quantités physiques à l'aide de capteurs variés, jointe au développement d'actionneurs comme les servomoteurs de tous types, ont permis de mettre l'information au centre du jeu. Alors que leviers, ressorts, cames et pistons n'offrent qu'une gamme de possibilités limitée, on peut à peu près tout faire en combinant des capteurs physiques, des algorithmes de théorie mathématique du contrôle et des servomoteurs.

L'exemple le plus ancien est probablement celui des avions, dont les systèmes de commande de vol et de navigation ont été progressivement électrifiés puis informatisés depuis les années 1960, conduisant à ce qu'on appelle l'avionique (*avionics* en anglais, qui abrège *aviation electronics*). Dans un avion standard, les ailes portent vers le haut, mais la queue vers le bas, ce qui assure la stabilité au prix d'un manque de manœuvrabilité et d'une dépense d'énergie supplémentaire. Les avionneurs ont décidé dans les années 1970 de fabriquer des avions de chasse instables, bien plus maniables. Mais ces derniers ne peuvent alors plus être pilotés par des hommes, qui n'ont ni la vitesse ni la précision de réaction nécessaires. Le contrôle fin des avions instables a d'abord été réalisé par des calculateurs analogiques, en application de la théorie cybernétique de Norbert Wiener, puis par des logiciels sur ordinateurs capables d'exécuter de façon bien plus précise des algorithmes plus sophistiqués. Le premier avion de transport à « commande électrique » a été l'Airbus

A320, mis en service en 1988 et devenu un des plus grands succès de l'histoire de l'avionique ; tous les avions modernes sont maintenant de ce type. J'ai mis « commande électrique » entre guillemets : encore une fois, ce nom reste collé à la matière et à l'énergie, occultant totalement que ce qui décide la commande, ce sont les algorithmes à travers les (gros) logiciels qui les implémentent ! Et toutes les autres fonctions essentielles sont informatisées : cockpit, système de navigation, système de communication, contrôle des réacteurs, du freinage et de la climatisation, surveillance permanente du fonctionnement avec détection des pannes et commande des pièces de rechange pendant le vol, etc., et bien sûr un système audio/vidéo évolué pour distraire les passagers au moins sur les longs courriers. L'informatique est ici soumise à de très fortes contraintes de sécurité régies par une norme internationale de certification des logiciels appelée DO-178C. L'expérience a montré que les avions informatisés avaient de meilleures performances et moins de risques d'accidents que les précédents.

L'automobile a suivi un peu plus tard, d'abord avec l'ABS (Anti Blocking System). L'idée était ici d'éviter que les roues se bloquent et dérapent, ce qui diminue très fortement le freinage. Une simple détection de décélération de la rotation couplée avec un bon algorithme d'automatique permet de relâcher le frein juste quand il faut et ce qu'il faut pour éviter les blocages, de plus en traitant indépendamment les quatre roues. Pour qu'un homme puisse faire la même chose, il lui faudrait quatre pieds et des réflexes hyperrapides ! Aucun système électromécanique n'a jamais été capable de cette qualité de freinage ; travailler avec l'information le rend possible. L'automobile s'informatise de plus en plus, d'une façon qui ressemble à celle des avions : contrôle moteur, régulateur d'allure, Stop&Go qui arrête le moteur aux feux rouges, direction, suspension, boîte de vitesses, airbags, GPS, radars, climatisation, tout est réglé par des ordinateurs et des logiciels. Un grand mouvement atteint maintenant toute l'industrie automobile : le passage à l'ère de la voiture autonome, conduite par des algorithmes de théorie du contrôle joints à de la reconnaissance de la route et des événements qui s'y produisent par apprentissage automatique. Mais il est encore un peu tôt pour savoir comment tout cela va évoluer.

Pourtant, les normes de sécurité de l'informatique automobile restent peu exigeantes et sa qualité très inférieure à celle des avions, bien qu'elle soit devenue le principal argument marketing des constructeurs, comme le montrent

systématiquement les publicités actuelles. Dans son schéma mental, l'industrie automobile est restée profondément ancrée sur la matière et l'énergie : on y parle encore du contrôle « électronique » du moteur, alors que la vraie valeur ajoutée du contrôle informatisé est dans la modélisation mathématique et le logiciel, donc dans l'information. De plus, cette industrie est tirée en permanence par la réduction des coûts, ce qui ne favorise pas les investissements intellectuels et techniques pour maîtriser les difficultés intrinsèques d'une informatisation complexe. Nous en reparlerons au [chapitre 9](#).

Les avions et les voitures fournissent d'excellents exemples de ce qu'on appelle maintenant les systèmes *cyberphysiques*, qui intègrent mécanique, physique, électronique et informatique. Bien d'autres systèmes cyberphysiques sont en train d'être mis en place, que ce soit au niveau des petits objets domestiques ou à celui de l'instrumentation de toute une ville : transports, réseaux de distribution, réseaux de capteurs pour analyser les flux de véhicules et de personnes, mesures de pollution, etc.

MODÉLISER ET SIMULER NUMÉRIQUEMENT, UNE RÉVOLUTION CONCEPTUELLE

Au xx^e siècle, la conception des objets techniques comme les avions, les voitures, les bâtiments et les ponts demandait souvent la construction de maquettes physiques et l'expérimentation sur ces maquettes. Par exemple, on utilisait des souffleries pour tester l'aérodynamique des avions et des voitures, ou des maquettes en bois pour visualiser en 3D la forme et l'agencement intérieur des bâtiments. Au xxi^e siècle, les maquettes physiques tendent à disparaître pour être remplacées par des *maquettes numériques* 3D ou 4D (animées) implémentées sur ordinateur à l'aide de logiciels dédiés. Un exemple caractéristique est celui du Boeing 777, le premier avion conçu informatiquement à l'aide du logiciel CATIA de Dassault Systèmes. Sa maquette virtuelle a défini à la fois toutes ses pièces et leur agencement. Elle a servi de référence à l'ensemble du projet et de sa documentation, et permis d'imprimer automatiquement les plans. Cette CAO (conception assistée par ordinateur) se généralise maintenant à toutes sortes d'objets techniques, du circuit électronique à la fusée en passant par l'automobile et la cafetière.

Mais les maquettes virtuelles ne sont pas destinées qu'à la conception des objets et au guidage de leur fabrication. Associées à des modèles mathématiques de leur comportement implémentés eux-mêmes par logiciel, elles permettent d'étudier virtuellement le fonctionnement dynamique de ces objets. Ainsi, l'aérodynamique d'un avion est calculée et améliorée à l'aide de modèles mathématiques conduisant à des calculs numériques de très grande taille sur des superordinateurs. Les « souffleries virtuelles » ainsi conçues s'avèrent plus fines et plus précises que les souffleries physiques. Sur un autre plan, l'ensemble du pilotage de l'avion est également simulé sur ordinateur, avec une simulation complète de l'environnement aérien et de terrain jointe à un couplage avec le calcul aérodynamique. Les premiers vols sont en fait des vols virtuels, et, quand l'avion décolle pour la première fois, ses pilotes d'essai avaient déjà beaucoup volé en simulation (on peut presque dire que ce vol sert à valider et éventuellement à corriger le simulateur). Les pilotes de ligne sont systématiquement formés sur simulateurs, et leur formation inclut des choses infaisables en entraînement réel, comme des pannes complexes ou des atterrissages en catastrophe, sur l'eau par exemple. Les voitures aussi sont simulées, et les auto-écoles virtuelles sont en préparation. Enfin, nous verrons aux [chapitres 7 et 8](#) que la modélisation et la simulation mathématico-informatiques prennent une place croissante en médecine et dans toutes les sciences, y compris les sciences de la vie dans lesquelles les mathématiques n'avaient jamais tenu un rôle majeur.

L'ÉVOLUTION DES ACTIVITÉS, AVEC SES DANGERS

La conséquence la plus importante de toutes ces transformations est une modification extrêmement profonde de tous les métiers associés :

Des activités autrefois distinctes demandant des formations très différentes des ingénieurs, chercheurs, médecins et employés sont maintenant largement unifiées. Il faut utiliser le même schéma mental et les mêmes connaissances informatiques que l'on travaille sur les textes, les photos, les sons, les vidéos, les images médicales ou les systèmes cyberphysiques.

Cette affirmation pose bien sûr un problème majeur, celui de la formation. Pour l'instant, celle-ci est limitée à l'éducation supérieure, essentiellement pour les études classiques d'ingénieur. Un élève arrive encore au baccalauréat avec zéro connaissance apportée par l'école de ce qu'est vraiment l'informatique. Un médecin ou un biologiste entame sa carrière sans aucune formation sur le sujet, alors qu'il va y être de plus en plus confronté. Je reviendrai sur ce problème essentiel au [chapitre 11](#).

Mais l'informatique n'a pas que des qualités et des avantages. Elle comporte aussi des difficultés et des défauts qui peuvent être dangereux dans les applications sérieuses et qu'il faut donc prendre au sérieux. On parle souvent des impacts de l'informatique à travers les changements rapides et profonds qu'elle produit de plus en plus rapidement dans des pans bien visibles de la société : problèmes d'équité des algorithmes, questions liées à la protection des données personnelles, diffusions de fausses nouvelles, etc. Ce sont effectivement des problèmes majeurs, dont je parlerai au [chapitre 11](#), et pour lesquelles je recommande le livre déjà mentionné *Le Temps des algorithmes* de Serge Abiteboul et Gilles Dowek ([Abiteboul & Dowek, 2017](#)). En revanche, au [chapitre 9](#), je parlerai en détail de deux dangers vraiment intrinsèques de l'informatique : les *bugs*⁷, qui sont des erreurs de conception ou de programmation souvent minuscules mais dont les conséquences peuvent être dramatiques, et les *trous de sécurité*, directement liés aux bugs, qui permettent l'invasion ou la corruption des systèmes informatisés par des personnes malintentionnées. J'expliquerai au [chapitre 10](#) comment bien travailler pour les éviter, en utilisant un bon savoir-faire et les ressources de la science informatique moderne.

CHAPITRE 3

Les piliers de l'informatique

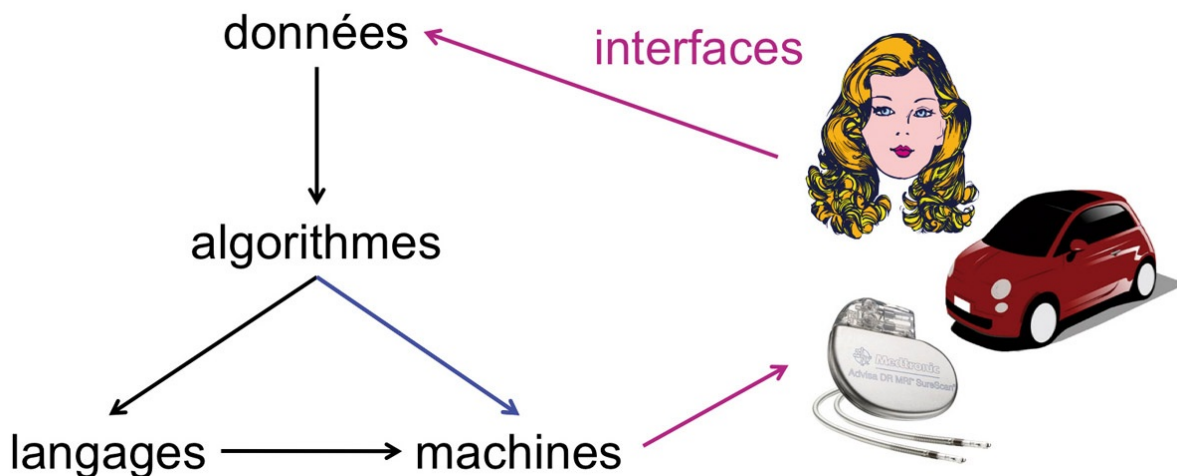


Figure 10. Les piliers de l'informatique.

Il est temps d'entrer plus directement dans la structure interne de l'informatique moderne. Comme illustré par la figure 10, elle repose sur quatre piliers : *données*, *algorithmes*, *langages* et *machines*, auxquels s'ajoutent les *interfaces* qui permettent les communications homme-ordinateurs, ordinateurs-ordinateurs et ordinateurs-objets informatisés, ici une voiture et un pacemaker. La flèche bleue allant des algorithmes aux machines est là pour illustrer le fait que les machines elles-mêmes sont entièrement conçues à l'aide d'algorithmes.

Les données

Les *données* sont l'objet de travail de l'informatique. Ce sont des quantités brutes numérisées (montants financiers, mesures physiques, textes, etc.), ou des groupements structurés d'autres données : numéro de Sécurité sociale = {sexe, année, mois, département, numéro de registre, numéro dans le registre}. Elles peuvent venir soit des hommes, par exemple par Internet, soit de capteurs physiques de toutes sortes échantillonnant le monde, soit de calculs sur d'autres données.

Je ne suis pas spécialiste des sciences des données et je conseille au lecteur qui souhaite approfondir le sujet de visionner le cours 2011-2012 de Serge Abiteboul au Collège de France (Abiteboul, 2012). Mais il m'est bien sûr indispensable de décrire suffisamment le domaine dans ce livre.

QUE VEUT DIRE « DONNÉE NUMÉRIQUE » ?

On utilise souvent le qualificatif « numérique » et l'expression « numérisation du monde » pour exprimer que toute donnée informatique peut être encodée comme un ensemble de nombres. Même si un humain peut s'exprimer dans une base quelconque, par exemple 10 la plupart du temps mais 60 pour les minutes et les secondes et 12 pour les œufs et les huîtres, les données informatiques sont codées en base 2 (avec seulement les chiffres 0 et 1) dans les ordinateurs. Cela a beaucoup d'avantages techniques et mathématiques, car le *bit* valant 0 ou 1 est la plus petite quantité d'information possible. Il peut être utilisé aussi bien pour former des nombres en base

2 que pour encoder les valeurs logiques *vrai* et *faux* utilisées dans l'ensemble des processus de décision, qui sont aussi importants en informatique que les processus de calcul arithmétique. Et, pour des raisons essentiellement pratiques, les bits sont souvent groupés par 8 pour former des *octets*.

Je pense utile pour la suite de préciser ce que tout le monde sait confusément à travers quelques exemples de la vie courante :

- Une mesure physique ou un montant financier sont directement des nombres.
- Un texte est facilement numérisé en considérant les lettres comme des chiffres dans une base suffisamment grande : 128 pour le vieux code ASCII¹, ce qui tient sur 1 octet, ou $16\,384 = 2^{16}$ pour le code ISO qui encode tous les alphabets connus sur 2 octets. Par exemple « informatique » se traduit en ASCII par la suite de nombres 73, 110, 102, 111, 114, 109, 97, 116, 105, 113, 117, 101, car 73 est le code de i, 110 celui de n, etc. Et 73 s'encode lui-même en binaire par l'octet 01001001. Pas facile à lire pour un humain, mais impeccable pour un ordinateur !
- Une image non comprimée est représentée par un grand tableau de pixels (*picture cell* = point coloré élémentaire), où chaque pixel est un triplet de nombres (r, v, b) encodant respectivement l'intensité lumineuse du rouge, du vert et du bleu dans un système de codage des couleurs déterminé (sRGB par exemple pour le Web). Une photo numérique moderne comporte une bonne dizaine de millions de pixels. La compression, par exemple en format JPEG, permet de réduire la quantité de nombres au prix d'une dégradation plus ou moins forte de l'image.
- Un son non comprimé est représenté par une suite d'intensités sonores échantillonnées régulièrement, chacune étant encodée numériquement. Par exemple, pour un compact-disc², la fréquence d'échantillonnage est de 44,1 kilohertz et l'encodage se fait sur 16 bits (valeur décimale entre 0 et 16 383). Dans un enregistrement Hi-Res Audio, de meilleure qualité si la prise de son et la restitution sont à la hauteur, la fréquence est de 96 kilohertz et l'encodage se fait sur 24 bits (entre 0 et 16 777 215).
- Un film ou une vidéo ne sont jamais que des suites d'images associées à des pistes sonores, donc leur encodage se fait à première vue en combinant les encodages de son et d'images précités. Mais la taille du résultat serait déraisonnable, et il est presque toujours indispensable de compresser beaucoup les films comme nous l'expliquerons plus loin.

- Une séquence ADN s'encode comme une longue suite des quatre lettres A, T, C et G correspondant aux quatre acides aminés adénine, thymine, cytosine et guanine, représentées en général à l'aide des 4 valeurs d'une paire de bits (par exemple A = 00, T = 01, C = 10, G = 11) ; elles pourraient l'être par le code ASCII ou ISO de leur lettre, mais ce serait utiliser plus de bits pour rien. Même chose pour l'ARN en remplaçant le T par U pour uracile.

On voit dans cette énumération la très grande importance des interfaces physiques : rien ne sert d'encoder un son en Hi-Res Audio si le système de captage ne fournit que 14 bits utiles en termes d'information sonore ; les 10 bits restants ne font que prendre de la place sur le support d'encodage et dépenser de l'énergie lors de l'encodage et du décodage ! En revanche, il est bon d'avoir 24 bits si on effectue des calculs complexes sur le son, car tout calcul dégrade la précision.

Tiens, tiens, ceci signifierait-il que la relation entre données et information n'est pas immédiate, et que la taille des données a un effet direct en termes de matière et d'énergie ?

Voici deux questions cruciales auxquelles nous allons répondre maintenant, en commençant par la seconde qui est plus simple.

LA TAILLE DES DONNÉES

Notons d'abord que la notion informationnelle de *taille d'encodage* désigne le nombre de bits ou d'octets (8 bits) que prend une information une fois encodée dans une donnée numérique. Elle est souvent appelée *poids de la donnée*, ce qui montre encore une fois la difficulté à détacher mentalement de la matière : le bit n'est pas déposé au pavillon des poids et mesures ! En informatique, il faut compter loin, ce qui n'est pas si simple car il y a deux méthodes pour cela.

La première méthode est due au fait qu'on aime beaucoup la base 10 dans la vie courante, en multipliant par 1 000 à chaque changement d'ordre de grandeur. En base 10, 1 *kilobit* vaut $1\ 000 = 10^3$ bits, 1 *mégabit* 1 000 kilobits = 10^6 bits, 1 *gigabit* 1 000 mégabits = 10^9 bits, 1 *térambit* vaut 10^{12} bits, 1 *pétabit* 10^{15} bits, 1 *exabit* 10^{18} bits, 1 *zettabit* 10^{21} bits, etc. Pour les octets, on multiplie tout par 8. On peut appliquer le même schéma à d'autres unités, comme l'*op*, pour le nombre

d'opérations par seconde et le *flop* pour le nombre d'opérations de calcul en virgule flottante par seconde. Bien que ces nombres aient l'air astronomiques³, ils sont bien réels en informatique : un DVD stocke environ 4 gigaoctets, ce qui est vite devenu moins que la taille mémoire disponible sur un ordinateur individuel banal, un disque dur de 2017 contient typiquement 4 téraoctets, les superordinateurs font des pétaflops et visent l'exaflop, et le trafic Internet en 2015 a été de 72,5 exaoctets par mois, soit 2,3 zettaoctets par an (il devrait presque tripler en 2020). Et soyons toujours admiratifs des anciens Indiens qui allaient beaucoup plus loin en sanskrit dans le nommage des puissances de 10 !

Il est tout aussi naturel de préférer la base 2, beaucoup mieux adaptée aux besoins internes de l'informatique, et de profiter de l'approximation $2^{10} = 1024 \sim 10^3$. On utilise alors les préfixes *kibi* au lieu de kilo, *mebi* au lieu de méga, *tebi* au lieu de téra, etc. La différence devient rapidement sensible : le correspondant du téraoctet (1 000 milliards de bits) est le *tebibit*, qui vaut 1 099 511 627 776 bits, soit en gros 10 % de plus. Ce système de comptage pour l'information est normalisé par l'IEEE (Institute of Electrical and Electronics Engineers), une des grandes organisations du domaine, et l'ISO (International Standards Organization). Mais son usage reste relativement confidentiel à ma connaissance.

DONNÉES VERSUS INFORMATION

Quant à l'autre question, celle de la relation entre donnée et information, la réponse est que *l'information contenue dans une donnée dépend essentiellement du contexte d'interprétation*. Si nous stockons dans une mémoire la valeur décimale 20 pour dénoter une température en degrés Celsius, cette valeur est une donnée, alors que « la température est de 20 degrés Celsius » est une information compréhensible en Europe mais pas aux États-Unis, où l'on parle en degrés Fahrenheit. Et une température de 20 degrés Fahrenheit (soit – 6,6 degrés Celsius) est froide pour nous mais parfaite pour un ours blanc. L'information est donc par essence une notion à géométrie variable.

Une chose dont il faut bien se souvenir : quand on met des données sur un support physique ou quand on les transmet sur Internet, elles ne sont pas du tout

interprétées. La notion de *neutralité* d'Internet, objet de beaucoup de discussions, exprime précisément que les données doivent être transmises telles quelles sans la moindre interprétation. Dans ce cas, l'information contenue dans la donnée est simplement vue par les réseaux et les supports de stockage comme réduite à sa chaîne de bits.

COMPRIMER LES DONNÉES

Regardons maintenant comment on peut compresser une musique numérisée sur un CD, qui se présente originellement sous la forme d'une longue suite de nombres écrits sur 16 bits. On peut réaliser deux types de compression :

- Une compression *sans perte d'information*, qui ne fait que factoriser des redondances dans la suite de nombres. Par exemple, « 3 3 3 3 3 3 3 » peut être traduit par « 7 fois 3 », ce qui peut s'encoder par une suite de nombres plus courte tout en gardant la possibilité de reconstruire facilement la suite initiale à partir de la suite comprimée. C'est le genre de chose qu'on fait de façon plus subtile dans les formats FLAC pour le son et TIFF pour la photo, qui gagnent en gros un facteur 2 par rapport à l'original en gardant exactement la même qualité sonore ou visuelle. La donnée est différente, mais l'information est la même *dans son cadre d'interprétation*, c'est-à-dire modulo décompression.
- Une compression avec perte d'information, avec par exemple MP3 pour le son et JPEG pour l'image. Ici, on utilise des particularités physiologiques de nos systèmes auditif et visuel pour compresser bien davantage, 6 fois par exemple, en ne dégradant pas trop la qualité. Le fichier source et le fichier comprimé contiennent des informations « proches », dans un sens qui n'est d'ailleurs pas facile à définir. Pour toute autre interprétation, les deux fichiers n'ont rien à voir.

Étant très gourmande en taille de données, la vidéo doit être comprimée bien plus que les photos en profitant des particularités de notre perception visuelle. Comme il adore tout ce qui bouge et tend à négliger le reste, on peut se contenter d'une résolution plus faible et d'une compression plus forte qu'en image fixe. De plus, après compression, une vidéo originellement à 24 images par seconde ne contient généralement que trois vraies images par seconde, les 21 images restantes résultant

simplement de déplacement de rectangles internes des vraies images. Cela suffit pour tromper l'œil et garder la fluidité du mouvement. Si le lecteur peut accéder à ces images intermédiaires, je gage qu'il sera surpris de voir que leur qualité peut être plus que médiocre, par exemple avec des dédoublements de parties d'objets mobiles. Une fois le film lancé, on ne voit plus du tout ces artefacts.

L'ANALYSE DES DONNÉES ET LE BIG DATA

On parle beaucoup en moment d'analyse de grandes données, ou *big data*. Ici, il s'agit d'extraire une information compacte et pertinente pour nous à partir d'une très grande masse de données, où chaque donnée peut encoder elle-même un très grand nombre de paramètres (autant que de pixels pour une image). Contrairement à ce que beaucoup de gens semblent croire, ce problème est vraiment difficile et l'interprétation des résultats l'est aussi, surtout quand s'y confondent corrélation et causalité. Je conseille la lecture de l'article « Storks deliver babies » ([Matthews, 2000](#)), en français « Ce sont les cigognes qui apportent les bébés », qui montre une corrélation significative entre le nombre de cigognes et le taux de natalité dans une liste de pays bien choisis – que devons-nous en déduire ? Mais, quand elle est bien conduite sur des ensembles de données pertinentes sur lesquelles on pose des questions pertinentes, l'analyse de données rend des services considérables. Et les nouvelles méthodes d'*apprentissage automatique* dont nous parlerons plus loin donnent souvent des résultats de qualité étonnante, même si on ne sait pas encore très bien pourquoi. La science des données est devenue récemment un des domaines les plus importants de l'informatique. Le Collège de France vient précisément de créer une nouvelle chaire de plein exercice « Sciences des données », occupée par le mathématicien Stéphane Mallat à partir de la rentrée 2017.

Les algorithmes

J'ai déjà beaucoup parlé des algorithmes, qui forment le cœur et la vraie spécificité de l'informatique. Je rappelle que ce sont des processus de calcul automatisables, c'est-à-dire des processus exécutables de façon systématique par des machines ne comportant aucune intelligence, quel que soit le sens de ce mot. Si intelligence il y a, elle ne peut (pour l'instant) venir que de notre habileté à concevoir et écrire les algorithmes et à collecter les données.

Je reviens aussi sur le fait que la notion d'algorithme n'est pas nécessairement liée à celles d'ordinateur et de logiciel. Sans connaître leur nom, nous connaissons et pratiquons des algorithmes depuis longtemps : apprendre à additionner des nombres, c'est apprendre une méthode systématique de propagation de retenues allant de droite à gauche, donc un algorithme mécanisable (ce n'est d'ailleurs pas la bonne méthode pour additionner des nombres sur un circuit). L'algorithme lui-même travaille sur des données qui représentent les nombres, sans même savoir ce que signifie pour nous un nombre. De façon plus générale, toute méthode systématique de travail humain faite pour être exécutée sans y penser peut être vue comme un algorithme. La biologie aussi exécute des algorithmes : le système visuel de notre cerveau en emploie de très sophistiqués et encore largement inconnus pour analyser les informations transmises par la rétine. Celle-ci est elle-même une étonnante machine algorithmique à extraire des informations de contraste, de mouvement, etc., à partir des données brutes fournies par les capteurs biologiques que sont ses cônes et ses bâtonnets puis profondément travaillées par les processeurs de traitement du signal que sont les cellules ganglionnaires connectées au nerf optique ([Sahel, 2016](#)). La pensée

algorithmique n'est donc pas limitée à l'informatique au sens usuel. Cependant, dans la suite, nous nous concentrerons surtout sur les algorithmes exécutés par les ordinateurs travaillant sur des données numérisées et fournissant des résultats numérisés.

Les algorithmes sont extrêmement nombreux et variés, et il est hors de question de présenter le sujet en détail ici. Il me semble cependant utile d'en présenter diverses caractéristiques bien connues des informaticiens mais dont l'expérience m'a montré qu'elles étaient largement inconnues des non-informaticiens. Bien que ce ne soit pas vraiment classique, il me semble d'abord utile de distinguer deux types d'algorithmes : les algorithmes *génériques* et les algorithmes *spécifiques*.

LES ALGORITHMES GÉNÉRIQUES

Les algorithmes génériques travaillent sur des données brutes en ne s'intéressant pas du tout à l'information qu'elles peuvent contenir. Ils servent par exemple à les stocker, les trier, les compresser sans perte d'information, les transporter, les copier ou les diffuser à l'identique, ce qui est la fonction principale d'Internet tant qu'il reste neutre au sens discuté précédemment. D'autres algorithmes génériques, utilisés par exemple par tous les sites Web commençant par « <https://> », servent à assurer une transmission sécurisée des données sensibles ([Abadi, 2011](#)). Il y en a deux sortes : d'un côté, les algorithmes de *chiffrement et déchiffrement*, qui font appel à des théorèmes difficiles et même à des conjectures mathématiques anciennes et toujours non résolues ; de l'autre, les *protocoles de sécurité*, qui doivent assurer que ni les connexions ni les données ne peuvent être compromises lors des échanges. Tous ces algorithmes sont très délicats, et les protocoles de sécurité, dont on parle moins dans les journaux, restent les vrais points faibles de la sécurité informatique. Nous en reparlerons aux [chapitres 9 et 10](#).

LA VARIÉTÉ DES ALGORITHMES SPÉCIFIQUES

De leur côté, les algorithmes spécifiques connaissent le type et le format des données sur lesquels ils travaillent : texte, musique, vidéo, mesures physiques, etc. Il

en existe des milliers, reposant heureusement sur un relativement petit nombre de principes algorithmiques fondamentaux. En voici quelques exemples.

Pour le *texte*, citons la mise en pages automatique, la recherche dans des documents ou sur Internet, la correction de fautes d'orthographe, etc. Ces algorithmes sont loin d'être intuitifs : qui sait comment un moteur de recherche fait pour répondre en un dixième de seconde à une question mal posée, en disant qu'il y a quelques dizaines de milliers de documents pertinents et en donnant une liste courte de ceux qu'il considère comme les mieux référencés ? Nous y reviendrons au [chapitre 5](#).

Le *traitement du son* appartient au domaine plus général du traitement du signal, qui est plein de mathématiques fascinantes initiées par Charles Fourier au début du XIX^e siècle pour comprendre la propagation de la chaleur. De nombreux algorithmes ont été développés par les acousticiens pour la compression des fichiers de son, l'antibruit dans les automobiles ou les lieux de travail, ou encore la reconstitution informatique d'acoustiques spécifiques dans les salles avec placement des sons en trois dimensions, très utilisée dans les concerts modernes.

L'*informatique musicale* développe des méthodes de synthèse de sons, créés soit directement par des algorithmes subtils, soit indirectement à travers l'exécution informatique en temps réel de modèles mathématiques des instruments physiques classiques (cordes, vents, percussions et même voix), ce qui autorise de nouvelles possibilités expressives en déliant leur jeu des limitations physiques des interprètes ([Manoury, 2017](#)). Elle développe aussi des algorithmes d'analyse en temps réel du son permettant par exemple le suivi automatique de partitions ou la co-improvisation homme-ordinateur⁴. Tous ces algorithmes pour la musique sont implémentés dans des logiciels très sophistiqués d'aide à la composition.

D'autres algorithmes permettent de transformer les sons, par exemple pour restaurer le son original de vieux 78-tours, changer votre voix de sexe et d'âge ou vous donner la voix de quelqu'un d'autre. D'autres encore cherchent à séparer les voix des locuteurs dans un brouhaha ; c'est ce qu'on appelle l'*effet cocktail*, pour lequel nous sommes étonnamment doués. C'est un cas particulier du domaine difficile appelé la *séparation de sources*, dont un autre exemple est la recherche des sources des influx neuronaux dans le cerveau en fonction de mesures par des casques externes ou des électrodes implantées, dont nous reparlerons au [chapitre 8](#).

Comme exemple de *traitement d'images*, nous verrons au [chapitre 6](#) que la photographie numérique met en œuvre de nombreux algorithmes pour passer des mesures brutes du capteur photographique à une image plaisante pour l'œil et bien comprimée pour ne pas prendre trop de place. Citons aussi la *stéganographie*, qui permet de signer une image par tatouage en y incorporant des éléments invisibles qui la rendent infalsifiable. Les algorithmes de traitement d'images sont aussi devenus essentiels en astronomie ([chapitre 8](#)). Couplés à de nouveaux capteurs physiques et étendus à la 3D, ils sont à la base des nouvelles modalités d'imagerie médicale qui révolutionnent la médecine, et ont permis l'essor de la radiologie interventionnelle avec ses opérations mini-invasives bien préférables pour les patients lorsqu'elles sont appropriées ([chapitre 7](#)). Ils sont enfin utiles dans bien d'autres domaines, comme la cartographie ([chapitre 6](#)), l'architecture, la cristallographie, la biologie, la paléontologie ([Goussard, 2016](#)), etc.

Pour la *synthèse d'images*, tout le monde connaît les effets spéciaux numériques utilisés au cinéma, qui sont de plus en plus spectaculaires. Mais, pour les améliorer encore et aborder réellement une réalité virtuelle de haute qualité qui aura beaucoup plus d'applications, les chercheurs s'attaquent à la *synthèse d'images réalistes*, qui demande de s'interroger sur les propriétés géométriques des composants de la nature et sur les algorithmes permettant de simuler efficacement ces propriétés ([Cani, 2015](#)). Enfin, la fabrication d'objets physiques à partir de descriptions informatiques est maintenant rendue possible par les imprimantes 3D, dont je ne parlerai pas dans ce livre mais dont les applications explosent à des échelles allant de la dentisterie à la construction des maisons.

La *localisation par GPS* (Global Positioning System), qui nous est devenue si familière, est fondée sur l'émission d'un temps précis par les satellites, qui contiennent des horloges atomiques. La synchronisation fine de ces horloges fait appel à des algorithmes qui doivent traiter les équations de la relativité générale ; de leur côté, les récepteurs doivent, eux, faire des calculs fins de géométrie sphérique en quatre dimensions. Et nos téléphones y associent de beaux algorithmes de recherches rapides d'itinéraires.

D'autres algorithmes permettent de *synchroniser finement tous les ordinateurs du monde* à quelques millisecondes près à partir des bases de temps du GPS et des horloges atomiques terrestres. Qu'on se souvienne que la simple harmonisation des

heures dans le monde n'a été mise en place à grand-peine qu'au début du xx^e siècle ! Bien que peu connu par rapport au fameux système d'exploitation Linux, le logiciel libre NTP (Network Time Protocol) qui permet cette synchronisation globale du monde est une des plus grandes créations collectives de l'informatique : des milliers d'informaticiens ont contribué bénévolement à son écriture et à sa délicate mise au point.

Comme déjà mentionné, les algorithmes issus de la *théorie du contrôle*, une branche de l'automatique mathématique, pilotent nos avions et contrôlent leurs réacteurs et la plupart de leurs fonctions critiques. Ils contrôlent déjà en partie nos voitures, et pourraient bientôt les piloter tout seuls en liaison avec les algorithmes de reconnaissance d'images. Le champ d'application est en fait celui de l'ensemble des systèmes *cyberphysiques*, dans lesquels l'informatique est en interaction constante avec le monde physique. Les robots en sont un autre exemple. Mais attention, les problèmes de sûreté et de sécurité y seront redoutables !

ALGORITHMES D'APPRENTISSAGE AUTOMATIQUE

On parle beaucoup en ce moment des algorithmes d'analyse de très grandes masses de données par traitements statistiques ou par *apprentissage automatique*, surtout depuis leurs grands succès en jeux, analyse d'images, reconnaissance de la parole, traduction des langues, conduite automatique des voitures, aide au diagnostic médical, analyse du comportement des internautes, etc. Ils sont utilisés dans de très nombreux autres domaines et jouent un rôle fondamental dans la science moderne : analyse de données physiques, astronomiques, biologiques, génétiques, radiologiques, etc. À partir de leurs seuls séquenceurs d'ADN, les biologistes pourraient d'ailleurs bien être rapidement en mesure de fabriquer plus de données qu'on ne sera capable de fabriquer de disques pour les stocker et d'ordinateurs pour les analyser. Les algorithmes seront cruciaux pour les compresser et les interpréter.

L'analyse de données massives repose de plus en plus sur les méthodes algorithmiques neuro-inspirées dites de « réseaux de neurones profonds » (*deep learning* en anglais), nées dans les années 1980 au sein du domaine de l'intelligence artificielle, mais qui n'ont réellement montré leur potentiel que récemment grâce à

l'augmentation de la puissance des ordinateurs et à la variété des données disponibles. Ces algorithmes ont été présentés au Collège de France en 2016 par Yann Le Cun, un de leurs créateurs (Le Cun, 2016). Ils sont très différents des algorithmes standard : par nature bien plus analogiques, ils reposent sur des couches multiples de « neurones artificiels », expression à ne pas prendre trop à la lettre, et leur comportement est surtout déterminé par une grande quantité de paramètres internes modifiés petit à petit par l'apprentissage.

Plus précisément, un réseau profond est constitué d'une série de couches d'opérateurs linéaires ou non linéaires (seuils par exemple), chaque opérateur transformant les données entrantes en fonction d'une grande quantité de coefficients internes, en général des nombres organisés en matrices, avant de passer son résultat à la couche suivante. Dans l'apprentissage *supervisé*, dominant actuellement, on fournit au réseau une suite d'exemples annotés, c'est-à-dire accompagnés du résultat attendu. Le réseau compare à ce résultat attendu les résultats qu'il calcule en fonction des données et de ses coefficients courants. Cette comparaison conduit à modifier les coefficients internes par une opération conceptuellement simple appelée *rétropropagation de gradient*. La structure générique de ces algorithmes peut être habilement spécialisée pour traiter des problèmes aussi différents que l'analyse d'images, la reconnaissance de la parole, la traduction des langues, le jeu de go, ou le diagnostic médical. Un but des chercheurs, plus lointain, est de parvenir à un apprentissage *non supervisé*, dans lequel aucune donnée notée n'est fournie au système, voir le [chapitre 11](#).

On voit clairement l'impact de ces méthodes dans l'amélioration récente et considérable des domaines cités dans le paragraphe précédent. Et, bien sûr, un grand événement de 2015 a été la victoire du programme AlphaGo sur Lee Sedol dans le jeu de go, exploit qu'on pensait irréalisable il y a quelques années, et encore amplifié par la victoire sur le numéro un mondial Ke Jie en 2017. AlphaGo est fondé sur un habile mélange d'algorithmes d'apprentissage profond et d'exploration aléatoire très rapide des configurations atteignables.

L'apprentissage profond pose des questions scientifiques nouvelles et très difficiles. La première est de comprendre pourquoi ses algorithmes marchent aussi bien en pratique. Il semble qu'ils associent empiriquement des formes mathématiques simples aux données réelles. Mais, si dans notre espace à trois dimensions les formes

sont les points, les courbes et les surfaces, dans celui de l'apprentissage la dimension de l'espace est déterminée par les millions de paramètres individuels des données (par exemple les pixels d'une image). La recherche de formes simples y devient bien plus délicate : c'est ce qu'on appelle la malédiction de la grande dimension. Les questions scientifiques associées sont fascinantes et vont demander des développements radicalement nouveaux, objets de la chaire de Sciences des données de Stéphane Mallat au Collège de France ([Mallat, 2017](#)).

Pour les utilisateurs, la définition des critères de succès par rapport à la fonction souhaitée est d'autant plus difficile que les algorithmes d'apprentissage sont par essence mouvants. Comment certifier qu'une analyse d'images est suffisante pour conduire une voiture en sécurité, alors que son algorithme va changer en permanence, soit par le téléchargement de nouvelles versions, soit par son propre apprentissage ? Après tout, c'est aussi vrai pour vous qui avez passé le permis de conduire une seule fois vers 18 ans, tout en changeant constamment après. Mais assurer une interaction adéquate entre les hommes et les algorithmes est un problème difficile à surveiller de près en permanence. J'y reviendrai aux [chapitres 7, 10 et 11](#).

ALGORITHMES ET MATHÉMATIQUES

J'arrête là cette énumération qui pourrait occuper un livre à elle seule sans même entrer dans les détails. Mais j'insiste sur un point essentiel : les algorithmes sont des objets de nature mathématique, même si leurs mathématiques ne sont pas celles qu'on apprend à l'école. Ils utilisent souvent des théories et théorèmes mathématiques qui n'ont pas été inventés pour eux. De bons exemples sont le rôle fondamental de l'analyse fréquentielle de Fourier en traitement du signal et en automatique et celui de la théorie des nombres en chiffrement. Mais la plupart des algorithmes actuels sont fondés sur des idées nouvelles, ce qui fait que les informaticiens doivent développer de plus en plus de nouvelles mathématiques pour en parler. Ces développements théoriques sont indispensables pour l'étude de la *correction* des algorithmes, c'est-à-dire de la vérification de leur adéquation au problème posé, ainsi que de leur *complexité*, c'est-à-dire de leur coût en temps de calcul, mémoire et énergie. Ces aspects essentiels sont traités plus en détail dans l'[annexe 12.1](#).

DE LA SÉQUENTIALITÉ AU PARALLÉLISME

Je voudrais maintenant signaler qu'une profonde modification est à l'œuvre dans le domaine des algorithmes. Les premiers algorithmes, comme le fameux algorithme d'Euclide pour le calcul du PGCD de deux nombres (leur plus grand diviseur commun), sont dits *séquentiels* : les opérations sont effectuées une par une, dans un ordre bien précis. Les premiers ordinateurs ont été tout aussi séquentiels, exécutant une instruction à la fois. Le modèle mental sous-jacent était resté celui du mathématicien travaillant avec une seule main et un seul crayon. Les algorithmes séquentiels ont dominé l'informatique jusqu'à la fin du xx^e siècle. Mais, avec la généralisation des réseaux et des machines multiprocesseurs au début du xxi^e siècle, les algorithmes deviennent de plus en plus parallèles, s'affranchissant du crayon unique pour travailler à plusieurs mains. J'en donnerai quelques idées ici, et voir [l'annexe 12.4](#) pour des compléments plus techniques.

Des algorithmes parallèles relativement simples et systématiques ont d'abord été développés pour les besoins du calcul numérique, et en particulier du calcul matriciel pour l'aérodynamique des avions, la météorologie, la physique, etc. Dans un produit de matrices, il n'y a pas de raison de séquentialiser les opérations, et il est naturel d'utiliser simultanément plusieurs processeurs de calcul. Plus tard, quand les réseaux d'ordinateurs sont apparus et quand l'industrie des télécommunications est passée du central électromécanique au central électronique, il a fallu développer de nouveaux algorithmes pour assurer la bonne transmission des données à travers des réseaux complexes : appelés *protocoles de communication*, ils sont par construction répartis entre plusieurs ordinateurs. Puis le Web a systématisé la notion de client/serveur : une session Web implique toujours une interaction permanente entre au moins deux ordinateurs, le vôtre, qu'on appelle le *client*, et celui du site que vous appelez, qu'on appelle le *serveur*. Si ce serveur est un moteur de recherche, il va lui-même faire travailler en parallèle des milliers ou des dizaines de milliers d'ordinateurs pour répondre à votre question et à toutes celles qui lui arrivent en parallèle d'autres utilisateurs. Tout cela sera détaillé dans le [chapitre 5](#).

Les algorithmes parallèles sont omniprésents dans vos ordinateurs et objets informatisés. Le circuit qui gère votre PC ou votre téléphone est devenu *multicœur* : il contient plusieurs processeurs qui communiquent entre eux soit en partageant de la

mémoire, soit en s'envoyant des messages à travers un réseau sur puce. Les différents composants de l'ordinateur accèdent en parallèle à sa mémoire. Les systèmes d'exploitation sont devenus des logiciels parallèles très complexes, qui sont répartis sur les différents processeurs pour gérer l'allocation des tâches sur les différents processeurs et celle des ressources dont ils ont besoin. Et les programmeurs d'applications cherchent eux-mêmes à utiliser au mieux les ressources parallèles de la machine pour améliorer leurs performances. Enfin, la synchronisation du temps entre ordinateurs dont nous avons parlé ci-dessus est en elle-même un gigantesque programme parallèle à l'échelle de la planète. La même chose se produit dans les systèmes cyberphysiques, qui sont eux-mêmes constitués de divers composants physiques. Par exemple dans votre voiture, il faut synchroniser finement les quatre freins pour garder la ligne sur des revêtements irréguliers ; on est dans ce cas dans le parallélisme *temps réel*, puisque le calcul doit respecter les contraintes de temps du système physique. Le temps réel est fondamental pour les systèmes cyberphysiques.

On peut maintenant dire que la programmation parallèle prend le pas sur la programmation séquentielle à l'ancienne. Malheureusement, l'algorithmique parallèle est d'un ordre de grandeur plus difficile que la séquentielle, les bugs y sont encore plus sournois, et l'approche mathématique y est pour l'instant moins développée. Les chercheurs ont encore beaucoup de pain sur la planche – ce qui n'est pas pour leur déplaire.

Les langages

Les *langages* servent à décrire les fonctions à réaliser et les algorithmes pour les réaliser. Leurs objectifs sont soit la *spécification* de fonctions et d'algorithmes, soit leur *programmation* pour l'exécution en machine. Les langages associés sont en général assez différents, car ces rôles sont distincts. Mais, dans tous les cas, un langage se définit par trois caractéristiques : ses *principes*, sa *syntaxe* (textuelle ou graphique), qui décrit les programmes légaux du point de vue de leur seule écriture, et sa *sémantique*, c'est-à-dire le sens donné à ses phrases.

LA SPÉCIFICATION

La spécification a deux objectifs : la définition précise de ce qu'il faut faire et ne pas faire, et, ce qui est tout aussi indispensable mais souvent négligé, la définition précise de ce qu'il faut vérifier sur la réalisation finale. Les deux critères principaux de qualité sont la *cohérence*, c'est-à-dire l'absence de contradictions, et la *complétude*, qui exprime que le moins possible de décisions importantes doit être laissé aux réalisateurs, et que, quand c'est le cas, cette délégation doit être explicite (par exemple pour des choix de structures de données ou d'algorithmes internes permettant de réaliser une tâche spécifiée, ce qui est du domaine de l'implémentation et pas de la spécification). La spécification est un domaine vaste et difficile, qui ne concerne pas que la description des fonctions à réaliser. Elle doit couvrir aussi des aspects non fonctionnels comme les contraintes liées à la machine d'exécution

(vitesse, taille mémoire, énergie, etc.) et à la sécurité informatique (résistance aux intrusions, protection de données, etc.). Elle doit couvrir aussi des aspects apparemment anodins, comme les messages d'erreur à transmettre en cas de mauvais usage par l'utilisateur. Une faille de sécurité a été ainsi trouvée dans le système de passeports biométriques français : des messages d'erreur trop précis pouvaient permettre de pister le propriétaire d'un passeport au cours de ses passages en douane, ce qui était explicitement interdit par les contraintes internationales ; la spécification n'avait pas mentionné les messages d'erreur (voir [chapitre 9](#)). Enfin, et c'est souvent négligé, la spécification doit comprendre une description suffisamment précise de l'environnement du système à réaliser, indispensable pour vérifier le système dans ses conditions réelles de fonctionnement.

Comme on le voit, tout cela est lourd. La faiblesse des spécifications, leur complexité excessive ou le fait qu'elles changent constamment font échouer bien des projets. De plus, quand elles existent, les spécifications sont souvent écrites dans des documents purement textuels ou graphiques dont il est difficile de vérifier la cohérence et la complétude. Tout ceci sera précisé aux [chapitres 9 et 10](#).

LES FORMALISMES DE SPÉCIFICATION

Les chercheurs et industriels travaillent depuis longtemps sur la création de langages formels ou semi-formels pour les spécifications, afin de les rendre plus claires et plus précises, de permettre d'automatiser certaines vérifications critiques, voire de supprimer les tests. Souvent créés avec des approches uniquement pragmatiques, beaucoup de langages de spécification graphiques ou textuels ont été proposés, mais sans jamais rencontrer le succès escompté. Les vrais succès sont plutôt à attendre de langages de *spécification formelle*, à base de logiques mathématiques, comme nous le verrons au [chapitre 10](#). Le lecteur intéressé pourra trouver beaucoup plus de détails dans les cours 2014-2016 que j'ai donnés au Collège de France.

LES LANGAGES DE PROGRAMMATION

La situation est assez différente pour les *langages de programmation*, ceux qui servent à écrire les programmes qui tourneront vraiment sur les ordinateurs. Le sujet est très actif et créatif depuis les débuts de l'informatique. Il a démarré dès les années 1950, quand les informaticiens ont réussi à sortir des difficultés du langage machine déjà présentées plus haut en concevant des langages de beaucoup plus haut niveau, qui permettaient d'écrire en termes de concepts abstraits humains au lieu de cases mémoires et d'instructions machine au ras des pâquerettes. Historiquement, le premier langage de (relativement) haut niveau fut FORTRAN, conçu par John Backus en 1954 pour le calcul scientifique ; il fut rapidement suivi par COBOL, conçu par un comité en 1959 pour les applications de gestion, et auquel l'informaticienne Grace Hopper a beaucoup contribué. Depuis ces temps anciens, le sujet a explosé, de nouveaux langages étant assez systématiquement introduits dès qu'il y a de nouveaux besoins, ce qui est fréquent.

Comme pour nos langues parlées, il existe une bonne dizaine de styles linguistiques assez différents fondant des milliers de langages ou dialectes, qui sont soit d'utilisation générale, soit spécifiques à des domaines d'application. La situation n'est donc pas du tout la même qu'en mathématiques, dont le langage est quasiment fixé pour le monde entier depuis la seconde moitié du xx^e siècle⁵. Comme on le devine aisément, il existe aussi une bonne dizaine de chapelles se regardant souvent en chiens de faïence. Je n'utiliserai pas de langage de programmation dans ce livre plutôt général, me bornant à en décrire les aspects les plus importants dans les [annexes 12.5](#) et [12.6](#). Les ouvrages et enseignements sur ce sujet abondent. Le lecteur pourra aussi se référer à la vidéo d'un cours du Collège de France que j'ai donné à l'Inria Rennes-Bretagne-Atlantique en 2015 ([Berry, 2015](#)).

LES COMPILATEURS

Mais un ordinateur ne fonctionne qu'avec des programmes écrits dans son propre langage machine. Pour pouvoir utiliser des langages de haut niveau, il a fallu développer des méthodes algorithmiques pour faire traduire par les ordinateurs eux-mêmes les programmes de haut niveau en langages machine ; les programmes les mettant en pratique sont appelés des *compilateurs*. Les compilateurs sont des objets

extrêmement importants puisque toutes les réalisations informatiques passent par eux. Il peut d'ailleurs exister beaucoup de compilateurs académiques ou industriels différents pour un langage donné. Mais les compilateurs sont très complexes, surtout lorsqu'on demande un haut niveau d'efficacité du code généré. L'[annexe 12.4](#) explique comment construire des compilateurs efficaces en s'appuyant sur la technique fondamentale du *bootstrapping*, déjà mentionné au chapitre précédent.

Comme nous le verrons au [chapitre 10](#), les informaticiens développent maintenant des méthodes très puissantes pour calculer encore plus profondément sur les programmes, en particulier afin de détecter leurs erreurs avant même de les exécuter. C'est ce qu'on appelle l'*analyse statique* des programmes. En bref, calculer sur les programmes avec d'autres programmes est devenu une des activités favorites des informaticiens.

Les machines et circuits électroniques

Quand on parle d'informatique dans les médias, c'est souvent en termes d'ordinateurs et de réseaux. C'est normal, puisqu'on reste dans la tradition du xx^e siècle : la matière, l'énergie et les ondes y sont directement visibles, contrairement aux cas des algorithmes et des logiciels. De plus les ordinateurs et réseaux sont bien évidemment les moteurs de l'informatique, et ils méritent cette attention même si elle est souvent trop concentrée sur eux. Je ne parlerai ici que des circuits électroniques qui forment leurs cœurs de calcul, et encore sommairement, d'une part parce qu'ils ne sont pas au centre de mon propos, d'autre part parce que l'histoire de leur évolution a déjà été très bien racontée dans un superbe livre illustré sur l'histoire de l'informatique ([Lazard & Mounier-Kuhn, 2016](#)). Je ne parlerai donc pas des mémoires ni des interfaces de périphériques, mais les réseaux seront discutés au [chapitre 5](#).

LA LOI DE MOORE

Le phénomène dominant dans l'industrie des circuits a été la *loi de Moore* stipulée en 1975 (dans sa dernière version) par Gordon Moore, créateur d'Intel. Cette loi disait que le nombre de transistors par unité de surface allait doubler en gros tous les deux ans. Elle a été parfaitement respectée depuis. Mais il faut bien comprendre que le mot « loi » est ici un peu trompeur, car il ne s'agit pas du tout d'une loi au sens de la physique : ce n'est rien d'autre qu'une *décision économique* concertée de toute

l'industrie pour respecter cette échéance. Son extraordinaire succès est clairement à mettre au crédit des physiciens des matériaux semi-conducteurs, qui ont fait des progrès technologiques demandant énormément d'imagination et de compétence pour repousser tous les obstacles, ainsi qu'à celui des fabricants des appareils et robots ultraprécis utilisés dans les « fonderies » de circuits – cette fois, le mot vient carrément du XIX^e siècle. Et il ne faut jamais oublier que les circuits sont conçus et vérifiés à l'aide de grands logiciels de CAO, développés par une grande industrie d'un très haut niveau. En bref, la conception d'un circuit, c'est essentiellement du logiciel très haut de gamme, et sa fabrication de la physique et de la robotique très haut de gamme.

L'effet de la loi exponentielle de Moore a été considérable : de quelques microns de largeur, les fils et composants imprimés sur des circuits par procédé photographique ont été progressivement amincis jusqu'à leur taille actuelle d'une dizaine de nanomètres pour les circuits les plus récents (en 2017). Le gain a donc été de l'ordre de 1 000 en taille linéaire. Mais ce gain passe au carré en pratique, puisqu'un circuit est rectangulaire. L'impact sur le nombre de transistors est donc encore plus considérable : ce nombre est passé de 2 400 pour l'Intel 4004, le premier microprocesseur intégré, à une vingtaine de milliards pour les plus gros processeurs graphiques actuels (GPU = *graphics processing units*).

Mais la loi de Moore n'exprime pas que la « puissance de calcul » des ordinateurs double tous les deux ans, contrairement à ce qu'on entend pourtant souvent. La puissance de calcul d'un circuit n'est pas une notion facile à caractériser, car elle dépend de deux facteurs au minimum : sa *fréquence d'horloge* et son *architecture*. Nous allons en parler ci-dessous. Et il ne faut jamais oublier que la loi de Moore n'est pas la seule responsable de la progression exponentielle des performances du matériel informatique. Ce qui compte vraiment, ce n'est pas que le microprocesseur, c'est le système global avec ses mémoires de masse, ses écrans, ses liens de communication et ses réseaux. Tous ces composants, fort différents les uns des autres, ont suivi le mouvement. Nous devons un grand merci aux physiciens, avec par exemple l'invention de la magnétorésistance géante en spintronique par Albert Fert et Peter Grünberg (tous deux prix Nobel de physique 2007), qui a révolutionné le stockage sur disque, et l'amélioration constante des écrans LCD (*liquid crystal display*) et des fibres optiques.

LA FRÉQUENCE D'HORLOGE

La *fréquence d'horloge* détermine la vitesse de séquençement des instructions machine dans un circuit. Elle a crû considérablement au cours du temps, passant de quelques mégahertz (millions de cycles par seconde) au début des années 1980 à plusieurs gigahertz (milliards de cycles par seconde) au début des années 2000, soit un gain d'un facteur 1 000 en vingt ans. Cependant, pour des raisons de dissipation thermique et de consommation électrique, la fréquence d'horloge est restée bloquée depuis quelques années aux alentours de 3 gigahertz, et il n'est plus prévu de l'augmenter. La raison technique est que la fréquence est linéairement liée au voltage, alors que la dissipation thermique est proportionnelle au carré de ce voltage.

On peut même dire qu'on est passé d'une course de vitesse à une course de lenteur. Prenons par exemple un circuit de téléphone portable. Comme il fonctionne sur une batterie de capacité limitée, il faut qu'il consomme le moins possible. Mais il faut aussi que son utilisateur ne soit pas gêné par sa lenteur, et même qu'il soit séduit par sa vitesse. Le jeu est alors d'éteindre des parties du circuit dès qu'elles ne sont plus utiles, et de ralentir le circuit en baissant son voltage et donc la consommation dès qu'il n'est pas nécessaire d'aller vite : la consommation électrique est proportionnelle au carré du voltage. Du coup, un circuit moderne comporte généralement plusieurs zones d'horloges battant chacune à vitesse variable, voire plus du tout quand les zones qu'elles contrôlent ne sont plus utilisées. Gérer la transmission d'informations entre zones d'horloges variables est toute une aventure.

LES PROGRÈS ARCHITECTURAUX : DU CIRCUIT OPTIMISÉ AU SYSTÈME SUR PUCE

Comment continuer à augmenter la vitesse alors que la fréquence n'augmente plus ? Il y a deux moyens d'action principaux, chacun profitant de l'augmentation du nombre des transistors : améliorer l'architecture des grands composants d'un circuit, et intégrer de plus en plus de ces composants sur la même puce pour éviter d'en sortir, ce qui fait gagner énormément en temps et en consommation électrique.

En termes d'architecture de composants, on peut utiliser deux approches complémentaires : améliorer les microprocesseurs d'usage général, et construire des

processeurs spécialisés pour des tâches fréquentes pour lesquelles le microprocesseur standard n'est pas à l'aise. Un bon exemple de processeur spécialisé d'importance majeure est celui du processeur graphique ou GPU (*graphics processing unit*), fondamental pour la vidéo qui effectue de gros calculs sur des matrices de nombres, et aussi pour l'apprentissage profond qui fait de même.

Les microprocesseurs modernes deviennent tellement complexes que la plupart des concepteurs de logiciels ne connaissent même plus leur fonctionnement interne – qu'ils n'ont d'ailleurs plus vraiment besoin de connaître grâce au découplage du matériel et du logiciel. L'idée centrale est de *paralléliser leurs calculs*, ce qui peut se faire de plusieurs façons. Dans les circuits *multicœurs*, on met plusieurs cœurs de calcul sur la même puce, maintenant des dizaines, bientôt sans doute des centaines. Dans chaque cœur, on utilise des *pipe-lines*, qui permettent de découper les instructions en petites rondelles et de traiter plusieurs rondelles en parallèle comme on le fait dans les chaînes de montage de voitures depuis Henry Ford. On peut ensuite *calculer spéculativement* : quand on doit exécuter un test « si c alors p sinon q » où c est un test booléen et p et q sont des suites quelconques d'instructions, il se peut qu'on puisse déjà exécuter p et q alors même qu'on ne peut pas encore évaluer c , par exemple parce qu'il fait appel à des ressources pas encore disponibles. On exécute alors spéculativement les deux branches p et q en parallèle, et dès que la valeur de c est connue, on garde la bonne branche et on jette la mauvaise, en avortant ses calculs s'ils ne sont pas finis. Et on peut bien sûr imbriquer récursivement plusieurs exécutions spéculatives à l'intérieur de p et de q . On gagne du temps, mais on perd de l'espace et de l'énergie. Facile à dire, mais sacrément difficile à faire... D'ailleurs, les circuits d'architecture très complexe comme les Itanium d'Intel n'ont pas survécu car les utiliser efficacement s'est avéré trop difficile pour les compilateurs de langages de haut niveau.

Pourquoi fait-on aussi des circuits spécifiques comme les GPU ? Simplement parce qu'ils réalisent efficacement des opérations sur lesquelles les microprocesseurs ne sont pas bons. Par exemple, une multiplication de matrices demande de faire un grand nombre de multiplications et d'additions de nombres indépendantes les unes des autres. Comme un microprocesseur n'a que quelques multiplieurs, il va être inefficace. Un GPU est donc construit comme un grand tableau de milliers d'unités de calcul arithmétique travaillant en parallèle, les autres fonctions de gestion du calcul

étant réduites à leur plus simple expression. Les difficultés sont alors de bien gérer les accès mémoire du GPU et de bien faire travailler ensemble microprocesseurs et GPU.

Passons maintenant aux *systèmes sur puces*, ou *SoC (systems on chips)*, qui sont par exemple au cœur des téléphones portables. Ces systèmes rassemblent sur le même rectangle de silicium plusieurs microprocesseurs et un grand nombre de fonctions spécialisées. Outre les GPU décrits ci-dessus, on y trouve les accélérateurs de calcul sur le son, le GPS, la radio, les gestionnaires de mémoire, les entrées-sorties rapides (USB, etc.), les gestionnaires d'écrans et de périphériques divers, etc. Une usine complète en quelque sorte ! Étant intégrés sur la même puce, ces composants peuvent fonctionner indépendamment mais de manière coordonnée en communiquant rapidement par des *réseaux sur puce* sophistiqués, qui ont remplacé les anciens *bus de données* inspirés des cartes électroniques. Les avantages sont nombreux : abaissement du coût de fabrication puisque tous les composants sont « fondus » en une fois, augmentation de la vitesse et réduction de la consommation électrique, etc. Par exemple, le circuit principal d'un téléphone haut de gamme, qui contient actuellement de l'ordre de 2 milliards de transistors, peut comporter des dizaines de fonctions différentes.

L'OUTILLAGE LOGICIEL DE CONCEPTION DES CIRCUITS

Le passage aux architectures sophistiquées et aux systèmes sur puce conduit à une complexité gigantesque, avec comme contrainte additionnelle qu'il faut garantir que tout soit prêt en même temps puisque le circuit est fondu en une seule fois et qu'il est impossible de corriger ses défauts de conception une fois fondu : contrairement au logiciel, la matière ne se télécharge pas. La seule solution est de fournir des modèles logiciels du circuit pour mettre au point les logiciels qu'il exécutera avant même que ses plans soient prêts, illustrant encore une fois la puissance du *bootstrapping*.

On peut dire que la conception et la vérification d'une famille de circuits modernes sont devenues aussi difficiles et aussi chères que le développement d'un avion de ligne, alors que la durée de vie industrielle de l'objet produit est beaucoup plus courte. Heureusement, un grand principe simplificateur reste à l'œuvre : les

architectes et designers⁶ de circuits n'ont quasiment plus à se soucier d'électronique. Ils travaillent de plus en plus au niveau des portes logiques en utilisant des logiciels de conception assistée complètement informatisés, et appellent d'autres logiciels très complexes construits en commun avec les électroniciens pour transformer ces circuits logiques en circuits physiques et savoir précisément quelle taille va faire le circuit physique final et à quelle fréquence il va fonctionner avant même qu'il soit fabriqué. L'industrie qui fabrique l'ensemble des logiciels dédiés à la conception des circuits est appelée CAO électronique, et elle est elle-même très complexe et de grande taille. Ses acteurs, créés à la fin des années 1980, sont des ex-start-up de plusieurs milliards de dollars de chiffre d'affaires total : Synopsys, Cadence Design Systems, Mentor Graphics, etc.

UNE INDUSTRIE TRÈS LOURDE ET EN MUTATION

Ce qui est très important mais peu connu, c'est que l'industrie des circuits physiques est devenue très lourde, même si elle fabrique un des objets matériels les plus légers qui soient. Elle devient en fait de plus en plus lourde : suivre la loi de Moore conduit à des usines de plus en plus chères, dont le coût augmente avec le nombre de transistors des circuits. Il est désormais de plusieurs milliards de dollars, avec une durée de vie des usines plutôt faible (de l'ordre de cinq à dix ans pour le haut de gamme). Enfin, les erreurs de conception logiques ne peuvent pas être réparées physiquement, sauf pour les circuits reconfigurables ou FPGA (*field-programmable gate arrays*), dont je ne parlerai pas ici. Un des derniers bugs publiés d'Intel, dû au mauvais dimensionnement d'un transistor sur 1,5 milliard, a coûté officiellement 1 milliard de dollars. Cela fait que beaucoup d'acteurs historiques abandonnent le domaine, qu'ils jugent trop capitalistique et trop risqué. Il ne reste plus que quelques fonderies dans le monde, principalement en Asie et aux États-Unis, et il n'est pas clair que l'Europe gardera encore longtemps des fonderies de circuits au meilleur standard mondial, ce qui va certainement lui poser des problèmes stratégiques.

Mais la conception logique des circuits est moins risquée, surtout maintenant que conception et fabrication sont bien séparées. Certains acteurs comme Intel et Samsung conservent et intègrent les deux aspects. D'autres, comme beaucoup de fabricants de

téléphones (qui en produisent au total plus de 1 milliard par an), font fondre les circuits à la demande en fonction de plans fournis par des sociétés de design – les *fabless*, comme on les appelle dans le domaine. L'Europe dispose quelques *fabless* majeures, dont la principale est la compagnie anglaise ARM, reine du microprocesseur basse consommation. Ses processeurs et circuits annexes sont embarqués à plusieurs exemplaires dans la plupart des téléphones et dans bien d'autres objets. Ils sont actuellement fabriqués à quelque 20 milliards d'exemplaires par an. Une autre *fabless* bien connue est Qualcomm, intégrateur majeur des circuits de téléphones (dont des ARM).

LA LOI DE MOORE VA-T-ELLE SURVIVRE ?

L'avenir pourrait ne pas ressembler au passé : l'industrie a décidé d'arrêter de suivre globalement la loi de Moore dans quelques années, car les problèmes de design et de CAO augmentent très vite quand la taille des composants se rapproche trop de la taille d'un atome, augmentant fortement les coûts associés et le risque d'erreurs. L'économie des circuits deviendrait intenable en continuant comme actuellement. Les conséquences de cette décision ne sont pas encore bien comprises. Pour les gros microprocesseurs visant les meilleures performances possibles, les fabricants ont encore intérêt à pousser la loi de Moore dans ses retranchements. Pour d'autres, comme ceux des téléphones, le combat est plutôt dans l'économie d'énergie et l'augmentation de la taille mémoire. Et une autre question étrange et non triviale pourrait se poser : si nous avons 1 000 milliards de transistors sur un circuit, aurions-nous la capacité intellectuelle d'en faire quelque chose ?

Mais l'abandon de la loi de Moore est aussi un cadeau inestimable à la recherche : depuis les années 1970, les circuits n'ont été conçus que sur un seul principe technologique, celui du silicium dopé. Actuellement en 2D, il pourrait passer en 3D dans un avenir proche, gérant les microcomposants en volume au lieu de seulement en surface. L'impact serait immédiat pour les mémoires. D'autres principes physiques plus rapides et moins énergivores pourraient prendre le relais dans l'avenir, et il sera peut-être enfin économique de les développer. Les matériaux du type graphène semblent avoir un gros potentiel. Mais les algorithmes d'apprentissage, de

nature tant analogique que numérique, pourraient bénéficier de techniques très différentes de celles dominantes aujourd'hui, comme par exemple les *memristors* développés par Hewlett-Packard, ou la spintronique déjà mentionnée.

Les bugs

Jusqu'à présent, j'ai présenté ce qu'apporte l'informatique et sur quoi elle se fonde. Il est temps de parler de son problème majeur en pratique, celui des *bugs*. « Mon ordinateur a buggé » est une phrase qu'on entend souvent quand l'ordinateur s'est « planté » ou a fait quelque chose d'incohérent, mais ceux qui la prononcent n'en connaissent pas souvent le sens réel. En particulier, collés comme d'habitude à la matière, ils ne réalisent pas que le bug est pratiquement toujours dans le logiciel, pas dans la machine. Cette mauvaise compréhension est bien naturelle, car le domaine n'est pas du tout intuitif : les bugs informatiques n'ont que peu de correspondants dans la matière, l'énergie et les ondes. Il me paraît donc essentiel d'expliquer ici leur nature.

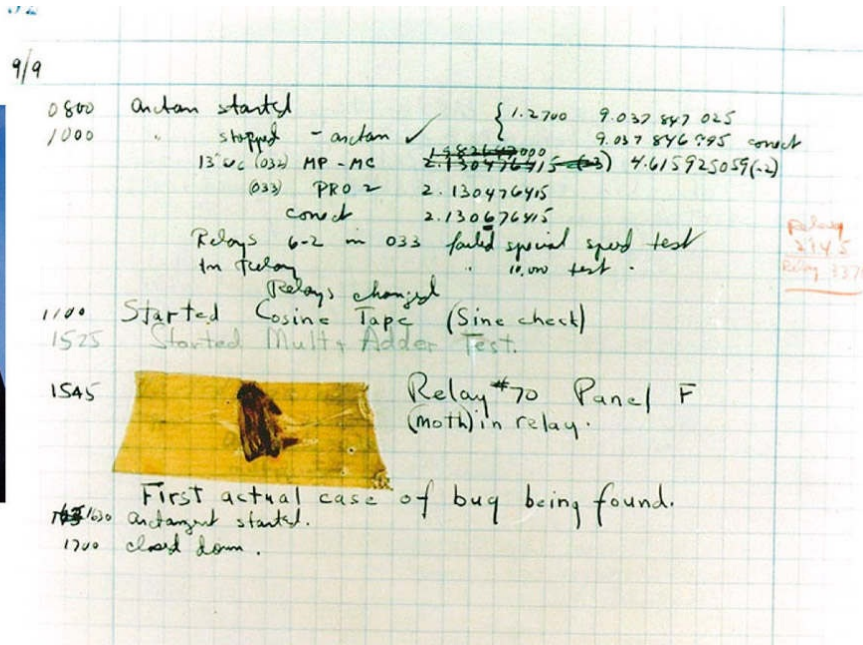


Figure 11. Grace Hopper et le premier bug répertorié.

Le mot *bug* est réputé issu d'un insecte coincé dans un relais de l'ordinateur sur lequel travaillait Grace Hopper en 1947 (voir la figure 11). Mais, contrairement à ce qu'on entend souvent, son usage n'a pas été créé par cet incident. Il était utilisé bien avant, faisant par exemple partie du vocabulaire de Thomas Edison, qui l'utilisait pour parler de la myriade des petits ennuis techniques qui ralentissaient ses projets.

D'OÙ VIENNENT LES BUGS INFORMATIQUES ?

La racine du problème est illustrée dans la figure 12. La programmation doit se voir comme un combat entre deux opposés absolus, séparés par un grand fossé :

- À gauche, l'homme. Ici Johannes Kepler, très intelligent, mais d'une rigueur limitée par les capacités humaines et vraiment lent.
- À droite, le microprocesseur, absolument brutal et stupide par nature, mais hyperexact et hyperrapide.

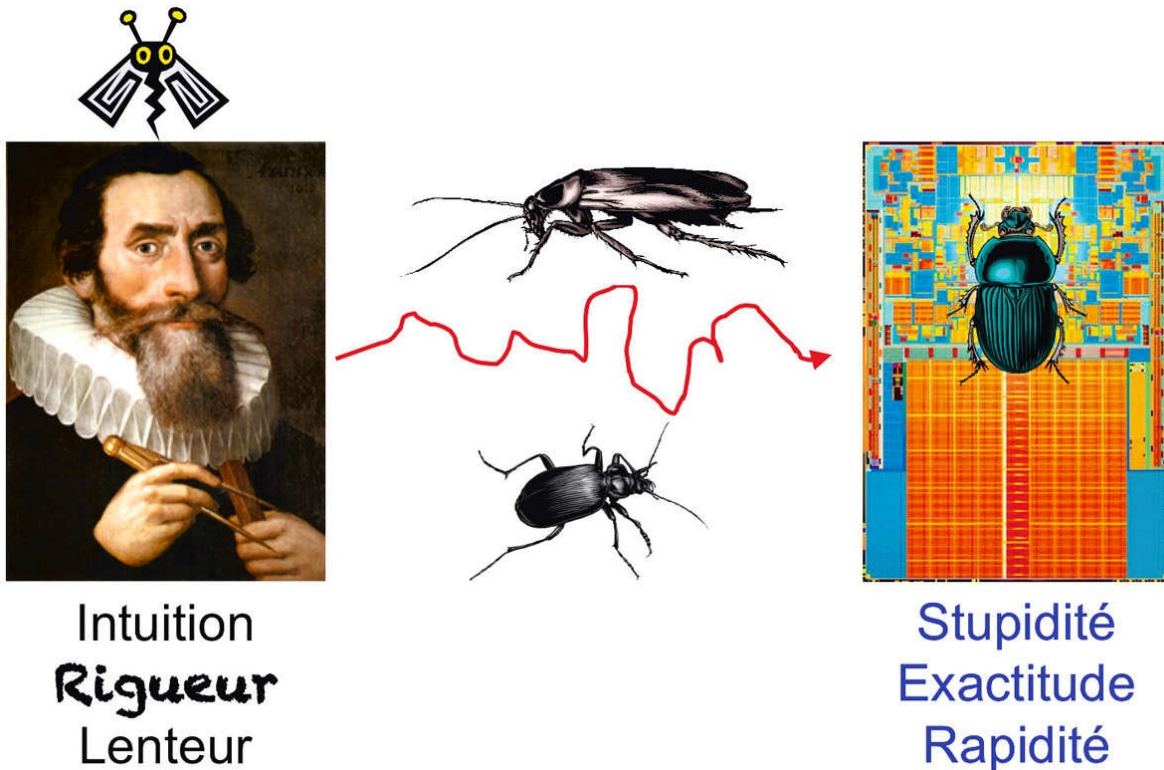


Figure 12. Le gouffre homme/microprocesseur.

Programmer, c'est faire passer les idées algorithmiques des hommes, abstraites par nature, dans le concret du circuit en réussissant à sauter par-dessus le gouffre qui les sépare. Si la technique utilisée n'est pas assez solide, elle provoque inmanquablement des erreurs, c'est-à-dire des bugs. Le vrai problème est que le circuit fait exactement ce qu'on lui dit, jusqu'au bout, et avec une conscience professionnelle totale. Si ce qu'on lui dit est mauvais, il le fait comme c'est écrit, point final⁷.

Donc la réalité est la suivante :

Un bug n'est pas une panne de la machine ni du logiciel, c'est une panne de ceux qui ont écrit le logiciel.

Une fois cela compris, il reste à expliquer pourquoi un programme buggé peut faire vraiment n'importe quoi. C'est le rôle de la figure 13 qui représente Manhattan sans Broadway. Supposons que j'habite à Manhattan-Nord et que je souhaite inviter

un ami qui habite dans le Sud. Bien entendu, la grille de Manhattan ne permet pas un trajet direct à cause des sens interdits et des travaux. Le plus simple pour moi est de dicter à mon ami la suite des directions à prendre à chaque carrefour, en utilisant trois instructions « machine » : T pour tout droit, D pour à droite et G pour à gauche. Le premier programme de la figure 13, en bleu, donne le bon chemin, illustré par la ligne médiane ; les deux programmes suivants correspondant aux lignes latérales droite et gauche résultent chacune d'un bug, c'est-à-dire d'une microerreur *humaine* de lecture, transmission ou écriture de la séquence de commande. Chaque chemin diffère de l'original en un seul endroit, par un virage intempestif de 90 degrés à droite ou à gauche. Ce qui est remarquable, que le résultat est catastrophique : ces minuscules modifications conduisent *n'importe où*, plus exactement à deux *n'importe où* distincts ! On en déduit que, de par son exactitude même, l'ordinateur est le plus extraordinaire *amplificateur d'erreur* jamais construit. Programmer faux est facile mais dangereux, programmer juste est utile mais difficile. Si on ne comprend pas ça, mieux vaut ne pas aborder l'informatique.

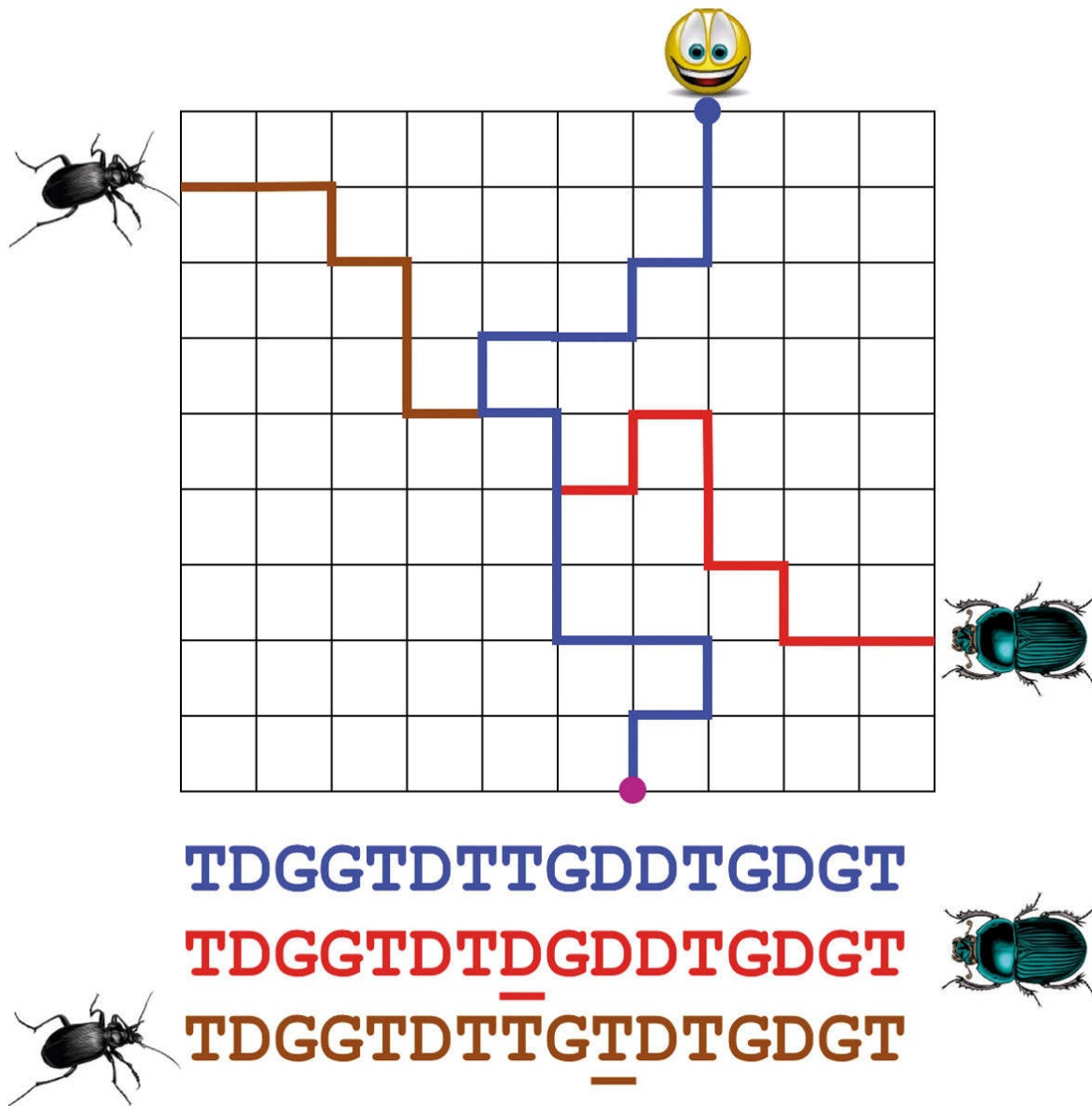


Figure 13. Bugs à Manhattan.

La situation est similaire à celle du fameux chaos déterministe, où l'on dit avec une bonne dose d'exagération qu'un cyclone aux États-Unis peut être créé par un battement d'ailes de papillon au Brésil. Mais, en informatique, il n'y a pas d'exagération car la situation est bien plus fréquente et pénible : il y a beaucoup de bugs, alors que les papillons brésiliens ont bien du mal à créer des cyclones.

POURQUOI LE PARALLÉLISME AMPLIFIE ENCORE LES BUGS



Figure 14. Lise et Laure en interblocage.

Notre programme Manhattan n'explique pas tout, car il est non seulement minuscule, mais strictement séquentiel. Pour les programmes parallèles, les choses se compliquent nettement. Illustrons cela encore une fois par des exemples de la vie courante.

Dans la figure 14, Lise et Laure, jumelles monozygotes qui symbolisent deux programmes parallèles, sont en *interblocage* (*deadlock* en anglais) : chacune veut le crayon et la règle pour dessiner, chacune tient un des deux objets, mais aucune ne veut lâcher le sien.

J'adore les bugs récurrents de l'application de téléphone du journal *Le Monde*, surtout quand elle est détectée comme étant bloquée par le système d'exploitation. Sous le système Android, l'écran affiche alors « Le Monde s'est arrêté » 😊. C'est

probablement la faute d'une Lise et d'une Laure en interblocage cachées dans le code. Sur un iPhone, le téléphone se bloque pour un bon moment, et le seul moyen que j'ai trouvé de le remettre ensuite en état est de supprimer puis recharger l'application. Un tel bug est particulièrement sournois, car le fait que les tâches paraissent momentanément inactives ne veut pas dire qu'elles le resteront toujours. Quand prendre une décision, et laquelle prendre ?

La figure 15 présente un bug encore plus sournois : Lise et Laure sont finalement tombées d'accord pour échanger leurs instruments, et il n'y a plus de blocage de leur côté. Mais Manon, plus petite, n'y a pas droit. Elle est alors dite *en famine* (*starvation* en anglais). Ici le système n'est pas bloqué globalement, mais un de ses composants l'est localement. C'est un bug de ce type qui a failli tuer le premier robot martien Pathfinder juste après son atterrissage. Le [chapitre 9](#) expliquera ce bug plus en détail, et en présentera d'autres allant du croustillant au tragique.

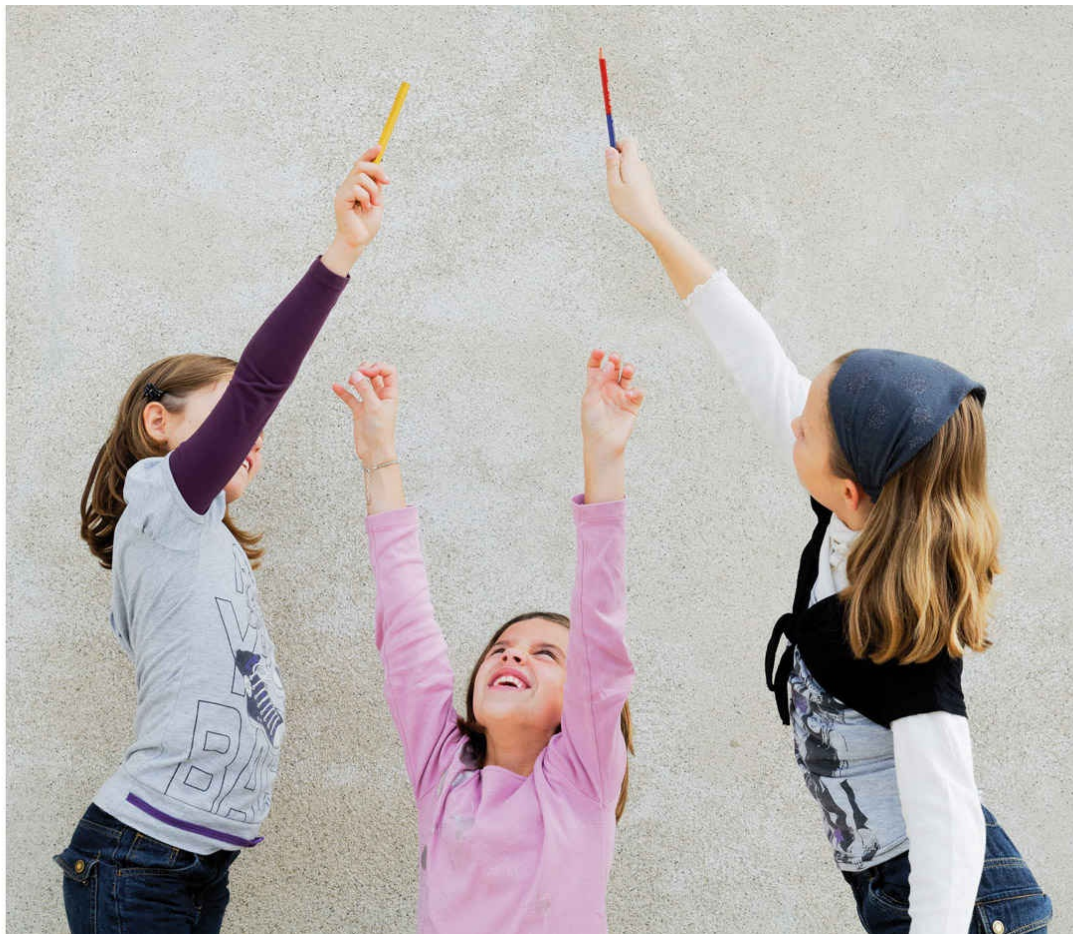


Figure 15. Manon en famine.

PEUT-ON ÉVITER OU CONTRÔLER LES BUGS ?

Une des grandes questions scientifiques de l'informatique est donc la suivante : *peut-on écrire des grands programmes sans bugs* ? On trouve plusieurs attitudes dans l'industrie ou même le monde académique :

1. La version « application de téléphone » en 2017 : on écrit le plus vite possible, on met en ligne, et on corrige les bugs après. Voir la phrase « correction de bugs et amélioration de la stabilité » systématiquement écrite dans le commentaire de chaque mise à jour. Le mot « stable » est devenu synonyme de « qui marche », serait-ce pour cacher les défauts ?

2. On fait de son mieux pour laisser le moins de bugs possible avant la mise en service, par des tests exhaustifs ou des processus de certification normés comme en avionique. Attention, le développement coûte beaucoup plus cher !

3. On utilise des méthodes formelles pour mieux programmer et garantir mathématiquement l'absence de certains bugs, voire la correction totale du logiciel.

La première option est hélas visible sur beaucoup de sites web et d'applications de téléphone. La deuxième option est heureusement la norme dans beaucoup d'industrie et de projets académiques. J'ai déjà mentionné la troisième approche dans la discussion sur les spécifications ; nous l'étudierons au [chapitre 10](#). Pour ma part, j'ai passé ma carrière à chercher comment chasser les bugs (et aussi à en faire), ainsi qu'à introduire les méthodes formelles dans les industries qui y étaient sensibles, en collaboration avec de nombreux collègues bien sûr. J'y ai aussi consacré plusieurs années de cours au Collège de France. Je conseille la maxime suivante, encore peu enseignée hélas :

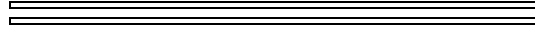
Programmer bien, ce n'est pas programmer vite, c'est programmer sans bugs (ou au moins essayer).

Enfin, on ne peut pas parler de bugs sans les relier aux problèmes de sécurité informatique, qui deviennent critiques partout. La plupart des trous de sécurité des

systèmes informatiques sont dus à des microbugs de conception et de programmation souvent ignorés car non reliés à la fonctionnalité de ces systèmes, mais qui peuvent avoir des conséquences dramatiques : mise hors service, sabotage, corruption, vol ou destruction de données, prise de contrôle à distance de voitures ou drones, chantage à la publication de données confidentielles, rançonnage, etc. Cela sera étudié au [chapitre 9](#), et des pistes de solutions seront présentées au [chapitre 10](#).

DEUXIÈME PARTIE

Études de cas



CHAPITRE 4

Les télécommunications : du fil à l'air

La communication entre les personnes et les institutions a toujours été un enjeu majeur du développement des sociétés. C'est incontestablement un des domaines où l'informatique a apporté le plus, au point que l'industrie des télécommunications, autrefois forte et indépendante, n'est plus vraiment dissociable de l'industrie informatique en général. Au début des années 1970, il était difficile et long d'avoir un téléphone fixe. Maintenant, il devient banal d'avoir un smartphone connecté à Internet, et les objets communiquent entre eux autant que les hommes. Nous allons étudier ici quelques-unes des causes centrales de cette évolution aussi rapide que profonde. Mais, auparavant, je pense utile de s'intéresser à une révolution bien plus ancienne mais tout aussi puissante, celle du télégraphe, surnommé par Tom Standage l'« Internet victorien » ([Standage, 2014](#)).

Le télégraphe, une révolution méconnue

Le télégraphe manuel, introduit par Chappe à la fin du XVIII^e siècle, avait donné à la France une bonne avance : les informations, militaires en particulier, pouvaient être transmises en quelques heures au lieu de quelques jours. Mais la vraie avancée a été la mise en place commerciale du télégraphe électrique à partir de 1838 par Cooke et Wheatstone en Angleterre, puis en 1840 par Samuel Morse aux États-Unis, dont l'appareil et le célèbre code ti-ta-ta-ti allaient s'imposer partout. La révolution qui en a résulté a été aussi rapide et profonde que la révolution Internet. Des câbles ont été posés en un temps record : en 1866, il y en avait plus de 66 000 kilomètres aux États-Unis, et l'océan Atlantique venait d'être traversé par un câble sous-marin. Les Indes ont été atteintes en 1870 et l'Australie en 1872. Tout cela a permis de réduire le temps de transmission des messages intra- ou intercontinentaux de jours, semaines ou mois en minutes. L'avancée a donc été bien plus grande que celles dont nous sommes si fiers actuellement. Le métier de télégraphiste, considéré comme qualifié, a crû à une vitesse impressionnante, et la demande s'est avérée telle qu'il a fallu inventer le pneumatique pour faire passer les messages assez vite du relais d'une compagnie à celui d'une autre. Les systèmes de codage se sont améliorés, avec par exemple la transmission directe de caractères par Émile Baudot, qui a donné son nom à l'unité de vitesse de transmission toujours en cours, le *baud*. Outre les avancées qu'il a permises pour la communication entre personnes, l'industrie et le commerce, le télégraphe a

permis l'essor de la signalisation ferroviaire et l'uniformisation des heures dans les pays et entre les pays, ce qui n'a pas été une mince affaire.

La téléphonie classique

Mais la suprématie du télégraphe a été de courte durée : il a été lui-même déplacé par l'arrivée du téléphone en 1876, avec une grande controverse quant aux inventeurs – Antonio Meucci, Graham Bell ou Elisha Gray. Ce nouvel appareil permettait de transmettre non plus des codes mais la voix. Les standards furent d'abord manuels, le standard électromécanique n'arrivant qu'en 1891¹. L'essor du téléphone a marqué le xx^e siècle autant que ceux de la radio et de la télévision.

LA TÉLÉPHONIE FILAIRE ANALOGIQUE

La téléphonie filaire analogique, celle d'avant les années 1970 (en tout cas en France), est illustrée par la figure 16. Elle se place au centre du triangle matière-énergie-ondes : une onde électrique transporte la voix le long d'une ligne métallique continue entre deux interlocuteurs, quelle que soit leur distance, et les centraux manuels ou électromécaniques servent à établir des contacts physiques entre des fils. La chose est assez simple dans son principe, mais les ressources ne sont pas bien utilisées : il faut autant de fils entre centraux qu'on veut faire passer de communications simultanées, et les fils sont bloqués même pendant les silences. De plus, le bruit augmente avec le nombre de contacts, ce qui rend les communications à longue distance difficiles. Tout cela a posé des problèmes considérables de passage à l'échelle, bien illustrés par le fameux sketch du « 22 à Asnières » de Fernand

Raynaud, qu'il est indispensable de montrer aux enfants pour leur illustrer la France des années 1950 (il est facile à trouver sur le Web).

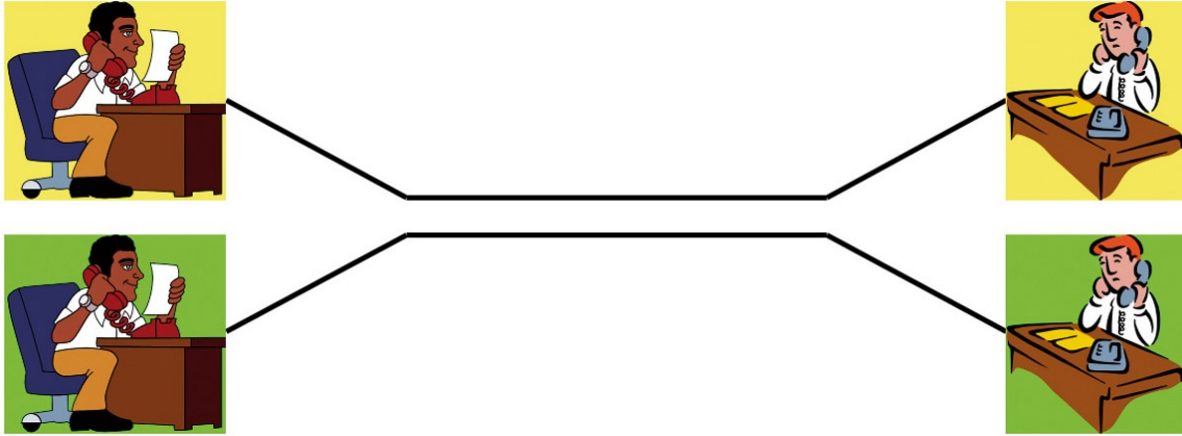


Figure 16. La téléphonie filaire analogique.

LA TÉLÉPHONIE FILAIRE NUMÉRIQUE

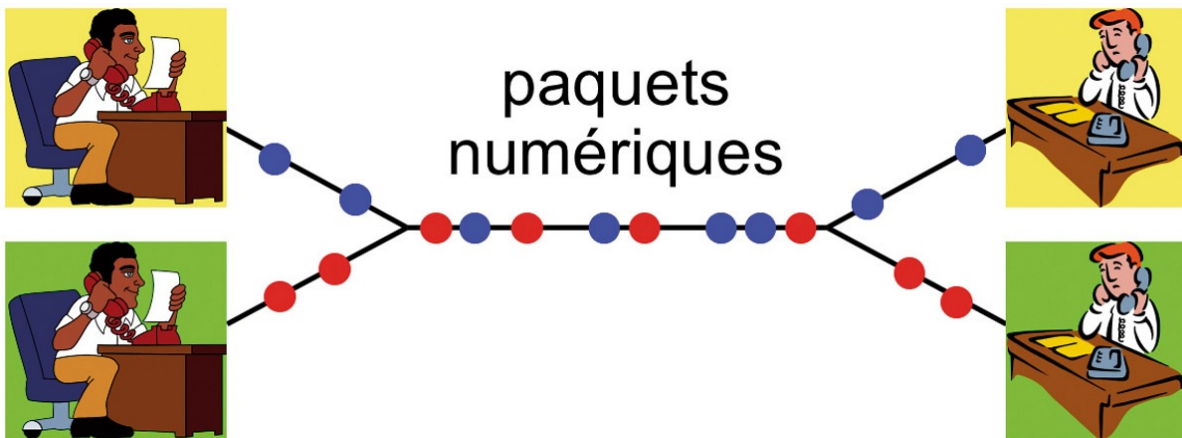


Figure 17. La téléphonie numérique.

Ces inconvénients ont disparu avec la téléphonie filaire numérique, introduite dans les années 1970. Comme illustré à la figure 17, celle-ci conserve les mêmes fils de cuivre entre les utilisateurs et les centraux, ce qui évite de toucher aux infrastructures locales. Mais le central auquel est connecté l'utilisateur devient un

ordinateur dont le logiciel transforme algorithmiquement le flot temporel régulier des paroles en un flot irrégulier de *paquets numériques*, chacun contenant la numérisation comprimée d'un segment temporel de parole. L'irrégularité du flux vient du fait qu'il n'y a plus besoin d'encoder les silences. Ces paquets sont ensuite transférés individuellement entre les centraux par des lignes mutualisées, généralement faites de fibres optiques ou de réseaux hertziens qui assurent des débits d'information bien plus importants que les fils électriques et peuvent aussi faire passer la télévision numérique. À l'autre bout de la fibre, dans le central connecté à l'autre interlocuteur, les paquets numériques sont décodés pour reconstruire le signal électrique analogique qui produira le son final comme avant. La compression numérique de la parole et le multiplexage des paquets permettent de bien mieux utiliser les ressources entre les centraux.

Le central numérique est toujours appelé « électronique ». Mais, comme d'habitude, cette conception matérialiste cache une réalité plus subtile : les centraux électroniques sont gérés par des logiciels gros et complexes, qui gèrent l'établissement des communications, le découpage en paquets et la compression du son, la réservation des ressources tout au long de leur parcours, la restitution du son à l'arrivée, et bien sûr aussi le calcul de la facture. Les bugs n'y ont pas été anodins : le 15 janvier 1990, une seule instruction mal placée dans un programme des centraux d'AT&T, alors leader mondial du domaine, a provoqué une réaction en chaîne nationale qui a conduit à des problèmes massifs d'établissement des communications à longue distance pendant plus d'une dizaine d'heures. Nous détaillerons ce bug au [chapitre 9](#). Ses conséquences financières et probablement humaines ont été considérables.

La téléphonie sans fil

Malgré ses qualités, la téléphonie « électronique » a conservé l'inconvénient majeur de la téléphonie classique : il faut être sur place pour recevoir l'appel. Les compagnies ont eu recours à toutes sortes d'expédients pour diminuer cet inconvénient : vendre des répondeurs téléphoniques, rendre possible de les appeler à distance, permettre de faire suivre automatiquement les appels, etc. Mais ce n'étaient que des palliatifs qui allaient perdre tout leur intérêt avec la généralisation de la téléphonie sans fil. Née dans les années 1950 aux États-Unis, celle-ci permet de s'affranchir de la géographie. Les premiers systèmes, lourds et chers, ont laissé la place au début des années 1980 aux téléphones cellulaires dits « de première génération » (1G), d'abord au Japon, puis dans les pays nordiques. En 1991, la Finlande a lancé la 2G, bien plus efficace et permettant des transferts de données. Comme la téléphonie numérique de la figure 17, elle travaille par paquets, mais les paquets sont transportés par des ondes externes, captés et retransmis par des antennes d'une façon qu'il n'est plus possible de dessiner aussi simplement. La 3G a été lancée en 2001 au Japon, avec des débits de données très supérieurs et bientôt la connexion à Internet. Les nouveaux besoins ont conduit à la 4G actuelle et à l'explosion des *smartphones*, maintenant plus nombreux que les habitants dans bien des pays, et la 5G beaucoup plus efficace pointe déjà son nez. Cette accélération des débits a eu une conséquence imprévue : pour une bonne partie des usages, le téléphone a supplanté l'ordinateur, et le gouffre s'agrandit de jour en jour. Une tendance possible pour l'avenir est qu'on posera son téléphone dans un *dock* ou plus simplement qu'on le gardera dans la poche, et qu'il sera alors automatiquement relié à son écran et sa

souris. Il pourrait alors pour beaucoup remplacer le PC (*personal computer*) qui n'aura plus d'intérêt que pour les professionnels comme moi.

Il est important de rappeler que, sur le plan technique, la mise en place de la téléphonie sans fil est loin d'avoir été une histoire simple. Elle fait appel à des algorithmes extrêmement sophistiqués, qu'il est hors de question de décrire ici. Un téléphone portable « intelligent² », aussi appelé smartphone, est un des objets technologiques les plus complexes jamais fabriqués, contenant des milliards de transistors et des dizaines de millions de lignes de code. Mais il reste beaucoup de gains à faire dans le domaine difficile de la diminution de la consommation électrique, ainsi que dans le traitement des déchets de cette industrie : étant extrêmement intégrés sur le plan physique, les portables sont particulièrement difficiles à recycler.

QUELQUES BELLES INVERSIONS MENTALES

Le téléphone sans fil, jugé au début peu utile et intrusif, a eu du mal à s'établir dans notre pays. À la fin des années 1990, il était de bon ton de dire que non, on n'en avait pas. C'est que beaucoup de gens n'avaient pas encore compris à quel point la souplesse du sans-fil et son affranchissement des contraintes spatiales allaient changer l'ensemble de la communication entre les hommes. Cela se voit de façon encore plus claire dans les pays qui n'avaient jamais vraiment connu le téléphone fixe : dans un gros village indien du Tamil Nadu, j'ai vu l'arrivée de la première ligne fixe le 1^{er} janvier 2001 !

Une belle inversion mentale compare deux situations éminemment fréquentes, une dans le passé et une maintenant. Quand je téléphonais à des amis dans les années 1970-1990, l'information que j'avais le plus souvent est qu'ils n'étaient pas chez eux. Quand je les appelle maintenant avec mon portable, la question que je pose immédiatement est : « Allô, t'es où³ ? »

Mais la réticence des anciens n'a évidemment pas existé pour les enfants, qui n'ont jamais connu et ne connaîtront sans doute jamais un monde sans portables. Pour eux, il fait partie de la nature préexistante, au même titre que la mer, la montagne, la plage ou le vélo. Une autre superbe inversion mentale est due au petit-fils d'un ami, âgé de 4 ans environ. En arrivant chez son grand-père, il voit un étrange téléphone

avec un fil venant du mur. Sa question est édifiante : « Papy, pourquoi t'as mis un antivol ? »

Et n'est-il pas amusant de voir à quel point le langage ne suit pas la technologie ? Même avec des portables, on passe des *coups de fil* et on *raccroche* quand on a fini 😊. Bon, d'accord, on dit encore « sur-le-champ » et « à tout bout de champ », alors que les laboureurs sont devenus rares...

Du cuivre à l'ADSL et à la fibre optique

On pourrait penser que le téléphone sans fil a rendu obsolètes les milliers de kilomètres du bon vieux fil de cuivre, en particulier avec l'essor des transmissions de données sur Internet dont le besoin en débit dépasse largement celui de la transmission de la voix. Ce n'est pas encore tout à fait le cas, grâce à une invention d'importance majeure faite en 1988 par Joseph Lechleider aux Bell Laboratories d'AT&T : l'ADSL (*asymmetric digital subscriber line*). L'ADSL permet de transformer à peu de frais la ligne vocale en ligne numérique, en utilisant 255 bandes de fréquences parallèles à des fréquences situées bien au-dessus des fréquences vocales (jusqu'à 1,1 mégahertz), chacune gérée par des techniques algorithmiques sophistiquées de correction d'erreurs. Avant l'ADSL, on arrivait à faire passer les données à 64 kilobits par seconde (Kb/s) sur le fil de cuivre, par exemple pour le réseau Transpac et le Minitel, longtemps fiertés françaises car formant le plus grand réseau public du monde avant Internet. Avec l'ADSL, on arrive à faire passer de l'ordre de 16 mégabits par seconde (Mb/s) dans les villes et même 512 Kb/s dans les campagnes lointaines, le débit dépendant de l'atténuation de la ligne et donc de la distance au central. Le facteur de gain va de 8 à 256 ! L'ADSL a permis des débits suffisants pour l'accès direct à Internet et aux chaînes de télévision numérique (au moins dans les villes), et, par un amusant retournement de l'histoire, a réussi à absorber la téléphonie vocale en faisant passer la voix sur Internet à l'aide du protocole VoIP, *voice over Internet protocol*, dont nous reparlerons dans le chapitre suivant.

Cependant, l'entretien des lignes de cuivre coûte cher et est visiblement de moins en moins bien assuré. Les opérateurs se tournent massivement vers la fibre optique, qui permet des débits bien plus grands et une fiabilité bien meilleure. Mais il n'est pas dit qu'elle atteigne les campagnards comme moi avant longtemps, malgré les promesses récurrentes...

Les progrès en algorithmiques de codage et de transmission

Même si les progrès en vitesse et qualité de la téléphonie sont spectaculaires et constatés par tous, il est rare qu'on s'interroge sur les raisons qui les ont permis. Une fois encore, ils viennent de deux sources conjointes, la physique et la théorie de l'information. Si la physique a permis d'améliorer considérablement les fibres optiques ainsi que les antennes émettrices et réceptrices, des changements tout aussi importants ont été à l'œuvre grâce à l'informatique. Ils se placent dans les algorithmes, circuits et logiciels qui gèrent l'optimisation logicielle et matérielle des ressources (longueurs d'onde, allocation temporelle et fréquentielle des paquets, gestion fine du réseau et des utilisateurs, etc.), ainsi que dans une révolution récente de la théorie et la pratique des *codes correcteurs d'erreurs*.

Comme je l'ai déjà mentionné, la théorie de la communication a été créée par Claude Shannon en 1948. Il s'intéressait en particulier au débit maximum d'information numérisée qu'on peut atteindre sur une ligne bruitée, ce qu'elles sont toutes en pratique. Le paramètre à prendre en considération est le *rapport signal/bruit* de la ligne, mesuré en décibels (dB). Pour simplifier, plus ce rapport est bas, plus la communication est mauvaise, et donc plus les bits d'un message peuvent se trouver erronés une fois arrivés. Comment garantir une transmission sûre malgré un taux d'erreur potentiellement élevé ? Il est nécessaire d'introduire de la *redondance* dans le message envoyé, en lui ajoutant des bits dédiés à la détection et la correction d'erreurs. Deux méthodes simples sont ici insuffisantes : le « je répète » utilisé

souvent en transmission vocale, bien trop coûteux en bits et peu efficace si la ligne est vraiment bruitée, et l'ajout d'un bit de parité mis à 1 si le nombre de bits envoyés est pair et à 0 sinon ; ce bit permet de détecter des erreurs simples, mais pas de les corriger.

Le grand apport de Shannon a été de calculer précisément le nombre n de bits minimal à ajouter à un message de taille m donnée pour le transmettre avec une probabilité de succès p par une ligne dont on connaît le rapport signal/bruit⁴. Shannon a montré qu'une façon optimale d'ajouter ces n bits était de les calculer à l'aide de fonctions aléatoires, c'est-à-dire de les tirer au sort de façon systématique et équitable en fonction des bits du message source. Mais cette méthode n'est pas utilisable pratiquement, car la complexité du décodage du message total obtenu serait exponentielle. La difficile théorie mathématique des codes correcteurs d'erreurs s'est ensuite grandement développée, fournissant divers codes efficaces, dont aucun n'approchait cependant la limite de Shannon.

Une grande révolution a été la création des *turbocodes* par Claude Berrou et Alain Glavieux dans les années 1990. Ces codes permettent d'atteindre en pratique la limite de Shannon grâce à un mécanisme itératif et probabiliste résolument nouveau et de coût raisonnable. Ils sont utilisés en télécommunications spatiales, en téléphonie mobile 3G et 4G, et dans bien d'autres domaines. D'autres codes nommés LDPC (*low-density parity check*) avaient été introduits par Robert Gallager en 1960, mais on ne connaissait pas d'algorithme efficace pour les traiter jusqu'en 1996, date à laquelle de nouveaux algorithmes ont été inventés pour eux. Turbocodes et LDPC se partagent désormais le monde des télécommunications, car les LDPC ont été préférés pour la 5G en téléphonie mobile.

Enfin, il est important de dire que les développements constants de la théorie de l'information sont à la base de tout ce qui concerne le *routage* des paquets dans les réseaux et les systèmes de réseaux virtuels *pair à pair* (*peer to peer*) placés au-dessus des réseaux physiques, qui suppriment le besoin de serveur central et sont à la base des protocoles de téléchargement massif et de la monnaie virtuelle Bitcoin, etc. Nous en reparlerons au chapitre suivant.

CHAPITRE 5

Internet, des hommes aux objets

Internet, dont le nom signifie littéralement le « réseau des réseaux », est la partie la plus visible de l'informatique. Pour la plupart des gens, il se réduit au Web, la toile d'araignée des sites avec lesquels ils peuvent interagir, bien qu'il contienne d'autres fonctions importantes. Internet existe depuis longtemps pour les chercheurs, mais il n'a réellement atteint le public qu'à la fin des années 1990, avec l'explosion du Web puis l'entrée fracassante des moteurs de recherche qui allaient tellement modifier les usages qu'on n'arrive même plus à se souvenir de comment on faisait les choses avant eux.

Les impacts d'Internet sont constamment discutés dans les cercles les plus variés. Il homogénéise le monde et offre des possibilités entièrement nouvelles dont on découvre tous les jours la puissance. Il pose aussi des problèmes fondamentaux dont beaucoup ne sont pas vraiment nouveaux mais se trouvent amplifiés de plusieurs ordres de grandeur par l'universalité et la vitesse du grand réseau : l'accès à l'information et les questions relatives à sa pertinence et à sa possession, la diffusion ultrarapide et à grande échelle d'informations justes ou fausses et de courrier « électronique » important ou ennuyeux, la facilité des contacts mais aussi les atteintes possibles à la vie privée, et, de façon générale, la sécurité des données et objets qui y sont connectés. Les discussions associées sont difficiles, les avis souvent

tranchés et pas forcément bien renseignés, les affirmations trop positives et les angoisses exagérées nombreuses. J’y reviendrai au [chapitre 11](#).

Mon objectif dans ce chapitre est plutôt d’explicitier quelques-unes des *causes* qui font que l’Internet est devenu ce qu’il est et pourra continuer sa croissance. Ces causes sont souvent méconnues, alors même qu’elles resteront celles qui permettront de faire évoluer le grand réseau encore plus rapidement en connectant non seulement les hommes mais leurs objets de tous les jours. Mais le mot « cause » n’est pas forcément bien adapté ici ; je souhaite plutôt décrire ce qu’on appelle en anglais les *enablers*, mot bien précis qui ne peut hélas être traduit en français que par une périphrase floue comme « ce qui rend le reste possible ». Les *enablers* d’Internet sont souvent associés à de véritables inversions mentales. Mon but est enfin de montrer qu’Internet pose *par essence* des problèmes majeurs et difficiles, dont l’ensemble des questions de sécurité numérique est un bon exemple. Là aussi, vouloir traiter ces problèmes sans s’intéresser à leurs causes m’apparaît être une fiction dangereuse.

La grande inversion mentale du réseau Internet

En tant que réseau, Internet repose sur une idée qui est en totale contradiction avec les fondements des télécommunications électriques ou électroniques classiques : transmettre l'information par une succession de petits *paquets de données* arbitraires (*datagrammes*) indépendants les uns des autres, et surtout *faire gérer les problèmes de qualité de la transmission par les ordinateurs ou téléphones des utilisateurs* et pas par le réseau lui-même. Ces idées, nées dans les années 1960, ont d'abord été mises en place à petite échelle dans le réseau ARPANET développé par la défense et la recherche américaine à la fin des années 1960, ainsi que dans le réseau Cyclades développé en France par Louis Pouzin et Hubert Zimmermann au début des années 1970. Techniquement, le réseau Internet est fondé sur le protocole de transmission *IP* (*Internet protocol*) et sur des extensions dont beaucoup connaissent le nom mais pas forcément le sens : TCP, UDP, FTP, VoIP, etc.¹. Je vais les expliquer brièvement. Vinton Cerf, chercheur à Stanford, y a joué un rôle majeur et est considéré comme l'inventeur principal des aspects réseaux d'Internet. Il a reçu pour cela le prix Turing 2004.

LA RÉSERVATION DE RESSOURCE EN TÉLÉPHONIE
CLASSIQUE

Avant de décrire les principes d'Internet, commençons par ceux de la téléphonie classique. En téléphonie électromécanique, la composition d'un numéro provoque l'établissement d'une ligne électrique continue entre les téléphones émetteur et récepteur de la communication, ce qui demande de déclencher une succession d'allocations de lignes et d'établissements de contacts dans un certain nombre de centraux intermédiaires. Un processus similaire reste à l'œuvre en téléphonie électronique filaire classique, celle que nous utilisons toujours pour nos téléphones fixes. La composition d'un numéro valable commence par établir un *routage* de la communication en calculant le chemin de central à central entre le téléphone de l'appelant et celui de l'appelé. Ce chemin fixe, calculé algorithmiquement en fonction de la carte du réseau, sera utilisé pour toute la communication entre les deux interlocuteurs. Mais, au lieu de contacter physiquement des fils comme auparavant, on procède à la réservation de ressources en mémoire et temps de calcul dans les ordinateurs des centraux concernés ; cela permet de garantir l'acheminement correct de tous les paquets de la communication jusqu'à sa fin signalée par le raccrochage de l'appelant ou de l'appelé, qui provoque la désallocation des ressources.

Cette méthode d'allocation statique de ressources possède deux avantages : elle garantit la qualité optimale de toute communication correctement établie, et, pour la compagnie de téléphone, elle rend simple la tarification et l'établissement des factures. Mais elle a aussi trois inconvénients : d'abord, la saturation des ressources à un point quelconque de l'itinéraire prévu provoque le non-établissement de la communication ; ensuite, toute panne de central ou de ligne sur l'itinéraire provoque la perte totale de la communication ; enfin, il faut mettre à jour la carte du réseau dès que des centraux sont supprimés, ajoutés ou en panne, ce qui devient de plus en plus complexe quand la taille du réseau augmente.

