ALGORITHMES & IA

Une histoire, une science, un monde
Des recettes antiques aux intelligences artificielles

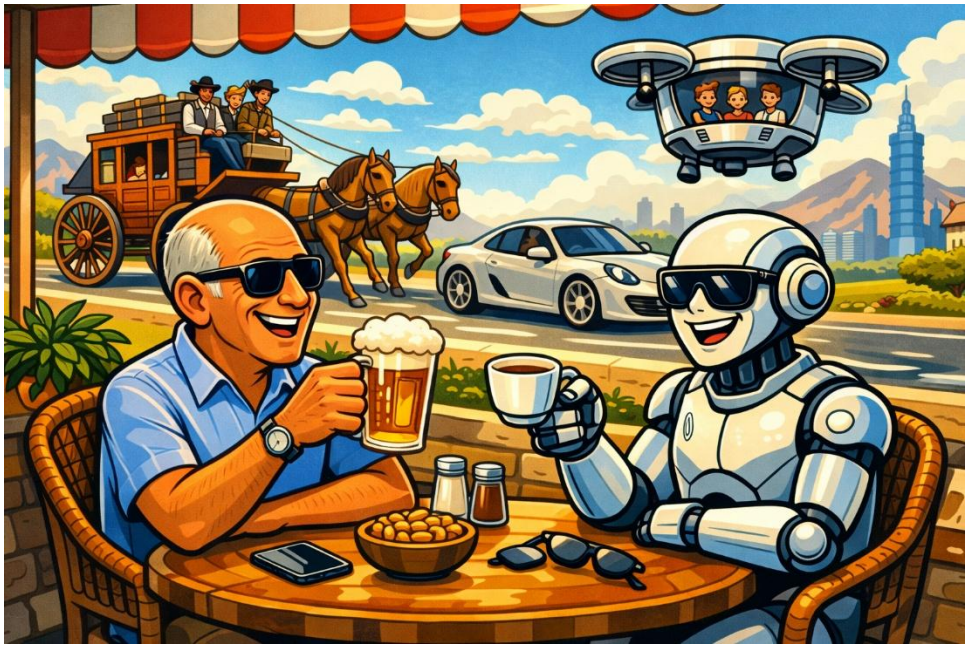
Des tablettes d'argile aux robots en passant par l'IA

À propos de l'auteur

L'auteur de cet ouvrage est ingénieur de formation et occupe la responsabilité d'un site de mathématiques consacré aux remarquables propriétés des nombres, rassemblées sur le site diconombre.fr. Tout au long de son parcours, il s'est passionné pour le calcul, l'arithmétique et la théorie des nombres. Il souligne que la rencontre avec divers procédés de calcul, autrement dit les algorithmes, est une étape naturelle dans l'exploration de ces domaines.

Il rappelle que, pendant des siècles, les recherches manuelles ont été le terrain privilégié des mathématiciens d'autrefois, qui ont ainsi mis en lumière de spectaculaires propriétés des nombres. Toutefois, l'avènement des ordinateurs et la puissance de calcul inlassable qu'ils offrent ont permis d'amplifier ces découvertes et de révéler des aspects insoupçonnés de la structure des nombres.

L'auteur, dont la formation en électronique et en informatique remonte aux années 1960, a mis à profit ses compétences techniques dès le début de sa carrière, notamment en pratiquant le langage machine, l'assembleur et Algol. Il s'est ensuite rapidement orienté vers le management, sans pour autant délaisser sa passion pour la programmation, qu'il a poursuivie dans le cadre de ses recherches sur les nombres, principalement avec Maple et Python. Tout au long de son parcours, il a également expérimenté différents langages de programmation tels que Basic, Pascal, C++, Logo, Lisp, et même Scratch avec ses petits-enfants.



Préface

Les algorithmes sont devenus les mécanismes invisibles de notre quotidien : ils trient nos messages, orientent nos trajets, recommandent nos lectures, pilotent nos machines. Aujourd'hui, ils apprennent, dialoguent, créent et prennent place au cœur de l'intelligence artificielle. Mais derrière cette omniprésence, une question essentielle demeure : que sont réellement les algorithmes ? Comprendre l'IA moderne — qu'il s'agisse d'apprentissage automatique, de réseaux neuronaux, de systèmes symboliques ou d'algorithmes évolutionnaires — exige d'abord de comprendre l'histoire longue et les fondements conceptuels qui ont façonné ces outils.

Loin d'être une invention contemporaine, les algorithmes plongent leurs racines dans les civilisations antiques. Des scribes égyptiens aux mathématiciens babyloniens, d'Euclide à Al Khwarizmi, de Fibonacci aux pionniers de l'informatique, une même quête traverse les siècles : décrire des méthodes claires, reproductibles, capables de résoudre des problèmes. Au fil du temps, ces méthodes se sont raffinées, les idées se sont structurées, les machines ont émergé, et les algorithmes sont devenus une science à part entière.

Ce livre propose un voyage à travers cette histoire, mais aussi une exploration des notions qui en découlent : complexité, structures de données, programmation, paradigmes de pensée. Il montre comment les algorithmes se déclinent aujourd'hui en familles variées — déterministes ou probabilistes, symboliques ou statistiques, explicites ou apprenants — et comment ces formes multiples irriguent l'intelligence artificielle contemporaine.

Au-delà de la connaissance, l'ambition est d'offrir une vision cohérente : comprendre comment les idées anciennes éclairent les technologies modernes, comment les fondements théoriques nourrissent les pratiques actuelles, et comment les algorithmes — qu'ils soient intégrés à nos outils ou incarnés dans des IA génératives — transforment notre rapport au savoir, à la décision et à la vie moderne.

J'espère que ce livre donnera à chacun le goût des algorithmes : leur rigueur, leur créativité, leur puissance, et parfois même, leur poésie.

ALGORITHME, subst. masc.

A.– *MATHÉMATIQUES*

1. *Anciennement*

a) Système de numération décimale en chiffres arabes.

b) ,,Ensemble des règles du calcul des nombres écrits dans le système décimal (les « quatre règles »).`` (LAL. 1968).

c) Ensemble des règles opératoires intervenant dans toute espèce de calcul. *L'algorithme de la division (Lar. encyclop.), l'algorithme de la multiplication* (FOULQ.-ST-JEAN 1962) :

- 1. Les mathématiciens trouveraient aussi dans la théorie indienne des nombres des **algorithmes** fort originaux. E. RENAN, *L'Avenir de la science,* 1890, p. 508.

