Représentation numérique de l'information

Notre parcours commence avec la notion d'information. L'information est la matière première de l'informatique : les algorithmes, les machines, les langages sont nés de notre désir de transformer, transmettre et stocker des informations. Nous commençons par détailler la manière dont les textes, les nombres, les sons, les images, etc. s'expriment comme des informations, c'est-à-dire comme des suites de bits *(*binary digits*, chiffres binaires), 0 ou 1, qui constituent les atomes d'information. Peu à peu apparaîtra l'idée essentielle de ce chapitre : comme beaucoup d'autres grandeurs, la quantité d'information se mesure quantitativement.*

Cours

Le codage numérique de l'information

Information, hasard et complexité Observons cette suite de chiffres :

14159265358979323846264338327950288419 71

Il y a sûrement plus d'informations dans cette suite-là que dans celle-ci :

00000000000000000000000000000000000000000

qui n'est qu'une suite de zéros. Ah oui, mais en quoi la première suite de chiffres est-elle plus complexe ?

* Est-elle aléatoire, chaotique, bref sans ordre ni régularité ?
* Ou bien organisée, fortement structurée, riche en information « utile » ?

Nous pouvons porter peu ou beaucoup d'intérêt à ces symboles, leur attribuer une valeur ou pas, valeur marchande ou d'un autre ordre, trouver du sens ou non à leur enchaînement. Indépendamment de ces éléments externes, qu'entendons-nous lorsque nous parlons de son contenu en information ou de sa valeur en information ?

En fait, si nous regardons bien le début de cette suite, nous reconnaissons les premières décimales de π, peut-être le nombre le plus célèbre de toute l'histoire des mathématiques : autant dire que son information est très utile… du moins pour les géomètres et les mathématiciens !

En tout cas, il est bien compliqué de retenir les chiffres de la première suite. Il est sans doute impossible de faire autrement que de les apprendre par cœur. Mais pour la deuxième suite, il suffit de retenir : « voilà 40 zéros ».

Cet exemple est instructif, car il nous montre que nous devons bien distinguer :

* le contenu brut en information, qui ne dépend pas du sens ;
* et la valeur d'une information, qui elle dépend du but fixé.

Ce que les différents éléments de formalisation de l'information nous offrent, c'est une vue précise du contenu brut de cette notion quotidienne : nous supposons qu'il y a un contenu brut d'information pour chacun de ces objets, indépendamment des considérations précédentes.

Le bit, atome d'information Est proposé ici que ce contenu brut soit donné en bits. Pourquoi utiliser un codage binaire ?

Tout d'abord… parce qu'il faut au moins deux symboles ! Pourrions-nous utiliser moins de deux symboles ? Il semble que non, car si nous avons un seul symbole, par exemple 0, pour une longueur donnée, par exemple 100000, il n'y aurait qu'une seule suite : 000000000000000000000000… qui ne peut donc décrire qu'un seul objet ! Et non servir à les décrire tous… L'information commence donc avec la dualité – opposition et complémentarité – du 0 et du 1, du chaud et du froid, du bas et du haut… Quand on a un symbole unique, tout est uniforme. Et il n'y a pas d'information.

Ensuite, il s'avère qu'en utiliser deux est bien pratique ! Bien sûr, nous pourrions choisir d'utiliser plus de deux symboles, mais n'en utiliser que deux a plusieurs avantages. Le premier est que ce choix est indépendant de la nature des informations décrites. Il aurait été dommage de concevoir, par exemple, des ordinateurs grecs qui utilisent 24 symboles et des ordinateurs latins qui en utilisent 26.

Par ailleurs, la description de l'information avec deux symboles est plus simple à mettre en œuvre dans un ordinateur. Cela se réalise électriquement par un interrupteur ouvert ou fermé, un courant électrique présent ou absent, etc. Ce choix rend aussi les ordinateurs plus robustes : s'il y avait plus de deux valeurs, le système physique pourrait plus facilement les mélanger. Dès que les plages d'incertitude des valeurs se recouvriraient, par exemple avec la décharge de la batterie, des erreurs se créeraient. Cela minimise donc les erreurs d'interprétation en cas de perturbation du signal. C'est sans doute bien la qualité essentielle du code binaire que sa capacité à résister au bruit.

De plus, pour utiliser le moins d'énergie ou d'espace, il s'avère – mais ce n'est pas un résultat évident – que le codage binaire est le plus robuste.

Quand les informations s'ajoutent Il y a une idée encore plus profonde ici. Savoir de quelqu'un si c'est un homme ou une femme, un jeune ou un vieux, quelqu'un de grand ou petit, c'est très schématique, mais cela nous donne déjà trois atomes d'informations sur lui, trois bits. Les tailles en information de deux informations indépendantes s'ajoutent, mais pas celles de deux informations redondantes : par exemple, si nous ajoutons que cette personne est un être humain, nous ne gagnons rien, car, si c'est un homme ou une femme, c'est un être humain.

Ainsi, pour des informations indépendantes, l'information est additive. Il en découle que le nombre minimal de bits à utiliser pour coder une information est exactement la quantité brute d'information qu'elle contient.

Par exemple, la quantité d'information contenue dans une suite de symboles binaires peut se définir comme le nombre de cases mémoire que la chaîne occupe dans la mémoire d'un ordinateur, quand on ne lui fait subir aucun traitement particulier autre que la mise sous un format compatible avec le système numérique.

Autre exemple, la taille en information du choix entre plusieurs options correspond au nombre de bits pour le coder. Par exemple, choisir entre quatre couleurs : noir, rouge, vert, bleu nécessite deux bits. Choisir entre les sept couleurs que nous voyons aujourd'hui dans l'arc-en-ciel nécessite trois bits :

violet	indigo	bleu	vert	jaune	orange	rouge
'111'	'110'	'101'	'100'	'011'	'010'	'001'

tandis que le code « 000 » n'est pas utilisé ici, ce qui importe peu.

Deux théories complémentaires, la théorie *probabiliste* de l'information, due à Claude Shannon et en lien avec les concepts de la thermodynamique, et la théorie *algorithmique de l'information*, construite d'Andreï Kolmogorov – le même Andreï Nikolaïevitch Kolmogorov qui fonda la théorie moderne des probabilités – à Charles Bennett, permettent aujourd'hui de capturer et d'analyser formellement ces idées. Avec des conséquences que nous expliciterons.

L'information : une quantité universelle Mais l'idée fondamentale est que *toute information a un reflet numérique*. Avant l'ère numérique, la musique et le son étaient enregistrés sur des disques vinyles ou des bandes magnétiques, les séquences d'images sur des photos ou des films argentiques, les textes et les informations symboliques ou numériques imprimées sur papier, donc étaient sujets à des traitements physiquement différents, tandis que leur stockage, leur duplication, leur transmission étaient le fait d'actions distinctes.

Avec l'ère numérique, tout ce que nous considérons être une ou « de l'information », au sens « humain » du terme, a un *codage numérique*. Nous allons le détailler dans ce chapitre.

Cela concerne la quasi-totalité de nos informations sensorielles : la vue, avec les caméras et les écrans comme capteurs et effecteurs, le son, avec les microphones et écouteurs et haut-parleurs comme capteurs et effecteurs, le sens de l'équilibre, que nous oublions souvent de compter parmi les sens, mais que les systèmes de réalité virtuelle actifs intègrent en simulant des mouvements perçus par notre système vestibulaire, et partiellement le toucher, à travers des capteurs de pression qui permettent par exemple de « toucher » une image, et aussi l'odorat et le goût – des capteurs d'odeurs ou de goût, dits « nez électroniques » sont expérimentés –, même si ces sens restent encore moins bien compris.

Cela concerne aussi ce qui est au-delà des sens : on qualifie d'information humaine toute donnée pertinente que le système nerveux central est capable d'interpréter pour se construire une représentation de notre environnement et pour interagir correctement avec lui. On rejoint ici le sens étymologique de l'information : *ce qui prend du sens*, c'est-à-dire donne une forme à l'esprit. Le mot vient du verbe latin *informare*, qui signifie « se former une idée de ». L'information est donc immatérielle.

Mais, pour « exister », elle doit être consignée, directement ou pas, sur un support matériel qui prend alors la valeur de document. Le *codage numérique* est cette opération fondamentale par laquelle on consigne l'information pour pouvoir la manipuler.

Nous verrons alors que cette numérisation de l'information, qui entraîne sa discrétisation et sa représentation par des suites de bits, 0 ou 1, a pour conséquence de permettre de la traiter, de la transformer, de l'analyser à travers des algorithmes génériques insensibles à la nature et au contenu de l'information, car ne voyant que les bits 0 ou 1 qui la composent, ce qui rend leur usage universel. D'autres algorithmes spécifiques du contenu en information seront aussi détaillés.

Puis, selon le sens que nous donnons au mot *information*, il faut formaliser le concept différemment. Pour fixer ce concept, nous adopterons la procédure suivante : fixons un but, puis déterminons la valeur de l'information, relativement à ce but. Nous allons découvrir que, en précisant le but, on obtient différentes théories, dont la théorie probabiliste de l'information et la théorie algorithmique de l'information détaillées ici.

Le codage de l'information : un travail de standardisation et de pertinence

Deux dernières idées clés méritent d'être soulignées ici. Dans l'exemple ci-avant du codage des sept couleurs, le choix des bits est tel que l'ordre de l'arc-

en-ciel a été respecté. Ce serait pratique, si nous pensions que cet ordre a de l'importance. Mais ce n'était pas obligé.

Quel est le premier point clé ? *Convenir d'un standard pour toutes et tous.* Bien entendu, il faut que nous soyons tous d'accord pour coder les couleurs de la même façon ! Décider, comme nous l'avons fait dans l'exemple précédent, de choisir une correspondance « universelle » entre l'objet et son codage numérique. Le codage de l'information est avant tout une affaire de… convention.

Quel est le second point clé ? *Selon le codage choisi, l'information se manipule, selon les cas, plus ou moins facilement.* Si nous nous référons à l'arc-en-ciel, c'est parfait. Mais si nous sommes dans le cas où les couleurs primaires sont le rouge, vert et bleu, il était plus parlant d'attribuer un bit R, V, B pour chaque couleur primaire – rouge eût été 100, et jaune 011 comme combinaison de vert et bleu –, tandis que ce sont d'autres couleurs que nous eussions codées ici.

Le codage de l'information est en lien profond et étroit avec la manipulation de cette information.

C'est de ce travail de codage standard et, de fait, pertinent que nous allons rendre compte ici.

Codage numérique du texte

Un domaine où les conventions sont la base du décodage est le codage des lettres. Les premiers codages standardisés sont les codes du « Télégraphe de Chappe », au XVIII^e^ siècle, du « Morse » et du « Baudot », au XIX^e^ siècle. Pour ce qui est de l'informatique, c'est l'ASCII (*American Standard Code for Information Interchange*, code américain normalisé pour l'échange d'information) dont les premières versions datent du début des années 60. Le but est d'échanger du texte de façon portable entre deux machines. Le principe est assez simple et consiste à associer à chaque lettre ou chiffre un entier entre 0 et 127, donc représentable sur 7 bits. Ainsi, le chiffre « 0 » correspond à 48, la lettre « A » à 65 et la lettre « a » à 97. On peut représenter les chiffres, les lettres latines majuscules et minuscules et les principaux symboles de ponctuation.

De plus, ce codage doit être compris par tous : il doit être normalisé et connu afin qu'il n'y ait pas un codage des lettres pour chaque pays, qui nécessiterait une traduction lettre à lettre. Il existe des organismes de standardisation : l'ISO (*International Organization for Standardization*) est international et l'AFNOR (Association française de normalisation) est l'organisme français. Les standards sont utiles dans les domaines industriels et économiques, mais également pour l'interopérabilité dans le domaine informatique.

Pour représenter un texte anglais ou un programme informatique, l'ASCII suffit, mais, pour les autres langues, cela ne suffit plus. Bien plus tard ont été

définis des codages pour les autres langues. On peut citer l'ISO 8859-1 dans les années 1980, aussi appelé « *latin-1* », qui permet de représenter nos caractères accentués. La lettre est associée à une valeur entre 0 et 255, donc représentable sur 8 bits, et est compatible avec l'ASCII. Cela signifie que les nombres entre 0 et 127 correspondent aux mêmes lettres en latin-1 qu'en ASCII. D'autres codages ont été définis pour les autres langues : le chinois, le japonais… Ces codages sont compatibles avec l'ASCII mais ne le sont pas entre eux. En conséquence, comment écrire un texte multilingue, comme un dictionnaire ?

Une idée naturelle est de créer un unique codage pour tous les caractères existants. C'est l'idée d'*Unicode*. Initialement, les valeurs associées étaient entre 0 et 65535, mais cette plage, que l'on croyait suffisante, s'est révélée trop petite pour représenter l'ensemble des caractères de l'ensemble des langues. Ce codage est compatible avec le latin-1, et donc l'ASCII, mais souffre de défauts. Par exemple, le é peut se représenter soit par le caractère « é » du latin-1 soit par la suite de caractères « ’e » et cette redondance rend la comparaison difficile. D'autre part, le symbole physique pour l'ohm Ω n'est pas le même que le ω majuscule, même s'ils sont indiscernables à l'œil.

Pour représenter les caractères unicode se sont construits des codages plus ou moins compliqués tel que UTF-8 avec l'idée suivante : caractère → codage unicode → codage UTF-8. L'idée d'UTF-8 est d'être compatible avec l'ASCII, mais sans stocker exactement la valeur de codage en unicode. Les détails techniques se trouvent sur `http://fr.wikipedia.org/wiki/UTF-8` et sont assez élégants. Ainsi, si l'on récupère un flux d'octets correspondant à des caractères codés en UTF-8, on peut s'y repérer car on reconnaît facilement le premier octet correspondant à un caractère, même si ce caractère est composé de plusieurs octets, quatre au maximum.

Un texte n'est pas toujours suffisant. On veut parfois représenter une lettre, un article, un livre qui contiennent à la fois le texte et sa structure. Cela peut être de façon basique uniquement la mise en pages du texte. Tels mots sont centrés, en gras et de taille 14 comme dans les suites OpenOffice ou Microsoft Office. Ce livre a été écrit en LATEX où la structure du texte est l'élément important. Ensuite, chaque élément de structure – les items d'une liste, les titres de paragraphes – est affiché de la même façon, modifiable par l'utilisateur.

Codage numérique des nombres

Au contraire du codage des caractères, le codage des valeurs numériques n'est pas un code arbitraire standard, mais est issu de l'arithmétique. En effet, 2741 pourrait être la succession des caractères « 2 », « 7 », « 4 » et « 1 ». Mais ce codage ne permet pas d'effectuer facilement des calculs arithmétiques tels que

l'addition ou la multiplication. Pour obtenir une représentation utilisable dans des calculs, il suffit cependant de passer de la numération à base 10 classique à la numération à base 2 de manière à n'avoir que des bits.