LE PROTOCOLE IP D'INTERNET ET SON PRINCIPE DE MEILLEUR EFFORT

À l'opposé, le protocole IP d'Internet est fondé sur l'idée que chaque paquet doit être traité de façon indépendante des autres, en étant acheminé par le réseau selon un principe non plus d'exactitude mais de « meilleur effort » (*best effort*). Les paquets

contiennent chacun l'adresse de leur destination selon un format fixe et les centraux sont remplacés par des *routeurs*. Quand un routeur reçoit un paquet, deux cas se présentent : si l'adresse du destinataire est celle d'un équipement directement connecté à lui-même, il le lui transmet ; sinon, il cherche un autre routeur à qui transférer le message, de préférence dans une direction qui le rapproche de la destination. Le choix de cet autre routeur se fait localement à l'aide de *tables de routage* gérées par des algorithmes probabilistes et adaptatifs très subtils, fondés sur des échanges de messages auxiliaires entre routeurs qui leur permettent d'estimer en permanence l'état du réseau. Les difficiles théories de l'information et des files d'attente sont critiques pour l'efficacité d'un tel mécanisme décentralisé.

Mais rien n'est magique : les paquets peuvent se perdre individuellement en cas de panne d'un routeur ou une ligne de transmission. Il est aussi possible qu'un paquet prenne un itinéraire qui s'avère inefficace ou tourne dans un cycle du réseau. Pour éviter l'encombrement du réseau par des paquets trop vieux, l'émetteur spécifie dans chaque paquet une *durée de vie*, qui est le nombre de routeurs maximum que le paquet doit traverser. Tuer les paquets errants est une fonction essentielle des routeurs.

Ces mécanismes dynamiques possèdent un avantage majeur par rapport à ceux statiques de la téléphonie : celui de rendre le réseau *modulaire* par construction. Puisque le calcul de la trajectoire se fait progressivement et d'une façon purement locale, il n'y a plus besoin de carte du réseau. L'ajout ou la suppression d'un routeur à n'importe quel moment se font sans demander aucune action spécifique sur le routage, les tables se mettant progressivement à jour de façon automatique. La difficulté de la gestion globale du choix des itinéraires disparaît donc complètement, et le passage à l'échelle pose beaucoup moins de problèmes.

TCP : COMPENSER LA PERTE DE PAQUETS

Selon le type de communication désirée, une perte de paquets peut être acceptable ou non. Par exemple, si l'on transmet un document écrit ou une photo, il faut vraiment transmettre tous ses paquets. Dans ce cas, on utilise le protocole fiabilisé d'échange de données *TCP* (*transfer control protocol*) placé au-dessus d'IP et exécuté sur les ordinateurs émetteurs et récepteurs de la communication. TCP gère la continuité de la

transmission de bout en bout d'un document ou d'une session interactive en renvoyant du récepteur à l'émetteur des messages d'acquittement ou de non-acquittement de paquets, qui provoquent dans ce dernier cas leur retransmission. Mais les paquets ne sont plus garantis d'arriver dans l'ordre, puisqu'ils peuvent prendre des routes différentes. Il faut donc leur donner un numéro d'ordre et les réordonner à l'arrivée. Ce numéro d'ordre sert aussi à redemander à l'émetteur l'envoi des paquets non reçus, qui peuvent tout à fait arriver en double s'ils se croisent avec le message de redemande ; les doubles doivent alors être détruits. TCP a aussi d'autres fonctions d'optimisation que je ne décrirai pas ici. L'ensemble IP plus TCP donne ce qu'on appelle *TCP/IP*, sigle souvent prononcé mais rarement expliqué.

Cette description sommaire ne donne bien sûr qu'une idée générale, les détails techniques étant très subtils pour assurer la correction et l'efficacité du protocole. Mais elle rend clair le fait que c'est aux utilisateurs et plus au réseau de s'assurer de la bonne transmission des données, réalisant ainsi l'inversion mentale suggérée plus haut.

ACCEPTER LA PERTE DE PAQUETS

Dans d'autres cas, la perte de paquets est acceptable car elle peut être compensée par d'autres types d'algorithmes. Par exemple, pour une conversation vocale à travers Internet, il est indispensable de suivre au mieux le temps réel pour assurer la continuité de la communication. Requérir la retransmission d'un paquet de voix non arrivé serait une mauvaise idée car cela demanderait de rompre la restitution vocale pendant un temps indéterminé ; il est bien meilleur d'accepter les pertes de paquets et de les compenser par des algorithmes de synthèse adaptative de sons intermédiaires, en exploitant le contenu des paquets effectivement reçus pour fabriquer un son continu acceptable. Ces algorithmes adaptatifs peuvent être très subtils et ne seront pas détaillés ici.

Mais cela implique que la qualité de la communication vocale peut varier selon la charge du réseau, ce que savent bien les utilisateurs des systèmes dits *VoIP* (*voice over IP*) offerts par les fournisseurs d'accès pour la téléphonie sur Internet ou par les logiciels de téléconférences. De même, la transmission continue de sons ou vidéos

haute définition reste problématique pour qui ne possède pas une connexion rapide. Certains souhaitent qu'Internet soit aussi doté de protocoles à qualité de service garantie comme en téléphonie, mais ce sujet est difficile et ne semble pas à l'ordre du jour.

Au total, les avantages d'IP, TCP, VoIP, etc., ont supplanté leurs inconvénients de façon évidente en permettant l'accroissement exponentiel de la taille du réseau. Il est fort douteux que de nouveaux réseaux de téléphonie classique soient déployés à l'avenir.

LA FAMINE DES ADRESSES IP

Il y a cependant un problème d'adressage, dont l'importance ne fait que croître au cours du temps. Au cœur d'Internet, le protocole IP de routage existe en deux versions : *IPv4*, qui date de 1980, et *IPv6*, finalisé en 1998. La différence entre les deux est majeure : IPv4 utilise des adresses sur 32 bits, alors qu'IPv6 utilise des adresses sur 128 bits.

La taille de 32 bits était normale en 1980, car elle correspondait à la taille standard des mots dans les ordinateurs de l'époque. Mais un calcul simple montre que le nombre d'adresses distinctes IPv4 est limité à 2^{32} , qui vaut à peu près 4 milliards. C'est bien moins que le nombre d'hommes, et, comme nous le verrons plus loin, il va vite y avoir beaucoup plus d'objets que d'hommes sur Internet. On ne peut pas adresser proprement le monde avec 32 bits. Pour suivre la croissance très rapide du réseau, il a fallu trouver toutes sortes d'astuces, en introduisant par exemple des parties visibles et des parties souterraines dans le réseau. En pratique, l'ordinateur sur lequel je tape possède plusieurs adresses IP suivant la façon dont je le regarde : par exemple, une interne au sous-réseau qui gère mon câble ethernet, une autre pour le wi-fi, et une adresse externe que vous pourriez utiliser si je vous la donnais (n'y comptez pas).

Le protocole IPv6 résout la question de façon beaucoup plus simple avec ses adresses IPv4 à 128 bits, qui donnent un nombre astronomique de possibilités d'adressage. On peut ainsi donner une adresse unique à chaque ordinateur ou objet

connecté en ne réutilisant jamais la même adresse, comme pour le numéro de Sécurité sociale des Français.

Passer de IPv4 à IPv6 a l'air simple, mais ne l'est hélas pas, entre autres, car il faut changer assez profondément tous les algorithmes de routage. En pratique, le passage est bien plus lent qu'on ne l'espérait, mais il est clair qu'il devra se faire un jour ou l'autre – la Suède a bien réussi à passer de la conduite à gauche à la conduite à droite².

Le World Wide Web

Pour l'utilisateur moyen, Internet se réduit à la toile d'araignée mondiale, le World Wide Web, ou Web en abrégé, qui lui permet d'accéder à une multitude de serveurs de façon homogène. Le Web a été introduit par Tim Berners-Lee (prix Turing 2017) et Robert Cailliau au CERN de Grenoble, à l'origine pour le partage des informations au sein de ce très grand laboratoire de physique et le travail collectif sur ces informations. Coup d'essai, coup de maître, l'avenir ayant montré la pertinence de leurs idées et la puissance de leurs réalisations. Le Web repose encore une fois sur des idées radicalement nouvelles par rapport à ce qui existait auparavant dans les réseaux classiques de type Minitel.

Si le principe client-serveur d'une discussion entre un utilisateur (client) et un ordinateur (serveur) est conservé, il est considérablement magnifié. L'adressage des serveurs depuis n'importe où dans le monde ne se fait plus à l'aide de numéros de téléphone mais à l'aide de *noms universels* ou URL (*uniform resource locators*) comme « www.college-de-france.fr », et l'on peut se connecter simultanément à un nombre quelconque de serveurs. Ensuite, les données à transmettre sont encodées de façon uniforme dans le langage *HTML* (*hypertext markup language*). Ce langage permet de spécifier comment une page doit être affichée et d'autoriser la navigation entre pages à l'aide d'*hyperliens* distribués comme <http://www.college-de-france.fr>, qui permettent d'accéder par simple clic à n'importe quelle autre page. Les pages sont transmises par le protocole *HTTP* (*hypertext transfer protocol*) placé au-dessus de TCP/IP. Et le Web n'est pas nécessairement lié à Internet : il peut être rendu

accessible par des passerelles sur tous les types de réseaux non IP, par exemple les réseaux de domotique où chaque lampe électrique peut recevoir une adresse Web.

Il existe d'autres protocoles, comme *HTTPS* (*HTTP secure*) pour les communications cryptées, *FTP* (*file transfer protocol*) pour le transfert de fichiers, etc. Le nom du protocole est spécifié par ce qui précède le « : » dans l'hyperlien, comme dans <https://mabanque.com> ou <ftp://ftp-sop.inria.fr> pour le serveur de transferts de fichiers de l'Inria Sophia-Antipolis.

Enfin, un *navigateur Web* multifenêtre donne une interface homogène au monde entier, faisant ainsi disparaître des particularités des ordinateurs individuels. Notons que ces navigateurs devenus familiers sont devenus des logiciels extrêmement complexes, comportant des millions de lignes de code.

Les notions de pages de données et de lien symbolique cliquable entre informations n'ont pas été introduites par Internet. Elles sont en fait anciennes, et étaient déjà utilisées dans des systèmes de gestion de documents internes à un ordinateur comme HyperCard d'Apple. Mais l'idée de les généraliser à l'ensemble des documents mondiaux en les standardisant est certainement une de celles qui ont le plus transformé le monde à la fin du xx^e siècle.

LES GRANDES ÉVOLUTIONS DU WEB

Au début, les pages Web étaient formées de textes, d'images et de formulaires de dialogue simples. Mais elles ont rapidement évolué pour accepter des contenus et des interactions beaucoup plus riches, avec en particulier le son, la vidéo et l'intégration de programmes exécutables sur le client. Cela a demandé de nombreuses extensions du langage HTML, ainsi qu'un gros travail sur l'universalité et l'efficacité des navigateurs qui est toujours en cours au sein d'une compétition féroce.

Cette évolution bien visible par les utilisateurs a demandé des évolutions majeures mais moins visibles de l'architecture de programmation du Web. Dans le Web primitif, le navigateur était un programme fixe qui recevait les pages du serveur, celui-ci ne gardant aucune mémoire des transactions. L'idée que le serveur ne devait pas garder de mémoire était très naturelle, car les transactions pouvaient être interrompues à tout moment pour cause de panne du réseau ou d'abandon par

l'utilisateur. Des données intermédiaires gardées dans le serveur n'auraient alors servi qu'à l'encombrer, sans que l'on puisse savoir quand s'en débarrasser.

Mais l'absence de mémoire dans les interactions s'est avérée vite pénalisante, car elle ne permettait pas des échanges suivis entre le client et le serveur. La notion de *cookie*³ est alors apparue, désignant des informations codées sous formes textuelles et stockées par le navigateur de façon cachée aux utilisateurs. Les cookies sont modifiés par l'interaction client/serveur et automatiquement transmis au serveur à chaque fois qu'on le contacte. Ils permettent donc d'implémenter une *mémoire dans le client* transmissible au serveur. De plus, les cookies peuvent demander au serveur d'appeler des programmes quelconques qui peuvent eux-mêmes continuer le dialogue avec l'utilisateur de façon arbitraire. Au lieu que le programme soit réduit au serveur, le programme est désormais *dans le serveur*. Ce mécanisme d'interaction fournit une souplesse bien plus grande, et a lui-même créé de nouveaux enjeux techniques : comme il fallait désormais que les concepteurs de site puissent écrire des programmes complexes pour engendrer dynamiquement de nouvelles pages Web interactives, de nouveaux langages de programmation spécialisés sont apparus, comme *PHP* (récursivement, *PHP hypertext preprocessor*).

Mais les *cookies* ne permettaient de faire que des choses simples, limitées chez le client à celles que sait faire nativement le navigateur. La notion de Web 2.0, assez floue, est apparue pour désigner tout ce qui pouvait augmenter les possibilités des utilisateurs, en particulier pour les faire collaborer entre eux le mieux possible au sein de nouveaux mécanismes comme ceux des réseaux sociaux. Techniquement, la méthode employée a été d'augmenter les capacités des navigateurs de façon presque arbitraire en permettant aux pages Web de contenir aussi des programmes exécutables localement par le navigateur. Ces programmes ont d'abord pris la forme de simples scripts, avant d'évoluer vers des programmes généraux écrits à l'intérieur des pages HTML dans le langage de programmation *JavaScript*. Ils sont envoyés aux clients par les serveurs pour être directement exécutés dans les navigateurs des clients, qui connaissent *JavaScript*⁴. Enfin, pour traiter des structures de données arbitrairement complexes, de nouveaux standards de représentation textuelle des données comme *JSON* (*JavaScript object notation*) ont été introduits. Le programme global est maintenant *réparti entre le client et le serveur*, et peut même l'être entre plusieurs clients et plusieurs serveurs qu'il devient possible de faire travailler ensemble.

LES SERVICES WEB

Au début du Web, les serveurs se contentaient d'envoyer des pages Web à afficher par le navigateur. Progressivement, ils ont aussi pu être utilisés comme des *services Web*, c'est-à-dire comme des sous-programmes appelables à distance avec des arguments quelconques pour rendre des résultats quelconques destinés à être exploités par des programmes plutôt qu'affichés. Toutes ces données transmises sont systématiquement encodées dans des formats textuels par un procédé qu'on appelle le *marshalling*. La notion de service Web permet aussi de se détacher des navigateurs si on le souhaite, car un service Web peut être facilement appelé depuis n'importe quel programme exécutable écrit dans n'importe quel langage.

Le Web devient donc autant un réseau de services distants qu'un réseau d'affichages de pages, ce qui est une évolution majeure. La *composition de services* permet de construire de nouvelles applications à partir d'existantes, par exemple en appelant les services de Google et Tweeter pour afficher dynamiquement les tweets émis dans Paris là où ils ont été émis sur la carte de la ville. Avant la généralisation des services Web, cela aurait été quasi impossible.

LE WEB CACHÉ DES APPLICATIONS TÉLÉPHONIQUES

Plus récemment, les technologies du Web ont été transportées hors des ordinateurs classiques avec la notion d'*application* introduite par Apple pour son iPhone, puis généralisée à tous les téléphones et toutes les tablettes. Les applications peuvent être *développées* (le mot programmer étant apparemment passé de mode) en utilisant les outils du Web et les navigateurs, mais elles cachent les fonctionnalités génériques des navigateurs en les remplaçant par des interfaces spécifiques. De plus, l'écran étant petit et les utilisateurs généralement non informaticiens, le multifenêtrage classique des navigateurs qui permet de travailler sur plusieurs sites à la fois a été abandonné au profit du monofenêtrage des applications. Mais, qu'on ne s'y trompe pas, ce sont bien les technologies du Web qui sont à l'œuvre en arrière-plan, même si on ne les voit plus directement.

VERS UNE PROGRAMMATION DISTRIBUÉE GÉNÉRALE

Ces nouvelles techniques de programmation du Web rendent possible le développement d'applications de plus en plus complexes faisant intervenir plusieurs serveurs et si besoin plusieurs clients, comme les jeux en réseau, les applications de recherche et d'écoute de musique, ou la téléconférence. On les appelle applications *multiparticipantes*, ou *multitier* en anglais. Par exemple, une application de lecture de musique peut télécharger des fichiers MP3 depuis plusieurs serveurs, glaner en parallèle sur d'autres serveurs les biographies des compositeurs et interprètes, permettre d'envoyer ses appréciations à ses amis en temps réel, etc., toutes ces fonctions étant bien sûr présentées de façon unifiée et élégante sur l'écran de l'utilisateur. La programmation du Web se dirige ainsi vers une programmation distribuée générale, qu'on appelle la *programmation diffuse*.

Mais cette programmation diffuse est particulièrement difficile à maîtriser car elle doit faire collaborer des ordinateurs distants tout en traitant toutes sortes d'événements et d'erreurs provenant de l'utilisateur, du réseau et des serveurs. Or les nombreux langages du Web, souvent développés incrémentalement avec des vues de court terme au fur et à mesure de l'évolution des besoins, se fondent des constructions de bas niveau⁵ et ignorent régulièrement les avancées scientifiques et techniques qu'ont connues les langages de programmation classiques au cours des quarante dernières années. Cela contribue à la qualité assez médiocre de beaucoup d'applications, difficiles à mettre au point et dont chaque mise à jour comporte désormais la mention rituelle « correction de bugs, amélioration des performances et de la stabilité » (une expression pudique pour admettre sans le dire que ça ne marche pas encore bien).

NOTE PERSONNELLE. Pour aider à résoudre ces problèmes, mon collègue Manuel Serrano de l'Inria Sophia-Antipolis a développé le langage *Hop.js* compatible avec JavaScript et spécifiquement dédié à faciliter la programmation diffuse. Sa version initiale *Hop*, dérivée du langage Scheme, avait été présentée en séminaire lors de mes cours 2009-2010 et 2013-2014⁶. *Hop.js* permet d'écrire un code source unique en automatisant les transferts de programmes et de données entre serveurs et clients. L'affichage précédemment mentionné des tweets sur la carte de Paris se fait en

quelque 20 lignes de code. L'usage de Hop.js rend cette programmation distribuée du Web bien plus simple et sûre. Un de mes travaux de recherche effectué avec Manuel Serrano et notre thésard Colin Vidal consiste à étendre Hop.js avec une couche réactive nommée HipHop.js⁷, inspirée de mon langage synchrone Esterel qui sera sommairement présenté dans l'[annexe 12.6](#). HipHop.js facilite l'*orchestration* des activités, c'est-à-dire le traitement maîtrisé des actions, événements et erreurs apparaissant à l'exécution quand on essaie de les coordonner. Cette orchestration est un des points les plus difficiles de la programmation Web. Nous pensons que bien la gérer à l'aide de langages bien plus riches intellectuellement que ceux qui existent actuellement sera nécessaire pour les applications complexes à venir.

Les moteurs de recherche

L'invention du moteur de recherche d'échelle mondiale est certainement une de celles qui ont le plus changé nos comportements : il fait maintenant partie de nos outils les plus familiers et les plus utilisés, au point que nous préférons souvent chercher dans un moteur plutôt que dans nos têtes ou dans les livres. Ce n'est pas que de la flemme : le Web contient beaucoup plus d'informations que nos têtes et nos livres !

Les moteurs modernes combinent quatre grandes idées⁸. La première idée est celle du *collecteur*, mot français officiel depuis 2013 pour traduire l'anglais *crawler*, nageur du Web. Partant d'une page assez centrale du Web (yahoo.com par exemple, disait-on lors de la naissance des moteurs), un collecteur ramasse les pages Web de proche en proche, en suivant les liens hypertextes contenus dans les pages déjà ramassées. La deuxième idée est de construire un grand dictionnaire des mots que ces pages contiennent ; comme le Web est multilingue et multialphabets et qu'il contient beaucoup de noms propres, de sigles et de fautes d'orthographe, ce dictionnaire atteint vite des millions de mots. La troisième idée est de fabriquer un très grand index associant chaque mot à toutes les positions dans lesquelles il se trouve dans toutes les pages qui le contiennent, cet index rendant ensuite facile la recherche des pages contenant des mots ou des suites de mots contigus et donc l'ensemble des réponses possibles à chaque question posée. La dernière idée, qui se révéla la plus déterminante et contribua à faire la fortune de Google, est celle du classement des réponses en fonction de différents critères, souvent secrets mais dont un des principaux est leur « popularité », critère très habile que je décrirai plus loin.

Le premier collecteur à analyser le texte intégral des pages collectées a été WebCrawler en 1994. Le premier moteur de recherche d'échelle mondiale a été AltaVista, créé par Louis Monier et Michael Burrows en 1995 chez Digital Equipment Corporation (DEC), alors deuxième constructeur mondial d'ordinateurs. AltaVista reposait sur une autre idée très originale pour l'époque : s'affranchir des bases de données classiques sur disques et mettre au maximum l'index en mémoire centrale pour améliorer l'efficacité de plusieurs ordres de grandeur. Tournant sur une machine DEC Alpha avec 130 gigaoctets de mémoire centrale et 500 gigaoctets de disques, chiffres énormes à l'époque, AltaVista indexait 100 millions de pages Web. Son succès a été explosif, et il a rapidement répondu à 13 millions de questions par jour.

Une chose dont je ne suis pas fier : je rendais souvent visite à Louis Monier à cette époque, et je lui demandais ce qu'il pouvait bien y avoir d'intéressant à faire ce genre de travail ; même des gens qui se pensent entraînés à la nouveauté peuvent rater des tournants historiques...

L'ALGORITHME PAGERANK

Cependant, avec l'explosion du Web, les pages rendues par AltaVista sont vite devenues trop nombreuses pour que l'utilisateur s'y retrouve. À la fin des années 1990, Sergey Brin et Lawrence Page, alors étudiants en thèse à Stanford, ont mis le doigt sur la bonne solution en inventant l'algorithme PageRank de classement des pages, publié en 1999. Négligeant de finir leur thèse, ils ont créé une petite société nommée Google⁹, qui a tout de suite crû exponentiellement et dont tout le monde connaît le devenir. L'idée initiale de PageRank était d'utiliser pour le classement divers critères syntaxiques comme le fait que les mots apparaissent directement dans les liens menant à une page ou dans le titre de la page, mais surtout un critère de *popularité* permettant de mettre les pages populaires avant celles qui le sont moins. Mais comment définir la popularité ? Une idée simple était de dire qu'une page était populaire si beaucoup d'autres pages pointaient dessus. Mais cette idée n'était pas satisfaisante, car il aurait été facile d'augmenter la popularité d'une page en créant des pages auxiliaires dont la seule fonction aurait été de pointer sur la page en question.

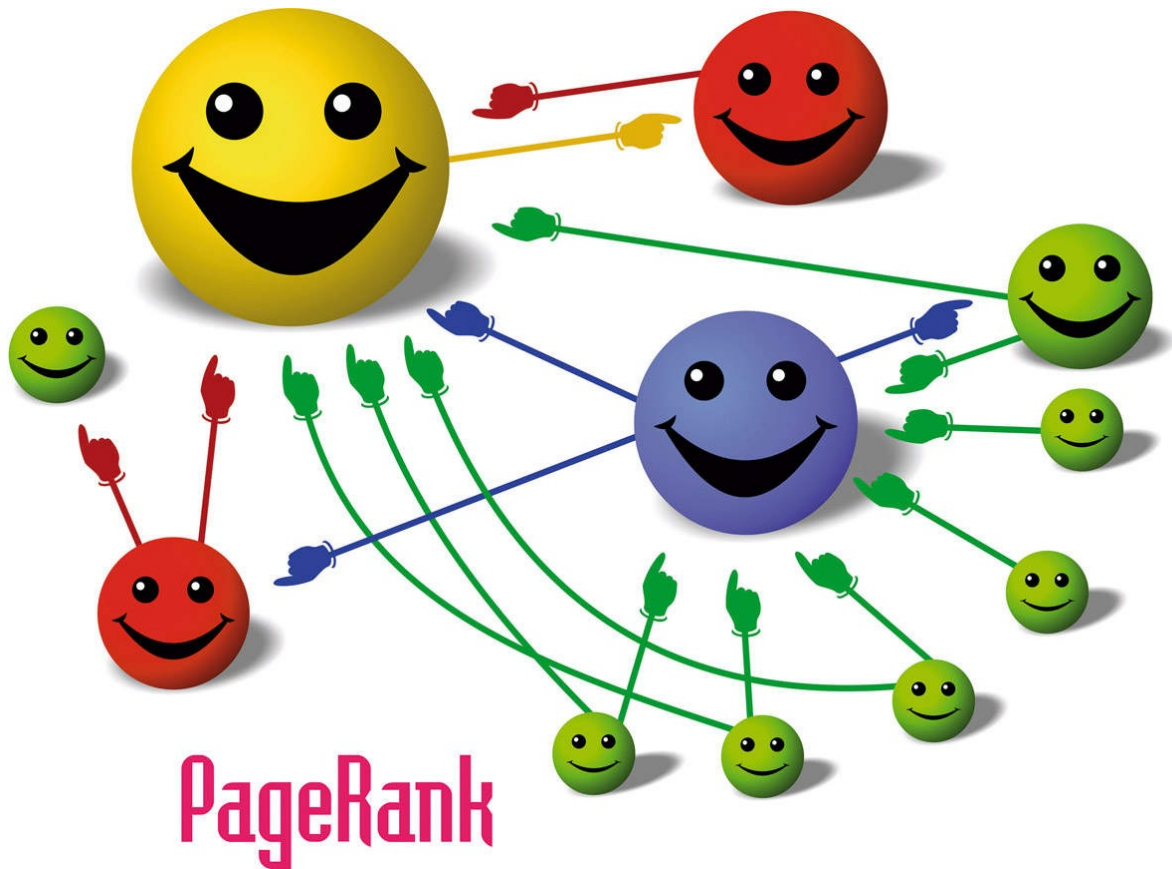


Figure 18. Résultat de PageRank (source Felipe Micaroni Lalli, CC BY-SA 2.5, <https://commons.wikimedia.org/w/index.php?curid=2776582>).

La bonne idée, illustrée par la figure 18, a été de dire qu'une page doit être vue comme populaire *si et seulement si elle est pointée par de nombreuses pages elles-mêmes populaires*.

Tout cela s'exprime par une formule mathématique simple mais profonde, qu'il est utile de présenter ici au moins pour les amateurs de formules. Selon les notations de Brin et Page, appelons N le nombre total de pages, A la page courante, $T_1, T_2, \dots, T_n$ les pages pointant sur A , et $C(T_i)$ le nombre de liens sortant de la page T_i . La probabilité (popularité) $PR(A)$ est définie par l'équation suivante en fonction de celles des $PR(T_i)$:

$$PR(A) = (1 - d)/N + d \times (PR(T_1)/C(T_1) + \dots + PR(T_n)/C(T_n))$$

Vu dans l'autre sens, chaque page T_i distribue sa probabilité multipliée par d équitablement à toutes ses pages sortantes, alors que toute page a une probabilité $(1 - d)/N$ d'être atteinte par un saut aléatoire¹⁰.

Au premier abord, une telle définition se mord la queue et ne constitue même pas une définition récursive simple avec une condition initiale claire comme en raffolent les informaticiens, puisque tout dépend de tout et réciproquement. Mais c'est en fait une équation mathématique classique dite de *point fixe* qui définit implicitement sa solution et pour laquelle on peut montrer qu'une solution unique existe¹¹.

Heureusement, on peut la résoudre de façon approchée par des méthodes itératives bien connues, ou encore en réalisant des simulations aléatoires simples. Une des méthodes est de modéliser un utilisateur du Web par un processus aléatoire qui parcourt le Web de lien en lien avec un comportement simple dépendant d'un paramètre d compris entre 0 et 1 (mis en pratique autour de 0,85) : quand il visite une page, il continue avec une égale probabilité d/m vers la cible de l'un des m liens contenus dans cette page, ou bien, avec la probabilité complémentaire $1 - d$, saute vers une page prise au hasard dans le Web. Pour éviter d'être bloqué par une page ne contenant aucun lien vers d'autres pages, il saute alors systématiquement sur n'importe quelle autre page Web, ce qui revient à dire qu'on suppose que toute page sans lien sortant est conceptuellement transformée en page avec un lien sur chacune des autres pages Web. La popularité d'une page est alors approchée par la probabilité que l'utilisateur la visite au cours d'un parcours de taille finie suffisamment grande. Cet algorithme est facile à implémenter et donne de bons résultats en pratique.

LE PROBLÈME DE LA TAILLE DU WEB

Nous avons vu les grands principes des moteurs de recherche. Il y en a maintenant de très nombreux, qui vont des très grands moteurs indexant tout le Web à d'autres plus limités comme celui de Wikipedia, d'une entreprise ou d'un site Web. Mais, au moins pour les grands moteurs, le nombre de pages à traiter est devenu absolument gigantesque, ce qui a un impact direct sur l'infrastructure de calcul et l'ensemble des algorithmes à déployer. François Bourdoncle, créateur d'Exalead, le premier grand moteur français, ose une comparaison intéressante : si les premiers

moteurs de recherche pouvaient se comparer à des fusées de clubs d'amateurs de haute qualité, les moteurs actuels se comparent plutôt à Ariane 5 ; ils sont de plusieurs ordres plus complexes et leur construction devient une industrie lourde. Ils indexent maintenant des *centaines de milliards de pages*, et sont implémentés d'une façon distribuée à grande échelle sur des *centaines de milliers de petits ordinateurs* au lieu d'un seul gros. C'est avec eux que sont nées les grandes fermes de calcul qui font aussi tourner les réseaux sociaux et le *cloud computing*, « calcul dans le nuage » qui modifie à grande allure l'écosystème du calcul mondial. Exalead existe toujours, mais un autre moteur français est Qwant, qui progresse bien et est moins intrusif que la plupart des autres.

Pour comprendre l'efficacité des moteurs actuels, il suffit de poser la question « Google » à Google France. Ce jour du 14 février 2017, le moteur m'a répondu en un rien de temps qu'il y avait 11 430 000 000 résultats pertinents, le premier classé étant bien sûr la page <https://www.google.fr> 😊.

Et les algorithmes de classement se sont compliqués en prenant en compte une variété de plus en plus grande de critères, y compris des critères commerciaux pour entraîner les consommateurs vers des achats ou des publicités à travers des analyses des textes et des comportements des utilisateurs. C'est cela qui rapporte le plus d'argent aux moteurs, et aussi aux réseaux sociaux. Souvenez-vous toujours de la maxime de cette industrie :

Si c'est gratuit, c'est vous le produit.

RECHERCHER DANS LES MÉDIAS

J'ai surtout parlé pour l'instant des moteurs de recherche dans les textes, ceux qui ont lancé l'idée. Mais il y a maintenant aussi beaucoup de travail sur la recherche de musiques, d'images ou de vidéos. Le sujet est nettement plus compliqué scientifiquement, mais il progresse constamment.

Depuis longtemps, des applications comme Shazam permettent de reconnaître une musique enregistrée à l'aide d'un simple téléphone ; elles fonctionnent en calculant des signatures numériques de ces musiques de façon robuste au changement

de volume et de qualité sonore. Plus récemment, des applications comme SoundHound, davantage fondées sur des techniques d'apprentissage automatique, permettent de reconnaître une chanson chantée directement dans le téléphone. Je dois dire que c'est assez bluffant. Pour les images, les algorithmes d'apprentissage par réseaux profonds (*deep learning*) permettent de reconnaître des gens, des animaux et des objets. Le réseau social Facebook utilise ces algorithmes pour digérer la quantité invraisemblable de photos postées tous les jours par son milliard d'internautes, par exemple afin de reconnaître les gens et de le signaler à leurs amis (ce qui est quand même très intrusif).

Par rapport aux années 1990 où le concept a été lancé, les moteurs de recherche ont considérablement évolué et leur développement est devenu une grande industrie et un grand consommateur d'ordinateurs. Mais on ira beaucoup plus loin à l'avenir : de nouveaux projets de recherche concernent la détection automatique des communautés d'intérêt sur Internet, permettant par exemple de diriger les questions qu'on pose vers ceux qui seront les plus aptes à leur donner des réponses pertinentes, ce qui pourra être bien plus intéressant que de subir le matraquage publicitaire « intelligent » habituel qui ne doit pas l'être tant que ça car il continue à me proposer des produits un mois après que je les ai achetés. Ce genre de mécanisme évolué, déjà plus ou moins en place dans certains sites commerciaux, trouvera, j'espère, de nombreuses et intéressantes utilisations non commerciales.

Le pair à pair, un modèle épidémique de diffusion de données

Revenons au modèle client-serveur du Web. Avec l'accroissement de sa taille et le changement de nature des données stockées, du texte à la musique, la vidéo et les films, un nouveau problème de passage à l'échelle s'est posé : l'*étranglement des serveurs*. Servir efficacement des films à des milliers ou millions d'utilisateurs avec des serveurs centralisés est devenu extrêmement difficile et coûteux en termes d'infrastructures matérielles et de communication. Pour contourner cette difficulté est apparue la notion de *diffusion épidémique de pair à pair* (en anglais, *peer to peer* ou *P2P*) : plutôt que de dépendre de serveurs en nombre nécessairement limité, pourquoi ne pas utiliser l'ordinateur de n'importe quel utilisateur comme serveur pour nombre raisonnable d'autres utilisateurs ? Par exemple, si un film est téléchargé par beaucoup d'utilisateurs d'un réseau P2P, le nombre de ses copies augmente automatiquement avec le nombre de téléchargeurs-serveurs. Donc l'échelle de la distribution augmente exactement comme celle de la consommation, annulant ainsi le problème de passage à l'échelle des serveurs classiques. Le pionnier du domaine a été Napster, lancé en 1999.

L'expression « diffusion épidémique » est bien adaptée : une musique ou un film contagieux se multiplieront comme un microbe ou un virus au cours des contacts informatiques entre les utilisateurs, sans même que ces utilisateurs aient connaissance de ces contacts et n'aient eu d'autre chose à faire que de s'inscrire une fois à un réseau pair à pair. La première diffusion épidémique sur le Web a d'ailleurs été celle

du virus informatique CodeRed, qui a très rapidement infecté une grande quantité d'utilisateurs grâce à une stratégie très simple : se répliquer le plus rapidement possible dans des ordinateurs dont les adresses IP sont tirées au sort en permanence, les tirages au sort étant faits par des ordinateurs déjà infectés. Sa distribution a suivi une courbe dite *logistique*, exponentielle au début puis atteignant un plateau avec la saturation des ordinateurs infectables.

Les algorithmes du pair à pair se sont continûment perfectionnés, tout en restant fondée sur l'idée centrale des tirages aléatoires. Pour un gros film dont le téléchargement prend beaucoup de temps, il ne serait pas raisonnable de compter sur un seul utilisateur-serveur qui peut parfaitement se déconnecter en cours de route ; il vaut beaucoup mieux couper le film en rondelles, toutes n'étant pas forcément au même endroit, et télécharger ces rondelles en parallèle en changeant de serveur en cas de rupture de communication. Pour cela, il faut bien choisir quelles rondelles chercher sur le Web et à quel moment, afin de concilier l'intérêt du téléchargeur avec ceux des autres pairs¹². Il est aussi utile d'explorer des relations de proximité avec d'autres pairs, qui peuvent être en termes géographiques, de vitesse de réseaux, ou même d'intérêts communs dans certains types de données¹³. Ces algorithmes sont difficiles.

LÉGAL OU ILLÉGAL ?

Au milieu des années 2000, la diffusion épidémique en pair à pair a augmenté de façon explosive à la fois pour le téléchargement et le *streaming* de musiques et films souvent mis illégalement sur le Web, ce qui a posé de gros problèmes à la fois aux propriétaires de ces contenus, spoliés, et aux opérateurs Internet qui devaient sans arrêt augmenter leur bande passante sans en tirer de revenus additionnels. C'est de là que sont parties les attaques contre la « neutralité du Net », le principe que les opérateurs doivent traiter de la même façon tous les paquets qui transitent dans leurs tuyaux, indépendamment du type de données qu'ils transmettent. Cette histoire n'est pas finie, même si la diffusion illégale semble se réduire avec l'essor du *streaming*.

LA SÉCURISATION DES TRANSACTIONS PAR BLOCKCHAINS

Mais le pair à pair n'est pas utilisé que pour des transferts illégaux, et il semble avoir un avenir radieux dans un autre domaine : celui des *transactions sécurisées*, bancaires ou autres. Prenons l'exemple d'une transaction financière : au-delà d'un montant faible, elle doit être sécurisée par une banque, organisme spécialisé soumis à des contraintes drastiques (hélas pas toujours sur tous les plans). Le principe globalement accepté est qu'une banque est un organisme à qui on peut *faire confiance*. Mais c'est aussi un organisme faillible, qui peut disparaître comme l'a montré un passé récent, et pour lequel l'État ne donne de garantie à ses utilisateurs jusqu'à un certain montant.

La monnaie virtuelle Bitcoin, introduite en 1999, fournit d'autres moyens algorithmiques de faire confiance, qui pourraient s'avérer bien plus solides. Elle fonctionne entièrement en pair à pair, donc sans aucune autorité centrale, et est gérée par des algorithmes sur *chaîne de blocs* (*blockchain* en anglais), qui pourrait en fait être employée pour sécuriser en pair à pair n'importe quel type de transactions récurrentes, financières ou non. Ces algorithmes suscitent donc un intérêt majeur dans beaucoup de domaines. Mais, comme ils sont bien décrits dans de nombreux documents et exposés¹⁴, je n'en parlerai que brièvement ici.

La technologie de la chaîne de blocs repose sur des algorithmes cryptographiques à clef publique classiques pour les transactions individuelles (cf. [annexe 12.1](#)), ainsi que sur des algorithmes originaux groupant les transactions individuelles en blocs cryptés, eux-mêmes validés informatiquement puis liés en une chaîne de blocs (*blockchain*) quasiment incassable car protégée par une signature en forme de chiffrement court du contenu. Les blocs sont dupliqués à un nombre quelconque d'exemplaires dans le réseau, et toute modification de transaction à l'intérieur d'un bloc est rendue quasiment impossible par la façon dont ils sont créés et validés. De plus, la sécurité des blocs et donc des transactions qu'ils contiennent augmente automatiquement avec le temps. Dans tout ce qui précède, « quasiment » signifie avec une probabilité extrêmement faible, ce qui est prouvé mathématiquement. Mais la validation des blocs est par essence un calcul cryptographique compliqué, donc très

énergivore, ce qui peut poser des limites pratiques à l'extension de la technique. L'avenir dira jusqu'à quelle échelle elle peut se développer.

Des objets informatisés à l'Internet des objets

Comme nous l'avons vu dans la première partie de ce livre, la plupart des ordinateurs sont maintenant enfouis dans des objets de toutes sortes, afin de les contrôler et de leur permettre de communiquer avec l'extérieur. Autrefois, ces objets étaient chacun informatisés de façon autonome pour réaliser des fonctions bien spécifiques, et quelquefois organisés en réseaux locaux spécifiques pour se transmettre des informations et être synchronisés. Mais la tendance actuelle est d'adopter aussi la technologie générique d'Internet pour les objets connectés, ce pour trois raisons : simplifier la collecte de données depuis ces objets pour pouvoir ensuite analyser ces données ; envoyer des consignes et des commandes aux objets depuis n'importe quel terminal Internet ; et pour les constructeurs, pouvoir mettre à jour facilement les logiciels qu'ils embarquent¹⁵. On voit par exemple les voitures pouvoir être jointes par autoradio et envoyer elles-mêmes des informations à leur fabricant ou au concessionnaire de leur propriétaire, ou encore l'éclairage et l'électroménager d'une maison être accessibles sur le Web.

Voici une réalité souvent ignorée : les objets connectés sont déjà très nombreux sur le grand réseau, et leur nombre va rapidement exploser pour rendre les hommes largement minoritaires, tant les possibilités nouvelles que cela va offrir sont nombreuses en domotique, gestion des villes, transports, médecine, etc. Et, comme toujours en informatique, des applications autrefois insoupçonnables seront faciles à développer. Par exemple, certaines assurances réalisent que le téléphone du

conducteur contient un GPS et un accéléromètre assez précis, ce qui leur permet de comprendre comment il conduit sa voiture et à terme de moduler ses primes d'assurance. Je laisse au lecteur le soin d'apprécier les qualités et les défauts d'une telle approche.

La sécurité informatique d'Internet

La sécurité informatique a toujours été un sujet important, mais elle est rendue bien plus difficile qu'auparavant par la connexion systématique des ordinateurs, tablettes et téléphones des particuliers, celle des systèmes informatiques des administrations, hôpitaux, banques, entreprises et partis politiques, celle en cours des objets, et même maintenant celle des usines.

LA GÉNÉRALISATION DES ATTAQUES

Les virus et autres logiciels malveillants existent depuis longtemps, et Internet a toujours été un excellent moyen de les disséminer. Mais le changement d'échelle du réseau a conduit à un changement d'échelle similaire dans ses attaques. Elles sont maintenant nombreuses et variées : attaques par intrusion directe dans les ordinateurs et téléphones, allant jusqu'à leur prise de contrôle complet ; attaques dites de *déni de service* consistant à noyer des serveurs sous les messages ; attaques d'*hameçonnage* (*phishing*) incitant les utilisateurs à visiter des pages malveillantes pour usurper leur carte bancaire ou leur identité ; attaques par *virus de rançonnage*, en anglais *ransomware*, aboutissant à crypter les données des utilisateurs ou entreprises d'une façon seulement décryptable contre rançon ; attaques sur les systèmes de vote « électronique », c'est-à-dire géré par logiciel ; etc. La panoplie ne fait que croître, en créant d'ailleurs de nouveaux mots à chaque fois.

Il n'y a même plus besoin d'une grande puissance de calcul chez soi pour réaliser ces attaques. Il est souvent plus simple de recruter des *bots*¹⁶ sur le réseau à l'insu de leurs possesseurs : ce sont soit des ordinateurs mal protégés (comme le vôtre ?), soit des objets informatisés de toutes sortes comme des caméras ou des ampoules électriques auxquelles les fabricants n'ont pas jugé utile de fournir une réelle protection car ils ne supposaient pas qu'ils pourraient être ainsi détournés. Un bon exemple est une attaque massive subie en 2016 par le serveur de noms majeur Dyn d'Internet, l'une des plus violentes qu'ait connue le réseau. Elle a été réalisée en recrutant un très grand nombre de caméras de surveillance¹⁷. Bien sûr, tous ces objets peuvent eux-mêmes être attaqués. Nous en reparlerons au [chapitre 9](#).

LE PROTOCOLE HTTPS

Pour assurer la sécurité des communications, les acteurs du Web ont développé le protocole sécurisé HTTPS (*HTTP secure*), invoqué automatiquement lors de toute connexion vers un site dont l'adresse commence par « `https://` ». Les connexions sont alors cryptées, ce qui doit assurer en théorie une parfaite sécurité. Mais ce protocole est très complexe, en particulier parce que la cryptologie est un sujet difficile et évolutif. Pour établir la communication sécurisée, il commence par une discussion entre le client et le serveur pour déterminer quel chiffrement utiliser et avec quelle clef. Cette discussion, gérée par le protocole *SSL/TLS* (*secure socket layer/transport layer security*), peut elle-même engendrer des failles sournoises. De plus, avec le temps, certains chiffrements s'avèrent trop faibles. Quand des attaques ou des faiblesses sont détectées, il faut leur trouver des parades puis mettre à jour tous les sites connectés, ce qui est très difficile de par leur nombre. Du coup, certains sites utilisent encore des méthodes cryptographiques obsolètes, que les hackers¹⁸ peuvent casser pour prendre des pouvoirs quelquefois considérables sur les sites en question. Heureusement, ces hackers ne sont pas que des « méchants ». Ils peuvent être aussi des chercheurs en cryptologie qui cherchent à détecter les menaces avant qu'elles ne puissent être exploitées, ou à établir de nouveaux protocoles prouvés mathématiquement incassables, au moins pour certains types d'attaques.

Il est devenu clair que les questions de sécurité sont celles qui vont freiner l'expansion de l'informatique en général et de l'Internet des objets en particulier. Après les avoir longtemps négligées, les individus et entreprises en prennent maintenant conscience. Mais cela ne suffit pas pour les traiter, car le sujet est intrinsèquement difficile et les gens bien formés très peu nombreux (sauf peut-être du côté des hackers malveillants). Je traiterai ce sujet plus longuement aux [chapitres 10](#) et [11](#).

CHAPITRE 6

Images, sons et cartes numériques

Comme je l’ai déjà expliqué au [chapitre 2](#), les domaines de la production, de la diffusion, et de l’analyse de l’image et du son font partie de ceux qui ont été le plus vite et le plus profondément révolutionnés par l’informatique. Ce sont aussi des domaines essentiels de notre vie, car la vue et l’ouïe sont nos sens principaux. La révolution informatique de l’image et du son ne touche pas qu’aux moyens de stockage et de diffusion de photos, musiques et films dont nous avons déjà parlé : les domaines d’action s’y sont aussi rapidement étendus.

Pour l’image, citons la photographie numérique, que nous étudierons en détail dans ce chapitre, l’imagerie satellitaire de la planète, utile de la météorologie à la cartographie en passant par l’agriculture, l’imagerie médicale multimodale (radiographie, scan, échographie, différentes modalités d’IRM) que nous étudierons au [chapitre 7](#), l’imagerie de l’univers discutée au [chapitre 8](#), la surveillance, la reconnaissance automatique de visages ou d’objets, la conduite automatique des voitures, etc. De son côté, la synthèse d’images réalistes, techniquement assez différente, sert au cinéma à la simulation des phénomènes naturels et à la réalité virtuelle ; voir par exemple le cours de Marie-Paule Cani sur le sujet au Collège de France ([Cani, 2015](#)). Pour comprendre la taille du champ de l’imagerie numérique, il suffit de jeter un œil à la conférence SIGGRAPH, la principale du domaine, qui rassemble tous les ans des milliers de personnes¹.

Le traitement informatique du son étant sensiblement plus difficile à expliquer, je n'en parlerai pas en détail dans ce livre. Mais il me paraît utile de le mentionner brièvement dans cette introduction à travers ses effets. L'analyse et la synthèse informatiques de sons sont devenues essentielles dans le champ musical : acoustique et sonorisation des salles avec spatialisation 3D, simulation en temps réel d'instruments de musique, développement d'instruments purement synthétiques, suivi de partitions en temps réel permettant une interaction homme/machine bien plus souple qu'auparavant pour la musique mixte hommes/ordinateurs, co-improvisation hommes/ordinateurs, etc. Toutes ces nouveautés ont un impact profond sur la composition, l'exécution et l'improvisation musicale des pièces contemporaines. Avec l'Ircam², créé par Pierre Boulez il y a presque cinquante ans, nous disposons d'un des meilleurs laboratoires mondiaux sur la liaison entre acoustique, informatique et musique ; y travaillent ensemble chercheurs, compositeurs, et réalisateurs en informatique musicale (RIM). Citons aussi le travail pionnier de Jean-Claude Risset sur tous les aspects de l'informatique sonore. Le lecteur pourra en savoir plus sur la relation entre informatique et musique en visionnant les cours de Philippe Manoury au Collège de France en 2017 sur la chaire annuelle de Création artistique ([Manoury, 2017](#)).

L'analyse d'autres signaux temporels de même type est également à l'œuvre en médecine avec l'analyse des électrocardiogrammes et des signaux émis par le cerveau, dans l'industrie avec le contrôle du bruit, en éthologie avec l'analyse des cris des animaux ou du sonar de la chauve-souris, etc.

Dans ce chapitre, je me concentrerai sur la photographie numérique, un sujet idéal pour illustrer la puissance de l'informatique, puis j'aborderai un peu la cartographie et le passage à la 3D. Je ne parlerai pas de la synthèse d'images, non pas parce que le sujet n'est pas intéressant, mais parce que ma compétence y est faible (voir plutôt [Cani, 2015](#)).

La photographie numérique

Tout le monde sait ce que c'est qu'une photographie, mais sans nécessairement réaliser que ce qu'elle transmet est de la pure information. Depuis son invention au XIX^e siècle jusqu'à la fin du XX^e siècle, la technique photographique a été essentiellement présentée sous l'angle de l'optique et de la chimie, ce qui a occulté son côté informationnel. Il revient maintenant en force.

Un appareil photo numérique moderne, qu'il soit un objet en soi ou intégré dans un téléphone, cas le plus fréquent aujourd'hui, est un étonnant concentré d'algorithmes de toutes sortes. Tous les progrès ne sont pas arrivés en même temps, et on peut identifier quatre étapes fondamentales. La première, à la fin du XX^e siècle, a été la fabrication algorithmique d'images par logiciel à partir de la sortie des nouveaux capteurs électroniques de lumière fournis par les physiciens. La seconde, plus récente, a été la correction automatique de la lumière et des défauts optiques, qui est maintenant systématiquement incorporée dans les appareils et les logiciels de postproduction. La troisième, encore plus récente, a été la mise en place de systèmes d'aide à la prise de vue, impensables auparavant : il ne suffit pas que l'image soit bonne, il faut aussi avoir bien pu la prendre. La dernière, qui commence, est une modification profonde de l'industrie associée, où l'on voit que les algorithmes deviennent plus importants que la matière.

Tous les algorithmes dont nous allons parler servent d'une part à faire ce qu'on savait faire avant, c'est-à-dire une belle image, et d'autre part à faire des choses difficiles ou impossibles à faire par des procédés physiques : augmentation massive de

la sensibilité, cadrages par des viseurs devenant eux-mêmes numériques pour fournir beaucoup plus d'indications utiles, mise au point et exposition ultraprécises et rapides, stabilisation des objectifs, correction algorithmique de leurs défauts optiques, améliorations de l'image par fusion algorithmique de prises de vue multiples, communication immédiate des images par Internet, etc. Nous retrouverons des questions et algorithmes similaires au [chapitre 7](#), lorsque nous parlerons des immenses progrès de l'imagerie médicale.

Ma connaissance de ce domaine vient principalement de ma passion pour la photographie et d'une collaboration avec la société française DxO Labs, un des leaders du domaine, et en particulier avec son directeur scientifique Frédéric Guichard³. Mais je ne dis pas cela pour faire de la publicité, car les algorithmes utilisés par les systèmes d'autres sociétés ou du domaine public (encore rares) sont similaires – et en compétition permanente. Pour le lecteur qui souhaite approfondir le sujet, je conseille la lecture du livre d'Henri Maître *Du photon au pixel* (Maître, 2015).

Avertissement. La discussion ci-dessous se concentrera sur les aspects techniques et ignorera les aspects vraiment fins de la photographie, comme la restitution des couleurs, ainsi que l'ensemble des questions artistiques. Il est clair qu'avoir un bon appareil et de bons algorithmes ne suffit en aucun cas pour être un bon photographe ! Mais ça reste indispensable pour tous, artistes compris.

DE L'ARGENTIQUE AU NUMÉRIQUE

La photographie argentique, née au milieu du XIX^e siècle, s'est lentement perfectionnée durant le XX^e siècle pour atteindre une grande qualité technique et artistique. Mais elle s'est fait largement supplanter par la photographie numérique, qui ne date pourtant en pratique que du début des années 2000, avec l'introduction des premiers scanners et appareils photo numériques. Ces premiers appareils numériques ressemblaient trait pour trait aux appareils argentiques de même catégorie, qu'ils soient reflex (visée à travers l'objectif) ou compacts. La grande différence était le remplacement du film par un capteur électronique CCD (*charge coupled device*), invention de William Boyle et George Smith en 1969 qui leur a donné le prix Nobel

de physique en 2009. La technologie a ensuite évolué vers les capteurs CMOS, plus sensibles, mais ce point ne sera pas important ici. La qualité de la photo numérique n'a atteint celle de l'argentique qu'un peu avant 2010 (selon moi), puis l'écart s'est continûment accentué en faveur du numérique – ce qui ne veut pas dire que la photo argentique a perdu de son intérêt, surtout pour les artistes. Un changement récent et profond est que l'acte de photographier est devenu totalement banal maintenant que l'appareil photo le plus répandu s'appelle le téléphone et que l'on peut envoyer ses photos à qui on veut par Internet dès qu'elles sont prises : le réseau social Facebook en reçoit maintenant plus d'un milliard par jour.

Comme dans bien d'autres domaines, le public reste attaché à la matière de l'appareil sans vraiment apprécier l'importance essentielle de ses algorithmes. Ainsi, quand on lit les revues de photographie, on y trouve surtout des dissertations sur les progrès des capteurs des appareils numériques. Pourtant, si les capteurs ont effectivement fait de beaux progrès, ce sont bel et bien les progrès des algorithmes qui ont été les plus déterminants. Disons que sur quatre ordres de grandeur de progrès en sensibilité dans les vingt dernières années, un est dû aux capteurs et trois aux algorithmes.

Mon propos est donc d'entrer dans le vif du sujet, en expliquant pourquoi la photographie numérique fournit tant de qualité et de souplesse en mariant de la physique et de l'informatique, toutes deux de haut vol. Sur le chemin, nous verrons que les apports principaux sont dus non pas à des perfectionnements des modes de pensée précédents, mais à des changements radicaux de ceux-ci conduisant à des solutions profondément nouvelles, qu'il aurait été impossible d'inventer et de mettre en œuvre avec les méthodes traditionnelles. Mais, auparavant, il me faut expliquer sommairement sur quoi repose la photographie, dont les invariants n'ont pas vraiment changé car ils sont directement liés à notre vision.

QUE VOIENT NOS YEUX, OU PLUTÔT QUE VOIT NOTRE CERVEAU ?

Dans la suite, je m'appuierai sur un leitmotiv bien connu des photographes : le but d'une photo n'est pas de reproduire fidèlement la réalité, comme on essaie de le

faire pour le son haute-fidélité ; il est de donner au cerveau une *impression de réalité*, ou encore une *impression précise qu'on souhaite lui donner* dans un but artistique, même si cette impression est obligatoirement différente de la réalité. Ceci demande des artifices techniques liés à l'organisation de notre système visuel, car nous ne regardons pas du tout une photo comme nous regardons la nature. Un bon exemple est le flou artistique⁴ : pour qu'il détache bien l'expression du visage, un bon portrait exige une mise au point sur l'œil le plus proche et un flou d'arrière-plan. Or, quand nous regardons un visage, nous ne percevons pas du tout un tel flou ! Son but est simplement de faire que le spectateur se concentre sur le visage et en particulier sur ses yeux, en les démarquant du reste par cet artifice. De même, le noir et blanc est très expressif, alors qu'il n'existe quasiment pas dans la nature. De façon générale, il est souvent indispensable de « truquer » plus ou moins la photo pour la rendre belle. Pour le comprendre, il me faut commencer par une brève description de comment nous voyons le monde⁵. Le lecteur intéressé pourra en apprendre beaucoup plus dans le cours de José Sahel au Collège de France ([Sahel, 2016](#)).

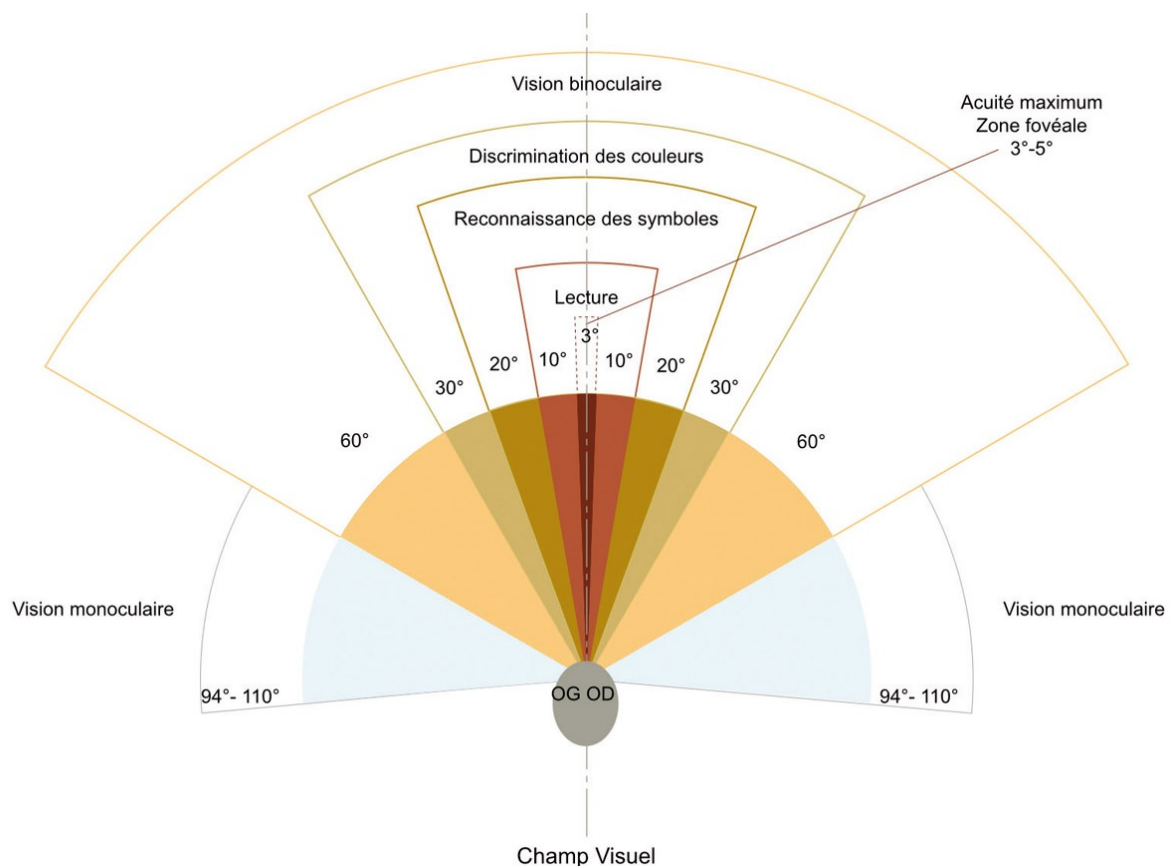


Figure 19. Le champ visuel. (By Rheto [Own work] CC BY-SA 3.0 via Wikimedia Commons.)

En simplifiant beaucoup, l'œil est une sphère dont le fond est tapissé par la rétine et qui possède à l'avant un cristallin, lentille de focale adaptable (tant qu'on est jeune), et un iris, ouverture de diamètre variable qui permet d'ajuster la lumière qui parvient à la rétine. Cette rétine est tapissée de deux sortes de cellules photosensibles : les bâtonnets, spécialistes de la vision de nuit car sensibles seulement à la quantité de lumière tant qu'elle est faible, qui ne nous intéresserons pas ici, et les cônes, qui existent en trois versions sensibles respectivement au rouge, au vert et au bleu. Les cônes sont principalement concentrés dans la *fovéa*, tache circulaire de 3 millimètres de diamètre au centre de la rétine. Comme illustré par la figure 19, nous ne voyons donc en couleurs et avec une haute résolution que dans un cône très mince de 5 degrés d'angle au maximum, et nettement moins bien autour. Cela implique que nous ne voyons *jamais* l'image globale qui forme pourtant notre perception. Elle est en fait reconstituée par notre cerveau en assemblant des mini-images que notre fovéa capte séquentiellement, l'œil se déplaçant constamment par des saccades conduisant de point d'intérêt en point d'intérêt, et l'iris changeant constamment son ouverture suivant la luminosité du point d'intérêt courant. Les saccades sont elles-mêmes déterminées par des algorithmes neuronaux sophistiqués qui analysent rapidement l'image périphérique à basse résolution pour détecter les nouveaux points d'intérêt potentiels. Au total, la perception harmonieuse et stable que nous ressentons ne résulte que d'un réarrangement de mini-images par la hiérarchie des sous-systèmes neuronaux du cortex visuel. Notre perception est donc franchement plus virtuelle que réelle, ce qui fait d'ailleurs le délice des magiciens qui sont experts pour la tromper.

Autrefois, on expliquait que la rétine avait pour fonction de transmettre l'image au cerveau. On sait maintenant que cette explication est carrément fausse. La rétine est en fait un processeur de traitement du signal très sophistiqué, qui ne transmet pas l'image mais des caractéristiques précises et parcimonieuses extraites de celle-ci : contours, contrastes, mouvements, couleurs, etc. Cette transmission se fait sous la forme d'une vingtaine de films parallèles traités ensuite par une succession d'aires cervicales qu'il n'est pas utile de détailler ici (voir [Sahel, 2016](#)). De plus, rien d'immobile n'est transmis par la rétine : pour voir ce qui ne bouge pas, nous

effectuons sans arrêt des microsaccades oculaires invisibles et différentes des macrosaccades décrites précédemment. Enfin, nous avons une étonnante capacité à bien reconnaître les couleurs dans des conditions de lumière très différentes, par exemple en passant de la lumière naturelle à celle des lampes à filament – à condition toutefois que nous ayons au moins dans notre champ de vision quelques objets dont nous connaissons déjà la couleur. Nous réalisons alors automatiquement un équilibrage global des couleurs de la scène, ce que ne font naturellement ni un film ni un capteur. Si notre système visuel contient des algorithmes biologiques pour ce faire, il nous a fallu en inventer pour la photographie.

Comment un appareil photo capte la lumière

Comme pour l'œil, la lumière entre dans l'appareil par un objectif pour être captée par un dispositif physique : soit la pellicule chimique pour l'argentique, soit le capteur électronique pour le numérique. Mais, cette fois, il n'y a pas de balayage par saccades et l'intégralité de l'image parvient en un coup au capteur.

Pour assurer la qualité de la photo, il faut que la quantité de lumière entrante reste dans la zone de bon fonctionnement du capteur. Cela se fait en réglant deux paramètres : l'*ouverture* et le *temps de pose*. L'ouverture se règle à l'aide du *diaphragme*, cercle de diamètre variable inclus dans l'objectif qui est l'analogue de l'iris. Plus il est ouvert, plus la lumière entre. Le temps de pose est simple à comprendre pour un obturateur mécanique classique : c'est celui pendant lequel l'obturateur est ouvert. Cela vaut aussi en cinéma et vidéo où l'on ne fait que prendre une série de photos indépendantes.

Notons cependant que, pour un appareil numérique sans miroir reflex, la lumière est constamment captée pendant la visée pour l'afficher soit sur l'écran, soit dans le viseur. Pour prendre une photo avec un obturateur classique, il faut donc fermer l'obturateur, vider le capteur, rouvrir l'obturateur et le refermer à la fin du temps de pose, ce qui est plus compliqué que pour une photo argentique. On peut utiliser aussi des « obturateurs électroniques » sans pièces mécaniques, en lisant ligne à ligne les données du capteur (*rolling shutter* en anglais) ; c'est systématiquement le cas pour les téléphones, mais c'est aussi le cas pour certains appareils haut de gamme qui

disposent des deux types d'obturateurs. Un grand avantage est que la prise de vue devient totalement silencieuse, ce qui précieux dans un concert ; un inconvénient est que tout ce qui bouge trop vite peut être déformé.

RESPECTER LA DYNAMIQUE DE LA LUMIÈRE

Le fait que l'appareil photo capte toute l'image d'un coup implique que l'ouverture du diaphragme reste constante pendant le temps de pose. Elle sera donc identique pour toutes les zones de l'image, contrairement à ce qui se passe avec notre iris qui s'ouvre ou se ferme quand nous balayons la scène. La pellicule ou le capteur doivent donc gérer une grande *dynamique* de lumière pour bien enregistrer à la fois une zone ensoleillée dans une partie de l'image et une ombre dans une autre partie. Ce n'est pas la moindre des difficultés, surtout en numérique.

La figure 20 schématise les courbes de réponse typiques d'un film argentique (à gauche) et d'un capteur silicium (à droite). La courbe argentique montre un démarrage doux dans les basses lumières et une saturation douce dans les hautes lumières. Le film est donc capable d'une dynamique élevée entre les basses et hautes lumières sans décrochage brutal, ce qui est une de ses grandes qualités. Au contraire, le capteur silicium démarre par une marche, engendrée par le bruit électronique inévitable car dû à la température ambiante, puis suit une droite quasi parfaite jusqu'à un point de saturation brutale. En d'autres termes il est très précis et linéaire au-dessus d'une lumière minimale mais sature brutalement s'il y a trop de lumière. En pratique, cela se traduit sur l'image par des à-plats blancs très désagréables. Il est donc essentiel de bien exposer les photos numériques et de ne surtout pas les surexposer sous peine de perte irrécupérable d'information. Les appareils numériques surexposaient fréquemment jusque vers 2010, date à laquelle ils ont eu la puissance de calcul suffisante pour exécuter des algorithmes de mesure de l'exposition bien réglés. Ceux-ci sont fort complexes car ils analysent finement la structure de l'image d'une façon que je ne détaillerai pas ici.

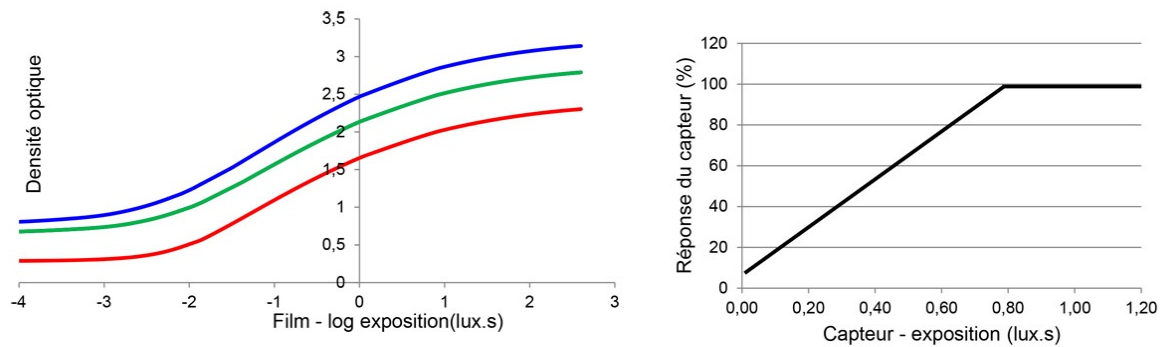


Figure 20. Courbes de réponse des films et capteurs (source DxO.com).

LA SENSIBILITÉ ET LE BRUIT

La *sensibilité* de la pellicule ou du capteur est un paramètre essentiel en photographie. Elle détermine l'intensité de la réaction de la pellicule ou du capteur à une lumière donnée, ou, de façon duale, la quantité de lumière à faire entrer pour avoir une image bien placée par rapport à la courbe de réponse. La sensibilité se mesure de façon linéaire en ISO⁶ (autrefois ASA⁷) : 100 ISO est 2 fois plus sensible que 50 ISO, alors que 400 ISO est 4 fois plus sensible que 100 ISO. Mais la sensibilité détermine aussi la qualité de l'image en termes de *bruit*, irrégularité en forme de granulation ou moutonnement qui gomme les détails et brouille les à-plats : le bruit augmente fortement avec la sensibilité. Nous allons voir que l'argentique et le numérique sont devenus très différents en termes de sensibilité et de bruit.

En argentique, le vénérable Kodachrome, chéri des amateurs de diapositives, est né en 1935 avec 10 ISO, a atteint 25 ISO dans les années 1960, et a terminé sa carrière en 2009 avec 64 ISO. Les films négatifs noir et blanc les plus communs sont de 100, 200 ou 400 ISO ; il est possible de les pousser de 400 à 800 ou même 1 600 ISO au développement, avec une forte perte de qualité toutefois. Pour les négatifs couleur, qui sont constitués de plusieurs couches photosensibles séparées par des filtres colorés, 100 et 200 ISO sont communs mais aller plus haut devient plus difficile car la qualité des couleurs se dégrade vite. La différence de sensibilité et la dégradation de qualité sont essentiellement liées à la taille des grains d'argent photosensibles : plus ils sont gros, plus la pellicule est sensible, mais plus l'image montre son grain et devient bruitée. Une pellicule de 100 ISO est bien adaptée à la

lumière du jour, avec un grain très fin et peu de bruit, alors qu'une pellicule de 400 ISO ou plus, nécessaire en basse lumière, montre un grain et un bruit bien visibles. Il faut cependant noter que les fabricants ont beaucoup travaillé sur l'agencement physique des grains et qu'un grain perçu gros mais bien géré peut avoir une belle force artistique, souvent exploitée par les portraitistes : voici un bon exemple de différence entre le regard et la photo, le regard n'ayant pas de grain ! Enfin, l'argentique a un autre inconvénient : la pellicule impose une sensibilité constante pour toutes ses photos, ce qui pose des problèmes quand la lumière change beaucoup ou quand on passe de l'extérieur à l'intérieur. Le flash devient vite nécessaire, mais il est difficile à bien utiliser car il a une fâcheuse tendance à transformer la peau des personnes en fromage blanc peu expressif et les yeux en taches rouges disgracieuses par réflexion de la lumière sur la rétine à travers l'iris (qu'il est heureusement facile de faire disparaître algorithmiquement).

Enfin, pour l'argentique, un point étonnant est qu'on ne comprend précisément comment marche un révélateur photographique que depuis une dizaine d'années, et que cette compréhension a permis de multiplier la sensibilité des films par 10 à qualité égale⁸. Mais c'est commercialement trop tard à cause du virage vers le numérique, sauf pour les hologrammes qui deviennent vraiment extraordinaires de finesse !

En photographie numérique, les principes sont similaires, mais la réalisation complètement différente : le capteur est formé de *photosites* stockant des charges électriques quand ils sont exposés à la lumière. Ces charges sont ensuite numérisées par un convertisseur analogique-numérique et transmises en série pour être stockées en mémoire. Alors qu'il n'existe que peu de tailles de pellicule, le capteur numérique peut être de taille très variable, allant du très grand pour le moyen format au minuscule pour le téléphone, en passant par le 24×36 des reflex haut de gamme (en souvenir de l'argentique mais aussi pour assurer la compatibilité des anciens objectifs), l'APS-C de surface moitié du 24×36 , le micro 4:3 de surface encore moitié, et le 1 pouce de diagonale ou moins des bons appareils compacts, encore plus petit. Chaque taille a ses avantages et ses inconvénients : les grands capteurs offrent intrinsèquement une meilleure qualité car ils captent plus de lumière, mais les petits permettent une bien meilleure compacité des boîtiers et objectifs et un prix moindre – à chacun son choix selon ses besoins (ou son snobisme).

Un avantage majeur de la photo numérique est qu'on peut régler la sensibilité image par image au lieu de la garder identique pour tout le film : cela se fait en appliquant de façon adaptée à chaque image une amplification analogique au courant électrique envoyé par les photosites au convertisseur analogique-numérique, ou encore une amplification numérique après la conversion. Un deuxième avantage, encore plus déterminant en pratique, est qu'on peut monter beaucoup plus en sensibilité qu'en argentique : un appareil moderne reste excellent à 1 600 ISO, voire 3 200 ISO, et les meilleurs appareils 24 × 36 ou plus permettent de monter à 25 000 ISO ou même davantage. Cela permet en particulier d'oublier les flashes en les réservant aux rôles pour lesquels ils sont indispensables comme l'éclairage indirect en studio et le débouchage des ombres au grand soleil.

L'analogie du grain argentique est le bruit numérique, bien moins artistique que le grain si on ne le traite pas de la bonne façon. Heureusement, un des grands progrès récents a été l'amélioration des algorithmes de débruitage numérique, très pauvres au début. Nous en reparlerons en détail plus loin.

LA PROFONDEUR DE CHAMP

Un autre paramètre physique essentiel en photographie est la *profondeur de champ*, qui décrit entre quelle distance proche et quelle distance lointaine les éléments de la photo apparaîtront nets. Pour notre perception, si nous avons une bonne vue ou sommes bien corrigés (et pas encore atteints par la presbytie), la profondeur de champ n'est pas une notion pertinente : lors des saccades, nous mettons au point sans arrêt en déformant notre cristallin selon ce que vise notre cône visuel. Donc notre cerveau n'intègre que des sous-images nettes, et notre perception est que tout est net. Au contraire, en photographie, la profondeur de champ est une notion globale : elle est la même pour toute l'image et ne dépend que de la distance de mise au point, de la focale de l'objectif et de son ouverture. Elle est d'autant plus faible que la mise au point est proche, que la focale est longue et que le diaphragme est ouvert. Elle est donc naturellement faible dans les appareils à grands capteurs, puisqu'ils demandent des objectifs de focale plus longue que pour ceux à petits capteurs pour prendre la même scène. À l'inverse, elle devient très grande pour les téléphones, dont le capteur

est minuscule. Une très grande profondeur de champ est sans problème pour la photo basique vue sur un petit écran ou pour la photo de paysages, mais elle est souvent mal adaptée à la photo artistique.

En pratique, le réglage fin de la profondeur de champ est délicat et mobilise beaucoup l'attention du photographe. Une profondeur de champ trop faible donne l'impression d'une photo floue, une trop grande est souvent peu attirante car elle uniformise tout et ne met pas en valeur les points que l'on souhaite souligner. Le flou artistique déjà mentionné repose précisément sur une gestion fine de la profondeur de champ. Par exemple, en portrait, la bonne mise au point est généralement sur l'œil le plus proche, car c'est celui que nous allons regarder en premier par focalisation inconsciente de notre système de vision ; savoir quoi d'autre dans le visage ou dans le fond doit être flouté pour accentuer l'expression appartient à l'art du portraitiste. En macrophoto, très délicate, la méthode est similaire : on met au point sur les yeux des insectes ou les étamines des fleurs, et on choisit la profondeur de champ en fonction de la taille de ce qu'on souhaite net.

Mettons-nous d'abord dans le cas où nous souhaitons une grande profondeur de champ, par exemple pour un paysage avec des composants proches et un arrière-plan que nous voulons tous nets. En argentique, quand on a chargé un film de sensibilité donnée, le seul paramètre sur lequel on peut jouer est le diaphragme : il faut le fermer pour augmenter la profondeur de champ, mais cela diminue la vitesse en proportion et augmente le risque de flou de bougé. En numérique on peut obtenir une profondeur de champ plus grande à vitesse constante en fermant le diaphragme et en augmentant la sensibilité, réglage possible de façon indépendante à chaque image.

Tout cela est utile, mais nous verrons plus loin que d'autres procédés rendent la vie encore plus facile : la stabilisation de l'objectif ou du capteur permet de photographier à des vitesses bien plus lentes sans risque de flou de bougé, ce qui permet de fermer le diaphragme et donc d'augmenter la profondeur de champ ; ensuite, des techniques algorithmiques de fusion d'images permettent d'obtenir une très grande profondeur de champ en fusionnant algorithmiquement plusieurs prises de vue faites avec des mises au point graduelles en distance. Nous verrons plus loin que d'autres méthodes fondées sur l'utilisation d'aberrations chromatiques bien contrôlées ont été utilisées sur les téléphones avant qu'ils n'aient de systèmes de mise au point autofocus. Enfin, le système Lytro⁹ permet de faire la mise au point *après* avoir pris la

photo, grâce à une modification du capteur associée à un traitement algorithmique très intéressant ; mais cette technique reste encore confidentielle.

À l'inverse, quand on souhaite diminuer la profondeur de champ, il faut ouvrir le diaphragme. S'il est déjà ouvert à fond, ce n'est pas possible. Il est donc difficile d'obtenir une faible profondeur de champ avec un téléphone, dont la focale est très courte et le diaphragme souvent ouvert à fond. Mais, en numérique, pour mettre en valeur un visage par exemple, on peut flouter automatiquement l'arrière-plan par logiciel¹⁰. Cette fonctionnalité devient disponible sur certains téléphones, avec des résultats pas encore totalement convaincants, mais ces choses peuvent progresser vite.

De l'image crue à une image visible

Entrons maintenant dans l'algorithmique de la photo numérique par sa partie la plus indispensable : la traduction de l'image *crue* (*raw*) délivrée par le capteur en une image finale décrite par exemple dans le format compressé standard JPEG (*joint photography expert group*), ou, si l'on veut éviter la perte d'information due à la compression, le format TIFF (*tagged image file format*) qui permet une compression sans perte.

L'IMAGE CRUE (OU RAW)

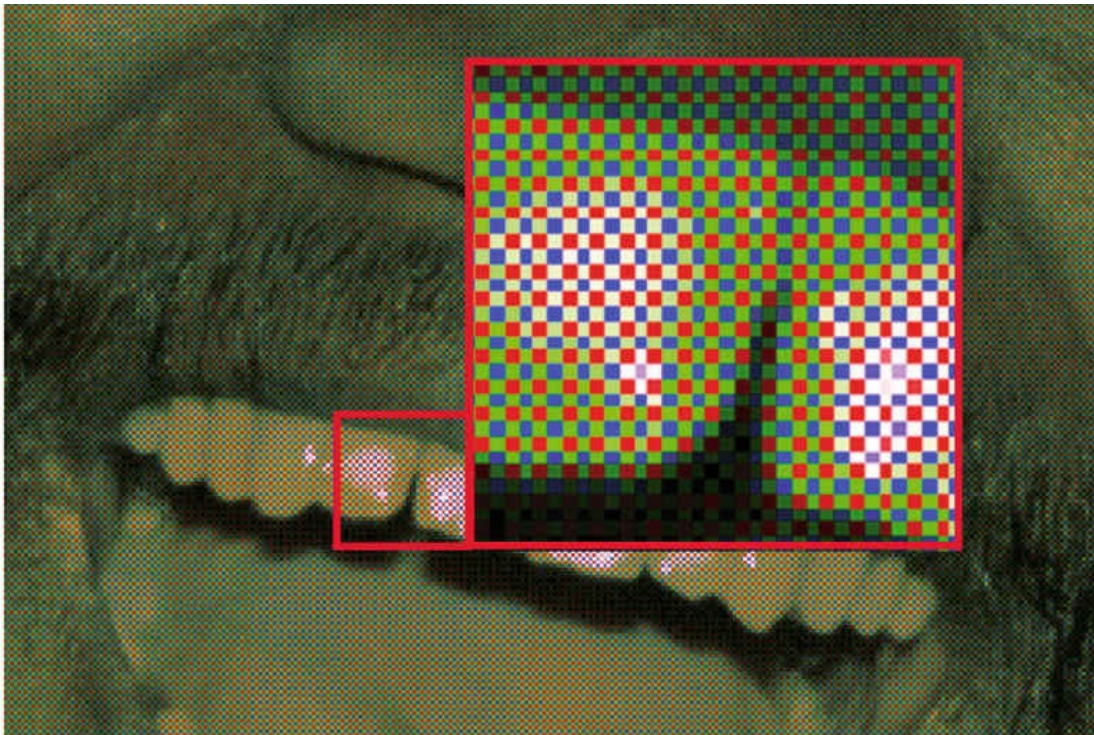


Figure 21. Fragment et zoom d'une image crue ou *raw* (source G. Berry/DxO.com).

Comme illustré par la figure 21, qui présente un petit morceau d'une image crue que nous allons voir se développer numériquement plus loin, la majorité des capteurs silicium sont constitués d'un réseau régulier de *photosites* électroniques groupés par quatre dans une géométrie dite *matrice de Bayer*. Il s'agit d'un assemblage de petits carrés comprenant deux sites verts pour un rouge et un bleu¹¹, ce choix étant dû au fait que notre œil est nettement plus sensible au vert. Chaque photosite a la forme d'un tunnel au fond duquel se trouve un composant électronique sensible à la lumière, qu'on peut appeler un compteur de photons. Le nombre de photosites du capteur est appelé *résolution*. De 4 ou 6 millions dans les premiers appareils, il est passé à environ 20 millions pour les appareils de 2016, le double ou le quadruple pour les appareils professionnels, et une bonne dizaine de millions pour les smartphones. Mais la résolution n'est pas tout, l'important étant l'équilibre entre le nombre de photosites et leur qualité, en bref la *quantité d'information utilisable* fournie par le capteur. Aucun post-traitement ne pourra l'augmenter : cette quantité d'information détermine définitivement la qualité des données qui seront fournies aux algorithmes, ainsi que la taille à laquelle on pourra agrandir l'image avec une bonne qualité visuelle. Par

rapport aux chiffres actuels, augmenter la résolution n'est plus vraiment indispensable, au moins pour les amateurs. Mais augmenter le nombre de bits utiles que peuvent fournir les photosites en diminuant leur bruit électronique et en captant plus de lumière reste important. Ce qui compte réellement est l'équilibre entre ces deux aspects.

Après la prise de vue, les charges accumulées par les photosites sont amplifiées et numérisées selon la sensibilité demandée, fournissant typiquement de 10 à 14 bits utiles par photosite, et les nombres obtenus sont séquentiellement transférés vers une mémoire de travail. Ces nombres sont associés à des *métadonnées* comme la date et l'heure, le type d'appareil et d'objectif, la sensibilité, la focale, l'ouverture, les coordonnées GPS et bien d'autres choses encore indispensables pour les traitements ultérieurs ou pour faciliter le classement et la recherche des photos. Tout cela forme ce qu'on appelle l'image crue (*raw* en anglais).

LE DÉMATRIÇAGE

Comme on le voit dans la figure 21, cette image crue est encore bien loin de l'image finale. Il faudra lui donner la forme la forme d'un tableau de *pixels* (pour *picture cells* en anglais), où chaque pixel est formé de trois nombres correspondant aux trois couleurs rouge, vert et bleu¹². La première étape purement algorithmique est de calculer ces pixels multicolores de la meilleure façon possible à partir des données brutes monocolores des photosites. On la nomme *dématriçage*, ou encore du joli mot franglais de *dérawtisation*. C'est une étape très technique et très difficile, car elle perd nécessairement de l'information (comme tout calcul) et surtout car elle peut introduire des *artefacts* si elle est mal faite, c'est-à-dire des défauts systématiques de l'image.

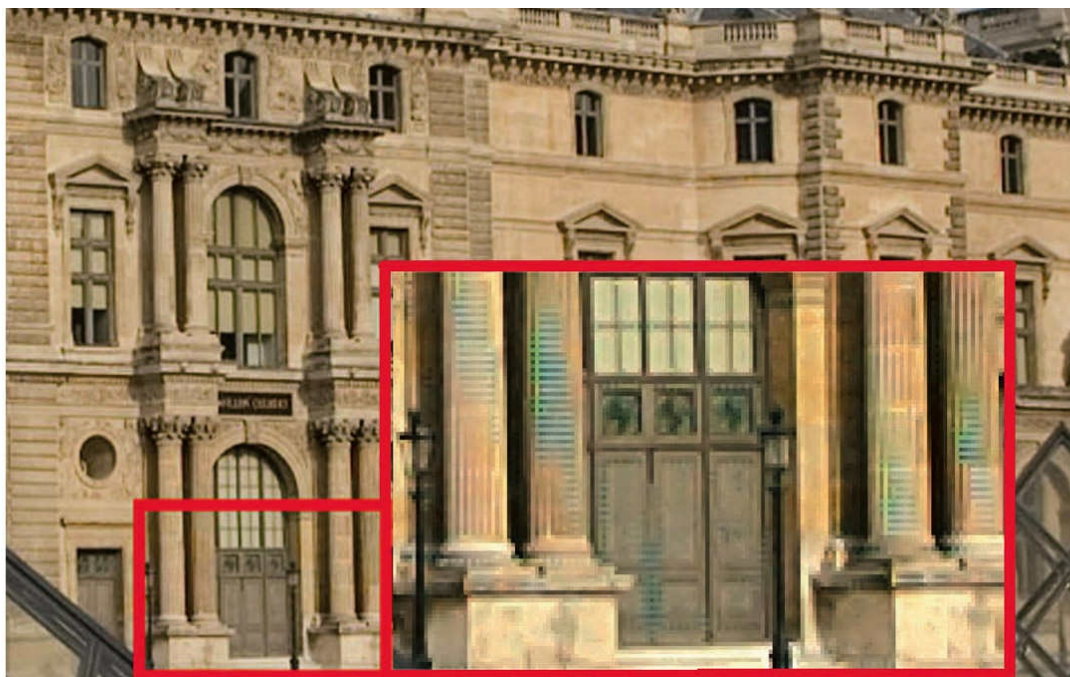


Figure 22. Effets de moiré (source DxO.com).

Les premiers dématriceurs avaient une forte tendance à introduire de désagréables effets de moiré sur les textures, similaires à ceux qu'on peut apprécier sur les vêtements ou rideaux de soie – mais tout n'est pas de la soie ! Un bon exemple est donné à la figure 22. Pour éviter le moiré, on plaçait devant le capteur des filtres physiques spéciaux dits d'*anti-aliasing*, qui diminuaient de fait la résolution et la netteté des images. Ces filtres ne sont plus indispensables grâce aux progrès algorithmiques du dématricage.

Le dématricage était au début un algorithme simple et local demandant peu de calculs, mais ce n'est plus du tout le cas. Les algorithmes modernes sont très sophistiqués car ils ne font pas que travailler de petite zone en petite zone de l'image : ils étudient la structure géométrique des éléments de l'image, cherchent ses lignes de force, examinent les relations entre ses différentes parties, etc. Ils intègrent aussi des éléments de la réduction du bruit dont nous parlerons plus tard.

Un dématricage est toujours effectué dans l'appareil, ne serait-ce que pour montrer le résultat sur son écran. Mais on peut obtenir de bien meilleurs résultats sur son ordinateur avec des logiciels spécifiques au prix de temps de calcul plus longs. Cela demande d'avoir gardé l'image crue : il faut toujours se souvenir du fait qu'il est

très difficile de retraiter une image comprimée en JPEG comme celle que rend par défaut l'appareil, surtout en ce qui concerne la colorimétrie pour laquelle beaucoup d'information est perdue lors de la compression. Quand on travaille en postproduction sur un ordinateur, tous les traitements qui vont suivre peuvent alors être réglés finement et de manière interactive, à condition d'avoir un écran bien calibré, c'est-à-dire qui montre les bonnes couleurs. Un tel écran se règle avec une sonde et un logiciel. La calibration est aussi indispensable pour que les tirages argentiques et impressions numériques soient fidèles. (Je n'expliquerai pas ici le fonctionnement des imprimantes jet d'encre, dont la qualité est devenue extraordinaire ; il faudrait un autre chapitre pour cela.)

RÉGLAGE DE LA LUMIÈRE

L'image dématricée n'est toujours pas regardable, et je ne la montre donc pas ici. La *mise en lumière et contraste* de la figure 23 amène les zones les plus claires près du blanc et les zones les plus sombres près du noir, en augmentant les contrastes généralement trop faibles sur l'image initiale. En effet, notre œil est particulièrement sensible au contraste, et, sauf intention explicite, une image peu contrastée est jugée molle. Attention toutefois : beaucoup d'appareils compacts et de téléphones fournissent encore des images trop contrastées, flatteuses sur petit écran mais devenant désagréables sur un grand écran ou à l'impression.



Figure 23. Lumière et contraste (source G. Berry/DxO.com).

LA BALANCE DES BLANCS

La *balance des blancs*, illustrée par la figure 24, intervient ensuite pour reconstruire sur la photo finale les couleurs que nous souhaitons voir et non celles qui ont été enregistrées, qui sont dépendantes de la qualité de la lumière : jaune en éclairage artificiel par lampes au tungstène, neutre au soleil, plus bleue par temps nuageux ou à l'ombre, etc. Rappelons que notre système visuel recale les couleurs en permanence par référence à celles qu'il sait déjà connaître. Mais, lorsqu'il regarde une photo sur écran ou sur papier, l'œil et le cerveau ne sont plus dans les conditions de la scène et ne font plus spontanément cet ajustement puisque les couleurs qu'ils perçoivent sont calées par l'environnement immédiat et pas par la photo regardée. Même si elles sont physiquement justes par rapport à la lumière réelle de la prise de vue, les couleurs vues sur la photo peuvent ainsi être perçues comme fausses et même désagréables. Le point essentiel est que les gris apparaissent bien gris, car cela suffit en pratique pour placer correctement les autres couleurs¹³.



Figure 24. Balance des blancs (source G. Berry/DxO.com).

En argentique, un problème est que la balance des blancs est constante pour tout un film. De plus, il n'existe pour un amateur que deux sortes de films, lumière naturelle ou lumière artificielle, cette dernière étant de plus adaptée aux lampes au tungstène qui vont disparaître. Fort heureusement, il est possible de faire quelques corrections par filtres de couleurs au moment du tirage papier, plus ou moins automatiquement sur une machine ou manuellement chez un professionnel, ou encore de passer par une numérisation de l'image à l'aide d'un scanner de photos – mais on repasse alors dans le monde de la photo numérique.

En numérique, la situation est idéale, car la balance des blancs peut être réglée différemment pour chaque image. Ce réglage peut se faire soit automatiquement dans l'appareil, à l'aide d'algorithmes subtils mais qui ne sont pas encore parfaits, soit manuellement en indiquant une « température de couleur¹⁴ » estimée (lumière du jour, nuageux, ombre, tungstène, etc., voire une température en degrés Kelvin), soit enfin semi-automatiquement en photographiant un gris de référence ou simplement une

feuille de papier blanc pas trop éclairée, l'appareil calant alors le gris pour déterminer le bon équilibre des couleurs. Tous les bons appareils rendent ces opérations faciles, et même les téléphones donnent de bons résultats. Enfin, en postproduction sur un ordinateur, si on a la chance d'avoir un gris dans l'image, il suffit de cliquer dessus pour que toutes les couleurs se replacent « magiquement ! ». Cette possibilité justifie à elle seule de conserver l'image *raw* dès que la lumière n'est pas simple, car la modification de la balance des couleurs est quasiment impossible une fois l'image comprimée en JPEG.

LE RENDU DES COULEURS

Mais les couleurs obtenues à l'étape précédentes sont généralement fades, car les caractéristiques des cônes de l'œil et des photosites du capteur sont assez différentes, en particulier pour les téléphones qui cherchent à capter le maximum de lumière pour leurs petits capteurs. Comme l'œil voit mal le fade, il faut rehausser les couleurs pour obtenir l'image de la figure 25 – mais pas trop sous peine d'obtenir une image d'apparence artificielle. C'est ce qu'on appelle régler la *saturation* des couleurs.



Figure 25. Rendu des couleurs (source G. Berry/DxO.com).

AJUSTAGE DE LA NETTETÉ

Ce n'est pas fini. Bien que très précise, l'image précédente nous apparaît floue, car elle manque des microcontrastes dont la rétine est friande. Pour obtenir l'image nette de la figure 26, on augmente savamment ces microcontrastes, en particulier aux contours et autres transitions fortes. Mais il ne faut surtout pas le faire trop sous peine de rendre l'image désagréable. Remarquons que pour augmenter la netteté, il faut en fait perdre de l'information !



Figure 26. Netteté (source G. Berry/DxO.com).

AMÉLIORATION DE LA LUMIÈRE

Une dernière manipulation, optionnelle, est de rendre l'image « plus lumineuse » au sens perceptif. Quand on prend une scène à fort contraste, il faut en général régler l'exposition pour que les blancs soient à la limite de la sensibilité du capteur : si elle est trop forte, les hautes lumières saturent et donnent des à-plats blancs désagréables. Mais, si la scène est très contrastée, les noirs peuvent se trouver *bouchés*, c'est-à-dire sans détails. On peut les déboucher algorithmiquement par une analyse fine de l'image qui segmente les zones à éclaircir et les traite en touchant peu aux zones claires, ce qui donne par exemple le résultat de la figure 27, où les détails du fond réapparaissent sans que les zones claires soient brûlées. Cet effet est réglable soit dans l'appareil, soit sur ordinateur. Comme elle éclaircit les zones sombres, ceux où le bruit est le plus sensible, cette correction sélective de la lumière risque d'y augmenter aussi le bruit. Le couplage avec le débruitage doit donc être soigné.



Figure 27. Après correction sélective de la lumière (source G. Berry/DxO.com).

Autres corrections algorithmiques

Après tout ce qui précède, l'image maintenant disponible est en gros celle que délivraient les premiers appareils numériques pour construire à peu près l'équivalent d'une image argentique. Mais c'est seulement maintenant que les vraies avancées algorithmiques de la photo numérique moderne vont intervenir, avec deux objectifs : corriger le bruit, et corriger les défauts de l'objectif.

DÉBRUITAGE

Les capteurs numériques sont par nature bien moins bruités que les films. Mais, comme les films, ils le deviennent de plus en plus dès qu'on travaille en haute sensibilité. Même s'ils se sont améliorés grâce aux progrès de leur conception et de leur fabrication, ce qui a permis de gagner de l'ordre d'un diaphragme, l'essentiel des améliorations a été fourni par le progrès des algorithmes de débruitage.



Figure 28. Un extrait d'une image très bruitée (source DxO.com).

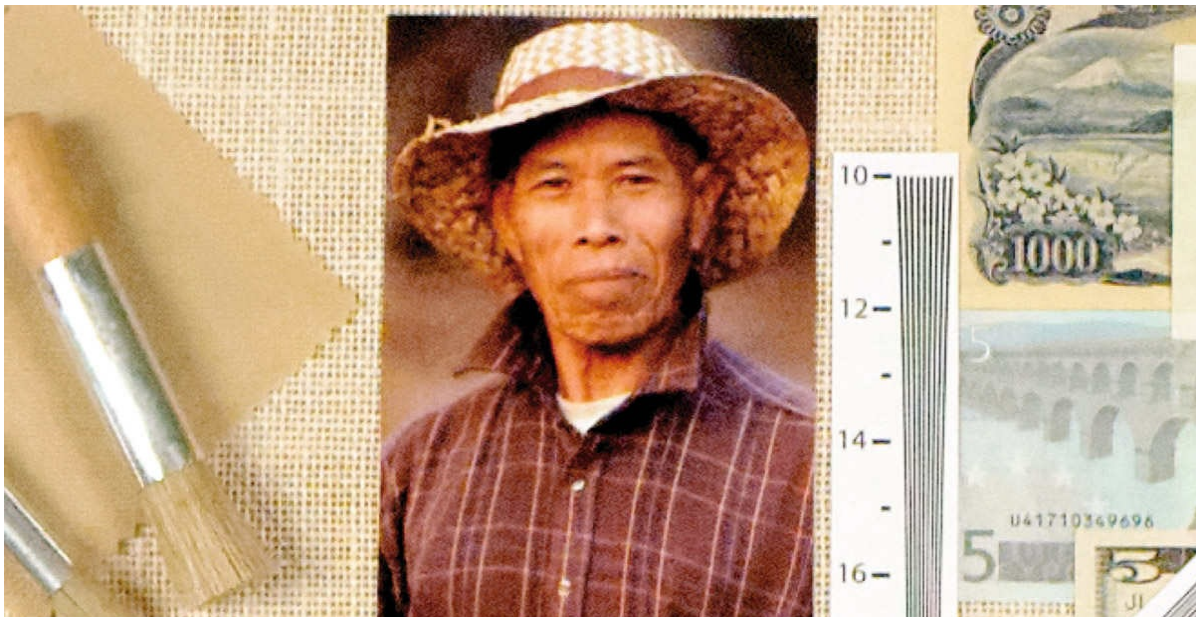


Figure 29. Image débruitée (source DxO.com).

La photo de la figure 28 montre une image volontairement très bruitée. La figure 29 montre un débruitage avec de bons algorithmes. On voit que le bruit est réduit et que les détails sont bien restaurés, ce qui n'était pas le cas avec des

algorithmes simplistes du début du siècle. Les couleurs sont aussi restaurées : en effet, elles se dégradent encore plus que les lumières à haute sensibilité.

De telles corrections contextuelles demandaient un énorme travail manuel avec les logiciels de retouche photo du xx^e siècle. Tous les appareils et logiciels actuels savent maintenant les réaliser, plus ou moins bien. L'idée de base est d'effectuer un certain lissage de l'image pour réduire le bruit par moyennage des pixels voisins, en profitant de la nature aléatoire de ce bruit : la moyenne de valeurs autour de 0 tirées aléatoirement tend vers 0 avec le nombre de ces valeurs. Mais les premiers algorithmes naïfs ont conduit à une forte perte des détails de l'image, eux aussi moyennés de façon aveugle. Les algorithmes modernes, très subtils, procèdent différemment en s'appuyant sur une analyse fine de la structure profonde de l'image et en agissant différemment en fonction de cette structure¹⁵ : à-plats ou au contraire zones texturées, lignes de force, etc. Le travail est bien fait dès l'appareil photo, mais pour les cas délicats il vaut bien mieux travailler sur ordinateur à partir de l'image *raw* car les meilleurs algorithmes sont bien trop gourmands en calcul pour être embarqués dans l'appareil.

La figure 30 montre qu'on peut aller bien plus loin en utilisant une technique spécifiquement numérique, la fusion d'images. Plutôt que de lisser un pixel avec ses voisins, on prend plusieurs photos de suite et on calcule la valeur finale de chaque pixel en lissant ses valeurs à travers les images successives. Même sur une petite impression comme dans ce livre, le gain est sensible, la différence devenant flagrante sur un agrandissement ; observez les détails du chapeau et la texture de la chemise. Nous verrons bien d'autres applications de la fusion d'images dans la suite de ce chapitre et dans le [chapitre 7](#) consacré à l'imagerie médicale.

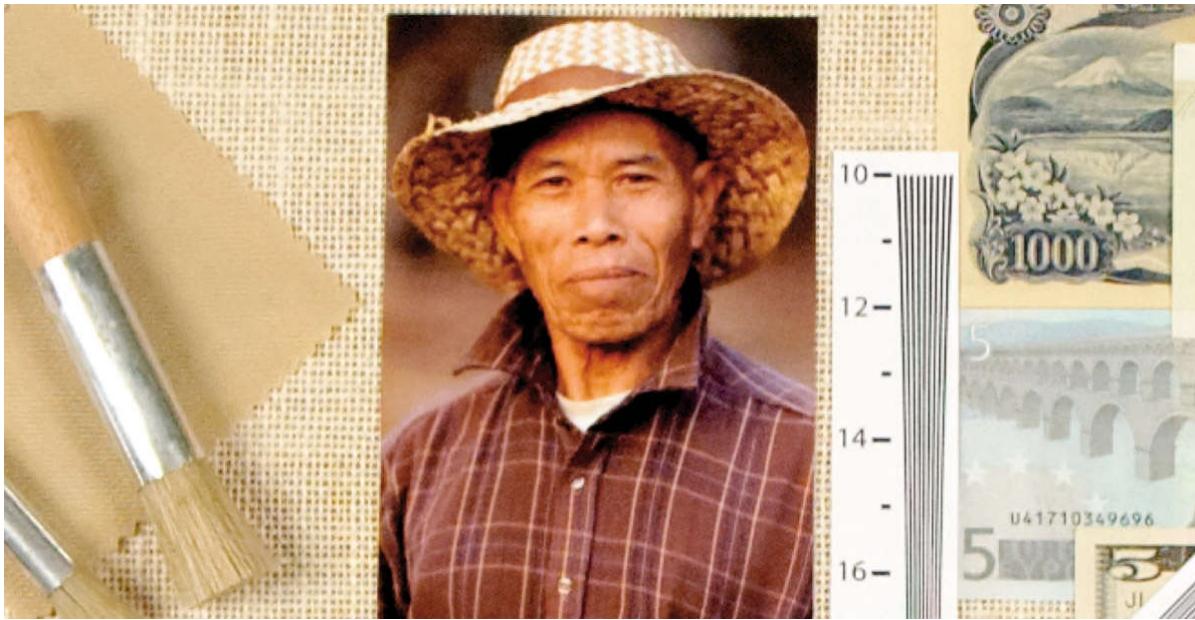


Figure 30. Débruitage par fusion de quatre prises (source DxO.com).

CORRECTION DU VIGNETTAGE

Le vignettage est illustré par la figure 31, prise au très grand-angle depuis la terrasse de mon bureau (si, si !). Elle montre un fort assombrissement des coins de l'image dû au fait que l'objectif leur envoie moins de lumière qu'au centre. Le vignettage peut constituer un effet artistique intéressant, mais il est le plus souvent gênant. Il est surtout indispensable de le supprimer lorsqu'on veut assembler plusieurs photos manuellement ou informatiquement, par exemple pour faire un panorama. Comme le vignettage est prévisible en fonction du capteur, de l'objectif et des paramètres de prise de vue, il est facile à annuler. Tous les appareils numériques le font en standard. La figure 32 montre sa correction automatique par DxO Optics Pro, qui rétablit l'équilibre des lumières. La différence est bien visible dans les coins, en particulier dans la végétation en bas à gauche. Tout autre logiciel compétent sait le faire, et tous les nouveaux appareils et téléphones intègrent directement cette correction. Mais, si l'on veut obtenir des effets artistiques intéressants, on peut aussi augmenter soi-même le vignettage.



Figure 31. Vignettage (source DxO.com).



Figure 32. Vignettage corrigé, distorsions aussi (DxO Optics Pro).

CORRECTION DES DISTORSIONS GÉOMÉTRIQUES

Les corrections suivantes ont pour but de défaire les distorsions géométriques que possède nécessairement tout objectif. Pour être efficaces, elles demandent une caractérisation de chaque couple boîtier/objectif à l'aide de mesures précises et nombreuses en variant l'ouverture et la distance de mise au point. Cela demande un travail de fourmi, toutefois largement automatisable.

J'ai appliqué une telle correction pour passer de la figure 31 à la figure 32. Ici, j'ai corrigé l'*anamorphose de volume*, qui est une déformation globale de l'image provoquée de façon inévitable par un objectif grand angle. Elle étire les bords par rapport au centre et courbe les lignes (les téléobjectifs ou zooms de longue focale ont tendance à faire l'inverse). La différence est surtout évidente quand on connaît la vue.

Mais le lecteur pourra quand même remarquer que les lignes sont globalement plus naturelles, surtout pour la rambarde en bas à droite.

La photo de la figure 33 a trois défauts bien visibles, tous corrigés automatiquement dans la figure 34 : un problème de lumière avec des vitraux clairs et des murs foncés, corrigé comme pour la figure 27, un peu de vignettage, et surtout un problème spectaculaire de distorsion géométrique sous la forme de forte courbure des lignes droites. C'est encore une déformation due au grand-angle, indispensable ici à cause du manque de recul. On peut aimer cet effet et le conserver, mais on peut aussi choisir de le corriger automatiquement en défaisant algorithmiquement ces distorsions à l'aide des tables préconstruites pour le couple appareil/objectif. Cette méthode, introduite vers 2005 par DxO Labs, est maintenant employée à l'intérieur même de la plupart des appareils modernes. Il faut noter que ce genre de distorsion est insupportable quand on photographie des personnes, car leurs têtes peuvent avoir des formes épouvantables si elles se trouvent près des bords. La correction est là indispensable.

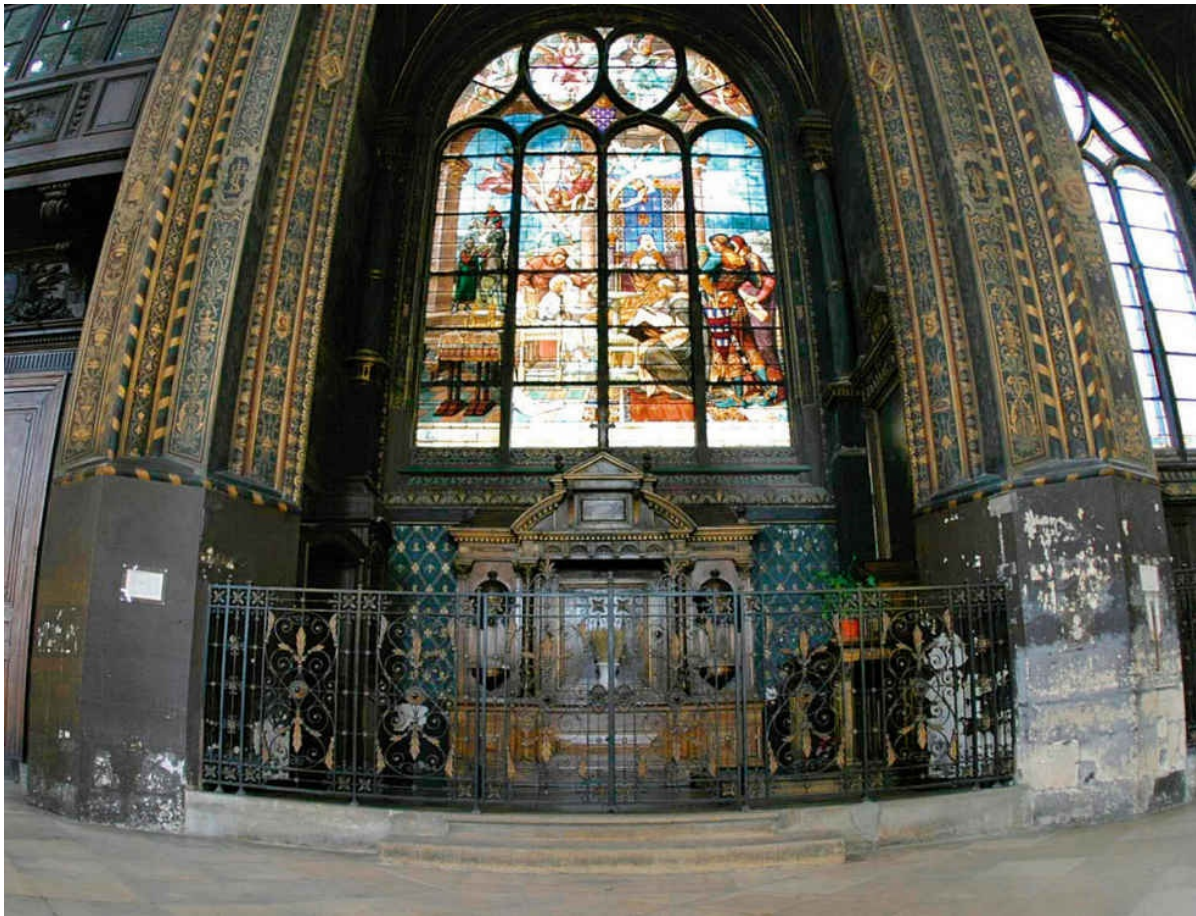


Figure 33. Avant correction de distorsion et de lumière (source DxO.com).

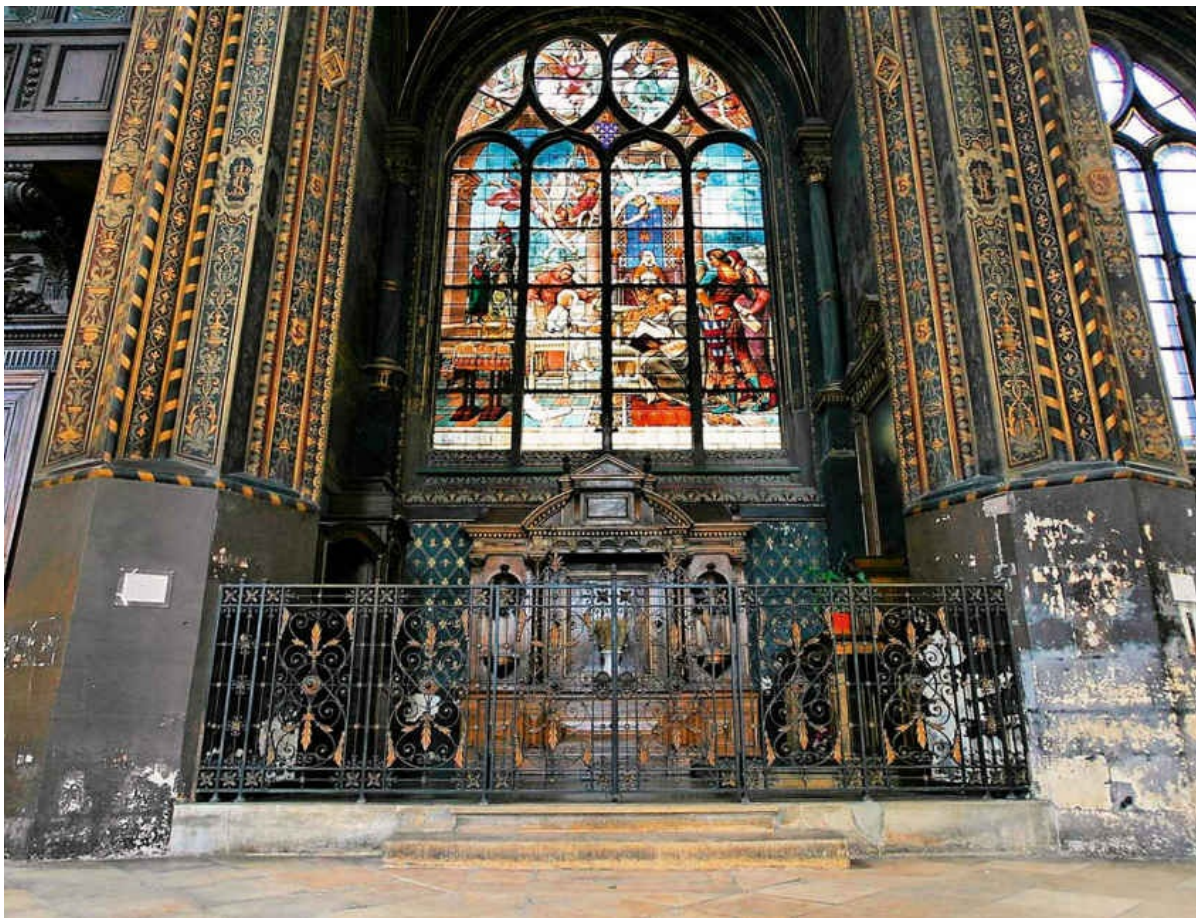


Figure 34. Après correction de distorsion et de lumière (source DxO.com).

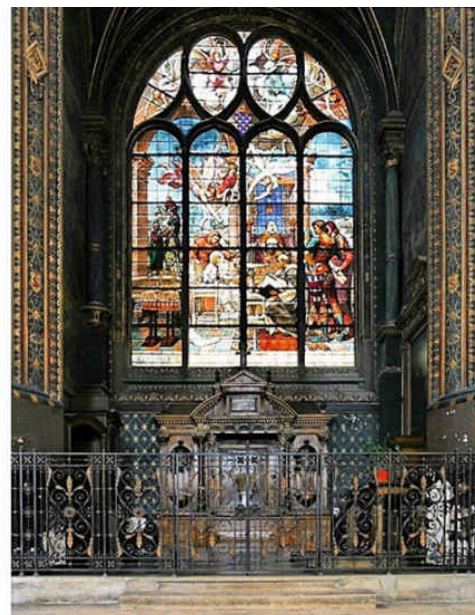
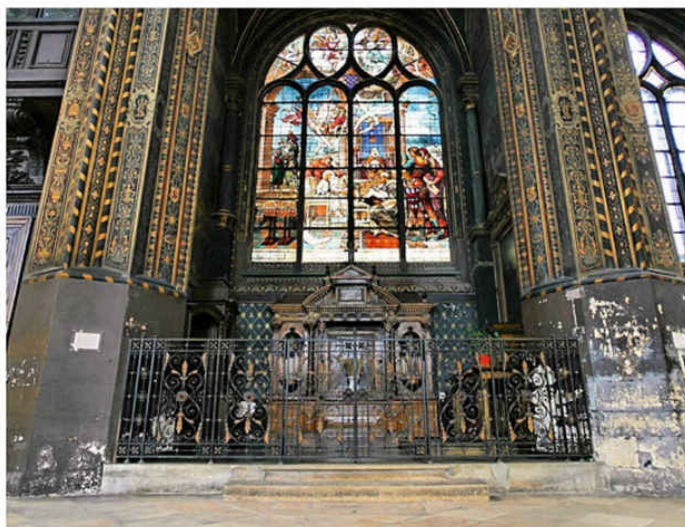


Figure 35. Correction automatique des fuyantes (source DxO Labs).

Mais les lignes fuyantes restent très prononcées dans la figure 34 par rapport à notre perception habituelle – encore un effet qui n’apparaît pas à notre cerveau lorsque nous visitons la cathédrale. Il est possible de corriger les fuyantes à l’aide d’algorithmes géométriques, soit automatiquement et complètement comme dans la figure 35, soit manuellement et moins violemment si l’on souhaite simplement garder des fuyantes moins prononcées. Notons qu’en photo d’architecture on utilise plutôt des objectifs à bascule et décentrement, qui permettent de corriger les fuyantes dès la prise de vue. Mais ces objectifs sont très chers et d’utilisation trop rare pour être intéressants pour un amateur, pour qui les corrections logicielles sont idéales.