2. *Sens mod.* Ensemble de symboles et de procédés propres à un calcul *algorithme du calcul intégral, algorithme du calcul des sinus, algorithme des puissances, algorithme des différences... (Lar. 19e); p. ext.* ,,ensemble de formules, de signes et de conventions accessibles aux seuls initiés`` (QUILLET 1965) :

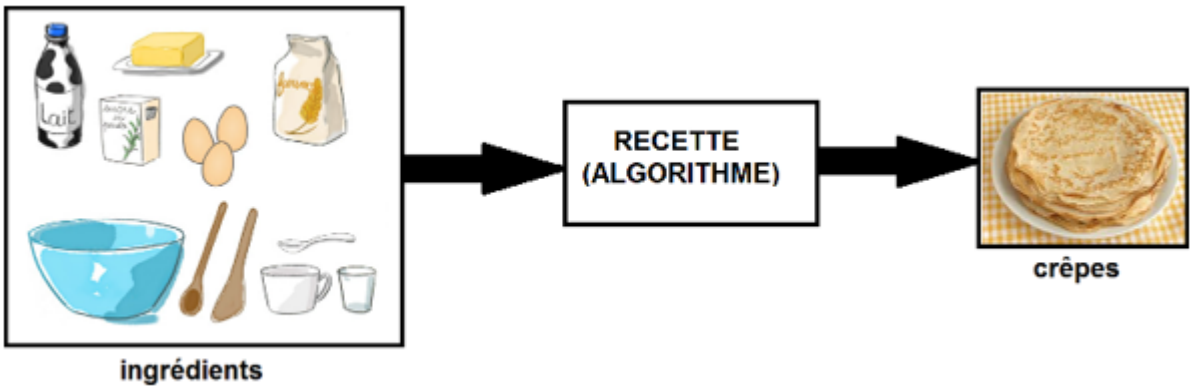
Définition par le CNTRL (Centre National de Ressources Textuelle et Lexicales)

Table des matières

1 Approche : pourquoi les algorithmes ?

Les algorithmes sont présents dans presque toutes les activités numériques, même lorsque nous n'en avons pas conscience. Ils permettent de résoudre des problèmes de manière systématique, en suivant une suite d'instructions précises.

Ce chapitre présente l'idée centrale d'un algorithme, son rôle dans le traitement de l'information et la manière dont il se distingue d'un programme ou d'une machine. Il offre une première vision d'ensemble pour comprendre pourquoi les algorithmes sont devenus indispensables dans le monde moderne**.**



Le saviez-vous ?	**Les algorithmes autour de nous** Un algorithme, c'est comme une recette de cuisine Prenons un exemple ultra-simple : préparer un bol de céréales. 1. Prendre un bol. 2. Ouvrir le paquet de céréales. 3. Verser les céréales dans le bol. 4. Ouvrir le frigo. 5. Sortir le lait. 6. Verser le lait sur les céréales. 7. Manger. Cette liste, c'est un algorithme. Pourquoi ? Parce que : • C'est une suite d'instructions précises. • Chaque étape est exécutable sans ambiguïté. • Si tu suis les étapes dans l'ordre, tu obtiens toujours le même résultat (un bol de céréales prêt à être mangé). Et si on oublie une étape ? Imagine que tu verses le lait avant les céréales… Le résultat ne sera pas le même ! Les algorithmes, comme les recettes, doivent être suivis à la lettre pour fonctionner.

1.1 Rôle des algorithmes dans la vie quotidienne

Les algorithmes interviennent dans des situations très diverses, souvent sans que nous en ayons conscience. Lorsqu'un téléphone calcule un itinéraire, il applique une méthode pour trouver le chemin le plus court. Lorsqu'un site de commerce en ligne propose un produit, il utilise une procédure pour analyser nos préférences. Lorsqu'un moteur de recherche classe des millions de pages, il suit une suite d'étapes pour déterminer lesquelles sont les plus pertinentes.

Ces traitements ne sont pas improvisés. Ils reposent sur des méthodes précises, conçues pour produire un résultat fiable à partir d'informations disponibles. Leur présence est devenue indispensable parce que les volumes de données à traiter dépassent largement ce qu'un humain pourrait gérer. Les algorithmes permettent d'automatiser des tâches répétitives, d'accélérer des calculs complexes et d'assurer une cohérence que l'on ne pourrait pas maintenir manuellement.

Cette omniprésence ne se limite pas aux outils numériques. Les algorithmes interviennent dans les transports, la santé, la finance, l'énergie, l'industrie. Ils organisent des flux, détectent des anomalies, optimisent des ressources. Ils contribuent à la sécurité, à la communication et à la prise de décision. Leur rôle dépasse largement le cadre technique : ils influencent la manière dont nous nous informons, dont nous travaillons et parfois même dont nous percevons le monde.

Le saviez-vous ?	**Des algorithmes dans la nature et le quotidien** Les algorithmes ne sont pas une invention humaine. La nature en regorge ! • Les fourmis : Quand une fourmi trouve de la nourriture, elle laisse une trace chimique (phéromone) pour guider ses congénères. C'est un algorithme de recherche de chemin. • Les abeilles : Leur « danse » pour indiquer l'emplacement d'une source de nectar est un algorithme de communication. • Ton cerveau : Quand tu reconnais un visage ou évites un obstacle en vélo, ton cerveau suit des règles logiques (même si tu n'en as pas conscience). Exercice mental : *Peux-tu imaginer un algorithme pour…* • Traverser la rue en sécurité ? • Choisir quoi regarder sur Netflix ? • Ranger ta chambre ? *(Réponse : Oui ! Ce sont des séquences d'actions logiques.)*

1.2 Définition intuitive et formelle

Intuitivement, un algorithme est une méthode pour résoudre un problème. C'est une suite d'étapes ordonnées qui, appliquées à des données, permettent d'obtenir un résultat.

Cette idée est simple : une recette de cuisine, une méthode de calcul ou un protocole scientifique peuvent être vus comme des algorithmes. Ce qui compte, c'est que les étapes soient claires, finies et reproductibles.

Le saviez-vous ?	**Les algorithmes en bref** Les algorithmes ne sont pas nés avec les ordinateurs. Ils existent depuis l'Antiquité. Euclide, par exemple, utilisait déjà une méthode précise pour simplifier les fractions (trouver le plus grand diviseur commun de deux nombres). C'était un algorithme, même si le mot n'existait pas encore. Ce mot vient d'ailleurs du nom d'un savant perse du IX^{e} siècle : al-Khwarizmi. En latin, son nom est devenu Algoritmi, et c'est ainsi que le terme "algorithme" est apparu. Un algorithme, c'est simplement une suite d'étapes claires pour résoudre un problème. On peut en écrire pour faire un calcul, trier des objets, préparer une recette ou décider d'un chemin à suivre. Avant de programmer quoi que ce soit, on commence toujours par imaginer l'algorithme. Les algorithmes numériques, eux, travaillent sur des nombres. Ils permettent d'effectuer des calculs de manière sûre et systématique. Et le plus étonnant, c'est qu'on n'a pas besoin d'un ordinateur pour exécuter un algorithme. Une feuille, un crayon… et c'est parti.