Nous pouvons représenter des nombres entiers positifs par la suite de bits de leur représentation binaire. Ce codage est très précieux, car il permet de faire directement des opérations sur ces nombres. Par exemple, ajouter deux entiers positifs revient à ajouter leur représentation binaire : on obtient directement le codage binaire du résultat. De même pour toutes les autres opérations numériques, pour comparer ces entiers entre eux, etc. Souvenons-nous que toutes les valeurs numériques sont non seulement stockées en binaire, mais aussi manipulées en binaire dans les ordinateurs. À une valeur abstraite de nombre entier, on associe donc un ensemble de bits stockés dans la mémoire qui correspond à la présence ou l'absence d'une tension, comme nous le verrons au quatrième chapitre. On peut ensuite raisonner sur ces nombres, mais en interne il n'y a que des portes logiques réalisées par des circuits électroniques.

Nous allons utiliser une notation binaire, ce qui consiste ici à utiliser les chiffres 0 et 1 et à utiliser la base 2 comme base de numération. Cela signifie qu'un nombre binaire sera noté $a_n a_{n-1} \ldots a_0$ comme par exemple 100010101_2, l'indice 2 indiquant la base dans laquelle le nombre est représenté. Ce nombre a pour valeur l'entier $a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \ldots + a_1 \cdot 2 + a_0 = \sum^n a_i \cdot 2^i$.

Pour l'exemple $100010101_2 = 1 \cdot 2^8 + 0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 = 256 + 16 + 4 + 1 = 277$.

Il est difficile de considérer n arbitrairement grand. En général, on a $n = 32$, ce qui permet de représenter les entiers jusqu'à 4 milliards environ, ou $n = 64$, ce qui permet d'aller jusqu'à $1{,}8 \times 10^{20}$, qui sont les valeurs par défaut de stockage et de calcul des ordinateurs.

Comment passer du codage usuel décimal au codage binaire ? Pour écrire le nombre décimal n en binaire, on utilise l'algorithme suivant :

1. Si $n \,\%\, 2 = 0$ (reste de la division entière), écrire 0, sinon écrire 1, à gauche de ce qui est déjà écrit.
2. Remplacer n par $n \div 2$ (division entière).
3. Si $n = 0$, on a fini, sinon on retourne au point 1.

Considérons le déroulement de cet algorithme sur $n = 11$

n	11	5	2	1	0
$n \,\%\, 2$	1	1	0	1	
	1	11	011	1011	

Pour écrire le nombre binaire n_2 en décimal, on utilise l'algorithme suivant :

1. On pose $x = 1$ (cette variable contiendra, à chaque étape, la valeur 2^i) et $r = 0$ (cette variable accumulera le résultat).

2. Si $n_2 = 0$, rendre le résultat r.
3. Si n_2 finit par un 1, remplacer r par $r + x$.
4. Remplacer n_2 par $n_2 \div 2$ (ce qui revient à effacer son dernier chiffre).
5. Remplacer x par $2 \times x$.
6. Revenir au point 2.

Considérons le déroulement de cet algorithme sur $n_2 = 100011$.

n	100011	100011	10001	1000	100	10	1	0
r	0	1	3	3	3	3	35	35
x	1	2	4	8	16	32	64	

Comment calculer sur ces nombres ? De façon naturelle, comme nous en avons l'habitude, mais en base 2. Nous posons les opérations de façon identique.

Ainsi

	1	0	0	0	1	0	1	0	1	02	=	554
+				1	1	1	1	0	0	12	=	121
	1	0	1	0	1	0	0	0	1	12	=	675

Une opération est devenue triviale : la multiplication par 2. Elle consiste à ajouter un 0 à la droite de la représentation binaire du nombre : ainsi $277 \times 2 = 100010101_2 \times 2 = 1000101010_2 = 554$.

Entiers signés Considérons maintenant également des entiers négatifs. Une solution est que le premier bit soit considéré comme un bit de signe et le reste des bits comme la valeur absolue exprimée comme ci-avant. Mais cela a deux inconvénients : le premier est l'existence de deux zéros, l'un positif et l'autre négatif, dont il faudra gérer l'égalité. Le second inconvénient est le fait que l'on perde une valeur représentée : avec n bits, on ne peut représenter les valeurs que de $-2^{n-1} + 1$ à $2^{n-1} - 1$.

La solution choisie est proche et s'appelle le *complément à 2*. Il est nécessaire de fixer la valeur de n : la représentation de -1 dépend du nombre de bits choisi. Pour un nombre binaire en complément à 2 sur n bits, on sépare le premier bit s et les $n - 1$ bits suivants. Ces $n - 1$ bits représentent, en binaire usuel, une valeur positive v. La valeur de ce nombre est alors v si $s = 0$ et la valeur est $-2^{n-1} + v$ si $s = 1$.

Pour $n = 4$, cela donne les valeurs suivantes :

0000	0001	0010	...	0101	0110	0111	1000	1001	...	1101	1110	1111
0	1	2	...	5	6	7	- 8	- 7	...	- 3	- 2	- 1

Les valeurs possibles stockées vont de -2^{n-1} à $2^{n-1} - 1$: il n'y a pas de valeur perdue, ni de représentations multiples de 0. On garde également un discriminant facile du signe : les valeurs strictement négatives ont le premier bit à 1 et les valeurs positives ou nulles leur premier bit à 0, ce qui est très utile pour savoir si une opération sera une addition ou une soustraction entre les valeurs absolues des nombres.

Nombres réels Après les nombres entiers viennent naturellement les nombres réels ou du moins leur approximation dans les machines. L'ordinateur n'ayant qu'une mémoire finie, il ne peut stocker tous les chiffres du résultat exact d'un calcul comme $\sqrt{2}$ ou 1/3. Il a une représentation interne d'une partie des réels en « notation scientifique », ce qui signifie que l'on garde un exposant et un nombre fixé de chiffres. On appelle cela un nombre à virgule flottante. Ces nombres, ainsi que les calculs, sont régis par un standard, l'IEEE-754, qui définit la répartition et l'usage des bits d'un nombre. Les formats usuels sont la simple précision (`float` en Java ou en C) sur 32 bits et la double précision (`double` en Java ou en C) sur 64 bits. Des formats étendus, avec davantage de bits pour l'exposant et/ou la fraction, existent souvent, mais ne sont pas toujours accessibles au programmeur.

Exemple d'un nombre à virgule flottante simple précision :

11100011010010011110000111000000

1		11000110		10010011110000111000000
s		e		f
signe		exposant		fraction
↓		↓		↓
$(-1)^s$	×	2^{e-B}	×	$1 \bullet f$
$(-1)^1$	×	$2^{198-127}$	×	$1.10010011110000111000000_2$

$$-2^{54} \times 206727 \approx -3{,}7 \times 10^{21}$$

Pour les valeurs spéciales de l'exposant, minimale et maximale, on a des valeurs exceptionnelles : ±∞, nombres plus petits appelés dénormalisés, ±0 et NaN (*Not-a-Number*). Nous n'entrerons pas dans les détails ici, voir les explications et liens depuis `http://fr.wikipedia.org/wiki/IEEE-754` pour aller plus loin.

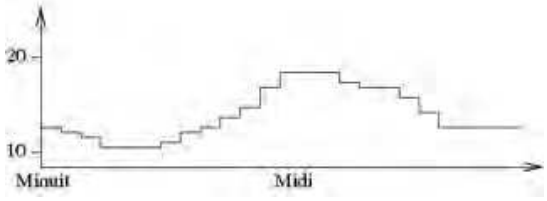
Avec ces nombres et ces valeurs, on peut effectuer des opérations. Chaque opération, addition, soustraction, multiplication, division, racine carrée, est parfaite : on a le même résultat que si l'on avait calculé avec une précision infinie et arrondi ensuite au format choisi. C'est une propriété très puissante qui permet de raisonner sur chaque étape de calcul et garantit une bonne précision

sur un calcul unique. En pratique, cependant, on fait de très nombreux calculs et il devient difficile de garantir la correction du résultat final. On peut résumer cela en disant que les ordinateurs calculent vite, mais faux.

Codage numérique des objets

Maintenant que nous disposons de la possibilité de coder des nombres, nous allons voir comment coder des objets numériques, c'est-à-dire créer un reflet numérique d'objets physiques, d'images, de sons, etc. dont l'information va être approximée par une suite de nombres. Tout le détail du codage des données usuelles de l'informatique est très bien décrit dans Wikipédia. Nous allons nous concentrer ici sur les points clés.

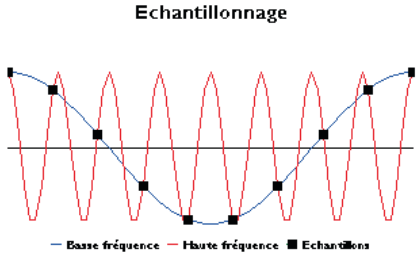
Coder une courbe ou un son Par exemple, comment coder la courbe de température de sa chambre au fil du jour ? On peut coder le nombre qui correspond à la mesure de température à chaque heure et les mettre bout à bout.



Ces 24 nombres vont donner une bonne approximation du chaud et froid, et du fait que… le chauffage est visiblement coupé la nuit… brrrr. Ou alors, c'est que la température extérieure est beaucoup descendue, et que l'isolation doit être revue.

Cette courbe n'est qu'une suite de codages binaires de nombres. Et le codage de chaque nombre a été discuté. Qu'en est-il du codage de la courbe elle-même ? Combien faut-il d'échantillons par seconde ? On comprend bien qu'une valeur à chaque heure suffit si le signal varie peu, mais non s'il varie beaucoup. Dans ce cas, quelle doit être la fréquence des relevés ? Le théorème de Nyquist-Shannon donne une solution définitive et simplissime : *il faut au moins deux points par oscillation du signal pour reproduire fidèlement sa dynamique lors du codage numérique*. Nous savons que chaque signal temporel contient des variations, lentes et rapides, de différentes fréquences que sa transformée de Fourier permet de calculer. Pour qu'un signal soit reproduit fidèlement lors de son échantillonnage – on dit aussi sa conversion analogique-numérique –, la fréquence d'échantillonnage doit être supérieure au double de la plus haute fréquence contenue dans le signal.

Si le codage ne respecte pas cette règle, il va y avoir un effet de crénelage, aussi appelé repli de spectre. Cela indique que l'on prend une sinusoïde pour une autre.



Il faut donc filtrer les hautes fréquences avant d'échantillonner un signal.

Par exemple, un son peut être représenté par une courbe qui correspond aux vibrations de la pression de l'air. C'est en effet le phénomène physique qui correspond au son de la voix, des bruits ou des instruments. Coder un son revient à coder une suite de valeurs numériques qui correspondent à cette courbe. Bien sûr, il faut beaucoup de valeurs : environ 44000 par seconde, pour obtenir une bonne approximation de toutes les vibrations sonores. L'oreille humaine entend les sons, au mieux, jusqu'à une fréquence aiguë de 22000 Hz (hertz, c'est-à-dire oscillations par seconde). Les notes de musique s'étalent d'environ 32 Hz pour le *do* le plus grave à 8000 Hz pour les notes les plus aiguës, tandis que des percussions génèrent des notes à des fréquences plus graves.

Il faut 16 bits pour coder correctement la valeur de l'intensité du son à chaque instant, donc à 0.0015 % près. Et, pour coder une minute de son, il faut : 16 bits × 40000 valeurs/seconde × 60 secondes = presque 40 millions de bits. C'est ce qui se passe dans les lecteurs MP3 par exemple, à une différence importante près. En effet, le codage de son avec le standard MP3 est « astucieux » : il met en œuvre des méthodes de compression qui tiennent compte de la perception humaine : elles ne reproduisent pas fidèlement le signal physique, mais les défauts sont inaudibles. Cette approximation supplémentaire économise de la place mémoire et du temps lors de la communication. Nous en parlerons plus loin.

Dans la mémoire d'un ordinateur, ce signal est stocké de manière générique par un tableau monodimensionnel de nombres.

Toutes les autres mesures physiques effectuées au cours du temps partagent la même problématique.

Coder une image Il est bien connu que l'on découpe une image en points élémentaires, ou « pixels » (*picture elements*), et attribue une couleur à chaque pixel, pour en faire un objet numérique. La couleur est représentée par un index. Une table de valeurs permet d'associer cet index aux valeurs numériques correspondant aux intensités lumineuses rouge, vert et bleu choisies : on parle de « pa-

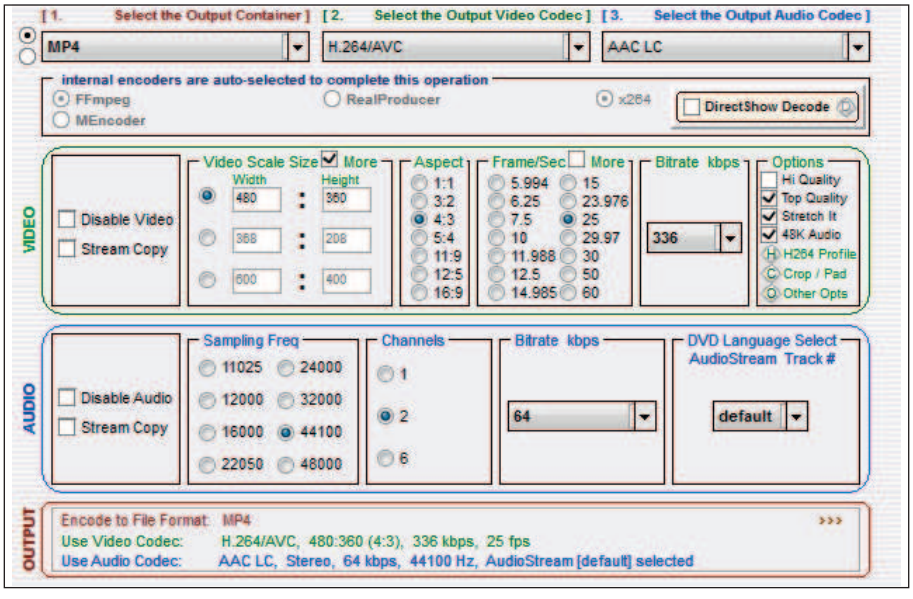
lette ». C'est une activité pédagogique fondamentale que nous proposons dans la partie didactique, que de travailler en détail sur ce codage qui reste bien intuitif.

Le codage le plus simple des couleurs est le codage « RGB » (Rouge-Vert-Bleu), sur trois canaux, la valeur codée entre 0 et 255, donc sur 8 bits, correspondant au taux de saturation en rouge, en vert, en bleu. Cela donne 3 × 8 bits par pixel. On peut stocker directement ce nombre, ou une « palette » de 256 couleurs qui permet de choisir parmi les 256^3 couleurs. Un quatrième nombre, dit *alpha*, peut aussi coder le pourcentage d'opacité, une valeur 0 rend le pixel invisible puisque complètement transparent et une valeur 255 le rend complètement opaque. On consultera les éléments de Wikipédia pour plus de détails et pour les détails de mise en œuvre.

Au-delà du format de stockage dans un fichier, les données de l'image, une fois décompressées, sont codées dans la mémoire de l'ordinateur par un tableau bidimensionnel de nombres correspondants aux pixels.