CORRECTION DES ABERRATIONS CHROMATIQUES

Les aberrations chromatiques provoquent des franges de couleurs, en particulier dans les zones à contours francs, comme illustré par l’image de gauche de la figure 36, elle-même agrandissement d’une petite partie d’une photo. Elles sont dues à un phénomène physique incontournable : l’indice de réfraction de tout verre dépend de la longueur d’onde de la lumière. Cela signifie que les rayons rouges et bleus, qui sont aux extrémités du spectre de la lumière visible, ne vont pas se focaliser sur le capteur au même endroit que le vert intermédiaire même s’ils ont été émis par la même source. Le résultat prend la forme de franges violettes le long des contours. Mais, puisqu’elles sont mesurables étant donné un capteur et un objectif, les aberrations chromatiques peuvent être corrigées algorithmiquement. C’est ce qui est fait dans l’image de droite de la figure 36.



Figure 36. Correction d'aberrations chromatiques (source DxO.com).

Notons que ce phénomène existe aussi en argentique. Comme il n'est pas possible de le contrer au niveau de la pellicule, les opticiens ont développé des trésors d'ingéniosité pour réduire ces franges en mélangeant différents types de verres dans les objectifs, dont des verres à faible dispersion chromatique, très coûteux. Avec un capteur numérique et des corrections algorithmiques, la situation est donc bien plus simple.

Par une intéressante inversion mentale, il peut être à l'inverse intéressant d'*augmenter* l'aberration chromatique, ici dans sa composante longitudinale qui détermine la taille de la tache provoquée sur le capteur par les rayons émis par un point donné de la scène. Le rouge, le bleu et le vert ne focalisant pas à la même distance de l'objectif, cette taille dépend de la longueur d'onde. Augmentons l'aberration chromatique longitudinale. Alors, sur le capteur, une des trois images de couleur sera beaucoup plus nette que les deux autres, ce qui est facile à déterminer

algorithmiquement. Cela permet d'avoir une idée de la distance à laquelle il aurait fallu régler l'objectif. Mais on peut alors rendre plus nettes les deux autres photos par des algorithmes utilisant cette information. En combinant les trois images nettes finales, on obtient alors une image nette partout ! Cette méthode typiquement informatique a été utilisée par DxO Labs pour les téléphones, avant que l'augmentation considérable de la résolution de leurs capteurs ait rendu la précision bien plus grande de la mise au point automatique (autofocus) indispensable.

La fusion algorithmique d'images multiples

Jusqu'à maintenant, nous avons expliqué des algorithmes qui s'appliquent aux données d'une photo individuelle. Mais, puisque les appareils numériques et téléphones savent aussi filmer, ils sont capables de prendre plusieurs photos pleine taille très rapidement (jusqu'à 50 ou 100 images par seconde pour les meilleurs). Cela ouvre des possibilités tout à fait nouvelles en fusionnant algorithmiquement ces images.

DÉBRUITAGE DU BRUIT PAR FUSION D'IMAGES DANS LES TÉLÉPHONES

Nous avons déjà parlé du débruitage par fusion d'images. Il est systématiquement employé dans les téléphones, à tel point que la photo peut donner l'effet d'être prise *avant* qu'on appuie sur le déclencheur et pas après ! En effet, dès que l'appareil photo du téléphone est mis en marche, il filme en continu. L'appui sur le déclencheur lui dit simplement qu'il faut fusionner une dizaine d'images (ou plus) prises avant et un peu après son moment. Comme il y a bien plus de chances que le téléphone soit bien immobile avant et que l'utilisateur bouge tout de suite après avoir appuyé, les constructeurs privilégient les images prises avant. Sur les iPhone récents, quand on regarde une photo, on peut effectivement la voir bouger un peu, par un visionnage comme film de la rafale d'images.

GESTION DES DYNAMIQUES EXTRÊMES

Une autre utilisation importante de la fusion est l'*augmentation de la dynamique*, ou HDR (*high dynamic range*), illustrée par la figure 37 pour les quatre images sources de la Sainte-Chapelle prises avec quatre diaphragmes différents et la figure 38 pour l'image finale. Ici, le problème est le très grand écart de luminosité entre les vitraux, très lumineux, et l'intérieur, très sombre. Dans notre vision, cet écart est traité par notre accommodation permanente sur la petite zone d'intérêt, alors que l'appareil photo prend toute l'image d'un coup. L'algorithme HDR sélectionne automatiquement les parties les mieux exposées dans chaque zone de chaque image puis les fusionne algorithmiquement pour donner l'image finale. Une bonne fusion agréable à l'œil n'est cependant pas facile à réaliser, car le résultat prend vite un aspect artificiel. (Il semble que des algorithmes d'apprentissage automatique pourront donner de très bons résultats à partir d'une seule image crue – à suivre !)

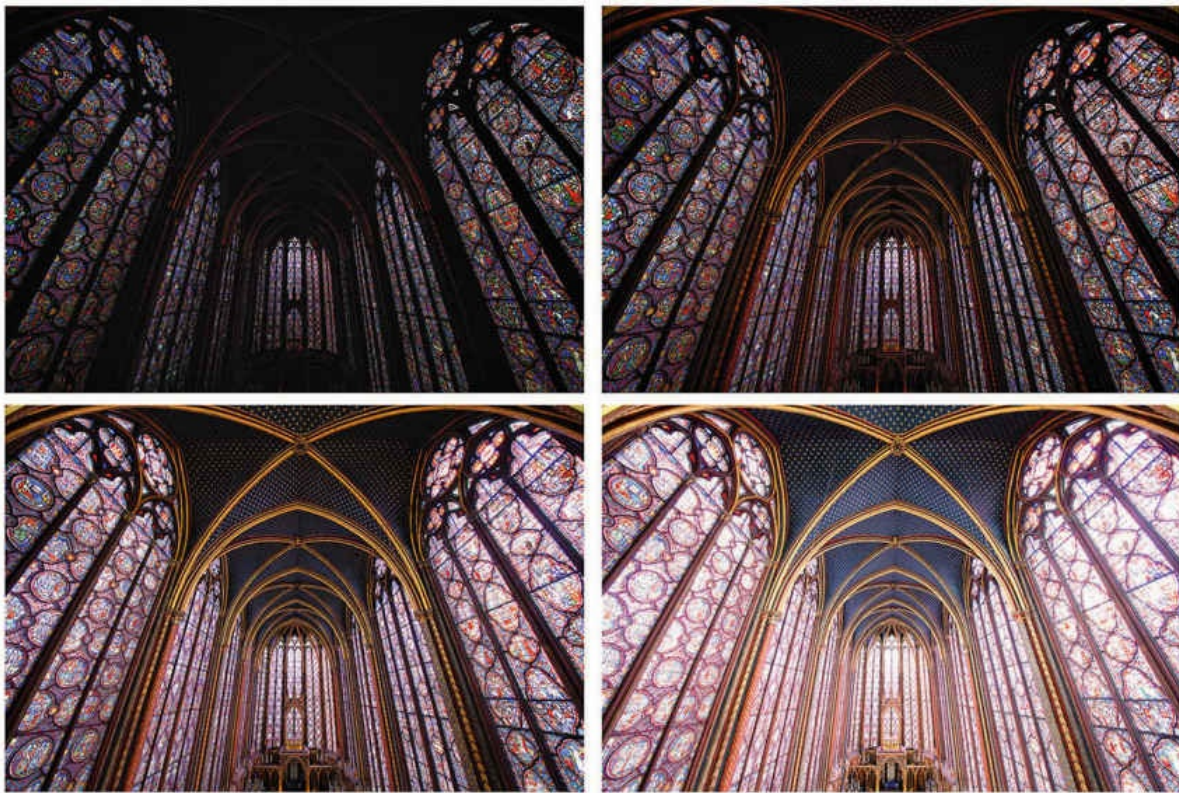


Figure 37. Quatre prises d'une séquence HDR (© DxO Labs-Hugo Uyttersprot/CMN-Sainte-Chapelle).

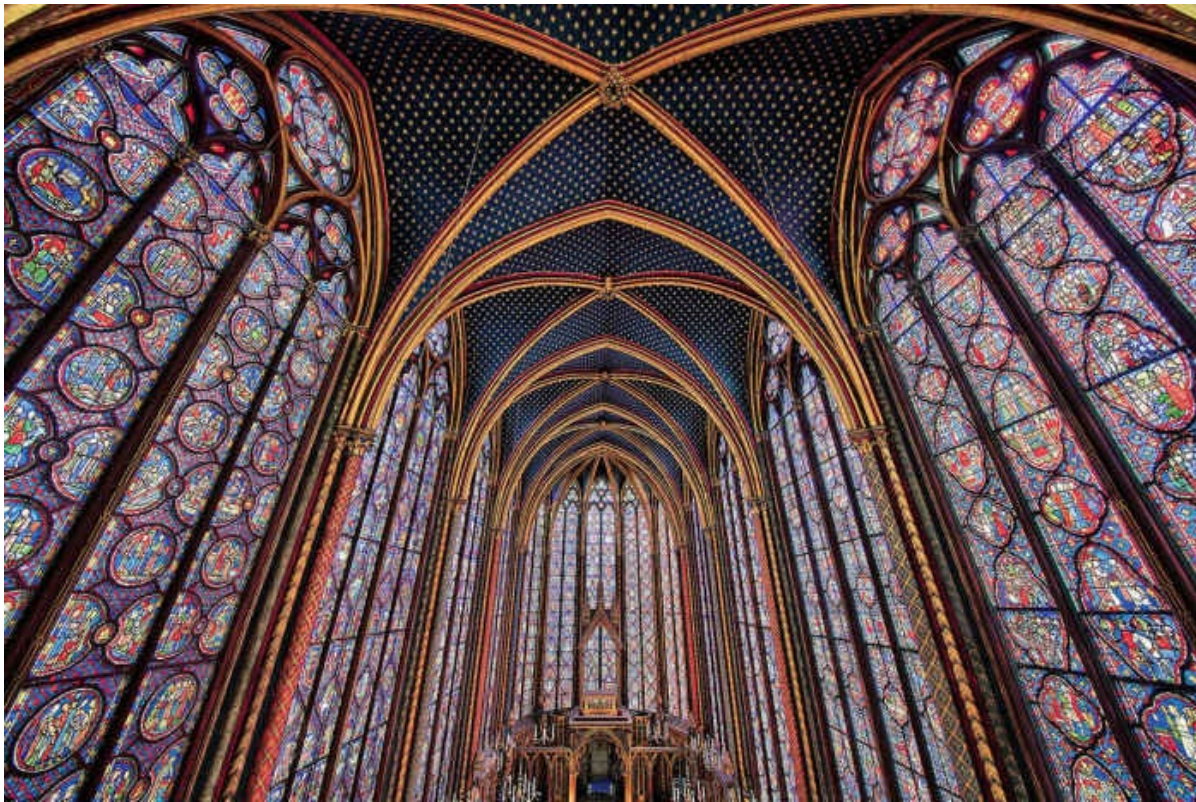


Figure 38. Résultat de la fusion HDR (© DxO Labs-Hugo Uyttersprot/CMN-Sainte-Chapelle).

FUSION DE MISES AU POINT SUCCESSIVES



Figure 39. *Focus stacking*, fusion de mises au point successives (source Muhammad Mahdi Karim, CC-BY-SA-3.0).

Une troisième utilisation de la fusion, particulièrement intéressante en macrophoto où la profondeur de champ est par construction très faible, est ce qu'on appelle le *focus stacking*. C'est la fusion d'images successives dont la mise au point est légèrement décalée, automatiquement bien sûr. L'algorithme de fusion détecte quelles parties sont nettes dans quelles images et fusionne ces parties nettes dans l'image finale. Le résultat, spectaculaire, se voit bien dans la figure 39, où les deux images de gauche sont les deux extrêmes d'une série de six et où celle de droite est l'image finale.

SUPER-RÉSOLUTION

Une autre application très intéressante est la *super-résolution*. Ici, il s'agit d'acquérir une série d'images en décalant mécaniquement le capteur d'une fraction de la largeur d'un photosite après chaque prise. Au niveau de l'information enregistrée, tout se passe comme si le capteur avait beaucoup plus de pixels qu'il n'en contient réellement. On peut ainsi améliorer sensiblement la définition de l'image finale – à deux conditions cependant : que la qualité de l'optique soit exceptionnelle, sinon aucun gain réel n'est possible, et que le sujet soit resté immobile pendant la prise de vue.

AUTRES APPLICATIONS

D'autres applications de la fusion d'images sont implémentées dans nos téléphones, par exemple pour faire des panoramas en déplaçant simplement son appareil. Si les algorithmes sont bons, le panorama l'est aussi.

Enfin, je mentionnerai une application amusante : la suppression des éléments parasites tels que les piétons dans une photo de monument ou de rue. Ici, on prend une série de photos espacées de quelques secondes. Si les piétons bougent, on a assez de matériau pour reconstruire algorithmiquement une photo sans piéton¹⁶ ! On peut aussi prendre plusieurs photos d'un groupe et en extraire les personnages les mieux mis en valeur dans chaque photo pour en fabriquer une meilleure où tout le monde est en valeur. L'imagination est sans limites dans ce domaine. Mais il ne faut plus du tout

croire qu'une photo correspond à une scène réelle, sans même qu'on ait besoin de longues et pénibles séances d'édition manuelle comme au temps de l'URSS...

Aider la prise de vue

Jusqu'ici, j'ai parlé des algorithmes intervenant dans la construction de l'image finale. Mais il y en a bien d'autres dédiés à la prise de vue, étape évidemment essentielle car déterminant définitivement les données entrant dans les traitements ultérieurs. Certains de ces algorithmes numériques remplacent les algorithmes analogiques qui existaient dans les derniers appareils argentiques, en particulier pour le calcul de l'exposition et la mise au point. Ils sont devenus très perfectionnés, effectuant une analyse informatique poussée de l'image pour comprendre quelle est sa nature et comment bien régler ces paramètres.

LA STABILISATION DE L'OBJECTIF OU DU CAPTEUR

Ici, il s'agit de corriger le fait que nous tremblons tous plus ou moins. L'idée est donc de stabiliser l'image en compensant les vibrations des mains de l'utilisateur. Le système concerné est vraiment cyberphysique : des algorithmes gèrent le couplage entre des gyroscopes très précis qui détectent les micromouvements et des lentilles mobiles dans l'objectif ou un capteur mobile dans le boîtier commandé par micromoteurs. Il devient même courant de coordonner ces deux types de stabilisations dans l'objectif et dans l'appareil. La stabilisation permet de gagner de l'ordre de quatre vitesses ou diaphragmes pour les prises à main levée, ce qui est extrêmement important en basse lumière ou en utilisant un téléobjectif sans pied, et permet aussi de se passer souvent du flash.

Cela veut aussi dire que chaque objectif stabilisé contient un microprocesseur et du logiciel, qu'il faut d'ailleurs mettre à jour de temps en temps !

LES NOUVEAUX VISEURS

Le viseur est évidemment un élément critique pour la prise de vue. Ici encore, l'informatisation change complètement la donne. Les appareils argentiques possédaient deux types de viseurs : les viseurs directs, utilisés pour les compacts, et les viseurs reflex, qui visent à travers l'objectif à l'aide d'un miroir qui remonte avant la prise de vue et redescend après. Les premiers ont pour eux la simplicité et la discrétion de la prise de vue, mais contre eux une précision de visée toute relative, sauf pour de célèbres appareils de reportage très coûteux. Les seconds ont pour avantage la précision de la visée, et pour inconvénients l'encombrement et le bruit.

Les appareils numériques haut de gamme ont longtemps conservé la visée reflex avec tous ses avantages, ce qui les a fait rassembler comme des frères à leurs prédécesseurs, à part pour la présence systématique d'un écran arrière. Certains appareils non reflex, assez rares, ont conservé la visée directe à l'ancienne, mais souvent avec des viseurs médiocres. Les plus simples et les téléphones se sont contentés de l'écran arrière pour la visée, ce qui a conduit à d'étonnantes positions des photographes penchés en arrière, tenant leur appareil à bout de main et ne voyant quasiment rien lorsque leur écran est au soleil.

Mais de nouveaux venus sont en train de changer la donne : les viseurs électroniques LCD (*liquid crystal display*) placés à hauteur d'œil, qui reprennent le principe ergonomique de la visée reflex. Cependant, au lieu que l'oculaire montre la lumière entrante réfléchiée par le miroir et redressée par le pentaprisme qui donne cet aspect caractéristique aux reflex, il montre directement un petit écran LCD de haute résolution intégré au boîtier. Au début, la qualité de visée était mauvaise, mais les écrans internes se sont tellement améliorés qu'elle est devenue très bonne ; il n'y a pas de doute qu'elle s'améliorera encore beaucoup grâce aux efforts des physiciens. Les avantages sont énormes en pratique, avec d'une part un gain de poids et une simplification mécanique, et d'autre part le fait qu'on peut intégrer à la visée des fonctions simples à réaliser avec des algorithmes mais complexes ou impossibles à

réaliser autrement. Citons-en quelques-unes d'usage absolument constant, aussi utilisables sur l'écran arrière :

- L'intégration optionnelle de l'affichage de tous les paramètres de la prise de vue.
- Celle de niveaux permettant de vérifier immédiatement l'horizontalité et la verticalité, dont on ne peut rapidement plus se passer.
- Le scintillement optionnel des contours des visages ou des objets lorsqu'ils sont au point, ce qui permet de faire rapidement et commodément des mises au point fines ou d'affiner l'autofocus.
- La mise en couleurs bien visibles des à-plats saturés éventuels pour éviter la surexposition, fatale en numérique.
- La possibilité de faire apparaître avant la prise de vue des *histogrammes*, diagrammes permettant de comprendre en un clin d'œil la qualité de la lumière entrante (ces histogrammes sont toujours visibles sur l'écran arrière après la prise de vue, mais c'est parfois trop tard).
- Le test de profondeur de champ en fermant l'objectif au diaphragme final prévu sans pour autant assombrir l'image, impossible sur un reflex à miroir.

Quelques-unes de ces fonctions peuvent aussi être ajoutées à un viseur reflex classique par superposition à l'image d'un écran LCD contenant les mêmes informations, mais le résultat est nettement moins agréable à l'utilisation. Ajoutons que l'image du viseur LCD est toujours lumineuse, même en basse lumière, et encore plus exacte que sur un reflex, car ce qu'on voit est exactement ce qui arrive au capteur.

LA MISE AU POINT

Une difficulté constante est de bien faire la mise au point. Tous les appareils savent la faire tout seuls, soit sur les visages ou les yeux qu'ils détectent algorithmiquement, soit sur le point le plus proche. Ils savent aussi suivre les objets mobiles¹⁷. Mais il arrive souvent que ce comportement par défaut ne soit pas adapté, par exemple parce qu'on veut faire la mise au point sur un élément d'arrière-plan. Il faut alors reprendre la main. Avec l'écran arrière ou celui d'un téléphone, une excellente méthode est de simplement toucher du doigt la zone qu'on veut voir nette.

Avec un viseur d'œil, on ne peut pas le faire, mais de plus en plus d'appareils permettent ce qu'on appelle le *focus peaking*, déjà mentionné ci-dessus : en tournant une bague sur l'objectif, on peaufine la netteté jusqu'à ce que les contours des visages ou objets qu'on veut nets scintillent. C'est facile à voir et très rapide à régler, donc particulièrement efficace. D'autres appareils vous proposent même de suivre votre regard dans le viseur et de mettre au point sur ce que vous fixez.

On voit ainsi que les nouveaux appareils font appel à une imagination sans bornes, bien soutenue par les algorithmes et logiciels : il ne s'agit pas seulement de s'interfacer avec les processus physiques, il faut aussi s'interfacer avec nos comportements.

Tout cela ne fera bien sûr pas immédiatement changer d'avis un professionnel devenu intime avec son viseur reflex, et c'est bien normal. Mais le sens de l'histoire paraît inéluctable.

La coconception des objectifs et des algorithmes

Les transformations décrites précédemment l'étaient du point de vue de l'utilisateur. Venons-en maintenant à ce qui change pour les fabricants, et qui est encore plus profond. Au xx^e siècle, les conceptions des appareils et des objectifs étaient très indépendantes, car il n'y avait que peu d'interactions entre les deux, au maximum par la languette de fermeture du diaphragme et la communication de l'ouverture par exemple. Tout est différent maintenant, et la situation peut se résumer en une phrase :

Boîtiers, objectifs, et algorithmes doivent être conçus de façon coordonnée, avec les algorithmes au centre.

Cette inversion mentale va en fait très loin. Prenons d'abord deux exemples.

SIMPLIFIER LES ZOOMS

Les zooms, inventés au début des années 1950 par Roger Cuvillier et Pierre Angénieux, permettent de faire varier continûment la focale sans changer l'objectif, en déplaçant des lentilles à l'intérieur de l'objectif. Ils sont vite devenus très populaires, et forment le gros des ventes aujourd'hui. Mais leur optique est complexe,

assemblant une ou deux dizaines de lentilles et demandant des calculs savants pour réduire les distorsions et les aberrations chromatiques. Cela veut aussi dire que leurs distorsions sont compliquées et difficiles à bien défaire algorithmiquement, y compris pour avoir un bon affichage de l'image finale dans le viseur dès la prise de vue.

Puisqu'on peut maintenant défaire ces défauts, il n'y a plus de raison de les minimiser optiquement : il devient préférable de concevoir des objectifs avec davantage de défauts plus faciles à corriger, ce qui permet de les faire plus simples, plus petits, plus légers et moins chers, tout en simplifiant aussi les algorithmes. Il reste pour moi étonnant de voir dans les journaux spécialisés des tests d'objectifs se limitant aux performances brutes de ceux-ci. Voici une illustration de plus du fait que le schéma mental évolue doucement au milieu de telles inversions. Puisque le résultat final est obtenu par la paire objectifs-algorithmes, c'est elle qu'il faudrait tester. Mais c'est évidemment plus difficile, car on peut employer différents logiciels pour le même objectif.

LA SUPPRESSION DU MIROIR

Encore plus important : si on passe du reflex à la visée électronique, on supprime le miroir, qui exige beaucoup de place pour remonter à la prise de vue. Les objectifs des reflex étaient placés loin de la pellicule pour laisser cette place. Cette distance est devenue inutile, ce qui fait qu'on peut grandement simplifier et alléger les objectifs en les plaçant beaucoup plus près du capteur, voire pratiquement collés à celui-ci comme dans les téléphones dont la finesse est un argument de vente essentiel. La qualité des résultats est tout aussi bonne que celle d'un excellent objectif classique, beaucoup plus gros et lourd, mais la conception même de l'objectif devient assez différente, et ne peut se faire que grâce à des algorithmes sophistiqués.

La cartographie numérique

Un autre domaine totalement révolutionné de manière similaire est la cartographie. Les premières cartes sont très anciennes, et ont été associées à des supports papiers jusqu'à la fin du xx^e siècle avant de basculer vers les supports numériques. Le téléphone est devenu le support préféré pour la lecture de cartes chez monsieur Tout-le-monde, et le grand écran chez les professionnels. Ce basculement a donné lieu à une intéressante inversion mentale que je vais décrire ci-dessous grâce à un exemple personnel.

MAIS PERSONNE N'A DE GPS SUR LUI !

En 2008, dans une réunion au ministère de l'Éducation nationale consacrée à ce que pourrait être le numérique dans l'éducation, je discutais avec des inspecteurs généraux et des inspecteurs d'académie. Ils m'ont demandé de donner un exemple de quelque chose qui allait être vraiment modifié par l'informatique et qui serait aussi directement relié aux programmes scolaires. J'ai choisi une superbe inversion mentale fournie par l'exemple de la cartographie, car s'orienter sur une carte était au programme du collège. Je leur ai demandé comment s'organisait la pédagogie. Ils m'ont répondu qu'on ouvrait d'abord la carte papier, qu'on cherchait où on était, puis où on devait aller, et enfin qu'on cherchait l'itinéraire. Je leur ai d'abord fait remarquer que, dans leur système, il fallait d'abord posséder la carte, de plus à la bonne échelle. Je leur ai ensuite dit qu'il me paraissait évident que ce n'était plus cela

qu'il faudrait enseigner à l'avenir, mais quelque chose de tout à fait inverse qui deviendrait rapidement naturel pour les collégiens et même leurs frères et sœurs encore à l'école primaire : on allume la carte informatique, qui contient d'origine tous les pays à toutes les échelles et avec plusieurs niveaux de détail ; on sait immédiatement où on est parce que c'est marqué sur la carte ; pour savoir où se trouve sa destination, on tape son nom, et l'itinéraire s'affiche immédiatement en plusieurs variantes selon qu'on est à pied, en voiture ou en transport en commun.

- Comment ça ?
- Il suffit d'avoir un GPS sur soi.
- Mais personne n'a de GPS sur lui !
- Sauf les jeunes, qui en auront toujours un dans leur téléphone !

J'ai vu un certain scepticisme dans l'assistance, car les smartphones n'étaient pas encore généralisés. Mais il m'a suffi d'attendre quelques années pour que ma prédiction (triviale) se réalise. Travaillant à l'époque sur les circuits de téléphones mobiles, je savais bien que l'intégration du GPS y était déjà en route et que les smartphones allaient totalement changer la donne.

LE TRAITEMENT NUMÉRIQUE DES INFORMATIONS CARTOGRAPHIQUES

Si les cartes numériques sont d'usage récent dans le grand public, le traitement numérique d'informations cartographiques de toutes sortes est ancien. On peut dire qu'il date du lancement des premiers satellites dédiés comme les SPOT (Satellites pour l'observation de la Terre) dont le premier a été lancé en 1986. Ces satellites nous ont rapidement envoyé des images et des télémessures de toutes sortes, qui ont permis d'organiser la cartographie d'une façon beaucoup plus globale qu'auparavant, et surtout d'utiliser les nouvelles informations calculées par la *télédétection*, qui analyse les images à l'aide d'algorithmes puissants et variés. La télédétection a permis des études poussées et à grande échelle dans des domaines inaccessibles auparavant, comme les zones agricoles, les forêts et leurs incendies, les estuaires des fleuves, les avalanches, les zones urbaines, etc.

En parallèle, le matériel cartographique déjà existant a été numérisé et mis en liaison avec les photos aériennes pour fournir des cartes unifiant sous format numérique bien des informations qui n'étaient disponibles que de façon séparée auparavant. Ce travail a d'abord été réalisé par les institutions officielles des États, en général propriétaires des données. Mais il a été rapidement révolutionné par de nouveaux acteurs venus d'un monde très différent, celui de l'informatique pure et dure. On pense en particulier à Google, société née de son moteur de recherche, encore toute petite en 2000, mais qui a fait un effort considérable et très différent de celui des institutionnels : fournir au grand public des cartes du monde qui soient gratuites, automatiquement à jour et interactives. Intégrées au téléphone portable et à son GPS, ces cartes ont rapidement pris l'ascendant sur celles des fabricants de GPS classiques : si, dans les années 2000, tout vendeur de matériel électronique avait un solide rayon dédié aux GPS pour automobiles ou randonneurs, il devient bien plus difficile d'en trouver maintenant. Et beaucoup de gens utilisent l'écran du GPS installé d'origine dans leur voiture pour y coller la ventouse du support de leur téléphone, en particulier parce que son ergonomie est souvent meilleure et parce que des applications populaires fournissent des informations sur l'état des routes et de la circulation obtenues de façon collaborative et non disponibles gratuitement sur les GPS classiques.



Figure 40. Vue aérienne virtuelle d'une vallée des Alpes (image © 2017 Google-DigitalGlobe ; données cartographiques © 2017 Google).

En même temps, de nouvelles fonctionnalités ont été intégrées à ces systèmes de cartographie universelle, en particulier en fusionnant données topographiques et données photographiques. Une bonne illustration de ce type d'intégration est fournie par la figure 40. Il s'agit d'une vue en trois dimensions d'un coin du massif de la Vanoise que j'aime bien. L'image a la forme classique d'une photo prise d'avion, sauf qu'il n'y a jamais eu d'avion pour la prendre mais seulement des algorithmes et logiciels calculant à partir de données topographiques diverses et de photographies aériennes ou satellitaires, jouant ainsi le rôle d'un avion virtuel. Pour la photographie, nous avons montré l'importance de la fusion d'images. Ici, on va plus loin en fusionnant images et données de types très différents. Attention, ce n'est pas simple à faire !

Les données peuvent aussi être fournies de façon collaborative par les utilisateurs. Ainsi, le projet OpenStreetMap permet de mettre à la disposition des autres utilisateurs ses propres informations sous forme d'annotations sur les cartes dès qu'elles sont enregistrées dans leur téléphone ou leur ordinateur. Et, bien sûr, Google

Street View permet d'explorer les coins et recoins des villes et même parfois de la campagne, d'une façon qui pose d'ailleurs problème : tout le monde souhaite-t-il vraiment que sa façade et son jardin soient mis dans le domaine public sans son assentiment ?

Tout cela pose évidemment des problèmes d'accès aux données. À qui appartiennent-elles ? Sont-elles libres d'accès, ou quel est leur prix ? Lesquelles doivent rester confidentielles ? Ces questions sont bien sûr âprement discutées un peu partout, avec des avis très divergents...

LE GPS ET SES SUCCESSEURS, DES INSTRUMENTS DEVENUS INDISPENSABLES

Le GPS est devenu l'outil numéro un pour la géolocalisation. Mais il ne sert pas qu'à cela, car il a un autre rôle très important : celui de *gardien de l'heure*, en particulier dans les réseaux informatiques. Pour comprendre cela, il faut d'abord comprendre son principe. Les satellites GPS contiennent des horloges atomiques hyperprécises développées par les physiciens. En consultant ces horloges, ils ont pour rôle principal d'envoyer dans l'espace des messages contenant leur heure précise d'envoi. Pour que l'ensemble du système fonctionne, il faut que les horloges des satellites soient très précisément synchronisées entre elles, ce qui demande des corrections fines de relativité générale.

Tout récepteur GPS est à l'écoute de ces messages. Supposons d'abord qu'il contienne lui-même une horloge atomique synchronisée avec celle des satellites. Une fois multipliée par la vitesse de la lumière, la différence entre l'heure de réception du message et l'heure qu'il contient, c'est-à-dire son heure d'émission, donne précisément la distance au satellite. Comme la position de celui-ci est parfaitement connue, le récepteur se situe sur la surface d'une sphère de centre le satellite et de rayon cette distance. En visant trois satellites, on obtient trois sphères dont l'intersection doit être un point, avec bien sûr une certaine incertitude, mais qui reste contrôlable. Cependant, le récepteur ne contient évidemment pas d'horloge atomique. Il s'en sort en visant quatre satellites, ce qui suffit pour caractériser ses trois

coordonnées d'espace et sa coordonnée de temps avec des calculs bien sûr plus compliqués (pas étonnant donc que le GPS use la pile).

LA SYNCHRONISATION MONDIALE DES ORDINATEURS

Le GPS permet de calculer avec une grande précision non seulement l'endroit où l'on se trouve, mais aussi quelle heure il est à cet endroit, cela avec une précision très supérieure à celle de tout autre système. Une conséquence inattendue est qu'il est utilisé pour une autre fonction essentielle : la *synchronisation temporelle mondiale* de tous les ordinateurs. Mais ceux-ci n'ont en général pas de récepteur GPS, qui ne fonctionne pas à l'intérieur des maisons et des immeubles. La synchronisation mondiale des ordinateurs se fait donc à partir d'un nombre respectable de stations terrestres GPS locales de coordonnées parfaitement connues et d'un système de messages réseaux transmis par exemple sur Internet, permettant ainsi un calcul stochastique de l'heure à quelques millisecondes près dès que l'ordinateur se connecte au réseau. C'est l'objet du protocole *NTP* (Network Time Protocol) qu'utilisent tous vos ordinateurs, en général à votre insu.

NTP est le résultat d'un effort collectif extraordinaire d'une communauté mondiale de chercheurs et d'ingénieurs, bien moins connue que celle du système d'exploitation ouvert Linux par exemple mais dont l'impact est également important. Un problème difficile et encore plus inattendu qu'il a eu à résoudre est la *sécurité informatique* de la synchronisation. Beaucoup de systèmes sensibles dépendent de la qualité de cette synchronisation, et casser le protocole NTP provoquerait de véritables désastres. Ces problèmes de sécurité sont effectivement traités en profondeur dans NTP et en représentent de fait une partie très importante.

Signalons enfin que de nouveaux systèmes beaucoup plus précis existent déjà et vont être rendus rapidement accessibles au grand public, comme par exemple le système européen Galileo, dont la précision sera bien meilleure que celle du GPS.

Le passage à la troisième dimension

L'espace qui nous entoure n'est pas en deux dimensions mais en trois. L'architecture et la sculpture travaillent naturellement en 3D, n'utilisant la 2D que pour les croquis et plans. La peinture, restée en 2D, a cherché à donner l'illusion de la 3D dès l'invention de la perspective à la Renaissance. L'algorithmique permet de réaliser beaucoup de prouesses dans ce domaine.

LA PHOTOGRAPHIE ET LE CINÉMA 3D

En photographie, la 3D a connu une certaine popularité avec les stéréoscopes à deux objectifs, nés dès le milieu du XIX^e siècle. Ils donnent l'illusion de la 3D en prenant deux images, une pour chaque œil, qu'il faut regarder avec un appareil binoculaire spécifique. Quand j'étais enfant, j'aimais beaucoup acheter des souvenirs de monuments sous forme de cartons incrustés de paires de minidiapositives à faire défiler dans un stéréoscope en plastique, une image de chaque paire par oculaire. Le cinéma aussi s'est mis tôt à la 3D : une des plus belles photos de *Life Magazine* a été prise a été prise lors d'une projection du film *Dial M for Murder* (*Le crime était presque parfait*) d'Alfred Hitchcock. On y voit les spectateurs ornés de superbes lunettes aux grosses montures en carton blanc. La 3D s'est introduite plus récemment dans la télévision, où elle a fait un beau flop, car trop fatigante à regarder pour pas assez de contenu.

Une possibilité algorithmique très intéressante et utilisée par exemple par certaines agences immobilières est de reconstruire la 3D à partir d'un lot de photographies prises sous divers angles en nombre suffisant. Les algorithmes mettent ces images en correspondance en détectant leurs points saillants et en effectuant des calculs géométriques de haut vol. Le résultat est impressionnant en visualisation 3D, dont je reparlerai dans le chapitre suivant.

LE SON 3D

Le son aussi passe en trois dimensions, soit dans les concerts comme je l'ai mentionné au début de ce chapitre, soit avec un casque utilisant un enregistrement *binaural*, c'est-à-dire fait avec un dispositif qui imite nos têtes, soit par transformation algorithmique. Le son 3D est assez impressionnant, et de nombreux exemples en sont disponibles sur le Web.

LES SCANNERS ET IMPRIMANTES 3D

Tout le monde connaît les scanners de documents 2D, maintenant intégrés dans les imprimantes. Mais le scan est aussi passé à la 3D. Il y a beaucoup de technologies disponibles, mais elles mélangent toutes de la physique et de l'algorithmique. Elles peuvent aller du scanner personnel pour petits objets aux systèmes laser complets capables de scanner une ville, en passant par ceux permettant de scanner des plantes et des arbres.

Le dual du scan 3D est l'imprimante 3D, très à la mode. Elle aussi existe à différentes échelles, de la fabrication de petits objets à celle de maisons entières. Le grand public connaît surtout celles qui impriment le plastique, mais il en existe pour des matériaux bien plus nobles : prothèses dentaires ou osseuses, pièces métalliques complexes, etc. L'impression 3D pose de très intéressants problèmes algorithmiques, par exemple pour l'impression de surplombs ou de parties creuses¹⁸.

Ne connaissant pas bien ces sujets, je n'irai pas plus loin ici.

CHAPITRE 7

L'informatisation de la médecine

Médecine et informatique ont des interactions variables suivant les domaines. Les aspects informatiques les mieux connus du grand public sont encore sous-représentés dans l'interaction des patients avec le système médical : il n'est pas souvent possible de prendre des rendez-vous par Internet, et, selon mon expérience, il est virtuellement impossible de faire suivre le dossier de son ancien généraliste au nouveau quand on déménage. La carte Vitale a mis un temps considérable pour s'imposer dans le paysage, et j'ai encore trouvé récemment des médecins qui ne la prennent pas. C'est bien sûr différent au sein des hôpitaux qui gèrent informatiquement les données des patients, y compris les données lourdes de leurs examens, mais je ne suis pas sûr qu'il soit uniformément facile de les faire passer d'un hôpital à un autre. Cependant, Internet est très utilisé en dehors des circuits médicaux officiels. De nombreux sites discutent des effets secondaires des médicaments ou de thérapies alternatives, mais lire le *Vidal* qui n'est pas fait pour l'individu standard a tendance à lui faire ressentir tous les effets secondaires d'un coup. Et il n'est vraiment pas facile de savoir où trouver des informations accessibles et fiables.

À l'opposé, certains domaines de la médecine (et de la dentisterie, dont je ne parlerai pas) sont de grands consommateurs et créateurs d'informatique. Les minuscules prothèses auditives contiennent maintenant des ordinateurs et des algorithmes de traitement du signal sophistiqués. J'y suis maintenant hélas condamné, mais celles que j'utilise ont deux programmes, dont l'un transpose d'une octave vers

le bas les aigus au-dessus de 4 kilohertz ; chez moi, dans la garrigue provençale, je peux choisir d'entendre les cigales ou pas ! Les pacemakers et pompes à insuline se perfectionnent de plus en plus et deviennent de vrais systèmes informatiques embarqués (apparemment pleins de trous de sécurité informatique, voir le [chapitre 9](#)). Les prothèses de hanche et de genou vont être fabriquées de façon personnalisée, par impression 3D. Dans la suite, je parlerai surtout de l'imagerie médicale, qui est devenue un outil fondamental pour le diagnostic, la chirurgie et l'étude du corps humain, et un peu de l'analyse des très grandes données et des outils d'assistance au diagnostic. Autrefois réduits à des systèmes experts assez peu performants, ces outils évoluent très vite grâce aux méthodes d'apprentissage automatique. Voyons cela avec plus de détails.

L'imagerie médicale

L'imagerie médicale est née le jour même de la création de la radiographie par Röntgen : le 22 décembre 1895 la première « radio », immensément célèbre, montrait clairement les os de son épouse. La radiographie a bouleversé le diagnostic et le suivi de nombreuses maladies. Longtemps de nature purement analogique, elle a aussi largement contribué à la fortune de Kodak, le plus grand fabricant de films. Elle n'est devenue massivement numérique qu'à la fin des années 1990, grâce à une image plus fine que celle des films et un coût inférieur, cependant sans vraie révolution. Mais bien d'autres méthodes d'imagerie sont apparues depuis la fin des années 1970, époque où de nouveaux procédés physiques sont apparus et où physique et informatique ont commencé à se marier de plus en plus, les deux domaines progressant de concert. Les imageurs modernes sont des machines faisant appel à une physique très subtile et largement inconnue du grand public, et c'est exactement la même chose pour leurs algorithmes. Je ne saurais trop recommander la lecture du magnifique livre *Le Cerveau de cristal* ([Le Bihan, 2012](#)) écrit par Denis Le Bihan, pionnier de l'IRM et fondateur de l'unité multidisciplinaire de recherche NeuroSpin située au CEA à Saclay, où une bande de chercheurs de tout premier plan mondial utilise les appareils les plus puissants du monde pour sonder nos cerveaux afin de mieux comprendre leur fonctionnement et leurs maladies ([Dehaene, 2007](#) ; [Dehaene, 2014](#)) . Par manque de place et surtout par incompetence, je ne parlerai pas de tous les modes d'imagerie, et j'omettrai par exemple l'échographie que je connais bien moins mais qui est aussi très informatisée. Je conseille vivement le visionnage des cours de Nicholas Ayache sur la chaire Informatique et sciences numériques 2013-2014 du

Collège de France, accompagnés de séminaires par des chercheurs en informatique et des médecins et chirurgiens de premier plan ([Ayache, 2014](#)).

L'informatisation de l'imagerie médicale

L'entrée de l'informatique s'est faite dans les années 1970 lors de l'introduction de la TEP (tomographie par émission de positrons), PET-scan en anglais, dont le principe physique avait été introduit dans les années 1950, puis de la TDM (tomodensitométrie), CT-scan en anglais, réalisée par les scanners à rayons X. Ayant des performances complémentaires, ces deux modes d'imagerie sont maintenant couplés dans la plupart des appareils. Les deux ne produisent plus directement des images, mais des séquences de mesures qu'il faut transformer en images 2D ou 3D à l'aide d'algorithmes et de programmes informatiques. Par exemple, le scanner à rayons X, qui utilise le même principe de base que la radiographie classique, produit une séquence complexe de mesures du rayonnement ayant traversé le corps au cours de mouvements conjoints de rotation et de translation de l'émetteur et des détecteurs. La reconstruction se fait en utilisant un vieux théorème de Johann Radon démontré en 1917. Cette reconstruction relativement simple peut donner des résultats interprétables, allant d'images de coupes successives à une reconstruction 3D complète. Des injections de produits de contraste adéquats permettent de réaliser diverses observations à partir du même appareil : os, organes, vaisseaux sanguins, etc. Grâce à sa simplicité et à sa remarquable résolution spatiale, le scanner est devenu un des principaux appareils de diagnostic.

Voici quelques images qui m'ont été fournies par mon ami Francis Besse, cardiologue au Centre cardiologique du Nord (CCN) à Saint-Denis (93). Je le remercie pour la variété et la qualité de ces images, toutes issues d'examens standard. Elles en disent plus que de longs discours.

La figure 41 montre à gauche une reconstitution 3D de l'aorte et à droite une coupe de cette aorte. Les images sont obtenues par CT-scan en mettant en évidence les vaisseaux par un produit de scintigraphie iodé.

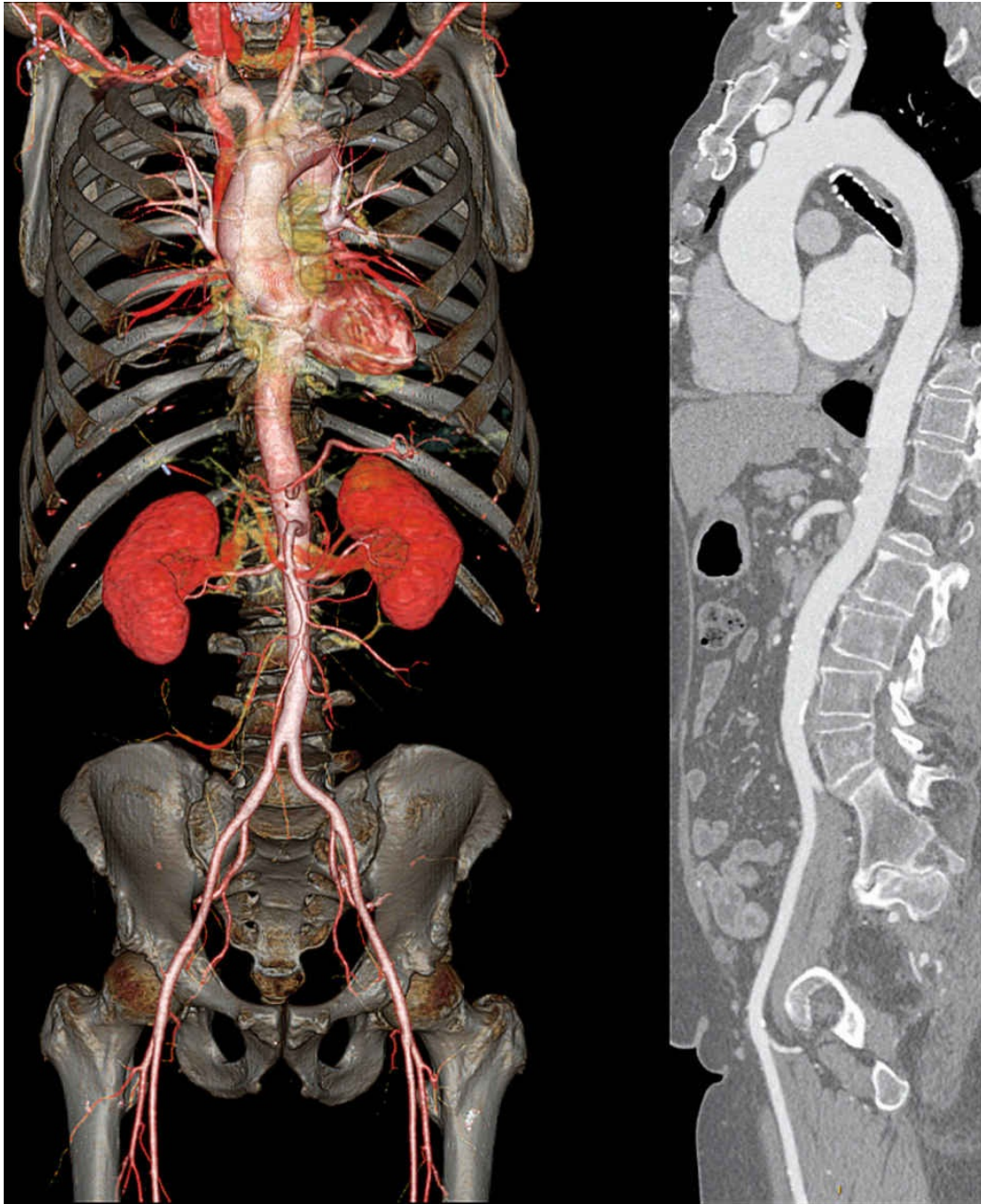


Figure 41. Image 3D et coupe de l'aorte (source CCN).

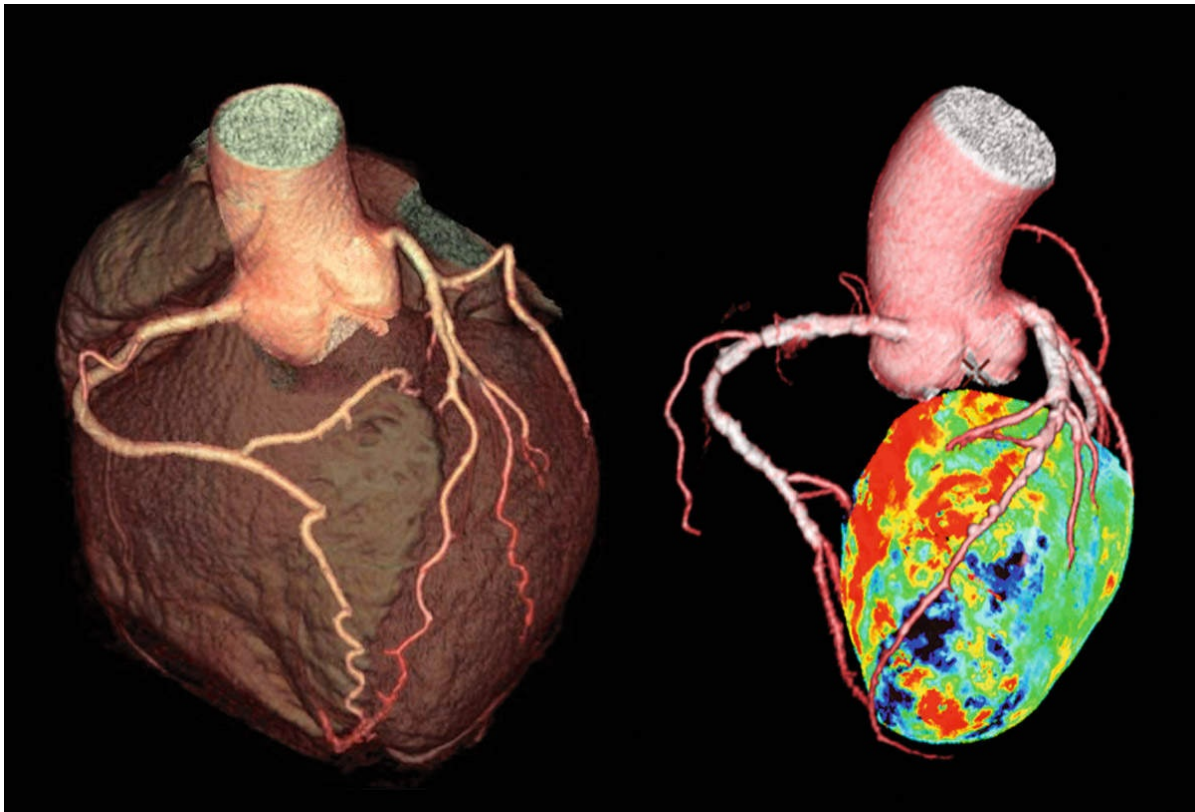


Figure 42. Images cardiaques (source CCN).

La figure 42 montre deux scans de la racine de l'aorte et des artères coronaires, avec à droite la scintigraphie du myocarde du ventricule gauche dont les fausses couleurs reflètent la perfusion du muscle cardiaque.

La figure 43 montre à gauche une reconstitution 3D d'un bras par CT-scan et à droite un long stent en forme de ressort posé dans le bras.

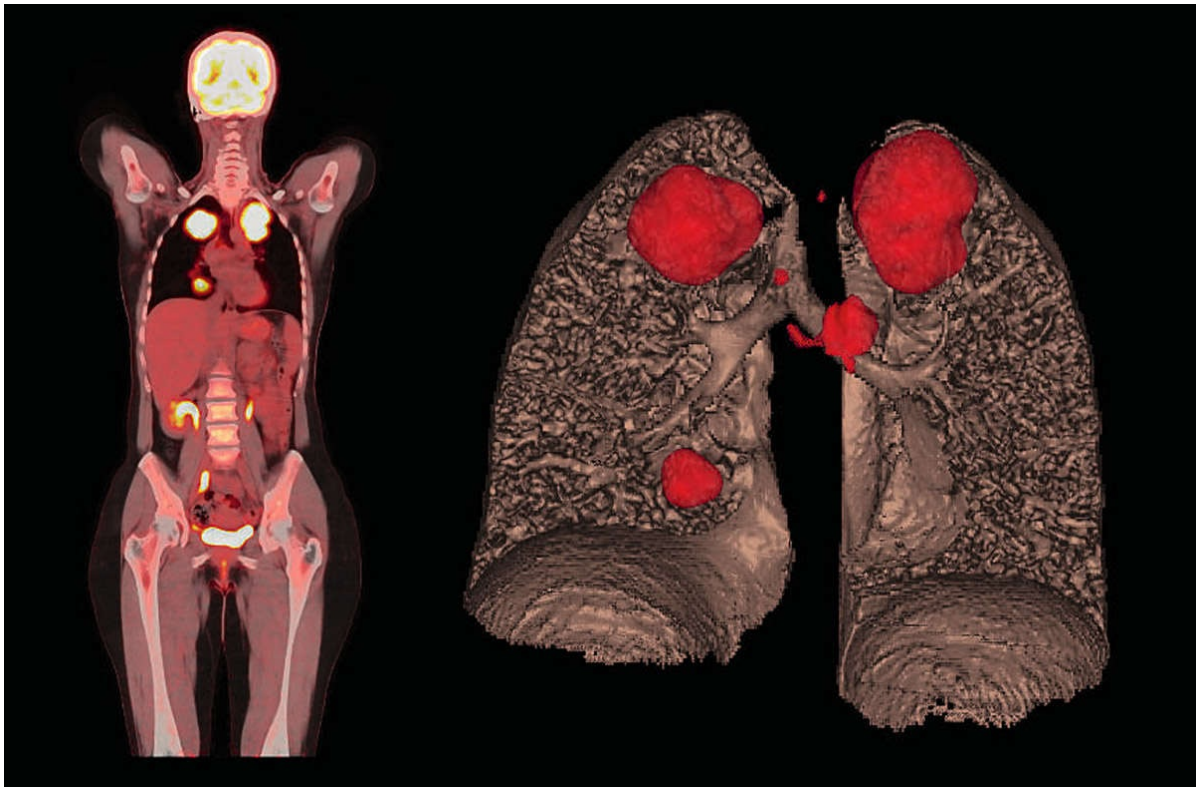


Figure 44. Lymphome pulmonaire (source CCN).

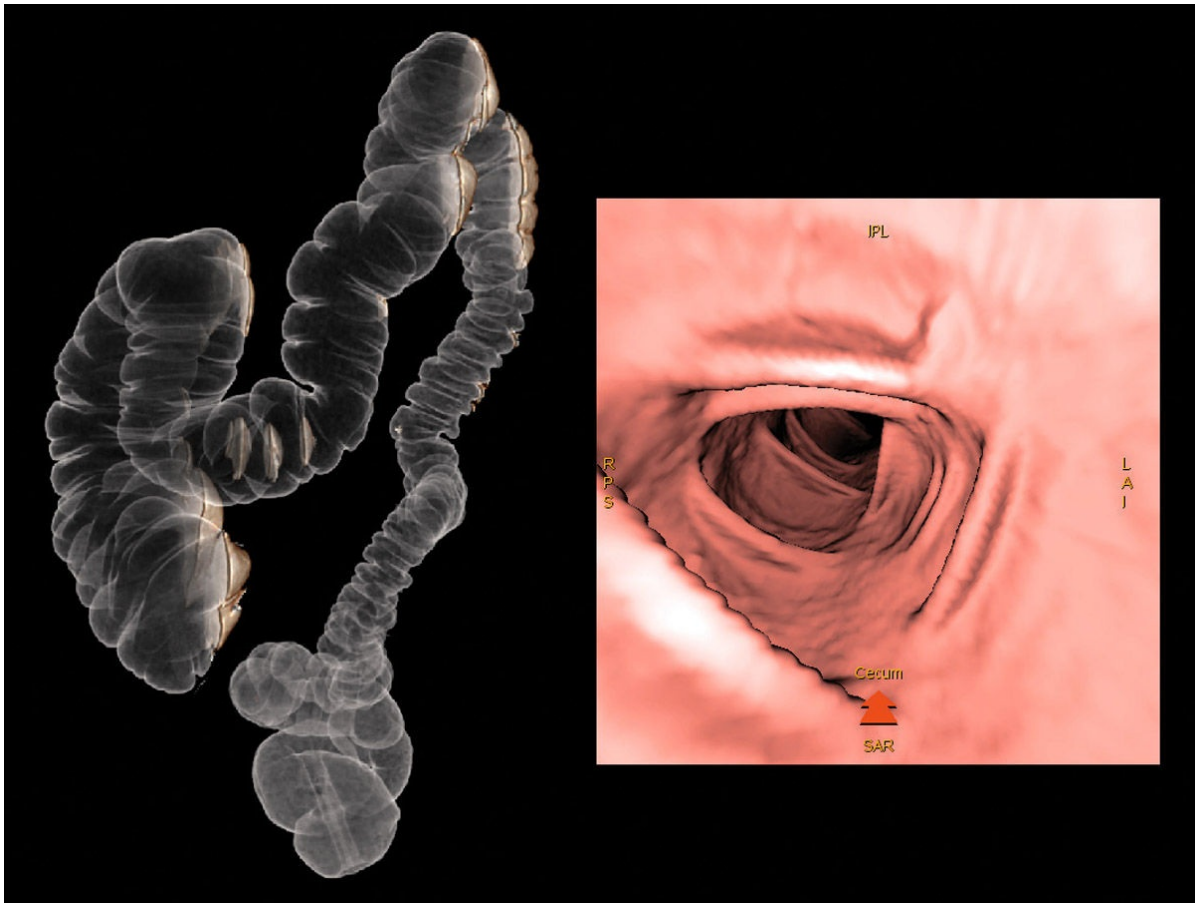


Figure 45. Coloscopie virtuelle (source CCN).

La figure 44 montre un lymphome pulmonaire, par recombinaison 3D de CT-scan et PET-scan.

Enfin, la figure 45 montre une coloscopie virtuelle, qui se fait sous imagerie en ayant au préalable gonflé le côlon avec de l'air. Cette coloscopie est moins intrusive que la coloscopie classique, mais ne permet bien sûr pas de faire des biopsies. Les deux images sont obtenues par reconstruction 3D, celle de droite ayant été prise lors d'un parcours virtuel dans le côlon.

L'IRM, IMAGERIE PAR RÉSONANCE MAGNÉTIQUE NUCLÉAIRE

Le rôle de l'informatique s'est encore accru avec l'invention de l'IRM (imagerie par résonance magnétique nucléaire, le mot « nucléaire » ayant été omis pour ne pas faire peur au public). L'IRM repose sur une manipulation fine du moment magnétique des nucléons des atomes à l'aide de très forts champs magnétiques et d'émissions précises de radiofréquences (voir [Le Bihan, 2012](#), pour une belle explication). Elle a l'avantage de ne pas du tout irradier le patient. Suivant ce qu'on cherche, ce principe unique se décline techniquement sous plusieurs formes aussi complémentaires que fascinantes : l'angio-IRM visualise les artères ; l'IRM fonctionnelle permet de visualiser l'activité du cerveau à travers l'apport sanguin à ses parties ; l'IRM de diffusion, inventée par Denis Le Bihan, permet de mettre en évidence l'anisotropie des mouvements des molécules d'eau, par exemple pour visualiser les trajets des axones des neurones, et s'invite aussi dans l'étude du fonctionnement dynamique du cerveau ([Dehaene, 2007](#) ; [Dehaene, 2014](#) ; [Le Bihan, 2012](#)).

Un appareil IRM est une prouesse de physique et d'informatique. Ce qu'il mesure est la vitesse de relaxation (du retour à la normale) du moment magnétique d'un noyau atomique plongé dans un champ magnétique intense et modifié par une impulsion de fréquence radio. Ce noyau peut être le simple proton d'un des atomes d'hydrogène d'une molécule d'eau, ou encore le noyau de l'isotope 13 de l'oxygène porté par l'hémoglobine pour l'IRM fonctionnelle (il faut que le noyau ait un nombre impair de nucléons, d'où le besoin de cet isotope). Le premier problème est que la modification du moment magnétique est minime, et que la mesure ne peut donc pas être très précise. Cela impose une approche statistique fine, réalisable uniquement par logiciel. D'autre part, on ne se contente pas d'une seule mesure globale, bien au contraire : pour chaque point, on utilise plusieurs angles successifs d'action des radiofréquences, et on balaye une quantité considérable de points. Cela produit une quantité considérable de données, et donc une quantité considérable de calculs très complexes mais à faire très vite pour synthétiser les images 2D ou 3D en fausses couleurs qui seront montrées au médecin. Les algorithmes de l'IRM sont particulièrement délicats, surtout du fait qu'ils ne sont pas carrés comme ceux de l'informatique classique. Un mauvais algorithme peut introduire des artefacts conduisant à une mauvaise représentation des zones concernées et donc à un diagnostic erroné. La non-détection d'une lésion qui existe vraiment peut être dramatique (faux négatif), la création artificielle d'une lésion qui n'existe pas chez le

patient peut l'être aussi (faux positif), par exemple en conduisant à des mutilations inutiles. C'est pour cela que de nombreuses et fortes équipes académiques et industrielles de recherche en informatique médicale travaillent avec les médecins, soit pour améliorer les algorithmes contenus dans les appareils, soit pour en construire de nouveaux utilisables hors ligne si on garde les mesures brutes des appareils comme on garde l'image source crue (*raw*) pour la photographie numérique experte (voir le [chapitre précédent](#)).

DES IMAGES INDIVIDUELLES, LA FUSION D'IMAGES

Au [chapitre 6](#), j'ai montré l'importance de la fusion d'images en photographie numérique. Elle est fondamentale pour améliorer les scènes de trop forte dynamique, réduire le bruit ou augmenter artificiellement la profondeur de champ. Selon mon leitmotiv, la fusion permet d'obtenir des résultats inaccessibles à la seule physique. On va beaucoup plus loin en imagerie médicale. Une fusion extrêmement utile est le passage à la 3D d'une série d'images prises en TEP, au scanner ou à l'IRM. Cette fusion fournit une image dans laquelle le médecin peut se promener virtuellement à loisir à l'aide des logiciels de visualisation 3D ou même des lunettes et salles d'immersion 3D, ce qui augmente ses capacités d'observation en le libérant de la mise en relation incessante d'images 2D. Ici, la reconstruction 3D se fait en fonction des caractéristiques physiques de l'appareil avec des algorithmes relativement classiques.

La fusion peut aussi s'appliquer pour superposer les différentes modalités d'IRM dans une seule image, comme illustré par la figure 46. Ici, il ne s'agit plus d'une simple amélioration : fusionner informatiquement des images différentes mais qu'on peut mettre précisément en correspondance géométrique permet au médecin d'agréger mentalement toutes leurs informations bien plus facilement que pour une image unique, donc de voir des choses qui ne seraient pas facilement observables autrement. Il est aussi possible de fusionner entre elles des images PET-scan, CT-scan, IRM, échographiques, etc. De plus, si l'on produit une image globale 3D, comme en bas de la figure 46, le médecin peut se promener virtuellement dedans en allumant ou non les différentes modalités d'imagerie qui l'intéressent.

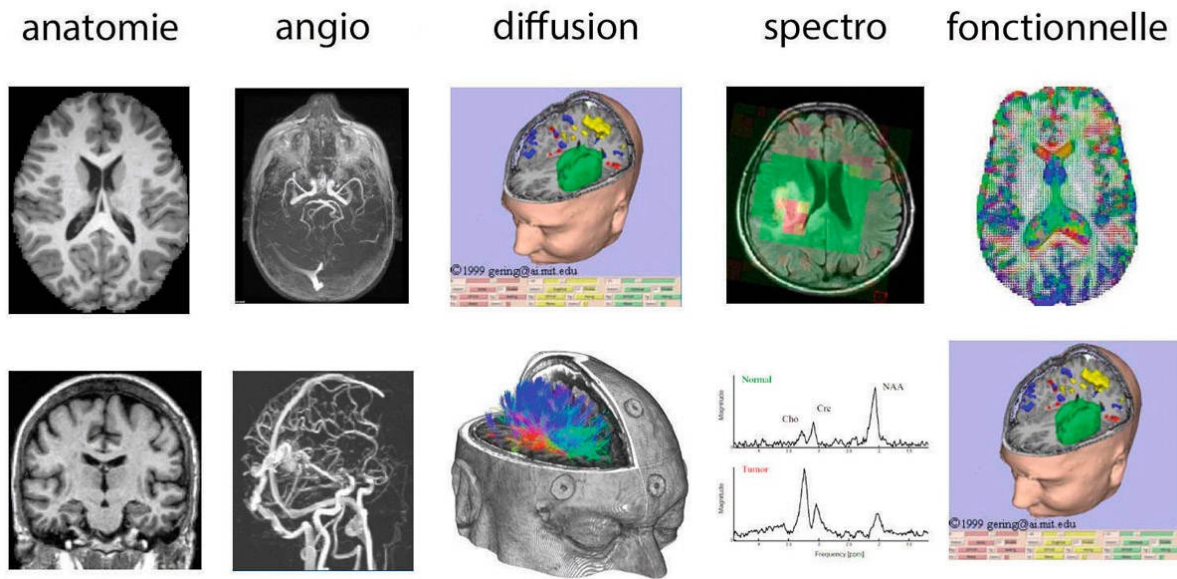


Figure 46. Superposition d'images médicales (source D. Le Bihan/N. Ayache).

La fusion d'images est aussi à la base d'autres modalités d'imagerie, comme la *biopsie optique* promue par la société française Mauna Kea Technologies¹. L'outil classique de biopsie destiné à prélever un échantillon des tissus pour l'amener sous un microscope est remplacé par une sonde optique composée de dizaines de milliers de fibres optiques, dont les images individuelles sont constamment fusionnées entre elles et avec les précédentes quand la sonde se déplace. On peut ainsi disposer d'une image précise au niveau cellulaire, ce qui peut permettre de réduire, voire éviter, les biopsies physiques traditionnelles.

LA QUANTIFICATION DES OBJETS D'INTÉRÊT

Une autre opération que la fusion peut aider est la quantification des objets d'intérêt dans les images, toujours pénible à réaliser la main. Par exemple, l'image en bas à droite de la figure 46 montre clairement une tumeur du cerveau, dont l'image peut résulter de l'utilisation simultanée de plusieurs modalités. Mesurer son volume est évidemment important pour surveiller sa croissance. C'est une opération quasi impossible à faire à la main, mais qui peut s'informatiser. La mesure automatique de tailles, courbures, etc., se généralise actuellement.

L'anatomie numérique

Directement liée à l'image, l'anatomie est également révolutionnée par l'informatique. Les sublimes planches 2D des livres d'anatomie et les maquettes d'articulations ne sont bien sûr pas mises au rebut, et les trésors qu'elles renferment serviront encore longtemps. Dans les années 1990, elles ont été magnifiées par les milliers de tranches découpées sur des humains congelés, disponibles dans les bases de données du Visible Human Project². Mais l'anatomie classique est souvent limitée par le fait qu'elle est le plus souvent réalisée à partir d'organes morts, ce qui fait perdre énormément d'informations – c'est évident pour le cerveau, qu'on découpe rarement *in situ*. L'imagerie informatisée permet maintenant de faire des choses impensables avant, comme étudier finement l'anatomie des organes en fonctionnement.

Regardons d'abord le cas du cerveau. La figure 47 montre une superbe image des faisceaux de fibres nerveuses qui connectent entre elles les zones de calcul du cerveau : c'est ce qu'on appelle la substance blanche. Une image de cette qualité ne peut pas résulter d'une dissection : elle est obtenue par une IRM de diffusion sur un sujet vivant. Elle n'est en fait que la projection sur un plan d'une image 3D. On imagine sans peine la masse de mesures et de calcul qu'il a fallu pour produire cette image. Un point particulièrement délicat est de bien détecter les croisements de fibres, car les mesures ne montrent qu'une direction dans ce cas ([Le Bihan, 2012](#)). Les meilleurs algorithmes de traitement d'images savent le faire.

L'imagerie, le traitement de données massives par approches statistiques et la visualisation 3D permettent aussi d'aborder un nouveau chapitre de l'anatomie :

l'anatomie statistique. La figure 48, extraite d'un film en 3D, montre la variabilité des lignes sulcales, lignes géométriques associées aux sillons qui permettent de bien caractériser la forme du cerveau. La position de ces lignes varie d'une personne à une autre, et la variation est plus ou moins forte selon leur position globale. L'image, en fausses couleurs, montre en rouge les endroits où la variabilité est forte, en vert les endroits où elle est faible, et en jaune les endroits intermédiaires. Cela permet de remplacer le cerveau type des coupes par un cerveau statistique. L'anatomie informatique peut également passer à la 4D, c'est-à-dire inclure les mouvements, voire en 4D statistique. De grands programmes sont actuellement consacrés à cela pour le cœur. Une application majeure de toutes ces approches est la possibilité de personnalisation des atlas anatomiques en projetant l'atlas sur les mesures d'un cerveau ou d'un cœur particulier.

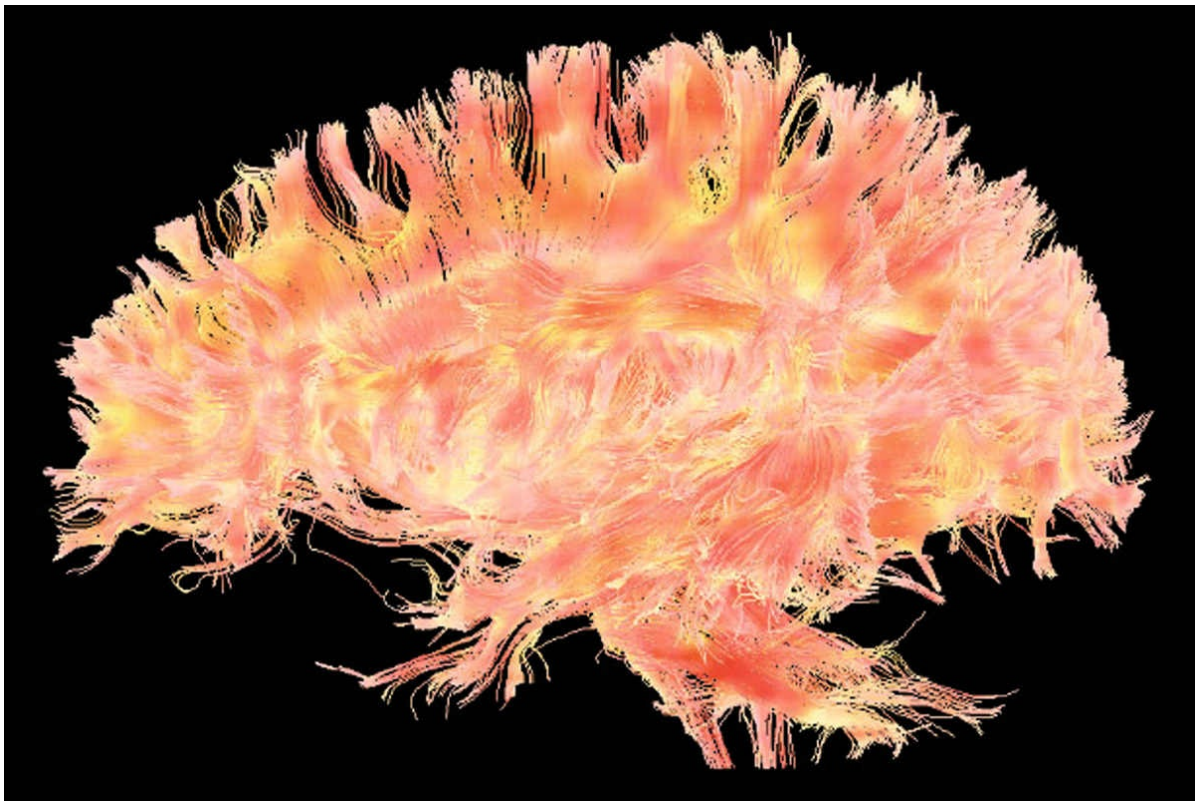


Figure 47. Les réseaux d'axones de la substance blanche en 3D (source CCN).

Enfin, l'anatomie informatisée est aussi précieuse pour l'enseignement, même chez les enfants. De nombreuses applications sur tablettes permettent de visualiser de façon interactive et en 3D les principaux organes du corps humain, ce qui permet de comprendre beaucoup plus vite et bien mieux à quoi ressemble son intérieur.

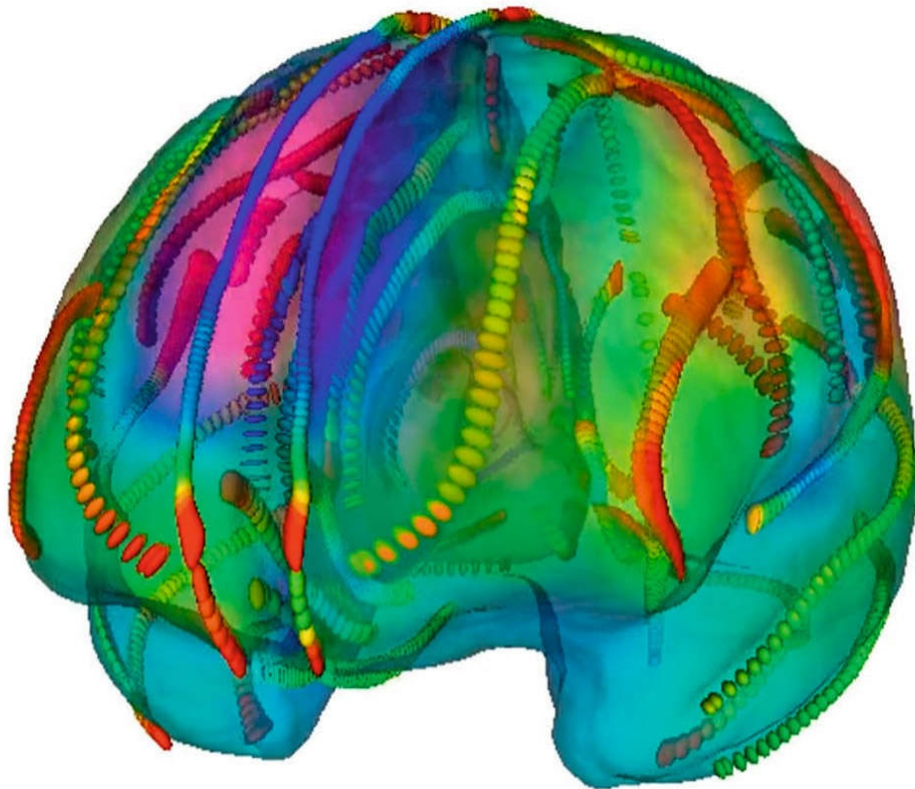


Figure 48. Anatomie statistique des lignes sulcales (source Inria Asclepios).

La radiologie interventionnelle

L'imagerie ne sert pas qu'au diagnostic et à la préparation des interventions. Certaines opérations moins agressives et souvent plus efficaces que les opérations chirurgicales classiques se déroulent directement sous imagerie. Les radiologues interventionnels qui les pratiquent débouchent les artères en cas de sténose ou d'AVC, y insèrent des stents, colmatent les anévrismes ou brûlent des tumeurs directement sous imagerie temps réel, CT-scan/PET-scan par exemple. C'est cela qui a permis la chirurgie mini-invasive, qui remplace les grandes incisions par de petites fentes où l'on insère une sonde, par exemple dans la cuisse ou dans le bras, pour la diriger ensuite au bon endroit dans le cœur ou le cerveau. Par rapport à la chirurgie classique, les lésions opératoires sont minimales et la récupération plus rapide.

Les radiologues interventionnels peuvent préparer les opérations sur des vues 3D préalablement acquises des organes du patient, et même éviter l'irradiation d'eux-mêmes et du patient pendant l'opération en recalant leur action sur ces mêmes images, à condition d'équiper leurs instruments externes (trocarts) de capteurs 3D pour analyser tous les mouvements qu'ils leur transmettent.

La figure 49 illustre l'environnement de travail d'un radiologue interventionnel, qui est très largement informatisé. L'opérateur va consolider avec une vis métallique le bassin fracturé. Il guide son geste en CT-scan (écran de gauche). Il s'aide d'un système de positionnement très précis qui repère la trajectoire de la vis et la matérialise en temps réel sur l'écran de droite.



Figure 49. Intervention sous imagerie (source CCN).

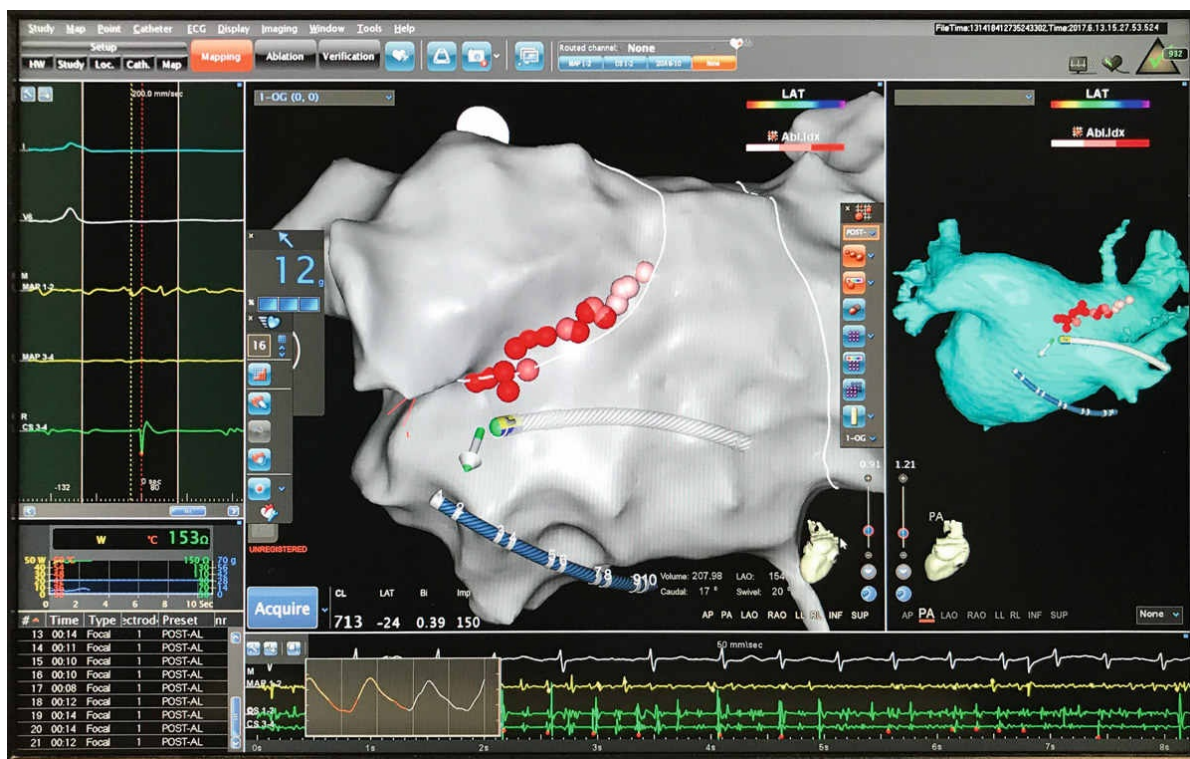


Figure 50. Écran de radiologie interventionnel (source CCN).

La figure 50 montre un écran de travail pour une opération par radiofréquence d'une oreillette cardiaque. Sur cette image on voit à droite en bleu une image 3D de l'oreillette gauche reconstruite à partir d'un scanner et à gauche en gris une image de la même oreillette reconstruite à partir du trajet endocavitaire parcouru par un capteur magnétique, localisé par un système placé sur le patient pendant l'examen. Le but est de repérer l'origine des veines pulmonaires et d'y thermocoaguler les fibres nerveuses à l'origine d'un trouble du rythme cardiaque. Les zones traitées sont matérialisées par les points rouges.

Les radiologues interventionnels peuvent tirer grand profit de la *réalité augmentée*, en superposant des images synthétiques à l'intérieur même du corps. La figure 51 montre une opération particulièrement délicate, la destruction par radiofréquence d'une tumeur du foie. Pour aider son parcours dans l'abdomen, le radiologue interventionnel se guide sur une projection virtuelle 3D dans l'abdomen du patient de son foie préalablement imagé, la projection devant suivre les mouvements de sa respiration.

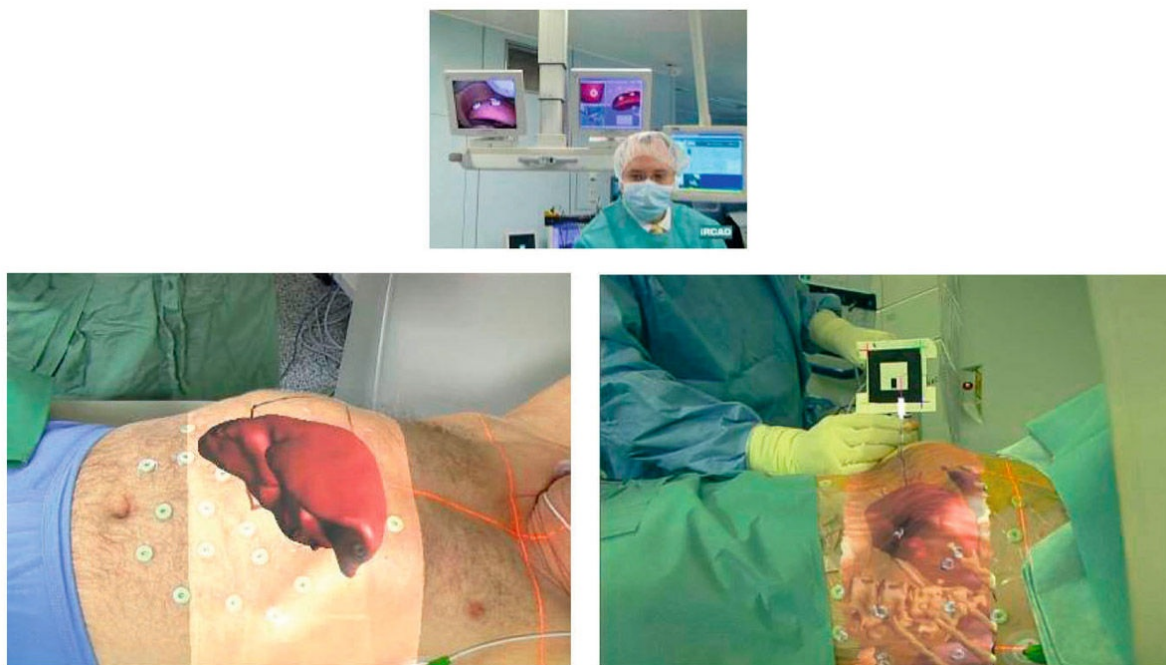


Figure 51. Opération du foie sous réalité augmentée (source Ircad/Inria).

LA SIMULATION D'OPÉRATIONS

Pour certaines interventions délicates, une simulation préalable peut être très utile. Comme nous l'avons vu au [chapitre 3](#), la simulation remplace la matière, l'énergie et les ondes du phénomène par le calcul sur l'information, lui-même déterminé par une modélisation mathématique des phénomènes dynamiques et l'implémentation d'algorithmes et de logiciels réalisant effectivement cette modélisation. La figure 52 montre une malformation évidente du ventricule du cœur chez un enfant, appelée tétralogie de Fallot. Cette malformation peut se corriger en coupant et en suturant au bon endroit ce ventricule. Mais comment déterminer ce bon endroit ? Maintenant qu'on comprend mieux l'anatomie et le fonctionnement 4D (espace et temps) du cœur, on peut simuler informatiquement ce cœur avec une précision suffisante pour précalculer le rendement de divers choix opératoires et choisir le meilleur. L'image de la figure 52 est extraite d'un film réalisé en 2008 par le projet Inria Asclepios dirigé par Nicholas Ayache, qui montre le fonctionnement temporel du cœur avant et après l'opération. Elle est en fausses couleurs : le rouge pour les endroits où la pression est bonne, le bleu pour ceux où elle est trop faible, le

vert pour l'intermédiaire. Un autre endroit où la simulation du cœur devient importante est la compréhension et le traitement chirurgical des infarctus et fibrillations, liés à des perturbations de la propagation électriques provoquées par les lésions³. Bien sûr, ces simulations n'ont rien de simple, car beaucoup de phénomènes interfèrent dans le cœur : propagation des signaux électriques qui vont déclencher la contraction, modification géométrique des fibres lors de cette contraction, mouvements du sang, ouverture et fermeture des valves et chocs associés, etc.

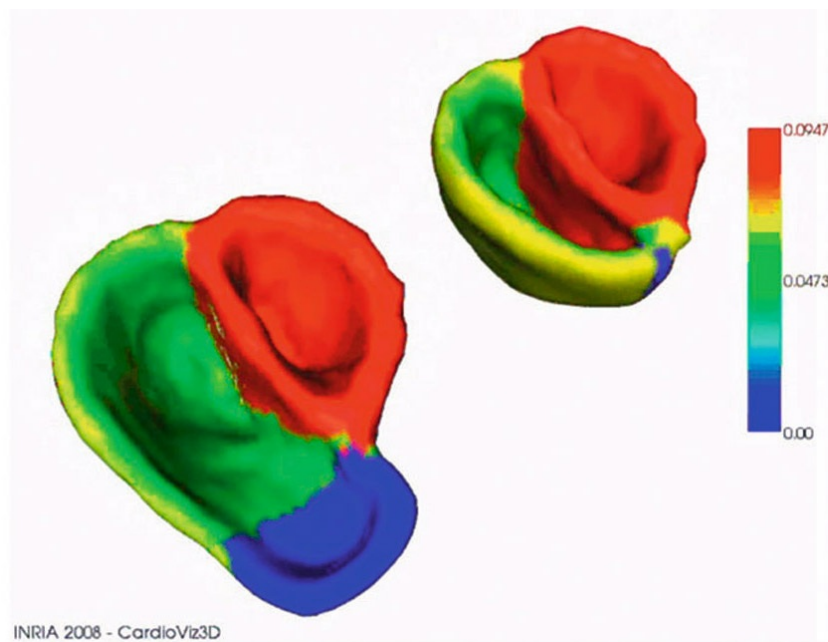


Figure 52. Correction d'une malformation cardiaque (source Inria Asclepios).

La simulation et la réalité virtuelle servent aussi à l'entraînement des chirurgiens pour des opérations réalisées par des robots chirurgiens. Cela se fait grâce à des instruments simulant d'une part l'action sur le patient et d'autre part les réactions sur les commandes du chirurgien à l'aide de retour d'efforts virtuels motorisés. Les robots chirurgiens sont eux-mêmes de joyeux mélanges de mécanique et d'informatique sophistiquées, que je ne connais pas assez pour en parler correctement ici. Mais on ne s'étonnera plus que le célèbre chirurgien de l'IHU de Strasbourg Jacques Marescaux compare l'équipement d'une salle d'opération moderne à celui d'un cockpit d'Airbus A380⁴ !

Le projet Shacra dirigé par Stéphane Cotin à l’Inria Lille contribue à l’usage de la simulation numérique pour l’entraînement des médecins et chirurgiens et la préparation des opérations. Il est en particulier très investi sur le sujet de la formation des médecins pour l’opération de la cataracte dans les pays en voie de développement, ce qui est très important car la cataracte est très fréquente et très destructrice chez les enfants en Afrique. La bibliothèque logicielle SOFA⁵ développée à l’Inria par le groupe de Stéphane Cotin est très utilisée dans ce domaine.

L'entrée en force de l'apprentissage automatique en médecine

Pas mal d'énergie a été dépensée il y a quelque temps pour fabriquer une assistance informatique au diagnostic à base de systèmes experts, mais sans vrai succès. La raison est probablement que ces systèmes sont fondés sur l'application systématique de règles soit logiques soit empiriques, alors que les règles du diagnostic médical ne sont pas vraiment connues, si tant est qu'il en existe.

Mais les techniques modernes d'apprentissage, qui ne sont plus fondées sur des règles de ce type, commencent à donner des résultats étonnants. L'apprentissage automatique est utilisé de façon routinière en imagerie, où il permet de reconnaître les personnes, les animaux ou les objets dans les photographies. Il obtient aussi de grands succès préliminaires en imagerie médicale, où il sera probablement mis en service bientôt.

Un bon exemple récent en dermatologie concerne la détection des tumeurs suspectes à partir de photos de la peau. C'est très important car un mélanome détecté tôt a de fortes chances de guérison, ce qui n'est plus du tout vrai s'il est détecté tard. Une expérience édifiante a eu lieu à Stanford. Un réseau d'apprentissage profond a été nourri par une grande base de données de 130 000 images de lésions de la peau représentant 2 000 maladies. Il a ensuite reçu à étudier 370 images de peau pour poser ses diagnostics, en parallèle avec un panel de 21 dermatologues aguerris⁶. Les résultats ont été vraiment comparables, avec un taux de succès de 91 %, le critère de succès étant de reconnaître soit qu'il n'y a rien, soit qu'il faut continuer par une

biopsie ou un traitement. Un des avantages est que cet algorithme pourrait à terme travailler sur des photos prises par des téléphones portables, voire être lui-même implémenté sur ces téléphones.

Un autre exemple concerne la détection des tumeurs du sein, qui touchent une femme sur huit et qu'un dépistage précoce permet de traiter avec de bien meilleurs résultats. Un défi mondial nommé The Digital Mammography DREAM Challenge a été lancé aux États-Unis pour comparer les progrès des algorithmes de détection à partir d'une base de données de 640 000 images provenant de 86 000 patients. En 2017, pas moins de 1 209 groupes de recherche ont participé au défi, et la victoire est revenue à la société française Therapixel⁷, une start-up de l'Inria fondée en 2013, qui a réussi à améliorer de 5 % le taux de faux positifs par rapport à l'état de l'art en utilisant des algorithmes d'apprentissage profond (*deep learning*). Malgré des moyens de calcul limités, le résultat a été très proche de ceux des meilleurs radiologues.

Bien d'autres applications majeures de l'apprentissage et plus généralement des méthodes statistiques sont en cours de construction et d'évaluation, en exploitant soit les données des très grandes bases de données comme celle de l'assurance maladie française, soit même les discussions sur Internet concernant par exemple la propagation des épidémies ou les effets secondaires des médicaments. N'étant vraiment pas compétent dans ces sujets, je n'en parlerai pas davantage. Ce n'est pas bien grave car ils ont pris le devant de la scène, au point quelquefois d'éclipser un peu trop les autres. Vous en entendrez donc forcément parler dans vos médias préférés. Mais attention à bien séparer les vraies informations des rêves éveillés, toujours bien plus attirants à première vue mais bien moins intéressants en pratique. Le sujet est suffisamment sérieux, riche et important pour pouvoir s'en passer.

Ne jamais oublier le patient

J'ai beaucoup parlé de technique dans ce chapitre. Mais il ne faut pas oublier un point essentiel : la médecine est loin de se résumer à la technique, car un patient n'est pas qu'un ensemble d'organes montés dans un corps bien compartimenté. J'ai assez d'expérience personnelle comme patient pour savoir qu'un bon médecin a besoin de me voir, d'interagir avec moi, de comprendre comment les différentes parties de mon corps et de mon esprit résonnent entre elles, et pourquoi le soin de la seule partie visiblement en cause ce jour-là peut ne pas suffire. J'ai passé récemment une radio et une IRM d'un genou en franche délicatesse, et j'ai été désagréablement surpris de recevoir un compte rendu froid sans voir personne (malgré de sérieux dépassements d'honoraires pour l'IRM). Il est possible que la radiologie standard devienne largement automatisée, comme le sont déjà les analyses de sang ou d'urine. Mais j'espère que je ne serai pas convoqué à l'avenir par un robot chirurgien qui m'opérera sans même s'intéresser à mon mode de vie ! La médecine est une pratique faite par des hommes pour des hommes, pas par des machines pour des ablations ou des prothèses. Ne confondons pas l'outil et la finalité. De ce point de vue, la notion moderne de « médecine personnalisée » doit être regardée de près : elle ne peut pas être que technique, et il ne faut jamais oublier que personnalisée dérive de personne.

CHAPITRE 8

Vers une informatisation massive des sciences

Le but de ce chapitre est de montrer à l'aide de quelques exemples comment la pensée informatique s'introduit en grand dans les sciences, avec des impacts déjà considérables qui vont aller en s'amplifiant rapidement. Nous discuterons des sciences naturelles et des mathématiques. Mais, avant d'entrer dans le vif du sujet, il faut faire un détour par une brève analyse des rôles respectifs des instruments et des mathématiques dans les sciences naturelles.

LE RÔLE FONDAMENTAL DES INSTRUMENTS DANS LES SCIENCES NATURELLES

Puisqu'une grande partie des phénomènes intéressant les chercheurs des sciences naturelles ne sont pas visibles à l'œil nu, l'instrumentation y est essentielle : les lunettes astronomiques puis télescopes et radiotélescopes ont permis de scruter le ciel, les microscopes optiques puis électroniques ont permis de scruter l'infiniment petit, les chronomètres puis horloges atomiques ont permis de mesurer finement le temps, etc. Jusque dans les années 1970, presque tous les instruments ont reposé sur les principes de la physique classique, et donc sur le triangle matière-énergie-ondes. Le recours à l'informatique est devenu nécessaire lorsque les instruments ont été

capables de fournir des masses de données bien trop grandes pour les possibilités de stockage et de calcul des hommes et du papier. Il ne s'agit plus seulement d'agréger quelques données et de les présenter sous une forme facilement lisible, image ou tableau de chiffres. Il faut maintenant les prétraiter d'une façon qui peut être très complexe. Par exemple, un accélérateur de particules produit une énorme quantité de mesures et d'images dont très peu ont de l'intérêt. Il faut supprimer celles qui ne montrent rien, rechercher automatiquement lesquelles présentent un phénomène espéré ou au contraire inattendu, puis en agréger beaucoup pour réduire le bruit inhérent à ces mesures très fines et permettre d'effectuer les bonnes déductions physiques.

À la toute fin du xx^e siècle, comme pour l'imagerie médicale au chapitre précédent, on est passé de l'informatique de service à une informatique faisant complètement corps avec les instruments, voire à une instrumentation mixte où la physique et l'informatique sont au même degré d'importance. Prenons un exemple : nous avons vu au [chapitre 6](#) qu'en photographie les algorithmes permettent de corriger les défauts des objectifs, et donc de simplifier et d'alléger ces derniers puisqu'ils n'ont plus besoin de minimiser eux-mêmes ces défauts. La même interaction constante entre physique et informatique est à l'œuvre en astronomie, où elle est essentielle pour les instruments optiques des satellites dont le poids est crucial ou pour les capteurs des télescopes terrestres dont la taille est gigantesque comparée à celle d'un appareil photo. Souvenons-nous par exemple que le fameux télescope spatial Hubble avait été lancé avec un miroir taillé avec une extrême précision, mais dont la forme n'était malheureusement pas la bonne ; du coup, ses images étaient floues. Ce défaut majeur a d'abord pu être partiellement corrigé algorithmiquement sur terre, ce qui a permis de bien améliorer ses images. Mais la qualité actuelle n'a pu être obtenue qu'en lui ajoutant dans l'espace des « lunettes correctrices », exploit extraordinaire mais vraiment coûteux ; cela n'empêche pas d'utiliser toujours des algorithmes pour obtenir à terre des images encore plus fines. Une évolution similaire se produit aussi en biologie pour l'imagerie cellulaire ([Lecuit, 2017](#)) ainsi que pour les séquenceurs d'ADN, qui mélangent intimement physique, biologie et informatique, et qui progressent si vite qu'on peut se demander s'ils ne vont pas bientôt fournir plus de données que nous ne saurons fabriquer de disques pour les stocker.

LE RÔLE CENTRAL DES MATHÉMATIQUES DANS LES SCIENCES PHYSIQUES

On dit souvent que le plus important progrès des sciences physiques a été l'idée de ne pas seulement observer la nature grâce aux instruments précités, mais de la *modéliser abstraitement* à l'aide d'équations mathématiques, de façon à pouvoir explorer autant les modèles eux-mêmes que la nature afin d'en déduire de nouvelles lois et de nouvelles expériences à faire. En physique des particules ou en mécanique quantique, les calculs précèdent et dirigent souvent l'expérimentation. Quand cela a-t-il commencé ? Il est généralement reconnu que les sciences mathématisées modernes sont nées avec Galilée, qui a écrit une célèbre phrase ici simplifiée :

La nature est un livre écrit en langage mathématique¹.

Cette phrase illustre effectivement les gigantesques progrès ultérieurs des sciences physiques, dans lesquelles s'inscrit bien sûr l'astronomie que pratiquait Galilée. Des savants comme Newton, Boltzmann, Maxwell, Einstein, Schrödinger et bien d'autres ont utilisé et souvent inventé des mathématiques profondes, qui les ont conduits à des succès théoriques et pratiques prodigieux. La conjonction d'expériences de pensée et de calculs mathématiques les a plusieurs fois conduits à des prédictions de faits ou lois contre-intuitifs, qui n'ont pu être vérifiés expérimentalement que bien longtemps après : courbure de l'espace, existence de chats de Schrödinger, etc. Les physiciens s'accordent souvent à dire que leur but ultime est de concentrer les lois de la nature en quelques équations les plus compactes possibles : une fois posées les bonnes définitions mathématiques des objets physiques concernés, aucune des équations de Newton, de Maxwell, de Boltzmann, d'Einstein ou de Schrödinger n'occupe plus d'une demi-ligne.

Mais ces équations sont difficiles à résoudre ou même à approximer, et leur mise en pratique donne lieu à des calculs numériques complexes. Il faut se souvenir avec respect de l'extraordinaire volume des calculs manuels auxquels se sont attelés les mathématiciens pour compiler les tables de logarithmes ou les astronomes pour calculer la trajectoire des planètes et de leurs satellites, ce qui est devenu banal maintenant. Respectons aussi les marins, pour qui l'observation des éclipses des

satellites de Jupiter, initiée par Galilée, est longtemps restée le moyen officiel de calculer l'heure précise et donc la longitude à l'aide de tables. Ce n'était vraiment pas pratique en bateau, et beaucoup de naufrages ont été dus à des erreurs d'estimation de la longitude. Il a fallu attendre la fin du XVIII^e siècle pour que les montres précises et le sextant fournissent un moyen plus commode de le faire en mesurant la hauteur du soleil au-dessus de l'horizon, avec d'ailleurs d'autres tables².

Il est donc naturel que les premiers ordinateurs aient été créés pour le calcul scientifique : ils donnaient la possibilité de faire rapidement et sans erreurs des calculs difficiles à réaliser de façon correcte par des hommes à cause de leur taille et leur caractère routinier et ennuyeux. À partir des années 1950, l'ordinateur et les logiciels écrits dans le langage de programmation d'apparence mathématique FORTRAN (FORmula TRANslator, 1954) ont permis d'automatiser et de rendre routiniers la plupart de ces calculs fastidieux, en supprimant progressivement le besoin de tables. Puis, la diminution de la taille des circuits a rendu ces calculs invisibles : qui a une idée de la masse de calculs que doit faire le GPS de son téléphone pour lui dire où il est ? J'aimerais faire un voyage dans le passé pour en montrer un aux navigateurs d'antan³...

LE COUPLE INFORMATIQUE-MATHÉMATIQUES DANS TOUTES LES SCIENCES

Les mathématiques n'ont cependant pas du tout joué le même rôle dans les sciences de la vie et même de la terre, qui sont longtemps restées beaucoup plus descriptives. Mais un phénomène profond est en train de rapidement changer la donne : l'irruption massive du couple informatique-mathématiques dans ces sciences, qui offre la possibilité d'établir de nouvelles façons d'observer, de comprendre, de modéliser la nature puis d'agir sur elle d'une façon bien plus profonde et efficace qu'auparavant⁴. L'imbrication entre sciences naturelles et informatique décrite au chapitre précédent pour la médecine va donc se généraliser. Il ne s'agira plus pour les chercheurs des sciences naturelles d'automatiser des calculs numériques ni de sous-traiter des informatisations ponctuelles à des ingénieurs souvent vus comme subalternes. Il s'agira au contraire pour eux d'employer toutes les ressources de la

pensée informatique et de ses algorithmes, machines et programmes pour attaquer leurs problèmes scientifiques sur un nouveau front, souvent mais pas nécessairement en conjonction harmonieuse avec les modèles mathématiques classiques.

Je vais illustrer cette nouvelle donne par quelques exemples. Mais il faut au préalable constater qu'elle commence à provoquer une certaine tension chez les chercheurs ou enseignants scientifiques installés, beaucoup n'ayant ni l'envie ni le temps de s'intéresser vraiment à une science informatique qui ne s'apprend pas « sur le tas » en quelques jours car elle est bien différente de la simple utilisation des logiciels de base. Il faudra probablement un changement de génération et une mise à jour profonde du système éducatif pour donner tout son poids à une évolution de cette ampleur.

LES RAISONS DE L'INFORMATISATION DES SCIENCES NATURELLES

En précisant un peu plus, on peut trouver quatre raisons à l'irruption de l'informatique dans l'ensemble des sciences naturelles. Trois d'entre elles sont directes, et une plus indirecte. La première raison, déjà analysée ci-dessus, concerne l'informatisation des instruments ; mais les chercheurs peuvent très bien utiliser les nouveaux instruments sans savoir en détail comment ils fonctionnent. La deuxième raison est qu'il faut absolument analyser finement les données massives que fournissent ces instruments. Ici, l'informatique apporte des progrès majeurs car elle permet le passage à la très grande échelle avec ses approches statistiques et la généralisation des méthodes d'apprentissage automatique. La troisième raison, déjà mentionnée au [chapitre 3](#), est la possibilité de *simuler algorithmiquement* les phénomènes considérés, soit par calcul numérique des solutions d'équations mathématiques faisant intervenir le temps, soit à l'aide d'approches algorithmiques ne reposant pas sur des équations classiques. J'illustrerai ce dernier point par l'utilisation d'automates cellulaires pour la simulation des dunes de sable. La simulation devient aussi très importante en physique de la matière condensée et de la matière molle, en océanographie, en climatologie, et dans bien d'autres domaines.

J'en viens à la quatrième raison, qui est plus indirecte mais tout aussi importante : l'informatisation massive de la communauté des chercheurs, dont les productions elles-mêmes sont complètement informatisées et dématérialisées. Internet rassemble l'intégralité des publications, beaucoup de résultats d'expériences, de très nombreuses discussions scientifiques, etc. Il ne faut pas oublier qu'Internet était disponible pour les scientifiques bien avant de l'être pour le public, que le Web est né des besoins internes du CERN, que les chercheurs échangent massivement à distance par courrier et téléconférences Internet, que les traitements de textes mathématiques modernes (TeX et LaTeX en particulier) ont unifié l'écriture scientifique autant que le langage mathématique a unifié les formalismes des diverses théories, qu'on ne peut même plus soumettre de « papier » en papier aux journaux et conférences, et enfin que les articles scientifiques eux-mêmes forment une grande masse de données qui commence à être étudiée informatiquement au même titre que les données dont ils parlent⁵.

L'INFORMATIQUE EN MATHÉMATIQUES

Les mathématiques ne font pas partie des sciences naturelles, et on ne voit pas *a priori* de raisons pour lesquelles l'informatique serait importante pour les raisonnements conceptuels des mathématiciens. Effectivement, assez peu d'entre eux y font appel pour l'instant, hormis bien sûr pour leurs publications. Mais l'informatique commence à jouer un rôle important dans plusieurs domaines mathématiques : l'aide au calcul numérique bien sûr pour les mathématiques appliquées ; l'aide au calcul symbolique avec des systèmes très sophistiqués comme Maple et Mathematica, capables de faire avec exactitude des calculs gigantesques hors de portée des hommes ; enfin, plus récemment, la démonstration complètement formelle de grands théorèmes, qui paraissait inaccessible il y a seulement une quinzaine d'années. Les raisonnements théoriques des logiciens sur la nature profonde des preuves en mathématiques passent à la pratique d'une façon surprenante. J'illustrerai tous ces points à la fin de ce chapitre. Mais passons maintenant aux exemples.

AVERTISSEMENT

Je vais illustrer ces points dans la suite, tout en souhaitant que le lecteur admette que je ne suis en aucun cas assez cultivé pour être sûr de moi partout. Je vais surtout rapporter et interpréter à ma façon des exemples qu'on m'a expliqués. N'hésitez pas à me signaler mes erreurs ou imprécisions, et surtout à me fournir de nouveaux cas d'intérêt pour de possibles éditions ultérieures !

L'informatisation de l'astronomie

Si la pratique de l'astrologie est très ancienne, c'est bien la lunette de Galilée qui a permis à l'astronomie moderne de démarrer. Elle a été lentement perfectionnée avec la construction des grandes lunettes et grands télescopes terrestres. L'astronome classique, symbolisé pour toujours par le Professeur Calys dans *L'Étoile mystérieuse* d'Hergé, alternait entre l'observation directe l'œil fixé sur l'oculaire, les calculs numériques sur papier incompréhensibles pour le commun des mortels et la consommation de caramels mous. Et même les premiers radiotélescopes, ceux qui ont permis la confirmation de l'existence des pulsars par Jocelyn Bell en 1967, avaient encore l'air de beaux bricolages de jardin branchés sur de l'électronique analogique⁶.

Le travail de l'astronome s'est ensuite complètement transformé : son œil a d'abord été remplacé par des films photographiques et son papier par un ordinateur. Puis, les films ont été remplacés par des capteurs CCD (voir le [chapitre 6 sur la photographie](#)), et des télescopes numériques hyperperfectionnés ont été embarqués dans des satellites. Les satellites sont eux-mêmes devenus des objets très complexes et très informatisés : leur mise en orbite est contrôlée informatiquement selon les algorithmes définis par les automaticiens, et leurs ordinateurs embarqués réalisent en permanence le contrôle de leur orbite et leur attitude, le pointage fin des instruments, le traitement préliminaire des images et données de toutes sortes, la communication de ces données au sol, etc. Tout cela nous a donné des résultats extraordinaires et bien médiatisés à travers les sublimes images montrées partout, dont bien peu de spectateurs réalisent d'ailleurs qu'elles ne sont pas des photographies classiques mais des résultats de grands calculs et que leurs couleurs ne sont pas forcément naturelles

mais choisies pour mieux illustrer les phénomènes d'intérêt, comme d'ailleurs pour l'imagerie médicale.

La plupart du temps, l'astronome travaille devant son écran en recevant ses données par Internet, ce qui ne l'empêche pas de faire des missions à Hawaïi, dans le désert de l'Atacama au Chili, et dans des tas d'autres endroits intéressants – le veinard. Ces données n'arrivent pas brutes des instruments mais sont préalablement travaillées à terre par des algorithmes sophistiqués, l'astronomie ayant été pionnière dans le traitement informatique des images. Et bien des astronomes développent eux-mêmes de nouveaux algorithmes pour mieux analyser les données qu'ils reçoivent.

L'EXEMPLE DES LENTILLES GRAVITATIONNELLES

Un cas particulièrement intéressant est celui des *lentilles gravitationnelles*, illustrées par la figure 53 présentée par Françoise Combes lors de sa leçon inaugurale au Collège de France ([Combes, 2014](#)). Les taches bleues de la photo de gauche résultent de la courbure des rayons émis par un objet très lointain provoquée par l'amas de galaxies plus proche qui le masque, selon la relativité générale d'Albert Einstein. Le même effet est produit dans le dessin de droite par la déviation simulée des rayons émis par le texte latin *Docet Omnia* (« Nous enseignons tout »), devise du Collège de France. On constate des aberrations sphériques et chromatiques violentes mais de même nature que celles dont nous avons parlé au [chapitre 6](#) pour la photographie, en les qualifiant alors de gênantes. Pour les astronomes, ces aberrations sont au contraire un cadeau de la nature, car elles leur donnent sur la galaxie et l'objet lumineux situé derrière des informations auxquelles il leur serait tout à fait impossible d'accéder autrement. Extraire ces informations demande le développement d'une algorithmique très sophistiquée.

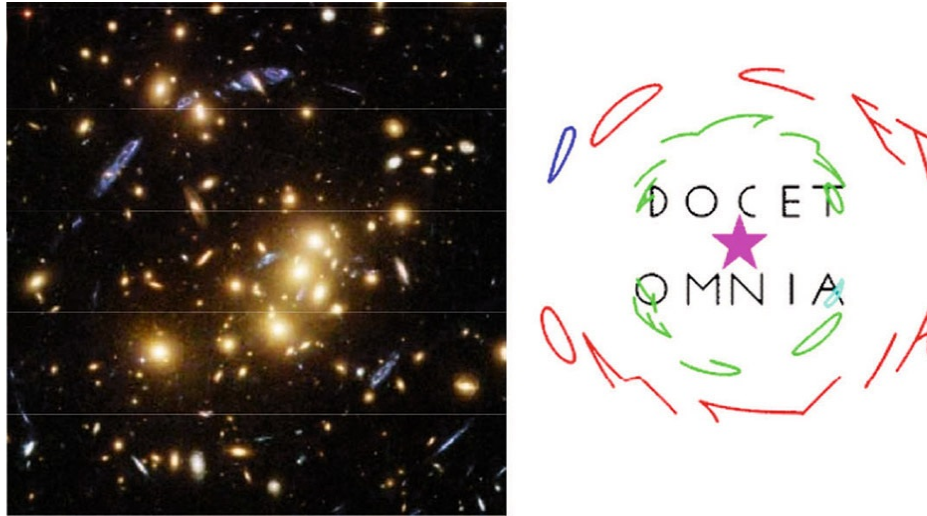


Figure 53. Lentilles gravitationnelles (source F. Combes).

LES ROBOTS ASTRONOMIQUES

Mais les astronomes ne fabriquent pas que des télescopes. Ce sont aussi de grands constructeurs de robots, qu'ils envoient sur la Lune, sur Mars, et même sur la comète Tchouri. Tout le monde se rend compte que ces robots aussi élégants que solides sont des prodiges d'invention mécanique. À l'inverse, pratiquement personne ne sait ni ne dit que ce sont aussi des concentrés d'informatique haut de gamme, que tout leur comportement est dirigé par des logiciels complexes, et que les envoyer jusque là-bas en bon état pose aussi des questions d'informatique redoutables. Ceux qui conçoivent et opèrent ces robots le savent bien, mais ici aussi, les commentateurs extérieurs au domaine restent collés à la matière. Le 18 septembre 2014, jour de l'annonce de ma médaille d'or du CNRS, j'étais invité à la belle émission *La Tête au carré* de Mathieu Vidard, juste après que Philae se fut posé sur la comète Tchouri. À ma connaissance, j'ai été le seul ce jour-là et les suivants à mentionner la prouesse informatique qui était derrière cet exploit...

Robots et satellites astronomiques ont aussi eu pas mal de difficultés à cause de bugs informatiques sournois et très instructifs. J'en décrirai plusieurs en détail au [chapitre 9](#).

SIMULER L'UNIVERS

En tant que scientifiques de la nature, les astronomes rencontrent une difficulté assez spécifique : s'ils peuvent observer finement la nature lointaine et envoyer des sondes et des robots sur les planètes et astéroïdes proches, c'est-à-dire dans le système solaire, ils ne peuvent pas vraiment expérimenter directement avec les étoiles, les supernovae, les galaxies et les amas de galaxies. Pourtant, ces objets les intéressent tout particulièrement maintenant qu'ils se demandent ce que peuvent bien être ces « matière noire » et « énergie noire⁷ » qui pourraient peut-être peupler l'univers. Par exemple, les supernovae, résultats d'explosions d'étoiles en fin de vie ou bien d'étoiles binaires contenant une « naine blanche », sont très intéressantes car elles participent grandement à la fabrication des molécules lourdes dont notre planète et nous sommes composés. Mais elles sont lointaines et peu fréquentes, et il n'est pas vraiment question d'en fabriquer au labo (pas la place, pas les budgets, et de violentes protestations des populations en vue).

Heureusement, les astronomes disposent maintenant d'un nouvel outil de luxe, la simulation numérique mathématico-informatique qui leur permet de remplacer leurs expériences de pensée précédentes par des *expériences de calcul*. J'en présentai ici deux cas.

QUE VA DEVENIR NOTRE BONNE VIEILLE TERRE ?

Le premier exemple concerne l'avenir de notre chère planète Terre, qui est loin d'être garanti tranquille, indépendamment de notre capacité à l'abîmer nous-mêmes. Si on ne sait pas vraiment comment elle est née, on sait mieux comment elle pourrait disparaître. Bien sûr, il n'en restera pas grand-chose quand le soleil explosera, ce qui est planifié pour dans 4 ou 5 milliards d'années. Mais il pourrait se passer des choses tout aussi tragiques bien avant au sein du système solaire, dues aux interactions gravitationnelles avec les autres corps célestes et en particulier les autres planètes et les astéroïdes. Après avoir établi les équations de la gravitation, Newton s'était posé la question de la stabilité du système solaire. En étudiant ces équations, Euler, Lagrange, puis Laplace ont cru démontrer cette stabilité. Mais Henri Poincaré a réduit à néant cette preuve en montrant l'importance des petites perturbations qui peuvent

conduire à des comportements différant exponentiellement en fonction du temps quand ce temps devient grand. Il a ainsi introduit ce qu'on appelle maintenant le *chaos déterministe*, notion mathématique précise qu'il ne faut surtout pas identifier au sens intuitif du chaos, ni interpréter en pensant que chaque papillon brésilien va provoquer un cyclone aux États-Unis. Le fait qu'un système soit chaotique signifie simplement qu'une modification minime des conditions initiales peut engendrer des divergences arbitrairement grandes à long terme. On sait maintenant qu'en fonction des données actuelles, on peut calculer les trajectoires assez précisément pour les prochains millions d'années. Mais on sait qu'on ne saura plus rien après 60 millions d'années, simplement à cause de la présence des astéroïdes dont on ne connaît pas suffisamment bien les perturbations : nous n'avons pas de moyen de connaître précisément les conditions initiales du calcul.