Sur le plan formel, un algorithme est une procédure définie de manière non ambiguë, qui transforme des données d'entrée en données de sortie. Il doit être suffisamment précis pour qu'un exécutant — humain ou machine — puisse suivre chaque étape sans interprétation personnelle. Il doit également se terminer après un nombre fini d'opérations. Cette exigence garantit que l'algorithme ne se perd pas dans une boucle infinie et qu'il produit un résultat.

Cette définition formelle permet d'étudier les algorithmes de manière rigoureuse. Elle offre un cadre pour analyser leur validité, leur efficacité et leurs limites. Elle permet aussi de comparer différentes méthodes pour résoudre un même problème et de choisir celle qui convient le mieux selon le contexte.

Autour de L'ALGORITHME	**Dans la vie quotidienne, nous utilisons plusieurs mots pour désigner une suite d'actions à accomplir. Pourtant, chacun porte une nuance qui éclaire ce qu'est réellement un algorithme.**
RECETTE	**Une recette est la forme la plus familière : elle guide l'action, mais laisse une large place à l'intuition. On peut « ajouter un peu de sel », « laisser cuire jusqu'à ce que ce soit doré ». La réussite dépend du jugement humain. Une recette est donc une procédure souple, tolérante aux approximations.**
MÉTHODE	**Une méthode est plus abstraite : elle décrit une manière générale de s'y prendre, un cadre intellectuel plutôt qu'une suite d'étapes précises. La méthode scientifique, par exemple, indique comment raisonner, pas comment exécuter chaque geste.**
PROCÉDÉ	**Un procédé se situe déjà dans un registre plus technique. Il vise la reproductibilité : on suit une séquence d'opérations pour obtenir un résultat constant. Mais il peut encore laisser une part d'appréciation à l'opérateur, comme dans un atelier ou un laboratoire.**
PROTOCOLE	**Le protocole, lui, réduit presque toute interprétation. Il impose un ordre strict, des conditions définies, des règles à respecter sans déviation. Dans un protocole médical ou un protocole réseau, chaque étape doit être exécutée exactement comme prévu.**
ALGORITHME	**L'algorithme est la forme la plus rigoureuse et la plus abstraite de toutes. C'est une procédure définie avec une précision telle qu'elle peut être exécutée mécaniquement, sans interprétation humaine, et qui garantit un résultat en un nombre fini d'étapes. Là où une recette accepte les « à peu près », un algorithme exige la clarté absolue. Là où une méthode donne une direction, un algorithme donne chaque pas.**
PROGRAMME	**Le programme informatique est la traduction concrète d'un algorithme en une suite d'instructions dans un langage compréhensible par une machine. Il ne tolère aucune ambiguïté : la machine exécute ce qui est écrit, rien de plus, rien de moins.**
Comprendre ces nuances permet de saisir pourquoi l'informatique — et plus encore l'intelligence artificielle — repose sur des instructions parfaitement définies. L'algorithme n'est pas seulement une suite d'actions : c'est une pensée mise en forme de manière à être exécutée sans équivoque.	

ANECDOTE

Euclide, le premier « support technique » de l'histoire

On raconte que le jeune roi Ptolémée Ier, agacé par la difficulté des Éléments, demanda à Euclide s'il n'existait pas un chemin plus simple pour apprendre la géométrie.

Euclide lui répondit : **« Il n'existe pas de voie royale pour la géométrie. »**

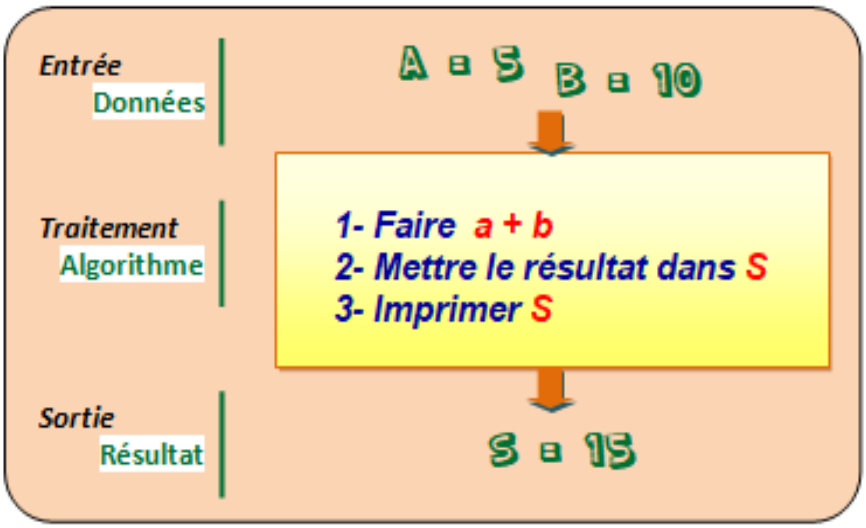
C'est dans ce même ouvrage que se trouve l'un des premiers algorithmes connus : l'algorithme d'Euclide pour calculer le plus grand diviseur commun.

Un roi voulait une solution rapide ; Euclide lui a offert un algorithme.

1.3 Notions clés : données, instructions, exécution, abstraction

Pour comprendre un algorithme, il faut distinguer plusieurs notions fondamentales. Les données représentent les informations sur lesquelles l'algorithme travaille. Elles peuvent être des nombres, des textes, des images ou toute autre forme d'information structurée. Les instructions sont les actions élémentaires que l'algorithme applique à ces données : comparer, additionner, déplacer, sélectionner.

L'exécution désigne le déroulement effectif de ces instructions. Lorsqu'un algorithme est exécuté par une machine, chaque étape est traduite en opérations simples que le processeur peut réaliser. Cette exécution peut être très rapide, mais elle reste fidèle à la méthode définie. La machine ne prend aucune initiative : elle applique strictement les instructions dans l'ordre prévu.



Un algorithme très simple : exécuter une addition

L'abstraction joue un rôle essentiel dans la conception des algorithmes. Elle permet de se concentrer sur la logique du traitement sans se soucier des détails techniques.

Un algorithme peut être décrit de manière générale, indépendamment du langage de programmation ou du matériel utilisé. Cette séparation facilite la compréhension, la communication et la réutilisation des méthodes.

ANECDOTE

L'algorithme qui a sauvé des rhinocéros

En Afrique du Sud, les braconniers utilisent des schémas prévisibles pour traquer les rhinocéros.

Des chercheurs ont développé un algorithme capable d'analyser les déplacements des animaux, les habitudes des braconniers et les caractéristiques du terrain. L'algorithme propose ensuite des patrouilles optimisées, qui interceptent les braconniers avant qu'ils n'agissent.