Coder une vidéo Une vidéo n'est jamais qu'une séquence d'images. Schématiquement, il suffit donc de coder les images et de mettre ces suites de bits bout à bout. Sans oublier de coder aussi le son. Là encore, il y a des astuces pour regrouper les parties qui se ressemblent et gagner de la place et du temps. Mais le principe reste le même.

La complexité vient du fait que le nombre de paramètres devient assez important, comme le montre ce panneau de paramètres du logiciel `Super`:



On doit choisir :
– le format conteneur qui contient des flux audio et vidéo respectant une quelconque norme, ici MP4 ;
– les codecs vidéo et audio qui décrivent les procédés capables de compresser ou décompresser les données dans un format normalisé ;
– l'encodeur qui est le nom de la bibliothèque logicielle qui peut effectuer ces opérations.
– Pour la vidéo, on doit encore choisir parmi les tailles d'images disponibles en pixels ; le ratio d'aspect de l'image réelle, qui est différent car les pixels ne sont pas forcément carrés, le nombre d'images par seconde, et le débit en nombre de kilo-bits par seconde, ce qui détermine le taux de compression, un débit faible conduisant à des images compressées de qualité réduite, tandis que d'autres options sont regroupées pour être sélectionnées globalement.
– Pour l'audio, on doit encore choisir la fréquence d'échantillonnage, le nombre de canaux, le son monophonique ou stéréophonique… et le débit.

Dans la mémoire de l'ordinateur les flux audio et vidéo sont séparés, ce dernier stocké de manière générique sous forme de liste d'images.

Qu'avons-nous découvert finalement ici ? La polyvalence des ordinateurs et des autres machines numériques : ils traitent des données très diverses mais toujours décrites de la même manière par des suites binaires, utilisant deux symboles.

Codage symbolique des valeurs

Au paragraphe précédent, nous avons vu comment approximer un objet numérique, coder un son ou une image. Coder un son, certes, mais pas une partition de musique. Une image, mais pas un dessin. Discutons la différence : une partition de musique ne représente pas tous les sons possibles – il est, par exemple, impossible de représenter le bruit d'un camion en duo avec le barrissement d'un éléphant. Elle se situe à un autre niveau de représentation et ne code que certains sons : les airs de musique. La notation musicale permet de définir une suite de symboles qui code la hauteur, aiguë ou grave, la durée, l'intensité, le timbre – le choix de l'instrument… Ensuite, en mettant bout à bout ces codes, nous obtenons la suite des notes. On code ainsi la partition musicale et son interprétation. Un tel codage existe, il est standard et se nomme MIDI (*Musical Instrument Digital Interface*). C'est une représentation *symbolique* de l'information.

Au-delà de cet exemple, détaillons cette façon de coder l'information.

Chaînes de caractères et noms de paramètres Si nous voulons "ah oui, relire la phrase lentement va aider à piger" coder une chaîne de caractères (`String`)

quelconque avec des mots quelconques, nous avons besoin de délimiter le début et la fin de la chaîne de caractère. Dans l'exemple qui s'est glissé dans la phrase précédente : `"ah oui, relire la phrase lentement va aider à piger"` est la chaîne de caractères à délimiter. Le premier guillemet délimite le début de la chaîne et le second en délimite la fin, tandis que chaque caractère est à coder en UTF-8, comme discuté précédemment.

On pourrait noter cette spécification de la manière suivante :

```
String:"[^"]*"
```

que nous ne manquerons pas d'expliquer dans quelques lignes.

Et si quelqu'un souhaite utiliser des guillemets, les caractères « » sont à sa disposition.

Définir un identificateur S'il s'agit maintenant de nommer un paramètre, d'identifier une variable, comme « *x* » en mathématiques, ou de référencer la propriété d'un objet, il ne faut utiliser qu'un sous-ensemble précis de caractères. Typiquement, un identificateur est une chaîne de caractères qui doit commencer par une lettre, ne contenir que des lettres ou des chiffres ou un caractère de séparation, par exemple « _ », tel que `mon_fichier, ou question_2,` etc. sans aucun caractère bizarre de ponctuation.

Selon le contexte, il vaut mieux en plus (1) ne pas utiliser d'accent, par exemple, dans les programmes, (2) ne pas mettre d'espace, pour éviter de laisser croire qu'il y a deux identificateurs `mon` et `fichier` au lieu de `mon_fichier`

Les majuscules sont à utiliser comme en français : nous les avons négligées ici pour ne pas alourdir le propos.

Lorsque, sur un site web, nous entrons notre « login » – le nom qui nous identifie –, selon les systèmes nous n'avons droit qu'à certains caractères, un peu comme ici. Pour choisir notre adresse électronique qui nous permet de recevoir et envoyer des courriels, nous avons le droit aux lettres non accentuées et aux chiffres, et au « . », « - » et « _ » comme séparateur – en fait, il nous est recommandé de toujours la définir explicitement sous la forme *prenom.nom@..*, comme par exemple `jean-sebastien.bach@somewhere.de` –, tandis que minuscule/majuscule ne sont pas prises en compte : tout est mis en minuscule. Il y a bien sûr une standardisation officielle, la RFC3696, mais de nombreuses applications n'acceptent pas l'ensemble des adresses valides ; il vaut donc mieux s'en tenir à un usage minimal.

Préciser de quel type est une valeur *La notion d'expression régulière*

On pourrait noter cette spécification de la manière suivante :

```
identifier: [a-z][a-z0-9_]*
```

qui se lit de la manière suivante :

`[a-z]` est une construction qui signifie : un caractère de `a` à `z`.

`[a-z0-9_]` est une construction qui signifie : un caractère de `a` à `z` ou de `0` à `9` ou le trait d'union _
`S*`, où `S` est une construction, signifie : `S` répétée 0 fois ou plus.

Donc, ici, `[a-z0-9_]*` signifie : une lettre, chiffre ou séparateur, répété 0 fois ou plus.

Nous découvrons là une façon formelle de décrire de quel type de suite de caractères il s'agit.

Par exemple, sachant que `[^"]` signifie : tous les caractères sauf `"`, nous voyons alors que `"[^"]*"` signifie : « le caractère ", suivi de tous les caractères sauf " 0 fois ou plus, c'est le sens de `[^"]*`, suivi du caractère " » : nous décrivons bien la `String` introduite précédemment.

Quel est l'intérêt de décrire *formellement* ce qu'est une chaîne de caractères ou un identificateur ? Pour être sûr que nous nous comprenons, jusqu'au moindre détail. Être précis est une bonne chose. Mais surtout, pour permettre à un *mécanisme algorithmique* de vérifier automatiquement si ce que nous donnons comme suite de caractères est une chaîne ou un identificateur *bien formé*. Ce mécanisme symbolique vérifie qu'il n'y a pas d'incohérence syntaxique. Cela ne veut pas dire que ce que nous allons spécifier est *valide*, c'est-à-dire constituera des données cohérentes, qui pourront être traitées, mais que les confusions syntaxiques ont été évitées.

Cette notion d'expression régulière est puissante en pratique et importante pour la formalisation des langages informatiques.

Définir un booléen. Une valeur binaire vraie ou fausse à deux valeurs se spécifie, en général :

```
boolean : false | true
```

où
`S | S'`, où `S` et `S'` sont deux constructions, signifie : soit `S`, soit `S'`.

Utiliser un `boolean` n'est qu'une façon plus « lisible » de dire « 0 » ou « 1 » en rendant explicite la sémantique vrai/faux de ce codage.

Définir une énumération. S'agit-il de définir par exemple une note de musique, sans altération, de la gamme occidentale ou une des sept couleurs de l'arc-en-ciel telles que nous les énumérons en Occident, pour éviter que la valeur soit mal définie, nous voilà conduits à définir d'autres types, comme par exemple

```
note : (do | re | mi | fa | sol | la | si)
```

ou

```
couleur : (rouge | orange | jaune | vert | bleu | indigo
| violet)
```

pour énumérer explicitement les valeurs possibles que prend une variable de ce type.

Là encore, ce n'est qu'une façon plus « lisible » d'attribuer un numéro à chaque item de cette énumération : tout se code en binaire, finalement. Mais ce serait juste indépétrable pour nous de ne pas substituer aux codes binaires bruts les corpus de mots qui désignent les objets réels dont nous codons un reflet numérique ici.

En pratique, deux types d'énumérations seront considérés :

– les énumérations *figées*, par exemple les noms des notes de la gamme, il y en a sept et cela n'aurait pas de sens dans la musique actuelle de créer un huitième nom. De même, si quelqu'un crée en plus de `false|true` une troisième valeur `bof`, nous sortons de la logique usuelle,

– les énumérations *adaptatives*, où en revanche cela a du sens de permettre d'ajouter de nouvelles valeurs, par exemple pour les couleurs, quelqu'un pourra ajouter `mordoré` ou `bleu_vert` pour désigner des couleurs intermédiaires de l'arc-en-ciel. Le système permettra alors d'ajouter *ponctuellement* une autre valeur – du coup, les valeurs de l'énumération ne servent que de valeurs exemples ou de valeurs par défaut – ou de *thésauriser* la nouvelle valeur en la gardant en mémoire pour servir d'exemple ultérieurement. C'est typiquement le cas d'un mot-clé qui sert à indexer un document : on donne une liste initiale qui s'enrichit au fil du temps.

Définir un nombre. De la même manière un nombre entier se spécifie :

```
integer : [-+]?[0-9]+
```

où

`S ?`, où `S` est une construction, signifie : `S` répétée 0 fois ou 1 fois, et
`S+`, où `S` est une construction, signifie : `S` répétée une fois ou plus.

La définition se lit « un signe - ou un + optionnel, c'est le sens de `[-+]?`, suivi d'un chiffre une fois ou plus ». Un nombre décimal se spécifiera :

```
float : $integer([.][0-9]+)?(e$integer)?
```

où `([.][0-9]+)?` correspond à l'ajout optionnel de décimales et `(e$integer)?` correspond à l'ajout optionnel d'un exposant, par exemple `314e-2 = 3.14`. Nous sommes ici en base 10 puisque nous spécifions comment un utilisateur va définir un nombre. Ici il n'y a pas de notion de précision machine, puisque le nombre est défini sous forme de chaîne de caractères : il n'y a pas un nombre limite de chiffres ou de décimales. C'est au moment où un calcul va s'effectuer que les limites entreraient en jeu.

Ce que nous gagnons ici, c'est que si la chaîne de caractères est bien formée, c'est-à-dire respecte la syntaxe donnée, *il y a bien un nombre qui correspond à cette chaîne de caractères* : ce n'est pas n'importe quoi.

La notion de type. Un concept important émerge ici. Les valeurs des variables ne sont pas n'importe quoi, elles ont un *type*. Le paramètre `age_du_capitaine` sera du type entier positif – voire, pour raffiner et détecter des incohérences sur sa valeur, un entier positif inférieur à 200. La définition d'une `gamme` musicale sera sûrement quelque chose comme `(do | re | mi | fa | sol | la | si) (mineur | majeur)` si nous oublions dièses et bémols. En écrivant :

```
age_capitaine : positive_integer
```

nous déclarons que la variable de nom `age_capitaine` est de type `positive_integer` et quelles que soient les variantes syntaxiques. Certains langages préfixeront le type, en écrivant plutôt `positive_integer age_capitaine;` ou même `positive_integer age_capitaine = 59;` pour donner un type et une valeur par défaut à la variable.

Nous devons bien garder à l'esprit que toute valeur se stocke dans une variable qui est référencée par une étiquette et qui est *définie par son type* – comme les « ensembles de définition » en mathématiques, qui permettent de bien définir les fonctions.

La bonne nouvelle est que les chaînes de caractères `String`, les nombres entiers `integer` ou flottants `float` et les booléens `boolean` ou les énumérations `enumeration` sont les cinq types de base des spécifications symboliques que nous rencontrons en pratique. Le reste correspond à des données structurées que nous présentons dans le sixième chapitre.

Quantifier l'information

Notion algorithmique d'information incompressible

Est riche en information un message… complexe à calculer.

Complexité du contenu en information Si le but est de pouvoir fabriquer/reconstituer une suite binaire *s* correspondant au codage d'un objet numérique – nous savons désormais que *tous* les objets numériques se codent sous la forme de suite binaire – et si nous supposons que nous disposons pour cela d'une machine *M* qui produit une suite binaire correspondant à un objet numérique, alors une vision naturelle de la valeur de l'information de *s* est *la longueur du plus petit programme écrit en binaire qui, lorsqu'on le donne à* M, *lui permet de reconstituer la chaîne* s.

Expliquons cette notion. Si le contenu est pauvre en information, par exemple 0000… avec un milliard de 0, alors nous nous attendons à ce qu'un tout petit programme, dans cet exemple « `écrire` 10^9 `fois "0"` », génère *s*. En revanche, s'il existait un contenu *s* tellement aléatoire qu'aucun programme

au monde ne puisse le prédire, alors le programme naturel pour le prédire est « `écrire "s"` », c'est-à-dire un programme où *s* est mémorisé explicitement avec l'instruction pour le produire sur la machine *M*.

On positionne donc la notion d'information par rapport à la *taille* du plus petit programme qui peut la produire. Cette mesure est bornée par l'ordre de grandeur de la taille de la suite elle-même, c'est, au pire, « `écrire "s"` ». La valeur en information est en quelque sorte le *contenu incompressible* de *s*. On parle de compression sans perte, puisque l'on remplace la chaîne *s* par quelque chose de plus court, le plus petit programme qui la génère.

Cette notion semble anecdotique, puisque liée à la machine *M*. C'est là qu'un résultat important intervient. Certes, les machines *M*, *M'* peuvent aller plus ou moins vite, mais, dès que l'on a affaire à une machine d'une certaine puissance, dès qu'elle exécute les mécanismes algorithmiques de base, alors elle peut exécuter « tous les algorithmes du monde ». Donc, tous les algorithmes qu'une *autre* machine peut calculer. Et réciproquement, évidemment. Donc, ce qu'elles peuvent calculer est le maximum de ce que n'importe quelle machine puissante peut calculer. Un téléphone portable, pourvu que son processeur soit un processeur standard, a la même potentialité de calcul que le plus grand super-calculateur du monde – en beaucoup moins rapide. C'est là la découverte fondamentale de Turing en 1936 : il y a des mécanismes universels de calcul et n'importe quel processeur de système numérique – ordinateur, smartphone, téléviseur numérique… – constitue un tel mécanisme universel.

En conséquence, pourvu que l'on se donne un mécanisme de calcul universel, l'ordre de grandeur du plus petit programme pouvant engendrer *s* ne dépend pas de la machine universelle que l'on utilise.

En fait, il ne dépend de cette machine que de la manière suivante. Prenons une machine *M* et écrivons le programme *P* qui simule une autre machine *M'*. Ce programme *P* prend chaque instruction de la machine *M'* et le traduit en l'instruction de la machine *M* pour l'y exécuter. Un tel programme existe de par l'équivalence des machines de calcul universel. On voit donc qu'engendrer la suite *s* sur *M* ou *M'* demande des programmes du même ordre de grandeur. Plus précisément, la taille de ce programme ne dépend de cette machine que par une constante additive, la longueur de l'émulateur *P* qui simule *M'* sur *M*. C'est une constante que l'on peut négliger en première approximation si on traite des suites suffisamment longues. C'est la découverte de ce résultat d'indépendance, ou théorème d'invariance, qui fonde la théorie algorithmique de l'information.