Cette étude mathématique peut être complétée par des simulations informatiques montrant que des événements catastrophiques (cette fois au sens habituel du mot) pourront se produire avant cette durée de 60 millions d'années : par exemple des collisions entre Mars et la Terre, Mars et Vénus, voire Vénus et la Terre. Si ça se passe, et si nous sommes encore vivants, nous verrons la planète arriver, mais plus rien après... Tout cela est bien analysé et exposé dans un article et des exposés de Jacques Laskar, faciles à trouver sur le Web ([Laskar, 2010](#)).

LA SIMULATION DES GALAXIES

En passant à l'échelle supérieure, les astronomes cherchent aussi à simuler la naissance et la vie des galaxies et systèmes de galaxies, en particulier pour comprendre ce qui se passe pendant les collisions de galaxies ([Teyssier, 2016](#)). Attendre et voir ce qui se passe n'étant pas possible, la simulation est la seule façon d'aller plus vite que le temps réel (voir le [chapitre 3](#)). Les simulations sont difficiles et coûteuses à cause du nombre et de la nature des objets à manipuler. Mais que valent-elles ? Ici, la situation est meilleure que pour le système solaire. S'il est évidemment impossible d'observer directement l'évolution des galaxies, à cause de sa lenteur à notre échelle, cette faiblesse est compensée par le fait qu'il existe des milliards de galaxies dans l'univers, de plus en plus jeunes quand elles sont de plus en plus loin :

puisque la vitesse de la lumière est finie et petite aux échelles astronomiques, ce que nous voyons loin dans l'espace est également loin dans le temps. Ainsi, les différentes formes et structures des galaxies peuvent s'observer à travers l'ensemble des directions de l'espace et reliées à leur âge. Même si les galaxies des simulations n'existent pas dans le ciel, on peut vérifier qu'elles ont bien les mêmes formes et densités d'étoiles que celles qu'on voit aux mêmes âges, et que les simulations engendrent toutes les formes visibles et rien qu'elles. En résumé, les astronomes veulent simuler les types et formes de galaxies, leur naissance, leurs interactions et leur mort éventuelle, pas le ciel tel que nous le voyons.

L'exemple des galaxies est aussi intéressant car il met en évidence le besoin de coupler différents types de simulations locales dans une simulation globale. Si les mouvements à grande échelle peuvent être vus comme régis par la seule gravitation, la naissance et la mort des étoiles jouent aussi un rôle essentiel. Or ces processus ne dépendent plus de la seule gravitation. Il faut donc d'autres modèles physiques pour les simuler, plus un couplage soigneux de ces modèles locaux avec le modèle global dans des simulations multiéchelles pour obtenir des galaxies virtuelles « naturelles », c'est-à-dire ressemblant à celles qu'on observe effectivement. De façon générale, il faut résister à l'idée naïve que la simulation doit être « exacte ». Elle doit être d'abord faisable et ensuite utile, ce qui impose à tout instant des compromis entre coût du calcul et précision du résultat – ce qui est d'ailleurs le cas dans presque toutes les utilisations de l'informatique dans les sciences naturelles.

LES IMAGEURS DE L'ASTRONOMIE

L'imagerie est évidemment fondamentale en astronomie, et il y a toujours eu le choix entre envoyer les imageurs dans l'espace ou les laisser sur Terre, les progrès étant permanents dans les deux cas. Mais il n'est pas encore possible d'envoyer de très grands miroirs dans l'espace : on ne peut qu'envoyer soit des miroirs relativement petits soit des grappes de satellites dont l'assemblage simule un très grand instrument spatial, par exemple pour la mesure de distances ou l'interférométrie. Cela fait que les grands instruments terrestres continuent à jouer un rôle essentiel. Sur Terre, les principaux problèmes sont l'absorption des rayons lumineux par l'atmosphère, les

perturbations que ses instabilités engendrent sur le trajet des rayons, et l'éclairage urbain qui est diffusé par les particules de l'atmosphère et brouille les observations.

Par exemple, ALMA (Atacama Large Millimeter/submillimeter Array), installé à plus de 5 000 mètres d'altitude dans le désert chilien de l'Atacama, est un interféromètre en ondes radio millimétriques ou submillimétriques. À ces longueurs d'onde, il faut aller le plus haut possible car les rayons sont vite absorbés par l'atmosphère. Dans le même désert, des télescopes optiques comme les quatre VLT (Very Large Telescopes) avec leurs miroirs de 8 mètres, et le futur E-ELT (European Extremely Large Telescope) de 39 mètres prévu pour 2024 sont situés à 2 500 mètres, sans crainte de pollution lumineuse dans ces contrées éloignées de toute ville. Pour corriger en temps réel les déformations du front d'onde dues à la turbulence de l'atmosphère, on y utilise l'*optique adaptative* : en déformant à une échelle de temps inférieure à la milliseconde un miroir secondaire où passe la lumière, on immobilise une étoile guide et donc aussi les autres étoiles. Le miroir secondaire est en fait pavé de micromiroirs commandés électriquement, l'ensemble étant contrôlé informatiquement.

Enfin, un des capteurs CCD actuellement installés sur un télescope à grand champ ne fait pas moins de 90 gigapixels, et se décharge en deux secondes grâce à une architecture ultraparallèle. On est bien loin des appareils photo du [chapitre 6](#)...

LES INTERFÉROMÈTRES RADIO GÉANTS

Un autre instrument très important pour les astronomes est l'*interféromètre radio* dans les ondes métriques, bien connu des amateurs depuis l'inauguration en 1965 du radiotélescope de Nançay. L'interféromètre LOFAR (Low Frequency Array) possède maintenant 20 000 antennes situées dans 48 stations, dont 40 aux Pays-Bas et une à Nançay, ce qui lui donne une surface de collecte de 300 000 mètres carrés. Il lui faut un superordinateur pour analyser les signaux – ici un Blue Gene d'IBM. Un projet interféromètre international encore plus grand nommé SKA (Square Kilometer Array) est prévu en Afrique du Sud et en Australie. Sa surface de réception sera composée de milliers ou millions d'éléments reliés par des fibres optiques. Son champ de vue global sera de 180 degrés, mais on pourra construire huit lobes d'observation

simultanés en programmant des retards de phase précis entre ces éléments. Pour calculer en temps réel les retards de phase déterminés par la direction dans laquelle on souhaite observer, typiquement celle d'une étoile ou d'une galaxie, reconstruire le lobe correspondant et extraire les résultats, SKA aura ainsi à traiter 1 téraoctet de données par seconde. Ce volume considérable interdit de stocker les valeurs sources des données reçues : en cas d'erreur dans une observation, la seule solution sera de repartir de zéro. On comprend pourquoi l'astronomie coûte cher...

L'informatisation de la physique

En physique des particules, l'informatisation en grand des appareils est devenue nécessaire très tôt, et d'abord pour les militaires : les expériences aériennes ou souterraines de bombes nucléaires étant dangereuses et mal acceptées par les populations (à juste titre), la décision est venue tôt de les remplacer par des simulations. C'est pour cela qu'a été lancé le Plan Calcul en France en 1966, à la suite du refus stratégique des États-Unis d'autoriser la vente d'un superordinateur CDC 6600 au CEA. Des simulations massives ont aussi été nécessaires pour les cœurs des centrales nucléaires au moment de l'essor de leur industrie. Mais l'informatique n'était alors vue que comme un outil de calcul pour les mathématiciens de l'analyse numérique qui étudiaient eux-mêmes les calculs pour les physiciens, certainement pas comme une science autonome.

LA RECHERCHE DE NOUVELLES PARTICULES

Dans les années 1990, les grands accélérateurs de particules ont inauguré l'ère des données massives et des projets de très grande taille en termes de chercheurs, d'ingénieurs et d'instruments. Le meilleur exemple est le LHC (Large Hadron Collider) au CERN, l'organisation européenne de recherche nucléaire. La détection d'une nouvelle particule comme le boson de Higgs se fait par l'analyse d'un très grand nombre de collisions, dont très peu sont productives, et qu'il faut donc analyser statistiquement par informatique. Les volumes de données produites sont

gigantesques : le LHC et ses détecteurs produisent de l'ordre de la dizaine de pétaoctets (millions de milliards d'octets) par an. On peut dire qu'on cherche non plus une aiguille dans une meule de foin mais un brin de foin dans une meule d'aiguilles. Il faut filtrer, trier et gérer toutes ces données, puis les rendre accessibles par réseau aux physiciens. Les besoins de communication entre chercheurs rendent aussi une informatisation à grande échelle indispensable, et c'est pour les maîtriser que Tim Berners-Lee a créé le Web. L'informatisation de ce type d'institut est donc très lourde.

AUTRES DOMAINES POUR LA SIMULATION

Dans d'autres domaines comme la physique quantique ou la physique statistique, les chercheurs ont adopté une informatique plus légère, davantage orientée vers des simulations numériques permettant d'affiner les modèles mathématiques et de préparer les expériences complexes qui permettront de vérifier les conséquences intéressantes des recherches théoriques. La simulation informatique devient un outil de travail d'usage constant au même titre que l'équation. Dans le cas de la fusion nucléaire à grande échelle, comme pour le projet de réacteur tokamak ITER ou de ses concurrents, la simulation est absolument critique pour savoir contrôler la géométrie des plasmas producteurs d'énergie, ce qui reste un problème très difficile.

Je ne donnerai pas davantage de détails sur ces domaines, qui ne sont pas très différents de ceux de l'astronomie dont j'ai parlé précédemment. Je choisis plutôt de décrire une avancée dans le domaine des sciences de la terre : la simulation des dunes de sable, due à l'équipe de Clément Narteau (Narteau, 2016) à l'Institut de physique du globe de Paris (IPGP). Un grand intérêt de la simulation des dunes est que celles-ci sont d'une beauté magique et pas réservées aux déserts terrestres, car elles constituent aussi un élément fondamental des paysages martiens. L'équipe de Clément Narteau s'est intéressée à l'observation de la variété et de la structure des dunes, ainsi qu'à l'expérimentation physique en fabriquant des dunes à partir de l'action du vent sur des tas de sable dans des déserts. Elle a ensuite décidé de compléter ces mesures et expériences par des simulations numériques. Ces simulations remarquablement réalistes et informatives ne reposent plus sur la résolution ou la numérisation

d'équations dans le monde des mathématiques du continu, mais sur l'utilisation du formalisme des *automates cellulaires*, systèmes dynamiques discrets de nature purement algorithmique introduits par le grand mathématicien John von Neumann en 1940. Les automates cellulaires ont été largement popularisés par le très simple et très élégant *jeu de la vie* (*game of life*) inventé par le mathématicien John Conway en 1970. Avant d'étudier les dunes, quelques préliminaires sont nécessaires.

LES AUTOMATES CELLULAIRES

Un automate cellulaire se définit par une grille discrète de cellules, en deux dimensions pour le jeu de la vie et trois pour les dunes, une liste d'états possibles de ces cellules, et des règles d'évolution de ces états au cours du temps vu comme une suite d'instantanés discrets. Ces règles sont *locales*, c'est-à-dire qu'elles définissent le nouvel état de chaque cellule seulement à partir de l'état courant de ses voisines proches. Toutes les cellules évoluent en même temps lors d'une transition du système, sans qu'aucune ait une connaissance globale de l'état de la grille. L'intérêt majeur est que l'ensemble des cellules peut exhiber des comportements macroscopiques fascinants à partir de règles microscopiques simples.

LE JEU DE LA VIE

Un exemple de jeu de la vie est représenté à la figure 54. Chaque cellule de la grille 2D peut avoir deux états appelés *vivant* ou *mort*, l'état vivant étant représenté par un pion (dont la couleur est sans importance dans la figure). La transition entre deux instantanés discrets est définie ainsi : une cellule vivante à l'instant t reste vivante à l'instant $t+1$ si elle a deux ou trois voisins vivants à l'instant t , alors qu'une cellule morte à l'instant t devient vivante à l'instant $t+1$ si elle a exactement trois voisins vivants à l'instant t . Cette loi s'applique à toutes les cellules en même temps. La figure 54 montre un canon qui envoie les « planeurs » situés en bas de la figure, avec une période 30. Chaque planeur glisse obliquement avec une période 3, comme le lecteur peut facilement le vérifier. À partir de la configuration finie de la moitié haute de la figure, les règles vont produire une infinité de planeurs de façon périodique.

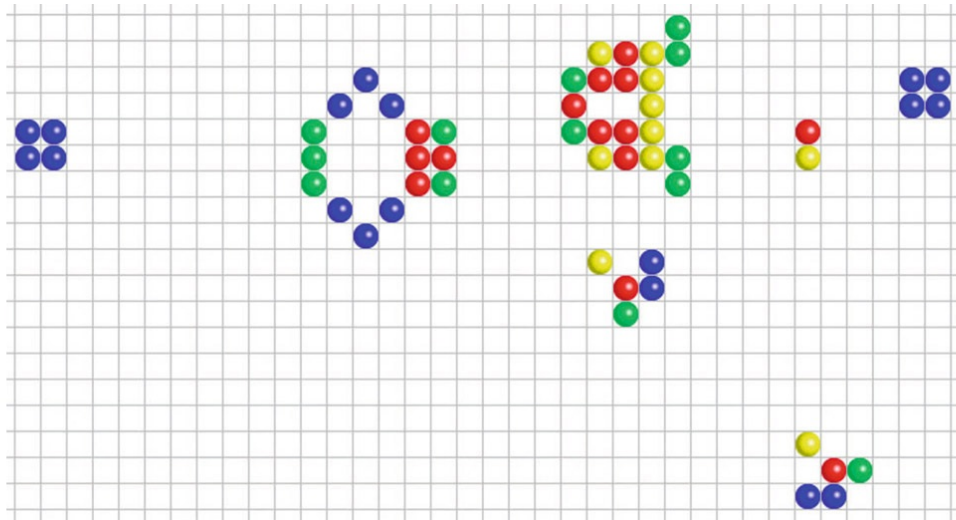


Figure 54. Un canon à planeurs dans le jeu de la vie (source Wikipedia).

Mais tous les jeux de la vie ne sont pas périodiques : malgré son extrême simplicité, le jeu de la vie est aussi puissant que la machine de Turing, et peut donc coder n'importe quel calcul de n'importe quel ordinateur. La plupart de ses propriétés sont donc indécidables. Mais ce codage est assez contourné et peu intéressant en pratique. Il est bien préférable de regarder les remarquables évolutions de forme en utilisant des exemples et logiciels facilement téléchargeables sur le Web.

LES DUNES DE SABLE

Revenons aux dunes. Leur forme dépend de quatre paramètres : la vitesse du vent, sa direction, qui peut être constante ou non, la nature des sédiments et leur alimentation (superficielle ou profonde par exemple). Elle se détermine dynamiquement en fonction de l'interaction entre ces paramètres au cours du temps.

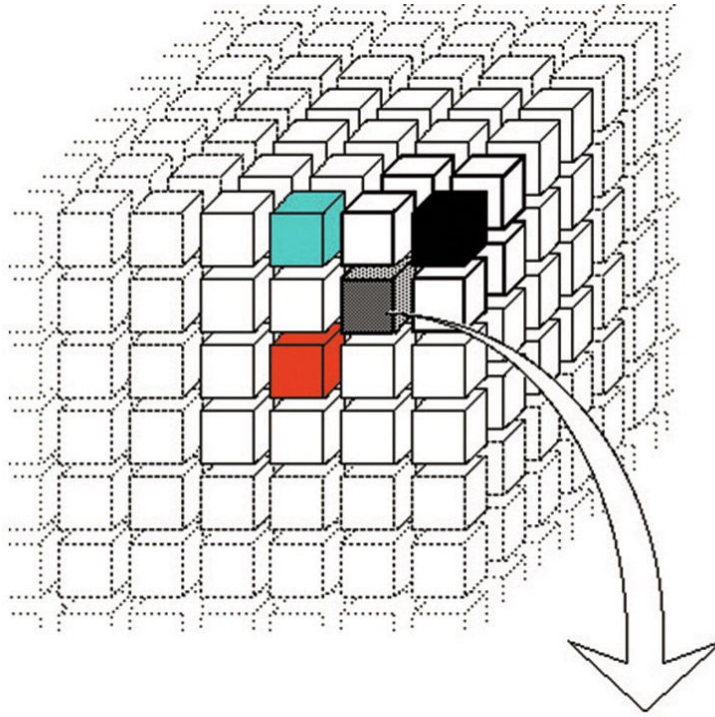


Figure 55. Grille de simulation pour les dunes (source C. Narteau).

L'automate cellulaire utilisé par l'équipe de Clément Narteau⁸ travaille en trois dimensions. Il repose sur un maillage de l'espace en petits cubes réguliers, illustré dans la figure 55. Chaque cube peut avoir 3 états : gazeux (l'air, en vert), solide stable (en noir) ou solide en mouvement (en rouge). Les expériences décrites ci-dessous utilisent 32 millions de cellules.

Les règles d'évolution sont décrites dans la figure 56. Elles conservent la somme des cubes noirs et rouges, c'est-à-dire la quantité de matériau. Les règles de déposition expriment comment un cube de sédiment mobile peut s'arrêter, celles d'érosion comment un cube fixe peut se mettre en mouvement, celles de transport comment un sédiment mobile peut se déplacer dans l'air, celles de gravité comment un cube arrêté peut tomber vers le bas, et celles de diffusion comment un sédiment stable peut se déplacer latéralement sous l'effet du vent. La flèche associée à chaque règle porte un nombre qui est sa probabilité d'application. L'automate cellulaire n'est donc pas déterministe mais stochastique. Les mouvements simulés ne sont pas identiques aux mouvements réels, puisqu'un cube ne peut pas se déplacer directement en oblique. La simulation n'est donc pas exacte, mais ça n'a pas d'importance :

comme pour les galaxies en astronomie, ce qui compte n'est pas la simulation précise des mouvements microscopiques mais l'obtention d'une bonne qualité des formes macroscopiques engendrées.

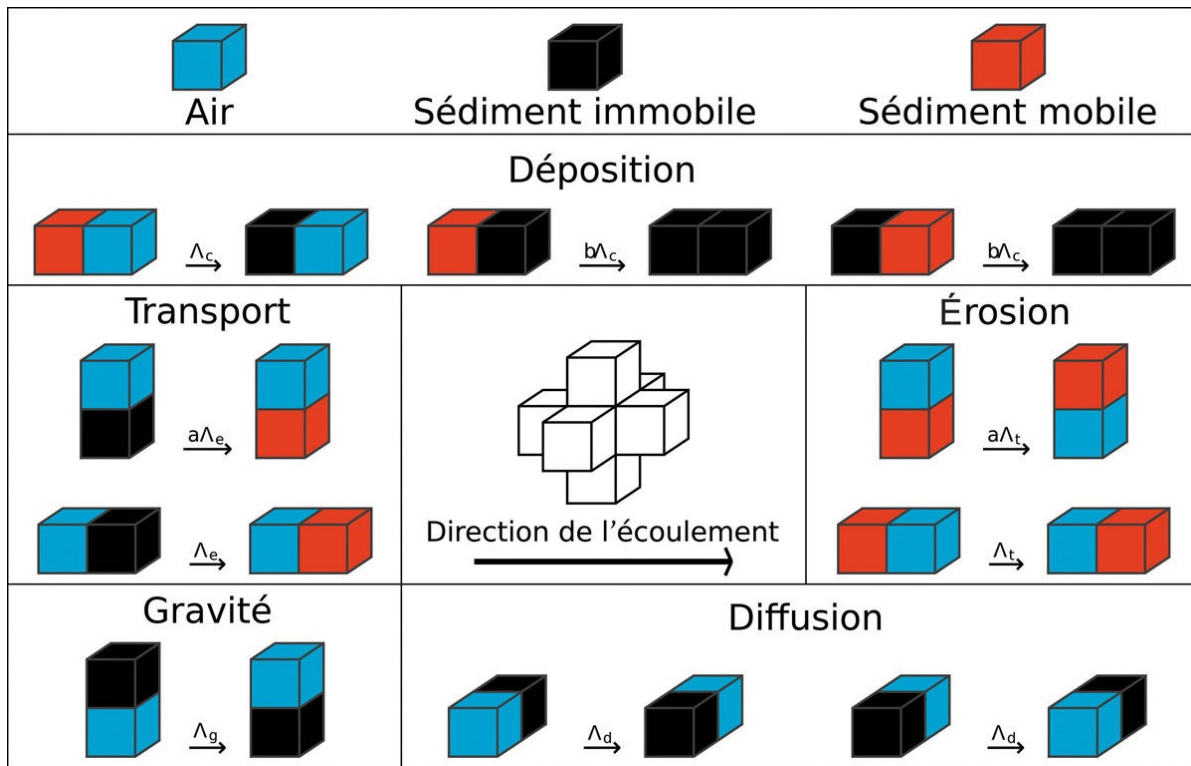


Figure 56. L'automate cellulaire des dunes (source C. Narteau).

Un dernier point s'impose : provoquer des avalanches lorsque les colonnes de sédiments stables engendrent des pentes trop importantes. Ces avalanches sont engendrées par un automate cellulaire déterministe auxiliaire, illustré par la figure 57. On s'intéresse alors à un peu plus de voisins, mais en restant quand même très local. L'écoulement et les cisaillements de l'air, qui vont déterminer les probabilités des règles à un instant donné, sont calculés par un troisième automate cellulaire, classique en mécanique des fluides et appelé un *gaz sur réseau*, que je ne décrirai pas ici.

Les trois automates sont couplés de la façon itérative suivante : (a) on calcule l'écoulement de l'air en prenant en compte la topographie courante ; (b) on tire au sort quelques cubes et quelques règles pour faire évoluer cette topographie en tenant

compte des probabilités déterminées par le nouvel écoulement de l'air ; (c) on traite les avalanches potentiellement engendrées ; (d) on recommence ces trois étapes jusqu'à l'obtention de la forme souhaitée. On est donc loin de la numérisation classique d'une équation.

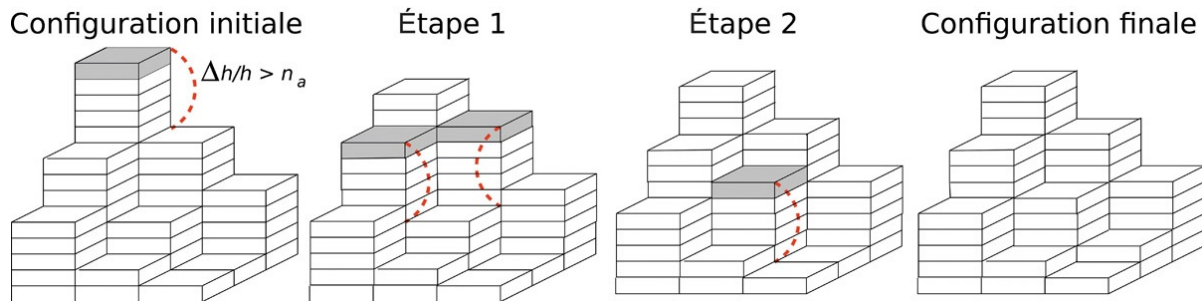
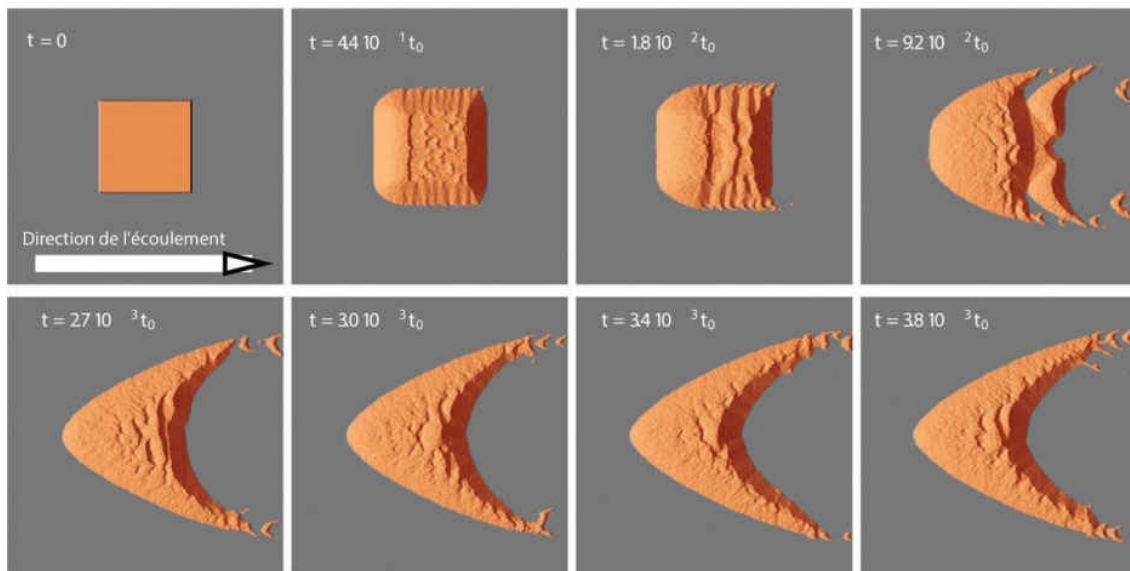


Figure 57. Traitement des avalanches (source C. Narteau).

Pour un régime de vent donné, stable ou instable, et pour une configuration initiale du sédiment mobile sur un sol non érodable, l'exécution itérative du couplage du gaz sur réseau, de l'automate principal et de celui des avalanches est capable de reproduire fidèlement les formes de dunes qu'on trouve dans la nature. Deux exemples sont donnés dans la figure 58, où le vent est supposé stable. On y voit en haut les *barkhanes* (dunes en croissant) et en bas les dunes linéaires. On observe aussi un phénomène bien réel qui n'apparaît pas sur les simulations continues antérieures et montre la finesse de la simulation : la formation de dunes secondaires au bout des cornes des barkhanes et au sommet des dunes linéaires.

Cette simulation a permis de comprendre pour la première fois la formation complexe des dunes en étoile, magnifiques structures d'étendue kilométrique qui apparaissent quand le vent peut prendre alternativement plusieurs directions stables. La figure 59 montre à gauche une dune réelle et à droite une dune simulée. La simulation a en particulier montré que les branches de l'étoile s'allongent dans la direction du flux sédimentaire à leur crête, ce qui n'était pas encore connu.

Barkhane



Dunes linéaires

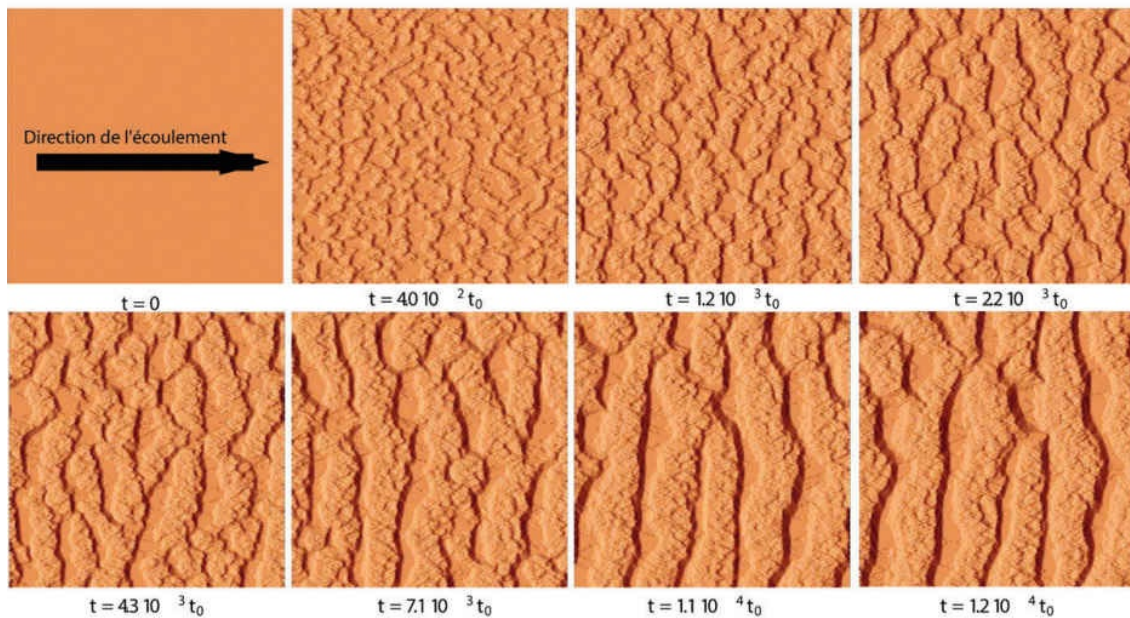


Figure 58. Deux dynamiques de simulation (source C. Narteau).

Enfin, la simulation permet d'étudier virtuellement des phénomènes de mélange de dunes, en permettant de détecter à l'aide de couleurs artificielles des mécanismes d'échanges de transferts de masses quasi impossibles à observer sur place. Par exemple, la figure 60 montre ce qui se passe pour un vent constant appliqué à deux tas

de sable alignés dans le sens du vent mais de tailles différentes, avec une vue de dessus et une de côté : les sédiments des deux tas se mélangent. En revanche, avec le même vent mais les tas décalés de la figure 61, les sédiments se mélangent très peu dans les dunes amont, alors que la collision conduit à l'éjection d'une dune aval où le mélange est pratiquement uniforme.

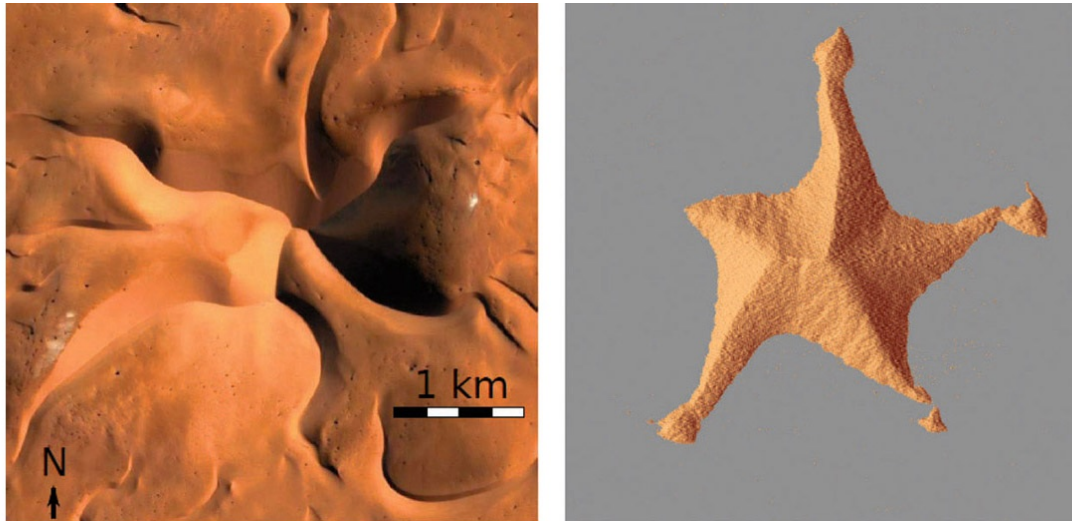


Figure 59. Dunes en étoile (source C. Narteau).

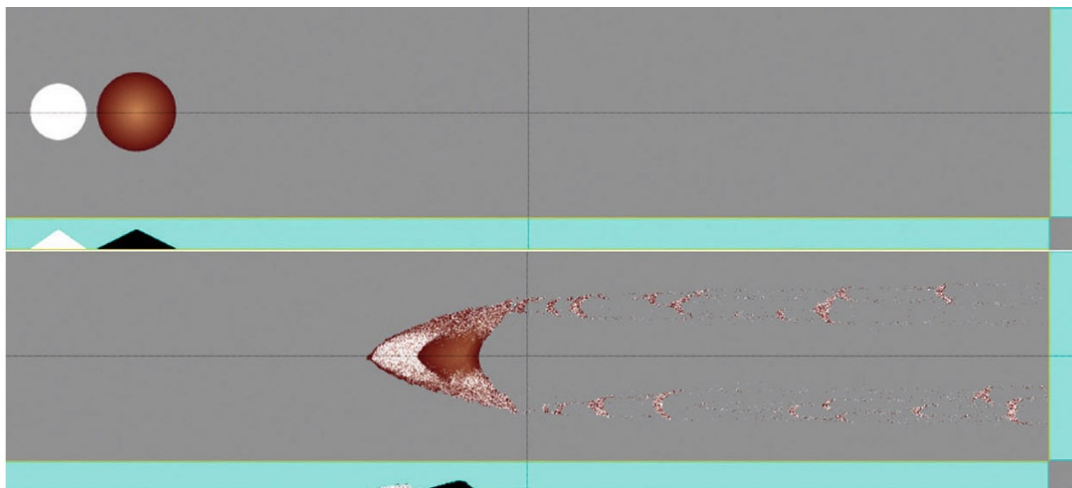


Figure 60. Tas alignés (source C. Narteau).

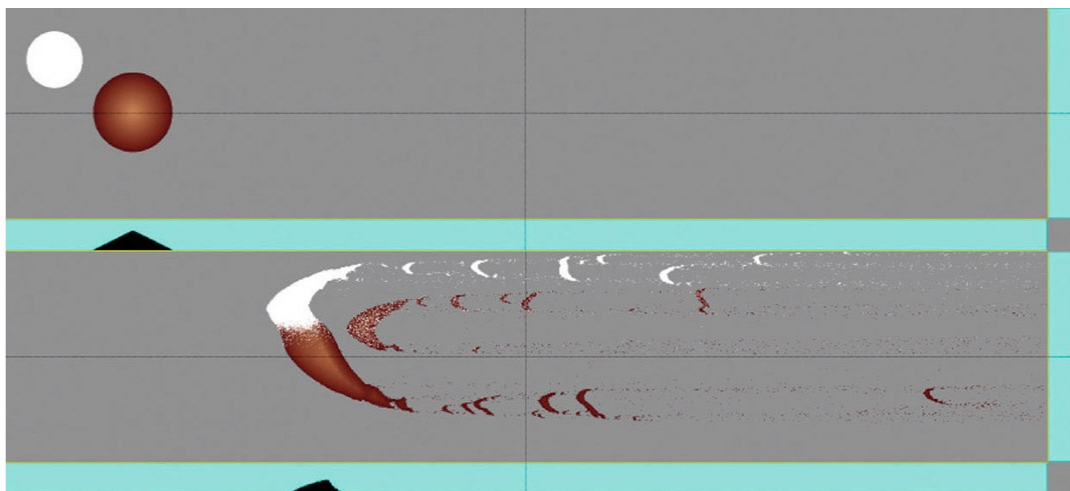


Figure 61. Deux tas non alignés (source C. Narteau).

J'invite le lecteur désireux d'en savoir plus sur ce beau sujet à visiter le site www.ipgp.fr/~narteau/geomorpho.html. Il y trouvera diverses simulations de champs de dunes, et aussi de dunes beaucoup plus complexes observées en Chine. D'autres travaux sur la simulation des dunes à partir d'équations de la physique peuvent être trouvés facilement sur le Web.

L'UNIVERS EST-IL DISCRET ?

En physique, certains chercheurs utilisent les automates cellulaires pour tenter de reconstruire une vision du monde entièrement discrète. En effet, la modélisation classique de la physique par les mathématiques du continu pose des problèmes souvent passés sous silence. D'abord, il n'est pas clair que les nombres « réels » méritent vraiment leur nom : sauf pour les nombres rationnels (aussi appelés fractionnaires), ce sont des objets de nature profondément infinie – qu'on pense aux décimales de π . On les utilise pourtant pour parler de quantités finies dans la nature. Ainsi, la modélisation classique par nombres réels viole un principe que les physiciens modernes admettent volontiers : un volume fini dans l'espace ne peut contenir qu'une information finie. De plus, pour les calculs, l'approximation des nombres réels par des nombres flottants pose des problèmes difficiles de validité des calculs, qui sont essentiels en pratique mais n'ont aucun sens physique. Enfin,

beaucoup de physiciens pensent que la nature est intrinsèquement discrète et que le continu devrait être considéré comme une approximation du discret, bien sûr extrêmement utile à cause des merveilleuses propriétés des mathématiques continues. Les physiciens et informaticiens du discret sont déjà capables de construire des automates cellulaires représentant bien la relativité restreinte d'Einstein et d'aborder la mécanique quantique à l'aide d'automates cellulaires probabilistes, mais ces théories sont encore jeunes et beaucoup de recherches difficiles seront nécessaires pour aller plus loin.

LA MATIÈRE MOLLE

Des discrétisations similaires sont à la base de la physique de la *matière molle*, celle des structures intermédiaires entre liquides et solides : mousses, gels, bulles, polymères, colloïdes, tensioactifs, etc., mais aussi cellules vivantes et colonies de micro-organismes. Ce domaine a donné le prix Nobel à Pierre-Gilles de Gennes (prix Nobel 1991), qui en fut un des principaux investigateurs avec toutes sortes de résultats étonnants. Comme pour les automates cellulaires, des lois strictement locales, même très grossièrement approximées, permettent de construire des comportements collectifs proches de ceux qu'on observe dans la nature. Par exemple, la croissance des cellules et leurs divisions pour former des tissus vivants peuvent être modélisées et simulées macroscopiquement de façon réaliste en considérant une cellule comme seulement réduite microscopiquement à une paire de points, qui se divise en deux paires quand la distance entre les deux points devient trop grande⁹. Il ne s'agit pas à proprement parler d'automates cellulaires, mais l'idée de transformations purement locales est analogue. Ce type de biophysique joue un rôle de plus en plus important dans l'étude du vivant.

Bio-informatique et biologie computationnelle

La biologie est un immense domaine scientifique composé de nombreux sous-domaines, de la biologie moléculaire et cellulaire à la biologie des espèces et des populations, en passant par la biologie des organes et des organismes et par celle du développement et de la morphogenèse¹⁰, également initiée par Alan Turing ([Turing, 1950](#)). Attention : la biologie est loin d'être un sujet que je domine, car je ne la connais que de façon indirecte à travers des lectures générales et des discussions avec des amis biologistes ou bio-informaticiens. J'espère que le lecteur me pardonnera les insuffisances ou possibles erreurs franches de ce chapitre, et m'aidera pour la suite en me communiquant ses corrections. Je vais donc me restreindre à quelques réflexions personnelles et à quelques exemples précis que je connais mieux.

BIOLOGIE ET INFORMATIQUE, DIFFÉRENTES MAIS COMPLÉMENTAIRES

La biologie classique est principalement une science de la matière et de l'énergie. Quels que soient ses sous-domaines, deux difficultés majeures pour les chercheurs sont d'un côté la complexité des mécanismes biologiques et de l'autre la lourdeur et la lenteur de l'expérimentation, qui reste absolument indispensable pour progresser. On est donc très loin de la facilité de l'expérimentation en informatique, presque toujours rapide, reproductible et conclusive, mais le plus souvent éloignée du monde du vivant.

C'est probablement pour cela que la plupart des biologistes n'ont pas eu un grand intérêt personnel pour l'informatique et n'ont pas demandé à leurs étudiants une formation à cette discipline. Enfin, comme j'ai pu le constater lors de nombreuses discussions, les biologistes sont moins enclins que les informaticiens à jongler avec une grande variété de niveaux d'abstraction, en particulier pour la modélisation et la simulation. Ce n'est pas surprenant, car ils sont le plus souvent plongés dans les détails très fins des processus qu'ils étudient.

Mais les choses changent vite : l'informatique a fait son entrée dans plusieurs domaines de la biologie, en commençant par sa participation au séquençage de l'ADN et à l'analyse des données du génome qui a conduit à ce qu'on appelle maintenant la *bio-informatique* (*bioinformatics* en anglais), puis à travers une expansion vers d'autres domaines comme la biologie des structures et des systèmes, domaine qu'on appelle plutôt la *biologie computationnelle* (*computational biology*). La terminologie me paraissant plutôt floue, je garderai bio-informatique pour tout ce qui concerne le traitement informatique des données biologiques et biologie computationnelle pour tout ce qui concerne la modélisation informatique de phénomènes biologiques.

On voit ainsi se créer de plus en plus de laboratoires mixtes regroupant des biologistes, des mathématiciens et des informaticiens. Par exemple, l'Inria et le CNRS ont de nombreux projets de bio-informatique et de biologie computationnelle, et, à l'Institut Pasteur, tous les entrants thésards ou postdocs sont désormais tenus de suivre une formation courte mais intensive à l'informatique. La même évolution se constate aussi pour la biophysique, ce qui fait que beaucoup de laboratoires deviennent réellement pluridisciplinaires au meilleur sens du terme.

Je présenterai surtout des cas où l'informatique peut aider la biologie, sans reparler de l'instrumentation, qui s'informatise comme partout ailleurs. Je noterai cependant que le chemin inverse de la biologie vers l'informatique est également possible : le stockage massif de données sur des brins d'ADN est très sérieusement étudié, même dans l'industrie, car l'ADN semble être le moyen le plus efficace en volume et le plus solide pour stocker l'information à long terme ; enfin, certains informaticiens étudient comment calculer avec de l'ADN au lieu de transistors.

LE SÉQUENÇAGE DE L'ADN

Les premiers apports de l'informatique ont concerné l'étude de l'ADN et du génome, sujet qui joue un rôle de plus en plus important en biologie et en médecine. Logiquement parlant, l'ADN est une très longue chaîne de bases A, T, C et G¹¹. Il contient des séquences particulières appelées *gènes*, qui déterminent chacun la construction d'une protéine essentielle à la cellule. Séquencer, c'est déterminer cette chaîne et isoler les gènes (qui s'avèrent moins bien délimités qu'on le pensait initialement). Vu la taille de l'ADN, qui va de millions de bases pour les bactéries à des milliards pour les mammifères, c'est impossible en un seul coup. Le séquençage par morceaux est né à la fin des années 1970, avec deux méthodes différentes dues à Walter Gilbert et Frederick Sanger, tous deux prix Nobel 1980 pour cela (le deuxième Nobel pour Sanger). Les premiers séquençages étaient longs et chers, et il a fallu des projets richement dotés pour décoder le premier génome humain en 2003, avec en particulier la contribution industrielle de la société Celera Sequencing de John Craig Venter, chercheur et entrepreneur que nous retrouverons plus loin.

Les techniques de séquençage initiales ont ensuite été supplantées par ce qu'on appelle le NTS (*new technology sequencing*), nom collectif de différentes techniques biologiques et informatiques qui permettent un très grand débit grâce à un parallélisme massif de toutes les opérations nécessaires. Je m'appuie dans la suite sur l'article récent ([Berger, Daniels & Yu, 2016](#)). L'ADN à séquencer, largement répliqué, est coupé en très nombreux fragments courts, de 50 à 200 lettres (davantage dans certaines technologies), qui sont ensuite chacune lues dans n'importe quel ordre et dans n'importe quel sens pour fournir une séquence ATCG. La lecture n'étant pas infallible, chaque séquence est accompagnée d'un score de qualité.

Le problème de la reconstitution de la séquence complète n'est alors plus du tout un problème de biologie, mais un problème d'informatique. Il faut recombinaison informatiquement toutes les sous-séquences présentées à l'endroit ou à l'envers, en s'appuyant sur le fait qu'elles ont de nombreuses parties communes. C'est un peu comme si on devait reconstituer un livre en n'ayant que des fragments de pages de nombreux exemplaires du livre, chacun pas en très bon état, en ajoutant que le livre est par ailleurs gigantesque puisque la taille du génome va de quelques millions à quelques milliards de bases. Les programmes informatiques pour ce faire sont très délicats et la reconstitution très chère en calculs, mais elle peut être rendue bien plus efficace si l'on peut s'appuyer sur la séquence précalculée d'un organisme similaire,

par exemple celui d'un homme pour le séquençage du génome d'un autre homme. En reprenant l'image du livre déchiré, il est plus facile à reconstituer si on possède une copie intacte, même pas forcément complètement identique.

Les performances du séquençage de l'ADN (ou de l'ARN) ont ainsi crû à une vitesse phénoménale, avec un gain d'un facteur 10 tous les ans : on peut maintenant séquencer un génome humain complet pour moins de 1 000 dollars, et ce coût va encore baisser. Le gain est bien plus rapide que celui de la loi de Moore, ce qui fait que les données sont encore plus massives que celles des physiciens ou des astronomes. Les problèmes informatiques de stockage, d'accès aux très grandes bases de données biologiques et de calcul risquent de devenir vite dominants.

QUE FAIT-ON AVEC LES SÉQUENCES ?

Le séquençage à haut débit permet de très nombreuses études absolument impossibles il y a seulement vingt ans, avec de très nombreuses applications en biologie fondamentale et en médecine. Je n'en mentionnerai que deux, d'un intérêt évident. La première est l'étude des variations ou mutations des gènes entre individus, qui peuvent être liées à des maladies comme les cancers¹² ou l'autisme¹³. La seconde est la comparaison des génomes des espèces entre elles, pour découvrir par exemple que nous avons hérité aussi des Néandertaliens, mais surtout pour construire des arbres phylogénétiques bien plus pertinents que ceux obtenus auparavant en se restreignant à des caractéristiques morphologiques et physiologiques.

Aucune étude ne peut être faite à la main sur ce volume de données. Beaucoup de logiciels ont été développés pour permettre d'étudier les ADN séquencés. Un des plus importants est BLAST (Basic Local Alignment Search Tool), développé aux États-Unis, qui permet de comparer l'ADN ou ses fragments à ceux stockés dans les grandes bases de données d'ADN, et en particulier de retrouver les gènes ([Altschul et al., 1997](#)). À partir de son développement initial en 1990, ses évolutions en ont fait un des logiciels scientifiques les plus utilisés, avec des articles cités plus de 65 000 fois (selon Google Scholar), ce qui est énorme. Un autre logiciel très important est PhyML, développé en France par Olivier Gascuel, et dédié à la détection des mutations par une méthode de maximum de vraisemblance.

DES AMÉLIORATIONS ALGORITHMIQUES MAJEURES EN VUE

Mais une difficulté nouvelle apparaît avec l'extension considérable de la taille des bases de données. Les algorithmes comme BLAST, de coût proportionnel à la taille de la base de données, risquent de ne plus passer à l'échelle. Une solution proposée dans l'article ([Berger, Daniels & Yu, 2016](#)) est de s'appuyer sur les progrès des algorithmes dits de *compressed sensing*¹⁴, fortement reliés à des questions fines de géométrie algorithmique et de plus en plus utilisés dans tous les aspects du traitement du signal. Dans l'étude du génome, ils pourraient permettre d'atteindre un coût fortement sublinéaire qui rendrait les analyses beaucoup plus abordables. L'idée utilisée ici est que les séquences vraiment importantes apparaissent comme très différentes les unes des autres, ce phénomène étant probablement une conséquence de la sélection naturelle. En caractérisant mieux ces séquences, on pourrait les indexer de façon beaucoup plus efficace dans des dictionnaires à accès rapides et coder aussi beaucoup plus efficacement leurs variantes.

LA RESYNTHESE DE L'ADN

Les algorithmes ne sont pas seulement nécessaires pour l'analyse de l'ADN, ils le sont aussi pour sa *resynthèse*, c'est-à-dire la synthèse d'un ADN complet à partir des quatre bases qui le composent. Cette synthèse à grande échelle a été réalisée de façon spectaculaire par l'équipe de Craig Venter ([Craig Venter, 2013](#)). Le jeu algorithmique et biologique est ici inverse de celui du séquençage. Il s'agit de reconstruire un ADN complet à partir des quatre bases A-T-C-G. C'est ici celui d'une bactérie, d'environ deux millions de bases. On sait maintenant synthétiser chimiquement de façon routinière de petites séquences, de longueur de l'ordre de la centaine de bases. Il a fallu recoller entre elles un grand nombre de telles séquences pour obtenir au caractère près l'ADN bactérien désiré. Cela n'a évidemment rien de simple, ni sur le plan biologique, ni sur le plan algorithmique pour trouver comment organiser les recollages. Pour être sûr de reconnaître ses copies par séquençage, cet ADN a été en plus marqué par addition des noms des membres de l'expérience et d'autres informations. Puis il a été introduit dans une autre cellule, celle d'une levure, qui est

biologiquement très différente car eucaryote (avec noyau) au lieu de procaryote comme la bactérie. La levure s'est effectivement mise à exprimer et dupliquer cet ADN, ce qui a fait disparaître le sien propre grâce à d'autres stratagèmes. Cette expérience extraordinaire a porté un coup sérieux à la théorie vitaliste, qui proclame que le vivant possède « quelque chose en plus » par rapport à la matière et l'énergie. Mais mon explication quelque peu incompetente ne vaudra bien sûr jamais la lecture du livre !

L'INFORMATIQUE POURRA-T-ELLE AIDER L'ÉPIGÉNÉTIQUE ?

Revenons sur un point : nous avons dit que l'ADN est une très longue suite de lettres. Ce n'est vrai que d'un point de vue logique car l'ADN est physiquement fortement replié sur lui-même dans une structure qu'on appelle la chromatine, pour n'occuper qu'un tout petit volume dans la cellule. Cela fait que l'expression d'un gène (passage du gène à la protéine qu'il code) n'est pas un processus simple, car il faut d'abord que le gène soit exposé dans la cellule pour qu'il soit traduit en ARN puis que cet ARN soit lui-même traduit en protéine. La chromatine joue donc un rôle essentiel dans la façon dont les gènes sont exprimés. De plus, sa structure peut être modifiée par divers facteurs dont l'environnement, et peut être préservée lors de la division cellulaire, ce qui fait qu'elle constitue une forme de mémoire cellulaire transmissible indépendante de celle des gènes. On entre ici dans le domaine de l'épigénétique ([Heard, 2013](#)), qui prend une importance de plus en plus grande en biologie.

Je pense que différents outils informatiques pourraient aider l'épigénétique, comme la géométrie algorithmique pour mieux comprendre les repliements ou comme les calculs de processus développés originellement pour la communication entre processus informatiques parallèles pour la formalisation de mécanismes de toutes sortes. Mais un usage bien différent de l'informatique et très utile aussi est celui du développement d'outils pédagogiques. Pour l'ADN et l'épigénétique, de remarquables animations sont disponibles sur le site australien Walter+Eliza Hall¹⁵. Elles résultent d'un travail important réunissant des biologistes et des informaticiens dans un projet pédagogique commun.

FORME ET FONCTION DES PROTÉINES

Le rôle des gènes est de coder pour des protéines. Celles-ci sont initialement construites par assemblages linéaires d'acides aminés selon le code linéaire de l'ARN qui résulte lui-même d'une transcription des gènes. Mais les protéines ne restent pas linéaires : elles se replient dans des formes complexes et ne sont fonctionnelles que si elles ont la bonne forme, qui peut être examinée par cristallographie ; voir la figure 62 pour l'ARN-polymérase, celle qui produit l'ARN à partir de l'ADN. De plus, ce sont des molécules déformables et vibrantes, dont la forme change lorsqu'elles sont en interaction avec d'autres protéines ou d'autres molécules. La modélisation informatique de la géométrie des protéines, de leur repliement et de leurs interactions est un champ de recherche majeur à travers le monde, voir par exemple le travail d'Alessandra Carbone à l'université Paris-VI ([Carbone, 2017](#)).

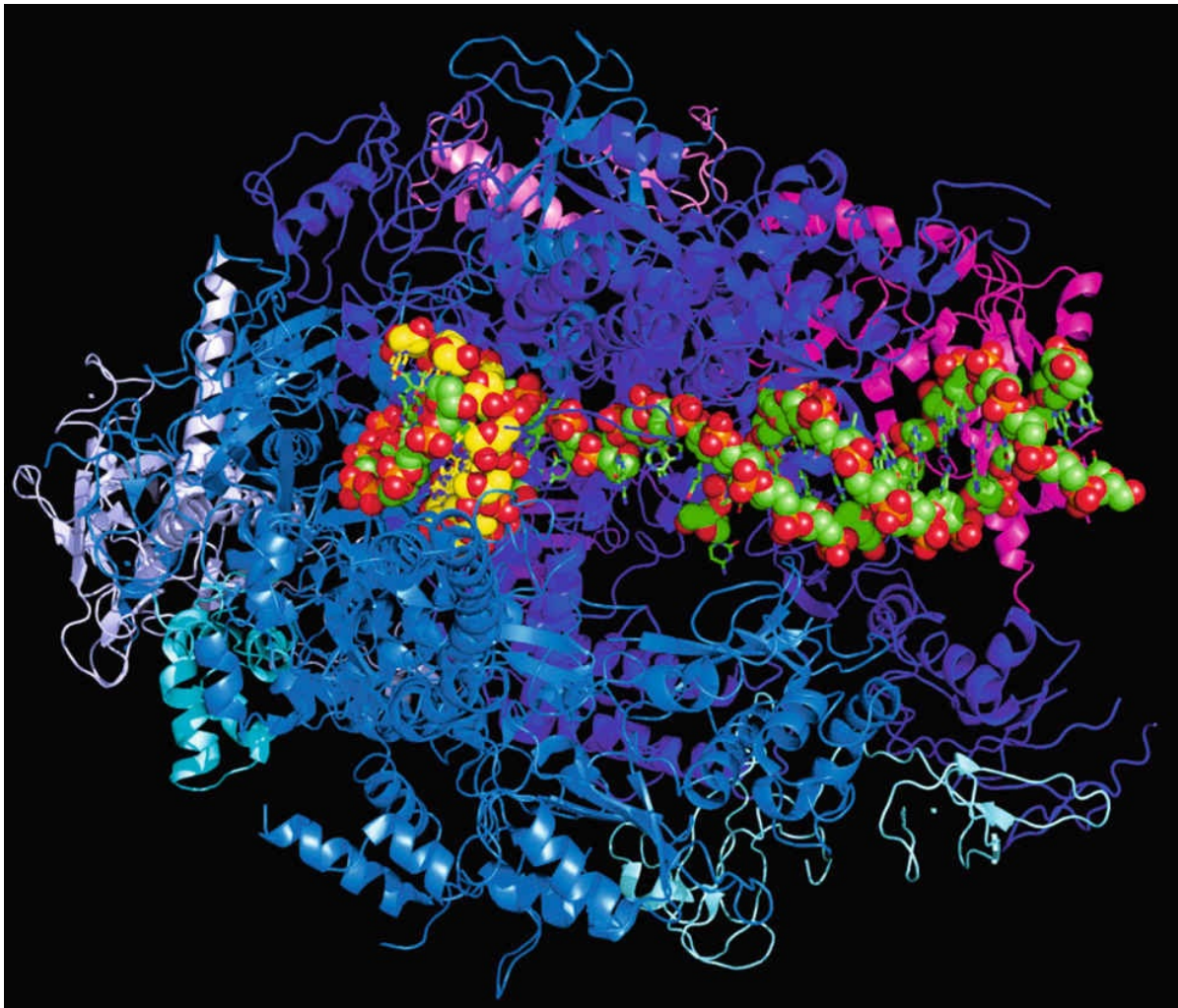


Figure 62. ARN-polymérase (source Frédéric Dardel, université Paris-Descartes, CC-BY-SA-3.0).

Les repliements dynamiques des protéines sont étudiés par de nombreux biologistes et informaticiens, et de nombreuses animations vidéo disponibles sur le Web les illustrent. La figure 63 montre un instantané extrait d'une simulation de repliement faite à l'Université de Stanford¹⁶. On y voit clairement que la simulation peut utiliser différents niveaux d'abstraction, certaines parties de la protéine étant abstraites sous forme de traits ou d'hexagones pour les cycles carbone, et d'autres étant soulignées en fausses couleurs. Également à Stanford, le projet Folding@home s'intéresse à la simulation massive des repliements avec des objectifs médicaux, qui demande des capacités de calcul considérables. Son originalité est d'adopter le principe du *crowd computing* : il vous propose d'utiliser votre ordinateur ou même

votre téléphone quand vous ne vous en servez pas pour calculer des repliements sans vous déranger !



Figure 63. Repliement simulé d'une protéine (source Stanford University).

SIMULER LA BIOCHIMIE CELLULAIRE

La cellule fermée par une membrane sélectivement poreuse pour communiquer avec l'extérieur est la base de tous les systèmes vivants. Par rapport aux systèmes artificiels que nous fabriquons, elle a l'avantage de savoir se diviser puis de se transformer en fonction des changements de l'environnement, par exemple en se spécialisant lorsqu'elle est née comme cellule souche. Comprendre le fonctionnement de la cellule en se plaçant à un niveau systémique plus global que celui des mécanismes biochimiques internes est un enjeu majeur en biologie computationnelle. Je prends ici brièvement comme seul exemple la modélisation des réseaux métaboliques, ceux qui expliquent des mécanismes biochimiques complexes à l'aide de nombreuses réactions en cascade engendrant des rétroactions (*feedbacks*). Cette

modélisation, très différente des précédentes, illustre bien une façon de penser très informatique que je n'ai pas encore présentée.

Un réseau métabolique se décrit sous la forme d'un grand graphe où chaque flèche représente une réaction biochimique. De nombreux exemples sont disponibles sur le Web. La construction pas à pas de ce type de réseau demande de très nombreuses expériences biologiques. Mais la présentation des réseaux reste souvent descriptive, et il faut résoudre une question essentielle pour avancer davantage : peut-on analyser *quantitativement* la dynamique des réseaux et prévoir ses résultats, par exemple en prédisant l'évolution au cours du temps des concentrations des différentes molécules ? C'est à ce travail que se consacre l'équipe Lifeware de François Fages à l'Inria. L'entreprise Dassault Systèmes collabore avec cette équipe, car elle est de plus en plus intéressée par la conception assistée par ordinateur de médicaments afin d'étendre son champ d'application traditionnel, plutôt orienté vers la mécanique.

L'idée est de modéliser les réseaux à l'aide d'une machine informatique abstraite appelée BioCham, extension habile de ma propre machine abstraite parallèle Cham (Chemical Abstract Machine), décrite dans l'[annexe 12.6](#). Sans entrer dans les détails, la BioCham permet de simuler et d'interroger les réseaux métaboliques à plusieurs niveaux d'abstraction qualitatifs et quantitatifs. Au niveau le plus bas, on peut s'intéresser seulement aux différents chemins dans les graphes, ce qu'ont toujours fait les biologistes, et demander par exemple quelles sont les différentes façons de produire une molécule, calcul déjà très difficile à la main sur les grands réseaux. Les résultats peuvent déjà être surprenants pour les biologistes. À un plus haut niveau, on peut quantifier les réactions, par exemple en étudiant l'évolution des concentrations des composants qu'elles utilisent¹⁷. On peut alors poser des questions sur l'évolution de la réaction en utilisant une logique temporelle comme celles que nous présenterons brièvement au [chapitre 10](#). On peut par exemple demander à un réseau : « Est-il possible de trouver des paramètres k_3 et k_4 compris entre 0 et 200 tels que la concentration de telle cycline oscille avec une période comprise entre 35 et 150 minutes ? » Le système peut alors trouver une solution, par exemple $k_3 = 10$ et $k_4 = 180$. Remarquons qu'il ne s'agit pas ici de programmation classique car on n'a pas besoin de définir un algorithme pour trouver la solution. On utilise la *programmation par contraintes*, spécialité de François Fages : on exprime simplement la contrainte que le résultat doit se trouver dans une certaine plage, et

c'est au logiciel BioCham de résoudre le problème par tous les moyens disponibles. Ses calculs font appel à des algorithmes subtils de résolution de contraintes logiques et numériques dont je parlerai brièvement au [chapitre 10](#) pour les systèmes SMT (Satisfaction modulo théories).

PEUT-ON VOIR LE VIVANT COMME UN SYSTÈME D'INFORMATION ?

La biologie computationnelle s'intéresse aussi à un aspect différent des précédents : voir les organismes vivants eux-mêmes comme des machines algorithmiques, qui exécutent biologiquement des algorithmes pour leurs propres besoins et exploitent aussi l'information qui leur parvient de l'extérieur à l'aide d'algorithmes. Certains cas sont évidents : pour marcher, une fourmi utilise des algorithmes de mouvement ordonné des pattes câblées biologiquement (nous aussi). D'autres sont moins immédiats : les fourmis émissaires cherchent de la nourriture au hasard, et, quand elles en trouvent, reviennent à la fourmilière en marquant leurs traces à l'aide de phéromones. Les autres fourmis empruntent cette trace et continuent à la marquer si elle est productive, la renforçant ainsi automatiquement. C'est ce qu'on appelle un algorithme *randomisé*, dans lequel l'utilisation du hasard permet de rendre l'exploration bien plus efficace qu'avec toute méthode déterministe systématique. Les éthologues ont étudié en grand ce type d'algorithmes, sans nécessairement connaître le mot. D'autres exemples concernant les vols d'oiseaux ont été présentés par Bernard Chazelle dans son cours de 2012 « L'informatique et les sciences » dans le cadre de la chaire Informatique et sciences numériques du Collège de France ([Chazelle, 2012](#)).

De nombreux exemples d'algorithmes naturels sont présentés dans le livre de Leslie Valiant (prix Turing 2010), *Probably Approximately Correct. Nature's Algorithms for Learning and Prospering in a Complex World*. Valiant y explique que le vivant utilise nécessairement des algorithmes très efficaces et « probablement approximativement corrects » pour survivre dans un monde hostile, et en détaille certains. Il exprime aussi l'idée que l'évolution elle-même peut être vue comme un

processus globalement algorithmique, ou tout au moins que l'analyser de cette façon est profitable.

Un bel exemple de ce point de vue algorithmique sur l'évolution est donné dans l'article récent « Sex as an algorithm : The theory of evolution under the lens of computation » ([Livnat & Papadimitriou, 2016](#)), écrit par Adi Livnat, bio-informaticien, et Christos Papadimitriou, acteur majeur de l'algorithmique moderne. Les auteurs s'y interrogent sur la prédominance de la reproduction sexuée sur la reproduction asexuée. Selon les biologistes, le sexe favorise le mélange des gènes, favorable à terme car il permet d'explorer un grand nombre de configurations individuelles ; mais, comme souvent en biologie, cette opinion est le plus souvent laissée purement qualitative. Livnat et Papadimitriou montrent que cette affirmation est effectivement supportée algorithmiquement, même bien mieux qu'on ne le supposerait intuitivement. Même si leurs idées ne sont pas encore vérifiées biologiquement, elles sont parfaitement convaincantes au moins pour un informaticien. Pour donner une idée de leur approche, je vais simplement reprendre ici leur exemple intuitif. Supposons que nous ayons deux populations, la rouge sexuée et la bleue asexuée. Supposons que 1 rouge généralement donne 3 descendants rouges, sauf dans 10 % des cas où il meurt sans progéniture et dans 2 cas sur 1 000 où il engendre une mutation bleue asexuée (ce qui est biologiquement possible). Pour les bleus asexués, qui ont plus de mal à se reproduire, supposons que 1 bleu donne en général 2 bleus, sauf dans 10 % des cas où il meurt sans progéniture et 1 cas sur 1 000 où il donne 1 bleu et 1 rouge sexué (tout autant biologiquement possible). Supposons même, cas apparemment défavorable au sexe, qu'il n'y ait qu'un géniteur initial et qu'il soit bleu, donc asexué. Alors, après 50 générations, les asexués ne sont en moyenne plus que 2 sur 1 000 ! La loi des probabilités est dure, mais c'est la loi...

Il y a beaucoup d'autres cas où voir le vivant comme un système d'information est pertinent. Par exemple, le biologiste Antoine Danchin présente souvent la cellule comme une machine à information, dans laquelle il ne faut pas seulement considérer la réplication de l'ADN et la transcription de l'ARN et protéines comme les seuls mécanismes informationnels fondamentaux ; il insiste sur le fait que la cellule a bien d'autres fonctions informationnelles, comme le nettoyage des vieilles protéines et le tri des jeunes protéines pour les incorporer dans ces cellules germinales. Ces fonctions font partie pour Danchin de ce qu'il appelle le « système d'exploitation » de la cellule,

en charge de maintenir ses ressources accessibles et en bon état comme pour celui d'un ordinateur. Je trouve cette comparaison tout à fait pertinente.

LA MODÉLISATION EN BIOLOGIE VÉGÉTALE

Je vais terminer cette section par un autre exemple que j'aime beaucoup et qui met en œuvre la plupart des considérations précédentes, la modélisation des plantes réalisée au sein de l'équipe INRA/Inria Virtual Plants à Montpellier¹⁸, dirigée par Christophe Godin. Cette équipe a développé de nombreux outils conceptuels et logiciels comme OpenAlea, bibliothèque logicielle pour la modélisation et la simulation qui regroupe plusieurs techniques algorithmiques. Les applications concernent la reconstruction informatique des plantes en fonction de scans 3D complets ou d'imagerie cellulaire des méristèmes (bourgeons terminaux), ou, à l'inverse, leur modélisation et leur simulation informatiques. Je ne prendrai ici qu'un seul exemple, celui de la modélisation des formes de plantes, en m'appuyant sur une idée fondamentale du mathématicien prolifique Andreï Kolmogorov, un des pères de la théorie moderne des probabilités.

L'idée est la suivante : la complexité apparente n'est pas forcément la complexité réelle, car des opérations mathématiques très simples peuvent produire des formes apparemment très complexes comme celles des plantes. La complexité de Kolmogorov d'une forme (par exemple) se définit par la plus petite taille d'un programme algorithmique capable d'engendrer cette forme. L'idée d'appliquer cette notion aux plantes n'est pas nouvelle : il y a longtemps qu'on y observe les nombres de Fibonacci ou d'autres quantités mathématiques connues, que ce soit dans les intervalles entre feuilles pour de nombreuses espèces ou dans l'arrangement des graines dans la fleur de tournesol, ce qui montre que ces structures apparemment riches sont en fait de petite complexité de Kolmogorov.

Mais on va aller ici beaucoup plus loin : quelle est la taille d'un programme permettant de simuler la vie entière d'une plante, par exemple d'un pommier, de ses cellules initiales à son âge de producteur de fruits, en tenant compte des contraintes mécaniques, du rythme des saisons, etc. ? Elle est petite, de l'ordre de quelques pages. On doit pour cela combiner plusieurs modèles, certains déterministes et d'autres

aléatoires. Les systèmes de réécriture de Lindenmayer, ou L-systèmes¹⁹, sont par exemple des outils simples bien adaptés aux lois de croissance de base. À partir de petits programmes, ils engendrent facilement de belles formes assez naturelles qu'on peut utilement moduler par de l'aléa pour les rendre encore plus naturelles. La croissance et la différenciation des méristèmes, critiques pour l'évolution de la plante, peuvent être simulées en partant du niveau cellulaire. On programme alors des lois relativement simples sur l'accroissement de la taille des cellules puis leur division et leur transformation en faisant intervenir principalement le taux d'auxine, une hormone fondamentale dans la croissance des plantes. L'impact de la gravité peut être simulé par des lois mécaniques, celui de la température par des modifications du système de croissance. Au lieu de se contenter de construire les superbes planches des livres de botanique, on peut ainsi atteindre une vision beaucoup plus dynamique du cycle de vie d'un pommier. Cela permet de le comprendre beaucoup mieux, car rien n'autorise autant une bonne compréhension qu'une bonne simulation. Ce qui est encore mieux, c'est qu'une fois le programme terminé, il peut être appliqué non seulement à un pommier mais à une famille de pommiers différents en changeant un petit nombre de paramètres.

Une modélisation bien conduite peut également s'appliquer à l'étude de l'effet de l'arrosage, ou à celle de l'impact des sécheresses ou du réchauffement climatique, avec des applications en agriculture. Ce champ peu connu du public est vraiment fascinant.

Les neurosciences computationnelles

LE CERVEAU VU COMME UNE MACHINE À INFORMATION

Les neurosciences ont été introduites et développées en France par Jean-Pierre Changeux ([Changeux, 2006](#)), que je remercie pour nos discussions éclairantes. Les neurosciences forment un domaine majeur de la biologie, naturellement proche de l'informatique sur au moins un aspect : le cerveau est vraiment une machine à information. Comme le montre la figure 64, je pense même qu'on peut transposer sans risque les piliers de l'informatique décrits dans la [figure 10 du chapitre 3](#) : les interfaces d'entrée et de sortie deviennent nos sens et nos muscles ; les données sont codées par des *potentiels d'action* (*spikes* en anglais) produits par les neurones sous la forme de séquences temporelles de pics électriques dépassant nettement d'un bruit assez élevé ; les algorithmes contiennent ceux de la vision, de l'audition, de la production des mouvements, de la parole, et bien d'autres sur lesquels nous avons beaucoup moins d'indications. Les langages internes peuvent être vus comme les diverses formes d'organisation spatiale et temporelle des potentiels d'action qui semblent exister dans le cerveau, mais qui sont encore mal comprises (la fréquence des pics est certainement utile, mais la synchronisation entre neurones probablement aussi²⁰). Enfin, la machine est formée du cerveau et d'autres structures avec lesquelles il coopère pour la transmission de l'influx nerveux depuis les capteurs vers les

actionneurs. De plus, la croissance du cerveau est elle-même dirigée par les algorithmes de la morphogenèse, d'où la flèche bleue de la figure 64.

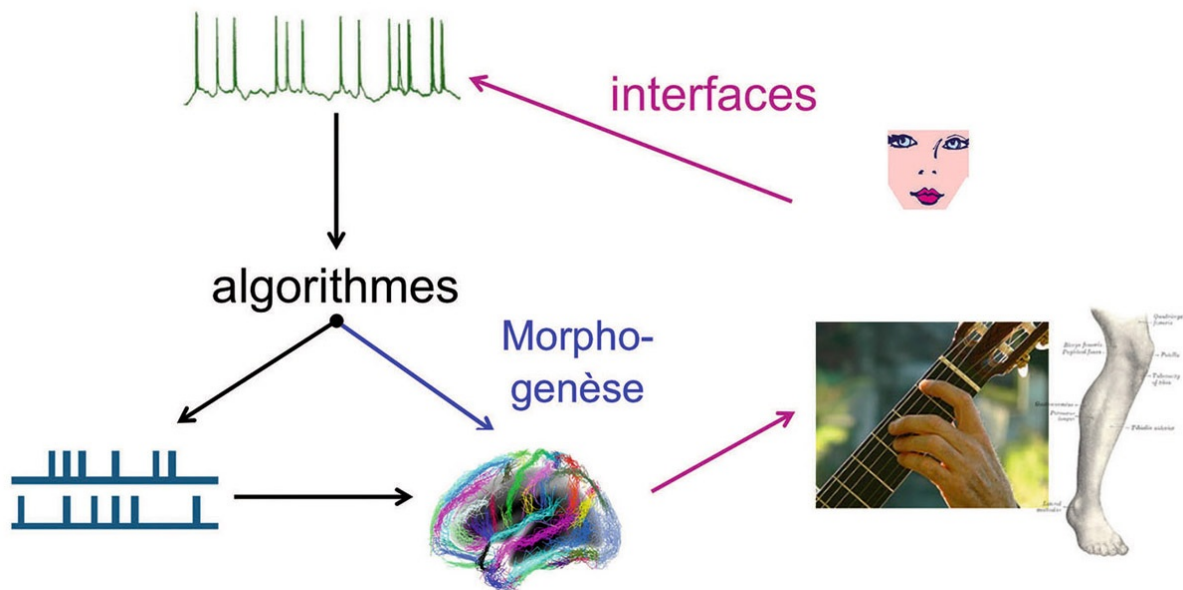


Figure 64. Les piliers des neurosciences.

La machinerie interne de notre cerveau, éminemment biologique, est très différente de celle des machines à information que nous avons nous-mêmes construites, nos ordinateurs. Le cerveau comporte en effet environ 80 milliards de neurones, chacun étant connecté en moyenne à environ 20 000 autres par des contacts biochimiques nommés synapses. Le parallélisme y est donc hypermassif. Les entrées de chaque neurone se font par de très nombreuses synapses, la sortie par un seul axone qui se ramifie en de très nombreuses dendrites connectées à beaucoup d'autres neurones par d'autres synapses. En simplifiant son fonctionnement, on peut dire que le neurone est un dispositif intégrateur : si les excitations de ses synapses d'entrée sont suffisamment nombreuses, puissantes et rapprochées, il se dépolarise électriquement en produisant un potentiel d'action qui va parcourir l'axone vers les dendrites. Le corps neuronal se repolarise ensuite, ce qui prend un certain temps pendant lequel le neurone ne peut pas se dépolariser.

Les synapses sont elles-mêmes des dispositifs à hystérésis, car une excitation y provoque une réponse décroissante dans le temps, et elles ont une mémoire locale : la façon dont la synapse transmet l'information entrante au corps du neurone receveur peut être modulée par l'activité passée, selon ce qu'on appelle la *plasticité synaptique*. Ce mécanisme est essentiel pour tous les apprentissages et probablement bien d'autres fonctions, y compris celles de la mémorisation d'informations complexes. Enfin, il faut ajouter que les potentiels d'action sont transmis par les axones à une vitesse allant de quelques mètres par seconde à environ 100 mètres par seconde dans les axones bien myélinisés ; c'est lent, mais le cerveau se débrouille visiblement très bien avec ça, beaucoup mieux même que nos algorithmes sur ordinateurs pour la plupart des problèmes de la vie courante.

Le cerveau est donc complètement différent de nos circuits électroniques dans ses composants et dans sa microarchitecture (c'est moins clair pour son architecture globale). Certes, nous atteindrons probablement aussi les dizaines de milliards de transistors sur un circuit, et certes la communication y est bien plus rapide que celle des neurones, puisqu'elle se fait à la vitesse de la lumière dans le métal. Mais un transistor d'un circuit est vraiment un objet rudimentaire par rapport à un neurone : il fonctionne en tout ou rien, sans aucune mémoire interne, et n'est relié qu'à un tout petit nombre d'autres transistors. Sur le plan temporel, le fonctionnement d'un circuit est brutalement cadencé par des horloges très rapides, alors que celui du cerveau est probablement beaucoup plus évolué, que ce soit par l'exploitation des relations temporelles entre potentiels d'action ou par l'usage qu'il fait d'une batterie d'oscillateurs lents (dits ondes alpha, bêta, gamma, thêta, etc.) dont le rôle précis est encore mystérieux. Ensuite, l'architecture des ordinateurs fait que la mémoire et le calcul ne sont que très peu imbriqués, alors que l'inverse semble vrai dans le cerveau. Enfin, le cerveau travaille dans un environnement très bruyant et n'est probablement pas intéressé par les calculs hyperfins comme les multiplications de grande taille que nous faisons faire à nos ordinateurs. L'essentiel pour lui est que l'organisme qu'il dirige se comporte plutôt bien dans un grand nombre de circonstances en fonction de données peu précises qui lui servent à maintenir une cohérence dans la compréhension des mondes extérieurs et intérieurs.

En résumé, même si les neurones et les transistors peuvent conduire tous deux à des machines universelles et peuvent s'avérer équivalents quant à leurs possibilités

théoriques, leurs mises en pratique sont vraiment très différentes. Attention, donc, aux comparaisons naïves entre les deux systèmes...

PROFITER DE LA GÉOGRAPHIE DU CERVEAU

La lenteur de transmission de l'influx nerveux a pour les neurosciences un grand avantage : pour que les performances globales restent acceptables, elle a imposé au cours de la sélection naturelle que chaque fonction majeure soit regroupée dans une structure occupant un espace géographique restreint du cerveau. Cela permet d'observer assez facilement l'activation de ces fonctions et les communications entre elles, soit par l'IRM fonctionnelle ([Le Bihan, 2012](#)), l'EEG (électroencéphalographie), la MEG (magnétoencéphalographie), ou encore l'implantation directe d'électrodes, surtout pour les animaux. Attention : comme nous l'avons vu au [chapitre 7](#) pour l'IRM, l'analyse des signaux EEG, MEG ou recueillis par des électrodes demande des calculs très complexes, surtout si l'on veut savoir finement d'où partent ces signaux. Mais il s'agit d'instrumentation fine, pas de compréhension des mécanismes informationnels dans le cerveau. Je n'en parlerai donc pas davantage car je l'ai déjà fait pour d'autres sciences.

On a pu ainsi étudier l'enchaînement des opérations dans des activités soit de bas niveau comme des réponses à des stimuli soit de haut niveau comme la lecture ([Dehaene, 2007](#)), et même plus récemment ce qui se passe lors de la prise de conscience d'une information ([Dehaene, 2014](#)). On a pu aussi observer pendant le sommeil des rats la transmission rapide d'informations entre l'hippocampe, siège de la mémoire du jour, et le cortex, siège supposé de la mémoire épisodique (permanente). Enfin, on a pu observer que le cerveau est un grand amateur d'oscillations temporelles, qui lui servent probablement à coordonner ses fonctions et à transmettre plus facilement de l'information entre zones²¹.

COMMENT ABORDER LES ASPECTS INFORMATIONNELS DE L'ACTIVITÉ CÉRÉBRALE ?

À un extrême du spectre, les mécanismes biochimiques de la dépolarisation et de la repolarisation reposant sur les transferts rapides d'ions par les canaux ioniques du neurone ont été étudiés en très grand détail, et les mécanismes internes des synapses sont de mieux en mieux compris. À l'autre extrême, voir en direct le cerveau travailler nous a beaucoup appris sur son fonctionnement global. Toutes ces recherches sont bien sûr essentielles pour la compréhension biologique du système nerveux et de ses maladies.

Mais, vues par un informaticien naïf comme moi, bien d'autres questions de nature moins directement biologiques méritent d'être étudiées. On peut les poser je pense à trois niveaux : la représentation de l'information, la microarchitecture, c'est-à-dire la façon d'organiser les calculs dans les structures fonctionnelles locales, et l'architecture, c'est-à-dire la façon de répartir les calculs. Les études précédemment décrites montrent bien que les neurones sont groupés en colonnes neuronales cohérentes vis-à-vis d'un but recherché (éléments microarchitecturaux), et que le cerveau est organisé en structures fonctionnelles communicantes et reliées par le grand réseau neuronal de la substance blanche. C'est déjà un élément important d'architecture, pas très surprenant pour un informaticien qui a été naturellement conduit à une architecture similaire pour ses ordinateurs, sauf pour la mémoire qui reste séparée du calcul dans les ordinateurs et un mystère pour la biologie. Mais nous en savons beaucoup moins sur la représentation et la mémorisation de l'information, ainsi que sur la façon dont les calculs sont effectués, non pas au niveau microscopique, mais dans les processus de haut niveau comme la vision, l'audition ou la pensée. Les questions sont donc nombreuses. En voici quelques-unes ci-dessous.

Quels sont les principes informationnels utilisés pour le codage de l'information par les trains de potentiels d'action, et quels sont les rôles du temps dans ce codage ? (J'ai mis le pluriel car il y en a certainement plusieurs, qui peuvent profiter du fait que laisser passer le temps est la seule chose presque gratuite en ce bas monde.) Ce codage est-il très global ou au contraire localement adapté au type d'information manipulé ? Fournit-il plus de souplesse que les codages utilisés dans nos circuits et programmes, et, si oui, en quel sens ? Comment est organisée la mémoire persistante et comment se relie-t-elle à la transmission des informations par potentiels d'action ? Quand est-elle intégrée aux dispositifs de calcul, ou à l'inverse réalisée par des entités spécifiques ? Comment sont représentées les informations manipulées dans les

structures locales du cerveau, en particulier en ce qui concerne les interactions entre données fugitives et mémoire ? Comment est organisée la tolérance aux erreurs de toutes sortes ? Quel est le rôle précis de chaque structure fonctionnelle en termes d'information transformée ou mémorisée ? Quel est le rôle précis des oscillateurs temporels dans la synchronisation locale ou globale des traitements d'information ? Pourquoi la microarchitecture et l'architecture neuronales permettent-elles au cerveau de travailler si vite et si efficacement à partir d'une infrastructure biologique aussi lente, tout en dépensant aussi peu d'énergie ? Comment le parallélisme massif est-il exploité pour compenser cette lenteur ? Comment le cerveau représente-t-il la réalité, ou du moins ce qu'il en perçoit ? Comment maintient-il une cohérence dans cet ensemble d'informations, et pourquoi le fait-il si souvent à tout prix ? Qu'est-ce qui le rend capable d'imagination ?

Il est clair que le niveau actuel des connaissances en neurosciences computationnelles (surtout le mien) ne permet pas de répondre à toutes ces questions. Je ne connais pas assez les grands projets du domaine comme le Human Brain Project européen²² pour savoir où les chercheurs en sont par rapport à elles. Mais les poser me semble intéressant, et j'aimerais bien personnellement disposer d'une ou deux réincarnations pour les travailler : ce champ d'investigation constituera certainement un paradis pour les chercheurs du XXI^e siècle.

Comme elles demandent d'une certaine façon de mettre au second plan les mécanismes fins de la biologie en s'appuyant sur un schéma mental plus informationnel, il est naturel que ces questions ne suscitent pas toujours un intérêt direct chez beaucoup de biologistes. Je voudrais cependant faire une remarque : c'est déjà un peu comme ça pour les circuits. Un physicien fondamental voit un transistor comme un beau dispositif quantique, alors qu'un électronicien le voit comme un amplificateur d'un signal faible dont on connaît la courbe de réponse. Mais, pour un informaticien, aucune de ces approches n'est vraiment pertinente : il lui suffit de voir le transistor comme un simple interrupteur fonctionnant bêtement par stabilisation soit près de 0V, soit près de 1V quand il est placé dans de bonnes conditions par l'architecture globale, ce fonctionnement électriquement dégradé étant idéal pour le calcul binaire. Les schémas mentaux du physicien, de l'électronicien et de l'informaticien sont donc très différents, ce qui ne veut en aucun cas dire que chacun a

raison ou a tort. Les problèmes à traiter ne sont pas les mêmes, les objectifs non plus, et la coopération est possible et souhaitable.

En fait, on peut même être plus brutal : sauf pour satisfaire sa curiosité, pour un informaticien, il est important de *ne pas comprendre* comment fonctionne réellement un transistor, dispositif analogique qui n'est pas du tout utilisé analogiquement dans un circuit. Trop se focaliser sur le transistor le conduirait inmanquablement à rater tous les aspects importants de l'architecture du calcul par circuits. La séparation physique/logique permise par les prouesses des électroniciens capables de nous calculer le temps de stabilisation électrique d'un circuit a été déterminante pour l'essor de l'informatique moderne, car elle a permis à ceux qui écrivent les logiciels d'oublier le fonctionnement interne des ordinateurs.

Les niveaux d'abstraction informationnels à utiliser pour le cerveau paraissent certainement bien plus complexes, mais pas forcément inaccessibles. Comme me l'a appris le physicien Jean-Marc Lévy-Leblond : « La science est l'art de simplifier les problèmes jusqu'à ce qu'ils deviennent solubles. »

Enfin, je voudrais dire mon scepticisme par rapport à une approche de pure réingénierie (*reverse engineering*) qui essaierait de remonter du fonctionnement des neurones à celui du cerveau. Même pour un objet bien plus simple comme un microprocesseur, le *reverse engineering* à partir des plans d'impression du circuit est devenu depuis bien longtemps pratiquement impossible si on ne connaît pas d'avance l'architecture et la microarchitecture. Pour le cerveau, l'utiliser seul me paraît aussi illusoire ; le progrès ne pourra exister que si l'on combine harmonieusement l'ensemble des approches allant du bas niveau au haut niveau et inversement. Mais cette réflexion n'engage bien entendu que moi.

RÉSEAUX DE NEURONES ET MÉMOIRES BIO-INSPIRÉES

Une approche bio-inspirée mais résolument informationnelle possède un autre avantage : elle permet d'aborder la question de la construction de nouvelles machines électroniques ou autres reposant sur des microarchitectures très différentes de celle brutalement binaire des ordinateurs. On peut déjà citer deux exemples de progrès

significatifs : celui des réseaux de neurones profonds déjà présenté dans ce livre, où le mot « neurone » est pour l’instant à prendre comme une approximation basique des neurones biologiques, et celui des structures de mémoires neuronales développées par Claude Berrou et Vincent Gripon à Télécom Brest, relaté récemment dans un livre de vulgarisation ([Berrou & Gripon, 2012](#)), et bien sûr plus profondément expliqué dans des articles scientifiques. Dans le [chapitre 4](#), j’ai déjà mentionné Berrou comme étant l’inventeur des turbocodes, qui ont révolutionné la correction d’erreurs et donc la transmission en télécommunications. Il est évidemment essentiel que des chercheurs majeurs en théorie de l’information se penchent sur la question.

Pour faire rapide, je dirai simplement que leur modèle de mémoire, appelé *sparse connected network* (SCN), est fondé sur des réseaux simplement binaires, dans lesquels une connexion existe ou pas. Les éléments insécables d’information sont représentés par des points (neurones) regroupés en groupes disjoints. On aura par exemple les groupes *grand*, *moyen* et *petit* pour les tailles, *bleu*, *rouge* et *vert* pour les couleurs, *triangle*, *carré* ou *rond* pour les formes. L’information « petit et bleu » est représentée par la connexion *petit-bleu*, alors que l’information « grand, bleu et rond » est représentée par les trois connexions *grand-bleu*, *grand-rond*, et *bleu-rond*. Ce modèle apparemment naïf a des propriétés mathématiques étonnantes. D’abord, le nombre d’informations qu’il peut stocker est proportionnel au *carré* du nombre de neurones, pas au nombre de neurones. Avec quelques milliards de neurones, cela permet de mémoriser énormément d’informations. Ensuite, c’est une *mémoire associative*, c’est-à-dire qui a la propriété de permettre de retrouver une information à partir d’une de ses parties : à partir de *bleu* on retrouve « petit et bleu » et « grand, bleu et rond », alors qu’à partir de *bleu* et *rond* on ne retrouve que « grand, bleu et rond » ; il permet de plus une navigation facile entre informations reliées entre elles : *bleu* permet de passer facilement de « petit et bleu » à « grand, bleu et rond ». Il est aussi facile d’ajouter des « neurones grands-mères » réagissant spécifiquement à une information bien précise, comme l’a montré une expérience réelle sur un patient dont un neurone s’allumait explicitement sur n’importe quelle évocation de sa grand-mère. De plus, cette mémoire réalise un *code correcteur d’erreurs* très efficace : couper quelques connexions ne change pas grand-chose à l’information mémorisée. Enfin, et ce n’est pas le moindre apport, les simulations montrent qu’une implémentation

neuronale peut être particulièrement efficace, bien plus qu'une implémentation naïve sur ordinateur classique.

Voici des caractéristiques que notre mémoire biologique possède, mais en aucun cas nos mémoires d'ordinateurs. J'ajouterai pour terminer que cette mémoire très simple est fortement liée à des structures mathématiques générales très étudiées qu'on appelle les *complexes simpliciaux*, déjà mentionnés sous une autre forme par Alain Connes comme probablement reliés à notre mémoire dans son livre, *Matière à pensée*, de dialogue avec Jean-Pierre Changeux ([Changeux & Connes, 1989](#)). Mais il n'est certainement pas encore clair que ce genre d'approche nous en apprendra sur notre propre mémoire, quoique rien ne paraisse exclu dans ce domaine totalement ouvert.

L'informatique en mathématiques

Quittons les sciences de la nature pour rejoindre les mathématiques. Comme nous l'avons vu au [chapitre 2](#), l'informatique et les mathématiques entretiennent une relation profonde depuis leurs naissances réciproques. Mais, au cours du xx^e siècle, cette relation a été quelque peu asymétrique au moins sur le plan social : les mathématiciens reconnaissaient que les mathématiques apportaient beaucoup à l'informatique, mais étaient beaucoup moins enclins à reconnaître l'inverse, en aidant par exemple l'informatique à se faire une vraie place dans l'enseignement. L'informatique restait considérée comme un outil : les mathématiciens appliqués concevaient leurs algorithmes puis les faisaient programmer par des ingénieurs, généralement en FORTRAN, langage très ancien qui n'a que timidement incorporé les avancées de l'informatique moderne, puis les faisaient exécuter sur des superordinateurs. Mais cette perception change sensiblement au xxi^e siècle, comme nous allons le voir. Commençons cependant par deux avancées importantes dues à l'informatique du xx^e siècle.

TEX ET LATEX, LES LANGAGES UNIVERSELS DE L'ÉCRITURE MATHÉMATIQUE

Le langage mathématique utilise systématiquement les lettres romaines grecques et cursives, ainsi que beaucoup de symboles cabalistiques variés, comme $\forall \exists \emptyset \vdash \perp \subset \supset \cap \cup \wedge \vee$ et $\leq \geq \infty \odot \Delta \nabla \int \sqrt{\sqrt[3]{}}$, tout cela incorporé dans de grandes formules,

matrices et diagrammes qui ornent élégamment tous les livres mathématiques. L'esthétique de ces formules n'est pas un gadget. Elle résulte d'une longue tradition, et est fondamentale pour la facilité de lecture et de compréhension. Les mathématiciens l'ont donc toujours soignée. Mais elle induisait toujours une difficulté pour les imprimeurs : comment écrire ces formules avec du plomb ?

Dans les années 1970, les informaticiens ont compris que les textes et formules mathématiques ne devaient plus être composés au plomb mais calculés algorithmiquement et imprimés par des linotypes ou des imprimantes. Le premier logiciel de composition informatique convaincant a été Troff, développé au Bell Labs (encore !) peu après la naissance d'Unix. Mais la vraie rupture eut lieu en 1978 avec le système TeX développé par Donald Knuth à Stanford (prix Turing 1974, mais pas pour ça), suivi en 1983 par sa simplification LaTeX par Leslie Lamport (prix Turing 2013, mais pas pour ça), aussi à Stanford. Knuth est un immense informaticien révérend de tous. Son livre multivolume *The Art of Computer Programming*, qu'il continue à écrire avec un enthousiasme jamais démenti, ne peut être qualifié que par un seul mot : *incroyable*. Juste en exemple : en 2016, je préparais mon cours sur les merveilleux algorithmes SAT, sur lesquels je reviendrai au [chapitre 10](#), quand je reçus son volume 4B, fascicule 6, entièrement consacré à ce problème en rassemblant toute la connaissance dessus et bien plus. Sa grosse centaine de pages d'exercices consterne même les spécialistes du sujet, qui pourraient passer des années à les essayer sans les réussir tous !

Knuth trouvait que les techniques d'impression de l'époque n'étaient pas assez souples pour ses besoins et que leurs résultats n'étaient pas assez beaux. C'est pour cela qu'il a développé TeX. Il s'agit d'un système de composition originellement dédié aux mathématiques, mais qui permet aussi d'éditer toutes sortes de documents techniques. TeX est fondé sur un langage de programmation puissant mais complexe à l'usage. Lamport l'a simplifié en introduisant LaTeX, qui est une couche conviviale au-dessus de TeX. TeX et LaTeX sont complétés par des outils associés comme des éditeurs/visualiseurs et BibTeX pour la bibliographie. Les résultats qu'ils donnent sont d'une beauté sans rivale, Knuth ayant soigné au millipoil près tous les détails fins de la composition. Comme la grande majorité des articles ou livres possédant des formules non triviales les utilise, en mathématiques, en physique, et dans bien d'autres sciences, ces logiciels sont sans doute devenus les plus importants outils de

communication scientifique. De plus, malgré la récompense de 1 dollar promise par Knuth pour chaque bug²³, plus personne n'arrive à en trouver dans son logiciel, pourtant particulièrement complexe ! Ces choses sont suffisamment rares pour être saluées bien bas.

LE CALCUL EN PRÉCISION ARBITRAIRE

Le calcul numérique classique s'effectue avec des nombres de taille limitée. Cette taille est généralement de 32 ou 64 bits pour les nombres entiers, et de 32, 64 ou 128 bits pour les nombres flottants qui servent à approximer les réels (voir le [glossaire](#)). En théorie des nombres et en chiffrement de données, les mathématiciens sont habitués à travailler avec de très grands nombres entiers. C'est par exemple en étudiant les nombres premiers jumeaux 824 633 702 441 et 824 633 702 443 que le mathématicien américain Thomas Nicely a mis en évidence le fameux bug de division du Pentium en 1991, dont nous reparlerons au [chapitre 9](#) : il s'est aperçu que la division flottante $1/824\,633\,702\,441$ rendait un résultat faux. Ces nombres n'étaient pas là par hasard, mais parce qu'ils étaient premiers jumeaux (séparés seulement de deux unités). Pour le domaine, ils ne sont pas très grands : ils ne tiennent pas sur 32 bits, mais sur 64. C'est peu par rapport à d'autres problèmes. Par exemple, dans le fameux protocole RSA utilisé pour la carte bleue et l'établissement des connexions sécurisées sur Internet, la clef publique de chiffrement a la forme d'un très grand produit de deux grands nombres entiers premiers, ce produit demandant typiquement 2 048 ou 4 096 bits. Chiffrer ou déchiffrer un message se fait en opérant sur de tels nombres, et le chiffrement n'est sûr que parce qu'on pense que les adversaires ne peuvent pas retrouver les deux nombres premiers en connaissant seulement leur produit ; ce n'est pas une certitude, mais une conjecture mathématique. Et l'on sait que les clefs plus courtes, par exemple de 1 024 bits, peuvent maintenant être cassées, comme l'a montré l'équipe de Patrick Gaudry à l'Inria Nancy.

Pour ce type d'applications, il faut disposer de bibliothèques de calcul sur des nombres de très grande taille. Suivant les applications, cette taille peut être soit fixe, soit variable, alors seulement limitée par les ressources de sa machine. D'excellentes bibliothèques sont disponibles dans le domaine public.

Les nombres peuvent aussi être rationnels (paires d'un numérateur et d'un dénominateur) ou flottants. Ces derniers sont composés d'une mantisse m et d'un exposant e , tous deux positifs ou négatifs. Ils permettent d'exprimer une très grande plage de dynamique, le flottant (m,e) représentant le nombre $m \times 2^e$. Mais, là aussi, il peut arriver que m,e , ou les deux deviennent très grands, voire de taille arbitraire. C'est le cas par exemple dans l'étude de grands systèmes dynamiques comme le système solaire : notre bonne vieille Terre va-t-elle rester bien en paix sur son orbite, ou au contraire va-t-elle être éjectée du système solaire ou encore entrer en collision avec Mars ou Vénus avant même l'explosion du soleil ? De tels calculs ne peuvent être conduits avec les nombres informatiques classiques.

De même, dans la superbe discipline qu'est la géométrie algorithmique ([Boissonnat, 2017](#)), il faut souvent travailler avec de très grands nombres flottants. Par exemple, les algorithmes géométriques ont très souvent comme entrée un grand nuage de points qu'il s'agit de trianguler. Souvent, les algorithmes considèrent trois points A, B et C d'un nuage de points donnés. Il leur faut typiquement décider si C est à gauche ou à droite de la droite orientée AB, cette décision faisant une grande différence pour le reste des calculs. Dans ce cas, il est impératif d'être constamment vigilant sur la taille des nombres à utiliser à chaque étape pour ne pas fausser les résultats tout en restant en taille fixe pour l'efficacité. Un des problèmes centraux du domaine est de déterminer la bonne taille.

LE CALCUL SYMBOLIQUE

Mais les mathématiques ne se contentent pas de calculs numériques. Les calculs symboliques sur des variables, des polynômes, des formules analytiques, des matrices, etc., y sont bien plus fréquents. Beaucoup d'entre eux sont décrits par des règles algorithmiques : multiplication de polynômes ou de matrices, dérivation et intégration de fonctions formelles définies par des formules explicites, calculs de racines de polynômes de degré 2 ou 3, calculs trigonométriques, etc. Il est particulièrement utile de réaliser par logiciel ces calculs fastidieux et propices aux erreurs, surtout si les formules sont grandes. Les logiciels de calcul symbolique sont développés depuis longtemps, dont deux très connus sont Maple et Mathematica. Alain Connes,

professeur au Collège de France dont beaucoup de travaux sont consacrés aux algèbres non commutatives associées à la mécanique quantique, me dit qu'il ne pourrait plus se passer de Mathematica, ses calculs n'étant plus faisables à la main. Les logiciels de calcul formel implémentent aussi des algorithmes de simplification, cruciaux en pratique mais toujours difficiles, et bien sûr les algorithmes de calcul numérique en précision arbitraire dont nous venons de parler. Le calcul symbolique a aussi des impacts importants en informatique. Par exemple, mon collègue Philippe Flajolet a initié leur utilisation pour calculer les performances d'algorithmes divers de routage de paquets dans les réseaux, en s'appuyant sur des calculs de séries de Taylor ou autres et en utilisant à fond l'analyse complexe développée au XIX^e siècle.

THÉORÈMES, PREUVES, ET VÉRITÉ

Les mathématiciens sont connus comme des champions de rigueur : ils considèrent qu'un théorème est vrai non pas parce qu'ils en ont envie, mais parce qu'ils l'ont *démontré*, c'est-à-dire qu'ils en ont donné une *preuve* (preuve et démonstration sont synonymes en maths). Mais qu'est-ce donc qu'une preuve ? Selon les logiciens, une preuve formelle doit être une liste finie de formules écrites dans le langage formel d'une théorie mathématique donnée, vérifiant la propriété suivante : toute formule de la liste est soit un *axiome*, c'est-à-dire une formule déclarée vraie dès la définition de la théorie, soit une formule *dérivée*, c'est-à-dire qu'on peut construire à partir de formules précédemment prouvées en appliquant une *règle de déduction* formelle de la théorie. La plus célèbre règle de déduction est *modus ponens* : si on a montré A et $A \Rightarrow B$, lu « A implique B » ou « A entraîne B », alors B est prouvé. Mais il peut en exister bien d'autres dans chaque théorie particulière. Les formules et les règles de déduction sont elles-mêmes définies précisément : on exige la présence d'un algorithme décidant en temps fini si une formule est bien écrite ou non, ainsi que d'un algorithme permettant pour chaque règle de déduction de calculer une nouvelle formule pour chaque liste de formules d'entrée. Cela veut dire que, au contraire de son invention qui demande de l'intelligence, *la vérification d'une preuve ne demande aucune intelligence et peut être faite en principe par un algorithme, et donc en utilisant un ordinateur.*

Même si les mathématiciens connaissent cette définition et s'accordent à la trouver adéquate, ils ne l'utilisent pas vraiment. Vis-à-vis de leurs objectifs, ils ont bien raison. D'abord, ce qui les intéresse le plus, ce sont les objets abstraits qu'ils manipulent et qu'ils voient comme associés à une notion intrinsèque de vérité, comme les nombres entiers, réels ou complexes. Les formules qui en parlent sont vues comme des écritures précises et élégantes pour exprimer leurs propriétés et calculer avec eux plutôt que comme des concepts d'intérêt primordial en eux-mêmes. Ensuite, les preuves vraiment formelles s'avèrent d'une taille et d'un ennui absolument insupportables pour un humain normal. Ceux qui ont essayé de s'y cantonner, comme Bertrand Russell et Alfred Whitehead dans leur *Principia Mathematica*, s'en sont vite rendu compte à leurs dépens.

Pour parler des objets abstraits qu'ils manipulent et écrire leurs preuves commentées, les mathématiciens utilisent un mélange d'écriture formelle et d'expressions de la langue parlée, éventuellement fleuri par un vocabulaire coloré. Ils font relire ces preuves par d'autres mathématiciens dans un processus de relecture qu'ils prennent très au sérieux. Mais, même s'il est de très grande qualité, ce processus n'est pas infallible : il n'est pas rare qu'on trouve ensuite soit que la preuve était un peu incorrecte, voire franchement incorrecte, soit que l'énoncé du théorème devait être remanié pour y ajouter des conditions supplémentaires qui peuvent elles-mêmes être créatrices de nouvelles notions. Par exemple, le célèbre théorème d'Euler exprime une relation fondamentale entre le nombre de sommets d'un polyèdre et son nombre de faces. Peu après sa démonstration, un contre-exemple a été trouvé – à l'époque, on appelait cela une exception. Le théorème a été remis droit en ajoutant l'hypothèse que le polyèdre devait être convexe, ce qui a permis d'introduire la notion encore plus fondamentale de *convexité*²⁴ en géométrie. C'est la vie du domaine, qui reste quand même de loin le plus rigoureux parmi toutes les sciences.

Mais, si les preuves sont vérifiables par des algorithmes, maintenant qu'on a des ordinateurs puissants, pourquoi ne pas les faire vérifier, voire construire par ces machines, et si possible automatiquement ? On sera alors certain que tous les théorèmes démontrés par cette méthode sont justes, à condition bien sûr de s'assurer une fois pour toutes que les fondements logiques et l'algorithme de vérification soient justes. Et, avec cette méthode, pourra-t-on tout savoir en mathématiques ?

Les réponses à ces deux questions sont négatives. Comme nous l'avons vu au [chapitre 2](#), l'*Entscheidungsproblem* de David Hilbert était la question de savoir s'il existe un algorithme général pour prouver les théorèmes mathématiques. Elle a été résolue négativement par Turing dans son papier fondateur de 1936, quand il a montré l'impossibilité de prouver algorithmiquement à coup sûr qu'une machine de Turing travaillant sur des données spécifiées va s'arrêter ou non. Et les théorèmes de Gödel, datant de 1931, montrent de plus l'insuffisance intrinsèque de la notion de démonstration par rapport à la notion de vérité mathématique : son premier théorème exprime que toute axiomatisation de l'arithmétique (c'est-à-dire du simple calcul des nombres entiers) est incomplète dans le sens qu'il existe toujours un théorème vrai mais non démontrable dans cette axiomatique ; son second théorème montre qu'aucune théorie cohérente qui contient l'arithmétique ne peut démontrer sa propre cohérence.

Je laisse ces énoncés dans cet état pas très formel, d'autant plus que je n'ai même pas défini précisément la notion de vérité d'une formule. C'est délicat, et le faire m'entraînerait trop loin pour les besoins de ce livre. Attention cependant à ne pas partir dans les discussions philosophiques sur les limites de la connaissance humaine : les théorèmes de Gödel ne font que comparer la notion technique de vérité dans une théorie mathématique à celle encore plus technique de démonstration, cela en n'utilisant que des méthodes de réécriture de ces formules. Ils ne posent pas pour autant de limites à l'intelligence humaine.

Je souhaite cependant donner une idée intuitive de la façon dont le premier théorème de Gödel se démontre. La seule relation qu'on exige quand on compare démonstration et vérité est qu'une formule démontrable soit vraie. Sinon, ce ne serait même pas la peine de se fatiguer. Gödel construit explicitement une formule qui encode l'information : « Je ne suis pas démontrable en arithmétique avec cette axiomatique ». Si elle était fausse, elle serait démontrable, et donc vraie, ce qui est impossible car une formule ne peut être à la fois vraie et fausse. Donc elle est vraie, et elle répond par là même à la question²⁵. CQFD !

LA PREUVE EN MACHINE

Même s'ils posent des limites à la mécanisation des preuves, les théorèmes précédents ne doivent pas nous impressionner outre mesure, car ils ne disent pas que la preuve en machine de théorèmes est inaccessible dans la pratique et les applications, les contre-exemples étant artificiels et sans conséquence pratique réelle. Les chercheurs en intelligence artificielle se sont donc attelés très tôt à la preuve semi-automatique, mais avec de petits exemples dans le cadre de théories simples. Par exemple, des algorithmes effectifs ont permis de prouver les problèmes de géométrie plane ou de résoudre des problèmes intéressants dans des champs bien délimités. Nous en reparlerons au [chapitre 10](#) car ces systèmes sont devenus importants pour la vérification de programmes.

Un préliminaire indispensable à toute automatisation est de bien comprendre comment vérifier les phrases purement logiques, éventuellement liées à une théorie particulière. Des algorithmes ont vite été développés pour le calcul booléen (*vrai*, *faux*, *et*, *ou*, *non*) et pour des formes restreintes du calcul des prédicats, où les quantificateurs $\forall$ (quel que soit) et $\exists$ (il existe) s'appliquent aux objets mais pas aux fonctions ni aux prédicats eux-mêmes ; c'est d'ailleurs le calcul des prédicats qui fonde le langage Prolog d'Alain Colmerauer, précurseur des langages à contraintes où l'on décrit le résultat souhaité mais pas comment le calculer.

Mais ces systèmes étaient automatiques, ce qui les limitait considérablement car même le simple calcul des prédicats n'est pas décidable algorithmiquement. C'est en étendant le λ -calcul de Church que le logicien Nicolaas de Bruijn a attaqué le problème général de la formalisation des mathématiques. Il a créé le premier assistant interactif de preuves mathématiques nommé Automath, précurseur des assistants modernes comme HOL (Higher Order Logic) de Michael Gordon, Isabelle de Lawrence Paulson et Coq, dont nous allons reparler tout de suite.

Nous étudierons tout cela plus en profondeur au [chapitre 10](#), en montrant les extraordinaires progrès de ces outils et leurs applications à la sûreté et la sécurité des systèmes informatiques. Mais décrivons maintenant les très grandes preuves de théorèmes importants réalisées récemment avec les assistants de preuve modernes.

LE THÉORÈME DES 4 COULEURS

La conjecture des 4 couleurs exprime que toute carte de géographie peut se colorier avec 4 couleurs de façon à ce que deux régions qui se touchent autrement que sur un coin soient de couleur différente. Cette conjecture n'est pas mathématiquement riche de conséquences, mais la difficulté de sa preuve a toujours intrigué les mathématiciens. Les tentatives successives ont eu une longue histoire, que je simplifie ici.

Le problème a été posé en 1852 par Francis Guthrie, qui s'intéressait au coloriage de la carte de l'Angleterre. En 1879, Alfred Kempe a publié une première preuve. En 1890, cette preuve a été montrée fausse pour 4 couleurs mais correcte pour 5 couleurs. En 1976, Kenneth Appel et Wolfgang Haken ont publié une nouvelle preuve bien plus évoluée, sous la forme de deux parties distinctes : d'abord, un article mathématique montrant qu'on pouvait se ramener à l'étude d'un nombre fini de cas, précisément 1 478 ; ensuite, la vérification sur ordinateur que toutes ces cartes étaient effectivement coloriables avec 4 couleurs. L'article fut publié après une relecture sérieuse, mais les mathématiciens étaient partagés sur la validité de la vérification informatique en tant que preuve mathématique.

Dans les années 2000, Georges Gonthier²⁶ (Inria) et Benjamin Werner (École polytechnique) décidèrent de mécaniser la preuve en s'appuyant sur l'assistant de vérification Coq développé en France à partir de la théorie des constructions de Gérard Huet et Thierry Coquand (Inria), étendue par Christine Paulin (université Paris-Sud). Cette théorie est issue à la fois des idées d'Automath et d'un grand théorème de Jean-Yves Girard²⁷. Outre sa parfaite rigueur logique, Coq intègre un langage de programmation évolué qui facilite la manipulation de structure de données riches au cours de la preuve, ainsi qu'un langage de tactiques pour conduire les preuves et en automatiser certaines parties. Après un travail intensif et pas mal de surprises, la preuve fut simplifiée et terminée par Gonthier et Werner en 2005. Elle fit sensation car sa mécanisation n'avait aucune mesure avec ce qu'on pensait réalisable dans le domaine.

Après les 4 couleurs, Gonthier chercha un problème bien plus important mathématiquement qu'il pourrait prouver en machine. Son choix se porta sur le redoutable théorème de Feit-Thompson, qui concerne la classification des groupes finis d'ordre impair, c'est-à-dire ayant un nombre fini d'éléments. Le théorème s'écrit en quelques lignes, mais la preuve de Walter Feit et John Thompson est un monument du xx^e siècle : initialement publiée en 1963, elle fut révisée sur des points locaux jusqu'en 1992. Divisée en six chapitres, elle fait plus de 250 pages du *Pacific Journal of Mathematics*. Après plusieurs années de travail de Gonthier en équipe avec d'autres informaticiens, la preuve est maintenant complètement en machine, écrite et vérifiée avec Coq. Sa taille est du même ordre que la preuve originale, qui a été suivie presque à la lettre, et où seules quelques coquilles mineures ont été détectées.

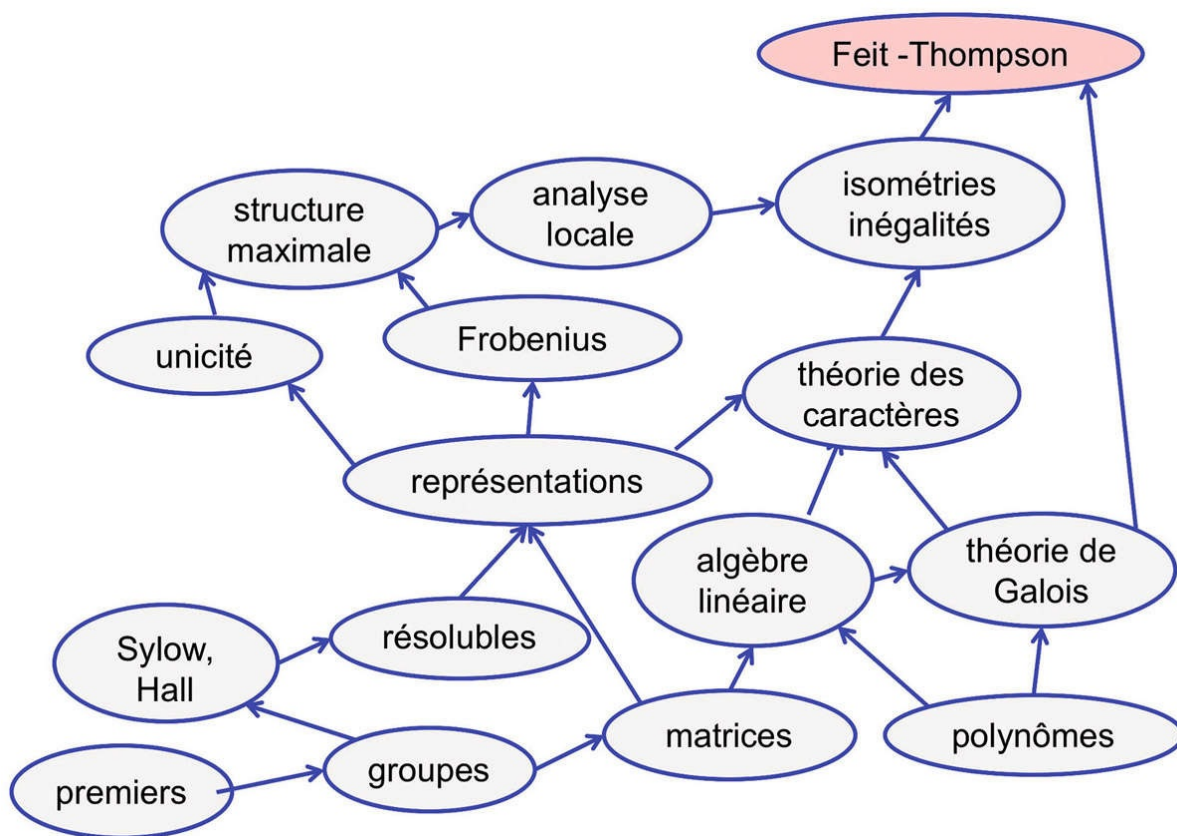


Figure 66. Architecture des théories pour Feit-Thompson (source G. Gonthier).

L'intérêt n'est pas seulement dans la redémonstration du théorème. Il est surtout dans l'architecture informatique qui a permis de rendre cette preuve faisable et la garder compacte relativement à la preuve humaine. Comme illustré dans la figure 66, cette architecture intègre et couple informatiquement dans une structure modulaire bon nombre des théories mathématiques fondamentales, pavant le chemin pour d'autres preuves du même acabit.