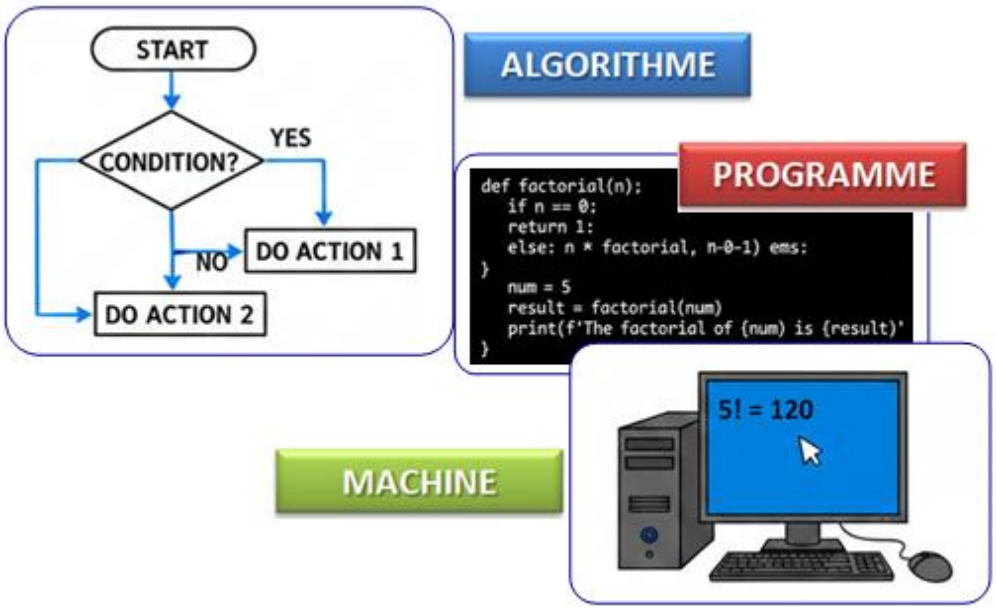
Résultat : dans certaines réserves, le braconnage a chuté de plus de 60 %.

Un algorithme qui protège une espèce menacée, en anticipant les comportements humains.

1.4 Distinction algorithme / programme / machine

Il est important de distinguer clairement l'algorithme du programme et de la machine, comme un ordinateur. L'algorithme est l'idée, la méthode. Il existe indépendamment de tout langage informatique. Le programme est la traduction de cette méthode dans une forme que l'ordinateur peut interpréter.

Il dépend d'un langage, d'une syntaxe et d'un environnement d'exécution. La machine, enfin, est l'outil matériel qui exécute le programme.



Les trois niveaux du système de traitement des informations

Cette distinction permet de comprendre qu'un même algorithme peut être programmé dans différents langages, et qu'un même programme peut être exécuté sur différentes machines. Elle montre aussi que l'efficacité d'un traitement dépend à la fois de la qualité de l'algorithme, de la manière dont il est programmé et des capacités de la machine. Un algorithme bien conçu peut être ralenti par une mauvaise implémentation (réalisation), et un programme optimisé peut être limité par le matériel.

Comprendre ces trois niveaux aide à analyser les performances, à identifier les sources d'erreur et à améliorer les solutions. Cela permet également de mieux saisir le rôle de chacun dans la chaîne de traitement : l'algorithme apporte la logique, le programme apporte la forme, la machine apporte la puissance.

Le saviez-vous ?	**Algorithmes humains vs algorithmes machines** Les humains utilisent des algorithmes sans même y penser : • Faire ses lacets : Une suite de boucles et de nœuds précis. • Jouer à Pierre-Feuille-Ciseaux : « Si l'adversaire choisit pierre, je choisis feuille. » • Trier ses vêtements : « Les t-shirts d'un côté, les pantalons de l'autre. » Mais les machines (ordinateurs, robots, smartphones) ont besoin d'algorithmes ultraprécis. Pourquoi ? • Un humain peut deviner ou improviser (ex. : « Ce gâteau a l'air cuit, même si la recette dit 10 minutes de plus »). • Une machine ne devine pas. Si tu lui dis de cuire un gâteau 10 minutes, elle le fera même si ça brûle. Exemple marrant : Un robot aspirateur suit un algorithme pour nettoyer une pièce. Si tu le places dans une pièce sans murs, il tournera en rond indéfiniment parce qu'il n'a pas d'instruction pour gérer ce cas !

1.5 Conclusion du chapitre

Ce premier chapitre a posé les bases nécessaires pour aborder l'étude des algorithmes. Il a montré leur importance dans la vie quotidienne, défini leur nature, présenté les notions essentielles qui les structurent et clarifié leur relation avec les programmes et les machines. Les chapitres suivants approfondiront ces idées en introduisant les outils de représentation, l'histoire des méthodes, les principes de programmation et les techniques d'implémentation. L'objectif est de construire une compréhension solide et progressive du rôle des algorithmes dans l'informatique et dans le monde contemporain.

ANECDOTE	**L'algorithme qui prédit les embouteillages… avant qu'ils ne se forment** Les systèmes modernes de navigation ne se contentent plus de mesurer le trafic : ils le prédisent. En analysant les vitesses, les habitudes des conducteurs, les événements locaux, les algorithmes anticipent où un bouchon va apparaître… parfois 10 à 15 minutes avant qu'il ne se forme réellement. Cette capacité prédictive repose sur des modèles mathématiques complexes, mais aussi sur des millions de données anonymisées. L'utilisateur voit simplement une route rouge sur sa carte. Derrière, un algorithme a vu l'avenir.

2 Comprendre un algorithme : représentations et outils

Comprendre un algorithme, ce n'est pas seulement lire une suite d'instructions. C'est réussir à en suivre la logique, à imaginer son déroulement et à vérifier qu'il atteint bien son but.

Pour cela, on utilise des outils simples : des schémas, des diagrammes de flux ou encore du pseudo-code. Ils permettent de représenter un traitement étape par étape, sans se perdre dans les détails d'un langage de programmation qui sera beaucoup plus exigeant en formalisme.

Ce chapitre montre comment ces représentations aident à analyser un algorithme, à en discuter avec d'autres et à mieux comprendre ce qui se passe dans la machine.

Le saviez-vous ?	**Représentations des algorithmes** Un **diagramme de flux** est un dessin qui montre le chemin que suit un traitement : on part d'un point de départ, on avance étape par étape, et on bifurque quand il faut prendre une décision. C'est une manière simple de "voir" l'algorithme. Un **organigramme** est un autre mot pour désigner ce type de schéma. On y retrouve les mêmes formes : rectangles pour les actions, losanges pour les choix, flèches pour le sens du parcours. C'est le terme le plus utilisé dans les manuels scolaires. Le mot anglais **flow-chart** veut dire exactement la même chose. On le rencontre souvent dans les documents techniques ou sur Internet. C'est simplement la version internationale de l'organigramme. Le **pseudocode** ressemble à un texte structuré : il décrit un algorithme avec des mots simples, sans utiliser un vrai langage de programmation.