Interprétation du contenu en information Cette mesure est liée à la notion d'aléa, au sens de « non prédictible ». Nous savons qu'il est facile de générer des

nombres entiers qui ont l'air aléatoires. Par exemple, la suite de nombres $a_0 = 0$, $a_n = (a_{n-1} + K) \% 2^n$ où l'on ajoute un grand nombre premier K au nombre précédent avant de ne garder que les n premiers bits – c'est ce que fait l'opération $p \% 2^n$, qui est le reste de la division euclidienne de p par 2^n – aura une distribution pseudo-aléatoire bien uniforme. Mais cette suite $s = \{a_0, a_1, \ldots, a_N\}$ de nombres a une bien faible complexité algorithmique, puisque la voilà générée par un tout petit programme. Une suite vraiment aléatoire est au contraire une suite s qui ne peut être prédite par aucun programme sauf « `écrire "s"` ». Plus formellement : c'est une suite binaire qui n'est produite que par des programmes plus longs qu'elle. L'étude de ces suites est un domaine de recherche ardu et très intéressant, et qui n'a pu émerger que grâce à l'informatique : c'est un des exemples de nouvelles mathématiques issues de l'informatique.

Un des premiers résultats de cette théorie est un résultat d'existence de suites aléatoires. En effet, il y a $N = 2^{1001} - 1$ suites binaires de longueur inférieure ou égale à 1000. Les programmes étant eux-mêmes des suites binaires, il y a au plus N programmes, de longueur inférieure ou égale à 1000, qui produisent une suite binaire. Or, parmi ces programmes, certains, comme le programme « `écrire 2000 fois "0"` », produisent une suite binaire de longueur strictement supérieure à 1000. Il y a donc strictement moins de N programmes, de longueur inférieure ou égale à 1000, qui produisent des suites binaires de longueur inférieure ou égale à 1000. Il existe donc une suite s, parmi les N suites binaires de longueur inférieure ou égale à 1000, qui n'est produite par aucun programme de longueur inférieure ou égale à 1000. Tous les programmes, par exemple le programme « `écrire "s"` », qui produisent la suite s sont donc strictement plus longs que s.

Voici la notion de complexité de Kolmogorov ou de contenu en information de Kolmogorov. Elle capture la complexité aléatoire de l'information et on peut la relier aux notions usuelles d'incompressibilité, d'*imprévisibilité*, donc d'absence de structure. Andreï Kolmogorov a introduit l'idée – Ray Solomonoff sans être aussi précis l'avait eue avant lui, et Leonid Levin, Per Martin-Löf et Gregory Chaitin ont fait le travail qui a permis de faire aboutir ce concept scientifique collectif.

Un cran plus loin, des mécanismes de calcul plus limités peuvent être considérés. Par exemple, des mécanismes ne servant pas à coder toutes les suites finies possibles mais seulement une certaine famille de suites, ou ne disposant que d'une quantité limitée de mémoire, ou devant travailler en temps proportionnel à la taille de s, etc. Nous parlons alors de complexité *descriptionnelle* ou de contenu descriptionnel en information d'une chaîne binaire s, lorsque nous ne supposerons pas que M est une machine universelle, mais un autre système

numérique. Un exemple très simple est la « table de référence » : considérons un nombre fini, disons L, de chaînes binaires $s_\ell, \ell \in \{1, 2, \ldots, L\}$ de longueur quelconque, par exemple les L films numériques d'une médiathèque. Dans ce cas fini, chaque film se décrit par son numéro $\ell \in \{1, \ldots, L\}$ donc un nombre de $\log_2(L)$ bits, dans ce cas très spécifique. Au contraire, la notion d'information de Kolmogorov est sans doute la moins arbitraire de toutes les définitions générales de contenu en information, du fait du résultat d'invariance évoqué ici.

Il y a un autre lien entre cette notion et l'informatique théorique. Cette notion de complexité est très bien définie mais… elle n'est en général pas calculable ! De même qu'il n'existe aucun procédé algorithmique pour garantir qu'un programme va ou non boucler à l'infini – on ne peut pas, sauf information additionnelle, exclure qu'il finisse par s'arrêter –, de même il est *prouvé* qu'aucun procédé de calcul, aucun programme, ne peut calculer la complexité en information de toutes les suites binaires finies. Pour calculer en pratique ce qui est donc incalculable – nous serions dans la même situation pratique si la quantité était calculable, mais de manière exponentiellement compliquée –, la démarche naturelle est d'en *approximer* la valeur. Les techniques consistent à compresser le message par les algorithmes que nous avons décrits précédemment et étudier dans quelle mesure la longueur du programme de décompression et de la chaîne compressée constitue une bonne approximation de la complexité de Kolmogorov.

Notion algorithmique d'information organisée

Complexité organisée et profondeur de l'information Avançons dans notre étude de l'information.

On dit souvent dans le langage courant qu'une pensée est profonde si elle est l'aboutissement d'un long temps de réflexion. La notion de profondeur logique proposée par Charles Bennett en 1988 capture de manière magistrale cette idée.

Ici, on ne cherche plus à définir une notion d'information aléatoire ou incompressible, mais une notion d'information liée à une complexité organisée. Regardons quelques exemples :

– simple et peu profond — un cristal
structure périodique d'organisation simple est sans aléa.

– simple et profond — π à 10^{-100} près
calculable par un petit programme, mais gros effort de calcul.

– aléatoire et peu profond — gaz parfait
pas prédictible donc pas reproductible, mais organisation triviale.

– aléatoire et profond — être vivant
pas prédictible et résultat de millions d'années d'évolution.

Nous voulons donc capturer le fait que cette information est « précieuse » par le fait qu'il a fallu un « effort de calcul » pour y arriver. Bennett propose à cette fin de prendre en compte *le temps de calcul du plus petit programme écrit pour une machine universelle qui génère le contenu binaire* en question. Le programme le plus court est celui qui a servi à calculer la complexité de Kolmogorov.

Notons tout de suite que ce n'est sûrement pas le temps de calcul du programme le plus rapide, qui est très probablement « `écrire "s"` », c'est-à-dire quelque chose de trivialement relié à la longueur du message et non à la complexité de son organisation.

Ce qui est remarquable, c'est que cette notion « tient bon », c'est-à-dire qu'elle commence à être reconnue comme une « bonne notion » de complexité organisée. Elle n'est pas additive si les données peuvent être reliées : deux moutons ne sont pas deux fois plus complexes que un, la croissance est lente : la profondeur augmente avec du calcul, mais lentement et il y a apparition spontanée de complexité organisée par le calcul.

Cette notion est robuste, c'est-à-dire qu'elle dépend relativement peu de la machine universelle que l'on se donne.

C'est bien sûr une quantité non calculable puisque basée sur la complexité de Kolmogorov.

Interprétation de la profondeur de l'information La proposition de Bennett n'est peut-être pas une proposition ultime, et il se peut que la notion de complexité organisée puisse encore être formalisée en se fondant sur d'autres bases ou en enrichissant l'idée de Bennett. Cette possibilité d'une évolution ne semble pas envisageable pour la complexité de Kolmogorov qui, elle, est une proposition définitive. Mais nous capturons là de manière formelle l'idée essentielle d'émergence, au cœur de l'idée évolutionniste darwinienne : un objet richement structuré et organisé ne peut pas sortir de rien, instantanément, mais demande un long processus d'interaction entre ses divers éléments, y compris son environnement, c'est-à-dire une sorte de calcul prolongé et cumulatif. La profondeur logique de Bennett mesure ainsi la quantité de calculs fixée dans un objet, c'est une mesure du contenu computationnel de l'objet, c'est une mesure de la longueur de la dynamique qui lui a donné naissance. L'émergence d'un objet profond ne peut être brusque, alors que l'apparition d'un objet complexe, au sens de la complexité de Kolmogorov, dans certaines situations se produit instantanément. C'est probablement ce qui explique de la manière la plus formelle pourquoi le leurre de « l'intelligence artificielle » – pouvoir programmer rapidement un être virtuellement intelligent – ne tient pas : ce qui fait l'intelligence a dû émerger de millions d'années de « calcul » biologique.

Lorsque le cerveau d'un bébé se construit en quelques mois, c'est en grande partie en « recopiant » ou « décompressant » le résultat de ces « calculs » biologiques millénaires.

Voici une des pointes de la recherche en informatique aujourd'hui. Que le lecteur ne s'y trompe pas : la présentation succincte de ces notions ne doit pas masquer (1) la très forte technicité de ces notions, car les « mots » cachent des formalismes difficiles, et (2) le chemin scientifique qui a pu les produire. Avant d'en « arriver » à la profondeur logique de Bennett, les cimetières académiques se sont remplis de « mauvaises » notions.

Notion probabiliste d'information

Quand la notion d'information est abordée avec une vision probabiliste, on considère que l'on accède aux événements observés par la mesure de leur probabilité d'occurrence. Et donc savoir quelque chose sur un événement, avoir de l'information, est lié à la probabilité que cet événement se produise. Un événement n'est pas « vrai » ou « faux », mais plus ou moins *probable*. Un rappel des bases des probabilités est proposé plus loin, au paragraphe « La vision probabiliste des choses : un rappel ».

Contenu probabiliste en information Comment définir une notion d'information dans ce contexte ? Assez naturellement, nous allons présupposer que le contenu probabiliste en information n'est fonction que de la probabilité des événements, puisque c'est ce qui donne l'information sur les événements.

Pour la conjonction d'événements, posons que les informations de deux événements indépendants s'ajoutent : si E_1 et E_2 sont indépendants, alors $h(E_1 \cap E_2) = h(E_1) + h(E_2)$, comme nous l'expliquions informellement en introduction.

Pour la disjonction d'événements incompatibles, posons que l'information se moyenne : si E_1 et E_2 sont incompatibles alors $h(E_1 \cup E_2) = p(E_1)\ h(E_1) + p(E_2)\ h(E_2)$. En effet, si l'événement $E_1 \cup E_2$ se produit, alors soit E_1 soit E_2 se produit, ce qui apporte une information soit $h(E_1)$ soit $h(E_2)$, et l'espérance de cette information combinée est donc $p(E_1)\ h(E_1) + p(E_2)\ h(E_2)$.

Finalement, il serait cohérent qu'un bit d'information corresponde à ce que nous gagnons en information de savoir qu'un élément binaire vaut 0 ou 1.

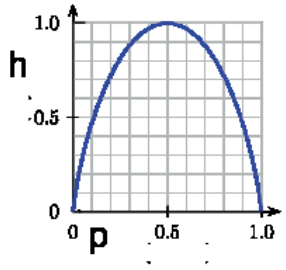
Avec ces trois leviers, il se trouve que la notion d'information est complètement et précisément définie. C'est l'*entropie* au sens de Shannon et sur un ensemble fini ou dénombrable. Elle vaut, en supposant que c'est une fonction continue :

$$h(E = .\{e_1, \quad e_2, \ldots, \}) = -\sum_{e_n \in E} p(e_n) \log_2(p(e_n))$$

Elle donne le *contenu moyen en information d'un événement d'une probabilité*. Il faut comprendre que c'est l'information que nous gagnons à connaître lors de la réalisation d'un événement aléatoire.

Regardons ce qui se passe si nous lançons une pièce de monnaie qui va tomber avec une probabilité p sur pile et $1 - p$ sur face. Son information vaut $h(pile, face) = -p\ log_2(p) - (1 - p)\ log_2(1 - p)$.
Si $p = 1/2$, c'est-à-dire qu'elle a autant de chance de tomber sur pile ou face, alors $h(pile, face) = 2(-1/2\ \log_2(1/2)) = 1$: nous gagnons exactement un bit d'information à savoir que la pièce tombe sur pile ou face.

Si, au contraire, $p = 0$, c'est-à-dire qu'elle va forcément tomber sur pile, alors puisque $0\log_2(0) = 0$ et que $\log_2(1) = 0$, $h(pile, face) = 0$: nous ne gagnons aucune information à savoir que la pièce tombe sur pile ou face, puisque ce sera pile. Si nous traçons l'entropie $h(pile, face)$ en fonction de la probabilité p



nous observons les éléments suivants :
– l'entropie est une quantité positive ou nulle ;
– l'entropie est minimale, égale à 0, si une des probabilités $p(e_n) = 1$, toutes les autres valant alors 0, donc si la situation est *sans aléa*, c'est-à-dire le résultat est certain ;
– l'entropie est maximale si l'aléa est maximal, c'est-à-dire que la distribution de probabilité est uniforme, sans privilégier aucune valeur ; ici, elle est égale à 1 pour un bit d'information. Elle est égale à $\log_2(N)$ si l'événement E prend N valeurs ;
– l'entropie est une fonction concave ; elle est symétrique : on peut inverser pile et face sans changer sa valeur, ou plus généralement inverser les valeurs de l'événement sans changer l'entropie.

Ce que nous voyons sur cet exemple est tout à fait général. La réalisation d'un événement apportera un maximum d'information si tous les événements sont équiprobables, donc complètement au hasard : s'il y a N réponses possibles et équiprobables, connaître cette information doit correspondre à N bits,

c'est-à-dire N « atomes » d'informations et vaudra $\log_2(N)$. C'est exactement le nombre de bits pour coder la suite binaire.

Cette notion d'information se relie à l'entropie dont il est question en thermodynamique, car elle propose une mesure du « désordre » au sein des données. Plus elle est importante, plus les données sont imprévisibles, donc susceptibles de contenir de l'information. Dans ce paradigme, on suppose que le récepteur est susceptible de faire certains calculs pour reconstituer *s* à partir de ce qu'on lui transmet. La machine *M* qui fait le décodage peut être prise comme référence, et on découvre alors que la théorie de Shannon doit être vue comme une version probabiliste de la théorie algorithmique de l'information de Kolmogorov et compatible avec elle dans le sens suivant : *le contenu algorithmique moyen d'information de Kolmogorov des chaînes binaires de longueur* n – *pondérées par les probabilités résultant des fréquences supposées – est du même ordre de grandeur que le contenu en information au sens de Shannon.*

De plus, cette notion d'information est une *véritable mesure physique, avec une unité, le bit.* C'est une quantité abstraite que l'on ne voit ni n'entend, comme l'énergie.

Cette notion a de nombreuses applications : plus des données sont prévisibles, plus il est facile de les compresser. De même, plus un message contient d'information au sens défini ici, plus le canal pour transmettre ce message devra avoir une grande capacité.

Si cette notion ne dit rien du sens qui peut être attaché à ces données, ni de la complexité nécessaire à leur création, elle est le pilier de plusieurs domaines de l'informatique et des sciences du numérique. Nous en verrons deux ici.