La preuve informatique de Feit-Thompson a eu beaucoup plus d'impact que celle des 4 couleurs. Elle a certainement contribué à l'ouverture d'un trimestre dédié à la formalisation des mathématiques au célèbre Institute for Advanced Study de Princeton, celui-là même où officiaient Gödel, Church et Turing en 1936, et bien sûr aussi Einstein. L'Institut Henri-Poincaré lui consacra aussi un trimestre de conférences et de travaux, et de nombreux chercheurs s'intéressent maintenant à ce sujet.

J'aime bien les trois phrases suivantes, dont les deux dernières sont une adaptation par Leslie Lamport de la première, due à Dick Guindon :

Writing is nature's way of letting you know how sloppy your thinking is.

(« L'écriture est ce qu'utilise la nature pour vous montrer combien votre pensée est négligente. »)

Mathematics is nature's way of letting you know how sloppy your thinking is.

(« Les mathématiques sont ce qu'utilise la nature pour vous montrer combien votre écriture est négligente. »)

Formal mathematics is nature's way of letting you know how sloppy your mathematics is.

(« Les mathématiques formelles sont ce qu'utilise la nature pour vous montrer combien vos mathématiques sont négligentes. »)

Ces phrases sont sans doute trop caustiques pour les mathématiciens, dont la recherche de rigueur reste exemplaire. Mais elles vont parfaitement s'appliquer à la preuve de programmes et de circuits au [chapitre 10](#). C'est pour ces objectifs que Lamport les avait formulées.

LA RÉOLUTION DE LA CONJECTURE DE KEPLER

Jusqu'ici, nous n'avons parlé que de redémonstration informatique de théorèmes déjà démontrés manuellement. Un nouveau pas a été franchi en 2013 par Thomas Hales, qui a démontré la très ancienne conjecture de Kepler posée en 1611 : quelle est la meilleure façon de ranger des oranges en occupant le plus petit volume possible ? Tout le monde pensait que c'était comme le fait un épicier compétent, c'est-à-dire en superposant en quinconce des couches hexagonales. Mais la preuve avait résisté à tous les assauts.



Figure 67. Thomas Hales rangeant optimalement ses oranges. (© 2008 Regents of the University of Michigan. This photograph was taken by Bob Kalmbach and is held by the Bentley Historical Library. It is licensed under the CC-BY-NC 4.0 license.)

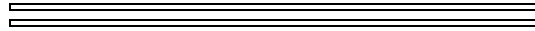
En 2003, Hales (figure 67) publia une preuve « à 99 % », fondée sur de gros calculs numériques en précision contrôlée implémentés en C++, qui lui ont permis d'établir de délicates inégalités suffisantes pour entraîner le résultat. Mais la partie logique de cette preuve restait informelle, et sa partie calculatoire trop dépendante de

C++ et de ses compilateurs. Hales décida donc de conduire une vraie preuve complètement formelle au sens des logiciens et des informaticiens. Il travailla pendant dix ans sur l'informatisation complète de sa preuve, cette fois avec deux systèmes informatiques directement liés aux mathématiques : le logiciel de calcul formel Mathematica pour conduire les énormes calculs polynomiaux nécessaires, et l'assistant de vérification formelle HOL (Higher Order Logic) de Michael Gordon dont nous reparlerons au [chapitre 10](#). Il acheva sa preuve en 2013.

C'est la première fois qu'une conjecture de cet ordre a été à la fois résolue et prouvée mathématiquement grâce à l'informatique. On attend la suite !

TROISIÈME PARTIE

Rendre l'informatique plus sûre



CHAPITRE 9

Bugs et trous de sécurité : deux dangers de l'informatique

Passons maintenant à deux côtés sombres mais incontournables de l'informatique, ceux liés aux bugs et aux trous de sécurité des systèmes informatisés, avant de montrer au chapitre suivant que rien n'est désespéré car nous disposons de méthodes de plus en plus efficaces pour en réduire le nombre et l'impact, voire les éliminer réellement dans certains cas importants. Comme déjà expliqué au [chapitre 3](#), les propriétés qui nous intéressent sont la sûreté et la sécurité. La sûreté est la garantie que les systèmes font ce qu'ils doivent faire et ne font pas ce qu'ils ne doivent pas faire, alors que la sécurité est la garantie qu'ils ne peuvent pas être pénétrés de l'extérieur pour modifier leur fonctionnement ou voler, détruire ou chiffrer des données. Les deux aspects sont fortement reliés, car les trous de sécurité sont le plus souvent liés à des microbugs passés inaperçus car n'étant pas reliés au fonctionnement normal et donc pas détectés par les tests fonctionnels classiques, souvent restreints aux cas considérés comme normaux. On réduit souvent la sécurité aux problèmes concernant les données. Mais ceux concernant le fonctionnement des systèmes sont tout aussi dangereux, voire plus : qu'on pense aux catastrophes que peut engendrer une prise de pouvoir sur les réseaux de distribution électrique ou sur les feux rouges d'une ville informatisée. Les récentes attaques mettant hors service des hôpitaux ou des sites industriels dans plusieurs pays pourraient bien faire figure d'entraînement à côté de ce qui nous attend dans l'avenir.

Les quelques exemples de bugs et trous de sécurité que je donnerai ici vont de l'amusant au tragique, en passant par certains qui ont pu être corrigés à temps de façon acrobatique avant que les vrais ennuis n'arrivent. Il est clair que les fabricants de logiciels et concepteurs d'appareils sont toujours réticents à admettre leurs bugs et analyser leurs causes ; les exemples bien documentés sont donc malheureusement rares. Mais ceux que je décrirai sont tous avérés. Chacun d'entre eux illustrera un principe déjà discuté au [chapitre 3](#) et hélas souvent vérifié : à petits bugs, grandes conséquences.

Tous les bugs n'ont bien sûr pas la même importance. Dans les systèmes un peu sérieux, on distingue souvent trois niveaux de sûreté des systèmes, caractérisés ci-dessous par des noms en anglais, universellement utilisés dans le domaine car plus précis et plus sobres que les périphrases françaises.

Le niveau le plus exigeant est celui des systèmes *safety critical*, ceux qui mettent directement en jeu des vies humaines. Ces systèmes se trouvent par exemple en avionique, ferroviaire et nucléaire. Ils sont soumis à des processus normés de *certification des logiciels* qui exigent des spécifications et documentations précises, des programmes bien écrits et correspondant fidèlement aux spécifications, des tests systématiques et des revues par des experts extérieurs, le tout étant bien géré informatiquement à travers les évolutions successives des logiciels avant et après la mise en production. Les processus de certification peuvent être internationaux, comme en avionique, ou nationaux, comme en nucléaire (ce qui m'étonnera toujours). Pour le moment, ils s'intéressent surtout au côté administratif du développement, ce qui ne garantit pas formellement l'absence de bugs ou de trous de sécurité. Ils sont cependant efficaces, et seront certainement complétés à l'avenir par les méthodes scientifiques que nous discuterons au chapitre suivant. Leur inconvénient est d'être très coûteux en temps humain.

Le deuxième niveau concerne les systèmes appelés *mission critical*, ceux pour qui la présence de bugs peut conduire à l'abandon de missions coûteuses mais sans impact direct sur les vies humaines. Un bon exemple est celui des systèmes spatiaux. Les méthodes de développement sont généralement proches de celles des systèmes critiques, mais sans processus de certification externe.

Le troisième niveau est celui des systèmes *business critical*, ceux pour qui les bugs peuvent tuer un produit ou même une compagnie. On ajoute parfois le

qualificatif *time critical*, pour dire que le projet doit impérativement se terminer avant une certaine date : on a vu par exemple un grand fabricant de consoles de jeux ne pouvoir produire le circuit d'un nouveau modèle de jeu qu'en février au lieu de novembre de l'année précédente ; dans le domaine du jeu où Noël est une date clef, ce n'est pas du tout la même chose !

Un point qui m'étonne toujours est que l'automobile semble encore échapper à la classification ci-dessus, et en particulier à ses deux niveaux supérieurs, alors même que les voitures sont de plus en plus informatisées, y compris pour leurs fonctions critiques. La qualité actuelle de cette informatisation est loin d'être toujours du plus haut niveau, comme nous allons le voir dans quelques exemples. Une raison est que cette industrie reste dominée par les coûts et que les contraintes des systèmes critiques les alourdissent beaucoup : aucun constructeur ne bougera le premier. Une autre raison est que les accidents étant dilués dans l'espace et le temps, ils ne font pas la une des journaux. Mais la principale raison me semble être que l'industrie automobile est encore guidée par un schéma mental qui est celui de la mécanique. Si les problèmes liés à la mécanique se détectent avec nos yeux ou nos oreilles, ce n'est absolument pas le cas pour ceux dus à l'informatique, où les inspections visuelles ou auditives sont inopérantes. Les problèmes liés aux logiciels sont d'autant plus dangereux que les bugs sont reproduits à l'identique dans toutes les voitures d'un type donné. J'espère que nos autorités politiques se saisiront un jour de ces questions, surtout avec l'arrivée de systèmes d'aide à la conduite de plus en plus perfectionnés, visant même des formes d'autonomie plus ou moins complètes de la voiture.

Les bugs de temps ne sont pas rares

LE ZUNE ET LA PS3 FAT N'AIMAIENT PAS LES ANNÉES BISSEXTILES

Notre premier exemple sera celui du Zune, lecteur MP3 commercialisé en 2006 par Microsoft. Le 31 décembre 2008 à minuit, tous ces appareils se sont arrêtés brutalement de fonctionner, n'obéissant à aucune commande. En fait, ils n'étaient pas arrêtés, mais simplement en train de vider leur pile sans rien faire d'utile. Recharger la pile et redémarrer son Zune le 1^{er} janvier faisait immédiatement réapparaître le même problème. Il fallait attendre le lendemain pour que l'appareil se remette en marche normale, du moins jusqu'au 31 décembre 2012 à minuit où le même phénomène ne manquerait pas de se reproduire de façon déterministe.

Que s'était-il passé ? Un bug idiot dans un programme tout simple écrit en langage C par un contractant, qui était en charge de l'heure. Pour convaincre définitivement le lecteur qu'un seul caractère suffit à changer un bon programme en un gros bug, je choisis de lui montrer le petit bout de code concerné, qui ne demande qu'une compréhension minimale de la programmation.

```
while (days > 365) {  
    if (IsLeapYear(year)) {  
        if (days > 366) {  
            days -= 366;  
            year += 1;  
        }  
    } else {  
        days -= 365;  
        year += 1;  
    }  
}
```

Figure 68. Le bug du Zune.

Les variables entières *year* et *days* contiennent respectivement les numéros de l'année et du jour, la fonction *IsLeapYear* teste si l'année est bissextile, et la condition « $x > y$ » teste si x est strictement plus grand que y .

La programmation étant une activité de précision, soyons précis pour décrire le comportement des instructions de test « if » et de boucle « while ». Un test « if » prend sa première branche (de la forme {...}) si sa condition entre parenthèses est vraie, et se termine dès que cette branche se termine ; si sa condition est fausse et si une seconde branche est spécifiée après le mot clef « else », le « if » n'exécute pas sa première branche mais exécute sa branche « else » et se termine dès qu'elle s'est terminée ; si la condition est fausse et qu'il n'y a pas de branche « else », le « if » se termine simplement.

Une boucle « while » (en français « tant que ») teste d'abord sa condition ; elle se termine immédiatement si cette condition est fausse, mais exécute son corps sinon. Dans ce dernier cas, si et quand ce corps se termine, la boucle « while » recommence la même opération en testant de nouveau sa condition, et ainsi de suite. Cela veut dire que la boucle tourne infiniment tant que sa condition reste vraie ; la seule façon d'en sortir est de rendre la condition fausse (il y a d'autres façons plus brutales en C, mais

elles ne sont pas pertinentes ici). C'est ce que nous allons voir se produire dans le minuscule bout de code de la figure 68, qui a suffi à tuer les Zune.

Le 31 décembre 2008 à minuit, juste avant que ce code ne soit appelé, le système incrémente *days* de 365 à 366. Comme « $366 > 365$ » est vrai, on entre dans le corps de la boucle « `while (days > 365)` ». Le test « `if (IsLeapYear(year))` » teste alors si l'année est bissextile, ce qui est vrai pour 2008. On s'engage donc dans sa première branche, le « `if (days > 366)` » interne. Comme la réponse est cette fois négative, ce « `if (days > 366)` » se termine sans exécuter son corps « `days -= 366; year += 1;` », celui qui aurait dû incrémenter l'année. Ceci fait aussi terminer le « `if (IsLeapYear(year))` » externe, ce qui provoque le retour à la boucle « `while` » avec *days* inchangé à 366. On repart donc pour un tour identique, et on boucle sur ces mêmes instructions, non pas indéfiniment comme on le devrait en théorie, mais jusqu'à ce que la pile soit vide et le Zune inerte.

Si on redémarre l'appareil le même jour une fois sa pile rechargée, la même chose se produit. Par contre, si on attend le lendemain pour redémarrer le Zune, *days* se trouve mis à 367 avant qu'on exécute ce code, le test « $days > 366$ » devient vrai, et tout se passe bien !

Il est clair que ce programme n'avait jamais été relu ni testé de façon correcte. Mais n'est pas le seul bug dû aux années bissextiles. Le 28 février 2010 à minuit, les consoles de jeux PS3 Fat de Sony sont aussi parties en vrille, cette fois en remettant la date au 1^{er} janvier 2000. La faute était dans un petit bout de code noyé dans un processeur auxiliaire. Les joueurs ont perdu leurs scores et n'ont plus été autorisés à se connecter au réseau. Vous pouvez trouver beaucoup de réactions d'utilisateurs vraiment mécontents sur le Web.

LE BUG DU RÉVEIL DE L'IPHONE 4

Voici un autre joli bug de temps qui a affecté les iPhone 4 d'Apple aux États-Unis et dans d'autres pays lors du passage à l'heure d'hiver du 1^{er} novembre 2010. Supposons qu'un utilisateur ait mis la veille son réveil à 7 h 30, heure à laquelle il souhaitait être réveillé en tenant compte du changement d'heure. Au cours de la nuit, l'heure interne de l'iPhone a été automatiquement retardée sur les téléphones, comme

prévu, le réveil affichant toujours 7 h 30 car lui ne devait pas changer. Mais un bug dont je ne connais pas les détails a fait que le réveil a sonné à 8 h 30 et pas 7 h 30 ! D'un côté, cela a donné une bonne excuse pour de très nombreux retards au bureau ou à des rendez-vous. Mais d'un autre côté, cela a fait rater à d'autres une quantité considérable de trains et d'avions, ce qui est bien moins drôle et a certainement provoqué des coûts indirects considérables !

UN BUG DU MERCREDI

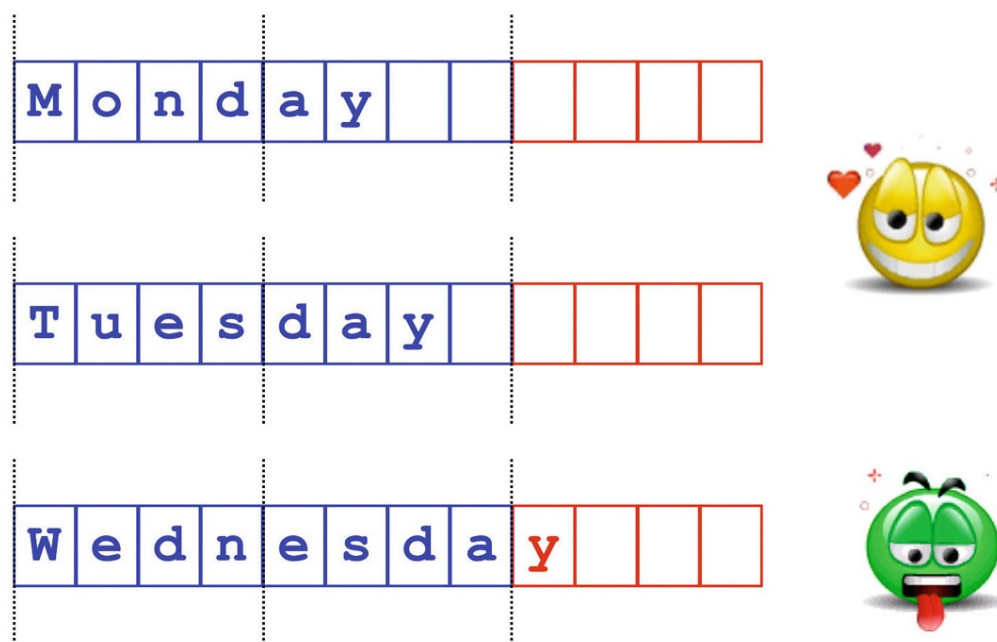


Figure 69. Le bug du mercredi.

Voici maintenant un bug original dans un système de gestion, illustré par la figure 69. Ce système devait être démarré tous les matins. Lors de son démarrage, il commençait par poser la question d'un démarrage normal ou d'un démarrage en mode maintenance. Pour démarrer la maintenance, on tapait « y », sinon, un simple retour chariot. De façon curieuse, le système démarrait en mode maintenance tous les mercredis en refusant obstinément de démarrer en mode normal !

La raison était simple mais subtile. Une des premières choses que faisait le logiciel était de demander au système d'exploitation de stocker le nom du jour dans

une mémoire préallouée de 8 caractères, soit deux mots machine à l'époque. Ensuite, le logiciel enregistrerait la réponse à la question du mode de démarrage dans l'octet de gauche du mot mémoire suivant, préalablement rempli de caractères blancs. En mode normal, c'est-à-dire quand l'opérateur ne tapait pas « y », aucun problème pour *Monday, Tuesday, Thursday, Friday, Saturday* et *Sunday* : le premier octet du mot suivant contenait toujours un blanc. Mais, pour *Wednesday* qui fait neuf lettres, le système écrivait « y » dans la case dédiée à la réponse à la question du mode de démarrage, ce qui faisait automatiquement démarrer le système en mode maintenance.

Cette fois, l'erreur humaine, aussi minuscule qu'une erreur de programmation, n'était pas dans le code mais dans la documentation du système d'exploitation qui spécifiait que la date serait écrite sur 8 octets au lieu de 12. En fait, ce « bug de doc » était même pire qu'un bug de programmation, puisqu'il était susceptible d'engendrer des ennuis analogues pour *tous* les utilisateurs du système d'exploitation en question.

LES RAPTOR N'AIMENT PAS LA LIGNE DE CHANGEMENT DE DATE

Le F22A Raptor est un des avions furtifs les plus avancés produits aux États-Unis : une merveille de technologie à 330 millions de dollars pièce. Le 11 février 2007, douze Raptor se rendaient de Hawaï au Japon. Quand ils ont traversé la ligne de changement de date, tous leurs systèmes informatiques de navigation, de communication et de jauge de carburant se sont éteints en même temps pour ne jamais se rallumer. Les pilotes se sont donc retrouvés sans instruments. Heureusement, il faisait beau et les Raptor étaient accompagnés par leurs ravitailleurs. Ils ont pu faire demi-tour, suivre les ravitailleurs et se ravitailler à vue, puis atterrir à Hawaï. Comme ils ne transportent pas de passagers, les avions de chasse ne sont pas soumis au régime *safety critical*, réservé aux avions de ligne !

LE PATRIOT RATE LE SCUD

Pour terminer, voici un accident dramatique lié au temps, cette fois dans un système vraiment critique : le 25 février 1991, lors de la première guerre du Golfe, la

chute d'un missile Scud irakien sur une caserne américaine à Dharan fit 28 morts et 98 blessés parmi les militaires présents. Pourtant, la caserne était protégée par des missiles Patriot, qui n'auraient dû n'avoir aucun problème à intercepter le Scud. Mais l'informatique de la rampe de lancement des Patriot était très ancienne et ne respirait pas la qualité. En particulier, au lieu d'être gérée en binaire, l'heure était gérée en décimal avec des arrondis douteux et dérivait régulièrement. C'était connu des militaires, et il était conseillé de redémarrer le système toutes les quatre ou cinq heures (!). Mais la tension était forte et le système est resté beaucoup plus longtemps en alerte, presque une centaine d'heures. Quand le Patriot a été lancé, l'heure avait dérivé de 0,34 seconde par rapport aux autres systèmes, ce qui lui a suffi pour rater la cible : comme le Patriot voyage à 1 676 mètres par seconde, l'erreur était de 569 mètres.

Le bug qui a tué Ariane 501

Peu de systèmes complexes sont fabriqués avec autant de soin que les systèmes spatiaux, tous jugés *mission critical*. Pourtant, les bugs informatiques n'y sont pas rares, et j'en citerai quelques-uns. Le plus célèbre est celui de l'échec du premier lancement Ariane 501 du lanceur Ariane, une fierté européenne et française bien justifiée par ailleurs. Comme il est particulièrement instructif, je propose de l'étudier en détail et d'en tirer plusieurs conséquences, celles-là mêmes qu'on trouve dans le rapport d'analyse officiel coordonné par mon ami Gilles Kahn¹.

Le début du vol fut nominal, mais, après 37 secondes, la fusée braqua brutalement et commença à se disloquer. Cela déclencha son système d'autodestruction et conduisit à une énorme explosion et à la perte des 4 satellites (non assurés) que la fusée emportait. Le diagnostic ne prit pas plus d'une semaine, car certaines données avaient été transmises par télémétrie et car les enregistreurs de vol avaient pu par chance être récupérés dans la mangrove de Kourou (Guyane) d'où était parti le lanceur. Il s'agissait d'un bug minuscule dans un logiciel embarqué dans les composants essentiels que sont les *systèmes de référence inertielle* (SRI), capteurs à base de gyrolasers qui déterminent l'attitude et la vitesse du lanceur, donc les paramètres essentiels pour son guidage.

LES SRI SE DÉCLARENT EN PANNE PAR ERREUR

Comme beaucoup de composants clefs des systèmes spatiaux, ces SRI sont doublés, avec un SRI actif et un SRI de secours identique qui calcule en même temps mais dont les données ne sont utilisées qu'en cas de panne détectée sur le premier. Dans Ariane 5, les SRI étaient d'un modèle qui avait fait ses preuves sur Ariane 4 et n'avait subi ni modifications ni tests additionnels. Après 36,7 secondes de vol, le SRI de secours a détecté une erreur interne et s'est déclaré en panne. Le SRI actif a fait la même chose immédiatement après. Dès qu'il a enregistré la panne du SRI actif, le calculateur de bord a basculé sur le SRI de secours, qui lui a renvoyé un message d'erreur au lieu d'un angle puisqu'il était aussi en panne. Interprétant quand même ce message comme un angle, le calculateur de bord a déclenché le virage brusque observé. Cela dit, comme elle n'avait plus de guidage, la fusée était perdue de toute façon.

Le vrai paradoxe est qu'il n'y avait en fait aucun problème car les deux SRI fonctionnaient parfaitement ! La panne qu'ils ont déclarée n'était due qu'à une erreur dans une partie du logiciel qui ne servait strictement à rien et n'aurait jamais dû être là. Le cas est très clairement examiné dans le rapport officiel, dont je reprends les points essentiels ici.

La cause de l'arrêt des deux SRI était identique, et liée à une variable flottante, c'est-à-dire à grande dynamique de valeurs potentielles grâce à un couple mantisse-exposant, comme dans $3,1415 \times 10^{12}$. Sa valeur devait être convertie en une valeur entière sur 16 bits pour stockage en mémoire flash. À exactement 36,7 secondes de vol, cette variable avait dépassé 65 535, qui est le plus grand nombre entier stockable sur 16 bits. Son stockage en mémoire flash a alors provoqué ce qu'on appelle un *débordement arithmétique* (ou *overflow* en anglais). Dans ce cas, le comportement dépend du langage de programmation. Si le code avait été écrit en C par exemple, on n'aurait rien vu du tout – ce qui n'aurait d'ailleurs pas voulu dire qu'il n'y avait pas de bug. Mais les SRI étaient programmés en ADA, langage à juste titre très sérieux en termes de détection d'erreurs à l'exécution. Le débordement arithmétique a provoqué ce qu'on appelle une *exception logicielle*. Si elle n'est pas traitée explicitement, comme c'était le cas pour les SRI, une telle exception arrête immédiatement l'exécution du programme ADA tout entier. Les deux SRI se sont donc arrêtés quasiment simultanément, puis déclarés en panne. Mais ils ne l'étaient en fait pas, *car*

la valeur stockée ne servait à rien, et tout ce qui servait fonctionnait normalement !
Le vrai bug était d'avoir détecté un faux bug²...

On voit très bien ici qu'un tel bug logiciel est très différent des pannes matérielles : le bug des SRI était parfaitement déterministe. Si on avait lancé 100 fusées Ariane identiques, elles auraient toutes explosé au même endroit pour la même raison, à moins bien sûr qu'elles n'aient eu d'autres ennuis avant.

LES VRAIES CAUSES DU PROBLÈME ÉTAIENT HUMAINES

Le vrai problème n'était en fait pas dans l'erreur de conversion mais dans ses causes humaines. Comme je l'ai déjà dit et répété, le bug n'est pas une panne de la machine, mais la conséquence d'une erreur humaine de conception – en fait ici d'une série d'erreurs provenant au fond d'une même cause, la sous-estimation des difficultés du logiciel et la transposition au logiciel d'un schéma mental pertinent pour la matière mais inadapté à l'immatériel.

La variable qui avait provoqué le débordement stockait la vitesse horizontale du lanceur. Il se trouve que cette vitesse avait servi au début d'Ariane 4, mais seulement avant le décollage ; elle n'était plus d'aucune utilité depuis longtemps, ni sur Ariane 4 ni sur Ariane 5. Comme pour d'autres variables du SRI, stocker périodiquement sa valeur en mémoire avait été justifié par le risque d'arrêts momentanés du compte à rebours, pour éviter de recommencer à zéro la très longue procédure de calibration des SRI. Mais la procédure de compte à rebours avait été modifiée, et ce stockage n'était plus d'aucune utilité. De plus, même quand cette variable avait été utile, le code correspondant n'avait jamais eu le moindre intérêt en vol, y compris pour Ariane 4. Pourtant, il n'avait pas été débranché de l'exécution après le décollage, le mode de calibration étant gardé actif quarante secondes après le décollage. La vitesse horizontale d'Ariane 4 étant faible, il n'y avait jamais eu de débordement lors du stockage de sa variable. Mais celle d'Ariane 5 était considérable, d'où le débordement constaté.

Au total, la conclusion fut étrange : l'accident était dû à un débordement lors de l'écriture d'une valeur sans intérêt dans une WOM (*write only memory*), la seule

forme de mémoire qui n'a vraiment aucune utilité !

Mais pourquoi donc avoir conservé cette variable inutile et le code inutile qui la manipulait, et avoir continué à garder en marche la procédure de calibration même en vol ? Je pense que c'est en fonction d'un vieil adage que j'ai entendu trop souvent : « en informatique, on ne touche à rien quand ça marche ». Et pourquoi n'a-t-on pas simulé le vol d'Ariane 5 avec les SRI en fonction, alors que sa trajectoire était très différente de celle d'Ariane 4 ? En fonction du même adage je pense. Et pourquoi n'avoir pas refait une analyse du code en question ? Il y en a en fait eu une, qui a analysé si les diverses variables pouvaient ou non conduire à des dépassements de capacité, mais qui n'a pas détecté le problème de la vitesse horizontale pour des raisons que j'ignore.

ANALYSES ET RECOMMANDATIONS DE LA COMMISSION D'ENQUÊTE

La commission d'enquête a émis de nombreuses recommandations pour la correction des logiciels d'Ariane 5, et, de façon plus générale, pour le développement et la validation des logiciels spatiaux. Leur point central était de ne pas se contenter d'adapter directement les méthodes applicables au matériel, mais de prendre vraiment en compte les spécificités du logiciel.

Sur le plan proprement informatique, la commission a recommandé d'arrêter toute fonction logicielle non indispensable pendant le vol, comme ici le mode de calibration des SRI, d'interdire aux équipements de cesser d'envoyer les meilleures données disponibles, ce qui aurait suffi à garder en vie le vol 501, de confiner les exceptions à l'intérieur des tâches et de prévoir des procédures de secours en cas d'exception logicielle.

Sur le plan des processus de conception du logiciel, elle a recommandé d'adopter les pratiques suivies ailleurs, et en particulier en avionique : définir la criticité des composants en tenant compte des logiciels qu'ils contiennent et pas seulement de leurs aspects matériels ; écrire des spécifications complètes et précises des programmes en identifiant toutes les hypothèses faites sur les paramètres fournis par les équipements ; vérifier dès la spécification qu'aucune valeur ne peut sortir de sa

plage nominale dans les conditions du vol près des trajectoires nominales ; intégrer ces trajectoires dans toutes les spécifications et tous les tests unitaires ou d'intégration, en allant jusqu'à la simulation complète du vol avec tous les équipements ; améliorer la coopération technique entre les partenaires du programme ; associer des experts extérieurs au projet pour y apporter un regard différent et conduire avec eux des revues externes des spécifications, tests et vérifications. En bref, elle recommandait de faire *avant* les études qu'elle a dû faire après.

Deux recommandations particulières demandaient un petit décodage. La première était *d'augmenter le nombre de données transmises par télémetrie*. L'idée était qu'il fallait non seulement transmettre des données concernant l'état du matériel, mais aussi des données concernant l'état interne du logiciel, afin de pouvoir en cas d'accident diagnostiquer des bugs sans avoir besoin de récupérer les enregistreurs de vol qui sont souvent introuvables après une telle catastrophe. La seconde était de *traiter les documents de justification avec le même soin que le code*. Si j'ai bien compris, il y avait bien eu commande d'un document de vérification des plages de variables qui conseillait d'en protéger certaines contre les exceptions logicielles, mais qui oubliait précisément la vitesse horizontale. Ce document n'a pas dû être bien relu, car le manque aurait pu être détecté par un autre que celui qui l'avait écrit. En mathématiques, quand un auteur annonce un théorème, des relecteurs vérifient systématiquement les démonstrations avec le plus grand soin, car on estime qu'on ne peut pas faire confiance à une seule personne pour affirmer qu'elles sont justes, surtout si c'est l'auteur ; il faut faire pareil pour la programmation des systèmes critiques.

Beaucoup de leçons ont donc été tirées de l'échec d'Ariane 501 en termes de développements informatiques. À l'origine, ce lanceur avait été conçu pour la navette spatiale européenne Hermès, qui n'a jamais existé. Peu après la naissance de ce projet, et à la demande de Jacques-Louis Lions, grand mathématicien alors président du CNES³, j'avais coordonné une mission d'expertise logicielle de l'ensemble du programme Hermès, avec une vingtaine de collègues connaissant bien le sujet. Nos conclusions avaient été claires : les méthodes et procédures d'analyse, de spécification et de réalisation choisies n'étaient pas à la hauteur des enjeux, les spécificités du logiciel étaient mal comprises car la plupart des acteurs n'avaient eu que peu de

contact avec le logiciel auparavant, et tout cela mettrait grandement les programmes en danger. Je dois dire que nos conclusions n'avaient pas été bien perçues et que notre mission n'a pas été poursuivie. À la suite de l'échec du vol 501, les choses ont bien changé et les recommandations de la commission d'enquête ont été vraiment prises en compte.

Parfois, il faut un accident majeur pour que les bonnes pratiques connues ailleurs arrivent à la surface, même lorsque cet ailleurs est au sein de la même entreprise.

UN IMPACT POSITIF : LA VÉRIFICATION PAR INTERPRÉTATION ABSTRAITE

L'explosion d'Ariane 501 a aussi eu un impact heureux sur le passage des techniques modernes de vérification de programmes de la recherche à l'industrie. En 1977, Patrick et Radhia Cousot avaient publié un article maintenant célèbre proposant une nouvelle méthode mathématique de vérification appelée *interprétation abstraite* (Cousot & Cousot, 1977). Il s'agissait d'approximer les valeurs prises par les variables à l'aide de plages de valeurs de diverses formes géométriques (intervalles, polygones, etc.), et d'assurer que ces valeurs restent dans les plages spécifiées en exécutant symboliquement le programme en temps fini sur des plages de valeurs au lieu de le faire exactement sur des valeurs précises. On peut alors obtenir des garanties suffisantes en un temps raisonnable.

Une implémentation prototype de l'interprétation abstraite par le regretté Alain Deutsch de l'équipe Cousot a permis de trouver automatiquement et rapidement le bug des SRI, rien qu'en analysant automatiquement leur code. Elle a été ensuite étendue en une implémentation industrielle, celle de l'outil Polyspace vendu par la société du même nom puis racheté plus tard par la société américaine The MathWorks. Le travail académique sur l'interprétation abstraite a ensuite continué de plus belle, ses progrès étant intégrés dans des outils performants d'analyse de programmes. À l'aide d'un nouveau système appelé Astrée, l'équipe de Patrick Cousot à l'École normale supérieure a pu garantir à 100 % l'absence d'erreurs à l'exécution dans l'intégralité du programme de pilotage de l'Airbus A380, gros code de centaines de milliers de lignes. Ce type de vérification, jugé totalement impossible

à cette échelle il y a vingt-cinq ans, est de plus en plus systématiquement utilisé en avionique, spatial, ferroviaire, nucléaire, automobile, et dans bien d'autres secteurs de l'informatique embarquée de haute sûreté. Nous y reviendrons au chapitre suivant.

D'autres applications industrielles de l'interprétation abstraite ont été réalisées par la société allemande AbsInt, avec en particulier le calcul d'une approximation supérieure garantie du temps maximal d'exécution d'un code embarqué (*worst case execution time*, en abrégé WCET), essentielle pour les applications temps réel et utilisée entre autres par Airbus : un dépassement temporel peut en effet compromettre l'efficacité et la sécurité d'un programme de pilotage. L'interprétation abstraite, qui a bien d'autres applications, est un des grands succès de la recherche française.

FAUT-IL DUPLIQUER LES LOGICIELS POUR SE PROTÉGER DES PANNES ?

Un système embarqué doit bien sûr se protéger des pannes toujours possibles. Un choix classique pour un lanceur de courte durée de vie est de doubler les équipements matériels critiques, avec un actif et un de secours comme pour les SRI d'Ariane. Si l'actif est en panne, on passe sur le secours. Si celui-ci est aussi en panne, on considère qu'on ne peut plus rien faire et la mission est abandonnée. Quand on considère les seules pannes matérielles, c'est une bonne méthode si ces pannes sont indépendantes. Dans ce cas, on peut multiplier les probabilités des pannes individuelles pour calculer la probabilité globale de panne : si on évalue la probabilité de panne d'un équipement pendant la durée du vol à 1/100, la probabilité de panne des deux n'est plus que de 1/10 000, ce qui est parfaitement acceptable en spatial.

Mais ce raisonnement ne marche plus du tout pour un bug de logiciel comme celui des SRI d'Ariane 5. À moins qu'elles n'aient explosé ou trop dévié de leur trajectoire auparavant, *toutes* les fusées lancées auraient explosé après 37 secondes de vol. Le logiciel concerné étant déterministe, le bug se serait produit *immanquablement* sur chaque SRI après 36,7 secondes de vol sur la trajectoire nominale. La probabilité globale de panne était donc $1 \times 1 = 1$.

On peut envisager deux méthodes pour se sortir du problème. La première est de *garantir formellement que le logiciel n'a pas de bug* (d'un certain type), ce qu'on

commence à savoir faire dans quelques cas, mais certainement pas dans tous les cas ; l'interprétation abstraite est une des possibilités, mais j'en décrirai d'autres dans le prochain chapitre. Mais la vérification formelle reste difficile et demande une expertise particulière qui n'est pas encore disponible dans la plupart des industries ; je l'ai cependant vue très bien employée pour la certification de systèmes nucléaires. La seconde méthode est de *diversifier les logiciels*. C'est la solution prise par Airbus pour ses avions depuis l'A320 (1987), qui utilise un système de redondance matérielle et logicielle très sophistiqué pour détecter et tolérer les erreurs matérielles et logicielles, que je ne peux décrire ici. Ces sujets sont toujours en discussion active.

LES BUGS ONT-ILS UNE PROBABILITÉ ?

Au paragraphe précédent, nous avons parlé de probabilités. Voici une question qui m'a régulièrement été posée par des non-informaticiens lors d'expertises variées de logiciels critiques :

- Quelle est selon vous la probabilité qu'il y ait un bug dans ce logiciel ?

- Réponse : 0 ou 1.

- Comment ça ?

- 0 s'il n'y en a pas, 1 s'il y en a !

- Mais c'est idiot, tout a une probabilité !

- Pour le matériel, oui. Pour la logique, l'informatique ou les mathématiques, pas forcément. Quelle est selon vous la probabilité pour que le théorème de Fermat soit vrai ?

Une question meilleure serait la suivante : « Quelle est la probabilité qu'un humain ait fait un bug dans la spécification ou la programmation de l'application et que ce bug n'ait pas été détecté lors de la validation, et qu'il ait une conséquence négative sur la mission ? »

Il est clairement impossible de donner une réponse à ce genre de question vraiment trop peu précise. Mais est clair que le taux de bug dépend de la qualité de la réalisation. Les avions sont excellents, les applications de téléphone généralement plutôt médiocres, quoiqu'en amélioration, et les voitures pas aussi bonnes qu'on le souhaiterait. De nombreux auteurs ont proposé des outils calculant des « métriques de

qualité » sur les codes sources pour prévoir en gros le nombre de bugs d'un programme. Si leurs résultats sont mauvais, ce qui reste fréquent, c'est très utile. Mais je suis loin de croire à leur validité quand les mesures sont positives. Et je ne suis pas sûr non plus que le mot même de probabilité soit bien adapté ici. Bien sûr, on peut calculer finement des probabilités *a priori* comme nous l'a appris le révérend Bayes il y a bien longtemps, mais de là à y croire si on n'est pas un vrai expert du sujet, c'est une autre histoire...

Quelques bugs martiens

Les missions sur Mars sont particulièrement coûteuses et traitées avec beaucoup de soin. Mais les bugs logiciels n'y ont pas été rares, voire y ont été plus nombreux que les problèmes matériels. Certains ont pu être réparés de façon acrobatique grâce à la compétence des ingénieurs, d'autres non. Voyons-en quelques-uns des deux sortes, qui ont l'avantage d'avoir été complètement analysés et publiés.

L'INVERSION DE PRIORITÉ DE PATHFINDER

Le premier véhicule robot à atterrir sur Mars fut le célèbre Pathfinder, le 4 juillet 1997. L'amarsissage, fort délicat, se passa de la façon prévue. Mais, après le redémarrage des ordinateurs de bord, la liaison avec le robot fut vite perdue, redevint nominale quelques instants, fut reperdue, etc. Les instruments de Pathfinder semblaient fonctionner correctement, mais il était impossible de recueillir leurs données. Après une nuit de travail acharné, le même comportement fut reproduit à terre par les ingénieurs de la NASA sur leur copie au sol du robot, et la source du problème identifiée. En voici l'explication, que je recopie d'une analyse disponible sur le Web⁴.

Les différentes tâches logicielles que doit effectuer le robot sont gérées par un système d'exploitation nommé VxWorks, un des grands classiques du domaine. Son rôle est d'allouer les ressources aux différentes tâches de calcul, de les démarrer, les suspendre en cas d'interruption, les relancer, gérer leur terminaison, etc. Comme il

n'y a qu'un processeur, une seule tâche peut être exécutée à la fois. Trois tâches sont concernées ici : la gestion du bus de données, qui permet de transférer des données du calculateur vers des mémoires externes, la communication, qui permet de discuter avec la Terre, et la météorologie. Ces tâches ont des priorités différentes. La plus prioritaire est la tâche du bus, fréquemment utilisée ; derrière vient la communication ; enfin, la moins prioritaire est la météo. VxWorks est un système *préemptif*, ce qui signifie qu'il permet de suspendre momentanément l'exécution d'une tâche si une autre de plus haute priorité demande à être exécutée.

Pour éviter des collisions entre tâches provoquant des conflits sur le bus de données, toute tâche qui veut envoyer des données à la mémoire externe doit commencer par mettre à 1 une variable partagée entre les tâches et appelée *sémaphore d'exclusion mutuelle*, ou *mutex* en langage technique ; la tâche doit impérativement la remettre à 0 ensuite pour permettre aux autres tâches de prendre le bus à leur tour. C'est ce qu'on appelle *prendre et rendre le mutex*, par analogie avec le bâton qu'on utilisait parfois avant la signalisation électrique pour éviter les accidents de chemin de fer sur les voies uniques : on ne pouvait entrer dans la voie qu'une fois le bâton physiquement pris, et on devait le laisser à l'arrivée pour que le conducteur du prochain train dans l'autre sens puisse le prendre à son tour. Ici, le mutex concernait l'accès au bus. Dans Pathfinder, lorsqu'une tâche avait pris le mutex, toute tâche qui voulait transmettre des données par le bus devait attendre que le mutex soit rendu pour le prendre à son tour.

La situation problématique était la suivante : quand la tâche de météorologie était active et voulait transférer des données à la mémoire externe, elle prenait le mutex, demandait à VxWorks le lancement de la tâche du bus, puis se mettait en attente de la fin de cette tâche. Or il était possible à la tâche de communication de démarrer entre le moment où le mutex était pris et celui où la tâche du bus était lancée. La tâche de communication se mettait alors en attente du relâchement du mutex. Plus prioritaire, la tâche du bus pouvait s'exécuter pour transmettre les données de la tâche météo. Quand elle avait fini son travail, elle rendait la main à VxWorks. La tâche de météo aurait dû alors reprendre la main pour rendre le mutex. Mais, étant moins prioritaire que la tâche de communication qui attendait le mutex, elle n'était plus exécutable ! Voilà une situation typique d'*interblocage (deadlock)*, illustrée par Lise et Laure dans la [figure 14 du chapitre 3](#). Heureusement, pour pallier toute défaillance de l'ordinateur

principal, un circuit auxiliaire surveillait le processeur principal et le redémarrait s'il voyait que plus rien ne se passait. Les choses se remettaient à fonctionner un petit moment jusqu'à ce que le même interblocage se reproduise à un moment ou à un autre.

Les ingénieurs de la NASA avaient déjà observé une ou deux fois des blocages à terre, mais les avaient attribués à des erreurs intermittentes du matériel (comme d'habitude). Pourtant, ce problème purement logiciel avait déjà été étudié et résolu par des théoriciens du calcul parallèle. La solution est simple : quand la tâche météo de faible priorité lance la tâche du bus de priorité maximale, elle doit *hériter de cette priorité*, c'est-à-dire passer elle-même en priorité maximale. Quand la tâche du bus se termine, la tâche météo peut alors reprendre la main en passant devant la tâche de communication. Ce mécanisme était parfaitement prévu dans VxWorks : il suffisait pour cela de mettre une variable globale à 1. Or cette variable avait été explicitement mise à 0, peut-être par peur de problèmes potentiels de performance accompagnée d'une analyse insuffisante des conséquences potentielles de cette décision.

Heureusement, par prudence, les ingénieurs de la NASA avaient laissé actif le mode *debug* de VxWorks dans Pathfinder, ce qui permettait d'intervenir directement dans le système. Ils ont pu envoyer à VxWorks un bout de code tout simple et le lui faire exécuter à distance juste avant un blocage. Ceci a suffi à remettre la variable d'héritage à 1, et donc à assurer la résurrection de Pathfinder. Ouf, changer un simple 0 en 1 a transformé le robot de moribond à en pleine forme. Mais il a fallu en gros huit minutes pour un signal émis depuis la Terre pour atteindre Mars, et donc plus d'un quart d'heure pour savoir si la manip a marché. On imagine le stress dans la salle de commande, puis l'ouverture des bouteilles !

LA MÉMOIRE FLASH TROP PLEINE QUI A FAILLI TUER SPIRIT

Le célèbre rover martien Spirit s'est posé sur Mars le 4 janvier 2004, trois semaines avant son jumeau Opportunity. Conçu pour être opérationnel pendant 90 jours martiens (un peu plus longs que les nôtres), il l'a en fait été pendant 2 208 jours, soit plus de cinq ans, en fournissant une moisson considérable d'images de données

géologiques. Pourtant, il a failli tomber hors d'usage dès son arrivée à cause d'un bug de logiciel, qui a heureusement pu être ici aussi résolu à distance par les ingénieurs du JPL (Jet Propulsion Laboratory). L'histoire a des points communs avec celle de Pathfinder, mais elle est un peu plus subtile et aussi plus instructive. Comme elle est parfaitement décrite dans un article écrit par des ingénieurs de l'équipe ([Reeves & Tracey, 2004](#)), je n'en rappellerai ici que les grandes lignes.

Après s'être posé sans problème, à la grande joie de ses créateurs tant l'opération est délicate, Spirit commença à communiquer normalement avec la Terre. Mais, au bout d'un quart d'heure, soit un quart d'heure avant la fin normale de la communication, son signal fut perdu. Pendant trois jours, les ingénieurs purent dialoguer par bribes avec le rover, mais aucune communication fiable de durée suffisante ne put être établie. Pas facile d'analyser ce qui se passe dans ces conditions, d'autant plus que les batteries se vidaient car les anomalies constatées empêchaient aussi Spirit de s'éteindre pendant la nuit martienne pour les économiser. Après avoir étudié et écarté beaucoup d'hypothèses de pannes matérielles, il fut compris qu'il s'agissait probablement d'un bug logiciel dans le FSW (*flight software*). Une analyse fine identifia enfin le problème, dû non pas au FSW lui-même mais à un bug dans une librairie du système d'exploitation DOS (*disk operating system*, celui des PC anciens) chargée de la gestion de la carte de mémoire flash servant à stocker les données acquises pour les communiquer ensuite à la Terre. Ce bug est très typique des ennuis que peuvent produire des couches profondes de l'informatique quand elles ne sont pas réalisées avec le plus grand soin.

La mémoire flash contient un système de fichiers DOS classiques, comme dans vos clefs USB et cartes photo, avec ses fichiers et ses répertoires hiérarchiques. La description de cette organisation hiérarchique est stockée dans une structure de données arborescente, également sur la carte flash pour la rendre autonome. Quand le système démarre, il construit une copie de cette structure de données en mémoire vive pour pouvoir adresser rapidement les fichiers de la carte flash, la mémoire vive étant beaucoup plus rapide et moins énergivore que la mémoire flash. La source du problème était une mauvaise gestion de la destruction de fichiers dans la mémoire flash : quand on détruisait un fichier, l'espace qu'il prenait pour ses données était bien libéré, mais pas celui qu'il occupait dans le descripteur de la hiérarchie des fichiers. Donc ce descripteur stocké dans la mémoire flash grandissait continuellement avec la

création de nouveaux fichiers mais ne diminuait pas lors de leur destruction ultérieure. Sa copie en mémoire centrale reconstruite à chaque redémarrage grandissait donc d'autant. Comme beaucoup de fichiers avaient été créés et détruits pendant le long vol depuis la Terre, la taille de la seule description avait dépassé la taille disponible en mémoire centrale, deux fois plus petite que la mémoire flash.

C'est alors qu'un autre bug devenait actif : quand il n'y avait plus de mémoire disponible, la tâche DOS qui devait construire le descripteur se mettait en attente au lieu de signaler une erreur ; or elle avait pris un verrou exclusif (mutex) sur la mémoire, ce qui fait qu'aucune autre tâche ne pouvait plus s'exécuter : l'interblocage stoppait toute activité. Heureusement, comme pour Pathfinder, un processeur auxiliaire redémarrait le système s'il le voyait endormi trop longtemps. Mais la même chose recommençait peu après. Spirit était donc entré dans un cycle infernal de redémarrages, ce qui lui interdisait même de s'endormir la nuit et lui prédisait donc une vie courte !

Après avoir souffert pour détecter le problème, les ingénieurs ont pu envoyer une commande inversant un seul bit dans le système, afin d'exclure la carte flash du redémarrage. Spirit était de nouveau opérationnel, mais il avait perdu l'accès à sa mémoire flash. Il a ensuite été possible de vider petit à petit cette mémoire flash. Spirit s'est finalement retrouvé en pleine forme, seize jours après la première occurrence de la panne – et les ingénieurs aussi.

Spirit est mort le 22 mars 2010, après s'être enlisé trop profondément dans le sable fin 2009. Les ingénieurs de la NASA ont alors abandonné l'idée de le sauver après avoir beaucoup essayé. Cette fois, l'informatique n'y était pour rien.

LA DISPARITION DE MARS CLIMATE ORBITER

Ici, il s'agit d'un bug humain tout bête, aussi bête qu'un bug banal de logiciel. La sonde américaine Mars Climate Orbiter a disparu le 23 septembre 1999, lors de sa mise en orbite martienne. La cause était une ridicule erreur d'unités lors de l'émission d'une correction de trajectoire : le Jet Propulsion Laboratory, qui avait spécifié et géré le satellite, travaillait en système métrique, alors que Lockheed Martin Astronautics, le fabricant, travaillait encore en unités américaines, pieds, pouces, etc. !

Savez-vous que seuls trois pays n'ont pas adopté le système métrique comme seul système d'unités officiel : le Liberia, le Myanmar, et les États-Unis ?

LE CRASH DE SCHIAPARELLI

Le 19 octobre 2016, la sonde européenne Schiaparelli s'est écrasée sur Mars au lieu d'y atterrir doucement comme prévu. Il semble que l'altitude ait été vue comme négative à cause d'une oscillation trop forte perturbant un capteur, ce qui a provoqué la coupure des rétrofusées à 30 mètres du sol. Mais l'enquête n'était pas terminée au moment de l'écriture de ce livre.

Avionique et ferroviaire, des domaines critiques et sérieux

Les avions ont été informatisés depuis longtemps. Pour les avions de chasse, rendus instables dans les années 1950 afin d'augmenter leur maniabilité, c'était indispensable car aucun humain ne pouvait les piloter. Pour les avions de ligne, l'informatisation massive a été initiée par l'Airbus A320, dont le premier vol date de 1987. Elle a permis d'automatiser et de raffiner beaucoup de fonctions, désormais calculées par logiciel : commandes de vol devenues électriques, actionnant des servomoteurs en fonction de mesures de capteurs et de consignes envoyées par le pilote à l'aide d'un joystick à la place des traditionnelles commandes hydrauliques actionnées par le manche à balai et les pédales ; système de navigation évolué ; pilotage automatique couplé aux fonctions de navigation ; réglage fin des réacteurs pour diminuer la consommation ; gestion du freinage et de la direction au sol ; autotest en vol de toutes les fonctions avec détection et affichage de pannes ; instruments du cockpit pour les pilotes, et même système de musique et de vidéo pour les passagers (pas toujours bien réalisé, mais bien sûr moins critique). L'informatique est devenue ainsi un des points clefs de la sécurité. Par ailleurs, il ne faut pas oublier que les avions sont désormais conçus de façon entièrement numérique, comme nous l'avons vu au [chapitre 2](#) : les maquettes virtuelles 3D, la modélisation mathématique et les simulations informatiques ont largement remplacé les maquettes physiques et les essais en soufflerie en vol, et la simulation des vols se fait avec les logiciels qui seront embarqués dans l'avion.

D'autres fonctions logicielles moins visibles du public mais très importantes pour les compagnies et la sécurité ont été informatisées, en particulier l'autodiagnostic qui permet à l'avion de détecter et de transmettre au sol des demandes de maintenance précises à réaliser lors de la prochaine escale. Tout cela a été fait très soigneusement, en étant soumis à une norme mondiale de certification appelée DO-178B, qui définissait des contraintes portant sur la spécification, la réalisation et la validation des programmes informatiques en tenant bien compte de leurs spécificités. Initialement concentrée sur les aspects liés à la rigueur de l'ingénierie selon des critères plutôt administratifs, cette norme a été améliorée dans la nouvelle norme DO-178C qui autorise, voire encourage, les méthodes réellement scientifiques de vérification formelle de programmes que nous discuterons dans le prochain chapitre.

L'informatisation a joué un rôle prépondérant dans la diminution des accidents d'avions, hors bien sûr actions humaines délétères. La même chose s'est produite pour les trains et métros, car l'industrie ferroviaire est habituée depuis longtemps à des contraintes de sécurité draconiennes. La signalisation et la conduite ont été largement informatisées, et les métros automatiques sont maintenant partout. Dans toutes ces industries, les bugs sont plutôt rares et pris vraiment au sérieux.

Mais les accidents peuvent être dus à des difficultés liées à l'ergonomie des systèmes informatisés, qui ne sont pas vraiment des bugs informatiques bien que celle-ci y joue un rôle important. Lors du fameux accident d'un Airbus A320 mont Sainte-Odile en 1992, un des facteurs identifiés était une possibilité de confusion dans la lecture des écrans informatisés du cockpit. Un écran affichait un paramètre de descente qui pouvait être soit en degrés soit en mètres par seconde, mais la distinction n'était pas affichée de façon suffisamment claire sur l'écran et le pilote a rentré une consigne dans le mauvais mode. Il faut bien comprendre que l'ergonomie des cockpits d'avion est un sujet difficile, où l'informatique ne résout pas tout. C'est la même chose pour les tours de contrôle, où les traditionnelles bandes de papier échangées par les contrôleurs n'ont pu être remplacées de façon satisfaisante par des écrans informatiques que récemment, après des études ergonomiques très poussées.

L'automobile, un domaine hélas moins sérieux

Pour les voitures, les choses se sont passées bien différemment. L'informatique embarquée a été introduite par les équipementiers et non par les constructeurs eux-mêmes. Elle est d'abord arrivée dans les freins, composants critiques où elle apportait un plus réel : l'ABS (Anti Blocking System) permet de freiner bien mieux que ne peut le faire un humain, en agissant indépendamment sur les quatre roues et en évitant leurs blocages. Qui pourrait freiner avec quatre pédales ? Puis les applications se sont progressivement étendues : régulateur d'allure (*cruise control*), injection « électronique » commandée par logiciel, qui a permis d'augmenter le rendement des moteurs en gérant finement la quantité de carburant à envoyer et le moment de l'allumage, airbags, radar ou caméra arrière, climatisation, tableau de bord évolué, autoradio numérique perfectionné, etc. Si l'allure générale des voitures n'a pas changé fondamentalement depuis le xx^e siècle, avec toujours quatre roues, un volant et des sièges, les fonctions informatisées sont devenues progressivement plus riches. L'automobile évolue maintenant vers une assistance à la conduite de plus en plus perfectionnée avec le projet à court ou à moyen terme de fabriquer des voitures sans conducteur, donc vraiment autonomes.

Maintenant que toutes les voitures savent parfaitement accélérer, freiner et tenir la route – ce qui était loin d'être vrai il y a quarante ans –, la valeur ajoutée perçue se déplace de la mécanique à l'informatique. Je le vois très bien dans les publicités que je reçois par Internet (contre mon gré), qui ne parlent plus que de ça. Et, si

l'autonomie est le grand sujet du jour, son avenir ne sera peut-être pas le même partout, car il y a une grande différence entre les façons de conduire et de régler la circulation entre les États-Unis, l'Europe et l'Inde par exemple. Dans certains pays où l'automobile est ancienne et les routes faciles, les gens considèrent que conduire n'est plus une activité intéressante. Dans d'autres où les familles s'équipent à toute allure pour conduire sur des routes chaotiques, la plupart des conducteurs sont tout neufs et leur conduite est plus ou moins erratique. De plus, les systèmes légaux ne sont pas les mêmes partout, et les responsabilités respectives des constructeurs et conducteurs en cas d'accident ne seront certainement pas faciles à trancher. J'y reviendrai au [chapitre 11](#).

QUELQUES BUGS AUTOMOBILES

Il faut bien le constater, les bugs sont beaucoup plus nombreux en automobile qu'en avionique ou en ferroviaire. C'est peu connu, car les constructeurs ne communiquent bien sûr pas là-dessus ; mais les garagistes sont bien souvent perplexes devant une « panne informatique », très différente d'une panne mécanique ou électrique et difficile pour eux à caractériser puisqu'ils ne peuvent pas faire d'inspection visuelle ou auditive des programmes embarqués comme ils le font pour la mécanique ou les circuits électriques.

POURQUOI LE MOTEUR NE S'ARRÊTE-T-IL PAS ?

Passons aux travaux pratiques, en allant comme précédemment de l'anecdotique au vraiment sérieux. Un soir, je rentre fatigué du travail. Au moment d'ouvrir la porte de chez moi, je me rends compte que le moteur diesel de mon monospace plutôt ancien et rustique tourne toujours, alors même que j'ai la clef dans la main. Je pressens immédiatement le problème informatique, car je ne vois pas comment arriver à ce résultat avec de la mécanique et de l'électricité. Je reviens donc à ma voiture, note qu'aucune erreur n'est signalée sur le tableau de bord, et tourne la clef une bonne vingtaine de fois dans les deux sens ; le moteur daigne finalement s'arrêter. Le problème se répétant le lendemain, je décide d'aller dans un garage. Les garagistes me

disent qu'ils n'ont aucune idée de combien de temps la réparation va durer ni de combien elle va coûter, puisqu'il s'agit probablement d'informatique. Quand je reviens au garage le soir, je les vois arborer un grand sourire et me dire : « Cette fois, nous avons trouvé. Si votre moteur ne s'arrêtait pas, c'est à cause d'un problème dans le contacteur du feu stop sur la pédale de frein. On l'a changé, tout marche. »

J'en déduis immédiatement trois choses : d'abord que ma voiture utilise des bus informatiques de données partagés entre divers équipements au lieu de fils séparés pour chacun, ce qui est une bonne chose pour économiser le cuivre ; ensuite qu'une détérioration du contacteur du feu stop l'a probablement conduit à envoyer des parasites électriques dans le bus, qui n'attend que des signaux bien précis, et qu'aucune détection d'anomalie n'a été prévue pour ce cas ; enfin que le démarrage et l'arrêt du moteur utilisent le même bus de données que le feu stop, ce qui ne paraît pas vraiment conforme aux contraintes de sécurité habituelles dans les systèmes embarqués et m'inquiète quant aux analyses de fiabilité sous-jacentes. Ici, le bug n'est apparemment pas dans le code, mais dans le design du système global, ce qui ne change pas grand-chose.

Un problème similaire mais cette fois visiblement dû au seul logiciel a conduit Ford à rappeler pas moins de 433 000 voitures à cause de l'impossibilité de couper le moteur ; cette fois, il s'agissait simplement de mettre à jour le logiciel.

PORTES BLOQUÉES

Mais méfions-nous aussi de ces mises à jour. Vers 2007, une collègue me raconte que son oncle venait d'acheter un monospace d'une marque française réputée. Il était vite revenu au garage pour faire corriger un problème peu banal : quand il enclenchait la manette des essuie-glaces, il arrivait que cela allume les phares au lieu de lancer les essuie-glaces, et réciproquement. Le problème venait certainement d'une erreur de multiplexage des données dans un bus commun. Le garagiste lui a alors appliqué la procédure standard, qui est de télécharger les nouvelles versions de l'ensemble des logiciels embarqués. Il est resté au volant en attendant la fin de l'opération. Quand on lui a dit que c'était bon, il a mis le contact et a entendu « clonk » : toutes les

portes étaient bloquées, et plus aucune commande ne répondait. Il a fallu casser la porte pour le sortir.

Ce phénomène n'est pas rare : le ministre des Finances thaïlandais s'est retrouvé bloqué dans sa voiture allemande haut de gamme avant un discours important, sans possibilité d'en sortir puisque ces voitures ne peuvent plus s'ouvrir de l'intérieur. La solution a été de casser la fenêtre de l'extérieur, ce qui n'a plus rien de simple. Et un homme est mort en 2015 dans une autre voiture haut de gamme : le conducteur était parti et la voiture au soleil. Impossible pour lui de l'ouvrir, car le choix du constructeur avait été de ne plus pouvoir ouvrir les portes de l'intérieur sans la clef – trop de sécurité nuit à la sécurité. Il n'est pas nécessaire de citer les marques, car les mêmes équipements peuvent se trouver dans plusieurs d'entre elles.

LE CONTRÔLE MOTEUR MEURTRIER DES TOYOTA CAMRY

J'arrête là cette énumération que je pourrais continuer longtemps, pour passer à quelque chose d'une échelle très supérieure en nommant ici la marque : le fait avéré que plusieurs types d'automobiles (Camry et autres) vendues par Toyota aux États-Unis ont donné lieu à de nombreux accidents dus à des problèmes de logiciel et provoquant des décès ou des blessures graves⁵. Tous ces accidents ont eu un point commun : la voiture s'est mise à accélérer toute seule, sans aucune action possible de l'utilisateur jusqu'à l'accident. Le problème a été longtemps nié par Toyota, qui a soit incriminé les conducteurs soit fait des rappels pour effectuer des modifications mécaniques liées aux tapis de sol ou à la pédale d'accélérateur. Mais, à la suite de l'injonction d'un juge américain, il a été très bien analysé de façon indépendante par un groupe d'experts dirigé par Michael Barr, qui a étudié le logiciel de contrôle de façon particulièrement profonde dans un volumineux rapport rendu public en 2013. Le résultat a été implacable, comme le montre un extrait du rapport :

There are a large number of functions that are overly complex. By the standard industry metrics some of them are untestable, meaning that it is so complicated a recipe that there is no way to develop a reliable test suite or

test methodology to test all the possible things that can happen in it. Some of them are even so complex that they are what is called unmaintainable, which means that if you go in to fix a bug or to make a change, you're likely to create a new bug in the process. Just because your car has the latest version of the firmware – that is what we call embedded software – doesn't mean it is safer necessarily than the older one. [...] And the conclusion is that the failsafes are inadequate. The failsafes that they have contain defects or gaps. But on the whole, the safety architecture is a house of cards. It is possible for a large percentage of the failsafes to be disabled at the same time that the throttle control is lost.

« Il y a un grand nombre de fonctions beaucoup trop compliquées. Certaines ne sont pas testables selon les métriques standard de l'industrie. Les programmes sont si compliqués qu'il n'est pas possible de construire une méthodologie de test et une suite de scénarios suffisante pour tester tous les cas qui peuvent se produire. Certaines fonctions doivent être considérées comme non maintenables : corriger un bug ou changer quoi que ce soit serait susceptible de créer un nouveau bug ailleurs. Le fait d'avoir la dernière version du logiciel embarqué ne signifie pas qu'elle sera plus sûre que la précédente [...]. La conclusion est que les systèmes de sécurité sont inadéquats, car contenant des défauts ou des trous. Au total, on peut dire que l'architecture de sécurité est un château de cartes. Il est possible qu'une grande partie des garde-fous logiciels soient inopérants au moment où le contrôle de l'accélération est perdu. »

Par exemple, l'équipe Barr a montré que des variables cruciales liées au contrôle moteur pouvaient être écrasées par un débordement de la pile d'exécution, dont la taille maximale a été grossièrement sous-estimée. Dans ce cas, le contrôle de l'accélération peut totalement disparaître sans aucun moyen de le remettre en fonction.

Toyota a très longtemps nié les problèmes, apparemment sans même essayer de faire vraiment marcher ce logiciel critique. Les choses ont changé depuis 2014 : Toyota a été condamné à des millions de dollars d'amende dans des cas précis, a

ensuite payé 1,9 milliard de dollars pour terminer une *class action*, puis 1,5 milliard de dollars d'amende après avoir reconnu avoir menti à la justice. Des centaines de procès sont encore en cours. On peut espérer que Toyota ait décidé de réécrire ses logiciels avec d'autres méthodes.

POURQUOI L'AUTOMOBILE N'A-T-ELLE PAS ENCORE LE SÉRIEUX NÉCESSAIRE ?

Il y a plusieurs raisons à cet état de fait. J'ai déjà mentionné que c'était surtout un problème de schéma mental, mais je les détaille davantage ici. D'abord, l'informatique est un peu arrivée comme une intruse dans ce domaine resté dominé par la mécanique, donc la matière et l'énergie. Elle est surtout traitée par les sous-traitants (Bosch, Valeo, etc.), devenus de plus en plus autonomes, et beaucoup de constructeurs ont tardé à investir eux-mêmes le sujet alors que ce sont eux seuls qui possèdent la vue globale des problèmes potentiels. Ensuite, l'automobile diffère profondément de l'avionique et du ferroviaire par trois caractéristiques : premièrement, c'est une industrie très concurrentielle à grands volumes et faibles marges, donc essentiellement dirigée par les coûts ; ensuite, comme elle reste mentalement centrée sur la mécanique, il y est encore souvent jugé que développer des logiciels est beaucoup plus facile qu'innover en mécanique, et ne pose pas de problème particulier ; enfin, les problèmes et accidents de voitures sont dilués et ne font jamais la une des journaux, contrairement aux accidents d'avions ou de trains – d'ailleurs, la compréhension de tout ce qui concerne l'informatisation des objets reste bien faible dans les mondes du journalisme et de la communication.

Comme les temps et coûts de développement d'un logiciel jugé critique sont bien supérieurs aux coûts des logiciels développés de façon standard, comme aucune autorité de certification n'a encore réellement compris la criticité des logiciels automobiles, et comme les utilisateurs n'ont généralement pas la moindre idée de l'existence ni de la nature de ces logiciels, les systèmes automobiles ne sont toujours pas officiellement considérés comme critiques et les normes applicables sont bien plus faibles qu'en avionique et ferroviaire. Ce ne sont bien sûr pas les constructeurs qui poussent pour changer ça.

Or développer de façon fiable des logiciels embarqués complexes comme ceux des voitures demande un investissement humain, intellectuel et technique considérable, avec une formation spécifique des ingénieurs en logiciel embarqué, un soin minutieux lors de la conception, et des campagnes de tests longues et fastidieuses. Les ingénieurs compétents sont de plus difficiles à recruter à cause de l'appel d'air d'autres industries plus ouvertes et du fait qu'ils vont se retrouver en état d'infériorité par rapport à ceux de la mécanique (ce que j'ai pu constater personnellement). Et l'accent permanent sur les coûts conduit à travailler de façon peu outillée, en ne se procurant pas les outils sophistiqués utilisés par ceux qui développent des systèmes critiques légalement soumis à certification. De plus, n'ayant généralement aucune formation informatique, les clients tendent encore à penser que les bugs y sont une fatalité normale. Enfin, dans les accidents d'avions ou de trains, on désigne immédiatement des commissions d'enquête compétentes avec lesquelles les constructeurs sont tenus de collaborer. Ce n'est pratiquement jamais le cas pour l'automobile.

Un changement majeur pourrait cependant intervenir à cause du coût des bugs, qui pourrait devenir très important pour les constructeurs, et de l'entrée d'acteurs complètement nouveaux comme Tesla qui ne portent pas le poids du passé et sont directement branchés sur les évolutions les plus récentes de l'informatique et la Silicon Valley. Gardons donc les yeux ouverts, et poussons pour une meilleure qualité de l'informatique automobile.

Un bug médical historique : le Therac-25

L'informatique devient de plus en plus présente dans l'appareillage médical, à mon avis sans que le milieu médical ne se saisisse toujours des dangers spécifiques liés aux bugs de logiciel autant qu'il le fait pour ceux des médicaments. Je vais parler ici brièvement d'un cas historique, celui de la machine de radiothérapie Therac-25 fabriquée par AECL (Atomic Energy of Canada Limited) qui a provoqué ou accéléré la mort d'au moins 5 patients et gravement blessé plusieurs autres par irradiation massive de juin 1985 à janvier 1987. Même si cet exemple est maintenant ancien, son intérêt est qu'il a été analysé avec une grande précision dans un rapport écrit en 1993 par Nancy Leveson et Clark Turner ([Leveson & Turner, 1993](#)), dont une version condensée de Nancy Leveson fournit une lecture très instructive ([Leveson, 1995](#)).

Le Therac-25 pouvait fonctionner selon deux modes exclusifs, délivrant des faisceaux de rayons X ou d'électrons très précis en géométrie et en dosage. Pour cela, il utilisait comme base un accélérateur linéaire d'électrons. En mode faisceau d'électrons, le Therac-25 délivrait une puissance de 5 à 25 mégaélectronvolts (MeV), un filtre permettant de donner la forme souhaitée au faisceau. En mode rayons X, il délivrait une puissance fixe de 25 MeV, un appareillage spécial placé devant le faisceau d'électrons de la source le convertissant en faisceau de rayons X. Dans ce mode, l'accélérateur devait fournir une puissance beaucoup plus grande à cause du faible rendement de la conversion. Le filtre à électrons et le convertisseur en rayons X étaient placés sur une table tournante devant le faisceau principal. Il fallait donc

absolument garantir que la table était bien positionnée et la puissance de l'accélérateur bien réglée avant de lancer le faisceau, sous peine de surirradier massivement le patient.

Le Therac-25 était une amélioration d'une machine précédente, le Therac-20, dans laquelle le bon positionnement de la table tournante était garanti par des dispositifs mécaniques. Pour le Therac-25, le fabricant avait décidé d'informatiser le système avec une interface par clavier et écran informatique standard commandant numériquement la machine. Il avait aussi décidé d'en profiter pour remplacer les dispositifs mécaniques de sûreté par des dispositifs logiciels. Le logiciel était largement issu de deux machines précédentes, le Therac-20 et son prédécesseur le Therac-6, sauf bien sûr en ce qui concernait la sécurisation logicielle de la position de la table tournante. C'était encore l'époque où le logiciel était considéré comme la partie facile d'une telle machine : dans ses propres analyses de sûreté, AECL avait inclus la possibilité de pannes de l'ordinateur mais pas celle de bugs logiciels. De leur côté, les organismes d'homologation n'avaient pas éprouvé le besoin de valider les logiciels de commande, car ils étaient déjà utilisés en partie par les machines précédentes et garantis testés par le constructeur lui-même.

J'en viens maintenant aux accidents, tous en lien avec l'interface homme-machine et la sécurité gérée par logiciel. L'interface avait visiblement pas mal de défauts, dont celui de donner très souvent des erreurs du type « MALFUNCTION XX » où XX était un numéro d'erreur, sans que l'opérateur ne dispose de la moindre documentation sur ces erreurs. Dans ce cas, il pouvait appuyer sur la touche « ↑ » du clavier pour relancer l'opération. Au bout de cinq essais sans succès, il fallait redémarrer la machine. Les opérateurs s'y étaient apparemment habitués.

Un premier accident se produisit le 3 juin 1985 aux États-Unis, brûlant profondément une patiente avec des séquelles graves et irréversibles. La cause ne fut pas élucidée, l'accident ne fut pas rapporté à la FDA (Food and Drug Administration, en charge des dispositifs médicaux) et rien ne se passa jusqu'à l'accident suivant, même lorsque la patiente irradiée intenta un procès en novembre 1985. Un autre accident eut lieu en Ontario en juillet 1985. Quand le traitement fut démarré, la machine s'arrêta au bout de cinq secondes avec un message « NO DOSE », indiquant qu'aucune dose n'avait été délivrée. L'opérateur appuya sur « ↑ », reçut le

même message, et recommença la même chose quatre fois avant que la machine ne s'arrête en demandant son redémarrage. Mais le patient se plaignait de fortes douleurs, et les médecins comprirent vite qu'il avait été sévèrement surirradié les cinq fois. Cette fois, AECL, la FDA et le CRPB (Canadian Radiation Protection Bureau) furent prévenus. AECL fit quelques modifications aux capteurs de la table tournante, et affirma que la sécurité avait été améliorée « de 5 ordres de grandeur » ! Le CRPB demanda quelques modifications physiques et la suppression de l'usage de la touche « ↑ ». AECL se contenta d'abord de diminuer de 5 à 3 le nombre des appuis sur cette touche avant redémarrage, puis de conseiller aux utilisateurs d'enlever le capuchon plastique de la touche. D'autres accidents du même type eurent lieu ailleurs sans plus de conséquences sur l'usage de la machine. Le constructeur continuait à prétendre qu'il était *impossible* que sa machine conduise à des surirradiations intempestives.

Le premier diagnostic réel de la cause de tous ces ennuis eut lieu au Texas à la suite de deux accidents conduisant tous les deux à la mort ultérieure des patients due aux séquelles de l'overdose. Le même opérateur était en place pour les deux. Il était très entraîné et se servait habilement du clavier pour entrer ses données opératoires, en tapant vite et en utilisant systématiquement la touche « ↑ » pour passer rapidement de champ en champ sur l'écran, ce à quoi elle servait aussi. Se souvenant d'une séquence de frappes pouvant avoir provoqué le second accident, il réussit à faire taper suffisamment vite et à un rythme précis cette séquence à un ingénieur d'AECL, provoquant effectivement l'émission illégale d'un faisceau non protégé – mais sans blesser de patients cette fois. Le bug était donc dans la gestion du clavier !

La source de ce problème absurde était enfouie dans des couches systèmes, toutes développées par AECL qui avait choisi de ne pas utiliser les ressources fournies par le système d'exploitation natif de l'ordinateur PDP-11 embarqué dans le Therac-25, pourtant connu pour son sérieux et sa fiabilité. Quand on appuie sur une touche, le traitement logiciel à effectuer n'est pas instantané et peut dérouler un nombre relativement important d'instructions machine. Il est essentiel que ce traitement soit *atomique*, c'est-à-dire que l'ensemble de ces instructions soit exécuté avant de commencer quoi que ce soit d'autre. Sinon peut apparaître une course très dangereuse entre plusieurs activités qui devraient être rigoureusement étanches les unes des autres, aboutissant typiquement à la corruption de paramètres essentiels stockés en mémoire. Ce problème, connu depuis toujours, se gère parfaitement à l'aide de

sémaphores d'exclusion mutuelle ou *mutex*, comme nous l'avons vu pour Pathfinder et Spirit. La façon dont ceux-ci étaient implémentés dans le système d'exploitation dédié du Therac-25 était incorrecte.

Le rapport de Nancy Leveson et Clark Turner analyse avec précision le fonctionnement du logiciel et la cause du bug principal, sans toutefois que les auteurs aient pu consulter le code réel que n'a pas communiqué AECL. Il montre aussi qu'il y avait d'autres courses potentiellement dangereuses entre activités dans le logiciel. Il se termine par une analyse des causes fondamentales des dysfonctionnements du Therac-25 qui peut s'appliquer dans bien d'autres domaines où l'informatique est entrée par la petite porte : la confiance aveugle dans le logiciel par rapport au matériel, la sous-estimation de ses difficultés spécifiques, la non-prise en compte des bonnes pratiques de développement et de vérification du domaine, les dangers de la réutilisation aveugle d'un programme même s'il a fait ses preuves ailleurs (*cf.* [l'explosion d'Ariane 501](#) plus haut), l'importance de la qualité des interfaces homme-machine, des messages d'erreur et de la documentation, le caractère indispensable d'inclure aussi dans le logiciel des autodiagnostic et des filets de sécurité en cours de fonctionnement, le manque de vraies investigations à la suite des accidents, la négation permanente par le constructeur de l'existence de problèmes liés au logiciel, la tendance à mettre un sparadrap sur les effets plutôt que de rechercher les causes, etc.

Un an et demi après le premier accident, la FDA aux États-Unis et l'AECL au Canada ont exigé des modifications substantielles de la machine, y compris le retour à des protections mécaniques assurant la bonne protection de la table tournante, tout en admettant ne pas pouvoir assurer que le logiciel était désormais exempt d'erreurs. La machine a ensuite fonctionné sans plus d'accidents.

Les bugs d'outillage

Je viens de mentionner l'importance de l'outillage dans le développement du logiciel. On peut faire des bugs quand on utilise mal les outils, comme on l'a vu pour Pathfinder où l'héritage de priorité du très solide système d'exploitation VxWorks n'avait pas été armé. Mais on peut aussi être à la merci de bugs internes des outils, potentiellement doublement dangereux car invisibles à l'utilisateur.

LES COMPILATEURS C SONT-ILS FIABLES ?

Prenons l'exemple d'un compilateur C, qui traduit le langage C en langage machine. C'est certainement un des programmes les plus importants de toute l'informatique, utilisé probablement des millions de fois par jour. Toute personne qui écrit un programme en C, ce qui est très fréquent pour le logiciel embarqué, s'attend à ce que le programme en langage machine obtenu par la compilation de son programme fasse exactement ce qu'il a programmé. Or, un bug dans le compilateur peut conduire à un code machine faux sans que l'utilisateur ne puisse s'en apercevoir. C'est particulièrement vrai quand on utilise les options d'optimisation dont disposent tous les compilateurs modernes : pour accélérer le code machine, ils peuvent réaliser des transformations très profondes du programme⁶ ; manque de chance, ces transformations sont elles-mêmes très délicates à valider pour les écrivains des compilateurs.

Un projet de recherche américain très intéressant a montré les limites des hommes dans l'écriture des compilateurs, alors même que ce domaine fait partie des plus sérieux de l'informatique (Yang, Chen, Eide & Reger, 2011). À l'aide d'un système évolué de génération aléatoire de codes C nommé Csmith, les auteurs ont fabriqué un banc de test de 1 million de programmes spécifiquement destinés à casser les compilateurs C. Le résultat a été clair. Ils ont trouvé des centaines de bugs dans les grands compilateurs libres comme GCC et LLVM, massivement utilisés, ainsi que dans un bon panel de compilateurs industriels, y compris ceux utilisés pour les systèmes critiques. Tous ces bugs ne sont pas graves : si le compilateur lui-même se crashe ou émet un diagnostic d'erreur interne, pas vraiment de souci. Par contre, si le compilateur génère silencieusement un code faux, on ne sait pas du tout ce qui peut arriver à l'exécution du programme compilé. Un seul compilateur a survécu sans aucun bug : CompCert de Xavier Leroy (Inria). Normal, il a été *mathématiquement prouvé correct* ! J'y reviendrai au prochain chapitre.

QUID DES SYSTÈMES D'EXPLOITATION ?

Les systèmes d'exploitation ont un rôle central dans la gestion des machines. Ce sont des logiciels très complexes, avec beaucoup d'algorithmes parallèles et distribués. Ils sont donc particulièrement propices aux bugs.

Un bug particulièrement remarquable a eu lieu sur le bateau de guerre *USS Yorktown*, fleuron de la marine américaine. Son informatisation massive avait été décidée pour lui donner une flexibilité maximale et réduire les coûts d'opération. Elle a été faite en utilisant des ordinateurs interconnectés tournant sous Windows NT, le système phare de Microsoft à l'époque. Lors de sa première sortie suivant son informatisation, un opérateur a entré par mégarde le nombre 0 dans un champ de formulaire. Cela a provoqué une division par 0, erreur normalement anodine si elle est bien gérée. Cela n'a pas été le cas : tous les systèmes informatiques du bord se sont progressivement arrêtés, provoquant même l'arrêt de la propulsion, et il n'a pas été possible de les remettre en fonction ! Le bateau a dû être tracté jusqu'au port 😊.

Windows NT et son cousin Windows CE dédié à l'embarqué n'étaient certes pas réputés très solides à l'époque. Mais même les systèmes réalisés avec le plus grand

sérieux ne sont pas immunisés contre les bugs stupides. Linux est certainement l'un d'entre eux, avec son code public écrit, lu et relu par des professionnels du meilleur niveau. Pourtant, un bug énorme a été trouvé en 2016 dans une de ses parties critiques, le démon *systemd*, qui est le premier processus lancé par le système : si on lui envoie à n'importe quel moment un message vide par un canal de communication tout à fait standard, le système ne permet plus aux autres processus de se lancer et interdit même de redémarrer l'ordinateur. Le bug datant de 2004, il a perduré douze ans dans les distributions de Linux. Mais il est vrai que ce genre de bug devient de plus en plus rare et que Linux reste un modèle de sûreté.

Bugs de systèmes distribués

LE CRASH DU TÉLÉPHONE INTERURBAIN D'AT&T

Le 15 janvier 1990, à 2 h 25 de l'après-midi, le téléphone interurbain d'AT&T aux États-Unis s'est effondré, 60 000 personnes n'ayant plus d'accès au réseau et 70 millions d'appels n'ayant pu aboutir pendant la durée de la panne. La moitié des 144 centraux passaient leur temps à redémarrer, forçant les autres à traiter tous les appels. Cela surchargeait les centraux encore en état, qui s'effondraient pour la même raison. Le problème débuta à New York, puis se répandit comme une avalanche dans tout le pays. Les ingénieurs d'AT&T travaillèrent tout l'après-midi sans succès, aucune des procédures standard ne rétablissant le réseau. Ce n'est que le soir vers 23 h 30, à l'heure où tout devient calme, que le réseau put être remis en fonctionnement. Il fallut plusieurs semaines à AT&T pour détecter et corriger définitivement le problème.

Que s'était-il passé ? Juste un bug aussi minuscule que stupide, sous la forme d'une instruction pensée inutile mais en fait fausse ajoutée lors d'une modification des logiciels des centraux, installée en novembre de l'année précédente pour rendre le traitement des appels plus rapide. Je vais simplifier l'explication sans la rendre fausse⁷. Dans certaines configurations, cette instruction provoquait une corruption de la base de données du central. Cette corruption était détectée par un mécanisme parallèle de vérification, qui envoyait un message à tous les autres centraux puis redémarrait le central, opération faite en quelques secondes et immédiatement suivie

d'envoi de messages aux autres centraux. Mais le même bug joint à l'addition des messages d'arrêt et de redémarrage provoquait le même crash des receveurs, qui redémarreraient à leur tour. Rapidement, le réseau fut saturé d'une avalanche de messages de crash et de redémarrage emmêlés avec ceux des appels téléphoniques, qui arrivaient à un rythme d'autant plus soutenu que tout le monde réessayait ses appels, ce qui généralisait les boucles crash-redémarrage. Heureusement, par précaution, tous les centraux n'avaient pas reçu la nouvelle version du logiciel, et le réseau ne s'est pas totalement effondré.

Pour les connaisseurs du langage C : le bug était dû au remplacement par « le programmeur inconnu » d'une branche *else* absente d'un test *if* par « else break ; ». Le programmeur croyait que le break terminerait le *if*, ce qui n'est pas le cas en C. Manque de chance, le *if* était inclus dans un aiguillage *switch*, qui est lui terminé quand un *break* est exécuté. Le *break* du *if* a donc terminé le *switch*, supprimant ainsi l'exécution des instructions qui suivaient le *if*. Comme ces instructions avaient justement pour but de mettre la base de données en état, celle-ci s'est trouvée corrompue, le mécanisme parallèle de vérification de cohérence décidant alors de redémarrer le système. Ce n'est pas la première fois que « break » conduit à ce genre de bug. Nommer le point d'arrivée en écrivant « break B », où B est l'étiquette de l'instruction à arrêter comme dans bien d'autres langages, aurait été une excellente idée.

L'AUTODÉNI DE SERVICE DE FACEBOOK (?)

J'ai le souvenir d'un jour où, à cause d'un bug probablement du même tonneau, Facebook s'est trouvé en autodéni de service pendant quelques heures, c'est-à-dire noyé sous ses propres messages. Si jamais quelqu'un en a gardé une référence, je serai preneur !

Bugs de circuits

Les circuits sont conçus et fabriqués avec un très grand soin, beaucoup plus grand que pour la plupart des logiciels. En effet, au contraire des logiciels, il n'est pas possible de les corriger une fois qu'ils ont été fabriqués et installés dans des machines. Les constructeurs communiquent très peu sur leurs bugs, sauf quand ils peuvent être contournés par des astuces logicielles, ce à quoi les fabricants de compilateurs peuvent réagir par des modifications locales. Mais certains bugs sont d'une tout autre nature. Le plus célèbre est le bug de division du Pentium Pro d'Intel, dont voici une brève présentation.

LE FAMEUX BUG DE DIVISION DU PENTIUM PRO

Le Pentium Pro était le nouveau microprocesseur phare d'Intel. Quelque temps après sa sortie, Thomas Nicely, mathématicien américain spécialiste de théorie des nombres, mit en évidence un bug étonnant. La division flottante $1/824\,633\,702\,441$ rendait un résultat faux. Ce nombre n'était pas là au hasard, car 824 633 702 441 et 824 633 702 443 sont des nombres premiers jumeaux, le sujet qu'étudiait Nicely. Le bug est bien trop subtil pour être décrit ici, mais il est passionnant pour qui aime les nombres, les systèmes de numération redondants dans lesquels chaque nombre peut s'écrire de plusieurs façons différentes, et le calcul booléen ([Muller, 1995](#)).

Intel reconnut évidemment le bug, cette division étant facile à vérifier sur n'importe quelle autre machine, détecta la faille dans le programme qui avait

engendré le circuit, mais minimisa son importance en disant que le bug ne se produisait qu'extrêmement rarement. Mais cet argument ne marcha pas du tout, allant même jusqu'à ternir la réputation de la firme. Intel se trouva obligé de proposer le changement de tous les Pentium Pro livrés. Au total, le bug lui coûta officiellement 475 millions de dollars. Nous verrons au [chapitre 10](#) qu'un tel bug ne peut plus exister maintenant qu'on sait *prouver* les unités de calcul des microprocesseurs. C'est d'ailleurs ce bug qui a vraiment lancé le domaine de la preuve formelle de circuits arithmétiques, comme quoi les mauvais bugs ont aussi du bon ! Et Intel publie maintenant ses algorithmes de calcul et leurs preuves.

Des bugs aux trous de sécurité

Longtemps traitée un peu par-dessus la jambe, la sécurité informatique est maintenant prise au sérieux, grâce (si l'on peut dire) aux exploits des hackers, toujours plus forts pour pénétrer les systèmes informatisés afin de leur voler leurs données, chiffrer des données pour les rendre inutilisables et demander des rançons, recruter leurs capacités de calcul pour les transformer en robots pour les attaques massives, ou attaquer des systèmes physiques qu'ils contrôlent. Les *hackers* disposent de plusieurs points d'entrée dans les systèmes. Les principaux sont les mots de passe trop faibles ou stockés dans des endroits non protégés, les hameçonnages (*phishing*) par courriers électroniques trompeurs, et l'exploitation de bugs de logiciels apparemment anodins et souvent liés à des aspects non fonctionnels des systèmes et programmes attaqués. On peut dire que ces bugs font le lit des attaques principales, en particulier parce qu'ils sont répliqués à très grande échelle : si un bug existe dans un système d'exploitation, un navigateur Web ou tout autre programme utilitaire, tous les ordinateurs utilisant ces logiciels peuvent être infectés. Donnons-en quelques exemples.

LES ATTAQUES PAR DÉBORDEMENT DE MÉMOIRE

Hormis celles des virus qui savent exploiter des bugs profonds dans les systèmes d'exploitation mais sont trop compliqués pour être décrits ici, les premières attaques ont souvent été liées à des bugs des navigateurs Web, programmes souvent

développés « à l'arrache » pour gagner rapidement des parts de marché. Par exemple, Internet Explorer de Microsoft a souffert de très nombreux débordements de buffers, analogues au bug du mercredi présenté au début de ce chapitre. Prenons le cas d'une case d'un formulaire banal d'une page Web, conçu pour recevoir 16 caractères. Le développeur de la page doit normalement vérifier que le texte tapé ne fait pas plus de 16 caractères, sous peine d'écraser les mémoires qui suivent celles réservées pour ce texte. Dans ce cas, le ou les mots mémoires suivants peuvent être corrompus d'une façon suffisamment astucieuse pour permettre de prendre le pouvoir sur une partie du système, voire sur tout le système de l'ordinateur. C'est difficile à faire, mais très efficace et parfaitement discret. La détection et la correction de ce type de bug peuvent coûter extrêmement cher, les navigateurs étant des programmes gros, complexes, et en évolution permanente.

LES ATTAQUES PAR INJECTION LÉGALE DE CODES MALVEILLANTS

Les navigateurs Web, gestionnaires de courrier ou utilitaires comme les visualiseurs de documents PDF, constituent d'autres points d'entrée potentiels pour les hackers, car ils peuvent exécuter des programmes inclus dans les pages ou documents qu'ils traitent. En cas de bugs de ces outils, ces programmes peuvent exécuter des actions malveillantes sans que l'utilisateur s'en aperçoive, par exemple en lisant, détruisant ou cryptant des données auxquelles ils ne devraient pas avoir accès. Certaines attaques récentes comme celles par le virus Petya apparu en juin 2016 chiffrent non seulement les données, mais l'ensemble des logiciels de la machine, y compris le système d'exploitation.

Les bugs qui permettent d'entrer profondément dans les systèmes ou les outils qu'ils hébergent sont souvent assez simples mais pas détectables par les tests fonctionnels habituels. Remettre en état les outils avant que les hackers ne profitent de leurs trous reste très difficile, et beaucoup d'outils d'utilisation massive restent vulnérables. Il n'est pas difficile de les reconnaître : ce sont ceux qui bénéficient de mises à jour de sécurité fréquentes, qu'il est d'ailleurs absolument indispensable de

faire pour éviter les infections par des trous récemment ouverts par les hackers et refermés par les constructeurs.

LE DANGER DES SYSTÈMES D'EXPLOITATION OBSOLÈTES

Beaucoup de systèmes informatisés encore en activité ou même tout récents sont vulnérables parce qu'ils fonctionnent au-dessus de systèmes d'exploitation périmés, par exemple Windows XP dont Microsoft a abandonné la maintenance en 2014 en déconseillant formellement toute utilisation ultérieure. En mai 2017, l'attaque WannaCry s'est répandue à très grande allure en chiffrant le contenu d'ordinateurs tournant sur Windows XP ou sur des versions postérieures pas gardées à jour par les utilisateurs. De nombreux hôpitaux ont été touchés en Angleterre, et de nombreuses infrastructures industrielles dans beaucoup de pays. On a parlé de 300 000 « victimes ». Les utilisateurs ayant été largement prévenus, le mot « victime » ne me paraît pas adapté. Je préférerais « autruche ». Quand on met la tête dans le sable, on prend des coups de pied où je pense sans même savoir pourquoi. Et les problèmes des hôpitaux n'étaient pas nouveau en Angleterre : en mars 2016, lors d'un séjour à Londres, j'ai appris qu'un hôpital majeur a dû stopper toute action et évacuer ses patients pendant quarante-huit heures suite à une attaque massive sur son système informatique. J'imagine à peine la complexité de l'opération et la joie qu'elle procure aux malades et aux soignants. Les attaques contre les hôpitaux ont été également fréquentes aux États-Unis.

Enfin, fin juin 2017, je viens d'apprendre deux nouvelles étonnantes. La première concerne le *Queen Elizabeth*, le tout nouveau porte-avions britannique qui vient juste de sortir du port pour ses essais : ses systèmes informatiques tournent sous Windows XP, tout aussi obsolète sur l'eau que sur terre. La seconde nouvelle est que la police de Londres tourne aussi sur XP et est en train de migrer sur Windows 8.1 « Vista », de sinistre réputation lui aussi. Mais pourquoi donc cette obstination à construire les conditions de l'échec ? C'est probablement encore une fois parce que les logiciels d'application n'ont pas été conçus pour être bien portables d'un OS au suivant, ce qui n'est certes pas simple mais réalisé correctement dans beaucoup d'endroits...

LES FAIBLESSES DES PROTOCOLES DE SÉCURITÉ

Regardons maintenant les trous de sécurité dus aux connexions réseaux. Le fondement des communications sécurisées de type https pour le Web est le *chiffrement* des données avant leur transmission et leur *déchiffrement*, ces deux calculs étant réalisés par des opérations arithmétiques sur les messages numérisés reposant sur des choix de clefs numériques de chiffrement et de déchiffrement. Ces notions sont étudiées depuis très longtemps et donnent lieu à des mathématiques de très haut niveau. On pense souvent que les faiblesses des communications sécurisées résident dans les algorithmes de chiffrement, qui pourraient être cassés avec une puissance de calcul suffisante s'ils sont trop faibles. Cela peut être vrai pour certains algorithmes anciens dont les clefs ne sont pas de taille suffisante ou dont les principes ont été effectivement cassés par les mathématiciens, mais les chiffrements récents sont pour l'instant bien plus résistants. La question n'est d'ailleurs pas simple, car la sécurité du chiffrement moderne repose souvent sur des conjectures mathématiques exprimant que tous les algorithmes de décodage sans possession de la clef sont trop chers pour être implémentables en pratique, et car plusieurs des chiffrements disponibles (mais pas tous) pourraient être cassés par de futurs ordinateurs quantiques. Mais les vraies faiblesses ne sont généralement pas là. Elles sont le plus souvent dans les bugs des *protocoles de sécurité* qui préparent l'établissement des communications sécurisées.

Ces protocoles ont pour but l'établissement de canaux de communication cryptés entre leurs utilisateurs. Le plus connu est TLS (*transport layer security*) : vous le mettez en jeu dès que vous essayez de vous connecter à un site Web sécurisé dont l'adresse commence par « https:// », ce qui couvre entre autres tous les sites commerciaux et professionnels et donc des centaines de millions ou de milliards de transactions par jour. TLS négocie un mécanisme de chiffrement compris à la fois par le serveur et le client puis procède à l'échange des clefs de chiffrement correspondantes. Ces clefs doivent bien entendu rester secrètes, malgré les attaques possibles du protocole. Celui-ci doit en particulier résister aux attaques *man in the middle*, qui consistent pour l'intrus en l'immixtion dans la communication pour écouter les messages et émettre de faux messages en se déguisant en l'un ou l'autre des interlocuteurs.

TLS est un protocole complexe dont certains aspects restent encore fragiles. Il fait donc l'objet d'un grand nombre d'attaques. La plupart profitent d'une façon ou d'une autre des bugs ou faiblesses de conception qu'il a encore. Pour information, 240 modifications ont été apportées à TLS dans la seule année 2015, que ce soit en fonction d'attaques détectées ou grâce aux travaux des chercheurs, qui soit cherchent à l'attaquer directement, soit trouvent des bugs en échouant à le montrer correct. Je n'entrerai pas dans les détails, le sujet étant bien trop vaste et délicat pour être traitable ici. Mais je signalerai simplement trois attaques qui ont défrayé la chronique. La première est Heartbleed, due à un bug introduit dans la version publique très utilisée OpenSSL par un développeur bénévole en mars 2012 mais seulement détecté en avril 2014. La faille touchait à peu près 10 % des téléphones et tablettes fonctionnant sous Android. La deuxième est Freak ([Adrian et al., 2015](#)), créée par des chercheurs de Microsoft Research, de l'Inria et de plusieurs universités américaines. Cette attaque profite de règles de maintien de compatibilité édictées par la NSA (National Security Agency) américaine pour rester compatible avec de vieux protocoles facilement cassables car opérant avec des clefs limitées à 512 bits (c'était l'époque où la NSA souhaitait déchiffrer facilement le plus grand nombre possible de communications). Par une attaque du type *man in the middle*, il est possible de forcer une transaction à opérer avec un de ces vieux protocoles, puis de casser sa clef trop courte. On peut ensuite décoder toute la communication et envoyer de faux messages, ce qui est l'idéal pour un attaquant. La troisième attaque, du même type, est Logjam, due à une équipe de l'Inria Nancy, qui se fonde cette fois sur une faille du protocole lui-même et pas seulement de certaines implémentations, en amenant TLS à utiliser un vieux chiffrement utilisant seulement des nombres premiers de 155 chiffres décimaux, ce qui est insuffisant. L'attaque bénéficie tout particulièrement des progrès majeurs faits à l'Inria Nancy dans l'équipe de Patrick Gaudry sur les algorithmes de calculs rapides de fonctions mathématiques fondamentales pour le chiffrement. Le lecteur intéressé pourra consulter ([Adrian et al., 2015](#)).

LES ATTAQUES PAR LES OBJETS

Les dénis de service distribués (DDoS = *distributed denial of service*) font partie des attaques les plus violentes auxquelles sont confrontés les sites et services Web, voire des pays tout entiers. Elles consistent à noyer ces sites ou services sous un nombre gigantesque de requêtes envoyées simultanément par un très grand nombre d'ordinateurs, qu'il n'est même pas nécessaire de posséder pour les attaquants. Il est bien préférable de profiter des failles logicielles de l'ordinateur de tout un chacun pour y injecter un petit programme spécifique, le transformant ainsi en *bot*, ou robot d'attaque émettant autant de messages qu'il le peut. L'Estonie et la Géorgie ont subi des attaques massives de ce type qui les ont paralysées plusieurs jours.

Mais les ordinateurs sont de mieux en mieux protégés par leurs antivirus et *antimalwares* chassant les intrus logiciels. Il est devenu bien plus simple pour les attaquants d'infecter les objets connectés les plus variés, dont les microprocesseurs ont la puissance de calcul suffisante pour contribuer efficacement à des attaques massives. En 2016 a eu lieu une attaque de très grande envergure sur le site américain Dyn, qui est un très important site DNS (*domain name server*) traduisant les noms de site, comme www.college-de-france.fr, en adresses IP numériques comprises par le réseau. Les objets utilisés ont été les caméras vidéo de surveillance, installées à marche forcée ces dernières années ; leurs fabricants n'avaient pas pensé à cet usage détourné, ou avaient fait semblant d'ignorer le problème car ajouter des couches de sécurité aux logiciels embarqués est difficile, coûte cher, et n'était pas vu comme un argument de vente. Qui penserait qu'il va être infecté par ses caméras ou sa machine à laver ?

LES ATTAQUES SUR LES OBJETS

Les objets connectés sont aussi des cibles idéales pour les attaques. Je ne citerai que trois exemples parmi une liste qui s'allonge tous les jours.

J'ai parlé précédemment des bugs des voitures. Leurs problèmes de sécurité ne sont pas moins nombreux. Un exemple simple est l'ouverture des portes à distance. Au début, la télécommande envoyait un simple code à 10 ou 12 bits. N'importe quel électronicien ou informaticien compétent pouvait fabriquer un émetteur envoyant successivement à la ronde les quelques milliers de codes possibles et entendre les

déclics des voitures qui s'ouvriraient quand elles ont reçu leur code. Puis les constructeurs ont introduit des chiffrements, mais quelquefois de façon bien trop faible. On a vu ainsi des chercheurs montrer qu'on pourrait ouvrir certaines des voitures allemandes les plus réputées avec une simple application de téléphone portable. C'est bien que les chercheurs le montrent, mais combien de hackers avaient trouvé le même procédé avant ?

Plus grave, certaines voitures sont maintenant connectées, en particulier pour effectuer à distance des mises à jour des logiciels embarqués. Par exemple, dans les Jeep Cherokee, la connexion se fait par l'autoradio. Les chercheurs américains en sécurité Charlie Miller et Chris Valasek ont montré qu'ils pouvaient prendre le pouvoir à distance sur les fonctions essentielles de ces voitures depuis le navigateur Web de leur ordinateur, de n'importe où dans le monde. Ils pouvaient ainsi mettre les essuie-glaces et le lave-glace, mettre l'autoradio à la puissance maximum (ce qui veut dire quelque chose pour une voiture américaine), freiner à fond ou désarmer les freins, mettre le moteur à fond ou le couper, et, plus récemment, prendre le contrôle de la direction. Cela a conduit Chrysler à rappeler 1,5 million de voitures, et l'histoire n'est probablement pas finie...

La domotique est aussi un domaine où les objets connectés sont porteurs de promesses mais aussi de danger. Un bon exemple est donné par le système SmartThings de Samsung, destiné à écouter et commander tous les objets connectés d'une maison, de l'éclairage aux serrures en passant par la surveillance, la détection des fuites et l'électroménager. Ce système intègre une architecture de sécurité assez sophistiquée, vendue bien sûr comme invulnérable. Son installation ne peut se faire qu'en se connectant à un serveur hautement sécurisé situé en Corée. Mais des chercheurs de l'Université du Michigan ont mis en évidence de sérieux points faibles dans le système, tous dus à des bugs et à des analyses insuffisantes de l'architecture de sécurité ([Fernandes, Jung & Prakash, 2016](#)). Ils ont ainsi montré qu'ils pouvaient déclencher les alarmes à volonté, mettre le four à fond, et surtout injecter leurs propres codes dans les serrures pour entrer et sortir quand bon leur semble, ce qui est bien plus malin que de voler celui du propriétaire.

Dans le domaine médical, les attaques peuvent être très dangereuses. Une bien jolie concerne les pacemakers : en 2008, des chercheurs de la Johns Hopkins University ont montré qu'il n'était pas difficile de prendre le pouvoir total sur les

pacemakers d'une firme leader du domaine à l'aide d'une simple télécommande. La possibilité de dialoguer avec les pacemakers par télécommande est importante pour que les médecins puissent modifier leur paramétrage sans avoir à réopérer. Mais assurer une vraie sécurité est absolument indispensable : sinon, une personne malintentionnée pourrait arrêter le pacemaker, voire tuer le patient d'une décharge électrique. Tout récemment, des chercheurs ont examiné la sécurité de divers dispositifs médicaux corporels (pacemakers, pompes à insuline, etc.). Bilan : des milliers de trous de sécurité détectés...

MÉFIEZ-VOUS DES MESSAGES D'ERREUR !

En matière de sécurité, il faut se méfier de tout, même de messages d'erreur paraissant anodins. En voici deux beaux exemples.

Plusieurs sites Web et applications de téléphone ont été évidents à percer à cause d'un message d'erreur émis trop précocement lors de la frappe du mot de passe. Ils trouvaient malin de signaler l'erreur au premier caractère erroné du mot frappé. Pour simplifier, supposons que le mot de passe ait 8 lettres majuscules ou minuscules. Il y a dans ce cas 52 possibilités par position, donc 52^8 mots de passe différents, de l'ordre de 50 000 milliards. Pas si facile à casser, sauf bien sûr si votre mot de passe est dans le dictionnaire. Mais, si le message d'erreur est émis dès le premier caractère faux, trouver le mot de passe est trivial : il suffit d'essayer au plus 52 caractères pour la première position jusqu'à ne plus recevoir de message d'erreur, puis les 52 suivants une fois connu le premier caractère accepté, etc. Au total, il suffit d'au plus $52 \times 8 = 416$ essais pour casser le mot de passe. C'est faisable à la main avec un peu de patience, et immédiat avec un ordinateur.

Le deuxième exemple est plus subtil. C'est celui du nouveau passeport français, pour lequel un bug de sécurité a été trouvé par des chercheurs du groupe de sécurité CNRS/Inria dirigé par Véronique Cortier à Nancy. Les contraintes internationales stipulent qu'il ne doit pas être possible aux autorités de tracer le possesseur d'un passeport à travers ses entrées et sorties multiples. Les données du passeport sont cryptées et contenues dans une puce RFID interrogeable à (petite) distance, et l'analyse du passeport aux postes-frontières se fait à l'aide d'un protocole de sécurité

spécifique. Il suffit d'avoir accès à distance à une exécution de ce protocole à un poste-frontière, et de la rejouer à l'aide d'un appareil adapté qu'on peut transporter sur soi. En rejouant cette session normalement devenue obsolète, on obtient le code d'erreur « 6A80 : Incorrect parameters » si le passeport est à portée de l'appareil, et le code « 6300 : No information given » sinon. Donc, on sait si le porteur du passeport est là ou pas, sans même lui demander ses documents. Émettre un seul message au lieu de deux suffit à éliminer le problème. Mais la sécurité des passeports à RFID a bien d'autres faiblesses, et ils n'apparaissent pour l'instant pas vraiment plus sûrs que les bons vieux documents papiers...

CHAPITRE 10

Comment rendre l'informatique plus sûre

Après avoir présenté cette liste d'échecs ou de problèmes dus à des bugs identifiés comme tels après coup, abordons une des questions centrales de l'informatique : comment concevoir et mettre en place des systèmes informatisés qui marchent comme nous le souhaitons, sans être affectés, voire détruits, par les bugs que nous y avons nous-mêmes introduits ? Ceci suppose de répondre à trois questions : pouvons-nous vraiment caractériser ce que nous appelons un bug, pouvons-nous détecter et éradiquer les bugs dans tous les cas, ou devons-nous accepter de vivre avec ? Aucune de ces questions n'est simple et toutes méritent d'être abordées ici.

À l'heure actuelle, on leur applique trois types de réponses : d'abord, mieux travailler pour réduire le nombre de bugs dès la conception ; ensuite, vérifier en permanence son travail pour attraper les bugs avant la mise en service, soit par des batteries de tests, soit à l'aide de méthodes formelles présentées dans ce chapitre ; troisièmement, mettre des garde-fous dans les systèmes pour détecter les bugs résiduels suffisamment tôt avant que leurs conséquences ne deviennent catastrophiques, ce qui permet aussi de transmettre de bons rapports de bugs aux concepteurs. Tout projet bien conçu applique ces trois principes, mais, dans certains domaines hautement darwiniens de l'informatique où il faut travailler vite et jouer des coudes pour trouver de la place, beaucoup semblent encore les ignorer. Mais il y a ici

une grosse différence entre matériel et logiciel : les circuits sont figés dès qu'ils sont fabriqués alors que les logiciels sont toujours en mouvement, surtout depuis qu'on peut en télécharger régulièrement de nouvelles versions. Cet aspect dynamique ne facilite pas les choses : si une nouvelle version corrige en général certains bugs, elle peut aussi en ajouter de nouveaux.

Un préliminaire indispensable : caractériser les bons et mauvais comportements

Un bug ne se définit que négativement par rapport à un comportement souhaité. Une question préliminaire s'impose donc : savons-nous réellement définir l'ensemble des comportements souhaités, et caractériser les déviations par rapport à eux ? Même si cette question n'a pas de réponse générale tant les situations sont variées, on peut au moins caractériser plusieurs classes d'applications définissant de critères assez différents.

CAS 1 : ON PEUT SPÉCIFIER COMPLÈTEMENT L'APPLICATION

Ce premier cas est celui où l'on sait dire exactement ce qu'il faut faire et ne pas faire, autrement dit définir précisément les critères de succès et d'échec. Le problème y est bien posé et la notion de bug bien carrée. Parmi les bugs du chapitre précédent étaient dans ce cas tous ceux liés au temps, celui de l'arithmétique du Pentium Pro, celui du téléphone longue distance d'AT&T, ou ceux des compilateurs C.

Heureusement, ce cas n'est pas rare, car il concerne une grande partie de l'infrastructure et de l'outillage de base : architecture et composants des circuits numériques, bibliothèques de fonctions mathématiques ou de manipulation des

principales structures de données, algorithmes séquentiels d'usage général (tri, recherche en table, etc.), algorithmes distribués dans les machines multiprocesseurs et les réseaux (protocoles de communication, maintien de cohérence de mémoires et bases de données distribuées, compilateurs de langages de haut niveau, etc.). Nous verrons que les approches formelles de l'informatique commencent à y produire des succès considérables, voire à changer vraiment la donne.

CAS 2 : ON PEUT SPÉCIFIER DES PROPRIÉTÉS PARTIELLES À RESPECTER ABSOLUMENT

Ce second cas, très fréquent mais moins carré, est celui des programmes pour lesquels on ne sait pas dire dans tous les détails ce qu'il faut faire ou ne pas faire, mais où on sait en revanche donner des listes de contraintes sur le comportement à respecter absolument. En d'autres termes, seuls certains critères de succès ou d'échec essentiels peuvent être définis proprement.

On divise habituellement les contraintes à satisfaire en deux types : contraintes de *sûreté*, qui expriment que certains mauvais comportements ne peuvent pas exister, et contraintes de *vivacité*, qui expriment que certains bons comportements doivent absolument finir par se produire. Prenons l'exemple d'un ascenseur. Deux contraintes typiques de sûreté sont que les cabines ne doivent jamais voyager la porte ouverte et qu'elles ne peuvent pas continuer à monter après le dernier étage. Deux contraintes typiques de vivacité sont qu'au moins une cabine doit atteindre tout étage où l'ascenseur a été appelé et y ouvrir ses portes (partie de la contrainte qu'il est facile d'oublier pour un débutant), et que chaque cabine doit finir par atteindre tous les étages demandés par ses passagers par leurs appuis sur ses boutons internes, puis y ouvrir ses portes. Nous allons voir plus loin que les méthodes formelles ont là aussi fait de grands progrès pour définir rigoureusement ces contraintes et montrer qu'elles sont satisfaites, et même pour produire des contre-exemples sous forme de tests si elles ne le sont pas. Nous verrons aussi qu'elles permettent d'utiliser positivement cette idée de production de contre-exemples en générant automatiquement des tests complexes pour vérifier le fonctionnement normal, ce qui est très dur à faire à la main.

Deux difficultés supplémentaires apparaissent par rapport au cas 1. D'abord, les contraintes peuvent être nombreuses, et chacune d'entre elles peut être ardue à bien spécifier. Ensuite, il est difficile de savoir si on a bien listé toutes les contraintes souhaitées ou si on en a oublié (voir le bug d'Ariane 501 dans le chapitre précédent). Enfin, et ce n'est pas le plus facile, vérifier la satisfaction des contraintes exige presque toujours l'explicitation du comportement et des contraintes propres de l'environnement dans lequel l'application est exécutée. Par exemple, pour l'ascenseur, il est essentiel de bien expliciter la liste et les propriétés des capteurs et actionneurs avec lesquels le logiciel dialogue, mais aussi de traduire des lois physiques variées. Un bon exemple est qu'une cabine d'ascenseur allant de l'étage 2 à l'étage 4 doit nécessairement passer par l'étage 3, propriété tellement ancrée dans nos esprits qu'il est fréquent de l'oublier. Or cette propriété apparemment triviale est essentielle pour réduire le nombre de possibilités à explorer par les méthodes de vérification. Il n'est pas rare que la liste des contraintes d'environnement soit de taille comparable à celle des programmes, mais on peut deviner que bien plus de bugs apparaîtront si on ne la construit pas, les programmeurs pouvant faire des suppositions fausses sur l'environnement d'exécution. Peut-être le bug de la sonde martienne Schiaparelli brièvement présenté au [chapitre 9](#) était-il dans ce cas, si la lecture de l'altitude produite par le capteur indiquait effectivement qu'elle était négative, chose évidemment impossible.

Pour la librairie DOS qui a provoqué les ennuis de Spirit, le comportement logique était le bon et il n'y avait pas de problème de corruption mémoire. Mais une contrainte évidente après coup avait été oubliée lors de la conception : que la mémoire allouée pour les descripteurs dans la mémoire flash soit proportionnelle aux besoins du moment et pas à l'intégralité de l'histoire passée.

CAS 3 : LA SÉCURITÉ INFORMATIQUE ET LA RÉSISTANCE AUX ATTAQUES

Le cas de la sécurité informatique est particulièrement délicat. Typiquement, il faut montrer que les algorithmes employés pour une tâche quelconque sur une machine ou un réseau garantissent des propriétés de confidentialité et d'intégrité des

programmes et données pour cette tâche, sans non plus affecter les données et programmes des autres tâches exécutées par le système utilisé. Il faut aussi montrer que des communications entre composants du système et avec l'extérieur sont bien sécurisées, et que les autres programmes et données des machines auxquels on accède par réseau ne peuvent pas non plus être corrompus ([Abadi, 2011](#)).

La sécurité étant un sujet particulièrement difficile, les failles sont nombreuses dans les systèmes actuels. Les détecter est l'objet d'une course permanente entre trois populations : celle des agresseurs (hackers) qui cherchent à les exploiter directement, celle des chercheurs qui cherchent à les trouver et à les corriger avant que les agresseurs ne puissent les exploiter¹, et celle plus récente de ceux qui cherchent à les trouver en premier pour les vendre aux enchères, soit aux constructeurs soit aux agresseurs.

Malheureusement, il y a un hic profond dans ce domaine : il est par essence impossible de définir dans quel environnement d'exécution on se place, contrairement au cas 2 ci-dessus, car la seule supposition raisonnable en sécurité est que les adversaires sont *arbitrairement puissants* : l'agresseur peut intercepter ou détruire les messages échangés sur les réseaux, en forger de nouveaux, et aussi utiliser ce qu'on appelle des *canaux cachés*, c'est-à-dire des moyens d'attaque qui n'ont pas été prévus dans la conception du système ou qu'on ne sait pas encore éviter même si on les connaît.