2.1 Le rôle des représentations dans la compréhension

Lorsqu'un algorithme est présenté sous forme de texte brut, il peut être difficile d'en saisir immédiatement la logique. Les représentations graphiques ou semi-formelles offrent une vision plus intuitive. Elles mettent en évidence les étapes essentielles, les choix possibles, les répétitions et les transformations appliquées aux données. Cette mise en forme aide à repérer les erreurs, à simplifier certaines parties et à comparer différentes méthodes pour résoudre un même problème.

Ces représentations jouent également un rôle pédagogique important. Elles permettent d'introduire progressivement la notion d'algorithme, sans imposer d'emblée la syntaxe d'un langage informatique. Elles servent de passerelle entre l'idée générale et l'implémentation concrète.

2.2 Les trois piliers d'un algorithme

Tous les algorithmes, même les plus complexes, reposent sur trois briques de base **:** la séquence, la décision, et la répétition.

La séquence : « Fais ça, puis ça, puis ça »

C'est la forme la plus simple : **une liste d'instructions exécutées les unes après les autres**.

Exemple : **Allumer une bouilloire**

- ➔ Brancher la bouilloire.
- ➔ Remplir d'eau.
- ➔ Appuyer sur le bouton « On ».

Schématiquement :

```
➔ Étape 1
➔ Étape 2
➔ Étape 3
```

La décision : « Si… alors… »

Les algorithmes deviennent intéressants quand ils **peuvent choisir** entre plusieurs actions.

Exemple : Décider quoi porter aujourd'hui

- ➔ Regarder par la fenêtre.
- ➔ S'il pleut : prendre un parapluie et un manteau.
- ➔ Sinon : prendre des lunettes de soleil.

Schématiquement :

```
Si [condition]
alors [action A]
Sinon [action B]
```

La répétition : « Fais ça jusqu'à ce que… »

Les boucles permettent de **répéter une action** tant qu'une condition est vraie.

Exemple : **Remuer une soupe**

1. Prendre une cuillère.
2. Tant que la soupe n'est pas homogène :
    - Remuer.
    - Vérifier

Schématiquement :

```
Tant que
[condition]
faire [action]
```

Autres exemples :

- **Un compte à rebours** : « Tant que le temps > 0, afficher le temps restant. »
- **Un jeu vidéo** : « Tant que le joueur a des vies, continuer la partie. »

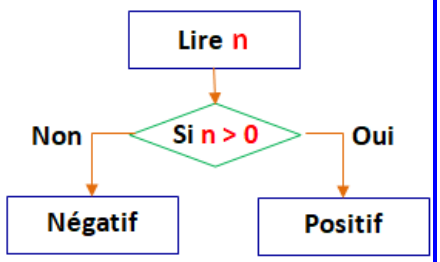
2.3 Le pseudocode : une description simple et précise

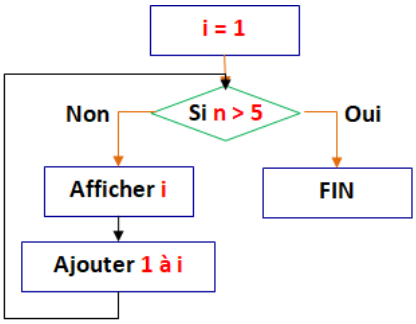
Le pseudocode est une manière d'écrire un algorithme en utilisant des mots du langage courant, associés à quelques conventions simples. Le texte du pseudocode ressemble à ce qui est noté « schématiquement » dans les tableaux ci-dessus. Il ne suit pas les règles strictes d'un langage de programmation, mais il reste suffisamment précis pour être compris par n'importe quel lecteur familiarisé avec la logique algorithmique. Il permet de décrire les étapes d'un traitement sans se soucier des détails techniques comme les types de variables ou la gestion de la mémoire.

Cette forme est particulièrement utile pour concevoir un algorithme avant de le programmer. Elle permet de se concentrer sur la logique, de tester mentalement le déroulement et d'identifier les points à améliorer. Une fois le pseudocode validé, la traduction dans un langage comme Python devient plus simple et plus sûre.

EXEMPLE

Pseudocode	Organigramme

Déterminer si un nombre est positif ou négatif
`DEBUT`
`  LIRE n`
`  SI n > 0 ALORS`
`  AFFICHER "Le nombre est positif"`
`  SINON`
`  AFFICHER "Le nombre est négatif ou nul"`
`  FIN SI`
`FIN`
	

Affiche les nombres de 1 à 5
`DEBUT`
`  i ← 1`
`  TANT QUE i ≤ 5 FAIRE`
`  AFFICHER i`
`  i ← i + 1`
`  FIN TANT QUE`
`FIN`
	

2.4 Les diagrammes de flux : visualiser le déroulement

Les diagrammes de flux, ou organigrammes, représentent un algorithme sous forme de blocs reliés par des flèches. Chaque bloc correspond à une action, une décision ou un début ou fin de traitement. Cette représentation visuelle met en évidence la structure de l'algorithme : enchaînement des étapes, bifurcations, boucles, conditions.

Les diagrammes de flux sont particulièrement utiles pour comprendre des algorithmes comportant plusieurs choix possibles ou des répétitions. Ils permettent de suivre le cheminement d'une donnée à travers les différentes étapes et de vérifier que tous les cas sont bien pris en compte. Ils sont également utilisés dans l'industrie pour documenter des processus complexes.

2.5 Les tables de décision et les traces d'exécution

Pour analyser le comportement d'un algorithme, il est souvent utile de simuler son exécution sur un exemple concret. La trace d'exécution consiste à suivre pas à pas les valeurs des variables et les étapes parcourues. Cette méthode permet de vérifier que l'algorithme produit bien le résultat attendu et de repérer les erreurs logiques.

Les tables de décision, quant à elles, permettent de résumer les différents cas possibles dans un algorithme comportant des conditions. Elles aident à s'assurer que toutes les situations ont été

envisagées et que les choix sont cohérents. Ces outils sont particulièrement utiles dans les domaines où la fiabilité est essentielle, comme la finance ou l'aéronautique.

Le saviez-vous ?

Exemple de TABLE de DÉCISION

Imaginons un petit algorithme qui décide si une personne peut bénéficier d'une réduction dans un musée :

- Réduction si **âge < 18**
- Réduction si **âge ≥ 65**
- Sinon, pas de réduction

On peut résumer les cas possibles dans une **table de décision** :

- **Table de décision — Réduction au musée**

Condition	Cas 16 ans	Cas 70 ans	Cas 40 ans
Âge < 18	Oui	Non	Non
Âge ≥ 65	Non	Oui	Non
Décision	**Réduction**	**Réduction**	**Pas de réduction**

Cette table permet de vérifier que **tous les cas possibles** ont été envisagés et que la logique est cohérente.

Est-ce la même chose qu'une TABLE de VÉRITÉ ?

Pas exactement, même si les deux se ressemblent.