Transmission d'information L'entropie de Shannon est utilisée pour numériser une source avec le minimum possible de bits sans perte d'information. Elle permet aussi de quantifier le nombre minimum de bits sur lesquels on peut coder un fichier, mesurant ainsi les limites que peuvent espérer atteindre les algorithmes de compression sans perte, que nous discuterons dans le sixième chapitre. Plus précisément, cette formulation donne le contenu moyen d'information de l'ensemble des chaînes binaires *s* quand on tient compte des probabilités des blocs de bits utilisés. Le « théorème de la voie sans bruit » indique que l'on ne peut pas en moyenne compresser plus *s*.

Le point clé est donc :

Une variable d'entropie *h(x)* sera « presque sûrement » codée avec *h(x)* bits.

La théorie de l'information de Shannon est donc aussi une théorie de l'information par compression qui, au lieu de considérer des suites quelconques, suppose que les suites que l'on transmet vérifient certaines propriétés statistiques. Finalement, la théorie de Shannon est une théorie du contenu en infor-

mation relativement à un but de compression et à une certaine distribution statistique des suites.

La compression de données ou codage de source est l'opération informatique qui consiste à transformer une suite binaire en une suite plus courte, contenant les mêmes informations, en utilisant un algorithme particulier. Les « fichiers zip » sont des exemples de données compressées. Il n'existe pas de technique de compression de données sans perte universelle, qui pourrait compresser n'importe quel fichier : on démontre que si une technique sans perte compresse au moins un fichier, alors elle en grossit également au moins un.

Estimation d'information à partir de l'entropie L'entropie de Shannon est également utilisée dans d'autres domaines comme, par exemple, pour la sélection du meilleur point de vue d'un objet en trois dimensions, recaler deux images différentes l'une sur l'autre en minimisant l'entropie des deux images, etc.

Discutons maintenant cette facette de la théorie de l'information. Que veut dire estimer une valeur incertaine ? L'idée est que si je ne sais rien de rien, je ne peux qu'agir au hasard. Si je ne sais rien de la pièce de monnaie, il n'est sûrement pas raisonnable de penser qu'elle va tomber plus sur pile que sur face. L'hypothèse minimale est, compte tenu de mon manque d'information, de supposer qu'elle va tomber aléatoirement sur pile ou face, donc qu'il y a une probabilité $p = 1/2$ de tomber sur pile, ou face. En d'autres termes, faute d'information, choisissons la solution *d'entropie maximale*. Celle où l'on confie au hasard ce que nous ne pouvons pas expliquer.

Ce choix méthodologique est un principe de simplicité ou parcimonie que l'on nomme parfois « rasoir d'Occam ». C'est un principe qui refuse de présupposer une cause « par-delà ce qui est nécessaire ». À l'instar de ce dialogue entre Napoléon « ne trouvant pas mention de Dieu dans le système » de Monsieur de Laplace qui se doit alors de préciser : « Je n'ai pas eu besoin de cette hypothèse [qui] explique en effet tout, mais ne permet de prédire rien. » Bref, ce que nous n'expliquons pas est de l'*information non encore obtenue*, de l'entropie. Et son modèle minimal est le hasard.

Ce qui est remarquable ici est que ce principe général s'incarne alors dans un formalisme précis et nous offre un moyen de *calculer*, et même d'estimer au mieux en fonction des mesures que nous avons.

Les grandes distributions de probabilité que nous manipulons usuellement ne sont rien d'autre que des lois d'entropie maximale.

– Supposons que nous connaissions la moyenne et la variance d'une variable aléatoire continue, mais rien d'autre. Quelle est la « bonne » distribution ? C'est-à-dire celle qui va rendre compte de la moyenne et variance, mais sans

rien ajouter de fallacieux ? C'est la loi normale, dite gaussienne, avec sa courbe de Gauss en forme de cloche.
– Supposons que des événements se produisent aléatoirement et que nous connaissions le rythme moyen de ces événements, mais rien d'autre. C'est la loi de Poisson, qui aura alors les vertus demandées. Et ainsi de suite. En fait, nous pouvons suivre la même démarche *dans tous les cas*.

Cherchons la « vérité » sur un événement E, c'est-à-dire les valeurs $p(e_n)$ de sa distribution de probabilité. Pour cela, nous disposons de quelques mesures, par exemple une mesure de moyenne ou de dispersion. Nous savons donc que la distribution de probabilité n'est pas au hasard, puisqu'elle vérifie ces équations de mesure. Pour le reste, le hasard demeure. Alors, parmi toutes les distributions de probabilité, nous allons prendre celle d'entropie maximale. Nous passons d'une situation où il y a des équations de mesure que nous ne savons pas très bien résoudre, à une situation où nous définissons parfaitement la solution. Nous en détaillons les équations en complément, plus loin, au paragraphe « La vision probabiliste des choses : un rappel ».

Il s'est passé quelque chose d'essentiel dans le calcul ébauché ici : nous sommes passés d'un souhait louable – « n'expliquer que l'information obtenue par nos mesures, le reste étant laissé au hasard » – à un *calcul effectif*, qui (1) nous donne la forme de la distribution de probabilité, (2) permet de poser les équations du calcul numérique pour déterminer les paramètres de cette probabilité. Nous sommes passés d'une spécification, le « quoi » qui formalise ce que nous voulons obtenir, à une implémentation, le « comment » qui fournit un algorithme pour le calculer.

La classe d'algorithmes considérée ici est celle d'estimation de valeurs numériques par minimisation. Nous donnons plus de détails au paragraphe « Estimation d'information par optimisation ». La notion est scientifiquement belle, car elle rend simple quelque chose de compliqué. Dont acte. Seuls les Shadoks, double privilège de l'artiste et de l'humoriste, gardent alors encore le droit de rétorquer à cela : « Pourquoi faire simple quand on peut faire compliqué ? ».

Que faire après avoir estimé au mieux des valeurs numériques ? Agir, c'est-à-dire prendre des décisions, le plus souvent. Donc, comparer deux solutions pour mesurer s'il y a une différence significative – par exemple, comparer les scanners d'un patient quelconque avec un patient de référence. La théorie de l'information nous donne une nouvelle clé pour cela : nous pouvons mesurer quelle part d'aléatoire il reste dans un événement aléatoire en question après avoir été expliqué ou prédit par un événement aléatoire de référence : c'est l'*entropie relative*. On parle de *divergence*, qui est une mesure de dissimilarité

entre deux distributions de probabilité. Dotés de cet outil, il est alors possible de comparer des modèles, de statuer sur le fait que ces informations sont indépendantes ou non, de regarder le poids d'une conséquence par rapport à une cause, etc. D'autres outils probabilistes comme l'inférence baysienne viennent compléter ces méthodes qui s'adossent à ces concepts centraux.

Voici notre arsenal moderne de base pour traiter et manipuler l'information.

Conclusion : manipulation de l'information

La première idée est celle de l'utilisation d'algorithmes génériques, c'est-à-dire de méthodes de calcul et de transport insensibles à la nature et au contenu de l'information, car ne voyant que les bits 0 ou 1 qui la composent. On trouve ces algorithmes dans toutes les opérations de stockage, de cryptage et de transmission de données. C'est grâce à eux que l'on peut rendre l'information pérenne et indépendante des supports. En numérique, on peut effectivement transporter, répliquer, sauvegarder et diffuser arbitrairement l'information sans aucune perte de qualité, ce qui est fondamentalement impossible avec les supports analogiques traditionnels. C'est évidemment essentiel pour les télécommunications, qui ont pour charge principale d'assurer le déplacement de l'information dans le temps et dans l'espace sans la modifier. De plus, transport et sauvegarde se font avec un gain de place et une sécurité majeure par rapport aux méthodes analogiques : une bibliothèque numérique tient sur quelques grammes et, si on en garde des copies multiples, il n'est pas grave que l'une d'entre elles brûle.

La deuxième idée est l'utilisation d'algorithmes spécifiques liés à un type et à un volume d'informations donnés. On cherche les fautes d'orthographe dans les textes, pas dans les images. On ne compresse pas de la même façon les textes, les photos, la vidéo et les sons ; on zoome sur une image, pas sur un texte ; il est bien plus simple de chercher de l'information dans un ensemble de textes que dans un ensemble d'images ; les informations liées au pilotage d'un avion sont fondamentalement volatiles, alors que celles liées à sa maintenance doivent être gardées toute sa vie, etc. On assiste à l'heure actuelle à un développement considérable d'algorithmes spécifiques, très divers mais tous fondés sur un petit nombre de principes scientifiques que nous décrirons plus tard.

La troisième idée est celle de la multiplicité des représentations. Une même chose peut être représentée de façon différente selon les cas et les utilisations. Par exemple, considérons l'exemple de la représentation d'une matrice. La façon la plus simple et la plus évidente est la représentation matricielle, comme tableau bidimensionnel. Cela permet que certains algorithmes, comme l'addi-

tion ou la multiplication de matrices, soient simples. Imaginons maintenant que, pour des raisons physiques, les matrices que l'on considère soient de taille importante mais creuses – avec beaucoup de valeurs nulles. Dans ce cas, il peut être rentable en termes d'occupation mémoire d'utiliser une autre représentation, par exemple une liste de tous les endroits où la matrice est non nulle avec sa valeur en ce point. Il faut alors utiliser un algorithme d'addition de matrices adapté à cette représentation. Il sera un peu plus complexe, mais beaucoup plus rapide si les matrices sont très creuses. La représentation de l'information et l'algorithme la manipulant sont donc intimement liés.

Exercices corrigés et commentés

Exercice 1. Jouons au jeu du portrait.

Devinez le nom d'une personne en ne posant que des questions auxquelles on répond par oui ou par non dans la liste suivante

Qui ?	Genre ?	Looké Emo ?	Moins de 15 ans ?	
Pierre	garçon	oh non ! 0	oui	'001'
Nadia	fille	good-good 1	oui	'111'
David	garçon	énormément 1	non	'010'
Hamdi	garçon	certes 1	oui	'011'
Marie	fille	surtout pas 0	non	'100'
Adèle	fille	évidemment 1	non	'110'
MingYu	fille	beurk 0	oui	'101'
Igor	garçon	horreur! 0	non	'000'

Pour que les choix restent binaires, la réponse à propos du « looké Emo » ne peut être que oui ou non, sans introduire les nuances explicitées ici.

Y a-t-il deux personnes qui ont les mêmes attributs dans la liste proposée ?

Combien faut-il au minimum de questions oui/non pour deviner une personne de ce tableau ? Dans ce cas, est-ce suffisant ?

Que se serait-il passé si la proportion entre garçons et filles n'était pas égale ? Ou si la répartition des « looké Emo » avait été différente ?

Si la réponse à propos du « looké Emo » avait été explicitée comme dans le tableau et non binaire, que se serait-il passé ?

Peut-on interpréter chacun des bits de la colonne de droite ?

Pour deviner une personne parmi seize, combien faudrait-il de questions binaires au minimum ? Quelles propriétés devraient avoir ces questions pour être en nombre minimum ?

Refaire le jeu avec quatre personnes de la classe en leur trouvant des bits d'information pertinents.

Correction. En trois questions, « Est-ce une fille ? » « Aime-t-elle le look Emo ? » « A-t-elle moins de 15 ans ? », nous allons forcément deviner parmi les huit personnes celle correspondant au portrait, car toute l'information est utile ici. D'ailleurs, nous avons symbolisé ce fait en utilisant un bit pour chacune des questions binaires, reporté dans la colonne de droite.

Deux questions binaires, c'est-à-dire deux bits d'information, ne suffisent pas ici. Par exemple, si on ne sait pas si la personne recherchée a moins de 15 ans, alors Adèle et Nadia ou Igor et Pierre, par exemple, ne peuvent pas être distingués, puisqu'ils pensent la même chose du look Emo. On constate bien ici que le nombre de bits correspond au nombre de questions binaires à poser pour deviner toute l'information. Cela correspond donc à la taille en information.

On voit ici que la réciproque est fausse : ce n'est pas parce qu'il y a trois bits d'information que nous pouvons distinguer huit éléments : si tous ou simplement plus ou moins de la moitié des garçons avaient eu moins de quinze ans, par exemple, les deux informations auraient été redondantes et le codage de l'information n'aurait pas été suffisant.

On mesure aussi qu'il faut bien s'entendre sur le codage choisi : dans le cas du « looké Emo » les réponses contiennent bien plus d'information que oui ou non, mais des appréciations de valeur ou des nuances de ton qui peuvent, ou non, être elles-mêmes codées lors de la numérisation de l'information.

Exercice 2. Combien de temps peut durer ce jeu ?

Deux personnes A et B jouent à un jeu. A choisit un nombre entre 1 et 1000 que B doit deviner.

Avec la première règle, A ne répond que « oui » ou « non ». Si B est très intelligent mais très malchanceux, de combien d'essais au maximum a-t-il besoin ?

Avec la seconde règle, A répond « oui », « plus petit », ou « plus grand ». De combien d'essais au maximum B a-t-il besoin ?

Correction. Avec la première règle, il faut évidemment 1000 essais au maximum. B n'a aucune stratégie possible et se contente d'énumérer les nombres possibles, de 1 à 1000 par exemple. S'il est très malchanceux, le nombre choisi par A est 1000 et n'est trouvé qu'au millième essai.

Avec la seconde règle, B peut utiliser l'information supplémentaire et se permettre d'être plus intelligent. L'idée est de couper l'ensemble des nombres possibles en 2 à chaque question. On ne commencera donc pas par 1, mais par 500. L'ensemble des possibilités {1, …, 1000} sera alors réduit, si 500 n'est pas le nombre recherché, soit en {1, …, 499 }, soit en {501, …, 1000 }. On appelle cela une dichotomie. Un des pires cas correspond alors à
$\{1, \ldots, 1000\} \to^{500} \{1, \ldots, 499\} \to^{250} \{1, \ldots, 249\} \to^{125} \{1, \ldots, 124\} \to^{112} \{1, \ldots, 111\} \to^{56} \{1, \ldots, 55\} \to^{28} \{1, \ldots, 27\} \to^{14} \{1, \ldots, 13\} \to^{7} \{1, \ldots, 6\} \to^{3} \{1, \ldots, 2\} \to^{2} 1$.

Il faut poser au pire 10 questions pour obtenir avec certitude le résultat. Ce 10 est en fait la partie entière supérieure du logarithme en base 2 de 1000. En effet, on scinde l'intervalle en deux à chaque étape. Pour obtenir le nombre d'étapes, il suffit donc de calculer $\lceil \frac{\ln(1000)}{\ln(2)} \rceil = 10$.

Exercice 3. Binaire/décimal.

Écrire en décimal les nombres binaires 10010, 111010, 1111111.

Écrire en binaire les nombres décimaux 9, 42, 2049 et 10010.

Correction. En appliquant l'algorithme, on obtient $10010_2 = 18$, $111010_2 = 58$ et $1111111_2 = 127$.

De même, on obtient : $9 = 1001_2$, $42 = 101010_2$, $2049 = 100000000001_2$ et $10010 = 10011100011010_2$.

Exercice 4. Binaire/décimal signés.

Écrire en décimal les nombres binaires sur 8 bits : 10010, 10111010, 11111111.