Un bon exemple de canal caché est fourni par une clef USB qu'on ramasse par terre puis qu'on insère de bonne foi dans un port d'un ordinateur. C'est cela qui a permis la fameuse attaque des centrifugeuses nucléaires iraniennes et d'autres systèmes industriels par le virus Stuxnet en 2010. De façon plus subtile, Adi Shamir, grand maître du domaine, a pu casser une implémentation très utilisée de son propre chiffrement RSA, décrit dans l'[annexe 12.1](#). Il l'a fait en écoutant avec un simple smartphone le bruit de l'alimentation de l'ordinateur portable qui chiffre les messages envoyés et déchiffre les messages reçus. Fondé sur les nombres premiers, RSA fait des multiplications et des divisions qui engendrent une consommation électrique fortement variable du processeur de l'ordinateur. À son tour, cette consommation irrégulière provoque des vibrations des composants dans son alimentation électrique. Analyser finement le bruit audio associé a permis à Shamir de détecter les étapes cruciales du calcul et de reconstituer la clef de déchiffrement. Bon, tout le monde ne

sait pas faire ça, et même Shamir n'a pas pu proposer de correctif pour cette implémentation de RSA...

Le même type de difficulté se produit pour un vote « électronique », c'est-à-dire bien entendu géré par logiciel. J'ai entendu des élus nationaux me dire que c'est un sujet facile car il suffit de compter les oui et les non. J'ai immédiatement répondu que cette vue était totalement simpliste : qu'est-ce qui me prouve que mon vote a bien été pris en compte ? Que personne n'a pu le consulter ? Que personne n'a pu prouver à quelqu'un d'autre qu'il a voté ceci ou cela, et donc vendre son vote ? Que personne n'a pu entrer de faux votes dans le système par un canal caché, ce qu'on appelle dans le vote classique bourrer les urnes (le bourrage préalable des chaussettes ayant plusieurs fois servi de canal caché physique) ? Que le compte global est le bon ? Qu'on peut procéder à un vrai recomptage fiable en cas de besoin ? Qu'il n'y a pas de canaux cachés encore inconnus ? Les experts sont clairs : on ne sait pas encore quelle est la bonne liste de critères pour quel type de vote, ni construire des votes électroniques à très grande échelle aussi sûrs qu'un vote papier bien conduit avec ses dépouillements, ses assesseurs et sa traçabilité qui limitent les canaux cachés. Cela n'empêche pas que bien des votes électroniques soient mis en place en toute confiance de leurs organisateurs, qui s'obstinent souvent à ignorer ces questions. Or pas mal de bugs et de possibilités de canaux cachés ont été trouvés par les chercheurs du domaine dans les systèmes existants, par exemple dans les machines à voter américaines lors de l'élection entre George Bush et Al Gore en 2000, et même dans les élections internes du CNRS. Il n'est pas encore clair qu'on puisse construire un système de vote général vraiment solide. Mais la recherche se poursuit².

Pour vérifier la sécurité, il faut donc se restreindre et définir d'abord vis-à-vis de quelles classes d'attaques on se place. Quand c'est proprement fait, cela nous ramène au cas 2 ci-dessus. Mais il faut toujours se souvenir qu'il pourra toujours exister des canaux cachés, inconnus par construction comme leur nom l'indique.

**CAS 4 : NI LES COMPORTEMENTS NI LEURS
CONSTRAINTES NE SONT BIEN DÉFINIS *A PRIORI***

C'est le cas des systèmes qu'on ne peut pas vraiment concevoir et mettre en place sans une expérimentation intensive. Un bon exemple est l'*interface homme-machine* (IHM), composant essentiel mais hélas souvent négligé de la plupart des applications. Il est difficile de spécifier les bons comportements à réaliser et les erreurs à éviter sans expérimentation fine et intensive avec les utilisateurs finaux.

L'interface homme-machine est en fait intermédiaire entre la science et l'art. Beaucoup d'innovations y ont été dues à de longues expérimentations hors des sentiers battus. Par exemple, le poste de travail moderne avec sa puissance de calcul, son écran graphique, sa souris, ses disques locaux et sa connexion réseau avec les autres postes a été inventé et mis au point dans les laboratoires de Xerox Parc à Palo Alto (au Stanford Research Institute [SRI] pour la souris), puis dans d'autres laboratoires académiques et industriels de la Silicon Valley. Ces travaux, fondamentaux pour l'informatique moderne, ont donné lieu à plusieurs prix Turing : Douglas Engelbart (1997) pour l'invention de la souris et de diverses autres interfaces, Charles Thacker (2009) pour la conception matérielle des premiers postes de travail dotés de disques locaux et celle du réseau ethernet qui les relie toujours, et Butler Lampson (1992) pour celle des environnements personnels distribués. C'est la même chose pour le Web et les interfaces tactiles des téléphones, qui n'ont pas suivi les chemins tout tracés. Dans ce domaine, il y a d'autant moins de critères de succès que certains aiment ce que d'autres détestent et réciproquement : des goûts et des couleurs, on discute avec acharnement, contrairement au dicton. C'est aussi un cas où les bugs inévitables des prototypes peuvent finir par suggérer d'excellentes idées.

Il est important de constater que les interfaces homme-machine ne concernent pas que les ordinateurs et téléphones : les cockpits d'avion, les centres de commande d'usine, les postes de travail des aiguilleurs du ciel et les interfaces pour la chirurgie robotisée en sont d'autres exemples soumis à des contraintes de qualité bien plus fortes mais jamais faciles à expliciter. Souvenons-nous que l'accident de l'Airbus A320 qui s'est écrasé sur le mont Sainte-Odile en 1992 a été attribué en bonne partie à une présentation et une entrée ambiguë des données de vol sur les écrans du cockpit (voir le [chapitre précédent](#)). Pour les aiguilleurs du ciel, il a fallu beaucoup de temps et de travail pour construire des interfaces informatiques aussi fiables et performantes que leurs traditionnelles bandes de papier.

CAS 5 : LES ALGORITHMES D'APPRENTISSAGE

Voici maintenant un cas plus récent et bien différent : celui des algorithmes d'apprentissage automatique, expliqués au [chapitre 3](#) et dans l'[annexe 12.1](#). Limitons-nous pour simplifier au cas des réseaux de neurones profonds (en anglais *deep learning*).

Le comportement d'un réseau profond est vraiment différent de celui d'un programme classique, car on ne peut plus comprendre ce qu'il fait en lisant le programme qui le construit. Son comportement est en effet défini bien plus par les valeurs du grand nombre de coefficients évolutifs mis à jour lors de l'apprentissage que par son programme, lui relativement simple et statique. Mais les coefficients ne fournissent par eux-mêmes que peu d'information directement interprétable par nous pour comprendre ce qui se passe vraiment lors des calculs, et on ne comprend même pas encore pourquoi les réseaux profonds réussissent aussi bien.

De plus, sauf pour les jeux comme le go dans lequel la notion de victoire est bien définie, l'apprentissage s'applique à des domaines où l'on ne dispose pas de critères de succès ou d'échec vraiment clairs – comme pour les humains d'ailleurs. En pratique, on analyse le taux de réussite de ces algorithmes en les faisant tourner sur des jeux de tests de référence et en les comparant à ce qu'obtiennent des spécialistes humains sur ces mêmes jeux de tests. Nous avons vu au [chapitre 7](#) qu'ils obtiennent des résultats impressionnants en analyse d'images médicales. De même, les voitures autonomes guidées par des réseaux profonds analysant les données produites par des caméras ou des lidars semblent bien se comporter sur la route. Mais, en 2017, les systèmes d'apprentissage ne sont pas encore assez répandus pour qu'on puisse en dire beaucoup plus pour l'instant.

Presque rien de ce que j'ai raconté sur les bugs dans les quatre cas précédents ne semble valable ici, ce qui impose d'analyser les succès et échecs d'une façon bien différente et pas encore bien comprise. Comment va-t-on donner le permis de conduire à une voiture autonome, comment va-t-on qualifier les bonnes et mauvaises décisions de ses systèmes automatiques, et qui sera responsable des erreurs ou accidents ? Comment vont se comporter les conducteurs face aux voitures autonomes, que ce soit la leur ou celles des autres ? Vont-ils être capables de reprendre la main en cas d'imprévu, ou n'auront-ils plus la vigilance nécessaire, et sauront-ils quoi faire

s'ils ne savent pas du tout comment les automatismes fonctionnent ? De même, comment va se comporter un médecin face à un diagnostic automatisé, s'il n'est pas sûr d'être d'accord avec lui ? Comment l'expliquera-t-il au patient, et qui sera responsable des mauvais diagnostics ?

Toutes ces questions fort délicates devront être résolues par l'expérimentation, et impliquer beaucoup d'autres acteurs que les informaticiens. Mais, encore plus que pour le reste de l'informatique, il serait bon que suffisamment d'hommes politiques et de juristes cherchent à se construire les compétences suffisantes. La construction du bon schéma mental est ici aussi indispensable. J'en reparlerai au [chapitre 11](#).

Le génie logiciel

L'énumération de bugs et trous de sécurité ci-dessus du chapitre précédent donne bien entendu une image partielle de l'informatique, insistant trop sur ses mauvais côtés. Heureusement, il est tout à fait possible de construire des applications qui marchent bien, en particulier dans les deux premiers cas de l'énumération précédente. Pour cela, il est indispensable de maîtriser le *génie logiciel*, ensemble des méthodes et procédures qui permettent de conduire efficacement le développement et la vérification des applications, à condition que les ingénieurs et décideurs possèdent le bon outillage intellectuel et technique. La grande difficulté est que ce domaine n'est pas inné et ne ressort pas des formations scientifiques et techniques traditionnelles : la formation adéquate reste souvent très limitée, hors des spécialités informatiques de l'enseignement supérieur.

MIEUX SPÉCIFIER

Le premier et peut-être le plus difficile des problèmes se place au niveau des spécifications. On les décrit souvent comme une réponse à la question : « Que doit faire le logiciel ? », en pensant que quelques documents textuels ou graphiques suffisent à y répondre. C'est très insuffisant, pour plusieurs raisons. D'abord, l'expérience montre que des documents de ce type contiennent fréquemment des *incomplétudes* et des *incohérences*, jamais faciles à détecter. S'il y a des incomplétudes, c'est-à-dire l'omission de parties importantes des spécifications, ceux

qui implémentent le système sont forcés de faire des choix, conscients ou non, sans nécessairement réaliser qu'ils peuvent être incompatibles avec d'autres parties des spécifications. Une incomplétude fréquente est d'oublier de spécifier ce que le logiciel ne doit *pas* faire. Par exemple, c'est le choix fait à l'implémentation de ne pas utiliser le mécanisme d'héritage de priorité de VxWorks qui a conduit au bug de Pathfinder décrit au chapitre précédent. La spécification que le logiciel ne devra jamais se bloquer avait-elle été formulée de façon explicite avec des contraintes de vérification ? Je l'ignore. Les incohérences sont encore plus graves : des spécifications incohérentes ne peuvent en aucun cas conduire à un bon produit. Mais tout ne doit pas pour autant être spécifié en détail : par exemple on doit laisser le choix aux implémenteurs d'utiliser tel ou tel algorithme pour réaliser telle ou telle fonction bien spécifiée. Le bon équilibre n'est pas facile à trouver.

Il ne faut jamais oublier que les spécifications ne doivent pas se contenter de décrire la tâche à effectuer. Elles doivent aussi décrire précisément dans quel environnement cette tâche doit s'exécuter et définir des contraintes non fonctionnelles comme des limites en temps de calcul, taille mémoire et énergie consommée, surtout lorsque ces contraintes sont critiques pour la correction pratique de l'implémentation. Par exemple, pour les applications temps réel liées au monde physique, « avant l'heure, c'est pas l'heure, après l'heure, c'est plus l'heure » : quand on appuie sur la pédale de frein d'une voiture, l'effet doit être quasi instantané, et quand on veut que deux notes de musique apparaissent simultanées, leurs ordres de déclenchement ne doivent pas être séparés de plus d'une quinzaine de millisecondes. Les spécifications doivent aussi définir ce qui doit se passer lorsque les commandes envoyées au système ne sont pas conformes à ce qui est attendu, mais quand même possibles. Beaucoup de trous de sécurité majeurs résultent d'un manque d'analyse de cet aspect.

Ajoutons que spécifier la vérification est aussi important que spécifier la programmation. Pour cela, de bonnes descriptions de l'environnement et de ses particularités sont indispensables pour définir les bons tests à effectuer, en particulier pour les systèmes critiques. Un exemple typique d'étonnante absence de maîtrise de cet aspect a été d'oublier de spécifier qu'il fallait tester le passage de la ligne de changement de date pour le bug des Raptor décrit au chapitre précédent.

Enfin, une des causes récurrentes de l'échec de développements informatiques est le *changement permanent des spécifications*, très fréquent quand les donneurs d'ordre

n'ont pas eux-mêmes une connaissance suffisante de la difficulté et de la lourdeur de l'informatique : changer les spécifications est un acte léger pour eux, mais pas pour les implémenteurs chez qui mettre à jour l'implémentation et les tests est toujours un acte lourd. Si, comme c'est fréquent, cela conduit à une accumulation de rustines du fait que les contraintes de délais de livraison ne permettent pas de vraiment réfléchir, l'échec se produit inmanquablement. La liste des projets ayant échoué de cette façon après avoir dépensé des sommes considérables est interminable, dans l'industrie, l'administration, et même la recherche.

Enfin, une question toujours délicate est la suivante : les spécifieurs et les implémenteurs doivent-ils être des personnes différentes ? Cela dépend des applications. Par exemple, pour le pilotage d'un avion, cette distinction est la plupart du temps naturelle : les spécifieurs sont des automaticiens, spécialistes du contrôle, et les implémenteurs des informaticiens. Il est rare que des individus maîtrisent suffisamment les deux aspects, et ni la spécification ni l'implémentation ne peuvent être faites par des individus seuls, même hypercompétents. C'est la même chose pour les circuits électroniques entre les architectes et designers (programmeurs en langage de description de circuits). En revanche, dans des domaines où la programmation et l'expérimentation sont intimement reliées comme l'IHM (interface homme-machine), ce sont souvent les mêmes individus qui gèrent les deux aspects. Jean-Marie Hullot, principal créateur des interfaces de la machine NeXT puis du système Mac OS X du Macintosh et de l'iPhone d'Apple, est à la fois un artiste des interfaces et un programmeur hors norme³.

LA GESTION INFORMATIQUE DES SPÉCIFICATIONS, PROGRAMMES ET TESTS

Pour tout projet de quelque envergure, gérer proprement et donc en machine les spécifications, programmes et tests est absolument indispensable. Ça l'est en particulier pour garder une mémoire précise des développements permettant de comprendre pourquoi telle ou telle décision a été prise, afin de pouvoir revenir en arrière en cas de mauvaise décision. C'est tout aussi essentiel pour conduire les indispensables *tests de non-régression*⁴, qui permettent de vérifier que

l'implémentation de nouvelles fonctionnalités ou la modification de fonctionnalités existantes ne modifie pas ce qui fonctionnait déjà. Heureusement, on dispose maintenant d'excellents outils pour cela, qui sont de plus en plus utilisés dans les projets industriels et académiques sérieux.

ASSURER LA TRAÇABILITÉ DE LA SPÉCIFICATION À L'IMPLÉMENTATION

L'exigence de *traçabilité* des décisions de la conception à la réalisation est une contrainte essentielle de toutes les procédures de certification de logiciels. Elle demande d'expliquer pour chaque décision de programmation à quelles spécifications elle correspond, pour chaque test quelle fonction il teste dans quelle configuration prévue par les spécifications, etc. Elle n'a vraiment de sens que si de bons documents de spécification existent. Il est probable qu'elle a été très insuffisante dans le cas du contrôle moteur de Toyota étudié au chapitre précédent. L'affirmation répandue « la spécification, c'est le code » est particulièrement dangereuse puisqu'elle interdit tout type de traçabilité des choix d'implémentation.

CONDUIRE DES REVUES EXTERNES

Les concepteurs et réalisateurs d'un projet ne sont pas les mieux placés pour analyser la façon dont ils travaillent et les résultats qu'ils produisent. Il est souvent utile d'organiser des revues et vérifications indépendantes par des consultants ou organismes extérieurs. C'est d'ailleurs obligatoire dans les domaines certifiés comme l'avionique, le spatial, le ferroviaire ou le nucléaire – mais hélas pas pour l'automobile. L'échec d'Ariane 501 et les accidents provoqués par le Therac-25 et le contrôle moteur de Toyota sont significatifs de ce point de vue : les expertises qui ont eu lieu après ont clairement montré à quel point il aurait été utile de les conduire avant les accidents.

Du test aux méthodes formelles

Le test est la méthode la plus utilisée pour mettre au point les applications informatiques. Il a plusieurs rôles : vérifier la correction des comportements prévus, vérifier tout ce qui peut se passer lorsqu'on envoie des commandes imprévues (ce qui est hélas souvent oublié) et chasser les bugs sournois qui se cachent dans les coins d'ombre. Il est tout aussi indispensable pour expérimenter différents designs possibles du système à réaliser, en particulier pour l'interface homme-machine.

LE TEST PAR SIMULATION DU SYSTÈME GLOBAL

Pour les gros projets et les systèmes cyberphysiques, le test est souvent conduit par des équipes spécifiques distinctes des équipes de conception, ce qui est d'ailleurs obligatoire pour les systèmes certifiés. Il peut demander la construction d'une infrastructure logicielle et matérielle spécifique qui peut être fort complexe.

Par exemple, en avionique, on construit des simulateurs très perfectionnés qui permettent de « faire voler numériquement » l'avion en partant de sa conception mécanique 3D et de modèles mathématiques de son comportement aérodynamique. Les maquettes physiques disparaissant petit à petit, la mise au point de l'ensemble du système se fait de plus en plus sur des simulateurs numériques, qui servent aussi à entraîner les pilotes et les mécaniciens. Un avantage est de pouvoir simuler un grand nombre de pannes ou de situations difficiles, ce qui est souvent impossible en vol réel.

Pour un grand système distribué, on peut commencer par des tests mettant en jeu un nombre restreint de sites, afin de pouvoir détecter et corriger des problèmes qui deviendraient inanalysables en taille réelle. Pour les circuits électroniques de petite taille, on peut procéder soit à des simulations logicielles, soit à des émulations matérielles par FPGA (*field-programmable gate arrays*), qui sont des circuits dont les unités logiques et les connexions sont reprogrammables par logiciels. Pour les systèmes sur puces (SoC) de grande taille qui intègrent différents composants cadencés par des horloges différentes et reliés par des réseaux, l'utilisation d'émulations FPGA devient impossible, mais peut alors utiliser des simulations logicielles à haut niveau d'abstraction où le comportement des unités est représenté par des programmes logiciels compacts et où la transmission d'information entre unités est gérée par des protocoles logiciels ; cela permet de gagner plusieurs de grandeur sur la vitesse de simulation, et surtout de tester les logiciels d'application avant même que les circuits physiques n'existent⁵.

TESTER OU PROUVER ?

Mais le test garde une insuffisance fondamentale qu'il ne faut jamais ignorer, et qui a bien été mise en lumière par une fameuse citation d'Edsger Dijkstra (prix Turing 1962) :

Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence. The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness.

« Le test de programmes est une manière très efficace de montrer la présence de bugs, mais il est désespérément inefficace pour montrer leur absence. La seule façon vraiment efficace d'augmenter significativement la confiance dans un programme est de donner une preuve convaincante de sa correction. »

Notons en passant que cette affirmation n'est pas vraiment nouvelle. Comme l'explique Gilles Dowek dans le livre *Les Métamorphoses du calcul* ([Dowek, 2015](#)), l'invention des mathématiques par les Grecs pourrait être en partie due à l'idée de *prouver* la correction d'algorithmes déjà bien connus en s'appuyant sur la logique naissante au lieu de la vérifier sur chaque exemple.

Les assertions logiques

Pour l'informatique, le papier fondateur de l'idée de preuve de programme est encore une fois dû à Alan Turing. Dans une note de 1949 à propos de l'implémentation du calcul de la factorielle en langage machine, il exprimait déjà l'idée que la meilleure façon de mettre au point le programme était de le truffier d'*assertions logiques* qui entraînaient la validité du résultat pour toute entrée, puis de prouver mathématiquement ces assertions en exécutant le programme de façon symbolique au lieu de le faire sur des arguments numériques particuliers.

Turing établissait ainsi un lien fort et applicable en pratique entre la programmation des ordinateurs et les mathématiques. Clarifier, perfectionner et automatiser ce lien allait devenir le sujet de beaucoup de recherches, dont certaines des miennes. L'idée de Turing a ensuite été reprise en 1967 par Robert Floyd (prix Turing 1978), puis étendue et raffinée par Edsger Dijkstra (prix Turing 1962) et Charles Anthony Hoare (prix Turing 1980). Les assertions constituent maintenant une façon fondamentale d'intégrer la vérification à la programmation, pouvant compléter le test et même parfois le remplacer totalement. Elles se sont avérées tout aussi utiles une fois les logiciels mis en service : elles peuvent être vérifiées à la volée lors de chaque exécution particulière, permettant ainsi de détecter un bug dès qu'il se produit au lieu de ne pouvoir observer que ses conséquences lointaines. Cela permet d'émettre des messages d'erreur bien plus pertinents, et, si besoin, de lancer des procédures de sécurité pour éviter des accidents.

UN EXEMPLE SIMPLE

```
int fact (int n) {  
    n ≥ 0  
    int r = 1, i = 0  
    r = i!  
    for (i := 1 ; i <= n; i++) {  
        r = (i-1)!  
        r := r*i ;  
        r = i!  
    }  
    i = n, r = n!  
    return r ;  
}
```

Figure 70. Les assertions de la factorielle.

Pour expliquer comment fonctionnent les assertions, il faut un minimum de technique – du niveau lycée rassurez-vous. La figure 70 montre un programme de calcul itératif simple de la factorielle d'un entier positif n , notée en mathématique « $n!$ ». Dans la figure, le code exécutable est écrit en noir, alors que les assertions sont écrites en italique bleu, les symboles italiques n , i , r dénotant les valeurs mathématiques courantes des variables informatiques de même nom. Le calcul se fait par une itération impliquant un entier i et un accumulateur r , la boucle « for » se lit ainsi : en partant de 1, et tant que i est inférieur ou égal à n , calculer le corps de la boucle puis incrémenter n .

Les assertions bleues sont des *prédicats*, c'est-à-dire des formules logiques à valeur *vrai* ou *faux* sur les valeurs des variables. Elles encadrent les instructions. Placées avant une instruction, elles s'appellent des *préconditions* pour cette instruction ; placées après, elles s'appellent des *postconditions*. Comme les instructions se suivent syntaxiquement, la postcondition d'une instruction est aussi la précondition de la suivante, celle qui est exécutée en séquence. L'interprétation est

simple : pour chaque exécution de chaque instruction, on doit montrer que si la précondition de l'instruction est vraie avant l'exécution, alors sa postcondition est aussi vraie après – à condition toutefois que l'instruction se termine et ne parte pas dans une boucle infinie.

Pour la factorielle, la précondition globale est n positif ou nul, ce qui s'écrit mathématiquement $n \geq 0$. Considérons maintenant la boucle « for ». Avant l'entrée dans la boucle, on a $r = 1$ et $i = 0$, donc $r = i!$ puisque $0! = 1$. Donc la première assertion $r = i!$ du corps du « for » est vraie juste après l'entrée dans la boucle. Considérons maintenant le corps de la boucle, qui est l'instruction « $r := r * i$; » où $*$ dénote la multiplication⁶. Il est mathématiquement vrai que $r = (i-1)!$ avant l'exécution de cette instruction implique $r = (i-1)! \times i = i!$ après son exécution. Comme l'assertion $r = i!$ est vraie à l'entrée de la boucle et est préservée par toute exécution du corps de la boucle, elle reste vraie quel que soit le nombre de tours de boucles, et donc aussi quand la boucle se termine – si jamais elle se termine. Heureusement, la terminaison de la boucle est simple à prouver ici : comme i part de 0 et augmente de 1 à chaque tour de boucle, il finira par valoir n , ce qui terminera la boucle. À ce moment, r , qui devient le résultat de la fonction, vaut $n!$. CQFD.

LES ASSERTIONS COMME SPÉCIFICATIONS

Les assertions sont aussi précieuses pour la spécification. Par exemple, l'assertion d'entrée $n \geq 0$ et l'assertion de sortie $r = n!$ plus la contrainte de terminaison du calcul sont suffisantes pour spécifier complètement ce que doit faire un programme de factorielle, sans le moins du monde indiquer comment le calcul doit être réalisé. On est ainsi dans le cas 1 ci-dessus des systèmes totalement spécifiables formellement. Certains langages comme Eiffel de Bertrand Meyer promeuvent ce type d'assertions sous le nom de *contrats* et les utilisent pour guider à la fois la spécification, la programmation et la vérification modulaires. Mais, dans beaucoup de cas, et en particulier dans les systèmes entrant dans le cas 2 ci-dessus, les assertions se contentent de parler de certaines contraintes critiques sans être suffisantes pour définir une spécification complète du problème à résoudre.

L'AUTOMATISATION DU CALCUL DES ASSERTIONS

L'exemple de la factorielle étant très simple, il est traitable à la main. En pratique, les programmes sont beaucoup plus gros et les assertions beaucoup plus compliquées, ce qui fait qu'aucun humain ne peut conduire une preuve vraiment crédible. Il faut donc trouver un moyen de mécaniser leur vérification, c'est-à-dire les faire prouver par un autre programme. Beaucoup de travaux ont été dédiés à cette question, dont le travail de Patrick et Radhia Cousot sur l'interprétation abstraite, déjà mentionné au chapitre précédent à propos du bug d'Ariane 501. Plus récemment, beaucoup de systèmes de vérification d'assertions s'appuient sur les systèmes de démonstration automatique de théorèmes et les assistants interactifs de preuve que je vais aborder dans la suite de ce chapitre.

Des systèmes de vérification sont de plus en plus souvent couplés aux langages et à leurs compilateurs. Citons par exemple l'outillage d'Eiffel, le langage F# de Microsoft Research qui intègre complètement programmation et vérification, et le vérifieur KeY développé en Allemagne et en Suède pour les très populaires langages Java et Python.

Je voudrais terminer cette discussion sur les assertions par un exemple significatif. Écrit en Java, le très astucieux programme Timsort est un tri remarquablement efficace développé en 2002 par Tim Peeters, puis devenu un standard pour les langages Java et Python et pour le système d'exploitation Android. Il est donc utilisé tous les jours à très grande échelle. Des chercheurs ont essayé de prouver que sa version Java était correcte à l'aide du vérifieur KeY. N'y parvenant pas, ils ont approfondi leur analyse et trouvé un joli bug⁷, qu'ils ont corrigé avant de prouver la correction du programme modifié.

ASSERTIONS TEMPORELLES ET VÉRIFICATION DE MODÈLES

Pour beaucoup de systèmes informatiques, et en particulier les circuits électroniques et les systèmes cyberphysiques, des assertions faisant référence au temps constituent la façon naturelle de spécifier et de vérifier des propriétés de sûreté et de vivacité définies plus haut. Reprenons notre exemple d'ascenseur. Nous exigeons

une propriété de sûreté : une cabine ne doit *jamais* voyager la porte ouverte. En mathématiques, ce genre de propriété s'exprime classiquement en introduisant une variable temporelle t pour parler explicitement du temps et en définissant le fait de voyager et l'état d'ouverture de la porte comme deux prédicats mathématiques fonctions du temps $\text{voyage}(t)$ et $\text{ouverte}(t)$, vrais ou faux à chaque instant t . Pour exprimer notre contrainte dans ce vocabulaire, nous pouvons écrire la formule mathématique suivante, où « $\neg$ » désigne la négation :

$$\forall t. \text{voyage}(t) \Rightarrow \neg \text{ouverte}(t)$$

Mais nommer explicitement t introduit de la lourdeur pour pas grand-chose. La *logique temporelle* permet de simplifier la formule en remplaçant la référence explicite au temps par une *modalité* temporelle « toujours », notée « $\Box$ ». On écrit alors :

$$\Box(\text{voyage} \Rightarrow \neg \text{ouverte})$$

Cette formule se lit de la même façon : « il est vrai à chaque instant que le fait de voyager entraîne que la porte n'est pas ouverte ». Nous avons aussi requis une propriété de vivacité : quand il est appelé à un étage, l'ascenseur doit finir par s'arrêter à cet étage. Cela s'écrit simplement à l'aide d'une modalité temporelle duale notée « $\Diamond$ » pour « un jour », en utilisant la variable entière n pour les numéros des étages :

$$\Box(\text{appel}(n) \Rightarrow \Diamond \text{arrêt}(n))$$

Cette seconde formule se lit : « il est toujours vrai qu'après avoir été appelé à un certain instant à l'étage n , l'ascenseur finira un jour par s'arrêter à cet étage ». Elle serait plus lourde en notation mathématique classique :

$$\forall t. \text{appel}(t, n) \Rightarrow \exists t' > t. \text{arrêt}(t', n)$$

La logique temporelle a été étudiée par beaucoup de chercheurs, et en particulier par Amir Pnueli, prix Turing 1996 pour ses contributions. Dans le monde de la conception des circuits électroniques, elle a donné lieu à un standard international nommé PSL (*property specification logic*). Elle est très élégante pour les formules simples, souvent suffisantes en pratique, mais les formules deviennent vite difficiles à écrire juste et même à lire dans les cas un tant soit peu compliqués. Pour le convaincre, je laisse au lecteur le soin d'ajouter à la formule précédente une seconde contrainte de vivacité plus fine pour l'ascenseur : une fois arrivé à un étage, l'ascenseur doit y ouvrir puis refermer ses portes avant de repartir.

Un domaine important de la vérification formelle se fonde sur l'écriture de formules temporelles appliquées aux *systèmes de transition finis*, formalisés par des graphes ayant pour nœuds les états du système et pour arcs orientés les transitions entre ces états. Ce type de formalisation est très utilisé pour les protocoles de communication, les systèmes cyberphysiques, et les algorithmes distribués. Le domaine est appelé la *vérification de modèles*, *model-checking* en anglais. Il a donné le prix Turing 2007 à Edmund Clarke, Allen Emerson et Joseph Sifakis (seul prix Turing français pour l'instant). Les algorithmes de vérification associés sont de deux types : ceux travaillant par exploration explicite efficace des états individuels du système, utilisés par le système SPIN de Gerard Holzmann (créé aux Bell Labs) et le système CADP de l'équipe d'Hubert Garavel à l'Inria, et ceux utilisant des techniques de calcul symbolique sur des ensembles d'états du système décrits implicitement par des formules logiques, introduits par Kenneth McMillan (Carnegie Mellon University puis Cadence Design Systems) dans les années 1990. La vérification symbolique a l'avantage de ne pas être limitée par le nombre d'états des systèmes pratiques, qui peut être gigantesque. Elle a été beaucoup développée depuis en utilisant les outils de démonstration automatique dont nous allons parler dans la suite.

Un autre domaine est celui des systèmes *temporisés*, c'est-à-dire dont l'évolution est régie en partie par le temps qui passe. Dans ce cas, les assertions temporelles doivent parler non seulement de la succession des événements mais aussi de leur temporalité. Ce domaine est bien traité par le système UPPAAL⁸ développé par Kim Larsen avec des équipes des universités d'Uppsala (Suède) et Ålborg (Danemark). UPPAAL a par exemple permis de trouver un bug temporel de réseau qui bloquait parfois les coûteux systèmes audio/vidéo de Bang & Olufsen ; cette société avait cherché en vain d'où venait ce bug pendant dix ans. D'autres systèmes de vérification temporelle sont fondés sur un modèle d'automates temporisés étudiés par exemple par Thomas Henzinger, auteur du système HyTech.

Enfin, signalons la logique temporelle TLA+ (*temporal logic of actions*) de Leslie Lamport (prix Turing 2013), qui permet de traiter des algorithmes distribués arbitraires, non nécessairement d'états finis. N'étant plus limité aux systèmes finis, il est traité par un assistant de preuve mathématique.

Tous les systèmes précités sont maintenant utilisés dans de nombreux projets industriels.

Les systèmes de démonstration automatique

Le développement de systèmes de démonstration automatique ou assistée de théorèmes mathématiques ou d'assertions de programme a été très tôt un objectif de chercheurs en intelligence artificielle, avant de devenir une discipline spécifique de l'informatique classique. Parmi les systèmes historiquement importants, citons Nqthm de Robert Boyer et J. Strother Moore à Austin (États-Unis), qui a donné suite au système ACL2 utilisé par exemple par AMD pour vérifier ses propres opérateurs arithmétiques après le bug du Pentium Pro. Citons aussi PVS de John Rushby et Natarajan Shankar au Stanford Research Institute (SRI), utilisé entre autres pour de nombreux projets spatiaux. Je ne les détaillerai pas ici.

De grands progrès ont été ensuite faits en s'adressant à deux catégories de problèmes décidables plus restreints mais de grande importance théorique et pratique souvent solubles efficacement en pratique : le calcul booléen et la satisfaction modulo théories. Examinons-les maintenant.

LE PROBLÈME SAT (SATISFACTION BOOLÉENNE)

Le *calcul booléen* est le tout bête calcul logique où les variables peuvent prendre les valeurs *vrai* et *faux*, ou de façon équivalente 0 et 1, et où les opérateurs sont la conjonction *et*, la disjonction *ou*, et la négation *non*. On y cherche souvent à savoir si une formule donnée est *satisfiable*, c'est-à-dire pouvant être rendue vraie en

choisissant bien les valeurs des variables, ou encore *tautologique*, c'est-à-dire vraie pour toutes les valeurs des variables. Par exemple, la formule « A ou $((\text{non } B) \text{ et } C)$ » est vraie si A est vrai, ou si B est faux et C vrai, ou encore si ces deux conditions sont toutes deux remplies ; elle devient fausse si A est faux et si B est vrai ou C faux, ce qui veut aussi dire qu'elle n'est pas tautologique.

Le calcul booléen a toujours joué un rôle essentiel dans l'industrie des circuits électroniques, binaires par nature. Mais ce qui le rend fondamental pour la vérification automatique, c'est qu'il permet aussi de coder beaucoup d'autres problèmes. Par exemple, un problème de Sudoku peut se coder très simplement comme une grosse formule booléenne comme expliqué dans l'[annexe 12.2](#). Cette formule a 729 variables et une bonne dizaine de milliers d'opérateurs. La résoudre en essayant toutes les possibilités des valeurs de variables serait absurde, car il faudrait en explorer $2^{729} \cong 10^{210}$, nombre bien plus grand que tout ce qu'on peut compter dans l'univers. Il faut donc trouver d'autres méthodes algorithmiques. Celles disponibles actuellement permettent de trouver la solution en un éclair. Bien sûr, en informatique, on utilise SAT pour des problèmes bien plus sérieux, y compris industriels.

L'histoire du calcul booléen est fascinante. Elle a bien sûr commencé avec George Boole au XIX^e siècle, qui a défini ce calcul et fait ainsi passer la logique de la philosophie à l'algèbre et aux mathématiques. La difficulté d'y faire des calculs de tête et même à la main est apparue très rapidement ; elle persiste toujours. Des logiciens se sont essayés à trouver des algorithmes efficaces, dont Lewis Carroll, l'auteur d'*Alice au pays des merveilles*. Le calcul booléen est devenu vraiment fondamental dans les années 1950, en particulier pour optimiser les opérateurs des ordinateurs en les faisant aller plus vite et utiliser moins de lampes et de transistors. Le lecteur intéressé pourra consulter la bibliographie sur les tables de Maurice Karnaugh, nées en 1953, qui ont longtemps servi à optimiser une formule booléenne à partir de sa table de vérité.

La première révolution date de 1971, avec l'introduction par Stephen Cook (prix Turing 1982) de la notion d'algorithme NP-complet qui sera détaillée dans l'[annexe 12.1](#). Cook a montré que le problème SAT de la satisfaction booléenne était équivalent à un grand nombre d'autres problèmes de grande importance pratique, c'est-à-dire qu'on saurait résoudre rapidement tous ces problèmes si on savait le faire pour SAT. Pour l'instant, tous les algorithmes connus pour le problème SAT sont

exponentiels dans le pire cas, donc aussi mauvais que l'énumération de toutes les valeurs possibles des variables. Cook a conjecturé qu'il n'existe pas d'algorithme polynomial résolvant tous les cas, c'est-à-dire tout le temps efficace en pratique pour résoudre SAT et tous les problèmes équivalents. Cette conjecture est classée comme un des sept problèmes mathématiques du millénaire posés par l'institut Clay, avec 1 million de dollars à la clef pour qui trouvera sa solution.

À première vue, ce résultat et cette conjecture sont déprimants, car il est impossible en pratique de résoudre un problème en temps exponentiel dans le nombre de variables dès que ce nombre est un peu élevé. Mais le cas le pire n'est pas forcément le cas utile. Les chercheurs ont donc cherché des *heuristiques*, c'est-à-dire des algorithmes capables de résoudre rapidement des cas utiles en pratique. Cette quête a commencé avec l'algorithme de Davis et Putnam en 1960, amélioré par l'algorithme DPLL (Davis-Putnam-Logemann-Loveland) en 1962. Au lieu d'énumérer toutes les valeurs de toutes les variables, l'algorithme leur donne une valeur une par une, jusqu'à soit montrer la formule satisfiable soit trouver une contradiction, c'est-à-dire la rendre fausse. Dans ce cas, on revient en arrière en changeant la valeur de la dernière variable affectée qui n'a pas encore pris les deux valeurs *vrai* et *faux*. DPLL marche bien sur les petits problèmes, mais s'effondre rapidement quand la formule grandit.

Une révolution bien plus importante en pratique a eu lieu en 1987 avec l'introduction par Randal Bryant (Carnegie Mellon University) des BDD (*binary decision diagrams*). Ce sont des graphes représentant une formule de façon compacte et rendant polynomiales les opérations de base *et*, *ou*, *non*. Les BDD ont conduit à des améliorations considérables de l'ensemble des outils de conception de circuits, ogres de calcul booléen, ainsi que des algorithmes de vérification formelle. Ken McMillan, déjà nommé plus haut, a utilisé les BDD pour implémenter les premiers logiciels de *model-checking* fondés sur la logique temporelle en représentant efficacement les grands ensembles d'états des systèmes étudiés. J'ai moi-même utilisé intensivement les BDD pour l'optimisation et la vérification formelle de gros programmes industriels écrits dans mon langage Esterel pour des systèmes embarqués et les circuits électroniques.

Mais l'histoire était loin d'être finie⁹. En 1991, une équipe italienne introduisit le nouvel algorithme GRASP dédié au problème SAT, qui n'était pas la plus grande

force des BDD. Partant du vieux DPLL, l'idée était de ne pas revenir en arrière brutalement sur les choix des valeurs, mais d'étudier pourquoi une contradiction s'était produite et de synthétiser une nouvelle formule logique à ajouter au problème initial pour détecter bien plus tôt une autre occurrence de la même contradiction. Il s'agissait donc réellement d'apprendre et de mémoriser de nouvelles informations au cours du calcul. GRASP a vraiment changé la donne. Puis est arrivée une autre amélioration de grande ampleur avec l'algorithme zChaff de l'équipe de Sharad Malik à Princeton. L'algorithme de GRASP s'y est trouvé considérablement amélioré par l'introduction de structures de données « paresseuses » extrêmement économes en calcul, qui permettent de ne quasiment rien faire la plupart du temps en se contentant de bouger une patte de temps en temps comme savent si bien le faire les chats. Depuis cette époque, les chercheurs se livrent à une course effrénée à l'efficacité avec des innovations impossibles à décrire ici, en se livrant à une compétition annuelle où les meilleurs systèmes se comparent entre eux. Comme les candidats ont chacun des performances assez variables selon les problèmes posés, la bonne méthode pour gagner est devenue d'en faire tourner plusieurs en parallèle et de choisir pour chaque formule celui qui termine le premier !

Les logiciels SAT modernes arrivent à résoudre des problèmes industriels ayant *des millions* de variables, ce qui était totalement impensable il y a seulement quinze ans. Le mot « industriel » est ici important. On distingue en effet plusieurs classes de problèmes : les problèmes *intrinsèquement durs*, pour lesquels aucun système connu ne passe à l'échelle ; les problèmes *aléatoires*, peu utiles en pratique mais qui intéressent beaucoup pour d'autres raisons les chercheurs en physique statistique ; et les problèmes *industriels*, directement issus de cas pratiques. Pour une raison parfaitement inconnue, ces derniers s'avèrent souvent solubles. On est un peu dans le même cas que dans l'apprentissage par réseau de neurones profonds : les algorithmes marchent bien, on ne sait pas pourquoi, et ce serait vraiment intéressant de le comprendre pour aller plus loin !

LA SATISFACTION MODULO THÉORIES

Puisqu'un ordinateur ne fait jamais que manipuler des bits en nombre fini, le calcul booléen peut en théorie exprimer et vérifier tout ce qui s'y passe. Mais, comme le dit si bien Yogi Berra : « En théorie, la théorie et la pratique c'est la même chose, mais en pratique, ce n'est pas vrai. » D'une part, le nombre de bits contenus dans un ordinateur est bien trop grand pour les possibilités des solveurs booléens, d'autre part le booléen est très mal adapté aux problèmes arithmétiques, pour lesquels les mathématiques nous ont fourni des formalismes bien plus puissants.

Les logiciels de *satisfaction modulo théories*¹⁰ (De Moura & Bjørner, 2011), en abrégé SMT, permettent de combiner le calcul booléen pour le raisonnement logique et des solveurs spécifiques dédiés à différentes théories décidables utiles en pratique : l'arithmétique de Presburger (celle des nombres entiers limitée à l'addition, la soustraction et la comparaison), la théorie des nombres réels (plus simple, ce qui n'est vraiment pas intuitif), l'arithmétique d'accès aux tableaux, la manipulation de pointeurs en mémoire, etc. Les algorithmes de chacun de ces solveurs peuvent être habilement combinés pour permettre de résoudre des problèmes faisant intervenir en même temps toutes ces théories. Mais on n'a rien sans rien : les calculs peuvent être chers et leur coût relativement imprévisible. Les solveurs SMT généraux comme CVC4, Z3 (Microsoft Research), Alt-Ergo (université Paris-Sud/Inria) sont cependant très puissants en pratique et souvent utilisés soit de façon autonome, soit en connexion avec des systèmes de vérification associés aux langages de programmation, soit encore avec des systèmes de résolution de contraintes pour calculer diverses sortes d'optimisations. Il existe aussi bon nombre de solveurs dédiés à des problèmes pointus spécifiques.

PROUVER AUTOMATIQUEMENT LA TERMINAISON

Prouver la terminaison d'un programme peut être très difficile, voire impossible, comme l'a montré Alan Turing dès 1936. Mais c'est une question très importante en pratique. Prenons l'exemple des fameux « écrans bleus » associés aux crashes de Windows, qui ont si longtemps fait perdre des données et du travail à bien des gens. De façon naturelle, les utilisateurs les attribuaient à des bugs de Microsoft. Mais c'était loin d'être toujours le cas. Très souvent, le problème venait des *drivers*

(« pilotes de périphériques » en français), qui soit crashaient eux-mêmes, soit ne terminaient jamais, provoquant indirectement un crash du système par terminaison d'une temporisation de sécurité contrôlant le *driver* : il vaut mieux arrêter tout plutôt que laisser attendre indéfiniment l'utilisateur.

Heureusement, le problème n'est pas insoluble en pratique : Byron Cook, alors chez Microsoft, a développé deux systèmes successifs de preuve de terminaison spécifiquement adaptés aux drivers (Terminator et T2). Ces systèmes ont été incorporés au kit de développement de drivers diffusé par Microsoft aux fabricants d'appareils connectables, ceux qui injectent justement des drivers dans Windows.

Les assistants de preuve

Les *assistants interactifs de preuve* permettent de vérifier des problèmes arbitraires non traitables actuellement par les méthodes automatiques. Comme pour les solveurs SMT vus précédemment, l'idée de base est de réduire les problèmes de vérification à des problèmes de logique formelle. Mais cette fois on utilise des logiques beaucoup plus puissantes qui ne sont plus décidables algorithmiquement. Les assistants imposent l'écriture explicite des preuves de correction, non pas directement mais à l'aide de tactiques de haut niveau, en se faisant de plus aider par une bonne dose d'automatisation pour faciliter la fastidieuse vérification de nombreuses parties cléricales qu'il n'est pas question de répéter à la main lors de toute étape de mise au point ou de modification d'une preuve. Un point important est que vérifier la correction d'une preuve est bien plus simple que de la construire, et complètement automatique. C'est cela qui donne une grande force aux assistants : la correction finale du programme vérifié ne dépend que de celle du module de vérification de preuves, en général simple et bien compris mathématiquement, en aucune façon du système d'aide à la construction et à la gestion des preuves, lui bien plus complexe. Cela répond en partie à la question classique : « Mais comment vérifie-t-on le vérifieur ? »

J'ai parlé au [chapitre 8](#) de l'utilisation des assistants pour de grandes preuves mathématiques. En informatique, il ne s'agit plus de prouver des théorèmes mathématiques mais de vérifier la correction de circuits et programmes. Techniquement, comme les énoncés à vérifier sont exprimés en logique formelle, ce sont bel et bien des théorèmes au sens des mathématiciens. Mais ces théorèmes

d'usage pratique n'ont généralement aucun intérêt pour les mathématiques. Cela n'empêche pas que leurs preuves nous en apprennent beaucoup sur les systèmes informatiques que nous voulons développer et vérifier, et que les assistants modernes permettent même dans certains cas l'extraction automatique de programmes garantis corrects à partir des preuves.

LES LOGIQUES UTILISÉES

Comment s'y prend-on pour traiter un programme avec un assistant de preuve ? Il faut d'abord formuler le programme et les conditions à vérifier dans une logique appropriée, ce qui n'est pas toujours simple. Les assistants sont fondés sur des logiques assez variées, toutes issues des recherches en logique mathématique conduites depuis les années 1930. Les plus simples sont les logiques dites de *premier ordre*, c'est-à-dire dans lesquelles les quantificateurs « pour tout » ($\forall$) et « il existe » ($\exists$) ne s'appliquent qu'aux objets de base (entiers, chaînes de caractères, etc.). Les plus riches sont les *logiques d'ordre supérieur*, dans lesquelles les quantificateurs s'appliquent aussi aux fonctions, aux fonctionnelles (fonctions prenant des fonctions en argument et retournant soit des objets soit d'autres fonctions), et ainsi de suite à tous les étages fonctionnels. Les logiques d'ordre supérieur sont beaucoup plus expressives, mais aussi nettement plus difficiles à traiter en machine.

Enfin, le *calcul des constructions inductives* de Thierry Coquand, Gérard Huet et Christine Paulin est une logique d'ordre supérieur remarquablement expressive tout en restant proche de la programmation classique (en OCaml). Il est à la base du système Coq dont nous avons parlé pour les mathématiques. Il a la particularité d'être fondé sur la *correspondance de Curry-Howard*, du nom des logiciens Haskell Curry et William Alvin Howard. Cette correspondance est souvent expliquée comme « calculer, c'est prouver », ce qu'on peut justifier simplement sur un exemple comme le tri : écrire un programme de tri, c'est prouver de façon constructive qu'il est possible de trier les objets qu'on reçoit en entrée ; chaque algorithme de tri fournit une preuve particulière de ce théorème. L'exploitation systématique de la correspondance de Curry-Howard permet à Coq d'intégrer un vrai langage de programmation directement dans le système de preuve, unifiant ainsi réellement programmation et

vérification. Il ne s'agit pas d'un langage général, car le calcul des constructions interdit l'écriture de programmes qui bouclent indéfiniment. Cela veut dire aussi que l'écriture du programme doit implicitement contenir aussi celle de sa preuve de terminaison (pour les connaisseurs, on peut écrire des récursions sophistiquées, mais pas des boucles while générales).

Coq permet aussi une extraction de programmes corrects par construction et raisonnablement efficaces à partir des preuves. Cette puissance supplémentaire de Coq est utilisée systématiquement par le compilateur C CompCert formellement vérifié décrit plus loin. Elle lui donne maintenant une place de leader dans le domaine, avec par exemple de gros projets comme DeepSpec¹¹ aux États-Unis, qui vise à systématiser la preuve formelle dans la construction de nouvelles infrastructures logicielles. À suivre...

LA PREUVE EN PRATIQUE

Une fois la logique et l'assistant choisis, il faut conduire les preuves à l'aide de l'assistant, qui va gérer l'ensemble des formules et des preuves en vérifiant à chaque étape la correction de chaque déduction. Les déductions ne sont heureusement pas faites une à une à la main. Elles l'étaient dans les tout premiers systèmes, par exemple dans le système LCF (*logic for computable functions*) de Robin Milner (prix Turing 1991) ; j'ai utilisé ce système pour prouver le « tri bulle » Bubble Sort dès 1972, sous la supervision de Milner lors de mon premier contact avec la vérification de programmes modernes. À cette époque, les déductions élémentaires devaient bel et bien être spécifiées une par une à la main. Maintenant, elles sont gérées par des langages de description de *tactiques de preuve*, qui permettent de gérer les preuves à bien plus haut niveau en profitant de beaucoup d'automatisations partielles¹². Les progrès actuels sont spectaculaires¹³, comme les exemples ci-dessous vont le montrer. Cependant, à la différence des méthodes automatiques précitées, l'usage des assistants de preuve demande encore une grande compétence technique et beaucoup de patience. Les approches automatiques et assistées sont toutes deux utiles, et beaucoup de travaux s'attachent maintenant à les faire mieux coopérer, que ce soit en automatisant

davantage d'étapes pour les assistants ou en fournissant des preuves lisibles par les assistants pour les vérificateurs SAT/SMT ou les vérificateurs par interprétation abstraite.

LA VÉRIFICATION DU MÉTRO METEOR

Le plus ancien logiciel de grande ampleur vérifié formellement à l'aide d'un assistant de preuve a certainement été le logiciel de conduite du métro Meteor de Paris (ligne 14) par la RATP en utilisant la méthode B de Jean-Raymond Abrial ([Abrial, 1996](#)). L'histoire est longue et intéressante¹⁴.

Au début des années 1980, la RATP avait décidé de rendre automatique la conduite du RER A de Paris à l'aide d'un système informatique embarqué. Mais sa direction, inquiète de la sécurité du système, commanda à Abrial un audit technique pour s'assurer que la réalisation correspondait bien aux spécifications. Abrial a rapidement répondu qu'il ne pouvait pas répondre à la question, car on ne lui avait jamais montré de spécifications au cours de toutes ses discussions avec les partenaires industriels du projet. À la fureur des industriels qui avaient réalisé le système, mais à juste raison, la RATP décida de ne mettre le système en service que partiellement : au lieu de transmettre les ordres au matériel, le logiciel les écrivait sur un écran et le conducteur devait les exécuter en vérifiant qu'ils n'étaient pas aberrants.

Un peu plus tard, la RATP décida de coopérer avec Abrial pour réaliser le logiciel embarqué de Meteor, qui devait lui être totalement automatique. L'idée était de construire et vérifier formellement le logiciel en utilisant la méthode B d'Abrial, dont la logique est une forme restreinte de théorie des ensembles du premier ordre. La RATP décida aussi de financer l'Atelier B, nouvel outil logiciel dédié à B et permettant d'assurer un raffinement graduel et prouvé des spécifications au code, avec un mélange de preuves automatiques et de preuves assistées. Cet atelier fut utilisé pour le développement et la vérification de l'ensemble des logiciels. Les chiffres sont éloquentes : 86 000 lignes de programmes ADA produites, 27 800 preuves réalisées automatiquement par l'Atelier B, dont 92 % automatiquement, et seulement 7 hommes-mois pour les preuves interactives. En confiance, la RATP prit la décision de supprimer tous les tests unitaires et d'intégration habituels. Le succès fut total, le logiciel n'ayant jamais montré le moindre bug depuis sa mise en service en 1998. La

conclusion a été nette : pour ce projet, la preuve a coûté beaucoup moins cher que le test. Plus tard, la ligne 6 du métro parisien a été automatisée de la même façon, et bien d'autres conduites de métros dans le monde ont été développées avec B.

La méthode B a plus tard été étendue en un nouveau formalisme plus évolué nommé Event-B ([Abrial, 2010](#)), traité par l'atelier Rodin¹⁵. Mais Event-B n'a pas connu le même succès industriel, peut-être parce que de nombreux autres systèmes de programmation et de vérification formelles sont apparus entre-temps.

LA VÉRIFICATION DE CIRCUITS

J'ai mentionné à plusieurs reprises le coût exorbitant des bugs de circuits. Souvenons-nous par exemple du fameux bug de division du Pentium Pro en 1991, décrit au [chapitre 9](#). En utilisant son système HOL-light dérivé du système HOL (Higher Order Logic) de Michael Gordon (Université de Cambridge), John Harrison (Intel) a réalisé au milieu des années 2000 la preuve complète de la correction des unités arithmétiques fixes et flottantes du Pentium, étendue maintenant à tous les microprocesseurs produits par Intel. Ici, une preuve assistée s'impose puisque ces unités calculent aussi la multiplication et la division, hors de portée des méthodes booléennes car y impliquant des calculs exponentiels dans l'état actuel des connaissances.

Un résultat similaire a été obtenu en 2005 par David Russinoff pour l'unité arithmétique du microprocesseur K5 d'Advanced Micro Devices (AMD), le concurrent principal du Pentium à l'époque. Cette fois, la preuve a été conduite avec ACL2, que j'avais classé parmi les systèmes automatiques mais qui demande ici un guidage très précis assez semblable à celui des assistants de preuve.

John Harrison a ensuite prouvé la correction des algorithmes matériels pour calculer les sinus ou autres fonctions mathématiques, et le groupe ACL2 continue de son côté à travailler sur ces mêmes sujets. Ces parties critiques de tout microprocesseur sont désormais prouvées plutôt que testées, car un test complet serait impossible de par la taille des nombres utilisés. De plus, la preuve apporte directement une garantie absolue de correction, car les opérations arithmétiques à effectuer sont définies mathématiquement sans la moindre ambiguïté.

LA VÉRIFICATION DE SYSTÈMES D'EXPLOITATION

Les systèmes d'exploitation, aussi nommés OS pour *operating systems*, gèrent les activités des utilisateurs et les ressources des machines. Ce sont donc des composants essentiels de toutes les infrastructures informatiques. Même si les applications des téléphones et les navigateurs des ordinateurs cachent aux utilisateurs les OS sous-jacents en leur fournissant des interfaces graphiques évoluées, ces applications et navigateurs font sans cesse appel à leurs ressources. Les OS comme Windows, Mac OS X/iOS, Linux et Android, ceux qui ont survécu à un long combat darwinien qui en a éliminé bien d'autres, sont des programmes très complexes et de grande taille. Il est hors de question de les vérifier formellement pour l'instant.

Mais ces grands systèmes sont mal armés pour faire fonctionner les processeurs des objets embarqués, qui disposent de relativement peu de puissance de calcul. Il existe pour eux une autre race de systèmes qu'on appelle des *micronoyaux*, ou *microkernels* en anglais. C'est pour l'instant sur eux que se sont portés les efforts de preuve.

Le premier micronoyau formellement vérifié a été seL4, une réimplémentation du micronoyau L4, originellement développé en Allemagne avec pour objectif une très grande efficacité. La construction et la vérification de seL4 ont été effectuées au laboratoire NICTA à Sydney (Australie) sous la direction de Gernot Heiser et Gerwin Klein¹⁶. Elle a constitué un travail de grande ampleur. Les défis ont été nombreux, le premier étant de faire travailler ensemble des chercheurs en OS, plutôt spécialistes des structures de données et algorithmes efficaces, et des chercheurs en vérification formelle, plutôt enclins à l'abstraction et aux approches théoriques.

La façon de travailler a été très nouvelle mais assez lourde. Des spécifications formelles de haut niveau ont d'abord été écrites en Haskell, un langage fonctionnel typiquement pas fait pour ça mais dont la puissance a été très utile pour coupler spécifications et implémentation. Il est hors de question ici de décrire l'ensemble des spécifications, mais j'en citerai quelques-unes visiblement essentielles : chaque appel au micronoyau se termine, exécute bien la fonction requise, laisse la mémoire en état cohérent, et ne modifie pas les zones de mémoire qu'il n'est pas censé accéder. Violier n'importe laquelle de ses propriétés conduirait à des bugs graves et difficiles à détecter une fois le système en marche. À ces contraintes de bon fonctionnement ont

été ajoutées de fortes contraintes de sécurité, bien sûr jamais complètes puisqu'il est impossible par construction de recenser tous les canaux cachés pouvant conduire à des trous de sécurité. La correction de l'ensemble a été prouvée en utilisant le vérifieur Isabelle de Lawrence Paulson. Puis un autre niveau de spécification, cette fois exécutable, a été écrit en Haskell. Toujours avec Isabelle, ces spécifications exécutables ont été prouvées conformes aux spécifications précédentes. Enfin, elles ont été petit à petit transformées en code C efficace, un peu comme pour B d'Abrial mais avec des contraintes d'optimisation bien plus fortes. Dans ce processus incrémental, chaque étape de transformation a été prouvée correcte, toujours avec Isabelle. Le système final seL4 n'est que 10 % plus gros et plus lent que l'implémentation très optimisée de L4, ce qui est absolument remarquable. Il a connu une grande diffusion.

Plus récemment, un autre micronoyau nommé ProvenCore a été programmé et vérifié formellement à partir d'une spécification standard nommée Minix. Cette fois nettement plus léger et intégré, ce travail a été réalisé en France dans le cadre industriel de la société Prove & Run créée par Dominique Bolignano, qui fournit une gamme d'outils prouvés pour la sécurité informatique. ProvenCore est devenu un des produits de l'offre commerciale de Prove & Run. Bolignano s'était déjà illustré dans sa société précédente Trusted Logic par l'utilisation des méthodes formelles en sécurité, par exemple avec des preuves en Coq pour des protocoles de la carte bleue ; Trusted Logic a d'ailleurs été rachetée par Gemalto, leader des cartes à puce. La preuve de ProvenCore a été réalisée avec un assistant maison fondé sur calcul des prédicats. Je ne détaillerai pas davantage ce cas, car il a été exposé par Bolignano lui-même dans un séminaire associé à mon cours¹⁷.

LA VÉRIFICATION DE COMPILATEURS

Au chapitre précédent, j'ai présenté le générateur aléatoire dirigé Csmith de programmes C qui a mis en évidence des centaines de bugs dans les compilateurs C principaux, qu'ils soient académiques ou industriels. J'ai aussi mentionné qu'un seul compilateur avait survécu, le compilateur CompCert de Xavier Leroy (aussi auteur de Caml et OCaml). Ce compilateur a effectivement été complètement prouvé correct en

utilisant Coq, ce qui a constitué un exploit retentissant qui en amènera certainement bien d'autres dans la suite. Il faut noter que Xavier Leroy connaissait déjà bien Coq, qu'il avait utilisé chez Prove & Run pour prouver la correction d'un protocole de sécurité au plus haut niveau du standard CMM (*capability maturity model*).

LE COMPILATEUR C FORMELLEMENT VÉRIFIÉ COMPCERT

Le compilateur C formellement prouvé CompCert a été développé à l'Inria par Xavier Leroy et son équipe. Prouver un compilateur traitant le langage C complet n'est pas réellement possible car, même s'il est normalisé, C n'est pas un langage bien défini. Dans son standard, on voit en effet qu'il y a un grand nombre de comportements non spécifiés, voire indéfinis, qui ne peuvent donc pas être formalisés. Il a fallu définir un sous-ensemble Clight de C dont la sémantique puisse être parfaitement définie en Coq et qui soit suffisant pour la plupart des applications, en particulier embarquées dans les avions ou autres systèmes de haute sécurité. Il a fallu ensuite formaliser les langages machine des processeurs ciblés (x86 d'Intel, ARM, etc.), puis écrire et prouver CompCert.

CompCert utilise 10 passes successives avec 9 langages intermédiaires pour simplifier sa réalisation et sa preuve. Chaque passe réalise une fonction spécifique du compilateur que je ne décrirai pas ici. Deux techniques différentes de vérification ont été utilisées selon la nature de chaque passe : soit écrire simultanément les algorithmes et leur preuve en Coq puis extraire automatiquement le code final OCaml de la passe, soit faire appel à un programme externe et vérifier en Coq la validité des résultats qu'il produit. La deuxième possibilité est en particulier utilisée pour les passes d'optimisation, sous-traitées à des programmes spécialisés qu'il serait stupide de réécrire en Coq. On vérifie alors que l'optimisation est juste, mais pas qu'elle est optimale, ce qui n'est pas un problème pour la correction. Le lecteur intéressé pourra en savoir plus en regardant la vidéo ¹⁸ (en anglais) de l'exposé donné par Xavier Leroy à la Royal Society anglaise quand il a reçu le Grand Prix Milner 2016, dont le jury international est composé à égalité de membres de la Royal Society, de l'académie allemande Leopoldina et de l'Académie des sciences française.

L'expérience CompCert a été marquante dans le domaine, et d'autres chercheurs cherchent maintenant à prouver des compilateurs pour des langages différents. Par exemple, Lionel Rieg a travaillé avec moi sur la preuve des algorithmes principaux de la compilation de mon langage Esterel, dédié à la construction formelle d'applications temps réel et à celle de circuits électroniques (voir l'[annexe 12.6](#)). Le travail est très avancé mais pas encore définitif.

Les preuves de sécurité

Le domaine de la sécurité informatique est particulièrement difficile du point de vue formel. Je rappelle d'abord que la plupart des systèmes de chiffrement reposent sur des conjectures mathématiques, comme celle de l'impossibilité pratique de factoriser un nombre calculé comme le produit de deux très grands nombres premiers, qui fonde RSA. Ce sont donc surtout les protocoles d'établissement de communication de type SSL/TLS qui sont visés, ou encore les systèmes de vote par logiciel.

En termes de sécurité, tout n'est pas formalisable puisque les canaux cachés sont par nature inconnus. Il faut donc se restreindre à la protection contre des attaques particulières, par exemple celles consistant à intercepter ou forger des messages lors de la négociation par un protocole de sécurité de type SSL/TLS, ou encore à des contraintes qu'on souhaite respectées pour un système de vote. Les formalismes sont variés et peuvent être assez différents de ceux que nous avons vus jusqu'à présent¹⁹. Si la logique classique peut suffire pour certains systèmes, il faut faire appel à d'autres méthodes très techniques pour les protocoles réseaux les plus généraux. Je serai donc très bref sur le sujet.

Je voudrais cependant préciser que les équipes françaises sont très en pointe sur le sujet : l'équipe de Bruno Blanchet à l'Inria développe le logiciel Prosecco, un standard du domaine qui a permis d'invalidier plusieurs propositions de protocoles et d'analyser d'autres en profondeur. Travaillant en collaboration étroite, l'équipe de Karthikeyan Barghavan à l'Inria et celle de Cédric Fournet à Microsoft Research ont conçu des attaques fortes sur TLS comme Freak, mais ont aussi défini une

implémentation partielle prouvée miTLS de TLS et commencent le chemin vers une implémentation prouvée réellement efficace et utilisable en production. De leur côté, au CNRS et à l'Inria Nancy, Véronique Cortier, Patrick Gaudry et leurs équipes ont développé le logiciel de vote en ligne sécurisé et formellement vérifié Belenios, qui n'est cependant pas encore adapté à des élections publiques générales où les contraintes sont très fortes. Bien d'autres projets sont en cours dans le monde, mais on ne sait pas encore qui sera le vainqueur de la course, des hackers ou des chercheurs. Je ne peux bien sûr pas citer tous les travaux actuels, que je ne connais d'ailleurs pas tous.

Quand utiliser la vérification formelle ?

La réponse est simple : *le plus tôt et le plus longtemps possible*, c'est-à-dire dès la mise en forme exécutable des spécifications et tout au long de la programmation et de la vérification. Vouloir réaliser vite des vérifications formelles en mode « pompier » quand un projet échoue alors que cela n'a pas été prévu dès le départ ne marche que rarement.

Malheureusement, il est difficile de convaincre les chefs de projets qu'il vaut mieux travailler de cette façon préventive et incrémentale au lieu de se contenter de la méthode classique qui consiste à tout programmer d'abord et à tout tester après. Comme les coûts additionnels arrivent plus tôt, ils résistent à cette idée, souvent sans réaliser que le bénéfice sera bien plus considérable en fin de compte. Le début d'un projet classique est en général consacré au développement des composants matériels ou logiciels et à leurs tests unitaires. Il se passe en général bien, et c'est le plus souvent dans les phases d'intégration que les ennuis commencent, avec la découverte tardive de problèmes de spécification en particulier. Mais, à cette époque, toute correction est beaucoup plus lourde et il n'est jamais clair qu'elle ne va pas introduire d'autres problèmes ailleurs. Concevoir et vérifier bien plus tôt avec des méthodes formelles est très efficace pour éviter cet écueil et travailler plus vite et pour moins cher.

Mais on sait bien que même si bien des fumeurs sont persuadés qu'il serait bon qu'ils arrêtent, la plupart ne le font que lorsqu'ils craignent d'être vraiment malades, et recommencent quand ces doutes s'estompent. Rien n'est plus difficile à vendre que la prévention...

Trois expériences personnelles intéressantes

Personnellement, j'ai toujours été passionné par les relations entre les mathématiques et la pratique du développement de systèmes informatisés. J'ai plutôt été contributeur dans le domaine des langages de programmation et utilisateur de systèmes de vérification formelle, en étant conscient du fait qu'il faut absolument comprendre ces deux aspects pour qu'ils se marient bien. Je raconte ici trois expériences personnelles successives dans ce domaine, qui m'ont demandé pas mal de travail mais m'ont apporté tout autant de satisfaction, ainsi qu'à mes utilisateurs je pense. Pour en savoir plus, visionnez les trois cours du Collège de France que j'ai donnés à Sophia-Antipolis en janvier 2014, sur le thème « Esterel, de la recherche à l'industrie²⁰ ».

ESTEREL ET L'AVIONIQUE

Au début des années 1990, je développais l'outillage de mon langage Esterel avec comme objectif principal la programmation des logiciels temps réel. Jean Vuillemin, mon premier maître en circuits, travaillait alors au Paris Research Laboratory (PRL) de Digital Equipment. Il m'invita à devenir consultant chez PRL, afin d'utiliser Esterel dans un autre domaine, le développement des nouveaux systèmes à base de circuits reprogrammables FPGA que son équipe construisait à cette époque. J'ai rapidement réalisé à ma grande surprise que la bonne façon de comprendre la nature

profonde d'Esterel était par le calcul booléen, bien mieux adapté que les techniques d'automates finis que j'utilisais depuis 1983. Cela m'a permis de traduire directement les programmes Esterel en circuits, et aussi d'améliorer considérablement sa compilation logicielle vers C. Cette nouvelle technologie fut rapidement introduite dans le premier compilateur Esterel diffusé alors par la société Ilog, plus tard rachetée par IBM.

L'environnement Esterel d'Ilog était alors utilisé par Dassault Aviation pour le développement des spécifications et des logiciels de son avion de chasse Rafale, dans le groupe dirigé par Emmanuel Ledinot, ingénieur de très haut vol. Un des problèmes de Dassault était le besoin de vérification formelle pour des sous-systèmes critiques. L'un d'entre eux était le BTM, boîtier de test et de maintenance, chargé des tests au sol et en vol de l'avion et de ses équipements. Ce boîtier coordonnait une vingtaine de tâches de test, certaines au sol et d'autres en vol. Plusieurs contraintes de sécurité étaient essentielles : il fallait par exemple montrer qu'il était impossible de déclencher en vol certains tests prévus pour le sol, comme l'essai de tous les ailerons en excursion maximale – ce qui aurait bien sûr provoqué un accident majeur. Le système initial était classiquement réalisé avec des tâches asynchrones gérées par un système d'exploitation classique, ce qui rendait les propriétés désirées improuvables à cause du gigantesque indéterminisme du comportement. L'équipe de Ledinot décida de reprogrammer toute la spécification du BTM en Esterel, langage parallèle mais déterministe conduisant à des vérifications bien plus simples.

Il se trouve que deux autres remarquables chercheurs de PRL, Olivier Coudert et Jean-Christophe Madre, travaillaient sur un nouveau logiciel nommé TiGeR, qui implémentait très efficacement les BDD de Bryant (voir ci-dessus) en y ajoutant de nouveaux algorithmes pour le *model-checking*. Nous pûmes connecter Esterel et TiGeR et prouver formellement toutes les propriétés souhaitées du BTM. Je pense qu'il s'agit d'une des premières vérifications formelles d'un système complexe en utilisation réelle. Bien d'autres preuves de ce type furent réalisées chez Dassault Aviation. L'histoire est racontée plus en détail dans l'article (Berry *et al.*, 2000).

ESTEREL V7 ET LA SYNTHÈSE DE CIRCUITS

En 2007-2008, j'étais directeur scientifique de la société Esterel Technologies. Mon centre d'intérêt principal était la synthèse efficace de circuits électroniques à partir de programmes de haut niveau, celle initiée chez Digital Equipment. Au Strategic CAD Laboratory de Portland (Oregon), avec mon second maître à penser les circuits Michael Kishinevsky d'Intel, nous avons défini une nouvelle version d'Esterel nommée Esterel v7, bien mieux adaptée aux circuits car facilitant les calculs sur les données. Avec mon équipe d'Esterel Technologies, j'avais développé ses outils de compilation, d'optimisation et de vérification formelle par liaison avec le vérifieur automatique SMT suédois Prover SL.

Nous travaillions alors avec une équipe de Texas Instruments sur la réalisation d'un système de gestion des données d'images pour la télévision haute définition (TVHD) sur téléphone, sujet nouveau à l'époque et fort difficile. La vérification formelle fut utilisée dès le début. En comparaison avec les méthodes de design classiques, le design fut plus rapide et bien plus sûr, et le circuit plus efficace en temps et espace physique. Selon la procédure en place à l'époque, le design obtenu fut envoyé en Inde pour être testé. L'équipe de test s'est alors plainte car elle ne trouvait pas de bugs, alors que ses supérieurs la jugeaient précisément selon le nombre de bugs trouvés ! Il n'a pas été facile de prouver au management que notre méthode était la bonne.

CORRIGER LES BUGS, MAIS SANS RECOMPILER

Dans le design de circuits, les contraintes sont particulièrement dures. D'abord, toutes les parties du circuit doivent être prêtes le même jour, celui de la mise en construction de la douzaine de masques photographiques qui serviront à créer le circuit final. Mais, comme ils contiennent des centaines de millions de traits, la création de ces masques est une opération longue et coûteuse : l'unité est le million de dollars. Quand le circuit est « fondu », on peut enfin le faire marcher à pleine vitesse. On trouve alors presque toujours des bugs résiduels bloquants, qui viennent généralement de spécifications erronées. Malheureusement, quand on veut corriger un bug de circuit, il n'est pas toujours possible de corriger le code source des

programmes et de le recompiler comme on le fait pour le logiciel : il faudrait pour cela refaire quasiment tous les masques, avec un coût additionnel important.

La seule solution pratique est de « patcher » les masques à la main, en rajoutant des portes logiques et en ajoutant ou redirigeant des fils. On laisse toujours quelques pour cent de place libre dans le circuit initial pour pouvoir le faire sans tout chambouler. Comme Esterel v7 est un langage de très haut niveau par rapport à celui des masques, le client nous a demandé de procéder en trois étapes : corriger le code source Esterel, ce qui est généralement simple, et tester ou vérifier formellement que son nouveau comportement est bien le bon ; trouver comment patcher à la main le circuit de façon à obtenir le même nouveau comportement ; enfin, montrer formellement et automatiquement que la correction du source Esterel v7 et le patch manuel du circuit induisent exactement le même comportement. Nous avons pu le faire. C'est là que nous avons vraiment apprécié de disposer un langage très bien défini et de bons outils de vérification formelle.

CHAPITRE 11

Où va nous mener l'informatisation du monde ? Une vision personnelle

J'espère que les analyses présentées dans les chapitres précédents ont permis au lecteur de se construire un meilleur schéma mental sur ce qu'est réellement l'informatisation du monde, ses apports, ses difficultés intrinsèques, et d'où vient sa puissance. J'ai surtout voulu illustrer le fait fondamental que la pensée et l'action informatiques sont les mêmes partout même si leurs effets sont très variés : elles font appel à la même science et aux mêmes outils quel que soit le champ d'application.

Il me faut maintenant aborder une question qu'on me pose toujours, comme à tous mes collègues : où va nous mener l'informatisation galopante du monde ? Cette question est évidemment légitime, mais je serais bien présomptueux de prétendre y répondre de façon vraiment fiable. Je vais cependant m'y essayer, mais après quelques bonnes pages de précautions pour que le lecteur comprenne clairement que je vais formuler surtout des hypothèses et pas des certitudes, ainsi que quelques pistes concrètes d'action. Contrairement aux chapitres précédents, je prendrai donc soin de ne jamais mélanger le « je » et le « nous ». Je parlerai aussi beaucoup de la situation de mon pays, la France, celui que je connais le mieux et dans lequel je m'investis le plus à l'heure actuelle. Mais la situation ne me paraît pas très différente dans les autres pays européens (aux États-Unis et en Asie, c'est évidemment très différent).

Préliminaires

« LA PRÉVISION EST UN ART DIFFICILE,
SURTOUT EN CE QUI CONCERNE L'AVENIR »

(Auteur initial pas clair, mais compétent. Il semble que ce soit Niels Bohr...)

Probablement par manque de compétence ou de largeur de spectre, je n'ai pas d'avis péremptoire sur beaucoup des questions posées par l'informatisation généralisée du monde, surtout en termes d'impacts sur la société. Le sujet me paraît d'autant plus délicat que le passé nous fournit peu de références à cause des changements très rapides de schéma mental et d'échelle induits par l'informatique moderne. J'ai passé pas mal de temps à lire des articles d'anticipation sur des sujets divers dont les affirmations se sont trouvées un peu plus tard totalement démenties par les faits, et je ne suis pas sûr de pouvoir faire bien mieux. Thomas Watson, P-DG d'IBM, affirmait en 1943 : « Je pense qu'il y a un marché mondial pour quelque chose comme cinq ordinateurs. » Ken Olsen, P-DG de Digital Equipment (alors second fabricant d'ordinateurs), disait en 1977 : « Un particulier n'a aucune raison d'avoir un ordinateur chez lui. » Bill Gates, P-DG de Microsoft, affirmait en 2004 : « Le spam sera sous contrôle d'ici deux ans. » Steve Ballmer, lui aussi P-DG de Microsoft, clamait en 2007 que « l'iPhone n'avait pas d'avenir industriel ». J'ai aussi longtemps entendu de la bouche de dirigeants et scientifiques français que « l'informatique est une mode qui va passer » ou que « l'informatique n'est pas une science ». Ce n'étaient pas des incompetents, loin de là, mais ils avaient

visiblement un problème de schéma mental, lié comme souvent à une trop forte certitude que l'avenir allait être une projection du passé et qu'ils en seraient les maîtres.

À l'inverse, et en très peu de temps, Internet et le Web ont complètement modifié le paysage mondial d'une façon que peu d'acteurs avaient anticipée. S'appuyant sur leur infrastructure, le moteur de recherche a radicalement changé notre façon de nous informer et d'organiser notre travail et notre vie de tous les jours, mais personne n'avait prévu sa naissance (personnellement, je ne me suis même pas aperçu de son importance alors que j'étais vraiment sur place...). C'est la même chose pour le téléphone portable devenu smartphone, qui a permis un accès presque ubiquitaire à Internet et a largement favorisé l'essor des réseaux sociaux. Ces réseaux bannissant la géographie modifient en profondeur les relations humaines, d'une façon qui n'avait été probablement anticipée que par leurs créateurs – et encore. Un phénomène similaire quoique moins connu du public s'est produit en parallèle : l'informatisation généralisée des outils scientifiques et techniques, et celle d'objets de plus en plus nombreux comme les voitures, dont le public se sert souvent sans savoir qu'ils sont bourrés d'ordinateurs et de logiciels, voire en ignorant ce qu'est un ordinateur ou un logiciel. Enfin, l'irruption à grande échelle des techniques d'apprentissage étend considérablement le champ d'application de l'informatique d'une façon impensable il y a seulement cinq ans.

Prudence donc : quelques évolutions sont certaines, d'autres assez prévisibles, mais les plus importantes seront probablement dues à des ruptures par essence imprévisibles qui provoqueront à leur tour des modifications profondes des schémas mentaux. Puisqu'on ne peut pas prévoir ces ruptures, on ne peut pas non plus les planifier, contrairement à ce que voudraient faire croire certains bureaucrates ou pseudo-visionnaires qui les datent même d'avance. Elles viennent le plus souvent de recherches fondamentales, pas forcément d'ailleurs sur le sujet auquel elles vont s'appliquer. Elles résultent souvent d'accidents de parcours. Pour moi, la bonne définition de la recherche fondamentale est précisément celle qui travaille sur des questions fondamentales, qu'elles soient théoriques ou pratiques.

La seule chose qu'on peut faire est de faciliter la naissance impromptue des ruptures en faisant confiance à la créativité et à la capacité de vagabondage mental et technique des chercheurs fondamentaux et appliqués, davantage qu'à leur capacité à

remplir des formulaires et des dossiers qui est hélas de plus en plus sollicitée. Cela doit se faire tout en facilitant les liens entre eux, les créateurs d'entreprises et les ingénieurs, tous aussi importants dans les rouages de la création ; certains personnages peuvent d'ailleurs être tout cela à la fois.

Cette opinion est banale et souvent répétée, mais il reste encore beaucoup de pain sur la planche pour qu'elle prenne vie. Et il y a deux idées à démonter avec force. La première est qu'il suffit de travailler beaucoup et de mettre beaucoup d'argent pour résoudre à peu près n'importe quel problème fondamental. Les cas où ça marche sont rares. La deuxième, pas moins naïve, est qu'il faut réduire les budgets des chercheurs fondamentaux pour qu'ils soient encore plus productifs. Qu'on vienne donc me dire ça en face !

ATTENTION (VRAIMENT) AUX EFFETS PERVERS

Quelles que soient les évolutions, il faudra être attentifs aux effets pervers qui ne manqueront pas de se produire et qui sont généralement violents dans le domaine. L'informatique crée de nouvelles dominances économiques et étatiques dont les abus de pouvoir peuvent devenir incontrôlables, des inégalités nouvelles entre ceux qui la maîtrisent ou l'anticipent et les autres, et de nouvelles dépendances chez les enfants, souvent trop enclins à foncer sur les objets informatisés à la mode. Les problèmes de sécurité informatique dont j'ai beaucoup parlé et que je ne reprendrai pas ici peuvent devenir massifs, et ce n'est pas en gardant l'habitude tenace de mettre sa tête dans un trou comme l'autruche qu'on verra arriver les coups. À l'inverse, il faut tout autant être méfiant par rapport aux affirmations exagérées qui sont toujours légion quand les choses progressent vite dans un domaine mal assimilé, soit qu'elles promettent la lune soit qu'elles promettent l'enfer.

Le problème central est que ces questions doivent être traitées avec compétence par des institutions et organisations rassemblant des sensibilités et des intérêts très variés : science, industrie, politique, droit, etc. Mais, je ne le rappellerai jamais assez, la compétence ne peut être acquise que par la compréhension dans ce domaine étranger aux anciens raisonnements. Cette compréhension est pour l'instant bien trop faible. La formation au schéma mental de l'informatique (pas déguisée pour le coup

en numérique) de tous ces acteurs et de la population générale est donc indispensable si on ne veut pas être continuellement ballotté par les flots. Elle est possible, et je souhaite continuer à y contribuer.

DES MÉDIAS PEU INFORMATIFS

À part la radio, qui est encore de haute qualité en France et qui donne le temps de s'exprimer à ceux qui y sont invités, il semble y avoir dans les grands médias en matière d'informatique une course à une information très courte et donnant vite le frisson : domination des GAFA et Uber sur le monde, destruction du travail, attaques informatiques incompréhensibles, espionnage généralisé avec disparition de la notion d'intimité personnelle, irruption d'une intelligence artificielle qui va rendre les machines plus intelligentes que les hommes et peut-être les chasser de la terre, remplacement magique par les MOOC de l'enseignement donné par des professeurs humains, perte de concentration chez les enfants accrochés à leurs téléphones et à leurs réseaux sociaux, etc. Tout le monde semble répéter la même chose, en rajoutant un peu dans le dramatique pour essayer de sortir du lot.

Ces questions sont tout à fait sérieuses, mais trop se focaliser sur elles a aussi des inconvénients car il en existe bien d'autres, dont celles que j'ai choisi de présenter dans les chapitres précédents. De plus, plutôt que d'essayer de les analyser sérieusement, on cherche de plus en plus à leur apporter une réponse rapide et péremptoire pour passer rapidement à un autre sujet, densité de l'actualité oblige. Et on ne parle généralement que du présent ou de l'avenir proche, en oubliant que l'avenir un peu plus lointain ne leur ressemblera certainement pas, ce qui conduit à avoir en permanence un mètre de retard. Tout cela se fait dans une incompréhension renouvelée entre les générations, à coups d'échanges de « arrête un peu avec ton téléphone » et de « vous pouvez pas comprendre », quelque peu dédaigneux mais pas forcément faux des deux côtés. Oui, l'informatique est puissante, mais ce n'est pas une raison pour remplacer la réflexion par la pensée magique ou la pensée instantanée, jetable dès qu'elle a été formulée.

Certes, des analyses variées et remarquablement pertinentes paraissent sur différents sites sur l'autre grand média qu'est Internet, mais elles ne sont pas

forcément faciles à trouver. Et je suis loin d'être sûr qu'elles ont autant de succès que les fausses informations et les rumeurs qui y foisonnent avec une facilité « électronique », car répandues à la vitesse de la lumière par les gourmands.

LA SORTIE DU TROU NOIR (GRIS, ALLEZ...)

Selon moi, cette situation que j'espère provisoire est liée en bonne partie à deux facteurs aggravants, à l'œuvre jusque très récemment. Le premier facteur est le fait que trop peu d'informaticiens ont souhaité expliquer ce qu'ils faisaient de façon compréhensible par le reste du monde. La vulgarisation a été faible par rapport à ce qui se passe depuis longtemps en physique, en astronomie ou en biologie, et souvent trop concentrée sur les aspects techniques. Beaucoup d'informaticiens travaillant dans leur coin discutaient avec leurs pairs et étaient aussi des experts du « tu peux pas comprendre » vis-à-vis du reste du monde. L'image de l'informaticien *geek* mangeant des pizzas froides et codant vingt-cinq heures par jour est tenace, et les informaticiens n'y sont pas pour rien, même si les geeks sont en fait largement minoritaires dans la profession. Le second facteur, que j'ai déjà mentionné dans l'introduction et qui est nettement plus important en pratique, est que beaucoup de personnes occupant d'importantes positions politiques, décisionnaires, scientifiques ou enseignantes, pourtant très sensées, ont longtemps choisi de faire purement et simplement l'impasse sur l'informatique et ses effets. Il a vraiment fallu attendre ces dernières années pour que le sujet arrive à la surface, et c'est loin d'être vrai partout.

Heureusement, la situation est en train de changer rapidement. D'abord, les informaticiens participent de plus en plus à la vie collective, à l'expression publique vers des publics variés et à l'enseignement. On le constate en particulier par l'action vigoureuse de la jeune SIF (Société informatique de France¹), ou celle d'Interstices², superbe site organisé par l'Inria en collaboration avec le CNRS, la SIF, la SMAI (Société de mathématiques appliquées et industrielles³) et l'EPI (association Enseignement public & informatique⁴), association de professeurs très impliquée dans l'enseignement de l'informatique. Les informaticiens dialoguent aussi de mieux en mieux avec les autres scientifiques, les industriels et les pouvoirs publics, l'ouverture se faisant dans les deux sens. Les start-up courageuses se créent à une vitesse peu

connue auparavant. Quelques informaticiens ou autres personnes vraiment compétentes écrivent des chroniques dans les grands médias, et l'informatique entre enfin (à petits pas) dans l'enseignement, où les associations susmentionnées et bien d'autres sont très actives.

Du côté institutionnel, l'intérêt pour le sujet augmente à vue d'œil : un secrétariat d'État au Numérique et un Haut Conseil du numérique ont été créés et sont très actifs. Les barrières psychologiques et institutionnelles se lèvent petit à petit, les non-informaticiens écoutant davantage les informaticiens et réciproquement. Les parlementaires commencent à se saisir des bonnes questions et à se renseigner auprès des bonnes personnes, et les juristes s'interrogent sur les conduites à tenir. Les radios font d'excellentes émissions scientifiques analysant en profondeur des sujets reliés à l'informatique et au monde numérique. Mais tous les obstacles ne sont pas levés, en particulier dans l'enseignement, domaine essentiel s'il en est. J'y reviendrai longuement plus loin.

Les évolutions du matériel

Je commence par le sujet du matériel, car même si l'informatique s'occupe principalement d'immatériel, ses progrès n'ont été rendus possibles que par les constants progrès du matériel qui supporte cet immatériel. On peut même dire que rien n'aurait été possible sans la très rapide évolution des circuits électroniques, mémoires de masse et réseaux. Souvenons-nous par exemple que chacun de nos téléphones ne peut fonctionner que parce qu'il a probablement plus de capacité de mémoire qu'il y en avait dans la France entière dans les années 1980, ou que les principes de l'apprentissage profond sont connus depuis vingt-cinq ans, mais que ce n'est que tout récemment que la puissance de calcul est devenue suffisante pour les faire passer à la pratique.

LA FIN PROGRAMMÉE DE LA LOI DE MOORE, UNE OPPORTUNITÉ ?

Au [chapitre 3](#), j'ai mis en avant la loi de Moore, celle qui a permis l'évolution exponentielle des circuits du petit et lent vers le très grand très rapide. Mais cette loi sur les circuits n'est pas le seul facteur d'évolution. La même trajectoire exponentielle a été prise par le stockage de données, les réseaux de communication, les dispositifs d'interaction homme-machine, les capteurs et actionneurs, et de façon générale l'interaction avec le monde physique. En bref, l'ensemble du système a suivi. Cette évolution a conduit à une augmentation fantastique du nombre de circuits et des

systèmes produits, qui va clairement aller encore en s'amplifiant. Mais la loi de Moore elle-même va-t-elle s'accélérer, se ralentir, ou encore se transformer profondément ? Je penche personnellement pour la troisième hypothèse.

LES ÉVOLUTIONS POSSIBLES DES CIRCUITS

Soyons précis : la loi de Moore annonçait le doublement tous les deux ans du nombre de transistors par unité de surface pour les circuits intégrés à base de transistors au silicium. Elle s'est appliquée pendant plus de quarante ans aux mémoires, aux processeurs et aux circuits spécialisés. Cela s'est fait à principes technologiques quasi constants par divisions répétées de la taille des composants, en gardant une seule couche active de transistors et de multiples couches de connexions. Mais on approche maintenant des limites physiques et surtout économiques de cette loi, à cause du mur de la chaleur à l'intérieur des circuits, du coût des usines, des coûts liés à la CAO et de ceux des bugs résiduels. Le mur de la chaleur est vraiment une limitation sérieuse : il y a longtemps qu'on ne peut plus augmenter la vitesse des circuits, car leur fréquence d'horloge maximale dépend linéairement du voltage alors que la chaleur dissipée dépend de son carré. On cherche au contraire à ralentir les circuits autant que faire se peut dès qu'ils n'ont rien d'urgent à faire, pour les laisser se refroidir et aussi éviter d'user la pile. Dans les gros circuits, on commence à atteindre l'ère du « silicium noir » ou *dark silicon*, celle où il faut arrêter un cœur dès qu'il chauffe trop et envoyer ses calculs sur un autre plus frais, une bonne partie du circuit étant à chaque instant en mode refroidissement. Ça ne simplifie ni l'architecture ni le fonctionnement dynamique.

Tout cela fait donc que la loi de Moore va finir par s'arrêter ou au moins sérieusement ralentir. Comme mes collègues, je l'ai entendu et démenti régulièrement au cours des vingt dernières années, mais cette fois ça semble vrai. Je l'ai déjà dit, il ne s'agissait pas d'une loi physique mais d'une décision concertée de l'industrie d'avancer selon la vitesse qu'elle impose. L'industrie va bientôt baisser collectivement les bras, sauf pour quelques grands fabricants qui continuent la course. Mais ce ralentissement peut ouvrir de nouvelles portes : ce n'est que maintenant que

d'autres procédés physiques peuvent devenir compétitifs, et la recherche peut donc être relancée en grand.

Le premier changement, qui commence à apparaître en production, est un usage bien supérieur de la troisième dimension en conservant la même technologie de base, par superposition de plusieurs couches de circuits actifs sur la même surface. Ces circuits 3D vont probablement se développer beaucoup, mais en restant cependant difficiles à refroidir.

Une piste très différente est fournie par les *memristors*, développés en particulier par Hewlett Packard (HP). Ce sont des « résistances à mémoire », dont la résistance peut être ajustée de façon continue par un courant électrique. Les memristors ont été utilisés pour des prototypes de mémoires 2D et 3D bien plus denses que les RAM classiques (voir le [glossaire](#)), atteignant chez HP la densité théorique extraordinaire d'un pétabit (un million de milliards de bits) par centimètre cube. De plus, cette mémoire est permanente, à l'inverse des RAM qui perdent leur information dès qu'on les débranche. Ses temps d'accès sont actuellement nettement plus lents que pour la RAM mais plus rapides que la mémoire flash. Des transistors minuscules peuvent aussi être construits avec les memristors, donc de la logique de calcul.

Une nouvelle recherche concerne les circuits fonctionnant de façon « neuronale » bio-inspirée, avec des applications potentielles à l'apprentissage automatique, à la simulation des processus de calcul dans le cerveau ou à bien d'autres applications qu'apportera inmanquablement l'imagination des chercheurs et ingénieurs. On en construit déjà à base de silicium classique, et les memristors sont aussi un bon candidat pour ce genre de fonctions. Mais je ne connais pas suffisamment ce domaine pour en parler avec compétence.

Enfin, beaucoup d'autres technologies de base sont étudiées, comme les circuits optoélectroniques, très utilisés pour les communications ultrarapides dans les superordinateurs et bientôt peut-être à l'intérieur même des circuits classiques, les mémoires et les circuits fondés sur la spintronique, qui avait déjà révolutionné le stockage magnétique (voir le [glossaire](#)), ou encore les circuits à base de graphène ou de ces autres matériaux fascinants que nous concoctent les physiciens.

Mais, dans tous ces cas, il y a loin du prototype à la production de masse à coût faible, et il ne faut pas encore tirer des plans sur la comète à partir de simples expérimentations et prototypes. Le silicium n'est pas mort, loin de là.

LA SAUVEGARDE DES DONNÉES

Le problème du stockage fiable et permanent des grandes données sur les mémoires de masse devient de plus en plus important avec la domination de la vidéo sur le Web et des nouvelles cueillettes de données réalisées par les instruments physiques et astronomiques, les séquenceurs de génome et les grands réseaux de capteurs. À part pour la vidéo, les données produites par les hommes prennent finalement peu de place par rapport à cette avalanche. Or les technologies de stockage permanent n'ont pas beaucoup changé depuis le xx^e siècle, même si elles se sont bien améliorées. Les disques tournent toujours et cassent sans prévenir. Les bandes magnétiques sont toujours dans leurs cassettes et restent très utilisées pour les sauvegardes. La mémoire flash, dont la capacité a considérablement crû et le prix a chuté, remplace de plus en plus les disques dans les ordinateurs ; elle est la seule présente dans les tablettes et téléphones. Sa fiabilité est excellente, mais pas parfaite car elle s'use à l'écriture. Les CD et DVD longtemps utilisés pour sauvegarder les données personnelles sont en voie de disparition car minuscules vis-à-vis des besoins actuels, et le BluRay est arrivé bien trop tard pour devenir intéressant (sauf pour les films « pressés », que je trouve à un prix bien bas puisque presque plus personne n'en veut). Tous ces mouvements vont se poursuivre.

Mais plusieurs problèmes subsistent : quelles données sauvegarder, sachant qu'on ne pourra pas tout garder ? Comment bien mieux les compresser quand c'est possible ? Comment assurer leur sauvegarde à long terme, alors qu'aucun des supports actuels n'est pérenne ? La première question n'aura pas de réponse générale car tout dépendra des contextes précis. Mais certains domaines comme la biologie sont en train de devenir problématiques à cause des performances des séquenceurs de génomes modernes qui croissent plus vite que celles des supports ; les biologistes sont perplexes sur ce qu'il faudrait garder et pour quels objectifs. Et les volumes de données en circulation sur Internet sont tellement énormes qu'il est impossible de tout stocker, contrairement à ce que pensent beaucoup de gens. La question d'améliorer la compression est difficile mais occupe beaucoup les scientifiques. On peut y attendre de belles améliorations, voir le *compressed sensing* dont j'ai parlé à propos des bases de données génomiques au [chapitre 8](#).

Pour la question de la pérennité des supports, la réponse semble possible, mais certainement pas facile à mettre en œuvre. Pour les œuvres d'art existant en exemplaire unique, la situation paraît plus simple, puisqu'on peut recopier les données à l'infini quasiment sans risque de modification grâce aux codes correcteurs d'erreurs. Mais ce n'est pas si simple. Pour ma part, je sauve mes quelques téraoctets de photos sur des serveurs disques répartis dans plusieurs maisons, mais ce n'est pas commode et beaucoup ne le font pas – et trop souvent perdent tout. On me propose de les stocker « dans le nuage » (*cloud storage*), c'est-à-dire sur des ordinateurs situés je ne sais où dans le monde et eux-mêmes bien sauvegardés. Les services sont nombreux et globalement fiables, mais la vitesse des réseaux est encore très insuffisante pour que ce soit commode pour moi, qui habite à la campagne. De plus, quelle assurance réelle aurai-je quant à la pérennité et la sécurité de mes données, et suis-je sûr qu'elles pourront être transmises à ma famille s'il m'arrive quelque chose, comme on dit ?

Le problème est évidemment bien pire pour les entreprises et les institutions comme les bibliothèques, l'INA (Institut national de l'audiovisuel), et les instituts analogues. Il n'est en fait pas nouveau. Quand France Inter a voulu faire un CD souvenir de sa célébrissime émission du week-end *L'Oreille en coin*, qui a ravi ma génération, cette radio s'est aperçue qu'elle n'avait quasiment plus de documents sonores : les bandes magnétiques ayant été envoyées à la benne car elles prenaient trop de place. Heureusement, il restait les enregistrements faits chez eux par les auditeurs !

Fabriquer des mémoires de grande taille, de grande durée de vie et de coût raisonnable reste difficile, mais c'est un sujet actif de recherche. Les meilleurs candidats semblent être des disques optiques spéciaux de grande qualité, et de façon plus surprenante et potentiellement plus productive, les brins d'ADN, molécule prodigieuse qui porte déjà la mémoire de notre espèce et que nous ne remercierons jamais assez. Microsoft s'y intéresse beaucoup, entre autres.

À QUOI RESSEMBLERONT LES MACHINES INFORMATIQUES DU FUTUR ?

Ce ne sont pas les circuits et les mémoires qui font nos machines, mais le savant assemblage de circuits, mémoires et interfaces sur l'extérieur qui détermine leurs architectures. Jusqu'à maintenant, celles-ci sont restées assez homogènes au sein d'un petit nombre de groupes : les superordinateurs, les serveurs d'entreprises, les ordinateurs personnels, les tablettes et téléphones, et les ECU (*electronic control units*) – c'est comme cela qu'on nomme les ordinateurs industriels embarqués par exemple dans les avions ou dans les voitures. Je pense que le paysage sera d'une bien plus grande variété à l'avenir.

Du côté des superordinateurs, la règle est déjà le parallélisme massif avec des centaines de milliers ou des millions de processeurs, ce nombre augmentant constamment. La cible visée est l'exaflop, soit $10^{18} = 1$ milliard de milliards d'opérations par seconde. C'est beaucoup, mais ça devrait être atteint assez vite. Les applications seront nombreuses en calcul scientifique et en traitement de données massives. Mais l'objectif de l'exaflop est aussi politique que technique : le premier qui l'atteindra assoira son prestige, et la Chine met toute sa puissance pour dépasser les États-Unis.

Les fermes de serveurs sont naturellement bien plus évolutives car elles sont faites à partir de composants complètement modulaires, allant de la « lame » intégrant processeur plus disques ou flash au conteneur complet en assemblant des milliers. Les grandes fermes possèdent à l'heure actuelle des dizaines ou centaines de milliers de serveurs. On les place là où il est facile de les alimenter en électricité et de les refroidir. Les algorithmes distribués et la réplication systématique des données leur permettent de résister aux pannes locales de serveurs individuels, qui sont nombreuses à cette échelle. Les fermes servent maintenant aussi bien à stocker des données qu'à fournir du stockage et du calcul par le *cloud computing*, d'une façon quelquefois très simple : chez certains fournisseurs, prendre 1 000 processeurs pendant 10 secondes ou un processeur pendant 10 000 secondes coûte le même prix.

En ce qui concerne les machines à usage personnel que sont les ordinateurs, tablettes et téléphones, la situation tend à se brouiller et l'avenir n'est pas clair pour moi. Un point essentiel est l'apparence physique, ce qu'on appelle plus élégamment le *form factor* en anglais. Il y a quelques années, les ordinateurs personnels étaient lourds mais puissants, les téléphones légers mais limités et les tablettes n'en différaient que par leur grand écran et l'absence de carte SIM (*subscriber identity*

module – saviez-vous ça ?). Maintenant, les ordinateurs s'allègent au point de se comparer aux tablettes, les tablettes se dotent au contraire de clavier, et les téléphones existent en plusieurs tailles. En dehors du fait qu'ils savent aussi téléphoner, leur puissance augmente continûment et bien plus vite que celle des ordinateurs, et leur taille mémoire pourrait devenir comparable. L'ergonomie est quasiment identique entre téléphones et tablettes, et celle des ordinateurs tend à la rejoindre. De plus, intéressant paradoxe, les téléphones deviennent aussi chers, voire plus chers, que les ordinateurs personnels, alors même qu'il s'en vend beaucoup plus. La généralisation des connexions sans fil aux haut-parleurs et aux télévisions pourrait supprimer l'avantage qu'ont encore les ordinateurs de par leurs prises de branchement externes, qui vont peut-être aussi disparaître pour les mêmes raisons. Peut-être la situation évoluera-t-elle pour le grand public vers des machines presque identiques à l'usage, ne différant que par le fait qu'on peut les mettre dans la poche ou pas, avec une grande variété de *form factors*. Peut-être aussi que le téléviseur disparaîtra, simplement remplacé par un écran et des haut-parleurs joignables depuis n'importe quel autre appareil informatique, ce qui bouleverserait une fois de plus l'industrie de l'électronique classique ; elle l'a déjà été il y a quelques années, même si personne ne s'en rend compte : un téléviseur moderne n'est jamais qu'un ordinateur très puissant connecté à un bel écran.

Les plus grands changements sont probablement à attendre dans la myriade d'ordinateurs couplés de façon invisible à des capteurs et actionneurs puis embarqués dans toutes sortes d'objets auparavant restés simplement matériels, des ampoules électriques aux automobiles en passant par les prothèses médicales. J'en ai déjà beaucoup parlé, donc je n'insisterai pas davantage. Mais j'ai un joli exemple de bricolage individuel éclairant (au sens propre), que m'a rapporté un de mes collègues de l'Inria. Un de ses amis fait du vélo sur la Côte d'Azur, et se méfie des conducteurs locaux (il a bien raison). Il a fixé à son maillot deux accéléromètres à trois sous utilisés dans les voitures et deux rubans lumineux LED à six sous, un à droite et un à gauche. Il les a connectés à un minuscule ordinateur à cent sous noyé dans une rondelle plastique aussi collée au maillot, tournant sous Linux et doté d'un programme adéquat. Dès qu'il se penche, le bon ruban clignote automatiquement. C'est joli, mais le clou de l'histoire est que, quand il rentre chez lui, il met le tout à la machine : tous les composants sont lavables, y compris l'ordinateur !

L'ÉCOLOGIE DE L'INFORMATIQUE

Voici un sujet dont on parle trop peu, sur lequel j'ai beaucoup d'inquiétude mais pas beaucoup de propositions. Pour commencer, les matières utilisées par les ordinateurs, téléphones et périphériques sont loin d'être anodines : métaux divers, terres rares extraites difficilement de mines dangereuses et devenues des enjeux stratégiques, verres spéciaux, etc. Ensuite, les objets informatisés et en particulier les téléphones sont à l'heure actuelle très difficilement recyclables. Produits par milliards et de durée de vie particulièrement courte, ils encombrant les décharges. Certes, changer de modèle n'est pas nécessairement stupide car les anciens modèles sont loin de rendre les services que savent rendre les nouveaux, surtout maintenant que téléphoner est passé de mode chez les jeunes par rapport aux selfies-MMS, aux courriers électroniques et à l'accès à Internet. Mais les fabricants ne se privent pas d'en rajouter en organisant l'obsolescence programmée. Certains fabricants proposent des téléphones modulaires et recyclables, mais sans succès pour l'instant. Nous ne pourrions certainement pas continuer cette trajectoire très longtemps.

Un autre problème majeur est la consommation électrique des ordinateurs et objets informatisés, déjà très importante et conduisant à une gabegie de petites piles et batteries pour tout ce qui est portable ou autonome. Comme je l'ai dit plus haut, les fabricants de circuits mettent une énergie considérable à faire qu'ils en consomment moins, mais l'augmentation explosive de leur nombre fait que cette réduction ne suffit pas. Pour les superordinateurs et fermes de calcul, les choses sont tout aussi sérieuses : on dit souvent qu'un superordinateur est ce qu'on peut acheter pour 20 mégawatts, refroidissement inclus...

Je ne sais pas ce qu'il faut faire pour réduire l'emprise écologique de l'informatique, sinon mettre les gens au courant de son existence. Ce n'est pas vraiment fait pour l'instant, contrairement à d'autres dangers dont je parlerai plus loin.

QUID DE L'ORDINATEUR QUANTIQUE ?

Un sujet très à la mode est l'ordinateur quantique, qui pourrait, selon ce que j'entends dans les médias, « être *infinitement* plus rapide que les ordinateurs actuels ». S'ensuit une explication confuse de ce qu'est la superposition quantique, sujet que

même les meilleurs physiciens n'ont vraiment réussi à expliquer sans équations au commun des mortels, pour de bonnes raisons d'ailleurs : ça ne fait vraiment pas partie de la vie commune.

Quelle est la situation réelle ? Je n'en suis pas un spécialiste, donc je ne répondrai à la question qu'avec précaution. D'après mes renseignements, il y a deux problèmes assez distincts, l'un physique et l'autre algorithmique. Sur le plan physique, l'ordinateur quantique n'existe pas encore, même s'il y a beaucoup de recherches sur le sujet. Beaucoup de chercheurs essaient différentes technologies pour fabriquer des *qubits*, ces fameux « bits quantiques » qui permettent la superposition quantique de valeurs. Mais la superposition qubit à qubit n'est pas très intéressante, car c'est la coordination de l'ensemble des superpositions dans le système global considéré qui est pertinente.

Considérons la question du cassage du chiffrement RSA décrit dans l'[annexe 12.1](#). Il s'agit de factoriser un nombre $n = pq$ où p et q sont deux grands nombres premiers, en ne connaissant ni p ni q . Avec un ordinateur classique, on ne connaît aucun algorithme capable de le faire en temps raisonnable pour de très grands nombres (2 048 bits par exemple). En 1994, le mathématicien Peter Shor, professeur au MIT, a réalisé une avancée fondamentale en montrant qu'un ordinateur quantique de taille suffisante pourrait effectivement réaliser cette factorisation très rapidement. Ce résultat, qui s'applique aussi à presque tous les chiffrements actuellement utilisés, est scientifiquement remarquable et potentiellement grave en pratique car casser tous les chiffrements d'Internet provoquerait un chaos de première grandeur. On pourrait certes remplacer en théorie les chiffrements actuels par d'autres résistants au quantique, mais ce ne serait pas une mince affaire.

Le premier problème est qu'on ne dispose pas encore d'ordinateur quantique, sinon tout petits. Les bits quantiques sont très délicats et tendent à perdre très vite leur capacité globale de superposition, surtout dès qu'il y en a beaucoup. C'est ce qu'on appelle la décohérence. On ne sait pas quand il sera possible d'avoir des machines universelles de taille raisonnable. Certains annoncent cinq ans, d'autres quarante, mais je ne crois pas du tout au sens de ce genre de chiffres, qu'il faut bien fournir pour être financé. En pratique, pour la factorisation, le record actuel implémenté est la factorisation de 143 sur 4 qubits. Elle peut être étendue facilement à 56 153 qui n'est en fait pas plus compliqué car ce nombre appartient à une classe facilement

factorisable avec un ordinateur standard ; 175 étant plus compliqué pour des raisons mathématiques inexplicables ici, il n'est pas encore factorisé en pratique.

De plus, contrairement à ce qu'on entend souvent, un ordinateur quantique général ne sera pas *infiniment* plus rapide qu'un ordinateur classique, mais seulement exponentiellement plus rapide dans les bons cas (ce qui est déjà excellent⁵). Un autre problème majeur est que l'algorithmique quantique est difficile, et que, d'après mes informations, on ne connaît que peu d'algorithmes réellement utilisables. Enfin, un ordinateur quantique est un objet physique très complexe, qui ne va pas tenir dans votre poche et qui ne pourra peut-être servir que de fantastique coprocesseur à un ordinateur classique. On est donc loin du rêve absolu parfois décrit.

En résumé, ce sujet est fondamental et magnifique, et ses applications pourraient être considérables. Mais la bête n'existe pas encore. Laissons donc les physiciens et algorithmiciens travailler sereinement, sortons des fantasmes arbitraires, et soyons prêts à entendre de vraies avancées intellectuelles et techniques majeures quand elles se produiront – ou pas, ce ne sont pas nos seules envies qui le décideront.

En revanche, un autre domaine avance vite : la communication quantique sécurisée, rendue possible par les découvertes pionnières d'Alain Aspect sur les corrélations quantiques à distance. Mais je n'ai pas la compétence pour en parler.

L'évolution des logiciels classiques

Je serai bien plus bref sur l'évolution des logiciels classiques, hors apprentissage, qui ne me paraissent pas devoir changer profondément de nature mais aller vers une bien plus grande diversification liée à une multiplication des applications et des besoins.

RIGUEUR OU BOUILLONNEMENT ?

Je pense d'abord qu'un gros travail doit être continué avec beaucoup de rigueur dans deux domaines essentiels qui sont ceux que je connais le mieux. D'abord, l'amélioration de la qualité de l'infrastructure globale, concernant au moins les systèmes d'exploitation, les langages de programmation, leurs compilateurs, leurs environnements de développement, les bibliothèques d'algorithmes fondamentaux séquentiels, parallèles et distribués, les réseaux et les interfaces homme-machine. Comme pour un immeuble, si les fondations ne sont pas très stables, rien de solide ne peut être bâti dessus. Il reste énormément à faire dans ce domaine. Ensuite, il faut consolider toutes les applications liées aux domaines où la sûreté et la sécurité informatiques sont essentielles : avionique, ferroviaire, automobile, médical, etc. J'ai suffisamment donné d'exemples où la qualité n'est pas au niveau requis pour ne pas avoir besoin d'insister davantage. Dans tous ces cas, le sérieux et le professionnalisme sont absolument indispensables, et les progrès de la science informatique sont appelés à jouer un rôle croissant.

D'autres domaines sont encore à une époque bouillonnante de grande créativité et diversité. La nouvelle façon d'organiser des applications individuelles sur les infrastructures standard des téléphones, pour l'instant limitées à iOS d'Apple et Android de Google, facilite à la fois la construction d'applications remarquablement imaginatives et la multiplication d'applications presque identiques dont les meilleures ou au moins les survivantes finiront par émerger d'une forme quelconque de sélection naturelle. L'avenir y est très ouvert, mais la mortalité considérable. Il faut quand même noter que le modèle d'interaction des applications qu'on utilise une à une sur les téléphones est à l'exact opposé de celui des PC ou du Web, dans lesquels la vision simultanée de plusieurs activités parallèles par multifenêtrage était la règle. La raison initiale était la petite taille de l'écran, mais la mono-application en a profité pour s'ancrer dans nos esprits. Il sera intéressant de voir si ces deux approches peuvent être réconciliées. Encore un changement de schéma mental à gérer...

Enfin, il serait souhaitable que les systèmes de gestion classique soient rendus plus souples et agréables à utiliser. Je suis toujours consterné de voir des grands projets comme le système Louvois de paye des armées dépenser des fortunes avant d'être jeté à la poubelle, ou de constater à quel point des systèmes de réservation d'avions et de trains peuvent être désagréables à utiliser, avec leurs écrans pleins de rectangles pleins de couleurs mais vides de sens qui font qu'on ne peut jamais voir tout ce qu'on veut en même temps sur l'écran ; heureusement, d'autres sont d'utilisation vraiment agréable et parfaitement compatibles, ce que j'apprécie vu le nombre de trains que je prends.

UNE ÉTRANGE RÉGRESSION DANS LES LANGAGES DE PROGRAMMATION ?

Un point qui me soucie est que les progrès réalisés dans les cinquante dernières années sur le sujet de la sûreté des langages de programmation sont quelquefois joyeusement jetés par-dessus bord. Des langages devenus extrêmement populaires comme JavaScript et Python n'ont pratiquement plus de typage ni d'analyse des programmes avant l'exécution, et y ajouter des choux et carottes n'est pas un problème. Ces approches laxistes ont déjà existé, par exemple avec LISP dans les

années 1960, mais c'était à l'époque héroïque de l'informatique. Elles avaient été presque partout remplacées par des approches typées garantissant mieux la qualité des résultats.

Certes, Python et JavaScript ont gagné le monde par leur facilité d'apprentissage et la simplicité avec laquelle ils se combinent aux nouvelles infrastructures (par exemple le Web pour JavaScript et les bibliothèques scientifiques pour Python), ce que savaient mal faire leurs prédécesseurs. Mais la facilité d'apprentissage n'est pas tout : comme le disait mon ami Gilles Kahn, parmi les bateaux, il y en a un qu'on apprend plus facilement à manier que les autres : le pédalo. Mais il vaut mieux ne pas trop s'en servir pour aller loin... J'espère qu'on se réapercevra bientôt que programmer vite et sans filet n'est pas la meilleure idée dès que les bugs ont vraiment des conséquences. Malheureusement, l'expérience a montré que tenter de remettre au carré un langage qui n'a pas été conçu pour ça fonctionne rarement. Je ne suis cependant pas sûr que mon opinion ait une grande force par rapport à celles des processus de sélection naturelle à l'œuvre depuis toujours dans le champ des langages – tout le monde n'a pas ma vision que le bon fonctionnement garanti est plus important que la facilité apparente de développement.

LOGICIELS OUVERTS OU FERMÉS ?