Table de décision

- Utilisée en algorithmique, gestion, assurance, finance…
- Sert à lister les cas possibles et les actions associées.
- Peut mélanger plusieurs conditions, même non logiques (âge, statut, type d'objet…).

Table de vérité

- Utilisée en logique booléenne et en électronique.
- Sert à montrer le résultat d'une expression logique (ET, OU, NON…).
- Les valeurs sont uniquement Vrai / Faux.

En résumé

Une **table de décision** est un outil d'analyse algorithmique.
Une **table de vérité** est un outil de logique formelle.
Elles se ressemblent, mais ne servent pas au même objectif.

2.6 L'importance de la rigueur dans la description

Décrire un algorithme de manière rigoureuse est indispensable pour garantir sa compréhension et sa reproductibilité. Une description trop vague peut conduire à des interprétations différentes, ce qui compromet la fiabilité du traitement. Les représentations introduites dans ce chapitre permettent d'éviter ces ambiguïtés en imposant une structure claire et en mettant en évidence les étapes essentielles.

Cette rigueur est également nécessaire pour analyser la complexité d'un algorithme, vérifier sa terminaison ou prouver sa correction. Elle constitue la base de l'étude scientifique des algorithmes, qui sera approfondie dans les chapitres suivants.

2.7 Conclusion du chapitre

Ce chapitre a présenté les principaux outils permettant de comprendre et de représenter un algorithme.

Le pseudocode, les diagrammes de flux, les traces d'exécution et les tables de décision offrent des moyens complémentaires pour visualiser la logique d'un traitement et en vérifier la cohérence. Ils constituent une étape essentielle entre l'idée initiale et l'implémentation dans un langage de programmation.

Le chapitre suivant montrera comment ces méthodes se sont développées au fil de l'histoire et comment elles ont évolué avec l'apparition des ordinateurs et de l'intelligence artificielle.

3 Histoire des algorithmes : des recettes antiques à l'intelligence artificielle

Les algorithmes ne sont pas nés avec l'informatique. Ils apparaissent dès l'Antiquité, dans les méthodes de calcul et les procédures mathématiques, puis se développent au fil des siècles à mesure que les sciences se structurent. L'arrivée des machines programmables marque un tournant : les algorithmes peuvent désormais être exécutés automatiquement. Aujourd'hui, l'intelligence artificielle introduit des formes nouvelles d'algorithmes capables d'apprendre à partir de données. Retracer cette évolution permet de comprendre comment une idée simple — décomposer un problème en étapes — a progressivement conduit aux technologies qui façonnent notre monde.

Bien avant que les mathématiciens ne parlent de « démonstrations » ou que les informaticiens n'inventent le mot « algorithme », les civilisations antiques manipulaient déjà des procédures systématiques pour résoudre des problèmes concrets. Le calcul n'était pas un domaine théorique : c'était un outil de survie, d'administration, de commerce, d'astronomie. Et pour qu'un calcul soit fiable, il fallait qu'il soit reproductible. C'est là que naissent les premières méthodes, ces suites d'étapes qui, sans être encore formalisées, possèdent déjà l'essence de l'algorithmique.

Plan :

1) **De l'Antiquité à 500 ap. J.-C. :** Les premières procédures de calcul émergent bien avant la formalisation mathématique.
2) **Le Moyen Âge (500–1500) :** Les méthodes de calcul circulent, se transforment et s'organisent entre trois mondes savants.
3) **Renaissance et révolution scientifique (1500–1800) :** Les algorithmes deviennent des outils explicites au service d'une science en plein essor.
4) **Le XIX^{e} siècle, un tournant décisif :** La logique, l'abstraction et la mécanisation redéfinissent la notion même d'algorithme.
5) **Le XX^{e} siècle marque la naissance de l'informatique moderne :** L'algorithme devient un objet scientifique central, fondant les machines programmables.
6) **Le XXI^{e} siècle est marqué par l'ère numérique et l'intelligence artificielle :** Les algorithmes s'invitent partout et structurent silencieusement la société connectée.

3.1 De l'Antiquité à 500 ap. J.-C. :

Les premières procédures de calcul émergent bien avant la formalisation mathématique.

L'Antiquité, cette période qui s'étend de l'émergence des premières civilisations jusqu'à la chute de l'Empire romain en 476 ap. J.-C., a vu naître les premiers algorithmes de l'histoire humaine.

Ces procédures de calcul, souvent gravées sur des tablettes d'argile, des papyrus ou des bandes de bambou, ont jeté les bases des mathématiques modernes.

Voici une description de cette époque fondatrice, où l'on découvrira comment les Babyloniens, les Égyptiens, les Grecs, les Indiens et les Chinois ont, chacun à leur manière, inventé des méthodes de résolution de problèmes qui restent d'actualité aujourd'hui.

Le saviez-vous ?

Aux origines du comptage : l'os de Lebombo

Découvert dans les montagnes du Lebombo, à la frontière entre l'actuel Swaziland et l'Afrique du Sud, un fragment d'os de babouin long d'une dizaine de centimètres occupe une place singulière dans l'histoire des mathématiques.

Daté d'environ **42 000 ans**, il porte une série d'entailles régulières, soigneusement alignées, dont la disposition ne doit rien au hasard. Ce petit objet, connu sous le nom d'os de Lebombo, est souvent présenté comme l'un des plus anciens témoignages d'une activité de comptage chez l'être humain.

Les entailles, au nombre d'une vingtaine, semblent avoir été réalisées en plusieurs sessions, comme si l'objet avait servi de registre rudimentaire.

On ignore ce que ces marques représentaient : un cycle lunaire, des jours, des prises de chasse, des objets, des personnes. Mais leur régularité suggère une intention de quantification, un effort pour fixer une suite d'événements ou de quantités dans une forme matérielle.

L'os de Lebombo témoigne ainsi d'un moment où la mémoire humaine a commencé à s'appuyer sur des supports extérieurs pour conserver des informations numériques. Cependant, qualifier cet os de « premier témoignage du comptage » demande prudence. Il est le plus ancien artefact connu présentant des marques interprétables comme un système de comptage, mais il n'est probablement pas le premier acte de comptage de l'humanité. Les gestes de quantification — compter des jours, des objets, des membres d'un groupe — sont bien plus anciens que les objets qui en gardent la trace. L'os de Lebombo représente donc la plus ancienne preuve matérielle conservée, non l'origine absolue du comptage.