Écrire en binaire sur 8 bits les nombres décimaux -9, -42, -2049 et -10010.

Correction. On a $10010_2 = 18$, sur 8 bits, son premier bit est 0. Puis, on a $10111010_2 = -27 + 58 = -70$ et $11111111_2 = -2^7 + 127 = -1$.

Ensuite, sur 8 bits, on a : $-9 = -2^7 + 119 = -2^7 + 1110111_2 = 11110111$ et $-42 = -2^7 + 86 = -2^7 + 1010110_2 = 11010110$. Enfin, 10010 nécessite 14 bits. Il en faut au moins 15 pour représenter −10010 qui ne peut donc pas être représenté sur 8 bits.

Exercices non corrigés

Exercice 1

Définir un codage binaire pour coder une information courante, par exemple les dates de naissance des personnes de la classe. Rechercher un code qui utilise un minimum de bits, *compte tenu* des données de la classe. Puis rechercher comment coder la date de naissance de n'importe quel élève du lycée.

Exercice 2

Chercher dans Wikipédia combien il y a de caractères codés en UTF-8 pour toutes les langues du monde. Expliquer en 2 lignes le lien entre le codage ASCII et UTF-8.

Exercice 3

Expérimenter les calculs en binaire : ajouter, soustraire et multiplier des nombres en binaire.

Exercice 4

Pour incrémenter un nombre binaire, il faut : 1. commencer par le bit de poids faible ; 2. inverser le bit ; 3. tant que ce bit est à zéro, recommencer 2. avec le bit situé à gauche.

Vérifier que cet algorithme est bien correct et décrire celui qui décrémente un nombre binaire. Et celui qui le multiplie par deux.

Exercice 5

Compter le nombre de bits pour coder une image, par exemple 1024 × 1024, en noir ou blanc, puis avec des niveaux de gris de 0 à 255.

Exercice 6

Ouvrir un fichier audio quelconque avec un logiciel comme `audacity` ou tout autre logiciel qui puisse lire les paramètres d'un fichier audio. Regarder la durée D du son en seconde, la fréquence F d'échantillonnage – 44 100 Hz ? Davantage ? –, le nombre C de canaux – mono ? stéréo ? –, et le nombre O d'octets pour coder un échantillon – 16 bits donc 2 octets ? Davantage ? Comparer la taille du fichier en octet au nombre $D \times F \times C \times O$ et en déduire le taux de compression.

Exercice 7

Concevoir le même exercice que le précédent mais pour une image.

Exercice 8

Sachant que le cerveau humain a environ 100 milliards de neurones qui ont chacun en moyenne 10 000 synapses qui transmettent environ 100 impulsions binaires, des potentiels d'action, par seconde, calculer la taille en information maximale de l'activité d'un cerveau.

Exercice 9

Ouvrir une image au choix avec un logiciel d'édition élémentaire d'images tel que `gimp` ou tout autre logiciel similaire et essayer de modifier la lumière et le contraste : laquelle de ces opérations correspond à ajouter/retirer une valeur constante à la valeur d'un pixel et laquelle correspond à multiplier le pixel par un nombre plus grand/petit que un ?

Exercice 10

Deviner et décrire avec des mots la transformation d'image qui correspond pour chaque pixel à faire la moyenne avec les pixels voisins. Indice : penser soit au zoom, soit à la mise au point d'une lentille optique.

Exercice 11

Prenons deux photos de suite d'une scène avec un objet fixe et un objet mobile et soustrayons les deux images, c'est-à-dire faisons la soustraction de chaque valeur du pixel de la deuxième image avec le pixel correspondant dans la première. Que s'est-il passé pour l'objet fixe ou mobile ? À quoi peut servir la soustraction d'images alors ?

Exercice 12

Exercice rédactionnel : expliquer en moins de 5 lignes le lien entre la complexité de Kolmogorov et la profondeur logique de Bennett.

Exercice 13

Que dire d'un poème qui aurait une très grande complexité de Kolmogorov ? et une profondeur logique de Bennett ?

Exercice 14

Combien a-t-on de chance de pirater mon code de carte de paiement – avec un code à 4 chiffres et le droit de faire 3 essais ?

Exercice 15

En combien de temps un moteur Internet qui peut essayer 1 000 mots de passe par seconde pourrait craquer un mot de passe de 6 lettres ? et de 6 caractères ASCII de 7 bits ? Est-ce bien utile de ne pas utiliser que des lettres pour son mot de passe ? Et si, pour se protéger, le site web multipliait par deux le délai à chaque essai – 1/1000 de seconde pour le 1er essai, 2/1000 de seconde pour le 2e essai… – que se passerait-il ?

Exercice 16

Calculer l'entropie d'un jet de dé à six faces non pipé. Et d'un dé avec « 1, 1, 2, 2, 2, 3 » sur les faces.

Exercice 17

Shannon propose une façon simple de déterminer l'entropie d'un texte donné pour un récepteur donné : A dispose du texte et demande à B de le deviner lettre par lettre, espaces compris. Si B devine correctement la lettre, on compte 1, car A, en lui répondant « oui », lui a transmis 1 bit d'information. Si B se trompe, on lui indique la bonne lettre et on compte 4.75, car un caractère parmi 26 lettres plus 1 caractère d'espace représente environ $ln_2(27) = 4.75$ bits d'information. Appliquer cette méthode à un texte courant. Constater qu'environ une lettre sur deux peut être ainsi devinée, la redondance du langage

courant était en conséquence d'un facteur 2, et le contenu informatif d'une lettre dans ce contexte de 2,35 bits seulement.

Exercice 18

En utilisant une calculette programmable ou tout autre outil de calcul, programmer la suite $a_0 = 0$, $a_n = (a_{n-1} + 7919)\%16$ et tracer ou faire compter les occurrences des valeurs : qu'en conclure ?

Exercice 19

Mon banquier m'a proposé cet investissement :

– on me donne $e = \exp(1)$ €,
– l'année suivante, je prends 1 € de frais et je multiplie par 1,
– l'année suivante, je prends 1 € de frais et je multiplie par 2,
– l'année suivante, je prends 1 € de frais et je multiplie par 3,
– …
– après n ans, je prends 1 € de frais et je multiplie par n,
– Pour récupérer mon argent, il y a 1 € de frais.

Dans 50 ans, pour ma retraite, combien d'argent aurai-je ?

L'intérêt de cet exercice repose dans l'utilisation de plusieurs calculatrices, logiciels de calculs… de façon à faire varier *grandement* la réponse obtenue.

Exercice 20

Codage décodage d'une phrase en arabe. Parmi les 28 symboles de l'alphabet arabe, avec ses symboles complémentaires, que l'on devine sur cette ancienne machine à écrire :

nous allons en considérer 16 :

ء	ا	إ	ؤ	أ	ئ	ا	ت	د	ش	ف	ك	ل	ن	و	ي
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Donner alors le codage binaire de la phrase, qui se lit de droite à gauche :

« كل شيء في ثنائي الكود »

Pour qui ne sait pas lire l'arabe : aller sur Google et essayer de traduire en arabe « tout se code en binaire ». Que remarquons-nous ? Aurait-il été facile de faire l'inverse ?

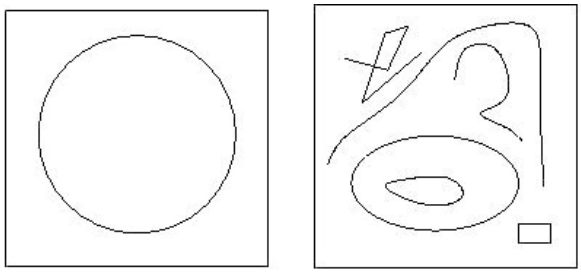
Questions d'enseignement

Découvrir le codage d'un dessin avec les élèves

Préambule : Plutôt que de se lancer dans des considérations générales de didactique, commençons par un exemple concret qui « fonctionne » bien avec les élèves. Il s'avère, en pratique, que la notion de tableau de pixels paraît naturelle aux jeunes élèves, qui sont entourés d'images au quotidien. En revanche, le « codage de chaque pixel » est en général un peu plus mystérieux. Dévoilons-le ici. Ce sera aussi l'occasion de montrer sur cet exemple concret la différence entre description numérique et symbolique d'un objet et l'approximation inhérente à la numérisation.

Ce paragraphe, pour tenter d'avoir valeur d'exemple, cible des élèves et non des enseignants.

Présentation de l'exemple : Regardons en détail comment coder un dessin au tracé simple. Comment faire pour décrire un dessin ? Par exemple, comment décrire le dessin de gauche ?



Une solution est de dire : « Cette image est formée d'un cercle. » Nous pouvons même être plus précis et indiquer les coordonnées du centre du cercle, son rayon, sa couleur, l'épaisseur du trait… À partir de cette description, n'importe qui pourrait reconstituer le dessin. On donne alors une description *symbolique* du dessin : « un cercle, dont le centre est le milieu de la figure, le diamètre les trois quarts de la figure, la couleur du trait est noire, son épaisseur est de 1 mm, tandis que le fond de la figure est blanc ». Il semble que nous n'ayons rien oublié : c'est-à-dire que *si nous donnons tous ces éléments à quelqu'un, il va pouvoir*

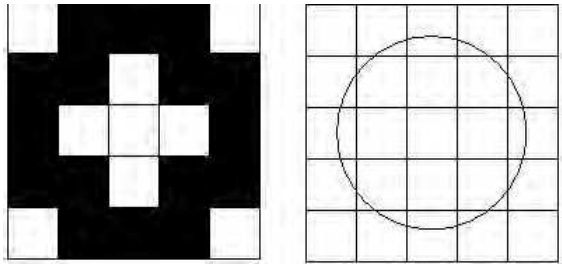
reproduire le même dessin. Nous pourrions, du reste, exprimer ces données de manière structurée :

```
{
 forme = {
   type = "cercle"
   centre = (size/2, size/2)
   rayon = 3/4 * size/2
  }
  trait = {
   couleur = "noir"
   largeur = "1 mm"
 }
 fond = {
   couleur = "blanc"
  }
 }
```

en supposant ici que `size` est la taille de la figure carrée. Le dessin est donc défini par sa forme, son trait et le fond. La forme a un type, ici un cercle, qui est entièrement défini par son centre et son rayon. Il faut, bien entendu, que la sémantique de chaque variable, `forme, type, trait, ...`, soit convenue et que les valeurs de ces variables soient comprises aussi.

Cette méthode marche bien pour le dessin de gauche. Mais tout décrire avec des mots serait bien moins pratique pour le dessin de droite ! En effet, comment facilement décrire en détail ces traits qui semblent tracés au hasard ? Qu'en serait-il avec un dessin qui représente un objet réel, ou un visage ?

Une autre méthode, qui a l'avantage de pouvoir être utilisée pour n'importe quel dessin, consiste à superposer un quadrillage au tracé, comme proposé à gauche :



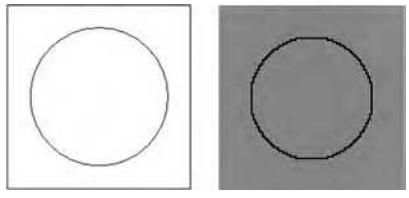
Chacune des cases de ce quadrillage s'appelle un pixel. On noircit ensuite les pixels qui contiennent une portion de trait, comme ci-dessus à droite. Puis, il nous suffit d'indiquer la couleur de chacun des pixels, en les lisant de gauche à droite et de haut en bas, comme un texte – dans notre culture occidentale. Ce dessin se

décrit donc par une suite de mots « blanc » ou « noir ». Comme seuls les mots « noir » ou « blanc » sont utilisés, nous pouvons être plus économes et remplacer chacun de ces mots par un seul symbole, par exemple le mot « noir » par la lettre « n » ou le chiffre « 0 » et le mot « blanc » par la lettre « b » ou le chiffre « 1 ».

Le dessin ci-avant, avec une grille de 5 × 5, se décrit alors par la suite de 25 chiffres : 1000100100011100010010001

Ces différentes descriptions sont équivalentes : le dessin se décrit toujours par une suite de symboles choisis parmi deux. Le nom de ces deux symboles importe peu, et nous pourrions les appeler noir et blanc, a et b, 0 et 1, pile et face, - et +, Dupond et Dupont… que cela ne changerait pas grand-chose. En général, on choisit 0 et 1.

De fait, cette description est assez approximative, et on constate une grande différence entre ces deux images, mais nous pouvons rendre la description plus précise en utilisant un quadrillage, non plus de 5 × 5, mais de 100 × 100 pixels.



À partir de quelques millions de pixels, nous ne serions plus capables de faire la différence entre les deux images. Cette méthode est donc approximative, mais universelle : n'importe quel dessin, même très compliqué, se décrit exactement comme un dessin simple.

Cette suite de 0 et de 1 s'appelle une suite binaire, ou parfois un mot binaire. Notre dessin se décrit ainsi avec 25 chiffres binaires, ou encore 25 bits ; sa longueur est 25.

Quel est le premier point clé ? *Convenir d'un standard pour toutes et tous.* Bien entendu, il faut que nous soyons tous d'accord pour décrire tous les dessins de la même façon ! Décider, comme nous l'avons fait dans l'exemple précédent, que le noir est représenté par 0 et le blanc par 1 et que les pixels se lisent de gauche à droite et de haut en bas. Le codage de l'information est avant tout une affaire de… convention.

Quel est le second point clé ? *Selon le codage choisi, l'information se manipule, selon les cas, plus ou moins facilement.* Bien entendu, nous aurions pu coder le dessin, par exemple, en commençant par le pixel du milieu, puis en énumérant les pixels en spirale. C'est une très bonne idée pour une machine mécanographique qui doit tracer ce dessin : au fil des bits, le stylo a toujours à tracer des

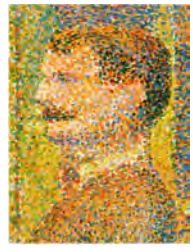
pixels consécutifs, donc évite de grands déplacements, d'une ligne à l'autre comme le codage proposé ci-avant l'implique. Mais c'est une idée un peu tordue, et si une personne doit tracer le dessin pixel à pixel à la main, elle sera plus embarrassée avec un codage en spirale. De même, si nous devons analyser ce dessin avec un logiciel, le codage facilitera ou non certains traitements, comme la détection de traits horizontaux.

Le codage de l'information est donc en lien profond et étroit avec la manipulation de cette information.

Décryptage de l'exemple. Que ce soit sous forme magistrale ou sous forme d'activité, il semble que la description d'une image soit le bon levier pour faire découvrir la notion de codage numérique. Le codage des nombres réclame plus de compétences calculatoires, et celui des sons et de la musique, bien qu'au cœur du quotidien des jeunes, se révèle finalement moins intuitif.

On peut facilement imaginer, pour le codage symbolique, un jeu à deux joueurs, où l'un doit donner, par écrit, des consignes précises sous forme rédigée puis sous forme symbolique, afin que l'autre reproduise tel carré, losange, rectangle avec un texte dedans, petit personnage stylisé, etc. L'intérêt est de montrer la nécessité de la rigueur et, surtout, de la standardisation : il faut se mettre d'accord sur le sens des mots. C'est à la fois un bel exercice de français et de structuration de la pensée.