L'informatique existe sous deux formes, ouverte ou fermée. La forme ouverte est aussi appelée *open source*, et *libre* si le logiciel peut être diffusé « librement » (je ne parlerai pas des questions de licences où diverses visions de la liberté s'opposent). Le logiciel ouvert a traditionnellement été préféré dans la recherche, la forme fermée dans l'industrie. Il y a une raison naturelle à cela : les chercheurs sont payés indépendamment des programmes qu'ils écrivent, pas les industriels pour qui les revenus du logiciel peuvent être la ressource numéro un – ce que j'ai personnellement vécu chez Esterel Technologies. Tout chercheur qui construit une start-up comprend vite la différence : il est difficile de gagner sa vie en vendant du gratuit, même si les services et la maintenance peuvent suffire pour vivre dans certains cas.

Les deux mondes étaient autrefois très séparés aux plans technique et idéologique, les chercheurs étant farouchement ouverts et les industriels farouchement

fermés. Les choses ont progressivement changé avec l'accroissement constant de la taille, de la difficulté et de l'impact des logiciels. On voit de plus en plus les industriels adopter et même favoriser les créations des chercheurs. Un bon exemple est Linux, système d'exploitation ouvert et particulièrement solide qui joue un rôle de plus en plus important dans un nombre croissant de domaines, des superordinateurs aux systèmes embarqués en passant par les serveurs et postes de travail. Les industriels soutiennent aussi d'autres logiciels libres qu'ils jugent meilleurs que les leurs et leur coûtent bien moins cher en maintenance. Mais on constate aussi l'effet inverse : nombre de mes collègues qui ne juraient que par Linux et le logiciel libre travaillent maintenant avec des machines Apple, le monde le plus fermé qui soit, et les laboratoires de recherche tirent de plus en plus de revenus des logiciels qu'ils licencient aux industriels. Il est vraiment souhaitable que les deux mondes continuent à coexister pacifiquement, chacun comprenant pourquoi l'autre n'a pas la même opinion sur tous les sujets.

Un mouvement très important est celui de la création collective de grands logiciels. Linux en est l'exemple phare, mais il faut bien sûr mentionner aussi les évolutions d'Internet et du Web, le grand logiciel distribué NTP pour la synchronisation mondiale de l'heure dans les ordinateurs, les bibliothèques pour le traitement d'image ou la génomique, les logiciels TeX et LaTeX pour la publication scientifique dont j'ai beaucoup parlé au [chapitre 8](#) et auxquels beaucoup de contributions sont arrivées de l'extérieur, la remarquable bibliothèque de géométrie algorithmique CGAL ([Boissonnat, 2017](#)), et bien d'autres bibliothèques scientifiques. J'arrête là cette énumération, qui pourrait prendre des pages. Après l'avoir longtemps boudé, voire combattu, les industriels adhèrent maintenant à ce mouvement, car ils se rendent compte qu'ils ne peuvent plus tout faire et que les logiciels développés collectivement peuvent être aussi bons que les leurs, voire meilleurs. Je pense que ce mouvement ira en s'amplifiant à l'avenir.

L'accès aux données et la tension public-privé

Si l'informatique était autrefois dominée par le logiciel, les données y prennent de plus en plus d'importance, leur propriété donc aussi. J'en ai peu parlé car ce n'est pas ma spécialité. On consultera avec profit le travail de Serge Abiteboul ([Abiteboul, 2012](#)), qui est, lui, vraiment compétent. Mais il est clair qu'un mouvement similaire à celui des logiciels s'y retrouve, avec le même type de conflits entre fermeture et ouverture. Beaucoup d'organismes de recherche et d'industriels se rendent compte que leurs données ont une valeur marchande, et beaucoup de chercheurs et d'ingénieurs de ces mêmes organismes sont intéressés à pouvoir accéder à un maximum de données provenant d'autres organismes. Un bon exemple est celui des données génomiques en biologie : séquencer et stocker les données coûte cher, ce qui fait que les instituts de recherche et industriels cherchent à valoriser leurs données. Mais les chercheurs académiques et les industriels savent qu'ils n'auront de bons résultats qu'en analysant un maximum de données, ce qui les fait pousser aussi pour une ouverture maximum. La situation n'est pas du tout claire pour l'instant, et je ne continuerai pas sur ce sujet par manque de compétence – je ne sais d'ailleurs pas qui en a une vision claire.

Un autre grand problème subsiste, celui de l'*anonymisation* des données. Par exemple, pour les données médicales, on va devoir croiser beaucoup de données de beaucoup d'individus. Comment anonymiser ces données éminemment sensibles ? Effacer les noms est évidemment insuffisant, remplacer les dates de naissance par les

années fait perdre beaucoup d'information par exemple pour corréler les maladies infantiles aux périodes dans l'année, etc. Mais garder trop de données peut permettre de retrouver précisément à qui elles appartiennent en croisant ces informations avec d'autres bases pour qui a la puissance de calcul suffisante. Ces questions sont encore largement ouvertes aux plans techniques et juridiques.

À propos d'anonymisation, ne nous leurrions plus : il est devenu quasiment impossible d'être anonyme sur nos ordinateurs et nos téléphones, vu le nombre de logiciels cachés qui nous marquent à la culotte en permanence. Si vous ne savez plus où vous habitez ni qui vous êtes, tapez *where am I* et *who am I* dans vos navigateurs et vous comprendrez... Et si vous mettez vos données sur les réseaux sociaux, vous devez savoir qu'elles ne sont plus qu'à vous. Il est bien dommage que le système juridique s'intéresse à ces questions si tard, maintenant que tout cela est profondément installé...

Apprentissage automatique ou intelligence artificielle ?

Voici deux sujets très à la mode en cette année 2017. Pour tout dire, on ne parle plus que d'eux, en dehors de la sécurité et du quantique bien sûr. On le fait souvent en les considérant comme équivalents, ce que je pense erroné. Voici quelques avis personnels, justifiés seulement par mon expérience sociale et mes intuitions, que d'autres contrediront certainement.

L'APPRENTISSAGE AUTOMATIQUE

Une chose est devenue évidente pour moi : à condition qu'il soit préalablement nourri par un grand nombre de données bien annotées, l'apprentissage automatique supervisé par renforcement ou mixte donne des résultats étonnants dans de nombreux domaines, en surpassant souvent des algorithmes plus classiques. C'est vrai dans des domaines de la perception, par exemple pour analyser des images médicales ([chapitre 7](#)) ou reconnaître la parole, alors même que nous ne savons pas vraiment comment notre propre cerveau fonctionne. Pour les images, il y avait déjà de bons algorithmes fondés sur des approches résolument mathématiques, mais c'était bien moins vrai pour la reconnaissance de la parole.

La traduction des langues est un autre exemple remarquable, car les algorithmes fondés sur des règles et sur des considérations linguistiques n'ont jamais donné de résultats vraiment satisfaisants, comme d'ailleurs pour le diagnostic médical.

Maintenant, les systèmes fondés sur l'apprentissage par lecture de très nombreux couples originaux-traductions deviennent impressionnants. Mais attention : les traducteurs actuels ne comprennent rien aux textes qu'ils traitent et ne connaissent pas de notions évoluées de style. Les littéraires ne les considèrent donc pas comme compétents pour traduire de la poésie ou de la grande littérature, et ils ont raison. Cela n'empêche que ces systèmes sont extrêmement utiles pour quiconque se promène sur Internet ou dans un autre pays, ou encore cherche à savoir ce que contiennent un document ou une notice. De plus, parvenir à traduire efficacement sans comprendre nous pose aussi des questions sur la nature profonde des langues et de leur apprentissage. Souvenons-nous quand même que les enfants aussi apprennent à parler seulement par exposition permanente aux langues, même s'ils ont des parties de leur cerveau bien préparées pour cela, et qu'ils n'abordent la grammaire et l'esthétique que bien plus tard.

Un grand mystère est qu'on ne comprend pas vraiment pourquoi les réseaux profonds donnent des résultats aussi bons sur autant de problèmes pour lesquels les algorithmes classiques pourtant beaucoup travaillés ne sont pas très bons, tout en s'appliquant à une classe de problèmes particulièrement variée. Cela dit, nous ne comprenons pas non plus comment marche notre « pifomètre », instrument biologique fort général qui guide une grande partie de nos actions. Pourquoi certains ont-ils un meilleur pifomètre que d'autres ? Mais, de la même façon que notre pifomètre s'applique un peu partout, un grand intérêt de l'apprentissage automatique est que son pifomètre synthétique peut s'appliquer à une grande variété de sujets (séparément pour l'instant, mais sans subir comme nous les moments d'exaltation, de fatigue ou de dépression). Et nous avons à faire constamment beaucoup d'activités traitables par ce moyen. Conduire une voiture est un bon exemple. Le voyons-nous comme une activité intéressante ? Cela dépend des pays. Aux États-Unis, conduire est plutôt vu comme une corvée ennuyeuse, ce qu'on comprend vite quand on va dans la Silicon Valley, et on y attend impatiemment la voiture automatique. En Italie ou en Inde, c'est plutôt un sujet de fierté...

Même s'il utilise les mêmes machines que les algorithmes classiques, je pense qu'il faut voir l'apprentissage comme un domaine très différent, de nature plutôt analogique comme le dit Stéphane Mallat ([Mallat, 2017](#)). Et je pense qu'il y a beaucoup de recherches à faire pour trouver de nouvelles façons matérielles de le

réaliser, par nature plus analogiques que nos machines binaires. J'ai dit au [chapitre 1](#) que le calcul analogique avait été supplanté par le calcul numérique à cause de sa trop grande sensibilité au bruit. Pour les neurones du cerveau, qui semblent fonctionner dans une sorte de mode mixte analogique-numérique, le bruit est non seulement omniprésent mais apparaît de plus en plus comme un paramètre indispensable au bon fonctionnement. Pouvons-nous en tirer parti pour construire des machines bio-inspirées mieux adaptées que celles qui se restreignent pour l'instant aux opérations matricielles sur GPU ? Cette idée est poursuivie par plusieurs laboratoires en conjonction avec les neurosciences. Je pense qu'il faut suivre leurs progrès de près, même si rien n'est garanti dans un sujet aussi fondamental, malgré les affirmations enthousiastes aussi destinées à obtenir des financements grandioses.

L'INTELLIGENCE ARTIFICIELLE, EN QUOI EST-CE DIFFÉRENT DE L'APPRENTISSAGE ?

Dans les médias, on fait en ce moment (2017) une confusion quasi complète entre apprentissage et intelligence artificielle. Certes, l'apprentissage automatique fait résolument partie des recherches en intelligence artificielle, surnommée IA en français et AI (*artificial intelligence*) en anglais ; certes il connaît de grands succès. Mais il n'est pas *toute* l'intelligence artificielle, loin s'en faut, et ses succès ne s'appliquent donc pas à *toute* l'intelligence artificielle.

Quel est en fait le sujet global ? Voici ce que j'en comprends, qui pourra probablement être complété ou contredit par d'autres, tant les avis sont divers dans ce domaine. Il a démarré dans les années 1950 par un magnifique article d'Alan Turing ([Turing, 1950](#)) – encore lui. Turing commence par poser une question simple : *les machines peuvent-elles penser ?* Notant immédiatement la difficulté de la définition du sens même de cette question, il propose de la remplacer par son test pragmatique bien connu, consistant à savoir si on peut construire une machine qui ne puisse être distinguée d'un humain lors d'une discussion écrite en aveugle. Il démonte une série d'objections possibles à la validité de ce test, puis aborde dans une dernière partie la question de l'apprentissage, qui lui permet de répondre à l'objection « une machine ne fait jamais que ce qu'on lui a dit ». Il pense alors qu'on pourra construire des

machines qui apprendront seules, et qu'on ne pourra même plus comprendre leur fonctionnement intime. C'est une jolie façon de dire que ces machines ne se limiteront plus de façon triviale à « ce qu'on leur a dit » : la connaissance proviendra autant des données emmagasinées que du programme. Nous sommes effectivement près de ce point avec l'apprentissage moderne (savoir si le test de Turing a été déjà réussi ou pas est une question beaucoup débattue chez ceux qui essaient, mais je me sens peu concerné).

Mais Turing n'employait pas l'expression « intelligence artificielle ». Le lancement du domaine eut lieu lors d'une réunion de chercheurs à Dartmouth College (États-Unis), en 1956, avec en particulier John McCarthy et Marvin Minsky, qui allaient devenir des leaders du domaine, avec tous deux de vraies avancées. Les débuts furent encourageants, avec la réalisation de programmes totalement nouveaux par rapport à ce qu'on faisait ailleurs : jeux comme les dames, raisonnements logiques, travail sur la langue, etc. Dans les années 1960, les financements abondèrent. Herbert Simon clama : « Les machines seront capables, dans vingt ans, de réaliser tout ce qu'un homme peut faire » ; et Marvin Minsky ajouta : « Dans une génération, le problème de créer l'intelligence artificielle sera pratiquement résolu. » Tout cela s'est bien sûr avéré profondément faux, et les fonds furent coupés en 1974. La première nuit de l'IA commençait.

Une nouvelle époque commença au début des années 1980, avec le début des systèmes experts et l'augmentation de la puissance des machines. Le Japon clama alors qu'il allait rapidement remplacer les ordinateurs classiques par des ordinateurs fonctionnant uniquement à l'intelligence artificielle, puis lança son projet de « cinquième génération » d'ordinateurs. L'idée était de construire des machines spécifiques, entièrement programmées dans une version parallèle de Prolog, le très novateur langage de programmation logique créé en France en 1972 par Alain Colmerauer (hélas décédé en mai 2017). Les Japonais se pensaient alors invincibles en matière de technologie, ce qui n'empêcha pas le projet de finir dans un fiasco complet, comme j'ai pu le constater lors d'une visite là-bas en 1985. La machine, très lente, était déjà dépassée par les microprocesseurs classiques, plus rapides, plus simples et bien moins chers. Ce fiasco mit à genoux la recherche japonaise pour un bon moment, et signa le début de la deuxième nuit de l'IA.

Tout cela n'avait cependant pas été sans fruits latéraux. Plusieurs sujets de première importance sont nés sous la bannière de l'IA, avant de la quitter pour prendre leur autonomie au sein de l'informatique « classique », qui élargissait ainsi son spectre : le calcul formel, la vérification de programmes, le traitement d'images et l'interface homme-machine en sont des exemples. D'autres chercheurs se considéraient encore comme membres de la communauté IA, mais de façon plus discrète et sans la prétention qui lui avait été si nuisible : ont été initiés dans les labos d'IA l'analyse d'images, la programmation par contrainte dont nous avons vu l'usage en biologie au [chapitre 8](#), le jeu d'échecs (culminant avec la victoire du Deep Blue d'IBM contre Kasparov en 1997 puis la domination continue des logiciels sur les hommes), le traitement de la langue, la robotique, la partie informatique des neurosciences, l'apprentissage automatique, etc. Beaucoup de ces contributions furent considérables, combinant algorithmes sophistiqués et augmentation constante de la puissance des machines.

ALGORITHMES BRILLANTS OUI, APPRENTISSAGE OUI, MAIS INTELLIGENCE ?

Actuellement, l'apprentissage automatique supervisé montre ses capacités et permet d'attaquer de façon nouvelle bien des problèmes difficiles intraitables avant. Mais résout-il le problème initial de l'intelligence artificielle ? Certainement pas, au moins selon moi.

Le problème est que dans la plupart des discussions on prend grand soin de ne pas définir le mot « intelligence », l'arrangeant à la sauce de la discussion en cours. Or ce mot est bien trop riche pour se laisser définir simplement. Quand je lis les dictionnaires, je ne suis pas sûr que les auteurs des définitions de l'intelligence en soient particulièrement fiers. On peut bien sûr définir l'intelligence comme « ce que mesurent les tests de QI », mais y croire, c'est autre chose. Marvin Minsky proposait une définition intéressante : il voyait l'IA comme la science de la conception de machines capables de faire des choses qui demanderaient de l'intelligence aux hommes. Mais cette tentative reste pour moi peu satisfaisante, par exemple par ce que la multiplication demande de l'intelligence aux hommes mais pas aux machines.

L'apprentissage est certainement indispensable à l'intelligence humaine, comme l'ont montré les enfants-loups. Mais celui qui est en place actuellement en informatique est très supervisé, et je ne pense pas que l'intelligence humaine puisse s'y réduire. L'apprentissage d'un savoir-faire est-il suffisant pour parler d'intelligence, ou faut-il aussi inclure la compréhension et la créativité, bien plus mystérieuses ? La créativité peut-elle résulter seulement de l'apprentissage supervisé, ou demande-t-elle aussi préalablement de maîtriser plusieurs formes d'apprentissage non supervisé, probablement bien plus difficiles et sur lesquelles les chercheurs n'ont pas encore vraiment avancé, plus d'autres qualités pas encore analysées ? L'intelligence est-elle seulement personnelle, ou n'est-elle pas plutôt collective, à travers la culture d'une population ? Je ne connais pas les réponses à ces questions.

Il faut aussi remarquer qu'un logiciel d'apprentissage du go joue très bien au go, mais qu'il ne sait faire rien d'autre. Son très long apprentissage ne lui sert à rien pour lire ou parler. Un enfant n'apprend pas très vite, mais il le fait dans un très grand nombre de secteurs. Les chercheurs du domaine reconnaissent bien que cette forme d'intelligence générale est loin d'être encore à leur portée.

Donc, quand j'entends des phrases comme « dans trente ans, la machine va être plus intelligente que l'homme, ce sera la singularité », je suis plus que sceptique. Ce n'est pas que je pense que la chose soit impossible, car l'affirmer me demanderait une compétence que je n'ai pas, mais je me demande bien quelle est la compétence réelle de ceux qui s'autorisent à la formuler ainsi dans l'état actuel des connaissances. J'ai la même impression quand on me dit que « dans vingt ans, on saura parfaitement stocker l'électricité », ce que je n'ai jamais entendu de la part des gens sérieux du domaine. Le problème est plutôt que médias et spectateurs sont prêts à avaler ce genre de phrases toutes crues, en rajoutant même pour faire plus vrai.

Comme pour l'informatique quantique, le sujet de la recherche d'une intelligence artificielle à divers degrés est magnifique et motive plein d'excellents chercheurs. Laissons-les travailler, profitons sereinement des fruits de leur travail, surveillons ses dangers potentiels, mais ne mettons pas la charrue avant les bœufs⁶ pour faire du sensationnel toujours plus facile à vendre que la bête réalité.

Enfin, quand le grand physicien Stephen Hawking nous dit que l'intelligence artificielle risque de détruire l'humanité, je suis tenté de lui répondre ainsi : « Certes, c'est possible à terme, mais pensez donc à la bombe H que vous physiciens avez

créée. Elle est vraiment efficace pour détruire l'humanité. Ce n'est pas une supposition, c'est tout de suite : il y en a déjà bien plus que nécessaire, gérées on ne sait pas comment. Pourtant, on en discute bien peu... »

L'informatique dans la société

Les algorithmes interviennent de plus en plus dans notre vie sociale, sans que les gens les comprennent. On leur donne des rôles de plus en plus importants, comme de nous envoyer des montagnes de publicité « ciblée », de nous dire quel livre lire et quel film voir, de décider de l'affectation des bacheliers, de gérer des systèmes de vote, de contrôler les organes de nos voitures, de dire à nos assurances comment nous conduisons pour qu'elles puissent moduler leurs primes, et bientôt peut-être de les conduire sans notre intervention. Tout cela est très peu discuté à l'heure actuelle, bien trop peu selon moi.

Mais, sur ce sujet, je n'ai pas grand-chose à ajouter au beau livre *Le Temps des algorithmes* de mes collègues Serge Abiteboul et Gilles Dowek ([Abiteboul & Dowek, 2017](#)), plus compétents que moi sur ces sujets. J'en reprends simplement quelques situations que je présente à ma sauce pour donner au lecteur l'envie d'aller plus loin en le lisant.

Grâce à la facilité de circulation de l'information, les notions de bien privé et de bien communautaire vont être remises en question. Un bel exemple est celui de la voiture. Avant, il était souvent indispensable d'en posséder une pour se déplacer librement, en louer une était pénible, et les gens ne le faisaient pas même dans les cas où c'était plus économique : réservation à faire, paperasse à remplir, horaires à respecter, etc. Mais s'il suffit de quelques clics sur son téléphone pour trouver une voiture tout près, pourquoi s'embêter à garer et entretenir la sienne, au moins quand on habite en ville ? Autolib' n'est probablement que le début d'un mouvement profond, qu'on voit aussi dans la location instantanée d'appartements au lieu de

chambres d'hôtel. Tout cela ne touche pas que les fournisseurs et les consommateurs, comme on le voit aux difficultés des systèmes légaux dans les villes américaines ou autres.

Si les voitures deviennent automatiques, qui sera responsable des mauvaises décisions ou des accidents éventuels (plus précisément, certains, comme avec les conducteurs humains) ? Aux États-Unis, les fabricants cherchent visiblement à ce que le conducteur soit responsable. En Europe, la situation est différente. En France, la loi interdit à un constructeur de se décharger de ses responsabilités. Et si la voiture demande au conducteur de toujours pouvoir reprendre la main, le pourrait-il vraiment ? Je pense illusoire de croire qu'un conducteur à qui on demande de ne pas conduire va disposer d'une attention aiguisée et des réflexes rapides en cas de vraie difficulté, d'autant plus qu'il aura certainement du mal à comprendre ce que la voiture a décidé et pourquoi. Ce phénomène est bien connu des pilotes d'avion, mais eux sont des professionnels. Jolis problèmes légaux en perspective...

Voici un autre bel exemple adapté de Dowek et Abiteboul : dans nos pays, les élections ont lieu à un débit de l'ordre d'une par an tous scrutins confondus, avec un nombre de choix faible à chaque fois, disons au maximum 32, ce qui fait que la réponse tient sur 5 bits. Donc la participation de chaque citoyen aux choix démocratiques engageant la collectivité est limitée au mieux à 5 bits par an. Au ^{xx}e siècle, quand les votes étaient organisés de façon lourde et demandaient de se déplacer, le bit était cher et c'était raisonnable. Mais maintenant ? Ne pourrait-on pas trouver ou au moins chercher d'autres solutions⁷ ?

Enfin, j'adhère pleinement à la vision d'Abiteboul et Dowek sur la nécessité de vérification externe des algorithmes utilisés par l'État et impactant directement nos vies. Un bon exemple est APB, l'admission postbac qui pose tant de problèmes en cette année 2017 (pas forcément de la faute de l'algorithme d'ailleurs). Qui nous dit que cet algorithme fait ce qu'il doit faire, et pourquoi le croirions-nous ? Le mieux serait que ses spécifications, la description mathématique de l'algorithme et le code source du programme puissent être rendues publiques et scrutées par des personnes compétentes et indépendantes, comme on le fait en avionique. Sinon, qui nous dit que l'algorithme et le programme n'ont pas de bugs pénalisants pour les étudiants concernés ? Un effort d'ouverture a été fait récemment, mais il reste insuffisant.

J'arrête là ces amuse-bouches en laissant le lecteur en trouver de nouveaux pour aborder une question qui me tient particulièrement à cœur, l'enseignement général de l'informatique.

Enseigner l'informatique, du primaire au lycée

L'introduction de l'informatique dans l'enseignement général pose deux questions différentes : celle de la matière scientifique et technique, et celle de l'usage des technologies numériques pour enseigner mieux dans toutes les matières. Je ne parlerai que de la première, n'ayant ni l'expérience ni la compétence requise pour la seconde.

Pour toute ma communauté, introduire la science et la technique informatiques dans l'enseignement général est essentiel pour que notre pays reste au niveau international qui doit être le sien. Je pense qu'il faut formuler cette problématique de façon directe, voire brutale, ce que j'ai souvent fait dans les médias :

Pouvons-nous encore accepter que nos enfants arrivent au baccalauréat en ne sachant comme maintenant absolument rien de la science et de la technique de l'informatique, alors que cette discipline va structurer leur vie et leurs métiers d'une façon de plus en plus profonde ?

Il est utile de faire une comparaison avec l'enseignement de la physique. Dès le XIX^e siècle, l'introduction des inventions de la physique dans l'enseignement a été rapide, car elles entraient bien dans le schéma mental collectif d'une société fascinée par les progrès scientifiques et techniques liés à la matière, à l'énergie et aux ondes : téléphone, moteur thermique, train, automobile, moteur électrique, radio,

télévision, etc. Une raison clef selon moi est que les sujets touchés soit faisaient directement partie intégrante du monde sensible, soit y revenaient par les objets matériels que sont les postes de radio et de télévision pour les ondes. Cela n'a absolument pas été le cas pour l'informatique, pas directement accessible par les sens et longtemps boudée par le système d'enseignement. Par exemple, plusieurs professeurs de technologie m'ont dit récemment qu'il n'était pas question pour eux d'enseigner l'informatique autrement que liée au monde matériel, le reste ne faisant par construction pas partie de leur champ de compétence et d'action. Ils enseignent donc principalement avec des robots, ce qui n'est pas un problème en soi car la jonction information-matière est très importante comme j'ai essayé de l'expliquer tout au long de ce livre. Mais il serait temps de reconnaître une fois pour toutes qu'il n'y a pas que le matériel dans la vie, que l'informatique est aussi de la technologie au même titre que la mécanique, et que c'est même probablement la technologie qui se développe le plus vite et qui va devenir la plus centrale dans notre monde car maintenant au cœur de toutes les autres.

Heureusement, ce problème de schéma mental trop rigide n'existe pratiquement plus dans l'enseignement supérieur, dont je ne parlerai pas ici, ni dans les classes préparatoires aux grandes écoles qui se mettent progressivement au goût du jour avec de vrais programmes scientifiques pour l'informatique. J'ai aussi déjà dit et j'espère avoir montré dans mes exemples que la recherche française en informatique au sens large était remarquable sur le plan mondial, et je n'y reviendrai pas, à part pour dire qu'elle s'implique de plus en plus dans l'enseignement et la diffusion scientifique. Je resterai donc concentré sur l'éducation générale du primaire au lycée.

LA SITUATION À L'ÉTRANGER

Au [chapitre 1](#), j'ai fait un petit rappel sur la triste histoire ancienne de l'informatique en France, que je ne reprendrai pas ici pour m'intéresser davantage au futur. Je souligne cependant que la situation a été assez variée en Europe. En Angleterre, à la suite d'un effort très important conduit par une association nommée Computing at School sous le leadership éclairé de Simon Peyton Jones⁸, grand chercheur dans le domaine de la programmation, un rapport remarquable de la Royal

Society ([The Royal Society, 2012](#)) a affirmé qu'un vrai enseignement d'informatique était une nécessité vitale pour le pays. En 2013, le Premier ministre Gordon Brown a fait siennes ses conclusions et demandé que l'informatique soit une matière scientifique à égalité des autres au baccalauréat, avec un enseignement commençant dès le primaire. La Pologne a également beaucoup bougé. D'autres pays comme la France et l'Italie se contentaient d'un enseignement de l'usage des ordinateurs et de quelques logiciels, dont le lecteur comprend mieux maintenant j'espère pourquoi il ne correspond pas du tout à la question de notre avenir. Plus de données européennes sont disponibles dans le rapport de l'association Informatics Europe ([Informatics Europe, 2017](#)) sur la situation en 2015.

Aux États-Unis, la situation de l'enseignement général n'est pas simple, mais le président Obama s'est personnellement engagé dans la bataille pour l'enseignement de l'informatique. En Asie, le sujet est considéré comme absolument majeur dans beaucoup de pays, dont l'Inde et la Chine qui fournissent des cohortes d'ingénieurs, de chercheurs et de sociétés ; quand j'allais chez Intel ou dans d'autres sociétés aux États-Unis, la proportion d'Indiens ou de Chinois était impressionnante. C'est la même chose pour les universités et centres de recherches, où les Européens sont d'ailleurs aussi très présents.

L'IDÉE D'UN VRAI ENSEIGNEMENT D'INFORMATIQUE S'INSTALLE EN FRANCE

Heureusement, les choses progressent chez nous. Avec certains collègues chercheurs, professeurs, et plus tard inspecteurs généraux travaillant en réelle harmonie, nous avons lentement pu convaincre de plus en plus d'acteurs majeurs de la société de l'importance et de l'urgence de la question et du fait qu'il était possible d'y apporter des réponses. Nous secondions depuis longtemps mon ancien directeur de thèse Maurice Nivat dans ses efforts pour la promotion de l'enseignement général de l'informatique, mais avec des résultats pour le moins mitigés.

En janvier 2008, je terminai ma première leçon inaugurale au Collège de France par une phrase assez directe à propos de l'enseignement de l'informatique : « En ce qui concerne l'éducation, la France rentre résolument dans le xx^e siècle. » Cette

phrase a touché l'oreille de conseillers du ministère de l'Éducation nationale, avec qui nous avons commencé à travailler. Je passe sur les attermolements et les changements de ministres. À la rentrée 2012, nous avons pu introduire un enseignement de spécialité « Informatique et sciences du numérique (ISN) » et commencer le travail sérieux, y compris la rédaction de livres pour les élèves et les professeurs ([Dowek & Archambault, 2013](#)). Cet enseignement a été ouvert dans certaines académies mais pas dans d'autres, avec un bon succès là où il l'était. Il a actuellement touché plus de 50 000 élèves (mais trop peu de filles, j'en reparlerai).

En 2013, j'ai eu l'honneur de coordonner un rapport de l'Académie des sciences sur l'enseignement de l'informatique ([Académie des sciences, 2013](#)), auquel beaucoup prédisaient un avenir de fond de tiroir. Ça n'a pas été le cas. En 2015, un enseignement optionnel « Informatique et création numérique (ICN) » a été introduit en seconde. Le tournant le plus important a eu lieu également en 2015, quand la ministre de l'Éducation nationale Najat Vallaud-Belkacem avec qui nous étions directement en contact a demandé au CSP (Conseil supérieur des programmes) de préparer des programmes d'informatique allant de la primaire au baccalauréat et de prendre contact avec notre groupe. Les choses n'ont pas été simples, en particulier parce que le CSP ne contenait que très peu de scientifiques sur 18 membres, dont aucun n'était familier avec le sujet. Mais la coopération a été bonne et des programmes je pense raisonnables établis, avec mise en place progressive à partir de 2016.

Depuis, la machine se lance doucement. Je passe sur les détails des enseignements décidés, qui sont soit nouveaux comme au Collège, soit des extensions de ceux qui existaient déjà. Le plus important est que certains professeurs se forment et s'organisent, des organisations comme le projet Class'Code⁹ de l'Inria proposent des activités des outils de formation des professeurs. Le site Interstices fournit aussi beaucoup d'informations, et bien d'autres sites français ou étrangers font de même. La SIF (Société informatique de France) s'implique également beaucoup.

Sur le plan du contenu de l'enseignement, on nous demande souvent si on sait ce qu'il faut enseigner en informatique et comment le faire, en ajoutant souvent : « et avec quel langage ? ». Ce sont des questions de mieux en mieux comprises, même si elles resteront par nature en évolution permanente. Avant de les développer, je signale un événement récent et important pour nous. La Fondation La Main à la pâte, créée en

2011 par l'Académie des sciences et des écoles normales supérieures de Paris et Lyon sous l'impulsion de Georges Charpak, fait depuis un gros travail sur le développement professionnel des professeurs vers un enseignement plus actif des sciences naturelles porté par une démarche d'investigation. Cette vision d'un enseignement actif, que j'ai bien connu en tant que président d'une école Montessori, est certainement celle qu'il faut adopter pour l'informatique : le mélange de réflexion, d'expérimentation et d'activités organisées par projets est la bonne façon de progresser. La Main à la pâte a maintenant décidé de s'intéresser à l'informatique. Coup d'essai, coup de maître : La Main à la pâte vient de sortir en liaison avec Class'Code deux livres de formation des professeurs très remarquables sous le nom *1, 2, 3... Codez !* ; un pour le cycle 3 (fin du primaire et 6^e) et un pour le cycle 4 (5^e, 4^e et 3^e). J'ai eu l'honneur de préfacer le second volume, et je recommande vivement la lecture des deux.

LES TOUT JEUNES ÂGES ET LE PRIMAIRE

Aux tout jeunes âges, du bébé à l'enfant en primaire, il faut parler davantage d'éducation que d'enseignement de matières. Il faut aussi bien avoir conscience du fait que l'éducation doit concerner tous les sens et leur interaction, en mettant la motricité au même plan que l'intellect, et en se souvenant que l'interaction avec les autres doit être en harmonie avec la progression personnelle. Dès qu'ils sont heureux, ce qui n'est hélas pas toujours le cas, les enfants sont des curieux et des expérimentateurs invétérés. L'éducation doit leur donner leur nourriture, pas leur dire qu'il n'y a qu'une façon de résoudre les problèmes. J'ai moi-même enseigné l'informatique aux 6-9 ans dans l'école Montessori Les Pouces verts que j'avais la charge et l'honneur de présider, et je peux dire que j'ai bien été souvent surpris par l'inventivité et le sérieux des enfants.

La première chose à faire est de résister fermement à la tentation trop fréquente de donner librement aux enfants des objets informatiques attirants comme les tablettes et téléphones. Trop de parents sont fiers d'observer à quel point leurs enfants sont des « petits génies », puisqu'ils savent immédiatement manipuler ce genre d'appareil alors qu'eux peinent encore. Non, ce ne sont pas des petits génies, la manipulation et l'expérience sont simplement naturelles pour eux. D'ailleurs, il ne faut jamais oublier

que nous, parents, sommes tout aussi incapables de faire des constructions pleines d'imagination avec des briques de Lego, chose pour laquelle on n'a jamais qualifié un enfant de petit génie. Trop de tablette polarise indûment l'attention et limite les apprentissages proprioceptifs et relationnels. C'est maintenant bien dénoncé par les pédiatres qui vont jusqu'à parler de formes de pré-autisme. Le sujet est sérieux, et il commence à être traité sérieusement.

En revanche, il serait diablement utile d'apprendre aux petits que lorsqu'on leur apprend l'addition ou toute autre opération de ce style, on leur apprend un algorithme, et, de façon plus générale, que toute opération qu'on peut faire de façon systématique est un algorithme. C'est loin d'être anodin, car c'est la première brique de la construction du schéma mental. Mais cette idée toute simple est loin d'avoir pénétré le système.

On peut ensuite construire des activités plus directement informatiques de deux façons, soit « débranchées », c'est-à-dire sans ordinateur mais très formatrices, soit avec des logiciels adaptés, mêlant expérimentation et réflexion dans le style de « La Main à la pâte ». Le volume 1 du livre *1, 2, 3... Codez !* destiné aux professeurs en présente plusieurs dans une progression naturelle. Il propose aussi un éclairage scientifique sur l'histoire de l'informatique, fort utile car il est toujours bon de ne pas dissocier les questions techniques des questions historiques, les secondes stimulant toujours la curiosité des enfants. L'outil utilisé pour l'expérimentation est le langage graphique et logiciel Scratch du MIT. Ce n'est pas le seul choix possible, mais il me paraît excellent. Scratch n'est pas qu'un langage de programmation graphique : c'est aussi un bel outil de création de scènes animées, qui permet une programmation parallèle d'activités synchrones de personnages ou d'objets qui peuvent s'envoyer des messages mais aussi en recevoir de leur environnement, par exemple quand ils arrivent sur les bords du cadre ou se cognent entre eux. Cela donne beaucoup de possibilités de programmation et d'interaction. Scratch permet aussi de jouer avec des décors, des mouvements, des sons, etc. L'expérience montre que les enfants se l'approprient facilement et y fabriquent des applications intéressantes. De plus, Scratch, logiciel gratuit, est en train de devenir un standard mondial pour l'enseignement, avec de grands réseaux d'échanges de problèmes et de solutions à travers le monde.

LE COLLÈGE ET LE LYCÉE

Ensuite, la progression est naturelle. Le collège est la place d'une expérimentation plus riche et complexe, qui peut toujours se faire en bonne partie en Scratch, mais aussi au tableau, avec d'autres langages, ou à l'aide de robots ou autres dispositifs. Les aspects mathématiques des algorithmes peuvent aussi y être abordés, en résistant à la tentation naturelle de déguiser le cours d'algorithmique en un cours de maths classique. Voir par exemple le volume 2 de *1, 2, 3... Codez !* ([Wilgenbush, Hirtzig & Calmet, 2017](#)).

Le lycée doit être l'endroit de la conceptualisation et de l'approfondissement, culminant avec de la vraie programmation d'algorithmes sérieux dans de vrais langages, jointe à l'analyse des raisons pour lesquelles ces algorithmes fonctionnent. Le couplage permanent de la théorie et de l'expérimentation doit être aussi important qu'en physique.

LA RELATION AVEC LES AUTRES MATIÈRES

À ces niveaux se posent aussi deux questions importantes : la relation avec les autres matières, et la certification des professeurs. Pour l'instant, il a été décidé de ne pas introduire de discipline informatique au sens propre, mais de partager l'enseignement de l'informatique entre mathématiques et technologie, avec une formation encore minimale des professeurs dont on ne voit pas bien comment elle va évoluer à l'avenir. C'est nettement mieux que d'autres propositions précédentes comme celle de saupoudrer un peu d'informatique dans les autres matières : de l'informatique pour le texte en français, de la cartographie informatique en géographie, des algorithmes mathématiques en maths, etc. Ce genre de proposition bâtarde faciliterait certes l'établissement des programmes et des budgets d'un point de vue bureaucratique, mais elle n'aurait aucun sens pédagogique. Limitée aux côtés strictement utilitaires de chaque matière, elle aurait nié l'unité conceptuelle et pratique de l'informatique, celle qui fait justement sa force. Heureusement, quand ce serpent de mer ressort de l'eau, il est de plus en plus facile d'y replonger sa tête.

UN PROBLÈME CLEF : LA FORMATION ET LA CERTIFICATION DES PROFESSEURS

L'institution ne met à l'heure actuelle que beaucoup trop peu de moyens dans la formation et le développement professionnel des professeurs qui vont enseigner l'informatique, ce qui risque de devenir rapidement un problème bloquant. Par ailleurs, elle résiste fortement à la création officielle d'un Capes et d'une agrégation d'informatique, les deux diplômes fondamentaux en France pour les professeurs. J'ai entendu plusieurs prétextes : si on les ouvre à l'informatique, il faudra le faire pour bien d'autres sujets qui le demandent aussi ; si on le fait, on n'aura plus de candidats profs de maths, qui sont déjà en nombre décroissant ; de plus, les informaticiens étant très bien payés ailleurs, il n'y aurait pas de candidats, etc. Tout cela n'est pas entièrement faux, d'autant plus que le métier de professeur a été malheureusement très dévalorisé au cours du temps par un glissement constant des salaires et de la considération des autorités et des parents d'élèves. Mais toute notre communauté pense que ces raisons tactiques ne doivent pas cacher le problème stratégique, et que la tactique ne doit jamais dominer la stratégie. Notre opinion est simple :

S'il n'y a ni Capes ni agrégation d'informatique, le sujet n'entrera jamais vraiment dans l'enseignement, et notre pays le paiera durablement.

Il pourrait aussi exister de vraies bivalences, comme par exemple en Allemagne où un professeur ne peut pas enseigner une seule matière ; mais, s'il en enseigne deux ou trois, il doit avoir les deux ou trois vraies qualifications correspondantes, ce qui ne changerait rien à la question.

Des solutions intermédiaires sont actuellement implémentées, comme un Capes moitié mathématiques et moitié informatique. Mais il va donner un diplôme unique qui s'appelle toujours Capes de mathématiques, ce qui fait que quelqu'un qui met la priorité sur l'informatique ne pourra pas y être admis par manque de compétences mathématiques. De même, il existe une option informatique à l'agrégation de mathématiques, mais elle parle surtout d'informatique mathématique. Enfin, la solution actuelle de coupure de l'informatique au collège entre mathématiques et

technologie pose un autre type de problème, celui des fréquentes mauvaises relations entre ces deux matières, même si ce n'est pas partout le cas. Où se trouvera l'unité de la matière si les professeurs collaborent mal ?

Notre travail est donc loin d'être terminé. Pour information, le Maroc vient de son côté de décider de créer une agrégation d'informatique, qu'il estime indispensable pour couvrir ses besoins en enseignants bien formés...

Le problème de la parité en informatique

Pour finir ce livre, la répétition étant la mère de la pédagogie, je souhaite répéter quasiment mot pour mot ce que j'ai écrit à la fin de l'introduction ([ici](#)).

Il reste cependant un problème majeur à résoudre : la très faible présence des femmes dans tout ce qui est lié à l'informatique. Le domaine est vraiment très loin de la parité, que ce soit dans la recherche, l'industrie ou l'enseignement. Je n'ai évidemment qu'une analyse partielle du sujet et pas de solution, mais je souhaite rapporter quelques expériences. J'ai par exemple entendu en 2015 deux jeunes femmes qui venaient de passer leur baccalauréat me dire que les conseillers pédagogiques de leurs lycées leur avaient clairement déconseillé d'aller vers l'informatique « car ce n'est pas un métier pour les femmes », alors même que les deux étaient vraiment tentées. Il y a deux drames conjoints : le premier est qu'on le leur fasse croire, le deuxième qu'elles le croient. L'une étudie le droit, mais a bien l'intention de revenir vers l'informatique ; je lui ai répondu qu'acquérir une double compétence était une excellente idée, car le droit de l'informatique est un sujet naissant, passionnant, et pas près de devenir obsolète. L'autre s'est dirigée vers la biologie, sujet tout aussi fascinant où les femmes sont à l'inverse largement majoritaires, mais où les débouchés sont hélas bien plus aléatoires. J'ai interrogé depuis d'autres jeunes chercheuses d'autres disciplines. J'en ai retiré l'impression que même celles qui étaient intéressées ne s'étaient pas senties assez bonnes en maths pour aller vers ce genre de sujet, ce qui est idiot mais rarement démenti par leur environnement, et qu'on leur conseillait plutôt les sciences de la vie. Un autre souci bien plus compréhensible est qu'elles ne se sentaient pas attirées par un métier

massivement masculin – voire elles étaient repoussées par l’image du *geek* mâle, si bien publicisée et pourtant très minoritaire dans les métiers de l’informatique. La même chose se produit apparemment dans presque tous les pays occidentaux.

Il faut tordre le cou à ce genre d’idioties, et le plus vite sera le mieux. Notre communauté se bouge beaucoup pour changer cet état de choses, mais sans grand succès jusqu’à présent. Toutes les idées et actions sont bienvenues, *en particulier dans le système éducatif et dès le plus jeune âge.*

CHAPITRE 12

Annexes

12.1. *La science des algorithmes*

L’algorithmique, science des algorithmes, est centrale en informatique. Je parlerai d’abord des algorithmes classiques, ceux qui prennent des données, calculent, puis rendent des résultats. Ils sont très nombreux, et l’algorithmique a pour objectif d’en découvrir de nouveaux ou d’améliorer les anciens, en veillant toujours à garantir leur *terminaison* sur les données concernées, leur *correction*, qui exprime que l’algorithme fait bien ce pour quoi il a été conçu, et leur *complexité*, qui est leur coût en temps de calcul, mémoire, ou énergie. Analyser ces propriétés fait toujours appel aux mathématiques, et y conduit souvent à de nouveaux développements.

DE L’INDÉCIDABILITÉ À LA COMPLEXITÉ ALGORITHMIQUE

Dès 1936, Turing avait formalisé la notion d’algorithme à l’aide de sa fameuse machine. Il avait aussi montré que tout n’était pas faisable par des algorithmes : son théorème d’*indécidabilité du problème de l’arrêt* établissait l’impossibilité de construire un algorithme T se terminant toujours et décidant si un algorithme précis A travaillant sur des données D va se terminer ou non. Ce résultat fondamental établit la limite la plus dure de l’informatique : si on peut construire une machine universelle simulant n’importe quelle autre machine, on ne peut pas en construire pour décider l’arrêt de cette machine.

Pendant longtemps, ne travaillant que sur du papier, les chercheurs se sont intéressés principalement aux questions de décidabilité/indécidabilité, sans vraiment se préoccuper de la faisabilité pratique des algorithmes qu'ils étudiaient : quels problèmes mathématiquement intéressants peut-on résoudre avec des algorithmes qui s'arrêtent toujours, et pour quels problèmes est-il impossible de construire de tels algorithmes ? Ces questions théoriques restent d'actualité. Mais, dans les années 1960, l'arrivée des premiers ordinateurs utilisables par les chercheurs a changé la donne. Les questions sont alors devenues plus pratiques : si on ne considère que des problèmes que l'on sait résoudre en temps fini, comme le tri de données, l'organisation des horaires de train ou les emplois du temps d'un lycée, quels sont les meilleurs algorithmes pour les résoudre, et combien coûtent-ils en temps de calcul et occupation mémoire en fonction de la taille des données ? On parle alors de *complexité* d'un problème ou d'un algorithme, et on peut classer les différents problèmes en *classes de complexité*. Dans la suite, n désignera la taille des données, sans qu'il y ait besoin de préciser davantage cette notion car les résultats sont très généraux.

Les algorithmes *linéaires* ont un coût linéaire dans la taille des données, c'est-à-dire proportionnel à n . Comme on ne s'intéresse pas au facteur de proportionnalité, qui est indépendant des données, on écrit ce coût $O(n)$. Les algorithmes *sublinéaires* sont encore moins chers, avec par exemple un coût en racine carrée ou logarithme de cette taille, voire un coût constant, indépendant de la taille des données.

Les algorithmes linéaires et sublinéaires sont évidemment très intéressants en pratique. Un exemple particulièrement important d'algorithme de complexité logarithmique est l'addition de deux entiers de n chiffres binaires. L'algorithme de propagation de retenues qu'on apprend à l'école est linéaire, car il faut additionner les chiffres et la retenue courante colonne par colonne. Mais, dans un circuit, on procède différemment en additionnant les colonnes en parallèle et en propageant les retenues de diverses manières astucieuses ; l'algorithme se fait alors en temps logarithmique $O(\log_2(n))$. La différence est considérable, le coût passant par exemple de 64 à 6 pour des entiers de 64 bits. Il est nettement plus difficile de voir qu'on peut aussi faire la multiplication en temps $O(\log_2(n))$, à condition toutefois d'occuper une surface de silicium $O(n^2)$, ce qui n'est pas rien. La *recherche en table*, qui est centrale dans les annuaires, les bases de données et les moteurs de recherche, est encore plus rapide,

parfois en temps constant. C'est essentiel pour les moteurs de recherche, qui indexent plusieurs milliards de pages Web : il vaut mieux que leurs algorithmes soient de coût minimal !

Les algorithmes de coût $n \times \log(n)$ sont plus chers, mais pas beaucoup : si n vaut 1 milliard, son logarithme binaire $\log_2(n)$ ne vaut que 30. Les algorithmes de tri, utilisés à grande échelle sur tous les ordinateurs pour des raisons très diverses, font fort heureusement partie de cette classe. Rappelons-nous d'ailleurs que le mot ordinateur vient d'« ordre » : l'ordinateur est autant une machine à trier qu'une machine à calculer.

Les algorithmes de complexité *polynomiale* ont un coût en puissance fixe de n : n^2 , n^3 , etc. Ils sont considérés comme faisables en pratique, même si cette question peut être débattue quand l'exposant est grand. La plupart des algorithmes calculant sur les matrices, par exemple en analyse numérique ou en apprentissage profond, font partie de cette classe.

Si on met maintenant n en exposant comme dans 2^n ou 3^n , on obtient la classe des algorithmes *exponentiels*, qui sont eux considérés comme infaisables en pratique dès que n n'est pas tout petit : rappelons-nous que 2^{250} , qui vaut à peu près 10^{80} , est considéré comme étant l'ordre de grandeur du nombre de particules dans l'univers. La comparaison avec $250^2 = 62\,500$ est éloquente !

Les classes s'étendent arbitrairement au-dessus, par exemple avec la complexité *superexponentielle* où le coup est donné par 2^{2^n} . Certains algorithmes utiles de calcul arithmétique ou de décision logique sont dans cette classe, mais il faut que leurs données soient petites.

LA CLASSE DES ALGORITHMES NP-COMPLETS

En 1973, Stephen Cook, prix Turing 1982, a fait une découverte fondamentale en caractérisant la classe d'algorithmes qui occupe actuellement le plus les algorithmiciens et les utilisateurs : celle des problèmes *NP-difficiles*, où NP signifie « non déterministe polynomial ». Sous cette appellation technique se cache une définition en fait simple, que l'on peut expliquer avec l'exemple du Sudoku. Si vous voulez résoudre un Sudoku, vous pouvez le faire en essayant tous les nombres

possibles dans toutes les cases blanches, ce qui prend évidemment un temps exponentiel. Mais si vous voulez simplement vérifier qu'une solution proposée est correcte, il vous suffit de vérifier que son tableau satisfait aux règles du Sudoku, ce qui est facile. Si vous disposiez d'un oracle vous donnant magiquement la solution, vous n'auriez qu'à la vérifier et le Sudoku deviendrait un jeu facile.

Les problèmes NP sont les problèmes de ce type. Il en existe des centaines d'usage pratique, allant du calcul de l'emploi du temps optimal dans les lycées ou de celui des horaires de trains au problème SAT de satisfaction booléenne étudié au [chapitre 10](#). Certains de ces problèmes, dont justement le problème SAT, sont dits *NP-complets* : si l'on savait les résoudre efficacement, on saurait alors résoudre efficacement *tous* les problèmes NP. Donc tous les problèmes NP-complets sont de complexités équivalentes. Pour SAT, la méthode consistant à essayer toutes les valeurs possibles des variables marche évidemment, mais elle est exponentielle et donc inutilisable en pratique. Savoir s'il existe des algorithmes non exponentiels pour résoudre SAT ou n'importe quel problème équivalent est l'une des plus grandes questions ouvertes en mathématiques. Beaucoup pensent que ce n'est pas le cas, mais, si c'était vrai, la face du monde pourrait être transformée, tant ces problèmes sont partout essentiels !

L'APPROCHE HEURISTIQUE

Il semble y avoir une contradiction dans la discussion précédente sur le Sudoku. Nous avons vu au [chapitre 10](#) que les algorithmes SAT modernes permettent de résoudre les Sudoku en un temps négligeable. Ce fait d'expérience n'est-il pas contradictoire avec l'affirmation que SAT est difficile, et peut-être exponentiel ? Pas du tout ! Il faut distinguer trois types d'analyses de complexité : celle du *pire cas*, celle *en moyenne* tenant compte de tous les cas possibles, faciles ou non, et celle des cas *utiles*, caractérisés par exemple par des listes de problèmes connus (*benchmarks*). Pour SAT et pour tout problème NP-complet, il est facile de construire des pires cas où tous les algorithmes connus sont exponentiels. Cela n'empêche pas que ces algorithmes marchent bien dans beaucoup de cas utiles, sans qu'on sache d'ailleurs pourquoi. Il est donc très utile de développer des algorithmes qui cherchent à être

efficaces en moyenne ou dans des cas classiques définis par des suites d'exemples (*benchmarks*), ce qu'on appelle des *heuristiques*.

Avec des heuristiques, on sait par exemple calculer des emplois du temps satisfaisants pour les lycées ou des horaires de train, même s'ils ne sont pas optimaux. Tous les algorithmes SAT actuels sont aussi fondés sur des heuristiques, qui sont si efficaces que des algorithmes développés depuis longtemps dans des domaines précis peuvent se faire battre par un simple recodage à la main du problème en SAT. Et SAT fait aussi le bonheur des chercheurs en physique statistique, qui y détectent des changements de phase brutaux et d'autres propriétés qu'on trouve d'habitude dans les assemblées d'atomes.

Mais il existe aussi des problèmes NP-complets où l'on sait qu'il n'y a pas d'heuristique efficace. Un bon exemple est la 3-coloration des graphes, pour lequel le lecteur trouvera facilement des références sur le Web. Ces algorithmes sont très utiles en cryptologie, par exemple pour signer des documents.

LES ALGORITHMES PROBABILISTES

Tous les algorithmes ne font pas que des calculs parfaitement déterminés d'avance. Les algorithmes *probabilistes* ou *randomisés* utilisent des tirages au sort pour explorer de façon approchée l'espace des solutions, la théorie des probabilités permettant de démontrer qu'ils convergent efficacement vers une bonne solution, parfois de façon tout à fait surprenante. Ils deviennent de plus en plus nombreux et performants, par exemple pour analyser des données ou des structures géométriques complexes, ou encore améliorer le routage dans les réseaux informatiques ou téléphoniques. Le PageRank de Google en est un exemple typique, voir le [chapitre 5](#).

L'APPRENTISSAGE AUTOMATIQUE

Les *algorithmes d'apprentissage* ont été mentionnés à plusieurs reprises dans les chapitres précédents. Une des grandes tendances actuelles est d'utiliser des réseaux souvent présentés sous la forme de réseaux de neurones profonds, en utilisant une analogie neurologique qu'il ne faut pas prendre au pied de la lettre, le fonctionnement

de nos propres réseaux de neurones restant largement inconnu. Les réseaux profonds s'appuient sur une structure en couche assez simple, alternant techniquement des couches linéaires et non-linéaires où le comportement de chaque couche est réglé par une matrice de coefficients jouant un rôle analogue à celui des poids synaptiques dans les neurones de notre cerveau. On entraîne ces algorithmes par *apprentissage supervisé* en les faisant calculer sur de très nombreuses données annotées, chaque nouvelle expérience ajustant les coefficients des couches par une rétropropagation simple des écarts avec la réponse correcte. Le nombre de coefficients peut être très élevé, se comptant parfois en millions.

Il y a vingt-cinq ans, ces algorithmes étaient capables de reconnaître l'écriture manuscrite des chiffres. Maintenant, ils reconnaissent la parole, traduisent les langues, gagnent au go (en conjonction avec des algorithmes probabilistes d'exploration des configurations), analysent les images médicales ou les scènes pour conduire les voitures, etc. Un avantage technique de ces algorithmes est qu'ils se parallélisent très facilement car leurs principales opérations sont des multiplications de matrices. Leur essor est dû en bonne partie à l'utilisation systématique des GPU (*graphical processing units*), experts dans ces opérations.

En dehors des réseaux profonds, il existe d'autres techniques comme celle des *support vector machines* (SVM) ou comme les techniques dites *bayésiennes* en l'honneur de la formule de calcul des probabilités conditionnelles introduite au XVIII^e siècle par le révérend Bayes. Je n'en parlerai pas ici.

Pour finir, il faut noter que les algorithmes d'apprentissage sont plutôt de nature analogique. À l'inverse des algorithmes classiques, on ne peut plus caractériser exactement la qualité des résultats, ce qui fait que la notion de bug que nous avons longuement étudiée au [chapitre 9](#) n'a plus vraiment de pertinence. Comment vérifier ces algorithmes et qualifier leur validité de ces algorithmes reste un problème ouvert, qui n'a peut-être pas de solution générale. En revanche, on pourra peut-être les utiliser pour mieux chasser les bugs dans les programmes à travers des dépouillements des codes sources ou des traces d'exécution.

LE CHIFFREMENT,
CLEF DE LA SÉCURITÉ INFORMATIQUE

Voici un exemple d'algorithme de chiffrement utile pour la sécurité informatique. Le chiffrement RSA est le premier chiffrement dit *asymétrique*, car les interlocuteurs ne possèdent pas de clef commune. Il a été inventé par Ronald Rivest, Adi Shamir et Leonard Adleman, tous prix Turing 2002, en implémentant efficacement pour la première fois une idée très novatrice de chiffrement à deux clefs asymétriques due à Whitfield Diffie et Martin Hellman, tous deux prix Turing 2015. Ayant créé une révolution dans le domaine, RSA est intensivement utilisé sur les cartes à puce et sur Internet. Son principe est que chaque interlocuteur fabrique deux clefs inverses, l'une publique et l'autre privée. La clef privée est constituée de deux très grands nombres premiers p et q , et la clef publique est leur produit pq . Tout message de Bob à Alice va passer par quatre étapes de chiffrement-déchiffrement : pour l'envoi, il va d'abord être chiffré par la clef privée de Bob, puis par la clef publique d'Alice ; à l'arrivée, il va être déchiffré par la clef privée d'Alice, puis par la clef publique de Bob. Le chemin inverse peut être pris par un message d'Alice à Bob, car les clefs sont utilisables dans les deux sens. On conjecture qu'il faut connaître les deux clefs privées pour décoder les messages, et qu'aucun algorithme sur ordinateur classique ne peut les recalculer en temps raisonnable dès que les deux clefs publiques sont trop grandes. Mais on sait aussi qu'un ordinateur quantique pourra casser RSA, s'il existe un jour.

D'autres chiffrements plus efficaces en taille de clef sont fondés sur le *logarithme discret*. Ils sont hélas tout aussi vulnérables aux ordinateurs quantiques. Mais on sait aussi faire des chiffrements qui leur résistent. C'est le domaine de la *cryptographie postquantique*, voir Wikipédia par exemple.

12.2. Résoudre un Sudoku avec SAT

Comme promis au [chapitre 10](#), voici comment résoudre un Sudoku avec SAT. Un Sudoku est une grille 9×9 où chaque case peut recevoir un chiffre de 1 à 9. Introduisons donc $9 \times 9 \times 9 = 729$ variables booléennes nommées $S(i,j,v)$ où i est le numéro de ligne, j le numéro de colonne, et v une valeur entre 1 et 9. Le problème s'exprime par une grande conjonction (un *et*) de disjonctions (des *ou*), chaque disjonction correspondant à une contrainte. C'est ce qu'on appelle en mathématiques une *forme normale conjonctive*, la meilleure nourriture pour SAT. Voici les éléments de la grande conjonction qui définissent la solution, avec les notations classiques $\wedge$ pour *et*, $\vee$ pour *ou*, et $\neg$ pour *non* :

- Pour chaque case initialisée dans l'énoncé du problème, la variable correspondante est ajoutée dans la conjonction. Par exemple, si la case (3,7) a la valeur 8 dans l'énoncé, on met dans la conjonction la variable $S(3,7,8)$. Pas cher, et la seule chose qui bouge entre deux Sudokus.
- Pour chaque case (i,j) , on ajoute à la conjonction le fait que la case doit avoir au moins une valeur dans la solution, ce qui s'écrit bêtement comme la disjonction de 9 variables $s(i,j,1) \vee s(i,j,2) \dots \vee s(i,j,9)$. On a donc 81 disjonctions de ce type. Facile.
- Pour chaque case (i,j) , on ajoute le fait que cette case ne peut pas avoir deux valeurs en même temps. Cela s'exprime pour toute paire de valeurs distinctes v et w de 1 à 9 par $\neg(S(i,j,v) \wedge S(i,j,w))$, ce qu'on préfère écrire de façon équivalente $(\neg S(i,j,v) \vee \neg S(i,j,w))$. Comme il suffit par symétrie de prendre $v < w$, on obtient

36 disjonctions pour chaque case, soit $81 \times 36 = 2\,916$ disjonctions au total. Une paille.

- Deux cases sont dites incompatibles si elles ne peuvent pas contenir le même nombre selon les règles du Sudoku. Pour chaque case, il y a 24 cases incompatibles, 8 dans la ligne, 8 dans la colonne, et 8 dans le carré 3×3 qui contient la case (avec des redondances, mais ce n'est pas grave). Prenons une paire de cases incompatibles (i,j) et (i',j') . Il faut exprimer que pour chaque valeur v , on ne peut pas avoir en même temps $S(i,j,v)$ et $S(i',j',v)$, ce qui s'écrit comme la disjonction $(\neg S(i,j,v) \vee \neg S(i',j',v))$. Comme on peut choisir par symétrie (i',j') après (i,j) dans l'ordre du dictionnaire (ou sur la grille l'ordre de la lecture), on peut ne considérer que $(81 \times 24)/2 = 972$ incompatibilités. À 9 valeurs et donc 9 disjonctions par paire de cases incompatibles, on rajoute 8 748 disjonctions dans la formule.

La formule finale, facile à engendrer par programme, comporte alors 11 845 disjonctions plus le nombre de cases remplies dans l'énoncé initial, la seule chose qui varie entre deux Sudokus.

On donne alors cette formule à n'importe quel bon solveur SAT, et ZAP, la solution s'affiche instantanément. Vous voulez vérifier qu'elle est unique ? Alors ajoutez à la conjonction pour toute case (i,j) qui était vide dans la donnée du Sudoku la formule $\neg S(i,j,v)$ où v est la valeur qu'avait trouvée le solveur. C'est ce qu'il faut pour interdire de retrouver la même solution. S'il y a une autre solution, ZAP, elle arrive. Sinon, le solveur répond UNSAT. Rigolo, non ?

12.3. De la séquentialité au parallélisme et à la distribution

Historiquement, l'informatique s'est développée dans un cadre *séquentiel*, c'est-à-dire où les opérations de base sont exécutées en séquence, une à la fois. Les calculs automatiques étaient donc faits par extension directe du fonctionnement du mathématicien opérant avec un seul crayon. Pour gagner de l'efficacité, on a vite cherché à *paralléliser* les calculs, c'est-à-dire à les faire exécuter sur plusieurs machines calculant simultanément.

LE PARALLÉLISME DANS LES ORDINATEURS

Les premiers algorithmes parallèles provenaient de l'*analyse numérique*, branche des mathématiques qui s'intéresse à la solution par calcul numérique à partir d'équations complexes. Ainsi sont nés les superordinateurs, très puissants et très chers, dont le plus célèbre reste l'emblématique Cray One. Dans ces superordinateurs, on peut paralléliser les calculs de deux façons principales. La première est la *vectorisation*, qui consiste à exécuter soit la même opération sur beaucoup de données en parallèle, ce qui est très bien adapté à l'opération fondamentale et hyperfréquente qu'est la multiplication de matrices. La seconde façon de faire est le *pipe-line*, qui fait passer les données dans une suite d'opérations différentes les unes des autres mais agissant en parallèle, comme pour les chaînes de montage de voitures. Le pipe-line est utilisé de façon intensive dans tous les microprocesseurs et extrêmement utile pour le

traitement du signal en matériel et logiciel ; nos télévisions aussi l'utilisent beaucoup. Les microprocesseurs utilisent bien d'autres méthodes de parallélisation, comme la *spéculation* que j'ai décrite au [chapitre 3](#).

Bien sûr, ces méthodes générales de parallélisation peuvent être couplées et augmentées par d'autres techniques plus spécifiques, en particulier pour augmenter la vitesse des accès mémoires, qui est souvent le point le plus critique. Tout est difficile dans un superordinateur, qui peut consommer des dizaines de mégawatts. Et l'objectif actuel est d'atteindre l'exaflop, soit un milliard de milliards d'opérations flottantes par seconde. C'est crucial pour les grandes simulations en météorologie, climatologie, physique nucléaire, astronomie, etc. Pour le calcul numérique, on cherche souvent à cacher le parallélisme à l'utilisateur. C'est le rôle des *compilateurs parallélisants* d'extraire un parallélisme maximum depuis le programme source. C'est souvent une opération naturelle car les programmes numériques utilisent essentiellement les opérations matricielles, parallélisables par nature. Mais il est toujours difficile de générer un code machine à la fois efficace et correct. La vérification formelle est ici très utile.

Enfin les techniques matérielles des superordinateurs se sont récemment démocratisées avec l'invasion des circuits GPU (*graphics processing unit*), qu'on trouve désormais sous une forme ou une autre dans tous nos ordinateurs et téléphones. Ils ont été créés principalement pour les jeux vidéo, mais ils trouvent maintenant beaucoup d'autres applications, dont l'apprentissage automatique.

LE PARALLÉLISME DANS LES SYSTÈMES D'EXPLOITATION

Plus récemment, le parallélisme s'est introduit dans nos ordinateurs personnels, d'abord de façon timide puis de façon massive. Il faut reprendre un peu l'histoire, où matériel, logiciel et algorithmique sont assez emmêlés. Les premières machines ne disposaient que d'un système d'exploitation simple permettant d'exécuter une seule tâche à la fois. Ils ont ensuite permis le *time-sharing*, c'est-à-dire l'alternance temporelle entre différents programmes pouvant appartenir à différents utilisateurs. Puis, dans les années 1970, le très élégant et efficace système d'exploitation Unix des

Bell Laboratories a envahi les miniordinateurs, en promouvant astucieusement le pipe-line au niveau du logiciel. Pour les micro-ordinateurs, les systèmes d'exploitation initiaux comme DOS de Microsoft et Mac OS 9 d'Apple ne permettaient d'exécuter qu'une tâche à la fois. Mais, avec Unix, puis Windows et Mac OS X, leurs systèmes d'exploitation ont rejoint ceux des gros ou des miniordinateurs. Enfin, dans les années 1990, les langages ont intégré la programmation par *threads* (fils d'exécution), qui permet le parallélisme à l'intérieur d'un programme.

En ce qui concerne les systèmes d'exploitation, on voit maintenant une grande unification des systèmes et des modes de programmation, en particulier à partir d'Unix. Né dans les années 1970, Unix a servi de base à Linux de Linus Torvalds, mais aussi à Mac OS X et iOS d'Apple, ainsi qu'à Android de Google. Capable de faire fonctionner efficacement les ordinateurs multiprocesseurs, Linux est utilisé depuis les téléphones jusqu'aux plus gros superordinateurs, en passant par les PC et beaucoup d'objets informatisés. Il est aussi connu comme le système le plus robuste à l'heure actuelle, et l'un des plus performants.

L'INFORMATIQUE DISTRIBUÉE

Avec l'arrivée massive des réseaux, une autre forme d'informatique parallèle est devenue indispensable : l'informatique *distribuée*, où les calculs se font par coopération d'ordinateurs situés n'importe où dans le monde. Les problèmes peuvent y être très différents de ceux des superordinateurs. Par exemple, une base de données peut être distribuée à travers le monde en plusieurs copies qu'il faut tenir à jour de façon cohérente. Les bases de données peuvent aussi conserver l'heure précise à laquelle une information a été entrée ou mise à jour n'importe où dans le monde, et répondre de façon cohérente à des questions sur cette information à partir de n'importe quelle copie de la base. Ce sont des problèmes difficiles et typiques de l'algorithmique distribuée. Le Web est aussi un grand domaine pour l'informatique distribuée, à cause du fait que clients et serveurs sont presque toujours géographiquement éloignés avec un fort risque de perte de réseau ou de paquets. Il y a bien d'autres problèmes d'algorithmique distribuée, qu'il est hors de question de décrire ici.

Il faut rappeler que le domaine de l'algorithmique distribuée possède un grand seigneur : Leslie Lamport, prix Turing 2013. Ce chercheur a développé et prouvé les algorithmes distribués les plus fondamentaux utilisés actuellement, combinant découvertes algorithmiques et vérification formelle en machine avec son vérifieur de logique temporelle TLA+ (*temporal logic of actions*). Il a aussi développé le logiciel LaTeX en étendant et simplifiant le logiciel TeX de Donald Knuth. LaTeX est devenu le logiciel de référence en publication scientifique.

ATTENTION À L'ALGORITHMIQUE ET À LA PROGRAMMATION PARALLÈLES

Pour ce qui est de l'invention des algorithmes et l'écriture des programmes, il faut toujours rappeler que le calcul parallèle est beaucoup plus difficile que le calcul séquentiel classique, et que les bugs peuvent y être particulièrement sournois. En particulier, dans le cas de la programmation distribuée, il faut absolument éviter qu'un amateur remplace un algorithme distribué éprouvé et prouvé par son invention du jour, qui a toutes les chances d'être pleine de bugs. Il existe de bonnes bibliothèques d'algorithmes et de bons livres sur la programmation parallèle et distribuée, n'en sortons pas !

12.4. Les compilateurs et le bootstrapping

Développer des langages de plus haut niveau introduit une difficulté majeure : un programme de haut niveau ne peut pas être directement exécuté par une machine. Il faut au préalable le traduire en langage machine à l'aide d'un compilateur. C'est précisément ici que le fait de pouvoir calculer sur les programmes intervient de façon cruciale. Il nous faut un minimum de notation pour expliquer comment on procède. Pour éviter toute confusion, je décrirai toutes les étapes en détail. Que le lecteur me pardonne s'il termine avec un petit mal de tête. Il s'agit ici de façons de faire absolument fondamentales en informatique, auxquelles il faut simplement s'habituer. Une fois que l'on est entraîné, elles procurent un grand plaisir.

Supposons qu'on écrive un programme P dans un langage de haut niveau L , et qu'on souhaite l'exécuter pour des données D sur une machine M , dont le langage machine propre sera aussi appelé M . Comment exécuter P avec D sur M , alors qu'il n'est pas écrit en M ?

La meilleure méthode est d'écrire *une fois pour toutes* en langage machine M un premier compilateur K , dont la seule fonction est de pouvoir traduire tous les programmes écrits en langage L dans le langage machine M ¹. En appliquant K à P , on obtient alors un résultat $K(P)$ écrit en M et donc directement exécutable par M . J'écrirai $K : L \rightarrow M$ pour « K traduit L en M ». Le langage L est appelé le *langage source* de K , le langage M son *langage cible*.

Si le compilateur K est correct, l'exécution de $K(P)$ pour D sur M calcule concrètement le résultat défini abstraitement par P pour D . Il suffit donc de réaliser cette exécution pour obtenir le résultat cherché. La conjonction de la machine M et du compilateur K nous permet de programmer non plus la machine M mais une « machine virtuelle L » de plus haut niveau qui comprendrait directement le langage L . L'informatique n'aurait jamais pu décoller sans cette abstraction majeure, qui a à la fois augmenté les facultés de programmation et libéré les fabricants de circuits : les microprocesseurs modernes se sont complexifiés à un point tel qu'il est devenu quasi impossible à un humain d'écrire un programme dans leur langage machine !

BOOTSTRAPPER LE COMPILATEUR

Mais la méthode que nous venons d'expliquer n'est pas suffisante en pratique, car elle est difficilement compatible avec la contrainte permanente d'efficacité en temps et en mémoire de calcul. Le compilateur K étant prévu pour compiler beaucoup de programmes P écrits en L par les utilisateurs, il doit respecter une double contrainte : être lui-même efficace, et produire des codes machine $K(P)$ efficaces. Mais construire directement un tel objet en langage machine M serait quasiment impossible à cause de la pauvreté expressive de M et de la complexité des algorithmes de compilation et d'optimisation.

Heureusement, on peut mettre en place un petit miracle nommé *bootstrapping*, mot anglais qui signifie « s'élever en l'air en tirant sur les lanières de ses bottes », comme déjà expliqué au [chapitre 2](#). C'est impossible avec la matière, mais possible avec l'information. Voici comment procéder :

- Comme expliqué ci-dessus, on écrit d'abord en langage machine M un premier compilateur K_M traduisant L en M , ce qu'on écrit $K_M : L \rightarrow M$. Puisque programmer avec M est difficile, on rend K_M le plus simple possible, en lui demandant ni d'être efficace, ni de produire un résultat efficace.
- On écrit en parallèle en langage de haut niveau L un compilateur efficace $K_L : L \rightarrow M$ conçu pour traduire un programme écrit en L en un programme efficace

écrit en M . Ce qui était difficile à écrire en M devient bien plus facile en profitant de la puissance expressive de L .

- Notons que les indices de K_M et K_L dénotent les langages dans lesquels ces compilateurs sont écrits ; nous garderons cette convention dans la suite. Le reste se fait en laissant calculer l'ordinateur :
 - Sur la machine M , on donne le programme K_L en entrée à K_M , qui peut être exécuté directement puisque écrit en M .
 - Appelons $K'_M = K_M(K_L)$ le résultat de ce calcul. Ce K'_M n'est rien d'autre qu'un programme écrit en M (langage cible de K_M) dont la fonction reste la même que celle de K_L puisque K_M n'est rien qu'un compilateur. Donc $K'_M : L \rightarrow M$ est un compilateur de L dans M tout comme K_M , mais qui produit cette fois un code M efficace – pile ce que ne savait pas faire K_M .
 - Cependant, même s'il est supérieur à K_M , notre nouveau compilateur K'_M n'est pas encore le programme qu'il nous faut, car il n'est pas lui-même assez efficace puisque K_M ne génère pas du code efficace.
 - Il suffit alors de refaire compiler K_L par K'_M . Le résultat $K''_M = K'_M(K_L) : L \rightarrow M$ a cette fois toutes les bonnes propriétés : d'abord, il traduit L en M , comme K_L dont il est une traduction. Ensuite, partant d'un programme P écrit en L , il génère un code efficace, comme K_L . Enfin, il est lui-même efficace puisqu'il est lui-même un résultat de K'_M qui génère du code efficace.

Ouf ! Voici donc comment les informaticiens se transforment en chats fondamentalement paresseux. Tout ce qui peut être fait par une machine plutôt que par eux est bon à prendre. Remarquons qu'à la fin il ne reste plus la moindre trace de K_M . Dur de s'y retrouver pour un archéologue du futur !

Bien sûr, la création des compilateurs efficaces modernes est bien plus complexe que ce simple schéma, en particulier parce qu'un même compilateur doit prendre plusieurs langages machine pour cibles. De plus, les mêmes principes de *bootstrapping* sont appliqués à de nombreuses échelles. Par exemple, en construisant des tours gigognes, on peut élever petit à petit le niveau de la programmation : on traduit souvent un langage de haut niveau L en deux étapes, en le traduisant d'abord vers le fameux langage C , autrefois vu comme de haut niveau mais maintenant plutôt vu comme un excellent intermédiaire de niveau moyen, puis en utilisant un des excellents compilateurs C pour finir le travail.

LE BOOTSTRAPPING EST PARTOUT

Terminons en mentionnant que le bootstrapping est utilisé en bien d'autres endroits, notamment pour le démarrage progressif des systèmes d'exploitation (Linux, Windows, Mac OS X, etc.) qui servent d'écran entre nous et les machines pour les rendre compréhensibles à l'usage, gérer leurs ressources, et les interfacer avec nous, les réseaux et les périphériques de tout type. C'est pour cette raison qu'on dit qu'on *boote* les ordinateurs.

12.5. Mais pourquoi autant de langages ?

J'ai discuté cette question dans la première séance de mon cours 2015-2016 au Collège de France², et je n'en reprendrai que quelques points clefs ici. Au [chapitre 3](#), nous avons dit qu'il s'était créé des centaines, voire des milliers de langages de programmation depuis la naissance de l'informatique, avec quand même un nombre limité de grandes classes de langages. Cela ressemble tout à fait à ce qui s'est passé pour les langues parlées de l'humanité. Il y a plusieurs raisons à ce nombre, outre bien sûr l'ego des créateurs de langages qui les oblige tous à trouver du nouveau et du différent.

La première raison a été l'augmentation continue des possibilités de calcul, qui a engendré une expansion permanente du champ de travail de l'informatique vers de nouveaux types d'applications de plus en plus variées et complexes, avec en corollaire une augmentation continue de la taille des programmes. Les langages de programmation initiaux, nés avec l'idée d'écrire des programmes pas très gros pour des algorithmes relativement simples, ont vite été débordés. Avec une grande créativité, les chercheurs ont alors mis en place de nouvelles idées et de nouveaux principes linguistiques pour s'adapter aux styles variés des nouvelles applications et permettre la construction modulaire de très gros programmes. La seconde raison est que beaucoup de dialectes plus ou moins incompatibles des langages principaux se sont créés, car il n'était pas simple de porter les langages et leurs compilateurs d'une machine à une autre, et bien sûr car chaque chercheur tient à poser sa petite pierre sur le chemin. Cette profusion de dialectes a conduit soit à des positions devenues irréconciliables, soit à une normalisation convenable dans les bons cas, sachant que

tout langage évolue à travers de multiples versions successives en fonction des retours utilisateurs et des nouveaux besoins. La dernière raison est que construire un langage et son outillage est devenu quelque chose de vraiment difficile, et qu'il vaut mieux de temps en temps repartir de zéro plutôt que de modifier par couches successives un langage et des compilateurs devenus trop lourds. En bref, les langages sont des objets vivants !

Il est de plus en plus clair qu'il ne faut pas espérer d'unification de toutes les idées qui sont disponibles actuellement, car une unification par le bas du type « anglais scientifique rudimentaire » qu'utilisent les chercheurs pour communiquer entre eux n'aurait pas de sens en informatique. Il faut aussi noter que la situation est bien différente de celle des mathématiques, où un langage universel existe vraiment, au moins dans le domaine des notations de base. Mais, en maths, les hommes sont à la manœuvre et pas les machines. D'ailleurs, maintenant qu'on sait aussi mettre en machine des démonstrations 100 % rigoureuses de grands théorèmes, on voit que les langages informatiques qu'il faut utiliser pour ce faire sont parmi les plus riches et les plus complexes jamais construits ! C'est le cas du langage Gallina, langage du système de vérification formelle Coq développé en France pour la vérification de programmes et de théorèmes mathématiques. Comme nous l'avons vu au [chapitre 10](#), Coq a été construit comme une implémentation du calcul des constructions inductives de Gérard Huet, Thierry Coquand et Christine Paulin, une merveille de logique. Il a permis de montrer en machine de grands théorèmes mathématiques, et aussi de conduire des preuves très importantes en informatique, comme celle du compilateur CompCert de Xavier Leroy pour un sous-ensemble du langage C. Coq et Gallina ont été doublement récompensés par l'ACM (Association for Computing Machinery), la principale association professionnelle mondiale en informatique, avec l'ACM SIGPLAN Programming Languages Software Award en 2013 et l'ACM System Award en 2014. La simplicité apparente du langage mathématique est donc bien trompeuse, dès qu'on le prend vraiment au sérieux !

12.6. L'évolution des langages

Voici sommairement comment l'évolution des besoins a conduit à de nouveaux principes et à de nouvelles familles de langages de programmation. Commençons par les langages généraux, non spécialisés vers une tâche particulière, en ignorant bien sûr des langages d'assemblage propres à chaque ordinateur, que personne n'utilise plus.

D'ADA LOVELACE AU λ -CALCUL DE CHURCH

L'histoire de la programmation a probablement commencé avec Ada Lovelace, fille de Lord Byron, mathématicienne et collaboratrice de Charles Babbage lorsqu'il essayait de construire son analyseur différentiel au milieu du XIX^e siècle. Elle a écrit le premier vrai programme pour cette machine qui n'existerait jamais, comprenant déjà la différence entre matériel et logiciel et la difficulté de programmer le logiciel. Elle a aussi tout de suite compris que le calcul automatique ne concernerait pas que les nombres mais pourrait s'étendre aux textes et à toutes sortes de données symboliques pour faire des calculs ne ressemblant pas à ceux des mathématiques.

Le tout premier langage pertinent pour nous est le λ -calcul d'Alonzo Church, créé en 1936 pour fonder les mathématiques. Il a échoué au début dans son objectif initial, trop ambitieux pour l'époque, mais il a été immédiatement montré équivalent à la machine de Turing. Ainsi, la théorie de la calculabilité a unifié dès le début l'aspect machiniste et l'aspect linguistique. Le λ -calcul continue à jouer un rôle fondamental en informatique : c'est un outil de base en logique car il constitue la bonne syntaxe

pour les preuves et leur développement, en particulier à travers la correspondance dite de Curry-Howard sur laquelle se fonde Coq et qui permet maintenant de faire revivre en partie le vieux rêve de la fondation formelle des mathématiques. Il sert directement de fondement aux langages fonctionnels et aux assistants interactifs de vérification formelle comme Coq, HOL de Michael Gordon et Isabelle de Larry Paulson en Angleterre. J'ai consacré la première partie de ma carrière à ses propriétés syntaxiques et sémantiques ; comme dit Georges Brassens, jamais de ma vie je ne l'oublierai.

UNE BRÈVE HISTOIRE DES LANGAGES DE PROGRAMMATION

Le premier vrai langage informatique, FORTRAN (FORMula TRANslator), a été créé par John Backus en 1954 pour le calcul scientifique. Il réalisait déjà une bonne abstraction des langages internes des machines, avec des variables symboliques, des expressions arithmétiques calquées sur celles des mathématiques, des sous-programmes, etc. Mais il reposait sur un modèle de mémoire très limité et la programmation y était d'une notable insécurité. Il est toujours là, bien sûr, avec beaucoup de changements incorporant des avancées ultérieures apportées par d'autres langages.

Ensuite vint LISP, un langage très différent créé par John McCarthy en 1957 pour les besoins de l'intelligence artificielle naissante. LISP, profondément original, différait de FORTRAN sur quasiment tous les aspects, mettant en avant deux idées nouvelles : celle de fonction en tant qu'objet manipulable comme les autres et celle de liste en tant que structure de donnée primordiale. LISP avait aussi l'avantage d'être interactif, c'est-à-dire que les programmes étaient exécutés immédiatement dès qu'ils étaient tapés. Il a eu énormément de dialectes et une influence considérable sur la recherche ultérieure. Puis Scheme (Jay Sussman et Guy Steele, années 1970) a gardé son style syntaxique tout en corrigeant quelques erreurs de design³. Dans un cadre encore différent, COBOL a été créé en 1959 par un consortium d'industries et d'agences gouvernementales pour la gestion, en s'inspirant des recherches pionnières

de Grace Hopper. C'était l'époque où les ordinateurs commençaient à supplanter les machines électromécaniques qui avaient fait le succès d'IBM.

Trouvant FORTRAN, COBOL et leurs successeurs pas très rigoureux, des scientifiques à tendance logicienne ont créé ALGOL 60 (ALGOritmic Oriented Language), en 1960 bien sûr. Ils y introduisaient beaucoup des notions qui ont fondé la programmation moderne : bonne gestion des variables, bonne vision des fonctions fondées sur la logique formelle, mécanisme d'appel des fonctions très propre, etc. Selon Tony Hoare : « Voici un langage très en avance sur son temps ; il n'a pas seulement été une amélioration de ses prédécesseurs mais aussi une amélioration de presque tous ses successeurs. » Pourtant, ALGOL 60 n'a jamais vraiment existé en pratique, en partie à cause de son absence de mécanismes d'entrée-sortie, et son successeur direct ALGOL 68 a été un fiasco. Mais il a effectivement fortement inspiré ses successeurs, dont Simula 67, le premier langage de simulation créé par Ole-Johan Dahl et Kristen Nygaard en 1967, et Pascal de Niklaus Wirth en 1970, qui a pris une grande place dans l'enseignement naissant de l'informatique. Le langage C, assez similaire à Pascal mais très simplifié et destiné tout particulièrement aux miniordinateurs naissants, a été créé en 1972 aux Bell Laboratories par Dennis Ritchie et Ken Thompson pour programmer le système d'exploitation Unix.

Très pragmatique, proche des machines mais n'ignorant pas la théorie, C allait rapidement envahir le monde industriel et pénétrer aussi le monde académique aux côtés des langages de naissance plus scientifique. Pour la première fois, on cherchait à faire un langage général simple, portable et puissant, mais capable de travailler sur de petites machines. C reste un grand standard, même s'il est de moins en moins utilisé par les hommes à cause de son niveau un peu trop bas et du peu de protection contre les erreurs qu'il procure. Il remplace désormais l'assembleur sur toutes les machines.

C, dont le type de syntaxe s'est rapidement imposé pour des raisons pas toujours claires, a servi de base à beaucoup d'autres langages développés ensuite, comme C++ (Bjarne Stroustrup, années 1980) et Java (James Gosling et Patrick Naughton, 1995), qui l'étendent par l'idée de *programmation objet*. Cette idée avait été introduite initialement par Alan Kay en 1973 pour Smalltalk, langage construit comme LISP pour l'intelligence artificielle mais ajoutant des concepts originaux issus du langage Simula 67 dédié, lui, à la simulation de processus physiques. Smalltalk s'est avéré excellent pour la construction d'interfaces homme-machine complexes, et a

définitivement installé la programmation objet dans ce domaine. Ses idées mêlées à celles de C ont aussi conduit au langage Objective-C utilisé pour les mythiques machines NeXT puis le Macintosh, créés sous la férule de Steve Jobs et l'action de Jean-Marie Hullot. Pour sa part, le succès de Java repose sur le fait qu'il résolvait un nouveau besoin apparu avec l'expansion du Web dans les années 1990 : le téléchargement dynamique de programmes, rendu possible par l'introduction de *machines virtuelles* simples calculant sur un code à demi compilé.

LA PROGRESSION DES LANGAGES FONCTIONNELS

De leur côté, les langages fonctionnels inspirés par LISP et certains aspects d'ALGOL continuaient leur chemin. Un pas décisif a été franchi par Robin Milner dans les années 1980, avec le langage ML et son système de types révolutionnaire qui allait marquer toute l'informatique moderne. Utilisant LISP dans son assistant de vérification formelle LCF (Logic for Computable Functions) pour la démonstration de correction des programmes, Milner était gêné de constater que rien n'y interdisait l'introduction de théorème faux à cause de l'absence de typage des objets et des constructions du langage : on pouvait y mélanger librement des choux et des carottes, ou encore des théorèmes justes et des théorèmes faux. Dans un langage comme Pascal, il existait bien des systèmes de typage assurant qu'on ne mélangeait pas les choux et les carottes. Mais ces systèmes ne produisaient qu'une faible sécurité et n'étaient pas capables d'avalier le λ -calcul. Milner a résolu son problème en introduisant un système de types merveilleusement élégant pour interdire tout mélange illégal de données et fonctions, qui donnait de plus la possibilité d'utiliser des types polymorphes comme celui d'une liste d'objets de n'importe quel type au lieu de listes séparées pour les entiers et les nombres flottants par exemple. Tous les systèmes de typage modernes et beaucoup d'études théoriques en sont issus. ML a été rapidement suivi par Standard ML, peu utilisé, puis par Caml de Xavier Leroy en France. Caml a ensuite absorbé une partie de la programmation objet en devenant OCaml, un langage standard dans la recherche, mais aussi utilisé dans l'industrie pour les applications où l'absence de bugs est critique. De leur côté, les Anglais ont développé le langage Haskell, qui a introduit d'autres concepts nouveaux et est très

utilisé dans certains domaines. Scala, développé plus récemment par Martin Odersky depuis 2001, est une unification impressionnante de la programmation fonctionnelle, de la programmation objet, et de la programmation parallèle ; il est compatible avec Java, ce qui fait aussi son succès.

Dans l'industrie, les langages fonctionnels étaient souvent regardés comme « intéressants, mais pour les chercheurs ». La situation change rapidement : par exemple Microsoft a créé et utilise dans ses produits le langage fonctionnel F#, alors qu'Apple a créé son langage fonctionnel Swift plus proche de C. Ces deux sociétés ont mis leurs langages et outils de développement à la disposition de la communauté. Il est maintenant largement admis que la programmation fonctionnelle est bien plus compacte et bien plus sûre que la programmation impérative classique et que ses variantes orientées objet. L'impact des langages fonctionnels ira certainement en grandissant.

LA MODULARITÉ

On pense généralement que le but principal des langages de programmation est d'aider à programmer les machines de façon efficace. C'est vrai, mais il y a deux autres contraintes, de nature très différente et beaucoup plus humaine. D'abord, il est indispensable que les langages parlent non seulement aux machines mais aux hommes : les programmes étant des objets susceptibles de modifications constantes, il est tout aussi important qu'ils puissent être relus par d'autres que leurs auteurs. La qualité de la syntaxe et de la sémantique qui définit le sens des programmes est ici essentielle (la sémantique mathématique des langages est un grand sujet d'informatique mathématique qui a occupé toute ma carrière, mais je ne vais pas en parler davantage ici). Ensuite, un programme devient rapidement un très gros objet, qu'il faut savoir bien gérer et maintenir. Ceci exige une architecture solide de la programmation et des bons outils de génie logiciel s'appuyant directement sur la sémantique du langage utilisé.

Or construire une architecture pour les hommes est très différent d'écrire des instructions pour les machines. Les meilleurs langages sont donc organisés en double couche : la couche algorithmique pour les machines et la couche de *modularité* pour

les hommes. Il m'est impossible de décrire ici cette couche, techniquement complexe. Mais je soulignerai que de bonnes solutions existent assez indépendamment de la couche algorithmique. On les trouve par exemple dans les langages fonctionnels comme OCaml, Haskell et Scala. Les langages objets ont aussi la prétention de résoudre ces problèmes, mais l'expérience montre que leurs solutions sont partielles et compliquées. Il reste beaucoup de travail à faire pour unifier réellement programmation fonctionnelle, programmation objet et programmation modulaire. OCaml et Scala se sont aventurés sur ces terrains, mais d'autres idées apparaîtront certainement dans l'avenir.

L'INTÉGRATION DU PARALLÉLISME ASYNCHRONE

Les activités parallèles sont dites asynchrones si elles n'ont pas de référence de temps commune. La question de l'intégration du parallélisme asynchrone dans les langages généraux a été posée assez tôt, mais n'est toujours pas vraiment résolue de façon réellement satisfaisante. Le premier langage vraiment parallèle a été CSP (Communicating Sequential Processes) de Tony Hoare en 1970. Il utilisait une communication par rendez-vous très propre et bien différente de la mémoire partagée utilisée habituellement pour le parallélisme « à la main ». CSP a été vite suivi du langage industriel OCCAM destiné à la machine Transputer, qui n'a eu qu'un faible succès. Le langage ADA, inspiré à la fois de Pascal et CSP, a été développé par Jean Ichbiah et son équipe vers 1980, sur commande du DoD (Department of Defense américain) pour l'informatique temps réel, dont l'importance allait grandissante dans des domaines comme l'avionique et le spatial. Il est intéressant de rappeler qu'il a été choisi après un concours comparant plusieurs propositions nommées par des couleurs – il s'appelait Green lors de sa soumission. ADA est très précis, a absorbé beaucoup de nouveautés déjà proposées ailleurs (modularité, exceptions, etc.), mais il est lourd et son parallélisme est vraiment difficile à contrôler. Il est resté relativement confidentiel. En parallèle, des travaux théoriques très importants ont été faits sur le parallélisme et la communication asynchrones, initiés encore une fois par Robin Milner. Ils ont cependant eu peu d'impact sur les langages, sauf en ce qui concerne les protocoles de sécurité informatique. Le parallélisme asynchrone est maintenant

disponible dans beaucoup de langages sous forme de *threads*, fils d'exécution déportables sur d'autres processeurs avec un mélange de mémoire locale et de mémoire partagée, qui demandent une interaction forte avec le système d'exploitation (ceux de Linux ont été développés par Xavier Leroy). Mais cette solution reste délicate à utiliser et ne peut pas être considérée comme pleinement satisfaisante. D'autres langages parallèles, inspirés des *acteurs* de Carl Hewitt qui échangeaient des messages comme par le courrier, ont pris des trajectoires différentes. Citons Erlang, langage fonctionnel très élégant développé dans l'entreprise de télécommunications Ericson dans les années 1990, et Scala, déjà mentionné ci-dessus.

GAMMA ET LA CHAM : PARLER SIMPLEMENT DU PARALLÉLISME ASYNCHRONE

Parler mathématiquement du parallélisme asynchrone n'est pas simple, et a donné lieu à beaucoup de formalismes théoriques plus ou moins riches. Je vais mentionner ici la Cham (Chemical Abstract Machine), machine abstraite parallèle que j'ai introduite en 1989 avec Gérard Boudol ([Berry & Boudol, 1992](#)). La Cham approfondit et généralise la belle idée de calcul chimique introduite par Jean-Pierre Banâtre et Daniel Le Métayer (Irisa Rennes) pour leur langage de programmation parallèle GAMMA. Elle est à la racine de la BioCham de François Fages, machine abstraite dont j'ai parlé au [chapitre 8](#) pour la modélisation des réactions biochimiques dans la cellule, et de bien d'autres méthodes de modélisation.

L'intuition est simple : alors que la machine de Turing et même l'ordinateur de von Neumann imitent le mathématicien avec son crayon et que les langages et formalismes parallèles classiques imitent plusieurs mathématiciens avec plusieurs crayons, GAMMA et la Cham imitent la chimie et son parallélisme de grain ultrafin. Montrons comment calculer les nombres premiers de façon chimique. Pour cela, il suffit de voir tous les nombres entiers supérieurs à 1 comme des molécules placées dans un grand chaudron, et de les laisser se rencontrer aléatoirement. La règle de réaction chimique est simple : tout nombre p a le droit d'absorber ses multiples, ce qu'on peut écrire $p, kp \rightarrow p$ en utilisant la multiplication et la notation standard des réactions chimiques. Alors, tout nombre non premier finira par disparaître, pourvu que

les rencontres soient nombreuses et aléatoires. Pouvez-vous imaginer un programme plus trivial que ça ? Six lettres, c'est dur à battre.

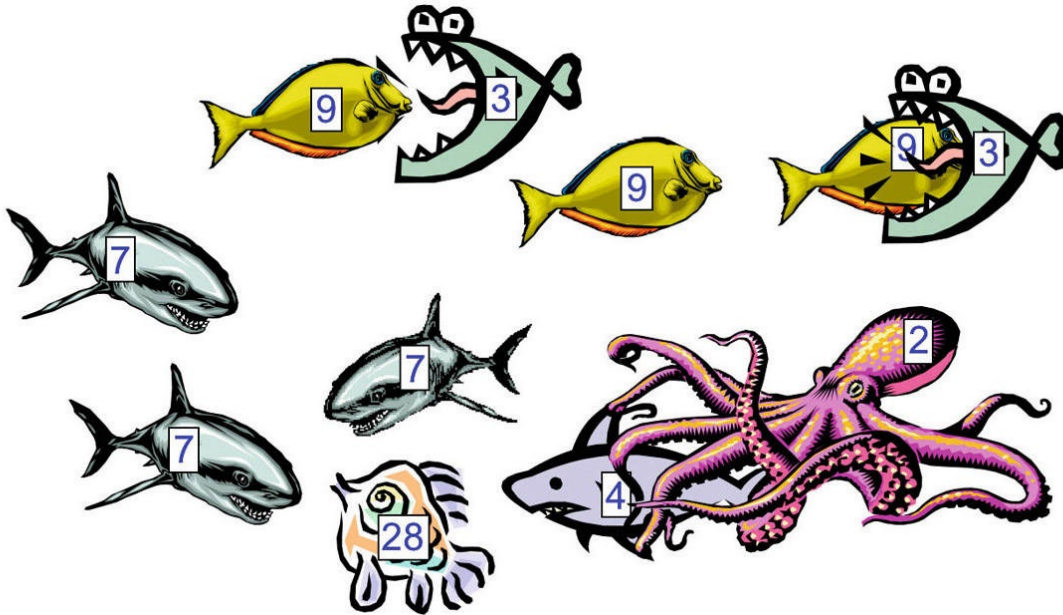


Figure 71. La Cham du crible de Darwin $p, kp \rightarrow p$ pour les nombres premiers.

Si l'on veut, on peut promouvoir la chimie en biologie et les nombres en dossards portés par des poissons carnivores, et laisser faire la sélection naturelle comme dans la figure 71. Les nombres premiers sont les survivants, d'où le nom de *crible de Darwin* que je donne à cette méthode. On peut même mettre autant de fois chaque nombre qu'on le souhaite ; le résultat est le même.

Le calcul chimique ou darwinien est bien plus simple que le classique crible d'Ératosthène, qui se limite à une taille finie et impose d'ordonner les nombres, de revenir au début chaque fois qu'un nombre a barré ses multiples, etc. Ces opérations ont à voir avec la main unique du mathématicien, mais rien avec les nombres premiers. Mais bien sûr, on y perd aussi avec le parallélisme : on ne peut jamais savoir quand le calcul est terminé. C'est la rançon du succès...

La Cham généralise cette idée chimique en simplifiant des calculs parallèles hiérarchiques bien plus riches, introduits par Robin Milner au début des années 1980. Il est intéressant de savoir que ce genre d'idée hypersimple mais productive n'est pas

forcément bien acceptée dans les milieux scientifiques : notre article de journal a été refusé par les deux relecteurs, en quelque sorte pour blasphème contre les théories dominantes, mais heureusement publié quand même par l'éditeur après nos réponses. Ainsi va la science...

LES LANGAGES SPÉCIFIQUES À DES DOMAINES D'APPLICATION

Tous ces langages visent la programmation générale. Mais ils ne sont pas nécessairement bien adaptés à toutes les tâches, et les chercheurs et industriels ont aussi développé de nombreux langages dédiés à des domaines particuliers, qu'on appelle en anglais *domain specific languages*, ou DSL en abrégé. Les plus répandus sont les *langages de script*, moins destinés à faire des calculs qu'à décrire des enchaînements complexes d'actions externes. L'un d'eux, le shell d'Unix (à travers ses variantes sh, ksh, bash, etc.), a révolutionné la façon d'utiliser les ordinateurs en remplaçant l'idée qu'une application devait prendre la forme d'un gros programme par celle qu'il était bien meilleur de la construire à partir d'un pipe-line de briques prenant la forme de petits programmes standard. De nombreux langages de script ont pris la suite : Tcl pour les interfaces graphiques, Perl, Python, et bien sûr JavaScript qui domine la programmation des clients Web. Python est par exemple utilisé pour conduire et cadencer de gros calculs faits en utilisant les très grandes bibliothèques de programmes disponibles, qui sont facilement appelables de l'extérieur. Les langages de script permettent une programmation rapide, mais qui reste propice aux bugs et difficile à vérifier. Un danger hélas très fréquent est de se ruer dessus pour aller vite au début, mais ensuite de faire tellement grandir ses programmes qu'ils risquent de devenir illisibles et invérifiables.

Pour illustrer la variété des applications des DSL, citons les langages HTML et CSS pour la description des documents Web, les langages de Maple et Mathematica pour les calculs mathématiques, ceux de Simulink et Modelica pour la simulation et le contrôle des systèmes physiques, Prolog et ses successeurs pour l'intelligence artificielle, Datalog qui est une version réduite de Prolog pour l'interrogation des bases de données, les langages de contraintes numériques utilisés en optimisation et

modélisation, Torch pour la création de réseaux de neurones profonds en apprentissage, etc. La liste s'agrandit tous les jours.

LES LANGAGES RÉACTIFS SYNCHRONES

Je ne peux pas terminer cette annexe sans mentionner le travail fait en France par plusieurs équipes sur les langages *réactifs synchrones*, destinés aux applications temps réel et plus généralement aux applications dont la fonction est d'interagir constamment avec leur environnement : pilotage d'avions, signalisation et conduite ferroviaires, conduite d'automobiles, systèmes de sécurité de processus industriels, robotique, mais aussi protocoles de télécommunication et conception de circuits électroniques. Les programmes écrits dans ces langages réagissent en permanence aux sollicitations de l'extérieur. Ils sont parallèles, mais au contraire du parallélisme asynchrone décrit plus haut, ils supposent que tous les acteurs partagent le même temps et peuvent agir de façon synchrone. C'est la bonne façon de faire pour les circuits électroniques synchrones (cadencés par des horloges) et pour les logiciels temps réel.

Les langages synchrones précurseurs étaient Esterel ([Berry, 2000](#)), conçu dans mon équipe à Sophia-Antipolis depuis 1982 à partir d'intuitions novatrices des automaticiens Jean-Paul Marmorat et Jean-Paul Rigault, Lustre, conçu par Paul Caspi et Nicolas Halbwachs à Grenoble, et Signal, d'Albert Benveniste et Paul Le Guernic à Rennes. Ces langages ont toujours été définis mathématiquement, et ce sont ces définitions qui ont directement conduit l'implémentation de leurs compilateurs et de leur outillage, y compris pour la vérification formelle. Esterel et Lustre sont désormais unifiés dans un langage appelé SCADE 6, qui est développé industriellement par la société Esterel Technologies, dont j'ai été directeur scientifique de 2001 à 2009. Le compilateur SCADE 6 (écrit lui-même en OCaml) a été certifié au plus haut niveau de la norme américaine DO-178C, qui régit l'avionique et est la plus stricte dans le domaine. SCADE 6 est utilisé par plus de 250 clients pour le développement des systèmes critiques en avionique, spatial, nucléaire, ferroviaire, ou autres. J'avais aussi développé un nouveau langage Esterel v7 chez Esterel Technologies pour rendre plus faciles et plus sûres la conception et la vérification des circuits électroniques. Il a eu

de remarquables succès techniques, mais il a malheureusement disparu avec ses clients lors de la grande crise de 2008.

Les idées d'Esterel et Lustre ont percolé dans de nombreux autres langages, comme Reactive C de Frédéric Boussinot, Reactive ML de Louis Mandel, et l'extension réactive synchrone SCL de C par Reinhard von Hanxleden et son équipe à l'université de Kiel (Allemagne). Esterel a aussi été repris et étendu dans HipHop, dédié à la difficile programmation de la gestion des événements et des erreurs sur le Web, pour laquelle JavaScript ne propose que des solutions rudimentaires. J'en ai déjà parlé au [chapitre 5](#).

Le lecteur intéressé pourra trouver tous les renseignements voulus dans mes cours 2012-2014 et 2017-2018 au Collège de France, tous disponibles sur le Web.

Glossaire¹

2D. En deux dimensions, donc sur un plan. Le papier est l'objet 2D le plus utilisé.

2G. Il était une fois, au xx^e siècle, un téléphone numérique sans fil dit de deuxième génération, qui servait encore à téléphoner...

3D. En trois dimensions, donc dans l'espace. Le monde est 3D, mais nous le simulons souvent en utilisant des outils variés : perspective en peinture ; une image par œil en stéréoscopie ou télévision numérique, avec divers procédés pour que chaque œil voie la bonne image : oculaires séparés, lunettes de couleur ou polarisées, lunettes actives à obturateurs intermittents, etc. Un tailleur transforme habilement un morceau de tissu plan en redingote ou en jupe plissée 3D, et l'origami japonais transforme habilement le papier 2D en sculpture 3D.

3G. A rendu le téléphone portable bien plus efficace, merci aux turbocodes de Claude Berrou. A aussi permis la naissance du smartphone. Mais condamné par la 4G.

4D. La 3D avec le temps en plus. Naviguer en 4D, c'est par exemple regarder un film en 3D. Intéressant au début, mais $4D/3D = 4/3 = 1,333...$, donc plus fatigant...

4G. Rend son téléphone bien plus rapide que la 3G, mais bien plus lent que la 5G.

5D. Y a pas.

5G. C'est pour bientôt, et là ça va foncer !

Actionneur. Dispositif électronique qui sert à actionner.

ADSL. Technique issue de la théorie de l'information, qui permet de transformer les anciens fils de téléphone en câbles réseaux performants en utilisant 255 bandes de fréquences au lieu d'une et en s'appuyant sur des codes correcteurs d'erreurs

sophistiqués. L'ADSL a donné un répit à nos bons vieux fils de cuivre, mais pas pour longtemps avec les attaques sauvages de la fibre optique et de la 5G des téléphones. Si nous savions les déterrer, nous pourrions revendre les fils, ce qui pourrait combler une partie de notre déficit chronique vu le prix qu'a atteint le cuivre.

Algorithme. Organisation mécanisable d'opérations élémentaires pour réaliser une tâche donnée. L'algorithmique est la science du développement, de l'analyse et de l'implémentation des algorithmes. Ne s'écrit jamais avec un « y », même si on a le rythme dans la peau. Mot tiré au XIII^e siècle du nom du savant persan Al Khwārizmī, qui nous a apporté aussi les chiffres arabes et l'algèbre. Respect !

Apprentissage automatique. Technique informatique permettant d'entraîner un algorithme à résoudre un problème précis : analyse d'images, compréhension de la parole, traduction des langues, analyse de données génétiques, etc. L'apprentissage *supervisé*, le plus utilisé actuellement, part de grandes bases de données annotées par des hommes (il y a un chien dans cette image, une tumeur dans cette radio, etc.). L'apprentissage *par renforcement* (ou pavlovien) repose sur une idée qui fonctionne avec les animaux et moins bien avec les enfants : à chaque réponse de l'algorithme, dire « c'est bien », « c'est pas bien », ou donner une note entre les deux, comme A, B, C ou D. Ces deux styles d'apprentissage peuvent être couplés. Dans l'apprentissage *non supervisé*, l'algorithme cherche tout seul à expérimenter et à déduire de ses expériences. Nous savons bien faire ça, quelquefois à nos dépens, mais les algorithmes n'y sont pas encore.

Binaire. À deux valeurs, notées 0/1 quand on parle de nombres et *vrai/faux* quand on parle de valeurs logiques. Il est très commun d'unifier ces deux visions en informatique et de considérer un bit tantôt comme une valeur logique ou comme un composant élémentaire d'une valeur numérique. L'informatique travaille quasiment toujours en binaire, et pas que pour une raison physique : le système binaire est le plus primitif et le seul à unifier arithmétique et logique. Pour l'utiliser à la place du système décimal, il suffit d'apprendre à compter sur le nez au lieu des doigts (1 je touche, 0 je touche pas).

Bit. L'objet binaire, élément de base de l'informatique et de la théorie de l'information. Il n'y a rien en dessous, rien de rien, vous aurez beau chercher.

Booléen. Le calcul logique et algébrique du vrai et du faux ou du 0 et du 1 avec les opérateurs *et*, *ou*, *non*. Introduit par George Boole vers 1850, il a révolutionné la

logique, et reste central pour toute l'informatique. On ne sait jamais s'il faut mettre une majuscule ou pas à l'adjectif, mais moi j'en mets une, il le mérite.

Bootstrapping. Technique astucieuse pour s'élever en l'air en tirant sur les sangles de ses *boots*. Sert en informatique à booter les ordinateurs, et à bien d'autres choses. Si les frères Wright avaient connu ça, ils n'auraient pas eu besoin d'inventer l'avion.

Boucle infinie. Voir ce mot.

Bug. Mauvais comportement d'un programme dû à une erreur de ses auteurs. À ne pas confondre avec une panne aléatoire d'une machine. Le mot est bien antérieur à l'informatique : Thomas Edison l'utilisait déjà pour parler des nombreux ennuis qu'on rencontre quand on passe d'un prototype à un produit. Ça ne s'est pas arrangé avec l'informatique.

Bus. Système de transport en commun des données dans les circuits et ordinateurs. Comme à la ville, on passe sa vie à l'attendre, et il est souvent plein.

Capteur. Dispositif électronique qui sert à capter.

Certification. Cadre légal pour la vérification des applications informatiques critiques (avioniques, ferroviaires, nucléaires, médicales, etc.). La certification est généralement conduite par des organismes indépendants, nationaux ou internationaux, et fondée sur le respect de processus de développement, de documentation et de test. La recherche actuelle en informatique peut permettre de remplacer dans certains cas cette certification administrative par une vraie démonstration mathématique. Les exigences de certification peuvent aller du très contraignant, comme en avionique, au plutôt laxiste, comme en automobile. On est content quand elle est là et qu'elle marche.

Cerveau électronique. Ancien nom de l'ordinateur. C'était tellement plus joli ! Essayez donc d'aller dans un supermarché en acheter un, vous verrez comment on vous regardera.

Circuit intégré. Assemblage de composants électroniques (essentiellement des transistors) dans une puce pour réaliser les calculs informatiques. Les circuits intégrés sont nés dans les années 1970 et ont révolutionné l'informatique. Ils comportent maintenant un grand nombre de transistors, qui se compte en millions pour les petits ou milliards pour les gros. Au poids, ils sont vraiment très chers, bien plus chers même que la romanée-conti.

Cloud computing, ou calcul dans le nuage. Façon d'effectuer des calculs à distance dans des « fermes de calcul » situées n'importe où dans le monde, en envoyant ses programmes et données par réseau et en récupérant ses résultats de la même façon. Le *cloud* est souvent plus souple et moins coûteux que le calcul chez soi, en particulier parce qu'on ne peut pas garder ses propres machines à jour des progrès alors que les fermes le font mieux grâce à un élevage intensif.

Compilateur. Programme qui permet de traduire un programme en un autre programme qui fait la même chose, mais pas pareil.

Complexité des algorithmes. Coût des algorithmes en temps de calcul, mémoire, énergie, ou toute autre mesure. Nomme aussi la science fort complexe qui cherche à évaluer et minimiser ces coûts.

Crash. Mot anglais pour dire plantage. Voir ce mot.

Décidable, indécidable, semi-décidable. Un problème qui peut être résolu à coup sûr par un algorithme est appelé *décidable*. C'est par exemple le cas pour le problème de savoir si on peut trier une liste d'éléments (essayez de le montrer). Il existe heureusement beaucoup de problèmes décidables, pour lesquels l'informatique s'attache à trouver des algorithmes efficaces. Mais il existe aussi beaucoup de problèmes *indécidables*, le plus fameux ayant été découvert par Turing en 1936 : quand on se donne un programme P et une donnée D , existe-t-il un algorithme s'arrêtant à coup sûr et disant si P va s'arrêter en temps fini ou non quand on lui donne D en entrée ? Le problème de Turing est dit *semi-décidable* : il existe un algorithme qui répond toujours oui si et seulement si P s'arrête sur la donnée D , mais soit répond non soit ne termine jamais sinon. Cet algorithme est trivial : lancer P sur D et regarder s'il s'arrête ! Mais il existe aussi des problèmes non semi-décidables, qui ne sont que le début d'une hiérarchie infinie de difficulté des programmes. Une prise de tête, mais une grande découverte.

Données. Les informations que vous avez données à votre ordinateur, sous une forme numérisée qu'il est capable de mâcher et de mettre en boîte, pour vous les ressortir après, soit seules, soit mélangées avec d'autres, en utilisant de savantes recettes appelées algorithmes.

Électronique. Dispositif physique fondé sur l'utilisation des électrons. Le périmètre d'utilisation du mot s'est curieusement étendu à des domaines pas du tout

électroniques : par exemple le courrier électronique est transporté majoritairement par des photons dans des fibres optiques.

Fibre optique. Dispositif physique en verre qui sert à transformer notre courrier électronique en courrier photonique, encore plus rapide. Classe !

Implémentation. Réalisation d'un algorithme sur un ordinateur. Contrairement à ce que pensent beaucoup, ce mot est bien légal, voir <http://www.academie-francaise.fr/implémenter> (ça marche même avec l'accent dans l'adresse, étonnant, non ?).

Inria. Grand institut français de recherches en informatique et mathématiques appliquées situé dans huit agglomérations françaises. Par ordre alphabétique : Bordeaux, Grenoble, Lille, Nancy, Paris, Rennes, Saclay et Sophia-Antipolis. Autrefois, acronyme d'Institut national de recherche en informatique et automatique.

Intelligence artificielle. Quête d'une intelligence non naturelle, implémentée sur un ordinateur par exemple. Certains disent que le sujet n'existera que quand l'ordinateur sera capable de pouffer de rire alors qu'on lui raconte une histoire drôle qu'il ne connaît pas. Mais ce sont des mauvaises langues. J'ai appris que Google et Facebook s'étaient cotisés pour construire le premier ordinateur réellement intelligent. Je connais la première question qu'ils vont lui poser : « Dieu existe-t-il ? » Et je devine la réponse : « Maintenant, oui. » Mais trêve de balivernes : pour les gens sérieux du domaine, c'est aussi une grande quête sur les possibilités réelles des machines comparées à nos propres limites, qui peut aussi nous éclairer sur ce que nous appelons sans trop de modestie l'« intelligence humaine ».

Internet. Le grand réseau qui connecte tout ce qui passe. On l'utilise pour lire le journal, bouquiner, tchatcher avec les copains, faire les courses et même travailler. Mais ce n'est qu'un début : bientôt, on l'utilisera aussi pour siffler sa voiture.

Langage de programmation. Moyen permettant d'écrire des algorithmes simples de façon nettement plus compliquée, que les ordinateurs savent pourtant avaler car ils ont horreur de la simplicité. Il y en a des milliers, mais un seul de vraiment bien : Esterel. Devinez pourquoi.