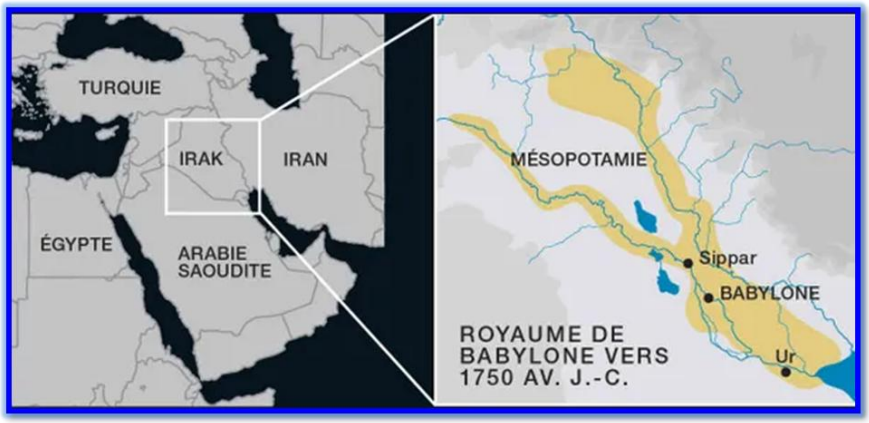
D'autres objets, comme l'os d'Ishango découvert près du lac Édouard et daté d'environ 20 000 ans, montrent des systèmes d'entailles plus complexes, parfois interprétés comme des prémices d'arithmétique. Mais l'os de Lebombo demeure le témoin le plus ancien d'une volonté humaine de fixer une suite numérique dans la matière. Il marque un seuil : celui où la pensée quantitative cesse d'être seulement mentale pour devenir visible, transmissible, durable.

Ce fragment d'os, fragile et discret, rappelle que les mathématiques ne sont pas nées d'un éclair de génie, mais d'une longue histoire de gestes simples, répétés, inscrits dans la matière bien avant d'être formulés dans des symboles. Il est l'un des premiers signes tangibles d'une intuition fondamentale : celle que le monde peut être compté, ordonné, mesuré.

3.1.1 Mésopotamie : L'argile et le calcul

Dès le III[e] millénaire avant notre ère, dans le croissant fertile de Mésopotamie (actuel Irak), les Sumériens, puis les Babyloniens, développent une culture mathématique sophistiquée, motivée par des besoins concrets : gestion des récoltes, construction de temples, commerce, et astronomie.

Les archéologues ont exhumé plus de 400 tablettes d'argile, datant de 2600 à 500 av. J.-C., qui révèlent une maîtrise remarquable des nombres et des algorithmes.



Le système sexagésimal et les tables de multiplication

Les Babyloniens utilisent un système de numération en base 60 (sexagésimal), hérité des Sumériens. Ce choix n'est pas anodin : 60 est un nombre « hautement composé », divisible par 2, 3, 4, 5, 6, 10, 12, 15, 20, et 30, ce qui facilite les calculs de fractions et les conversions. Dès 2600 av. J.-C., ils dressent des tables de multiplication, de division, et des listes de carrés et de cubes, gravées sur des tablettes d'argile. Ces tables servaient de référence pour les scribes, qui pouvaient ainsi résoudre rapidement des problèmes de partage de terres, de calcul de salaires, ou de conversion de mesures.

Un exemple célèbre est la tablette Plimpton 322 (vers 1800 av. J.-C.), qui contient une liste de triplets pythagoriciens (nombres entiers a, b, c tels que $a^2 + b^2 = c^2$). Les historiens ont montré que ces triplets étaient générés par un algorithme, probablement lié à la résolution d'équations du second degré. La tablette montre aussi que les Babyloniens savaient calculer des racines carrées avec une grande précision : une tablette donne ainsi une valeur approchée de $\sqrt{2} = 1{,}4142129$, soit une erreur de moins de 0,00001 par rapport à la valeur réelle.

Résolution d'équations et algorithmes pratiques

Les tablettes scolaires de Nippur (vers 2000 av. J.-C.) révèlent que les scribes babyloniens maîtrisaient la résolution d'équations linéaires et quadratiques. Ils utilisaient des méthodes itératives pour extraire des racines carrées et cubiques, et des algorithmes pour résoudre des problèmes de partage proportionnel ou de calcul d'intérêts. Par exemple, pour résoudre une équation du type $x^2 = a$, ils appliquaient une procédure répétée de moyenne arithmétique, connue aujourd'hui sous le nom d'algorithme de Babylone.

Un problème typique pourrait être : Un champ a une aire de 60. La somme de ses côtés est 14. Quelles sont les longueurs des côtés ? La solution, gravée sur la tablette, suit une série d'étapes logiques : poser l'équation, calculer la moitié de la somme des côtés, puis appliquer une formule équivalente à notre résolution de l'équation du second degré.

Astronomie et prédiction

Les Babyloniens étaient aussi des astronomes hors pair. Ils utilisaient des algorithmes pour prédire les mouvements des planètes, notamment Jupiter, en calculant leur vitesse angulaire et leur position sur l'écliptique.

Une série de tablettes datées de 350 à 50 av. J.-C. décrit des procédures de calcul abstraites, où le temps est traité comme une variable mathématique. Ces méthodes, bien que moins géométriques que celles des Grecs, montrent une capacité à modéliser des phénomènes complexes, et ont influencé plus tard les astronomes grecs et indiens.

L'algorithme qui a découvert une planète avant les astronomes

En 2017, un algorithme entraîné sur les données du télescope Kepler a repéré un motif lumineux que les astronomes avaient manqué.

Il s'agissait d'une minuscule variation dans la luminosité d'une étoile, trop faible pour être remarquée à l'œil humain. L'algorithme a insisté : « il y a quelque chose ici ». Après vérification, les astronomes ont confirmé l'existence d'une exoplanète, Kepler-90i.

C'était la première fois qu'une planète était découverte non pas par un humain, mais par un algorithme. L'ironie est que les données étaient disponibles depuis des années : l'algorithme a simplement vu ce que personne n'avait remarqué.

Une planète entière, cachée dans un tableau de chiffres.

3.1.2 Égypte antique : Papyrus et pyramides

En Égypte, les mathématiques étaient principalement au service de l'État : construction des pyramides, gestion des greniers, calcul des impôts, et arpentage des terres après les crues du Nil.

Les sources principales sont le papyrus Rhind (vers 1650 av. J.-C.) et le papyrus de Moscou (vers 1850 av. J.-C.), qui contiennent des problèmes concrets et leurs solutions sous forme d'algorithmes.

Arithmétique et fractions

Les Égyptiens utilisaient un système décimal, mais sans notation positionnelle. Leur méthode de calcul des fractions était originale : toute fraction était exprimée comme une somme de fractions unitaires (de la forme 1/n). Par exemple, 2/5 était écrit comme 1/3 + 1/15.

Le papyrus Rhind contient une table de décomposition des fractions de la forme 2/n en fractions unitaires, ainsi que des algorithmes pour effectuer des multiplications et des divisions en utilisant uniquement des additions et des doublages.