Il est assez intéressant de relier le travail de pixélisation d'une image avec la démarche pointilliste du mouvement impressionniste, qui porte cette fulgurante intuition de la relation entre la couleur et la forme comme l'illustre ce dessin de Georges Seurat :



car, au-delà de l'aspect interdisciplinaire, cela montre que la démarche n'est pas exclusivement technologique, et qu'elle implique une certaine représentation du monde. Le lien entre codage numérique et représentation artistique est loin d'être anecdotique, et constitue un levier pour permettre d'accéder aux deux champs culturels.

Pour en savoir plus sur ce point précis : *L'art assisté par ordinateur (2011), Anne-Charlotte Philippe, http://interstices.info*

Manipuler une image

Une image est un tableau à deux dimensions de points lumineux, c'est-à-dire de pixels. Ici, chaque pixel a une valeur d'intensité entre 0, pour le « noir », et 255, pour le « blanc ».

Regardons, sous forme de trame d'exercice, quelles opérations nous aident à mieux comprendre les manipulations que nous pouvons faire sur ces images.

Changer chaque pixel Par exemple, pour chaque valeur d'intensité $I_{i,j}$ d'un pixel de coordonnée (i, j), où i est l'abscisse et j l'ordonnée, calculons :

$$I_{i,j} \leftarrow 255 - I_{i,j}$$

donc remplaçons les valeurs claires par des valeurs foncées et inversement. Nous venons tout simplement de faire une « image négative », les valeurs de luminance sont inversées.

De même :

– Augmenter ou baisser la *luminosité* d'une image revient à ajouter ou soustraire une valeur constante à la valeur de chaque pixel.

– Augmenter ou baisser le *contraste* d'une image revient à multiplier ou diviser par un gain constant la valeur de chaque pixel.

Ce sont des opérations très faciles à expérimenter lors d'un petit exercice de programmation.

Combiner deux images pixel à pixel Que se passe-t-il si on « additionne » deux images ? C'est-à-dire si l'on construit une nouvelle image dont la valeur de chaque pixel est la somme des valeurs des pixels de chaque image ? Si chaque image est dessinée sur une feuille de papier-calque, et si les couleurs sombres s'interprètent comme de la transparence, cela revient en quelque sorte à superposer les deux calques. C'est assez amusant à expérimenter numériquement.

Si l'on « soustrait » les deux images de manière similaire à ce qui a été fait pour les ajouter, une autre fonctionnalité émerge. Prenons deux images d'une scène où un objet bouge sur un fond fixe et soustrayons pixel à pixel. Pour les parties immobiles de l'image, les valeurs sont similaires. Pour les parties de l'image correspondant à l'objet mobile, les intensités des pixels vont varier. On a donc commencé à détecter ce qui bouge.

Ici les zones jaunes et bleutées correspondent aux variations d'intensité liées aux mouvements. Cet opérateur est à la base de tous les détecteurs visuels de mouvement, biologiques ou artificiels.

Modifier les pixels en fonction des voisins Considérons une image et modifions un pixel en recalculant sa valeur avec celles des pixels voisins. Par exemple, faisons la moyenne avec les quatre voisins de gauche, de droite, de dessus et de dessous :

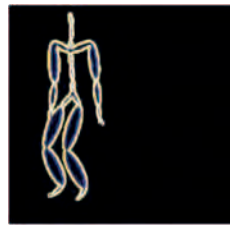
$$I_{i,j} \leftarrow \frac{1}{4}(I_{i+1,j} + I_{i-1,j} + I_{i,j+1} + I_{i,j-1})$$

comme illustré ici. Nous sommes en train de faire un *lissage* de cette image en faisant cette moyenne, c'est-à-dire de gommer les petites variations parasites de luminance d'un pixel à l'autre, donc de rendre l'image un peu floue en ne gardant que les variations de lumière à plus grande échelle.

Si nous répétons cette opération sur toute l'image plusieurs fois de suite, l'image deviendra de plus en plus floue, les détails étant gommés progressivement du plus petit au plus grand. Il y a des mécanismes non linéaires plus sophistiqués mais basés sur le même principe de filtrage qui permettent de réduire le bruit de l'image, associé aux petites variations de l'intensité, tout en préservant les formes de cette image, associées aux contours, c'est-à-dire aux grandes variations de l'intensité.

Si, au lieu de moyenner l'information de luminance, on calcule plutôt ses variations d'un pixel à l'autre, ce sont les éléments de contraste, donc les contours de l'image, qui vont émerger :

$$I_{i,j} \leftarrow |I_{i+1,j} - I_{i-1,j}| + |I_{i,j+1} - I_{i,j-1}|$$



comme illustré ici, où nous donnons un exemple simple de tel opérateur.

D'autres opérateurs, dits de « morphologie mathématique », sont des calculs locaux non linéaires particulièrement utiles pour filtrer, segmenter et quantifier des images.

Ces opérateurs, à la manière de briques logicielles, se combinent et s'enchaînent. Ils sont la base des systèmes de vision industrielle, qui permettent de concevoir des applications qui détectent la présence d'un objet, calculent les caractéristiques d'un ou de plusieurs éléments de l'image, etc.

Le principe de la dichotomie

S'il est un mécanisme algorithmique général accessible aux lycéens qui illustre ce que peut être une notion abstraite d'informatique, c'est bien le principe de recherche dichotomique. Revoyons-le ici, dans le cadre d'une activité scolaire.

Exemple : devinons l'âge d'un élève

Devinons l'âge d'un élève du secondaire, entre 11 et 18 ans inclus.

Avec une première règle, nous ne répondons que « oui » ou « non ». De combien d'essais au maximum avons-nous besoin ?

Avec une seconde règle, nous répondons « oui », « plus jeune », ou « moins jeune ». De combien d'essais au maximum avons-nous besoin ?

Nous pouvons alors proposer un codage binaire des âges entre 11 et 18 ans, avec un nombre minimal de bits.

Décryptage de l'exemple Nous constatons que, dans ce cas facile à mentaliser, il est très simple de réaliser que, dans le cas oui/non, nous avons à poser 8 questions : « A-t-il 11 ans ? A-t-il 12 ans ? A-t-il 13 ans ? A-t-il 14 ans ? A-t-il 15 ans ? A-t-il 16 ans ? A-t-il 17 ans ? A-t-il 18 ans ? » avec le risque, si nous manquons de chance, de devoir poser les huit questions avant de connaître la solution, tandis que, dans le cas oui/plus/moins, une meilleure méthode peut être intuitée en demandant d'abord si l'élève a moins de 15 ans. Nous pourrons expliquer en détail que si la réponse est oui, alors nous savons que l'élève a entre 11 et 14 ans ; si la réponse est non, alors nous savons que l'élève a entre 15 et 18 ans. Dans les deux cas, nous sommes passés d'une fourchette de 8 ans à une fourchette de 4 ans.

On parle de dichotomie (couper en deux) pour ce processus de recherche où, à chaque étape, on coupe en deux parties l'espace de recherche.

Il est assez facile de montrer par un petit calcul de probabilité, ou de faire intuiter par le langage que, dans le cas oui/non, si l'âge peut être quelconque entre 11 et 18 – que la probabilité est uniformément répartie, donc –, peu importe l'ordre dans lequel nous posons les questions. Nous pouvons le faire par

exemple dans l'ordre croissant de 11 à 18. Avec de la chance, nous poserons une seule question, au pire sept questions, puisqu'au septième non… la huitième réponse sera forcément oui. Nous en déduirons ou intuiterons qu'en moyenne nous poserons quatre questions, nous en poserons : 1, 2, 3, 4, 5, 6 ou 7 avec une probabilité de 1/7, soit la moitié. Ce résultat étant évidemment général.

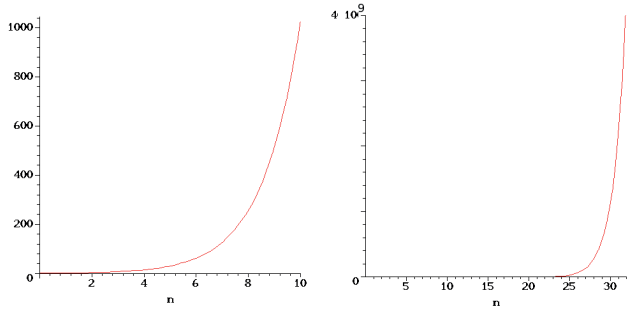
Si nous continuons à poser une question qui divise l'intervalle de recherche par deux, nous arrivons à une fourchette de deux ans, puis, à la troisième question, à une fourchette d'un an… et nous avons trouvé la solution, l'âge de l'élève. Cela en exactement trois questions au lieu de huit, et il faut alors faire calculer à l'élève que :

– s'il y avait le choix parmi 1, alors 0 question aurait suffi !
– s'il y avait le choix parmi 2, alors 1 question aurait suffi.

etc. jusqu'à 16, puis lui faire intuiter qu'une question de plus permet de multiplier par deux le nombre de choix à discriminer et lui proposer quelques ordres de grandeur.

– S'il y avait le choix parmi 256, alors 8 questions auraient suffi.
– S'il y avait le choix parmi environ mille, jusqu'à 1024 précisément, mais qu'importe, alors 10 questions auraient suffi.
– S'il y avait le choix parmi environ un million, jusqu'à 1048576 précisément, alors 20 questions auraient suffi.
– S'il y avait le choix parmi environ quatre milliards, jusqu'à 4294967296 précisément, alors 32 questions auraient suffi.

Nous sommes évidemment en train de faire découvrir le plus concrètement du monde la notion d'exponentielle et de logarithme :



Ce calcul est du reste directement lié au codage des entiers positifs, et nous pouvons facilement calculer combien de bits sont nécessaires pour les coder :

Entiers	0 à 1	0 à 3	0 à 7	0 à 15	0 à 255	0 à 1023	0 à environ 4 milliards	0 à plus de 1.8×10^{20}
Codés sur	1 bit	2 bits	3 bits	4 bits	8 bits	10 bits	32 bits	64 bits

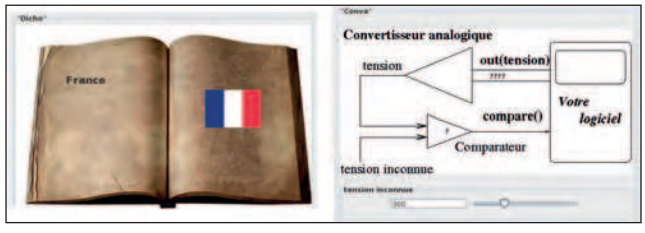
Nos ordinateurs utilisent aujourd'hui souvent 32 ou 64 bits pour coder les nombres entiers, ils peuvent donc coder directement des nombres très grands et, si besoin était, en utilisant plus de bits… des nombres vertigineusement grands !

Pour chercher quelqu'un parmi les 60-70 millions de Françai-se-s, dans le cas oui/non, nous aurons en moyenne à poser 30-35 millions de questions et seulement 26 dans le cas oui/plus/moins. Il faut faire prendre conscience de la différence.

Ce qu'il faut faire voir ici est que la différence entre le cas oui/non et le cas oui/plus/moins est liée à la structure de donnée. Si les Françai-se-s sont « en vrac » sans aucun ordre, alors impossible de définir un plus/moins et il faut poser les questions une à une. Mais dans le cas où il y a un ordre total, par exemple à la suite d'un tri par ordre alphabétique, alors le mécanisme de dichotomie est possible. Le lien entre algorithme et codage de l'information est bien illustré ici.

Ce qu'il faut rendre explicite ici, c'est que nous avons divisé l'intervalle de recherche par deux, donc que nous avons gagné un atome d'information, un bit d'information. Le fait que les informations soient triées par ordre croissant permet de les associer à un codage numérique binaire, « 000 » pour 11, « 001 » pour 12, etc., *isomorphe* aux 8 âges et que nous cherchons finalement un code binaire parmi 8. Du coup, chaque question dichotomique correspond à deviner un bit en commençant par celui de poids fort.

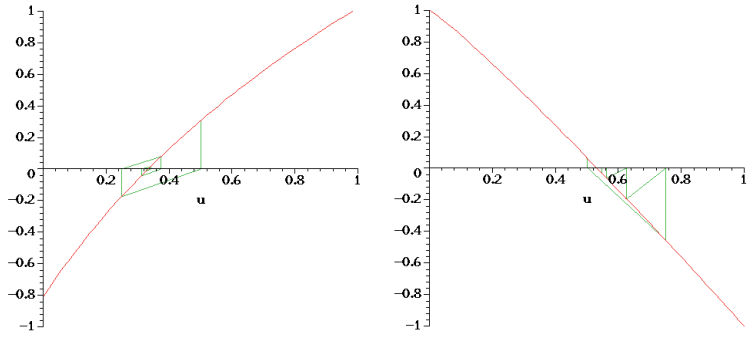
Généricité du principe algorithmique Qu'il s'agisse de trouver un mot dans un dictionnaire ou de considérer un tout autre problème, tel que convertir une tension électrique en valeur numérique, le même principe va s'appliquer. Pour « deviner » la valeur d'une tension électrique continue, un ordinateur numérique doit en général comparer cette valeur à une valeur de référence qu'il va produire en sortie, pour de proche en proche cerner cette valeur, comme le montre le diagramme ci-après :



Là encore, la façon de fonctionner de ces convertisseurs à approximations successives est de procéder de manière dichotomique en divisant l'espace de

recherche par deux à chaque étape. Cela permet d'atteindre rapidement les précisions requises, le millième en 10 étapes, le millionième en 20, comme vu précédemment.

Tous ces problèmes sont finalement reliés au problème mathématique suivant : résoudre une équation de la forme $f(x) = 0$, $x_{min} < x < x_{max}$, où f est une fonction continue monotone dans l'intervalle $]x_{min}, x_{max}[$, donc a au plus une solution, puisque bijection vers un intervalle réel. Si elle change de signe dans cet intervalle, c'est-à-dire $f_{(xmin)}f_{(xmax)} < 0$, elle a une solution unique. Et le même mécanisme de dichotomie permet de résoudre le problème comme illustré sur l'exemple suivant en coupant l'intervalle de recherche en deux, soit en gagnant un bit, à chaque étape :



Les courbes vertes correspondent au test des valeurs positives/négatives des fonctions rouges jusqu'à se rapprocher des zéros recherchés.

Le mécanisme de dichotomie, vu du point de vue du codage, est un bon levier didactique pour faire comprendre les notions essentielles et complémentaires revues ici.

Compléments

La vision probabiliste des choses : un rappel

Si la notion d'information est abordée avec une vision probabiliste, on considère alors que l'on accède aux événements observés par la mesure de leur probabilité d'occurrence. Un événement n'est pas « vrai » ou « faux », mais plus ou moins *probable*.