Logique formelle. Définition et calcul de la vérité des formules mathématiques. La logique formelle a commencé avec Aristote, a longtemps été considérée comme une activité philosophique, et n'est devenue un domaine scientifique qu'au milieu du XIX^e siècle avec George Boole, Gottlob Frege, etc. Kurt Gödel, Alan Turing, Alonzo

Church, puis Stephen Kleene ont établi les relations précises entre calcul et logique, concentrées maintenant dans la correspondance de Curry-Howard, qui dit en gros « calculer, c'est prouver ». Cette correspondance et les résultats obtenus par des logiciens comme Nicolaas de Bruijn et Jean-Yves Girard ont permis le développement des théories et pratiques actuelles pour la vérification de programmes (HOL, Isabelle, Coq, etc.). Vous suivez ? Sinon, relisez lentement trois fois.

Loi de Moore. Décision industrielle de multiplier par deux le nombre des composants électroniques d'un circuit tous les deux ans environ. C'est grâce à elle qu'on peut perdre beaucoup de temps à regarder des stupidités dans nos téléphones dits intelligents. Mais elle a sérieusement du plomb dans l'aile. On la regrettera.

Machine abstraite. Machine conceptuelle qui sert à des tas de résultats mathématiques, mais qu'il serait vraiment idiot de rendre concrète.

Machine de Turing. La machine abstraite qui a servi à Alan Turing pour prouver des résultats mathématiques décoiffants, mais qu'il serait complètement idiot de rendre concrète (voir [Machine abstraite](#)). Ça a cependant été fait avec des briques Lego à l'Université d'Aarhus au Danemark et à l'école normale supérieure de Lyon, ce qui en dit long sur ces institutions.

Mathématiques. Rassembler ses souvenirs d'école et lire d'autres livres (ça ne manque pas). Grandes copines de l'informatique, et surtout des méthodes formelles.

Mémoire de masse. Mémoire de grande taille destinée à conserver des données de manière permanente. On y trouve pour l'instant des bandes magnétiques, des disques, des mémoires flash, etc. D'autres technologies arriveront certainement dans le futur. Curieusement, la technologie permet que la masse de données pèse de moins en moins lourd avec le temps.

Mémoire flash. Réalisation de mémoire de masse sur des circuits électroniques très compliqués, bien plus rapide et plus fiable que les disques, mais aussi bien plus chère à volume égal. Quand même nettement moins rapide que la mémoire vive. C'est elle qu'on trouve dans les clefs USB et dans les cartes pour appareils photo ou téléphones. En général pleine d'images et de musiques, dont une bonne moitié de provenance douteuse.

Mémoire vive (ou RAM). La mémoire rapide de votre ordinateur, effectivement très vive, mais qui oublie tout quand on lui coupe le courant. Il est important d'avoir des RAM rapides, sinon votre ordinateur rame.

Méthodes formelles. Méthodes mathématiques pour la spécification, la programmation et la vérification des applications informatiques. Il faut être formel : ce sont les méthodes de choix quand la qualité des réalisations finales est essentielle.

Mot. Élément de base de la mémoire et du langage machine, composé de quelques octets. C'étaient autrefois des petits mots ; avec les progrès de la technologie, ce sont maintenant des gros mots.

Nombres fixes. Nombres entiers représentés par un nombre fini de bits, souvent 16, 32 ou 64. Les nombres fixes existent en deux variantes : les entiers naturels, allant par exemple de 0 à 65 535 pour 16 bits, et les entiers relatifs où le bit de poids fort sert de signe, qui vont de -16 192 à 16 191 pour 16 bits. Pour 32 et 64 bits je vous laisse compter. Et vous pouvez aller aussi loin que vous voulez avec les bibliothèques logicielles adéquates.

Nombres flottants. Nombres à grande dynamique représentés sous la forme d'une mantisse m et d'un exposant e , la valeur du nombre étant alors $m \times 2^e$. Les nombres flottants et leurs opérations sont normalisés par la norme IEEE 754, pas très poétique malgré son joli nom. Ils sont fondamentaux pour les calculs d'analyse numériques, la simulation, l'apprentissage profond et bien d'autres applications. Mais ils sont surnois, car ils demandent de contrôler finement leurs débordements pour ne pas partir en vrille. Et, si vous les mettez dans l'eau, ne vous attendez pas à ce qu'ils flottent (dans le mercure, en revanche, c'est possible).

Octet. Bouquet de 8 bits. Les bits se vendent toujours par 8, contrairement aux huîtres et aux œufs, qui se vendent par 12, probablement depuis les Sumériens. Heureusement que les Sumériens n'ont pas inventé l'ordinateur, ça n'aurait pas été pratique car 12 n'est pas une puissance de deux.

Ordinateur. Machine pour l'instant électronique permettant d'exécuter les programmes est de se connecter avec le monde physique. L'ordinateur est l'opposé de l'homme : au lieu d'être intuitif, d'une rigueur toute relative et lent comme nous, il est brutalement stupide, parfaitement rigoureux, extraordinairement rapide et pas du tout orgueilleux. Nous faisons une addition en quelques secondes, il en fait des milliards par seconde. Mais ses moyens intellectuels limités lui demandent d'en faire énormément pour faire quoi que ce soit d'intéressant, comme monter une mayonnaise avec un bras robot. Ses erreurs viennent des logiciels que nous écrivons pour lui, pas

de son propre fonctionnement. La phrase classique : « L'ordinateur a buggé » n'est qu'une mauvaise excuse.

Ordinateur quantique. Ordinateur fait avec des qubits, susceptible d'être exponentiellement plus rapide que les ordinateurs classiques sur certains problèmes combinatoires grâce à des superpositions quantiques d'états qui ne peuvent se stabiliser que sur des solutions. Difficile d'en dire plus.

Ordinateur Shadok. Les Shadoks, étrange peuplade que l'équipe de Jacques Rouxel à l'ORTF a visitée à quatre reprises, n'avaient que quatre cases mémoire dans leur cerveau. C'était assez limitant. Cependant, un jour, ils sont tombés sur un livre d'Émile Henriot que leur avait apporté Jacques Rouxel. Une maxime les a frappés : « La culture, c'est ce qui demeure dans l'homme lorsqu'il a tout oublié. » Ils ont immédiatement compris que cette maxime s'appliquerait encore mieux à eux qu'aux humains, vu leur impossibilité technique de ne pas oublier. Ils ont lu le plus vite possible tous les livres qui leur tombaient sous l'aile, afin de les oublier exactement aussi vite. Leur culture a donc grandi à la même vitesse. Insatisfait, le professeur Shadoko, mon idole personnelle, décida d'inventer l'ordinateur, une machine capable d'oublier encore plus vite qu'eux. Cela a permis d'augmenter de plusieurs ordres de grandeur la culture des Shadoks. Avez-vous déjà réalisé à quel point notre culture augmenterait si nous arrêtions définitivement Internet, d'un seul coup, d'un seul ? Vertigineux...

Panne. Problème réservé au matériel. Un logiciel peut avoir des bugs mais ne tombe jamais en panne, c'est pour ça qu'il faut préférer les logiciels aux machines. D'autant plus que quand on ne met pas un logiciel sur une machine, il ne va même pas exhiber de bug. Allez comprendre...

Parallélisme. Propriété d'un calcul ou d'une implémentation qui peut faire plusieurs choses à la fois. Les mathématiciens avec leur unique crayon ne faisaient qu'une chose à la fois. Mais les physiciens lui ont rajouté un « e » pour obtenir le CrayOne, une des premières machines parallèles. Maintenant, le parallélisme est partout : le nombre d'activités parallèles sur Internet est hallucinant, je n'arrive plus à suivre même avec deux doigts.

Plantage. Mot français pour dire *crash*. Voir ce mot.

Programmation. Écriture d'un programme (quelquefois surnommée codage). Activité très gratifiante pendant le court moment où on pense que ça marche, après

qu'on a passé des jours où ça marchait presque, et avant qu'on se rende compte que ça ne marche pas vraiment...

Protocole de sécurité. Système compliqué d'échanges de messages électroniques pour tomber d'accord sur la façon d'échanger davantage en toute intimité. À la cour du Roi-Soleil, un protocole simple mais efficace a été abondamment utilisé : un baisemain, un mouchoir qui tombe négligemment d'une main consentante, et le tour est joué... Avec les mouchoirs en papier actuels, ça marche moins bien.

Puce. Boîtier contenant un ou plusieurs circuits et communiquant électriquement (ou parfois optiquement) avec l'extérieur par ses pattes. N'aime pas qu'on les lui coupe, ça la rend sourde. Une chose extraordinaire : alors qu'on lutte depuis toujours contre les puces des chiens et des chats domestiques, voilà-t-y pas qu'on leur injecte maintenant des puces électroniques sous la peau !

Qubit. Élément de calcul d'un ordinateur quantique. Difficile d'en dire moins.

Randomisé. De *random*, « au hasard » en anglais. Un algorithme randomisé utilise dans ses opérations le tirage au hasard, qui permet par exemple d'explorer efficacement un espace de recherche en évitant de se coincer dans des minimas locaux. Les animaux aussi randomisent, relire le [chapitre 8](#) et chercher les fourmis. Une fameuse méthode de recherche randomisée s'appelle la « méthode de Monte-Carlo ». Mais pourquoi donc ?

Récuratif, récursivité. On dit qu'une définition est *récursive* si elle s'appelle elle-même. La définition récursive mathématico-informatique la plus simple est celle de la fonction factorielle : « $fact(n) = \text{si } n = 0 \text{ alors } 1 \text{ sinon } n \times fact(n - 1)$ ». Un autre exemple, plus accessible, est la définition récursive de l'oignon : un oignon, c'est soit rien, soit un oignon entouré d'une peau d'oignon. La récursivité était autrefois déconseillée ; elle est maintenant encouragée car c'est de loin la meilleure méthode de programmation pour les fainéants. Mais le mot a un autre sens : la théorie de la récursivité est la théorie des fonctions calculables par des algorithmes quelconques, pas forcément récuratifs. La collision des mots vient du fait qu'il a été historiquement plus simple d'étudier cette théorie en utilisant systématiquement des définitions récursives, car les logiciens sont des fainéants.

Réseau. Moyen de faire communiquer entre eux des ordinateurs ou des objets connectés. Selon l'échelle, un réseau est dit sur puce quand il est cantonné à un

circuit, personnel (associé à un individu), local (à un bâtiment), ou global (à l'échelle du monde). On ne va pour l'instant pas plus loin, mais qui sait ?

Sécurité informatique. Ensemble des méthodes qui consistent à protéger les programmes et données qu'on a à cœur. Sujet (et phrase) difficile à comprendre pour les non-initiés.

Séquentialité. Propriété d'un calcul qui ne fait qu'une chose à la fois. Opposé de parallélisme, et réciproquement. Mais bien utile aussi : essayez de faire en même temps une multiplication de la main droite et une division de la main gauche, et vous y reviendrez.

Simulation. Activité qui consiste à faire semblant de faire quelque chose d'utile en remplaçant la nature par un ordinateur.

Smartphone (ou téléphone intelligent). Téléphone connecté sur Internet et capable de fonctions autonomes ou synchronisées avec les ordinateurs : photographie, cartographie, accès à Internet, jeux, SMS, et même téléphone pour les plus de 18 ans qui se parlent encore au lieu de s'envoyer des SMS ou de tchatter. Le téléphone est rendu « intelligent » par ses fonctions et connexions, donc par les logiciels que des hommes qui se trouvent intelligents ont écrits pour lui. Contrairement à un homme, il n'est en lui-même pas plus intelligent allumé qu'éteint. Mais ça pourrait changer un jour si l'intelligence artificielle progresse suffisamment. Peut-être se servira-t-il un jour de vous au lieu de l'inverse. Mais je ne suis pas sûr de voir ça de mon vivant (après, qui sait ?).

Spécification. Ensemble des documents définissant l'objectif et les contraintes d'un projet ; action de construire ces documents. Un programme sans spécification, c'est comme une maison sans plan : ça peut marcher, à condition que ce soit tout petit.

Spécification formelle. Utilisation de techniques à bases mathématiques pour l'écriture des spécifications et l'étude de leur cohérence et de leur complétude. C'est clair, non ?

Spintronique. Utilisation du spin des électrons dans les circuits électroniques. Le spin est le moment magnétique de l'électron, à valeur binaire *haut* ou *bas* – mais il n'y a pas d'étiquette sur l'électron pour l'indiquer. La première grande application de la spintronique a été l'utilisation de la « magnétorésistance géante » dans les disques magnétiques. L'idée est d'écrire et de lire les bits sur leur surface comme des petites zones magnétisées dans un sens ou dans le sens opposé. Quand ils passent devant la

tête de lecture magnétisée dans une direction fixe, la résistance des fils métalliques ne varie pas de la même façon pour les électrons de spin haut et bas selon que le champ du bit est parallèle ou opposé à celui de la tête – qui l'eut cru. Ce principe a donné le prix Nobel 2007 à Albert Fert et Peter Grünberg. En réduisant la taille des bits et en facilitant énormément leur écriture et leur lecture, il a surtout permis l'accroissement de la taille des disques de quelques mégaoctets à quelques téraoctets, avec un impact énorme sur l'informatique moderne. La spintronique trouvera probablement d'autres applications dans les circuits intégrés. Très rigolo en tout cas.

Structures de données. Organisation logique ou physique des données pour les rendre facilement accessibles par des algorithmes, leur bonne organisation étant cruciale pour la performance des algorithmes. Sec, non ?

Sûreté. Qualité que doit avoir un programme de ne pas faire n'importe quoi. Votre voiture est sûre si elle ne décide pas brusquement de bouder et de ne plus freiner.

Système d'exploitation. Programme implanté sur une machine et servant à gérer toutes ses ressources propres, l'accès aux réseaux et aux objets connectés à l'ordinateur, et l'interaction avec les utilisateurs. Objet typique de luttes acharnées entre les petit-boutistes (Windows, Mac OS X, iOS) et le gros-boutistes (Linux, Android). Il faut dire que les coquetiers sont incompatibles.

Test. Application de la méthode essais-erreurs pour trouver des bugs dans les programmes ou les circuits. Si vous en trouvez c'est bien, sinon il faut continuer. Le jour où votre logiciel est mis en circulation, il va être utilisé par des milliers de testeurs hostiles. Cachez-vous !

Trou de sécurité. Bug permettant d'entrer illégalement dans un système informatique. Comme pour un ballon de baudruche, un trou minuscule permet de réduire un système à néant – adieu vos données chéries. Et comme pour un ballon de baudruche, il est très difficile de trouver des rustines qui collent vraiment pour boucher le trou mais qui ne restent pas collées au doigt à la place.

Vérification. Action de vérifier qu'un circuit ou un programme fait pile poil ce qu'il doit faire, ni plus ni moins.

Vérification formelle. Utilisation de techniques à bases mathématiques pour la vérification. Dans les bons cas, utilisable toute seule pour remplacer le test en garantissant sur facture et rubis sur l'ongle que tout marche. Sinon, utile pour

complémenter le test en garantissant des propriétés critiques, par exemple en étant sûr de ne pas voir votre ascenseur informatisé oublier de s'arrêter au dernier étage (certains n'ont jamais été retrouvés).

Bibliographie

- ABADI M. (2011), *La Sécurité informatique*, Collège de France/Fayard, chaire Informatique et sciences numériques, 2010-2011, <http://www.college-de-france.fr/site/martin-abadi/index.htm>.
- ABITEBOUL S. (2012), *Sciences de données. De la logique du premier ordre à la Toile*. Collège de France/Fayard, chaire Informatique et sciences numériques, 2011-2012, <http://www.college-de-france.fr/site/serge-abiteboul/index.htm>.
- ABITEBOUL S., Dowek G. (2017), *Le Temps des algorithmes*, Le Pommier.
- ABRIAL J.-R. (1996), *The B-Book : Assigning Programs to Meanings*, Cambridge University Press.
- ABRIAL J.-R. (2010), *Modeling in Event-B : System and Software Engineering*, Cambridge University Press.
- Académie des sciences (2013), *L'Enseignement de l'informatique en France. Il est urgent de ne plus attendre*, rapport RA-153, http://www.academie-sciences.fr/pdf/rapport/rads_0513.pdf.
- ADRIAN D., BHARGAVAN K., DURUMERIC Z., GAUDRY P., GREEN M., HALDERMAN J. A.,..., ZIMMERMANN P. (2015, octobre), « Imperfect forward secrecy : How diffie-hellman fails in practice. 22nd ACM Conference on computer and communications security (CCS '15) », Denver, <https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf>.

- ALTSCHUL S. F., MADDEN T. L., SCHÄFFER A., ZHANG J., ZHANG Z., WEBB M., LIPMAN D. J. (1997), « Gapped BLAST and PSI-BLAST : A new generation of protein database search programs », *Nucleic Acids Research*, 25 (17), p. 3389-3402.
- AYACHE N. (2014), *Le Patient numérique personnalisé : images, médecine et informatique*. Collège de France/Fayard, chaire Informatique et sciences numériques, 2013-2014, <http://www.college-de-france.fr/site/nicholas-ayache/index.htm>.
- BERGER B., DANIELS N. M., YU W. (2016, août), « Computational biology in the 21st century : Scaling with compressive algorithms », *Communications of the ACM*, 59 (8), p. 72-80.
- BERROU C., GRIPON V. (2012), *Petite mathématique du cerveau. Une théorie de l'information mentale*, Odile Jacob.
- BERRY G. (2000), « The foundations of Esterel », dans *Proof, Language and Interaction : Essays in Honour of Robin Milner*, MIT Press, Foundations of Computing Series.
- BERRY G. (2007), *Manifeste pour la réhabilitation du Pavillon des poids et mesures* (vol. 1), *Viridis Candela, Correspondancier du Collège de Pataphysique*, n° 1, p. 33-60.
- BERRY G. (2008), *Pourquoi et comment le monde devient numérique*, Collège de France/Fayard, chaire d'Innovation technologique Liliane Bettencourt 2007-2008, <http://www.college-de-france.fr/site/gerard-berry/index.htm>.
- BERRY G. (2009), *Penser, modéliser et maîtriser le calcul informatique*, Collège de France/Fayard, chaire Informatique et sciences numériques, 2009-2010, <http://www.college-de-france.fr/site/gerard-berry/index.htm>.
- BERRY G. (2013), *Le Temps et les Événements en informatique*, Collège de France/Fayard, chaire Algorithmes, machines et langages, <http://www.college-de-france.fr/site/gerard-berry/index.htm>.
- BERRY G. (2015), *L'Importance des langages en informatique*, cours du Collège de France donné à l'Irisa Rennes, <http://www.college-de-france.fr/site/gerard-berry/course-2015-11-04-16h00.htm>.
- BERRY G., BOUDOL G. (1992), « The chemical abstract machine », *Theoretical Computer Science*, 96, p. 217-248.

- BOISSONNAT (2017), *Géométrie algorithmique : données, modèles, programmes*, Collège de France/Fayard, chaire Informatique et sciences numériques, 2016-2017, <http://www.college-de-france.fr/site/jean-daniel-boissonnat/index.htm>.
- CANI M.-P. (2015), *Façonner l'imaginaire. De la création numérique 3D aux mondes virtuels animés*, Collège de France/Fayard, chaire Informatique et sciences numériques 2014-2015, <http://www.college-de-france.fr/site/marie-paule-cani/course-2014-2015.htm>.
- CARBONE A. (2017), « Réinventer la biologie », interview dans le blog Binaire du journal *Le Monde*, <http://binaire.blog.lemonde.fr/2017/04/20/amarrage-de-proteines/>.
- CHANGEUX J.-P. (2006), *L'Homme neuronal*, Fayard (1983) et Hachette Littérature.
- CHANGEUX J.-P., CONNES A. (1989), *Matière à pensée*, Odile Jacob.
- CHAZELLE B. (2012), *L'Algorithmique et les Sciences*, Collège de France/Fayard, chaire Informatique et Sciences numériques, 2012-2013, <http://www.college-de-france.fr/site/bernard-chazelle/index.htm>.
- COMBES F. (2014), *La Matière noire dans l'univers*, Collège de France/Fayard, chaire Galaxies et cosmologie, <http://www.college-de-france.fr/site/francoise-combes/index.htm>.
- COUSOT P., COUSOT R. (1977), « Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints », *Proc. 4th ACM Symposium on Principles of Programming Languages*.
- CRAIG VENTER J. (2013), *Life at the Speed of Light : From the Double Helix to the Dawn of Digital Life*, Penguin Books.
- DE MOURA L., BJØRNER N. (2011), « Satisfiability modulo theories : Introduction and applications », *Communications of the ACM*, 54 (9), p. 69-77.
- DEHAENE S. (2007), *Les Neurones de la lecture*, Odile Jacob.
- DEHAENE S. (2014), *Le Code de la conscience*, Odile Jacob.
- DOWEK, G. (2015), *Les Métamorphoses du calcul*, Le Pommier.
- DOWEK G., ARCHAMBAULT, J.-P. (2013), *Informatique et sciences du numérique. Manuel de spécialité ISN en terminale*, Eyrolles.
- FERNANDES E., JUNG J., PRAKASH A. (2016), « Security analysis of emerging smart home applications », *Proc. 37th Symposium on Security and Privacy*.

- GOUSSARD F. (2016), *Quand l'informatique voyage dans le temps, ou l'apport de l'imagerie 3D en paléontologie*, Collège de France, chaire Algorithmes, machines et langages, Colloque Arts et sciences, de nouveaux domaines pour l'informatique, <http://www.college-de-france.fr/site/gerard-berry/symposium-2016-05-27-10h00.htm>.
- HEARD E. (2013), *Épigénétique et mémoire cellulaire*, Fayard/Collège de France, chaire Épigénétique et mémoire cellulaire, <http://www.college-de-france.fr/site/edith-heard/index.htm>.
- Informatics Europe (2017), *Informatics Education in Europe : Are We All in the Same Boat ?*, <http://www.informatics-europe.org/news/382-informatics-education-in-europe-are-we-on-the-same-boat.html>.
- KLEIN E., PERRY-SALKOW J. (2011), *Anagrammes renversantes ou le Sens caché du monde*, Flammarion.
- LASKAR J. (2010), *Le système solaire est-il stable ?*, Séminaire Poincaré XIV.
- LAZARD E., MOUNIER-KUHN P. (2016), *Histoire illustrée de l'informatique*, EDP sciences, Klee group.
- LE BIHAN D. (2012), *Le Cerveau de cristal. La nouvelle science de la neuro-imagerie*, Odile Jacob.
- LE CUN Y. (2016), *L'Apprentissage profond. Une révolution en intelligence artificielle*, Collège de France/Fayard, chaire Informatique et sciences numériques, 2015-2016, <http://www.college-de-france.fr/site/yann-lecun/index.htm>.
- LECUIT T. (2017), *Dynamiques du vivant*, Collège de France/Fayard, chaire Dynamiques du vivant, <http://www.college-de-france.fr/site/thomas-lecuit/index.htm>.
- LEVESON N. (1995), « Medical devices : the Therac-25 », Software Systems Safety and Computers, Addison Wesley, <http://archive.wikiwix.com/cache/?url=http%3A%2F%2Fsunnyday.mit.edu%2Fpapers%2Ftherac.pdf>.
- LEVESON N., TURNER C. (1993), « An investigation of the Therac-25 accidents », *IEEE Computer*, 26 (7), p. 18-41.
- LIVNAT A., PAPADIMITRIOU C. (2016), « Sex as an algorithm : The theory of evolution under the lens of computation », *Communications of the ACM*, 59 (11), p. 84-93.

- MAÎTRE H. (2015), *Du photon au pixel*, ISTE Éditions.
- MALLAT S. (2017), *Sciences des données*, Fayard/Collège de France, chaire Sciences des données, <http://www.college-de-france.fr/site/stephane-mallat/index.htm>.
- MANOURY P. (2017), *L'Invention de la musique*, Collège de France/Fayard, chaire de Création artistique, 2016-2017, <http://www.college-de-france.fr/site/philippe-manoury/index.htm>.
- MATTHEWS R. (2000), « Storks Deliver Babies ($p = 0.008$) », *Teaching Statistics*, 22.
- MINSKY M., PAPERT S. (1988), *Perceptrons*, MIT Press.
- MULLER J.-M. (1995), « Algorithmes de division pour microprocesseurs : illustration à l'aide du "bug" du Pentium », *Technique et science informatiques*, 14 (8).
- NARTEAU C. (2016), *Sculptures éoliennes au sein des mers de sable*, Collège de France, chaire Algorithmes, machines et langages, Colloque Arts et sciences, de nouveaux domaines pour l'informatique, <http://www.college-de-france.fr/site/gerard-berry/symposium-2016-05-27-14h00.htm>.
- REEVES G., TRACEY N. (2004), « The Mars rover spirit FLASH anomaly », IEEE Xplore Digital Library, <http://ieeexplore.ieee.org/abstract/document/1559723/?reload=true>.
- ROUX D. (2017), *Recherche, invention et innovation*, Collège de France/Fayard, chaire d'Innovation technologique Liliane Bettencourt 2016-2017, <http://www.college-de-france.fr/site/didier-roux/index.htm>.
- SAHEL J. (2016), *Voir encore. La restauration visuelle en perspectives*, Collège de France/Fayard, chaire d'Innovation technologique Liliane Bettencourt 2015-2016, <http://www.college-de-france.fr/site/jose-alain-sahel/course-2015-2016.htm>.
- SOBEL D. (1998), *Longitude*, Seuil.
- STANDAGE T. (2014), *The Victorian Internet*, Bloomsbury Publishing.
- TEYSSIER R. (2016), *L'Univers dans un ordinateur. Simuler les galaxies et les étoiles*, Collège de France, chaire Algorithmes, machines et langages, colloque Arts et sciences, de nouveaux domaines pour l'informatique, <http://www.college-de-france.fr/site/gerard-berry/symposium-2016-05-27-09h15.htm>.
- The Royal Society (2012), *Shut Down or Restart ? The Way Forward for Computing in UK Schools*, <https://royalsociety.org/~media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf>.

- TURING A. (1936), « On computable numbers, with an application to the Entscheidungsproblem », *Proceedings of the London Mathematical Society*.
- TURING A. (1950, octobre), « Computing machinery and intelligence », *Mind*, 59 (236), p. 433-460.
- TURING A. (1952), « The chemical basis of morphogenesis », *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237 (64), p. 37-72.
- VALIANT L. (2013), *Probably Approximately Correct : Nature's Algorithms for Learning and Prospering in a Complex World*, Basic Books.
- WILGENBUSH D., HIRTZIG M., CALMET, C. (2016), 1, 2, 3... *Codez !* (volume 1, cycle 3), Le Pommier.
- WILGENBUSH D., HIRTZIG M., CALMET, C. (2017), 1, 2, 3... *Codez !* (volume 2, cycle 4), Le Pommier.
- YANG X., CHEN Y., EIDE E., REGER J. (2011), « Finding and understanding bugs in C compilers », *Proc. ACM SIGPLAN conf. on Programming Languages Design and Implementation (PLDI)*, San Jose (USA).

Index

Al Khwārizmī 50

Algorithmes 21, 43, 50, 74, 435

- 3-coloration 439
- à coût constant 436
- apprentissage automatique 79, 353
- automates cellulaires 239
 - gaz sur réseau 242
 - jeu de la vie 239
- bibliothèques 409
- biologiques 259
- blockchain* 139
- bug 84
- calcul symbolique 277
- chiffrement 76, 275
 - logarithme discret 442
 - postquantique 442
 - RSA 407, 441
- complexité 82, 435-436, 439
 - classes 436
 - de Kolmogorov 261
- compressed sensing* 252

contrôle 79
Conway, John 239
Cook, Stephen 370, 438
correction 82, 435
d'addition 436
Davis & Putnam (SAT) 370
de débruitage 157
de localisation 78
de recherche en table 437
de synchronisation 83
de synchronisation temporelle 78
de traitement d'images 77
de traitement du son 76
de tri 437
distribués 347, 447
DPLL (SAT) 370
emploi du temps 438
en astronomie 79
en biologie 75, 79
en cryptologie 439
énergie 435
en musique 76
en physique 79
en télécommunications 115
et mathématiques 81
exponentiels 437
factorisation quantique 407
Flajolet, Philippe 277
fourmis 259
génériques 75
géométriques 276
GRASP (SAT) 371
heuristiques 370, 439
humains 75
indécidabilité 435
jeu de la vie 239
Lamport, Leslie 447
linéaires 436
logarithmiques 437
mémoire 435
moteur de recherche 437
naturels 259
Neumann, John von 239
 $n \times \log(n)$ 437
PageRank 440
Papadimitriou, Christos 260
parallèles 82
parallélisation 445
polynomiaux 437
probabilistes 440
protocoles de communication 83
protocoles de sécurité 76

- randomisés 259, 440
- résolution de contraintes 258
- rétropropagation de gradient 80
- RSA (Rivest-Shamir-Adleman) 275
- SAT 274, 439, 442
- sémaphore 310
- séparation de sources 77
- séquentiels 82
- Shor, Peter 407
- spécifiques 75-76
- statistiques 79
- sublinéaires 436
- Sudoku 438, 442
- superexponentiels 437
- sur le texte 76
- sur matrices 82
- téléphone 112
- temps de calcul 435
- terminaison 435
- tirage au sort 440
- Turing, Alan 435
- Valiant, Leslie 259
- zChaff (SAT) 371

- Algorithmique 435
 - distribuée 447
 - parallèle 84
 - pensée 75

- Apprentissage automatique 74, 79, 136, 440
 - AlphaGo 80
 - analyse de scène 441
 - analyse d'images 415
 - bayésien 441
 - conduite automobile 416
 - critères de succès 81
 - deep learning* 79, 136, 353
 - en médecine 198, 216
 - Therapixel 217
 - tumeurs du sein 217
 - en photographie HDR 180
 - et hommes 81
 - images médicales 441
 - jeu de go 441
 - malédiction de la dimension 81
 - non supervisé 80
 - pifomètre 416
 - reconnaissance de la parole 415, 441
 - réseaux de neurones profonds 79, 136, 440
 - supervisé 80, 440
 - support vector machines* 441
 - Torch (langage) 464

traduction des langues 415, 441

Astronomie 222, 228

aberrations 230

astéroïdes 233

Bell, Jocelyn 229

Combes, Françoise 230

énergie noire 232

fausses couleurs 229

galaxies 233

grappes de satellites 235

interférométrie 235

ALMA (Atacama Large Millimeter/submillimeter Array) 235

LOFAR (Low Frequency Array) 236

SKA (Square Kilometer Array) 236

Laskar, Jacques 233

lentilles gravitationnelles 230

matière noire 231

Philae 231

Poincaré, Henri 232

Professeur Calys 228

robots 231

simulations informatiques

collisions de galaxies 233

galaxies 233

multiéchelles 234

système solaire 233

télescopes 221

E-ELT (European Extremely Large Telescope) 235

Hubble 222

optique adaptative 235

VLT (Very Large Telescope) 235

Automates cellulaires 239

dunes de sable 240

jeu de la vie 239

stochastiques 241

von Neumann, John 239

Automobile 29

ABS (Anti Blocking System) 63, 317

accélération incontrôlable 321

aide à la conduite 294

airbags 317

auto-écoles virtuelles 65

autonomie 318, 353

bug d'arrêt moteur 318-319

bug de contrôle moteur 321

bug de multiplexage 320

bug de portes bloquées 320

bugs 318

class action 322

commission d'enquête 321

- consommation 29
- cruise control* 317
- freinage 29
- informatisation 317
- radar 317
- régulateur d'allure 317
- sécurité informatique 341
- services informatisés 29
- sûreté 293

Biologie 45

- ADN 45
- bio-informatique 249
- biophysique 247
- cellulaire 257
- cellule 51
 - croissance 247
 - division 247
- Changeux, Jean-Pierre 263
- compressed sensing* 252
- computationnelle 45, 249
 - Carbone, Alessandra 256
 - Fages, François 257
- Craig Venter, John 250, 253
- Danchin, Antoine 260
- Dassault Systèmes 257
- éthologie 259
- expérience 56
- expérimentation 248
- fourmis 259
- Gascuel, Olivier 252
- génomique 45, 249
- Gilbert, Walter 250
- imagerie cellulaire 223
- Institut Pasteur 249
- Livnat, Adi 260
- logiciel
 - BioCham 257
 - BLAST (Basic Local Alignment Search Tool) 252
 - OpenAlea 261
 - PhyML 252
- morphogenèse 2 248
- neurosciences 263
- organisme 51
- Papadimitriou, Christos 260
- protéines
 - Folding@home 256
 - simulation du repliement 256
- réseaux métaboliques 257
 - analyse quantitative 257
- resynthèse de l'ADN
 - Craig Venter, John 253

- Sanger, Frederick 250
- séquençage de l'ADN 223, 248
 - Craig Venter, John 250
 - Gilbert, Walter 250
 - NTS (*new technology sequencing*) 250
 - Sanger, Frederick 250
- séquenceurs ADN 79
- simulation
 - BioCham 257
 - repliement des protéines 256
- système nerveux 51
- Turing, Alan 248
- Valiant, Leslie 259
- végétale 261
 - Godin, Christophe 261
 - méristème 261
 - simulation 261
 - Virtual Plants 261

Boole, George 46

Bugs 22, 66, 84, 96, 291-292

- accélération incontrôlable 321
- AECL 325
- à Manhattan 98
- apprentissage automatique 441
- architecture de sécurité 322
- Ariane 501 300, 348, 364
- AT&T 331, 346
- automobiles 318
- Barr, Michael 321
- break mal placé 332
- class action* 322
- commissions d'enquête 321, 324
- compilateur C 329
- confusion d'unités 314
- contrôle moteur 321
- course parallèle 327
- coût 322, 324, 334
- deadlock 100, 311
- débordement arithmétique 301
- débordement de pile 322
- de circuits 94, 389
- de micronoyaux 381
- des navigateurs Web 335
 - Internet Explorer 335
- de systèmes distribués 448
- DOS 313, 348
- Edison 97
- failsafe* défectueux 321
- famine 101
- Ford 319

[GCC](#) [330](#)
[Intel](#) [94](#)
[interblocage](#) [100](#), [311](#)
[iPhone 4 \(réveil\)](#) [297](#)
[Le Monde](#) [100](#)
[le moteur ne s'arrête pas](#) [318](#)
[les contrôler](#) [102](#)
[les éviter](#) [102](#)
[Leveson, Nancy](#) [325](#)
[ligne de changement de date](#) [299](#)
[Linux systemd](#) [331](#)
[LLVM](#) [330](#)
[Mars Climate Orbiter](#) [314](#)
[martien](#) [309](#)
[mercredi](#) [297](#)
[messages d'erreur](#) [326](#)
[méthodes formelles](#) [102](#)
[mont Sainte-Odile](#) [316](#), [352](#)
[multiplexage](#) [320](#)
[mutex](#) [328](#)
[Nicely, Thomas](#) [275](#), [333](#)
[overflow](#) [301](#)
[= panne humaine](#) [98](#)
[parallélisme](#) [100](#)
[Pathfinder](#) [101](#), [309](#), [355](#)
[Patriot](#) [299](#)
[Pentium Pro](#) [333](#), [346](#), [368](#), [379](#)
[portes bloquées](#) [320](#)
[priorité](#) [311](#)
[probabilité](#) [308](#)
[Queen Elizabeth](#) [337](#)
[Raptor](#) [298](#), [356](#)
[schéma mental](#) [323](#)
[Schiaparelli](#) [315](#)
[Sony PS3 Fat](#) [296](#)
[Spirit](#) [312](#), [348](#)
[starvation](#) [101](#)
[téléphone](#) [100](#), [102](#), [308](#)
[test](#) [102](#)
[Therac-25](#) [324-325](#), [358](#)
[Timsort](#) [365](#)
[TLS](#) [339](#)
[Toyota](#) [321](#), [358](#)
[trous de sécurité](#) [22](#), [103](#)
[Turner, Clark](#) [325](#)
[USS Yorktown](#) [330](#)
[vote électronique](#) [351](#)
[Windows NT](#) [330-331](#)
[Zune](#) [294](#)

[Calcul](#) [43](#)

Cartographie

- 3D [192](#)
- accès aux données géographiques [192](#)
- carte
 - informatique [189](#)
 - papier [189](#)
- fusion de données [192](#)
- Google [191](#)
- Google Street View [192](#)
- GPS [189-191](#), [193](#)
- itinéraire [189](#)
- numérique [190](#)
- OpenStreetMap [192](#)
- orientation [189](#)
- photos aériennes [190](#)
- satellite [190](#)
- SPOT [190](#)
- télédétection [190](#)
- téléphone [189-192](#)

Certification des logiciels

- DO-178B [316](#)
- DO-178C [62](#), [316](#)
- SCADE 6 (Esterel Technologies) [465](#)

Circuits [88](#), [357](#)

- 3D [400](#)
- architectes [357](#)
- architecture [89](#)
- ARM [95](#)
- batterie [90](#)
- bio-inspirés [400](#)
- bootstrapping* [93](#)
- bugs [94](#), [389](#)
- bus [93](#)
- Cadence Design Systems [94](#)
- CAO [89](#), [94](#)
- conception logique [94](#)
- consommation [90](#)
- coût
 - de la CAO [399](#)
 - des bugs [399](#)
 - des usines [399](#)
- dark silicon* [399](#)
- design [389](#)
- designers [357](#)
- de téléphone [90](#)
- Digital Equipment [387](#)
- dissipation thermique [90](#)
- électroniques [88](#)
- émulation FPGA [360](#)
- entrées-sorties [92](#)

Esterel Technologies 388
Esterel v7 389
fabless 95
fonderies 89, 94
FPGA 94, 360, 387
fréquence d'horloge 89-90, 94
gigahertz 90
GPS 92
GPU (*graphics processing unit*) 89, 91-92, 445
graphène 96, 401
Hewlett Packard 96, 400
Intel 94, 388
Itanium (Intel) 92
Kishinevsky, Michael 388
loi de Moore 88-89, 94, 398-399
 arrêt 95
masques 389
mémoire 89
 flash 401
memristors 96, 400
Mentor Graphics 94
microprocesseur 89, 91-92, 98
Moore, Gordon 88
multicœurs 83, 91
mur de la chaleur 399
nanomètre 89
optoélectroniques 401
parallélisation 91
patch 389
Pentium Pro 333
pipe-line 91
portes logiques 93
processeur graphique 89
puissance de calcul 89
Qualcomm 95
radio 92
RAM 400
réseaux sur puce 92
Samsung 94
silicium 95
silicium noir 399
simulation logicielle 360
SoC 92, 360
spéculation 91, 445
spintronique 96, 401
synchrones 464
Synopsys 94
système sur puce 92, 360
télécommunications 115
transistors 89
voltage 90
Vuillemin, Jean 386

CNRS 37, 397
biologie 249

Collège de France 15, 146
chaire Algorithme, machines et langages 15
chaire Création artistique 146
chaire d'Informatique et sciences numériques 15, 259
chaire d'Innovation technologique Liliane Bettencourt 15
chaire Sciences des données 74
Docet Omnia 230

Cybernétique 62

Dématérialisation 30

Données 21, 67
Abiteboul, Serge 413
ADN 70
stockage 403
stockage de données 249
analyse 73
anonymisation 414
base de données
distribuée 447
big data 73
bit 68
causalité 73
chantage à la publication 103
cigognes 74
cloud storage 402
compressed sensing 402
compression
avec perte 73
FLAC 72
sans perte 72
TIFF 72
contexte d'interprétation 71
corrélation 73
dans le nuage 402
destruction 103
fermeture 413
films 69
génomique 402, 414
géographiques 192
image 69
INA (Institut national de l'audiovisuel) 403
information 70-72
Internet 72
interprétation 72
mémoires de masse 401
mesure physique 68
nombres flottants 246, 276
octet 68

- ouverture [413](#)
- poids [70](#)
- propriétés [192](#), [413](#)
- sauvegarde [402](#)
- science des données [74](#)
- son [69](#)
 - Hi-Res [70](#)
- stockage [398](#)
- structure de [85](#)
- support physique [72](#)
- texte [68](#)
 - code ASCII [70](#)
 - ISO [70](#)
- vidéo [69](#)
- vol de [103](#)

Électronique [64](#)

Enabler [118](#)

Énergie [42](#)

Enseignement

- 1, 2, 3... Codez !* [429](#)
- agrégation [433](#)
- Angleterre [426](#)
- Chine [426](#)
- Class'Code [428](#)
- classes préparatoires [425](#)
- Computing at School [426](#)
- CSP (Conseil supérieur des programmes) [35](#), [428](#)
- débranché [430](#)
- de l'informatique [35](#)
- États-Unis [426](#)
- formation des enseignants [36](#)
- Inde [426](#)
- Informatics Europe [426](#)
- Informatique et création numérique (ICN) [427](#)
- Informatique et sciences du numérique (ISN) [427](#)
- Inria [428](#)
- Interstices [428](#)
- La Main à la pâte [428](#)
- Montessori, Maria [34](#)
- Peyton Jones, Simon [426](#)
- Pologne [426](#)
- réseaux d'échanges [431](#)
- robots [425](#)
- Scratch [430](#)
- SIF (Société informatique de France) [428](#)
- supérieur [425](#)

Génie logiciel

- revues indépendantes [358](#)

GPS

- Galileo [194](#)
- horloges atomiques [193](#)
- relativité générale [193](#)
- synchronisation temporelle [193](#)

Herzstark, Curt

- Curta [53](#)

Heuristiques [439](#)

Hilbert, David [46](#)

Hopper, Grace [97](#)

Images

- compression
 - JPEG [73](#)
- conduite automatique [145](#)
- fusion [172](#), [179](#), [207-208](#)
- JPEG [69](#)
- pixel [69](#)
- radiologie interventionnelle [211](#)
- reconnaissance automatique [145](#)
- satellites [145](#)
- SIGGRAPH [146](#)
- sRGB [69](#)
- surveillance [145](#)
- synthèse [145](#)

Information [21](#), [26](#), [43](#), [45](#), [53](#)

- données [71](#)
- mémoire [58](#)
- numérisation [58](#)
- support [21](#)

Informatique [25](#), [53](#)

- Abiteboul, Serge [23](#)
- actionneurs [61](#)
- Airbus A320 [62](#), [308](#), [315](#)
- Airbus A380 [306](#)
- algorithmes [50](#), [67](#), [74](#)
- Alphabet [32](#)
- Amazon [32](#)
- analyse statique [87](#)
- API (*application programming interface*) [55](#)
- Apple [32](#), [297](#)
- atomicité [327](#)
- autobus [29](#)
- automates cellulaires [226](#)
- automobile [29](#), [317](#), [393](#)
 - ABS [63](#)
- avionique [62](#)

- avions [29](#)
- base de données [347](#)
 - Datalog [464](#)
 - distribuée [447](#)
 - temporelles [447](#)
- Bell Laboratories (AT&T) [446](#), [456](#)
- bibliothèques [55](#)
- big data [73](#)
- bit [68](#), [70](#)
- bootstrapping* [56](#)
- bugs [66](#), [96](#)
- calcul dans le nuage [56](#)
- calcul symbolique [228](#)
- CAO (conception assistée par ordinateur) [29](#), [64](#)
 - Boeing 777 [64](#)
- capteurs [61](#)
- cartes à puce [382](#)
- certification des logiciels [62](#), [292](#), [316](#)
- Chazelle, Bernard [259](#)
- cinéma [60](#)
- cloud computing* [56](#), [60](#), [135](#), [404](#)
- CNRS [37](#)
- computer science* [53](#)
- critères de succès [346](#)
- crowd computing* [256](#)
- dans la société [422](#)
- Dassault Systèmes [33](#), [64](#)
- dématérialisation [30](#)
- développeur [26](#)
- difficulté [28](#)
- Digital Equipment Corporation (DEC) [131](#)
- distribuée [447](#)
- données [67](#)
 - massives [31](#)
 - temporelles [447](#)
- Dowek, Gilles [23](#), [361](#)
- dualité matériel-logiciel [49](#)
- écologie [406](#)
- effets pervers [394](#)
- élections [423](#)
- enseignement [273](#)
- environnement personnel distribué [352](#)
- EPI (association Enseignement public & informatique) [397](#)
- Esterel Technologies [465](#)
- et mathématiques [273](#)
- et médias [395](#)
- et sciences [31](#)
- Facebook [32](#)
- Fages, François [257](#)
- fermes de serveurs [404](#)
- génie logiciel [354](#)
- géométrie algorithmique [276](#)

Google 32, 131-132
GPS 78
Hopper, Grace 86, 456
IBM 50, 387, 392, 456
IEEE 71
IHM (interface homme-machine) 351, 409-410
llog 387
imprimantes 3D 29, 78, 196, 198
Informatics Europe 53
information 53
InfoSys 33
INS2I (CNRS) 37
intelligence artificielle 2 281
Interaction homme-machine 398
interfaces 67
Internet 34
IRIA 37
ISO 71
Jobs, Steve 457
Knuth, Donald 274
 TeX 447
 The Art of Computer Programming 274
Lamport, Leslie 286
 LaTeX 274
langages 67, 84
liseuse 59
livre numérique 59
logiciel 49, 55
Lovelace, Ada 454
machine abstraite
 Boudol, Gérard 461
 Cham (Chemical Abstract Machine) 258, 461
 machine de Turing 461
machines 67
maquettes numériques 64
matériel 49, 55
mécanique 61
médecine 31, 197
métadonnée 162
méthodes formelles 345, 348
Microsoft 32, 392, 446
mode qui va passer 392
musique 59
niveaux d'abstraction 248
nuage 60
numérisation 68
octet 68, 70
outil 34
parallèle 444
parité 37, 433
pas une science 392
pensée algorithmique 51

- personnalisation 28
- pilliers 67
- programmation 34
 - difficulté 55
 - par contraintes 258
- programme 49
 - calcul sur 87
- programme enregistré 49
- prothèses 29
- réalité augmentée 213
- scan 3D 196
- science de construction 56
- sécurité 30, 76, 291, 410
- séquentielle 444
- services 27
- SIF (Société informatique de France) 397
- simulation numérique 29, 31
- Sony 296
- souffleries virtuelles 65
- spécifications 84
- Stanford Research Institute (SRI) 352
- Steinbuch, Karl 53
- streaming* 60
- sûreté 30, 291-292
- synchronisation temporelle
 - GPS 193
- synthèse d'images 78
- taxis 27
- TCS (Tata Consultancy Services) 33
- téléchargement 56
- temps réel 84
- trains 29
- traitement du signal 53
- trous de sécurité 66, 291
- unbundling* 50
- usages 34
- vélos 29
- vidéo 60
- VTC 27
- Xerox Parc 352

Inra (Institut national de la recherche agronomique) 261

Inria 15, 37, 130, 261

- biologie 249

Intelligence artificielle 2 417

- 5^e génération (Japon) 419

- AlphaGo 80

- Deep Blue 419

- définition 420

- démonstration mathématique 368

- jeu d'échecs 419
- McCarthy, John 418
- Minsky, Marvin 418
- réseaux de neurones profonds 79
- Simon, Herbert 418
- singularité 421
- test de Turing 418
- Turing, Alan 417

- Interface homme-machine 351
 - Engelbart, Douglas 352
 - Hullot, Jean-Marie 357, 457
 - iPhone 357
 - Mac OS X 357
 - NeXT 357
 - souris 352
 - spécifications 357
 - test 359

- Internet 19, 34, 72, 114, 117, 392, 413
 - adresse IP 120, 340
 - algorithmes adaptatifs 122
 - best effort* 120
 - Bitcoin 116, 139
 - Cerf, Vinton 119
 - cloud computing* 135
 - datagrammes 118
 - des objets 30, 79, 141
 - détection de communautés d'intérêt 136
 - diffusion épidémique 137
 - enablers* 118
 - exaoctet 71
 - Facebook 149
 - FTP (*file transfer protocol*) 119, 125
 - impacts 117
 - inversion mentale 118
 - IP (*Internet protocol*) 119-120
 - IPv4 (32 bits) 123
 - IPv6 (128 bits) 123
 - Kermarrec, Anne-Marie 1
 - liens hypertextes 131
 - Massoulié, Laurent 1
 - médecine 197
 - meilleur effort 120
 - modularité 121
 - moteur de recherche 83, 130, 393, 437
 - navigateur Web 125
 - neutralité 72, 139
 - objets connectés 141
 - of things* 30
 - P2P 137
 - pair à pair 116, 137

- Napster [137](#)
- paquets [118](#)
 - durée de vie [121](#)
 - perte [121-122](#)
- passage à l'échelle [121](#)
- peer to peer* [116](#), [137](#)
- publications scientifiques [227](#)
- réseaux sociaux [8](#) [23-24](#), [127](#), [135-136](#), [149](#), [393](#)
- routage [116](#)
- routeur [120](#)
- sécurité [117](#)
- son [122](#)
- streaming* [139](#)
- synchronisation temporelle [194](#)
- tables de routage [120](#)
- TCP/IP [122](#)
- TCP (*transfer control protocol*) [119](#), [121](#)
- téléchargement [116](#)
- transactions sécurisées [139](#)
- UDP (*user datagram protocol*) [119](#)
- vidéo [122](#)
- vie privée [117](#)
- VoIP (*voice over Internet protocol*) [114](#), [122](#)
- Web [117](#), [134](#)
 - 2.0 [127](#)
- Yahoo [131](#)
- zettaoctet [71](#)

Inversion mentale [27](#), [113](#), [118](#), [178](#), [187](#), [189](#)

Ircam [146](#)

IRIA

- Lions, Jacques-Louis [37](#)

Langage [84](#)

- de programmation [84](#), [86](#)
 - compilateur [87](#)
- de spécification [84-85](#)
- graphique [86](#)
- sémantique [84](#)
- syntaxe [84](#)
- textuel [86](#)

Langages de programmation [409](#), [452](#)

- λ -calcul [454](#)
- acteurs [461](#)
- ADA [378](#), [460](#)
- ALGOL 60 [456](#)
- Backus, John [455](#)
- Banâtre, Jean-Pierre [461](#)
- C [381](#), [456-457](#)
- C++ [457](#)

- Caml [382](#), [458](#)
- CLight [383](#)
- COBOL [86](#), [455](#)
- Colmerauer, Alain [281](#), [419](#)
- compilateurs [92](#), [409](#)
- CSP (Communicating Sequential Processes) [460](#)
- Dahl, Ole-Johan [456](#)
- Datalog [464](#)
- de haut niveau [86](#)
- dialectes [86](#)
- DSL (*domain specific language*) [463](#)
 - CSS [463](#)
 - de script [463](#)
 - HTML [463](#)
 - JavaScript [463](#), [465](#)
 - Perl [463](#)
 - Python [463](#)
 - shell [463](#)
 - Tcl [463](#)
 - Torch [464](#)
- Eiffel [364](#)
- Erlang [461](#)
- Esterel [130](#), [371](#), [386](#)
 - v7 [388-389](#)
- F# [458](#)
- FORTRAN [86](#), [225](#), [273](#), [455](#)
- Gallina [453](#)
- GAMMA [461](#)
- Gosling, James [457](#)
- Haskell [4](#) [381](#), [458-459](#)
- Hewitt, Carl [461](#)
- HipHop.js [130](#)
- Hoare, Tony [456](#)
- Hop [130](#)
- Hop.js [130](#)
- Hopper, Grace [456](#)
- Java [365](#), [457](#)
- JavaScript [127](#), [130](#), [411](#)
- Kay, Alan [457](#)
- langage machine [55](#)
- Le Métayer, Daniel [461](#)
- Leroy, Xavier [330](#), [458](#)
- LISP [411](#), [455](#), [457](#)
- machines virtuelles [457](#)
- Meyer, Bertrand [364](#)
- Milner, Robin [457](#), [460](#)
- ML [2](#) [457](#)
- modularité [55](#), [459](#)
- Naughton, Patrick [457](#)
- Nygaard, Kristen [456](#)
- Objective-C [457](#)
- OCaml [7](#) [382-383](#), [458-459](#), [465](#)

- OCCAM [460](#)
- Odersky, Martin [458](#)
- parallélisme [460](#)
 - mémoire partagée [460](#)
 - threads* [460](#)
- Pascal [456-457](#)
- Peyton Jones, Simon [1](#)
- PHP (PHP *hypertext preprocessor*) [127](#)
- Prolog [281](#), [419](#), [464](#)
- Python [365](#), [411](#)
- réactifs synchrones [464](#)
 - Benveniste, Albert [464](#)
 - Boussinot, Frédéric [465](#)
 - Caspi, Paul [464](#)
 - Esterel [464](#)
 - Halbwachs, Nicolas [464](#)
 - Le Guernic, Paul [464](#)
 - Lustre [464](#)
 - Mandel, Louis [465](#)
 - Marmorat, Jean-Paul [464](#)
 - Reactive C [465](#)
 - Reactive ML [465](#)
 - Rigault, Jean-Paul [464](#)
 - SCL [465](#)
 - Signal [464](#)
 - von Hanxleden, Reinhard [465](#)
- Ritchie, Dennis [456](#)
- SCADE 6 [464](#)
- Scala [458-459](#), [461](#)
- Scheme [130](#), [455](#)
- sémantique [459](#)
- Serrano, Manuel [130](#)
- Simula 67 [456-457](#)
- Smalltalk [457](#)
- Standard ML [458](#)
- Steele, Guy [455](#)
- Stroustrup, Bjarne [457](#)
- styles [86](#)
- Sussman, Jay [455](#)
- Swift [458](#)
- syntaxe [459](#)
- temps réel [464](#)
- Thompson, Ken [456](#)
- threads* [446](#)
- typage [458](#)
- Wirth, Niklaus [456](#)

Logiciel [74](#), [409](#)

- Automath [282](#)
- bibliothèques scientifiques [413](#)
- biologie
 - BioCham [257](#)

- BLAST (Basic Local Alignment Search Tool) [252](#)
- OpenAlea [261](#)
- PhyML [252](#)
- bootstrapping* [56](#), [87](#), [93](#), [451](#)
- CATIA [64](#)
- CGAL [413](#)
- compilateur [55](#), [87](#), [347](#), [364](#), [448](#)
 - bootstrapping* [450](#)
 - C [346](#), [382](#)
 - CompCert [330](#), [382](#)
 - Esterel [387](#)
 - GCC [330](#)
 - LLVM [330](#)
 - optimisation [383](#)
 - parallélisant [445](#)
- Coq [282](#), [453](#)
- diversification [308](#)
- HOL (Higher Order Logic) [282](#)
- HyperCard (Apple) [125](#)
- Isabelle [282](#)
- LaTeX [227](#), [273-274](#), [413](#), [447](#)
- libre [412](#)
- ligne de code [112](#)
- Linux [331](#)
- Maple [228](#), [277](#), [463](#)
- Mathematica [228](#), [277](#), [287](#)
- Modelica [463](#)
- moteurs de recherche [57](#)
- navigateur Web [125](#)
- NTP (Network Time Protocol) [78](#), [194](#), [413](#)
- open source* [412](#)
- Simulink [463](#)
- SOFA (Inria) [216](#)
- téléchargement [56](#), [346](#)
- télécommunications [115](#)
- TeX [227](#), [273-274](#), [413](#)
- TiGeR (BDD) [387](#)
- troff* [274](#)
- versions [346](#)

Logique

- λ -calcul [47](#), [49](#), [454](#)
- Aristote [45](#)
- assertions de programmes [361](#)
- assistant de preuve
 - Automath [282](#)
 - Coq [282](#)
 - HOL [282](#)
 - Isabelle [282](#)
- axiome [278](#)
- Boole, George [46](#), [369](#)
- calcul booléen [387](#)

calcul des constructions 375
Cantor, Georg 2 46
Carroll, Lewis 369
Church, Alonzo 47, 49, 286, 454
Coq 282
Coquand, Thierry 375
correspondance de Curry-Howard 376
Curry, Haskell 376
de Bruijn, Nicolaas 282
de premier ordre 375
d'ordre supérieur 375
Girard, Jean-Yves 283
Gödel, Kurt 46, 286
Gordon, Michael 282
Howard, William Halvin 376
Huet, Gérard 375
incomplétude 46
Karnaugh, Maurice 369
logiciens 45
machine de Turing 47
modalité temporelle 366
Paulin, Christine 375
Paulson, Lawrence 282
Pnueli, Amir 366
prédicat 362
Principia Mathematica 279
quantificateur 375
règle de déduction 278
Russell, Bertrand 279
SAT 274
satisfiabilité 369
tautologie 369
temporelle 366, 447
thèse de Church 49
TLA+, *temporal logic of actions* 447
Turing, Alan 47, 49
Whitehead, Alfred 279

Machine de Turing 47
non déterministe 48
programme 49
universelle 48

Machines 21

Mathématiques 31, 46, 86
algèbre 51
Al Khwārizmī 50
analyse complexe 277
analyse numérique 444
Appel, Kenneth 282
appliquées 53, 273
axiome 278

Babbage, Charles 454
calcul booléen 46, 368
calculs 45
 matrices 277
 polynômes 277
 symboliques 228, 277
Cantor, Georg 2 46
chaos déterministe 233
chiffrement 275
Church, Alonzo 47, 49, 286
codes correcteurs d'erreurs 115
complexes simpliciaux 272
conjecture de Kepler 286
Connes, Alain 277
convexité 279
Conway, John 239
Coq 283
Coquand, Thierry 283
de Bruijn, Nicolaas 282
démonstration 278
Entscheidungsproblem 46-47
équations 45
et informatique 227
Feit, Walter 284
Fibonacci 262
fondation 46
Fourier, Charles 76, 82
Gaudry, Patrick 276
géométrie 281
Gödel, Kurt 46, 286
Gonthier, Georges 283
Guthrie, Francis 282
Haken, Wolfgang 282
Hales, Thomas 286
Hilbert, David 280
Huet, Gérard 283
incomplétude 46
Institute for Advanced Study 285
Kolmogorov, Andreï 261
langage 86
LaTeX 274
Lions, Jacques-Louis 37, 305
Mallat, Stéphane 74, 81
Maple 228, 277
Mathematica 228, 277
médaile Fields 34
modèles 64
Neumann, John von 239
nombres premiers 275, 461
 crible de Darwin 462
 crible d'Ératosthène 462
nombres réels 246

Paulin, Christine 283
Poincaré, Henri 232
preuve 278
 relecture 279
Principia Mathematica 279
prix Abel 34
relecture 279
Russell, Bertrand 279
Shannon, Claude 34, 52, 115
SMAI (Société de mathématiques appliquées et industrielles) 397
Tex 274
théorème 278
théorème de Feit-Thompson 284
théorème des 4 couleurs
 preuve Appel-Haken 282
 preuve Gonthier-Werner 283
théorème d'Euler 279
théorie axiomatique 46
théorie de la communication 52
théorie de l'information 52, 115
théorie du contrôle 53, 62-63, 79
thèse de Church 49
Thompson, John 284
traitement du signal 53
Turing, Alan 47, 49
Whitehead, Alfred 279
Wiener, Norbert 62

Matière 42, 231

Matière-énergie 248, 323

Matière-énergie-ondes 109, 221, 425

Mécanique 58

 servomoteurs 61

Médecine

 apprentissage automatique 198, 216

 dermatologie 217

 Therapixel 217

 tumeurs du sein 217

 assistance au diagnostic 198

 biopsie 208

 carte Vitale 197

 chirurgie mini-invasive 211

 Cotin, Stéphane 216

 Haïssaguerre, Michel 1

 imagerie médicale 198

 artefacts 206

 Ayache, Nicholas 199

 Besse, Francis 200

 biopsie optique 208

- Centre cardiologique du Nord 200
- échographie 199
- fusion d'images 207-208
- IRM 205
- quantification 208
- radiographie 198
- Radon, Johann 200
- réalité augmentée 213
- TDM (scanner) 199
- TEP 199
- tumeurs 208
- informatisation 197
- Internet 197
- Le Bihan, Denis 199
- logiciel SOFA (Inria) 216
- Marescaux, Jacques 216
- Mauna Kea Technologies 208
- NeuroSpin 199
- pacemakers 198
- pompes à insuline 198
- prothèses
 - auditives 198
 - genou 198
 - hanche 198
- radiologie interventionnelle 211
- Röntgen, Wilhelm 198

Médias

- et informatique 395

Méthodes formelles 345

Moteur de recherche 83, 130, 437

- AltaVista 131-132
- Bourdoncle, François 135
- Brin, Sergey 132
- Burrows, Michael 131
- collecteur 131
- crawler* 131
- dictionnaire 131
- Exalead 135
- Google 131-132, 135, 191
- images 136
- index 131
- Monier, Louis 131
- musique 136
 - Shazam 136
 - SoundHound 136
- Page, Lawrence 132
- PageRank 132, 440
- popularité 131-132
- vidéos 136
- WebCrawler 131

Musique 59, 61

- Boulez, Pierre 146
- co-improvisation 77, 146
- disque vinyle 58
- distorsions 61
- enregistreur numérique 60
- Manoury, Philippe 146
- modélisation d'instruments 77
- moteur de recherche 136
- RIM (réalisateur d'informatique musicale) 146
- Risset, Jean-Claude 146
- streaming* 60
- suivi de partitions 77, 146
- synthèse de sons 77
- vinyle 59

Neurosciences 263

- Berrou, Claude 271
- cerveau 263-264
 - bruit 266
 - microarchitecture 265, 267
 - oscillateurs 265
- Changeux, Jean-Pierre 263
- codage de l'information 268
- Dehaene, Ghislaine et Stanislas 199
- Dehaene, Stanislas 267
- dendrites 264
- EEG (électroencéphalographie) 266
- Gripon, Vincent 271
- IRM fonctionnelle 266
- Le Bihan, Denis 266
- MEG (magnétoencéphalographie) 266
- mémoire 266-268
 - modèle Berrou-Gripon 271
- neurones 264
- plasticité synaptique 265
- potentiels d'action 264
- spikes* (potentiels d'action) 264
- synapses 264
- Wiener, Sidney 1, 2
- Zugaro, Michaël 1

Numérique 21, 25

Numérisation 58

- de la parole 110

Ondes 42

Ordinateur 21, 44, 49, 51, 53-55, 58, 74, 437

- 5^e génération (Japon) 419
- actionneur 405
- amplificateur d'erreur 99

- calculette [54](#)
- capteur [405](#)
- CDC 6600 [237](#)
- consommation électrique [406](#)
- Cray One [444](#)
- données [55](#)
- ECU (*electronic control unit*) [403](#)
- embarqué [405](#)
- exaflop [71](#), [404](#), [445](#)
- form factor* [404](#)
- IBM
 - Blue Gene [236](#)
 - Deep Blue [419](#)
- lavable [406](#)
- multiprocesseurs [446](#)
- PC [112](#)
- PDP-11 [327](#)
- pétaflop [71](#)
- pipe-line [444](#)
- programme [55](#)
- programme enregistré [54](#)
- quantique [407](#), [442](#)
 - qubit [407](#)
- réseau [83](#)
- séquentiel [82](#)
- superordinateur [403](#), [444](#)
- tablette [59](#), [404](#)
- vectorisation [444](#)
- vs téléphone [112](#)

Paquet numérique [110](#)

Parallélisme [444](#)

Pensée algorithmique [75](#)

Photographie

- 24 × 36 [157](#)
- 3D [195](#)
- aberrations chromatiques [159](#), [177](#), [188](#)
- agrandissement [162](#)
- aide à la prise de vue [147](#)
- algorithmes [149](#)
- anamorphose de volume [175](#)
- Angénieux, Pierre [187](#)
- anti-aliasing* [163](#)
- à-plats blancs [154](#), [169](#)
- APS-C [157](#)
- argentique [59](#), [61](#), [148](#), [156](#), [159](#), [166](#), [178](#)
- artefact [163](#)
- artistique [158](#)
- autofocus [159](#), [179](#)
- balance des blancs [165-166](#)

- Boyle, William 149
- bruit 154-157, 169
- calibrage d'écran 164
- capteur 147, 149, 153, 170
 - CCD 149
 - CMOS 149
 - silicium 154
- compact 149
- contraste 164
- convertisseur analogique-numérique 157
- correction des défauts optiques 148
- corrections automatiques 147
- couleurs 165
- Cuvillier, Roger 187
- débruitage 157, 169-171
- dématriçage 163
- de paysage 158
- dérastisation 163
- diaphragme 153, 158-160, 170, 184
- diapositives 155
- distorsion 188
- distorsion géométrique 175
- DxO Labs 148
- dynamique 154, 180
- éléments parasites 183
- exposition 148
- film 154
- flash 156-157
- flou 158
 - artistique 151, 158
 - de bougé 159
 - floutage 160
- focale 158, 160, 162
- focus stacking* 182
- franges de couleurs 177
- fusion d'images 148, 159, 172, 179, 182, 192
- fuyantes 177
- GPS 162
- grain 156
- grand-angle 175
- gris de référence 167
- Guichard, Frédéric 148
- HDR (*high dynamic range*) 180
- hologrammes 156
- image crue 160
- iPhone 180
- ISO 155-156
- JPEG 69, 160, 164, 167
- Kodachrome 155
- lissage 172
- lumière 164, 169, 175
 - artificielle 166

- naturelle 166
- Lytro 159
- macrophoto 159, 182
- Maître, Henri 148
- matrice de Bayer 161
- métadonnée 162
- micro 4
 - 3 157
- microcontrastes 168
- miroir 188
- mise au point 148, 158-159, 175, 186-187
- moiré 163
- Morel, Jean-Michel 1
- négatif 155
- netteté 168, 179, 182
- noir et blanc 151
- noirs bouchés 169
- numérique 30, 59, 145, 148, 156, 159, 166, 178
- objectif 153, 188
- ouverture 153, 162, 175
- panorama 183
- paysage 159
- pellicule 153
- photosite 156-157, 161-163
- pixel 162
- portrait 158
- profondeur de champ 158-160
- quantité d'information 162
- raw* 160, 162, 172
- réduction du bruit 164
- résolution 161
- révélateur 156
- revues 149
- rolling shutter* 153
- saturation 154, 167
- selfies 406
- sensibilité 147, 149, 155-157, 159, 162, 169
- Smith, George 149
- stabilisation 159, 184
- stéréoscope 195
- super-résolution 182
- surexposition 154
- sur Internet 149
- taille de capteur 157
- téléphone 59, 147, 157-158, 160, 180, 185-186
- température de couleur 167
- temps de pose 153
- TIFF 160
- vignettage 173
- viseur 147, 184, 186-188
 - à-plats 185
 - écran 184

- électronique 188
- focus peaking* 186
- histogrammes 186
- LCD (*liquid crystal display*) 185-186
- niveaux 185
- paramètres 185
- profondeur de champ 186
- reflex 149, 153, 184-185, 187-188
- scintillement 185
- yeux rouges 156
- zoom 187

Physique 64, 115

- accélérateur de particules 222
- aérodynamique 65
- Aspect, Alain 409
- biophysique 247
- boson de Higgs 237
- capteur 68
- CCD (*charge coupled device*) 229, 236
- chat de Schrödinger 224
- climatologie 227
- de la matière condensée 226
- de la matière molle 227, 247
- discrète 246
- dunes de sable 226
- Einstein, Albert 230
- entropie 45
- expérience 56
- Fert, Albert 89
- fibres optiques 57, 90, 115
- Galilée 223
- Gennes, Pierre-Gilles de 247
- Grünberg, Peter 89
- Hawking, Stephen 421
- horloges atomiques 193, 221
- IPGP (Institut de physique du globe de Paris) 238
- ITER (tokamak) 238
- LCD (*liquid crystal display*) 90
- LED (*light emitting diode*) 1
- Lévy-Leblond, Jean Marc 270
- LHC (Large Hadron Collider) 237
- masses de données 222
- mécanique quantique 246
- mesure 68
- Newton 232
- newtonienne 42
- océanographie 227
- photon 57
- relativité générale 230
- relativité restreinte 246
- simulation numérique 238

spintronique 89, 96
statistique 45, 439
transistor 57, 112

Plan Calcul 237

Prix Nobel 34

Fert, Albert (2007) 89
Gennes, Pierre-Gilles de (1991) 247
Gilbert, Walter (1990) 250
Grünberg, Peter (2007) 89
Sanger, Frederick (1990) 250

Prix Turing 34

Adleman, Leonard M. (2002) 441
Berners-Lee, Tim (2017) 124, 238
Cerf, Vinton (2017) 119
Clarke, Edmund (2007) 367
Cook, Stephen (1982) 370, 438
Diffie, Whitfield (2015) 441
Dijkstra, Edsger (1962) 360-361
Emerson, Allen (2007) 367
Engelbart, Douglas (1997) 352
Floyd, Robert (1978) 361
Hellman, Martin (2015) 441
Hoare, Charles Anthony (1980) 361
Knuth, Donald (1974) 274
Lampert, Leslie (2013) 274, 447
Lampson, Butler (1992) 352
Milner, Robin (1991) 377
Pnueli, Amir (1996) 366
Rivest, Ronald L. (2002) 441
Shamir, Adi (2002) 350, 441
Sifakis, Joseph (2007) 367
Thacker, Charles (2009) 352
Valiant, Leslie (2010) 259

Problème

de l'arrêt 49
indécidabilité 435
indécidabilité 435
indécidable 49
NP-complet 438
NP-difficile 438

Programmation

diffuse 129
multiparticipants 129
multitier 129
orchestration du Web 130
parallèle 84
par contraintes 258
séquentielle 84

Programmes 21
 enregistrés 49

Réseaux 88-89, 398
 ARPANET 118
 client-serveur 124
 Cyclades 119
 datagrammes 118
 DNS 340
 domotique 125
 ethernet 352
 Internet 117
 Minitel 114, 124
 moteurs de recherche 57
 pair à pair 116
 Pouzin, Louis 119
 routage 116, 277, 440
 Thacker, Charles 352
 Zimmermann, Hubert 119

Réseaux sociaux 7 23-24, 127, 135-136, 393
 Facebook 136

Robots 6 24, 79, 89, 231, 334
 astronomiques 231
 chirurgiens 215, 219

Schéma mental 21, 28-29, 35, 65, 188, 302, 323, 391-392, 395, 410, 424-425
 automobile 293

Sciences
 analyse de données 226
 bio-informatique 247, 249
 biologie computationnelle 247, 249
 de la terre 238
 expériences de pensée 224
 informatisation 221
 des instruments 226
 des publications 227
 instrumentation 221
 masses de données 222
 modélisation mathématique 223
 simulation algorithmique 226

Sécurité informatique 291, 349, 394, 410
 Adleman, Leonard M. 441
 Android 339
 antimalware 340
 antivirus 340
 attaques 334-335
 canaux cachés 350
 DDNS 340
 dédi de service 340

- Freak 339, 385
- Heartbleed 339
- Inria 339
- Logjam 339
- man in the middle* 338-339
- Microsoft Research 339
- par les objets 143, 340
- PDF 336
- Petya 336
- Stuxnet 350
- sur le serveur Dyn 143, 340
- sur les objets 143, 341
- sur NTP 194
- synchronisation temporelle 194
- vente aux enchères 349
- WannaCry 336
- Barghavan, Karthikeyan 385
- bot* 142, 340
- chantage 103
- chiffrement 76, 337
 - asymétrique 441
 - clef 338
- code malveillant 336
- confidentialité 103, 349
- Cortier, Véronique 2 343
- débordement de mémoire 335
- déni de service 142
- destruction de données 103
- Diffie, Whitfield 441
- domotique 342
- Fournet, Cédric 385
- Gaudry, Patrick 340
- hackers 143-144, 334, 349
- hameçonnage 142, 334
- Hellman, Martin 441
- https 143, 337-338
- intégrité 349
- Internet 142
- Jeep Cherokee 341
- messages d'erreur 343
- Microsoft Research 385
- Miller, Charlie 341
- mises à jour 336
- navigateurs Web 335
- NSA (National Security Agency) 339
- ordinateur quantique 338
- ouverture des portes 341
- pacemakers 342
- pair à pair
 - CodeRed 138
- passport français 343
- phishing* 142, 334

- protocoles de sécurité 76, 337-338
- rançonnage 103, 142, 334
- ransomware 142
- Rivest, Ronald L. 441
- RSA (Rivest-Shamir-Adleman) 350, 441
- Samsung SmartThings 342
- Shamir, Adi 350, 441
- signature 439
- SSL (*secure socket layer*) 143, 384
- TLS (*transport layer security*) 143, 338, 384-385
- trous de sécurité 66, 103, 291, 334
- Valasek, Chris 341
- vérification formelle 384
- virus 335
 - CodeRed 138
 - Petya 336
 - Stuxnet 350
 - WannaCry 336
- vol de données 103
- vote électronique 350
- Web 142

Shannon, Claude 52, 115

Simulation

- avion 65, 359
- biologie
 - des plantes 262
 - du repliement des protéines 256
- dunes de sable 238
- galaxies 233
- Narteau, Clément 238
- système solaire 233

Smartphone 112

- consommation électrique 112

Son 146

- 3D 195
- acoustique 146
- analyse 146
- binaural 195
- modification de la voix 77
- MP3 73
- restauration 77
- séparation de sources 77
- simulation d'instruments 146
- spatialisation 146
- synthèse 146

Spécifications 84

- absence de 378
- ascenseur 347, 365

- changement permanent 356
- cohérence 84
- complétude 84
- contraintes 85
- contrats 364
- de la vérification 356
- de l'environnement 85, 356
- des tests 356
- formelles 86
- incohérentes 355
- incomplètes 355
- langages 85
- logique temporelle 366
- messages d'erreur 85
- non fonctionnelles 85, 356
- sécurité 85, 347
- vivacité 347

Sûreté 292

- business critical* 293
- certification des logiciels 292
- mission critical* 293
- revues 292
- safety critical* 292
- test 292
- time critical* 293

Synthèse d'images

- Cani, Marie-Paule 145

Système

- cyberphysique 64, 79, 184, 359, 365
 - avionique 359
- distribué
 - tests 359

Système d'exploitation 50, 55, 83, 298, 380, 409, 446, 451, 460

- Android 339, 365, 380, 410, 446
- cellulaire (en biologie) 261
- DOS 446
- héritage de priorité 311
- iOS 380, 410, 446
- Leroy, Xavier 460
- Linux 78, 194, 380, 412-413, 446
 - threads* 460
- Mac OS 9 446
- Mac OS X 380, 446
- microkernel* 380
- micronoyau 380
 - L4 380
 - Minix 381
 - seL4 380
- mode debug 311

- mutex [310](#)
- PDP-11 [327](#)
- préemptif [310](#)
- sémaphore [310](#)
- time-sharing* [446](#)
- Torvalds, Linus [446](#)
- Unix [446](#), [456](#)
- VxWorks [310](#)
- Windows [374](#), [380](#), [446](#)
 - écran bleu [373](#)
 - Vista [337](#)
 - XP [337](#)

Télécommunications [26](#), [107](#)

- ADSL [52](#), [114](#)
- antennes [115](#)
- baud [108](#), [114](#)
- Baudot, Émile [108](#)
- Bell Laboratories (AT&T) [114](#)
- Berrou, Claude [116](#)
- câbles [108](#)
- codes correcteurs d'erreurs [115-116](#)
- consommation électrique [114](#)
- correction d'erreurs [114](#)
- Ericson [461](#)
- fibres optiques [114-115](#)
- filaire [114](#)
- FTTH (*fiber to the home*) [1](#)
- Gallager, Robert [116](#)
- gigabit/seconde [52](#)
- Glavieux, Alain [116](#)
- intercontinentales [108](#)
- LDPC (*low-density parity check*) [116](#)
- Lechleider, Joseph [114](#)
- limite de Shannon [116](#)
- probabilité de succès [116](#)
- protocoles [83](#)
- rapport signal/bruit [115](#)
- redondance [115](#)
- Shannon, Claude [115-116](#)
- télégraphe [43](#)
- téléphone [43](#)
- télévision numérique [110](#), [114](#)
- turbocodes [116](#), [271](#)
- VoIP (*voice over Internet protocol*) [114](#), [119](#), [122](#)

Télégraphe [26](#), [43](#), [107-108](#)

- Chappe [107](#)
- Cooke et Wheatstone [108](#)
- Morse [108](#)

Téléphone [26](#), [43](#), [83](#), [90](#), [93](#), [95](#), [107-108](#), [141](#), [149](#), [225](#), [404](#)

- 2G [111](#)

- 3G [112](#)
- 4G [112](#)
- 5G [112](#), [116](#)
- « 22 à Asnières » [109](#)
- algorithmes [112](#)
- Allô, t'es ou ? [113](#)
- antivol [113](#)
- application [128](#)
- AT&T [111](#)
- Bell, Graham [108](#)
- bruit [109](#)
- carte SIM (*subscriber identity module*) [404](#)
- central électronique [110](#)
- coup de fil [113](#)
- fibres optiques [110](#)
- filaire
 - analogique [109](#)
 - numérique [110](#)
- fixe [113](#)
- Gray, Elisha [108](#)
- inversion mentale [113](#)
- iPhone [392](#)
- logiciels [111](#)
- Meucci, Antonio [108](#)
- numéro [124](#)
- obsolescence programmée [406](#)
- panne [120](#)
- paquets [110](#)
- photographie [59](#), [147](#)
- portable [34](#), [112](#)
- profondeur de champ [160](#)
- raccroché [113](#)
- Raynaud, Fernand [109](#)
- recyclage [112](#), [406](#)
- répondeur [111](#)
- réserve de ressources [119](#)
- routage [119](#)
- sans fil [111](#), [113-114](#)
- smartphone [112](#), [190](#), [393](#)
- standard [108](#)
 - électromécanique [108](#)
- tarification [120](#)
- VoIP (*voice over Internet protocol*) [114](#), [119](#)

- Tests [345](#), [359](#)
 - avionique [359](#)
 - circuits
 - émulation FPGA [360](#)
 - simulation logicielle [360](#)
 - SoC [360](#)
 - système sur puce [360](#)
 - de spécification [356](#)

- de système distribué [359](#)
- génération automatique [348](#)
- non-régression [358](#)

Théorie

- de la communication [34](#)
- de l'information [34](#)

Trous de sécurité [291](#)

Turing, Alan [33](#), [47](#), [49-50](#), [286](#), [361](#), [373](#), [435](#)
morphogenèse [248](#)

- Vérification [84](#)
 - Abrial, Jean-Raymond [377](#)
 - AbsInt [306](#)
 - ACL2 [368](#), [379](#)
 - Alt-Ergo (SMT) [373](#)
 - AMD [368](#)
 - analyse statique [87](#)
 - ascenseur [365](#)
 - assertions [361](#)
 - assistant de preuve [374](#)
 - Coq [453](#)
 - HOL [455](#)
 - Isabelle [455](#)
 - LCF (Logic for Computable Functions) [457](#)
 - Astrée [306](#)
 - Atelier B [378](#)
 - BDD [371](#)
 - B (méthode) [377-378](#)
 - Bolignano, Dominique [381](#)
 - Bryant, Randal [371](#)
 - BTM (Dassault Aviation) [387](#)
 - BubbleSort [377](#)
 - CADP [367](#)
 - calcul booléen [368](#)
 - Clarke, Edmund [367](#)
 - CMM (*capability maturity model*) [382](#)
 - CompCert [376](#), [382-383](#)
 - compilateur
 - C [383](#)
 - Esterel [383](#)
 - Conchon, Sylvain [1](#)
 - contrats [364](#)
 - Cook, Byron [374](#)
 - Coq [376](#), [382](#)
 - Coudert, Olivier [387](#)
 - Cousot, Patrick et Radhia [306](#), [364](#)
 - Csmith [382](#)
 - CVC4 (SMT) [373](#)
 - de compilateur [382](#)
 - de modèles [367](#)
 - Deutsch, Alain [306](#)
 - Dijkstra, Edsger [360-361](#)
 - DPLL (SAT) [370](#)
 - Emerson, Allen [367](#)
 - Event-B [379](#)
 - factorielle [361-362](#)
 - Floyd, Robert [361](#)
 - Garavel, Hubert [367](#)
 - Gordon, Michael [379](#)

GRASP (SAT) 371
Harrison, John 379
Hoare, Charles Anthony 361
HOL-light 379
HOL (Higher Order Logic) 287, 379
Holzmann, Gerard 367
Intel 379
interprétation abstraite 306, 364
Isabelle 381
K5 (AMD) 379
Key (Java/Python) 365
Klein, Gerwin 380
Larsen, Kim 367
Leroy, Xavier 382
logique temporelle 366
Madre, Jean-Christophe 387
Malik, Sharad 371
McMillan, Ken 371
Meteor 377
méthodes formelles 102
micronoyau 380-381
 ProvenCore 381
 seL4 380-381
Microsoft 374
Milner, Robin 1
model-checking 367
NICTA 380
Nqthm, Boyer & Moore 368
OS 380
Paulson, Lawrence 381
Pentium 379
Polyspace 306
postcondition 363
précondition 362
prédicat 362
Prove & Run 381
ProvenCore 381
Prover SL 388
PVS 368
RATP 377-378
RER A 378
Rieg, Lionel 383
Rushby, John 368
Russinoff, David 379
SAT 368, 442
Satisfaction modulo théories 373
satisfiabilité 369
seL4 380
Shankar, Natarajan 368
Sifakis, Joseph 367
SMT (Satisfaction modulo théories) 258, 373, 388
SPIN 367

- Sudoku [369](#), [442](#)
- T2 (terminaison) [374](#)
- tactiques de preuve [377](#)
- tautologie [369](#)
- Terminator [374](#)
- test [102](#)
- tests de non-régression [358](#)
- Timsort [365](#)
- TLA+, *temporal logic of actions* [447](#)
- traçabilité [358](#)
- Trusted Logic [382](#)
- Turing, Alan [361](#)
- UPPAAL [367](#)
- WCET [306](#)
- Z3 (SMT) [373](#)

Vidéo

- compression [73](#)

Virus

- CodeRed [138](#)
- Petya [336](#)
- ransomware* [142](#)
- WannaCry [336](#)

Vision

- aires cervicales [152](#)
- bâtonnets [75](#), [151](#)
- cônes [75](#), [151](#)
- cristallin [151](#), [158](#)
- équilibrage des couleurs [152](#)
- et cerveau [150](#)
- fovéa [151](#)
- ganglions [75](#)
- iris [151-152](#), [154](#)
- microsaccades [152](#)
- œil [151](#)
- perception virtuelle [152](#)
- résolution [151](#)
- rétine [75](#), [151](#)
- saccades [152](#)
- traitement du signal [152](#)

World Wide Web [29](#), [83](#), [134](#), [227](#), [238](#), [352](#), [392](#), [437](#), [447](#)

- Berners-Lee, Tim [124](#), [238](#)
- Cailliau, Robert [124](#)
- client [83](#)
- client-serveur [124](#)
- cloud computing* [135](#)
- composition de services [128](#)
- cookie* [126](#)
- crawler* [131](#)
- domotique [125](#)

- HipHop.js [130](#)
- Hop.js [130](#)
- HTML (*hypertext markup language*) [124](#), [126-127](#)
- HTTP (*hypertext transfer protocol*) [125](#)
- HTTPS (*HTTP secure*) [125](#), [143](#)
- hyperliens [125](#)
- JavaScript [127](#)
- JSON (*JavaScript object notation*) [127](#)
- marshalling* [128](#)
- moteur de recherche [83](#), [130](#), [437](#)
- navigateur [125](#)
- orchestration [130](#)
- PHP (*PHP hypertext preprocessor*) [127](#)
- réseaux sociaux [7](#) [23-24](#), [127](#), [135-136](#), [393](#)
- serveur [83](#)
- services Web [127](#)
- streaming* [139](#)
- téléchargement [138](#)
- URL (*uniform resource locator*) [124](#)
- Web
 - 2.0 [127](#)
- Wikipedia [135](#)

Ce livre se termine sur la page 512. Ce n'était pas voulu, mais, pour un informaticien, c'est une aubaine : nos nombres préférés sont les puissances de 2, et $512 = 2^9$ n'est pas la moindre puisqu'elle s'écrit 1 000 000 000 en binaire.

Pour mes 64 ans (2^6), probablement la dernière puissance de 2 pour moi, j'avais essayé d'impressionner mes petits-enfants en leur disant que j'avais un million d'années (en binaire), mais sans grand succès...

TABLE

Titre

Copyright

Dédicace

Avant-propos

CHAPITRE 1 - Introduction - La construction d'un nouveau schéma mental

Le schéma mental de l'informatique

Structure du livre

Informatique ou numérique ?

La pensée algorithmique

Les dangers de l'immobilité

Heureusement, les esprits changent

Première partie - Du matériel à l'immatériel

CHAPITRE 2 - Les grands principes

Le triangle du XXe siècle

La naissance de la science du calcul et de l'information

De la calculatrice mécanique à l'ordinateur

Comment l'information unifie et transforme tout

CHAPITRE 3 - Les piliers de l'informatique

Les données

Les algorithmes

Les langages

Les machines et circuits électroniques

Les bugs

Deuxième partie - Études de cas

CHAPITRE 4 - Les télécommunications : du fil à l'air

Le télégraphe, une révolution méconnue

La téléphonie classique

La téléphonie sans fil

Du cuivre à l'ADSL et à la fibre optique

Les progrès en algorithmiques de codage et de transmission

CHAPITRE 5 - Internet, des hommes aux objets

La grande inversion mentale du réseau Internet

Le World Wide Web

Les moteurs de recherche

Le pair à pair, un modèle épidémique de diffusion de données

Des objets informatisés à l'Internet des objets

La sécurité informatique d'Internet

CHAPITRE 6 - Images, sons et cartes numériques

La photographie numérique

Comment un appareil photo capte la lumière

De l'image crue à une image visible

Autres corrections algorithmiques

La fusion algorithmique d'images multiples

Aider la prise de vue

La coconception des objectifs et des algorithmes

La cartographie numérique

Le passage à la troisième dimension

CHAPITRE 7 - L'informatisation de la médecine

L'imagerie médicale

L'informatisation de l'imagerie médicale

L'anatomie numérique

La radiologie interventionnelle

L'entrée en force de l'apprentissage automatique en médecine

Ne jamais oublier le patient

CHAPITRE 8 - Vers une informatisation massive des sciences

L'informatisation de l'astronomie

L'informatisation de la physique

Bio-informatique et biologie computationnelle

Les neurosciences computationnelles

L'informatique en mathématiques

Troisième partie - Rendre l'informatique plus sûre

CHAPITRE 9 - Bugs et trous de sécurité : deux dangers de l'informatique

Les bugs de temps ne sont pas rares

Le bug qui a tué Ariane 501

Quelques bugs martiens

Avionique et ferroviaire, des domaines critiques et sérieux

L'automobile, un domaine hélas moins sérieux

Un bug médical historique : le Therac-25

Les bugs d'outillage

Bugs de systèmes distribués

Bugs de circuits

Des bugs aux trous de sécurité

CHAPITRE 10 - Comment rendre l'informatique plus sûre

Un préliminaire indispensable : caractériser les bons et mauvais comportements

Le génie logiciel

Du test aux méthodes formelles

Les assertions logiques

Les systèmes de démonstration automatique

Les assistants de preuve

Les preuves de sécurité

Quand utiliser la vérification formelle ?

Trois expériences personnelles intéressantes

CHAPITRE 11 - Où va nous mener l'informatisation du monde ? Une vision personnelle

Préliminaires

Les évolutions du matériel

L'évolution des logiciels classiques

L'accès aux données et la tension public-privé

Apprentissage automatique ou intelligence artificielle ?

L'informatique dans la société

Enseigner l'informatique, du primaire au lycée

Le problème de la parité en informatique

CHAPITRE 12 - Annexes

12.1. La science des algorithmes

12.2. Résoudre un Sudoku avec SAT

12.3. De la séquentialité au parallélisme et à la distribution

12.4. Les compilateurs et le bootstrapping

12.5. Mais pourquoi autant de langages ?

12.6. L'évolution des langages

Glossaire

Bibliographie

Index

Éditions Odile Jacob

Des idées qui font avancer les idées

Retrouvez tous nos livres disponibles en numérique
sur odilejacob.fr

Retrouvez-nous sur Facebook

Suivez-nous sur Twitter



L'Hyperpuissance de l'informatique

Algorithmes, données, machines, réseaux

Ce n'est que récemment que l'on a commencé à mesurer à quel point l'informatique est en passe de transformer notre société, et même de la bouleverser de fond en comble.

C'est à décrire et à analyser les fondements de l'hyperpuissance de l'informatique que Gérard Berry se consacre dans ce livre qui fera date.

Il montre en effet de façon non technique comment la science et la technologie informatiques mettent l'information au cœur de l'action, qu'elle soit produite par les hommes ou par les machines.

Algorithmes, données, machines et réseaux conduisent surtout à un nouveau schéma mental bien différent de celui des siècles précédents, qui confère un pouvoir étonnant à ceux qui le comprennent et l'organisent.

Pour donner concrètement à comprendre le mode de pensée inhérent à l'informatique, Gérard Berry passe en revue cinq domaines de transformations massives : les télécommunications, Internet, la photographie et la cartographie, l'informatisation de la médecine, et celle en cours de toutes les sciences.

Il analyse ensuite en détail deux dangers de l'informatique, les bugs et les trous de sécurité, qui peuvent parfois transformer des systèmes informatisés en dangers publics, et montre comment la science moderne permet de mieux contrôler ces dangers.

Enfin, l'auteur donne sa vision de l'évolution de l'informatique, bien loin des fantasmes trop souvent partagés.

Pour la première fois, un livre qui explique tout de l'informatique, de son monde, ses fondements, ses applications, et la révolution qu'elle représente.

Gérard Berry est professeur au Collège de France où il dirige la chaire Algorithmes, machines et langages. Il est médaille d'or du CNRS.

Notes

1. <http://binaire.blog.lemonde.fr/>.

2. Plus précisément : « Les trucs en -
tique, informatique, bureautique,
télématicque, productique, ce sont des
modes qui vont passer. »

3. Turing a aussi conduit les premières discussions sur l'intelligence artificielle dans les années 1950 et créé en 1952 la discipline biologique de première importance qu'est la *morphogenèse*, science des formes du vivant et de leur croissance.

4. Dont l'EPI, association
Enseignement public &
informatique,
<https://www.epi.asso.fr>.

5. Un enseignement de spécialité avait été mis en place avec succès dès 2012 dans les terminales scientifiques, mais il était resté très minoritaire.

Notes

1. Merci à Jean-Marc Lévy-Leblond pour cette belle remarque. L'ambiance devait être bien différente quand les ombres bougeaient !

2. Bien qu'il soit naturel pour un musicien moderne qui utilise beaucoup d'informatique d'« avoir l'algorithme dans la peau ».

3. Orthographe éminemment
variable dans notre alphabet...

4. Ma mère m'a légué la sienne, avec laquelle elle enseignait l'algorithmique à ses élèves de collège et de lycée dans les années 1960-1970. N'essayez pas de me la racheter, je ne m'en séparerai pour rien au monde ! Pour tout savoir sur son fonctionnement, son démontage et son remontage, le lecteur intéressé pourra consulter le site <http://www.vcalc.net/cu.htm>.

5. Avec la mode étrange des *selfies* qui consiste à se photographier soi-même, avec les monuments en arrière-plan demi-caché. Je selfie donc je suis. J'adore photographier les touristes prenant des selfies avec leurs perches !

6. C'est hélas moins vrai quand on habite comme moi la campagne française, les accès Internet étant pour l'instant loin d'avoir la même qualité que dans les villes.

7. Le mot français officiel est « bogue », mais personne ne l'utilise dans notre métier.

Notes

1. American Standard Code for Information Interchange.

2. Objet à garder précieusement pour la culture historique des enfants, car il va vite disparaître !

3. J'ai toujours adoré les expressions fleuries comme « quantité industrielle », « nombre astronomique » (Berry, *Manifeste pour la réhabilitation du Pavillon des poids et mesures*, 2007).

4. Voir le concert de mon colloque
2016 du 31 mai 2016 « Arts et
sciences, de nouveaux domaines
pour l'informatique »,
[http://www.college-de-
france.fr/site/gerard-
berry/symposium-2015-2016.htm](http://www.college-de-france.fr/site/gerard-berry/symposium-2015-2016.htm).

5. Pas tout à fait, chaque domaine particulier pouvant introduire ses propres notations. Mais le cadre général est clair, bien que pas vraiment formalisé.

6. C'est comme ça qu'on appelle les programmeurs dans le domaine.

7. Il y a aussi des bugs dans les circuits, encore une fois dus à leur conception plus qu'à leur fabrication car la qualité de leurs transistors et de leurs connexions est soigneusement testée dès leur fabrication. Mais ils restent rares par rapport à ceux des logiciels, car cette industrie est particulièrement sérieuse.

Notes

1. Pas partout ! En 1977, quand je suis arrivé à Sophia-Antipolis, j'appelais encore mes cousins du Gers en demandant à la standardiste de Toulouse : « Le 23 à Beaucaire-sur-Baïse »...

2. Je n'aime pas beaucoup ce qualificatif typique des exagérations du marketing : même s'il m'est très utile, mon smartphone me paraît à peu près aussi intelligent éteint qu'allumé. L'intelligence est dans la conception de l'objet et dans ses logiciels créés par les hommes, pas dans l'objet lui-même – au moins pour l'instant.

3. Cette inversion est merveilleusement illustrée par un dessin de Jean-Jacques Sempé, que les chanceux qui possèdent son calendrier pourront trouver à la date du 6 février.

4. De façon plus générale, sa théorie a permis de revisiter la notion classique d'entropie en la reliant précisément à celle d'information, ce qui a permis de faire entrer vraiment l'information en physique.

Notes

1. Les télécommunications adorent les sigles incompréhensibles. Personnellement, j'adore FTTH (*fiber to the home*), et j'aimerais bien en avoir une, de fibre, à la maison !

2. Pour les voitures le premier mois
et pour les camions ensuite, comme
on le rappelle en Belgique.

3. Encore un terme connu de tous et souvent prononcé, en général sans que sa signification soit comprise.

4. L'idée de construire des programmes dans un serveur puis de les envoyer à un client pour exécution et interaction avec le serveur avait déjà été implémentée dans le langage Java, très populaire et souvent utilisé sur Internet, mais qui n'a jamais été complètement intégré dans le Web et pose des problèmes de sécurité.

5. Comme les *event handlers* de JavaScript pour la gestion des événements, qui sont simples en théorie mais dont le comportement à l'exécution est très difficile à contrôler et même à comprendre.

6. Voir <http://www.college-de-france.fr/site/gerard-berry/seminar-2010-01-20-10h15.htm> et <http://www.college-de-france.fr/site/gerard-berry/seminar-2013-05-28-10h00.htm>.

7. Voir <http://www.college-de-france.fr/site/gerard-berry/seminar-2013-05-28-11h00.htm>.

8. Voir la vidéo du séminaire de François Bourdoncle donné dans mon cours de 2008, <http://www.college-de-france.fr/site/gerard-berry/seminar-2008-03-14-11h30.htm>.

9. Orthographe alternative de Googol, qui signifie 10^{100} , pour signifier que le Web allait devenir énorme.

10. Dans leur fameux article initial « The anatomy of a large-scale hypertextual Web search engine », Brin et Page écrivaient en fait $(1 - d)$ au lieu de $(1 - d)/N$, et $PR(A)$ valait N fois la probabilité de A .

11. Techniquement, l'équation définit en fait le vecteur propre d'une matrice stochastique associée à la connectivité des pages.

12. Voir le séminaire de Laurent Massoulié dans mon cours 2008 sur les algorithmes du pair à pair, <http://www.college-de-france.fr/site/gerard-berry/seminar-2008-03-14-12h00.htm>.

13. Voir le séminaire d'Anne-Marie Kermarrec dans mon cours 2010 sur la personnalisation dans les moteurs et les réseaux sociaux, <http://www.college-de-france.fr/site/gerard-berry/seminar-2010-01-20-10h50.htm>.

14. Voir par exemple
<http://www.scilogs.fr/complexites/la-puissance-de-la-blockchain/>.

15. Souvent appelés *firmwares*, car proches du matériel et donc entre *hardware* et *software*. Mais ce sont bel et bien des *softwares* maintenant qu'ils ne sont plus tout petits. Les mots ont la vie dure.

16. Ordinateurs robots utilisés
comme esclaves.

17. Voir de nombreuses références
sur la page
<https://fr.wikipedia.org/wiki/Cyberattaque>

18. Mot qui a complètement changé de sens récemment, car il désignait auparavant les programmeurs habiles et gentils : « Je t'ai hacké un joli programme qui va résoudre ton problème. »

Notes

1. Dans ces domaines où les choses vont très vite, les conférences sont bien plus importantes que les journaux.

2. Dont j'ai l'honneur de présider le Conseil scientifique et la chance de travailler avec plusieurs de ses membres.

3. Voir le séminaire de Frédéric Guichard, dans mon cours de 2008, <http://www.college-de-france.fr/site/gerard-berry/seminar-2008-03-21-11h10.htm>, ainsi que deux conférences que nous avons données au colloque sur la photographie à l'Académie des sciences en décembre 2015, <http://www.academie-sciences.fr/fr/Seances-publiques/la-photographie.html>.

4. Voir les superbes portraits de scientifiques de mon ami Didier Goupy, photographe professionnel, <http://www.didiergoupy.com/>.

5. Notre vision peut d'ailleurs être envisagée comme un magnifique exemple d'algorithmique implémentée de façon biologique, mais c'est un autre sujet.

6. ISO = International Standard Organization.

7. ASA = American Standards Association.

8. Voir la conférence de Jacqueline Belloni dans le colloque sur la photographie de l'Académie des sciences en 2015, <http://www.academie-sciences.fr/fr/Seances-publiques/la-photographie.html>.

9. <https://www.lytro.com/>.

10. Dans ce cas, on abîme volontairement l'image. Ce n'est pas entièrement nouveau, certains tireurs savaient bien le faire en utilisant des caches, mais au prix d'un travail considérable.

11. Les marques Fuji et Sigma ont adopté d'autres géométries.

12. Notons que le terme *pixel* est utilisé abusivement par le marketing des fabricants pour parler des photosites des capteurs, qui n'ont pourtant chacun qu'une couleur : il n'y a pas vraiment 24 mégapixels de couleurs arbitraires dans une image crue de 24 mégaphotosites, au contraire d'une image finale.

13. Attention, cela devient difficile avec les lampes LED dont le spectre de lumière est très haché. Certaines sont cependant construites pour éviter cela, regardez leur IRC, indice de rendu des couleurs, qui doit être supérieur à 90.

14. Température théorique à laquelle il faudrait chauffer un corps noir idéal pour qu'il émette cette lumière.

15. Le meilleur algorithme actuel a donné le grand prix Inria/Académie des sciences à son créateur, Jean-Michel Morel, mathématicien de l'université Paris-Sud. Il est implémenté dans le système français DxO Optics Pro.

16. Voir par exemple
<http://lifehacker.com/remove-people-from-your-photos-with-this-photoshop-scri-1245505649>.

17. La mise au point peut occuper à elle seule plusieurs microprocesseurs dans l'appareil.

18. Voir le séminaire de Sylvain Lefebvre dans le cours au Collège de France de Marie-Paule Cani du 6 mars 2015 (Cani, 2015), <http://www.college-de-france.fr/site/marie-paule-cani/seminar-2015-03-06-11h30.htm>.

Notes

1. Voir le cours du 3 juin 2014 de Nicholas Ayache et les séminaires associés de Jean-Paul Galmiche et Sacha Loiseau ([Ayache, 2014](#)), <http://www.college-de-france.fr/site/nicholas-ayache/course-2013-2014.htm> et <http://www.college-de-france.fr/site/nicholas-ayache/seminar-2014-06-03-18h00.htm>.

2.

<https://www.nlm.nih.gov/research/visible/>

3. Voir l'exposé « Cardiologie du futur à l'ère du patient numérique » de Michel Haïssaguerre dans le cours de Nicholas Ayache du 29 avril 2014, <http://www.college-de-france.fr/site/nicholas-ayache/seminar-2014-04-29-18h00.htm>.

4. Voir l'exposé « Chirurgie du futur guidée par l'image numérique » de Jacques Marescaux dans le cours de Nicholas Ayache du 29 avril 2014, <http://www.college-de-france.fr/site/nicholas-ayache/seminar-2014-04-29-17h30.htm>.

5. <https://www.sofa-framework.org/>.

6.

<http://news.stanford.edu/2017/01/25/artificial-intelligence-used-identify-skin-cancer/>.

7.

<http://www.incubateurpacaest.org/Actualites-projets/Therapixel-gagne-le-Digital-Mammography-Challenge>.

Notes

1. « La philosophie est écrite dans ce vaste livre qui constamment se tient ouvert devant nos yeux (je veux dire l'Univers), et on ne peut le comprendre si d'abord on n'apprend pas à connaître la langue et les caractères dans lesquels il est écrit. Or il est écrit en langue mathématique, et ses caractères sont les triangles, les cercles et autres figures géométriques, sans lesquelles il est humainement

impossible d'en comprendre un seul mot, sans lesquelles on erre vraiment dans un labyrinthe obscur. »

2. Les astronomes anglais de la Royal Society ayant fait tout ce qu'ils pouvaient pour empêcher les merveilleux chronomètres de John Harrison de prendre le dessus sur les satellites de Jupiter – lire à ce propos le superbe livre *Longitude* de Dava Sobel ([Sobel, 1998](#)).

3. Et leur apporter aussi des citrons et de la choucroute pour prévenir le redoutable scorbut !

4. La même chose commence à se produire pour les sciences sociales, dont je ne parlerai pas ici par incompétence.

5. Avec hélas des effets potentiellement délétères, comme ceux des classements des chercheurs par leur nombre de publications ou de citations, qui incitent certains à publier des papiers à peine cuits sachant qu'ils pourront être évalués ou cités sans même être vraiment scrutés et vérifiés. Cela met trop de pression sur le système, et en particulier sur les jeunes. Les institutions devraient vraiment

s'interroger sur les effets pervers de ces méthodes « algorithmiques » de recrutement ou d'évaluation.

6. Voir le beau film *Einstein et la relativité générale, une histoire singulière*, réalisé et diffusé par l'Institut Henri-Poincaré.

7. *Reine ignorée*, selon la belle anagramme d'Étienne Klein et Hubert Perry-Salkow (Klein & Perry-Salkow, 2011).

8. Voir la vidéo de Clément Narteau dans mon colloque « Arts et sciences, de nouveaux domaines pour l'informatique » du 31 mai 2016, <http://www.college-de-france.fr/site/gerard-berry/symposium-2016-05-27-14h00.htm>.

9. Merci à Jean-François Joanny de l'ESPCI (École supérieure de physique et chimie industrielle, Paris) pour cet exemple.

10. Construction dynamique des formes dans le vivant. À l'aide d'une belle modélisation mathématique, Turing a expliqué pour la première fois pourquoi les léopards ont des taches et les tigres des rayures.

11. Adénine, thymine, cytosine et guanine (remplacée pour l'ARN par l'uracile U).

12. Voir par exemple le Cancer Genome Atlas Pan-Cancer Analysis Project,

<https://www.ncbi.nlm.nih.gov/pubmed/24044246>

13.

Voir

<https://www.pasteur.fr/fr/centre-medical/fiches-maladies/autisme>.

14. « Acquisition comprimée », en français, mais je n'ai jamais vraiment vu cette expression utilisée.

15. Rendez-vous à l'adresse
<https://www.wehi.edu.au/wehi-tv/wehitv>.

16.

Voir

<https://www.youtube.com/watch?v=Pjt1Q2ZZVjA>.

17. Voir la vidéo du séminaire donné le 23 mai 2008 par François Fages dans mon cours 2008, <http://www.college-de-france.fr/site/gerard-berry/symposium-2008-05-23-10h45.htm>.

18.

<https://team.inria.fr/virtualplants/>.

[19](#). Voir l'entrée « L-systèmes » sur Wikipédia, qui montre de beaux exemples de mauvaises herbes.

20. Voir le séminaire du 9 avril 2014
« Le temps dans le calcul neuronal »
par Romain Brette (ENS Paris,
maintenant Institut de la vision),
[http://www.college-de-
france.fr/site/gerard-berry/seminar-
2014-04-09-17h00.htm](http://www.college-de-france.fr/site/gerard-berry/seminar-2014-04-09-17h00.htm).

21. Voir par exemple sur le Web les travaux de Michaël Zugaro et Sidney Wiener, que j'ai le plaisir de côtoyer au Collège de France.

22.

<https://www.humanbrainproject.eu/en/>.

23. Ça ne lui revient pas cher : ceux qui ont reçu ses chèques ont bien pris soin de les encadrer au lieu de les toucher.

24. Un polyèdre est convexe si, quels que soient deux points A et B à l'intérieur du polyèdre, tous les points du segment qui joint A et B sont aussi dans le polyèdre.

25. Il s'agit en fait d'une belle extension de l'argument diagonal qu'avait utilisé Georg Cantor longtemps avant pour montrer qu'il y avait strictement plus de nombres réels que de nombres entiers.

26. Que j'avais eu l'honneur d'avoir
comme étudiant de thèse à la fin des
années 1980.

27. Pour les amateurs, ce théorème remarquable exprime en gros que « le λ -calcul de second ordre, dans lequel les λ -termes sont typés par d'autres λ -termes qui en expriment le polymorphisme générique, est fortement normalisable ». Les mots sont compréhensibles un par un et la phrase dégage une certaine poésie, mais elle ne parle pas forcément à tous...

Notes

1.

<http://esamultimedia.esa.int/docs/esax-1819eng.pdf>.

2. Une de mes devises est : « Je ne fais jamais d'erreur. Une fois, j'ai cru en avoir fait une, mais je m'étais trompé. »

3. Centre national d'études spatiales.

4.

Voir

http://www.cse.chalmers.se/~risat/Report_

5. <http://embeddedgurus.com/barcode/2013/10/an-update-on-toyota-and-unintended-acceleration/>.

6. Voir l'exposé de Xavier Leroy dans mon cours 2007-2008, où il montre que même le minuscule programme de la factorielle peut se traduire en plusieurs pages de code machine : <http://www.college-de-france.fr/site/gerard-berry/seminar-2008-02-08-11h30.htm>.

7. Ce bug est bien décrit à l'adresse
<http://users.csc.calpoly.edu/~jdalbey/SWE>

Notes

1. En anglais, le terme *exploit* désigne l'exploitation d'une faille et pas un exploit à la française.

2. Voir par exemple les travaux de l'équipe de Véronique Cortier, CNRS/Inria Nancy.

3. Voir son séminaire dans le cours
du 31 mars 2017 de Didier Roux au
Collège de France,
[http://www.college-de-
france.fr/site/didier-roux/seminar-
2017-03-31-11h00.htm](http://www.college-de-france.fr/site/didier-roux/seminar-2017-03-31-11h00.htm).

4. Attention, on dit *regression tests* en anglais, sans négation.

5. Voir le séminaire de Laurent Maillet-Contoz (STMicroelectronics) et Matthieu Moy (Verimag Grenoble) dans mon cours 2014-2015, <http://www.college-de-france.fr/site/gerard-berry/seminar-2014-01-29-17h00.htm>.

6. Je me suis permis d'écrire « $r := r * i ;$ » au lieu de « $r = r * i ;$ », comme en C, pour que le signe « $=$ » soit réservé à l'égalité mathématique. Il faut se souvenir que C a été initialement conçu pour de toutes petites machines. Ses auteurs ont préféré remplacer les « $:=$ » et « $=$ » des langages précédents par « $=$ » et « $==$ », car le premier est bien plus fréquent que le second. Ils gagnaient

ainsi quelques caractères dans leurs programmes.

7. <http://www.envisage-project.eu/proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it/>.

8. Voir mon cours du 30 mars 2016 sur UPPAAL et l'exposé de Kim Larsen au Collège de France, <http://www.college-de-france.fr/site/gerard-berry/course-2015-2016.htm>.

9. Voir mon cours du 16 mars 2016 et le séminaire associé de Laurent Simon, <http://www.college-de-france.fr/site/gerard-berry/course-2016-03-16-16h00.htm>.

10. Voir mon cours du 23 mars 2016 et le séminaire de Sylvain Conchon, <http://www.college-de-france.fr/site/gerard-berry/course-2015-2016.htm>.

11. <https://deepspec.org/>.

12. Il faut noter que les langages fonctionnels modernes sont largement issus du système de types introduit par Milner dans son langage fonctionnel ML (la base d'OCaml) pour garantir formellement qu'il n'était plus possible de démontrer des théorèmes faux, comme on pouvait le faire en LCF.

13. Voir l'exposé de Christine Paulin dans mon cours du 18 mars 2015, <http://www.college-de-france.fr/site/gerard-berry/seminar-2015-03-18-17h30.htm>.

14. Voir la vidéo du séminaire de J.-R. Abrial dans mon cours du 1^{er} avril 2015, <http://www.college-de-france.fr/site/gerard-berry/seminar-2015-04-01-17h30.htm>.

15. Voir <http://www.event-b.org/>.

16.

Voir

<http://web1.cs.columbia.edu/~junfeng/09fa/e6998/papers/sel4.pdf>.

17. Voir la vidéo aux adresses
<http://www.college-de-france.fr/site/gerard-berry/seminar-2015-03-11-17h30.htm> et
<http://www.provenrun.com/products/provenrun>

18. <https://royalsociety.org/science-events-and-lectures/2016/11/milner-award-lecture/>.

19. Voir le cours de Martin Abadi au Collège de France ([Abadi, 2011](http://www.college-de-france.fr/site/gerard-berry/seminar-2015-03-25-17h30.htm)), le séminaire de Véronique Cortier dans mon cours du 25 mars 2015 (<http://www.college-de-france.fr/site/gerard-berry/seminar-2015-03-25-17h30.htm>) et celui de Stéphanie Delaune dans mon cours du 6 avril 2016 (<http://www.college-de-france.fr/site/gerard-berry/seminar-2016-04-06-17h30.htm>).

20. <http://www.college-de-france.fr/site/gerard-berry/course-2013-2014.htm>.

Notes

1. <http://www.societe-informatique-de-france.fr>.

2. <https://interstices.info>.

3. <http://smai.emath.fr>.

4. <http://www.epi.asso.fr/>.

5. La fameuse machine quantique D-Wave, effectivement assez grosse, a été souvent présentée à tort comme un ordinateur quantique général. Elle n'exécute en fait qu'un seul algorithme, l'optimisation par recuit simulé. C'est déjà tout à fait intéressant, mais pas magique.

6. Comme on disait au XX^e siècle, du temps des bœufs et des charrues – si vous avez une expression du XXI^e siècle équivalente, je suis preneur !

7. À condition toutefois de pouvoir mettre en place des systèmes de vote à distance suffisamment fiable, ce qui n'est ni simple ni impossible si on définit bien le problème, voir le [chapitre 10](#).

8. Il est l'un des auteurs principaux du langage Haskell et a développé pour lui un excellent compilateur – c'est aussi un orateur exceptionnel.

9. <https://project.inria.fr/classcode/>.

Notes

1. Je ne peux malheureusement pas utiliser la lettre *C* pour parler de compilateurs, car elle est le nom d'un des plus célèbres langages de programmation. Cela prêterait à trop de confusions.

2. Voir la vidéo sur
<http://www.college-de-france.fr/site/gerard-berry/course-2015-11-04-16h00.htm>.

3. Pour les initiés, LISP s'inspirait du lambda-calcul de Church, mais remplaçait sa liaison statique par la liaison dynamique. Pratique en mode interprété, la liaison dynamique était quasiment impossible à compiler et provoquait aussi des bugs très sournois. Scheme est donc repassé à la liaison statique.

Notes

1. Rédigé la nuit par Gérard Berry, cette fois en tant que Régent de Déformatique au Collège de Pataphysique. Informaticien le jour, Déformaticien la nuit. Définition : l'Informatique, c'est la science de l'information, la Déformatique, c'est le contraire. Devise empruntée à Oscar Wilde : « Appuyez-vous sur les principes, ils finiront bien par céder. »