Pour définir correctement une « tribu » d'événements et leur probabilité nous avons besoin que :
– l'ensemble vide ∅ donc impossible soit un événement, il sera de probabilité $p(\emptyset) = 0$,
– l'ensemble de tout ce qui est possible Ω soit un événement, il sera de probabilité $p(\Omega) = 1$, tandis que les événements E sont des sous-ensembles de tout ce qui est possible $E \subseteq \Omega$.

On ne dit plus ce qui est vrai ou faux sur le monde, on donne une valeur numérique de probabilité qu'un événement se produise. Deux visions différentes du monde se distingueront par ce qui est considéré comme un événement et par la probabilité d'un tel événement.

Il faut pouvoir combiner de manière logique les événements. C'est-à-dire que :
– pour un événement E, son contraire *non* E, c'est-à-dire son complémentaire parmi tout ce qui est possible $\neg E = \Omega - E$ est un événement de probabilité $p(\neg E) = 1 - p(E)$;
– pour un ensemble dénombrable d'événements $\{\ldots E_i, \ldots\}$, leur disjonction, c'est-à-dire leur réunion $E\cup = \cup_i E_i$ soit un événement, qui correspond au fait que E_1 ou E_2 *ou* … se produise
et si les événements sont *incompatibles* – ne peuvent se produire ensemble –, c'est-à-dire si $E_i \cap E_j = \emptyset$, $\forall i, j$, alors la probabilité de la disjonction est une somme $p(E\cup) = \Sigma_i p(E_i)$;
– pour un ensemble d'événements $\{\ldots E_i, \ldots\}$, leur conjonction, c'est-à-dire leur intersection $E\cap = \cap_i E_i$ soit un événement, qui correspond au fait que E_1 et E_2 *et* … se produisent,
et on posera la définition que les événements sont *indépendants*, si et seulement si la probabilité de la conjonction est un produit $p(E\cap) = \Pi_i p(E_i)$;
– tandis qu'un événement E_1 *implique* E_2 si $E_2 \subseteq E_1$; ce qui implique $p(E_1) \geq p(E_2)$. La réciproque est bien sûr fausse.

À partir de ces briques de base, toutes les propriétés logiques s'expriment en termes d'opérations ensemblistes sur les événements, avec les probabilités qui en découlent.

Ce que nous mettons en avant ici en faisant ce bref rappel, c'est qu'il y a substitution des opérations logiques de base *et*, *ou*, *non* par des calculs sur les probabilités, tandis que l'on fait un lien profond entre une vision ensembliste des événements et une vision logique, ce qui fonde la théorie moderne des probabilités.

Bref, la vérité est relative et ne se déduit plus : elle se mesure et se calcule.

Il faut noter que ces définitions, qui semblent naturelles, sont en fait pleines de subtilités techniques : la notion d'indépendance est une définition sur les probabilités, celle d'incompatibilité une notion ensembliste avec une contrainte sur les probabilités, certains éléments sont des axiomes tandis que d'autres s'en déduisent. Tous les sous-ensembles ne sont pas nécessairement des événements, mais cette subtilité intervient dans des espaces infinis continus, les définitions étant pertinentes et valides y compris dans ce cas.

Dans le cas fini ou dénombrable développé ici, il faut juste considérer les valeurs de probabilités $p(e_n)$ sur chaque événement « atomique » ou valeur e_1, … e_n , … tandis que les probabilités sur chaque événement E_i s'en déduisent additivement $P(E_i) = \sum_{e_n \in E_i} p(e_n)$. On notera N le nombre de valeurs.

De la maximisation d'entropie

Pour mettre l'estimation par maximum d'entropie en équation, nous posons que nous disposons de quelques fonctions $\mu_1(e_n), \ldots, \mu_M(e_n)$ qui associent aux probabilités $p(e_1), \ldots, p(e_N)$, des valeurs moyennes de la forme $\sum_n p(e_n)\mu_1(e_n) = v_1, \ldots, \sum_n p(e_n)\mu_M(e_n) = v_M$. Ce sont des valeurs pondérées par les probabilités $p(e_n)$ puisque nous n'accédons pas à une valeur aléatoire, mais uniquement à sa valeur moyenne. Prendre celle d'entropie maximale, en équation, cela s'écrit :

$$\max_{p(e1),\ldots,p(e_N)} -\sum_n p(e_n)\log_2(p(e_n)) \; ; \forall m \sum_n p(e_n)\mu m(e_n) = v_m$$

et un peu de calcul nous permet d'écrire la solution sous la forme :

$$p(e_n) = \frac{1}{Z} e^{\sum_m \lambda_m \mu_m(e_n)}$$

où la constante de normalisation $Z = \sum_n e^{\sum_m \lambda_m \mu_m(e_n)}$ permet que la somme des probabilités $\sum_n p(e_n) = 1$ soit bien normalisée, tandis que les paramètres $\lambda_1, \ldots \lambda_M$ se calculent numériquement à partir des équations de mesure. *e*

Estimation d'information par optimisation

Rendons explicite une méthodologie profonde utilisée pour trouver la solution d'entropie maximale : nous avions des équations de mesure et nous ne savions pas si ces équations avaient une seule solution, plusieurs solutions ou n'avaient aucune solution exacte, mais uniquement approchées. Et cela ne nous a pas empêchés de trouver… la « bonne solution » ! Que s'est-il passé ? Nous avons tout simplement changé de point de vue.

Prenons des équations $z_1(x_1, x_2, \ldots, x_N) = 0$, $z_2(x_1, x_2, \ldots, x_N) = 0$, … $z_M(x_1, x_2, \ldots x_N) = 0$. Nous savons que s'il y a autant d'équations que d'incon-

nues et si les équations sont linéaires et que le système d'équation n'est pas dégénéré, il y a une unique solution, les lycéens en calculent dans le cas de une ou deux équations. S'il y a une équation, mais de degré deux, alors nous pouvons avoir une, deux ou zéro solutions. Les choses se gâtent. Et elles se gâtent vite. Par exemple, N équations non linéaires avec des polynômes ou des fractions rationnelles de degré D peuvent avoir D^N solutions – ce qui est vite énorme – réelles ou toutes complexes, voire à l'infini, ou encore numériquement instables, etc. Et bien entendu il n'y a aucune formule exacte sauf dans quelques cas d'école.

Le changement de point de vue est alors le suivant : (1) au lieu de chercher « la » solution, nous *cherchons une solution proche d'une solution raisonnable* $(\bar{x}_1, \ldots \bar{x}_N)$ et (2) au lieu de résoudre ces équations nous *minimisons l'erreur de ces équations.* Cela se met en équation de la manière suivante : chercher à minimiser un critère de coût

$$\min_{x_1,\ldots x_N} \underbrace{\sum_n \frac{|x_i - \bar{x}_i|}{V_n}}_{\text{plus proche solution}} + \underbrace{\sum_m \frac{|z_m(x_1, \ldots x_N)|}{\varepsilon_m}}_{\text{qui minimise l'erreur}}$$

Détaillons.

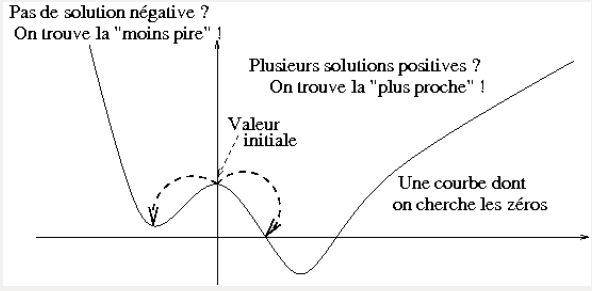
1. Le premier terme est d'autant plus petit que la solution $(x_1, \ldots x_N)$ est proche de notre solution initiale raisonnable $(\bar{x}_1, \ldots \bar{x}_N)$, puisque c'est la somme des valeurs absolues des erreurs, il vaut zéro si nous sommes à la solution raisonnable, plus si nous nous en éloignons. Ici, V_n permet de pondérer la précision.

2. Chaque terme de droite $\frac{|z_m(x1, \ldots x_N)|}{\varepsilon_m}$, avec $\varepsilon_m > 0$ vaut zéro.

si l'équation est vérifiée, tandis qu'une pénalité positive apparaît sinon ; en ajustant ε_m à une plus ou moins grande valeur, ce terme de pénalité est ajusté.

Nous prendrons des valeurs minimales de ε_m si nous voulons forcer l'équation à être vérifiée – un théorème du multiplicateur de Lagrange garantit que chercher les valeurs minimales de ε_m permet, sous de bonnes conditions, d'annuler l'équation $z_m(x_1, \ldots x_N) = 0$ – ou une valeur de ε_m correspondant à la précision de la mesure.

Nous illustrons l'idée ci-dessous, avec une courbe qui n'a pas de racine négative, et plusieurs racines positives. Si nous cherchons une solution négative, la moins mauvaise est celle proche de zéro, et si nous cherchons une solution positive, la meilleure est celle la plus proche de notre valeur initiale.



Qu'avons-nous gagné ici ? En terme de *spécification* nous avons bien défini ce que nous cherchons dans tous les cas. Nous voyons que c'est en ajoutant la notion de solution initiale raisonnable que nous éludons le problème des solutions multiples et que c'est en remplaçant « résoudre » $z_m(x_1,\ldots x_N) = 0$ par minimiser $\frac{|z_m(x_1,\ldots x_N)|}{\varepsilon_m}$ que nous contournons le problème d'équations inexactes ou redondantes.

Qu'avons-nous gagné d'autre ici ? En terme d'*implémentation*, il est beaucoup plus facile de *minimiser* un critère de coût que de résoudre une équation.

Implémentation du paradigme Il suffit de chercher à chaque étape à améliorer un peu la solution actuelle, en essayant quelles solutions voisines sont meilleures, et de réitérer jusqu'à ce que l'information à gagner devienne négligeable. Des résultats mathématiques permettent de savoir quand un tel mécanisme marche et avec quelle performance.

Nous avons donc non seulement un problème bien posé, mais des classes d'algorithmes numériques généraux pour le résoudre efficacement et on peut dire que la grande majorité des algorithmes non linéaires de traitement des signaux de parole ou image, ou d'autres mesures physiques, de calcul de boucles sensori-motrices dans les systèmes artificiels, par exemple dans les robots, sont construits sur ces principes de base. Ne nous trompons pas : les techniques numériques qui marchent à grande échelle sont bigrement complexes, mais les principes généraux sont là.

De plus, à chaque étape, si nous avons gagné quelque chose : trouvé une valeur « moins mauvaise » que la précédente, alors nous avançons vers la solution. En termes de système interactif, les conséquences sont énormes : l'utilisateur ou le robot n'a pas à attendre que… le système trouve une solution, ou non ! À chaque instant, il peut accéder à la moins mauvaise des

solutions, donc prendre une décision sans attendre. Par ailleurs, l'utilisateur peut aider le mécanisme en lui fournissant une solution initiale, ou plusieurs solutions initiales en concurrence, que le calcul va raffiner. Nous sommes au cœur d'une réalité générale des mathématiques du numérique : il est souvent difficile de résoudre un problème global, par exemple trouver une trajectoire, mais beaucoup plus facile de raffiner par le calcul une solution approximative. C'est un des piliers de la conception des systèmes numériques et nous en avons donné les clés ici.

Pour quelques illustrations sur ces points précis : *Pourquoi ne pas confier au hasard ce qui est trop compliqué à estimer ? (2007) Thierry Viéville, http://interstices.info/hasard-estimation* ou *Une solution au problème de la génération de trajectoires (2006) Thierry Viéville, http://interstices.info/generation-trajectoires.*

Contenu en information et neuroscience computationnelle

La complexité d'un cerveau, en termes de nombre d'états, semble être de l'ordre de grandeur du milliard d'ordinateurs que nous avons actuellement dans le monde. Cette complexité du système nerveux, que nous voyons aujourd'hui comme une « machine à traiter de l'information », conduit à deux visions.

Selon la première, le cerveau ne serait rien d'autre qu'une « machine programmable » mais d'une complexité énorme. La seconde met en avant ses propriétés complètement opposées à un processeur d'ordinateur. Le traitement dans le cerveau est complètement distribué, chaque groupe de neurones mettant localement à jour son état, avec émergence d'une fonctionnalité globale de cette population de traitement locaux. *A contrario*, le parallélisme au niveau des processeurs reste aujourd'hui limité et les modèles de machines discutés ici sont intrinsèquement séquentiels et centralisés. De plus, tandis que le processeur effectue ses opérations à une durée inférieure au milliardième de seconde, chaque neurone ne met à jour son état qu'à l'échelle de la milliseconde. Par ailleurs, tandis que le processeur calcule sans erreur – et surtout sans erreur qui ne soit pas fatale ! –, le système nerveux a un comportement intrinsèquement stochastique : les valeurs ne sont pas exactes, mais fluctuent sans que cette fluctuation ne prenne de sens déterministe, ce qui ne gêne pas les traitements, voire aide à explorer de nouvelles solutions lors de certains traitements. Cette profonde fracture entre ces propriétés conduit à penser que c'est dans la confrontation, et non l'assimilation, du cerveau à un processus calculable que se situe la vision scientifique la plus fructueuse.

Pour aller plus loin

Une présentation générale des grands concepts et enjeux des sciences numériques : Gérard Berry, *Pourquoi et comment le monde devient numérique*, Leçon inaugurale au Collège de France (2007). http://www.college-de-france.fr/default/EN/all/inn_te

Pour en savoir plus sur les théories de l'information : Jean-Paul Delahaye, *Théories et théorie de l'information*,)i(nterstices, (2009) http://interstices.info/information, Jean-Paul Delahaye, *Information, complexité et hasard*, Hermès (1999), Jean-Paul Delahaye, *Émergence et complexité de Kolmogorov*, et Jean-Paul Delahaye, *Émergence : de la fascination à la compréhension*, http://www.science-inter.com/Pages%20AEIS/programme.htm auxquels ce chapitre emprunte largement.

Plus d'informations techniques sur les formats de données dans Wikipédia : *Format_de_données*

http://fr.wikipedia.org/wiki/Format_de_données

sur fr.wikipedia.org donc *Image_numérique*

http://fr.wikipedia.org/wiki/Image_numérique

sur Wikipédia pour la description détaillée du codage des couleurs. Pour la description des fichiers *PNG*

http://fr.wikipedia.org/wiki/Portable_Network_Graphics

avec *GIF*

http://fr.wikipedia.org/wiki/Graphics_Interchange_Format

et *JPEG*

http://fr.wikipedia.org/wiki/JPEG

En savoir plus sur les expressions régulières *Expression Régulière*

http://fr.wikipedia.org/wiki/Expression_régulière

dans *Wikipédia*

En savoir plus sur les liens entre cerveau et ordinateur *Le cerveau, un ordinateur ?*

http://interstices.info/cerveau-ordinateur

(2008) Aurélien Liarte, Yves Geffroy,)i(nterstices.

En savoir plus sur les calculs sur ordinateur *Pourquoi mon ordinateur calcule-t-il faux ?*

http://interstices.info/a-propos-calcul-ordinateurs

(2008) Sylvie Boldo,)i(nterstices

En savoir plus sur l'histoire des sciences de l'information

http://www-sop.inria.fr/science-participative.film