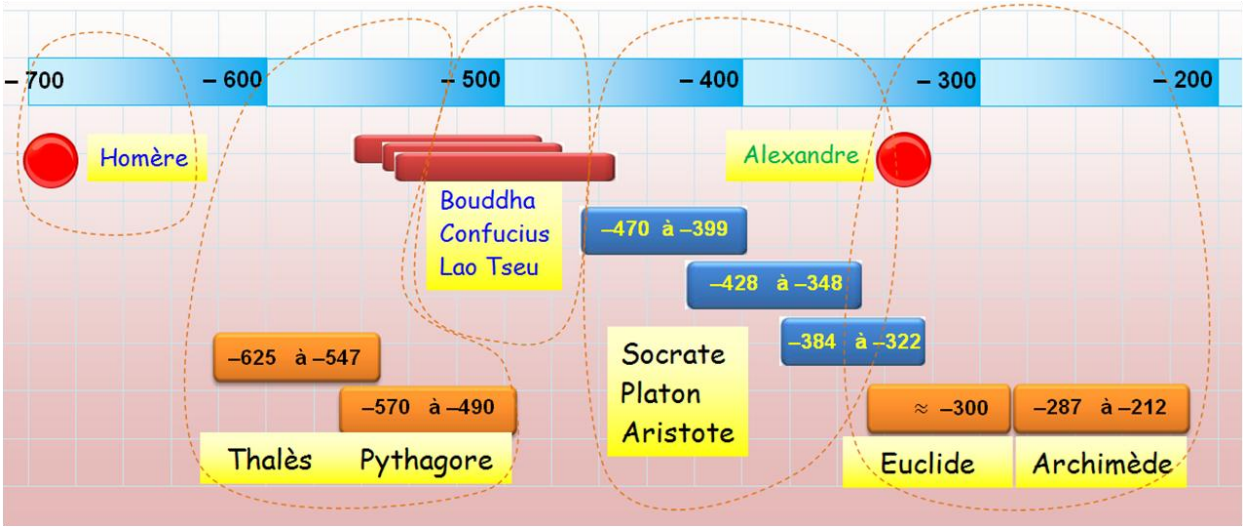
Géométrie et arpentage

La géométrie égyptienne était avant tout pratique. Pour calculer l'aire d'un champ, les arpenteurs utilisaient des formules empiriques. Par exemple, l'aire d'un cercle était approximée par $(8/9 \times \text{diamètre})^2$, ce qui donne une valeur de π proche de 3,16. Pour les pyramides, ils savaient calculer des volumes et des pentes, et utilisaient des cordes à nœuds pour tracer des angles droits.

Un problème du papyrus Rhind demande : « Si un champ circulaire a un diamètre de 9 khet, quelle est son aire ? » La solution donnée est : « Soustrais 1/9 du diamètre, soit 1 khet. Le reste est 8 khet. Multiplie 8 par 8, ce qui donne 64 setat. » Cette méthode, bien qu'approximative, montre une approche algorithmique de la géométrie.

3.1.3 Grèce antique : La naissance de la démonstration

Avec les Grecs, à partir du VI^{e} siècle av. J.-C., les mathématiques prennent un tournant théorique.
Les algorithmes ne sont plus seulement des recettes pratiques, mais deviennent des objets d'étude et de preuve.
Les figures majeures de cette période sont Thalès, Pythagore, Euclide, Ératosthène, et Archimède.



L'algorithme d'Euclide et la théorie des nombres

Euclide, vers 300 av. J.-C., formalise dans ses « Éléments » une méthode pour trouver le plus grand commun diviseur (PGCD) de deux nombres, connue aujourd'hui comme l'algorithme d'Euclide.
Cet algorithme, décrit dans le Livre VII, repose sur la division euclidienne : on divise le plus grand nombre par le plus petit, on remplace le plus grand par le plus petit, et le plus petit par le reste, jusqu'à obtenir un reste nul. Le dernier reste non nul est le PGCD.
Par exemple, pour trouver le PGCD de 48 et 18 :

- 48 ÷ 18 = 2 reste 12
- 18 ÷ 12 = 1 reste 6
- 12 ÷ 6 = 2 reste 0 Le PGCD est donc 6.

Cet algorithme, toujours enseigné aujourd'hui, est un exemple parfait de la puissance de la pensée grecque : une procédure simple, générale, et prouvée.

Le crible d'Ératosthène et les nombres premiers

Ératosthène, directeur de la bibliothèque d'Alexandrie vers 230 av. J.-C., invente une méthode pour lister tous les nombres premiers inférieurs à un nombre donné, appelée le « crible d'Ératosthène ».

L'algorithme est d'une élégance remarquable :

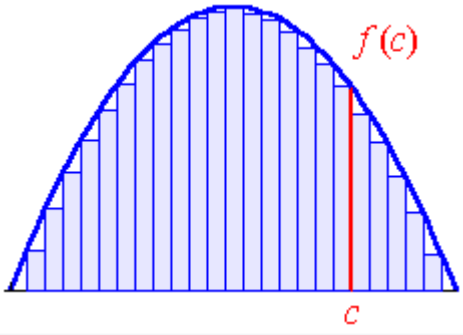
1. Écrire tous les entiers de 2 à N.
2. Barrer tous les multiples de 2 (sauf 2).
3. Passer au nombre suivant non barré, et barrer tous ses multiples.
4. Répéter jusqu'à √N. Les nombres restants sont premiers.

Ce crible, encore utilisé dans les cours d'informatique, illustre la capacité des Grecs à concevoir des procédures systématiques et optimisées.

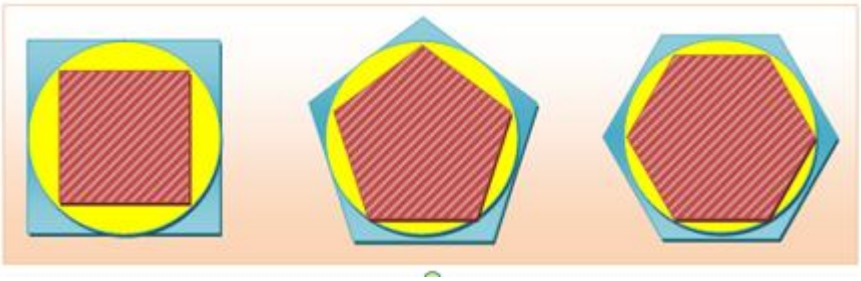
Archimède et les algorithmes infinitésimaux

Archimède (287–212 av. J.-C.) développe des méthodes pour calculer des aires et des volumes, préfigurant le calcul intégral.

Pour estimer l'aire sous une parabole, il utilise une procédure de découpage en tranches de plus en plus fines, puis fait tendre leur nombre vers l'infini. Cette approche, appelée « méthode d'exhaustion », est un ancêtre des algorithmes numériques modernes.



Calcul de l'aire de la parabole par empilement de fines tranches



Calcul de l'aire du disque par la limite des aires des polygones inscrits et circonscrits

3.1.4 Inde antique : Zéro, algèbre et astronomie

En Inde, à partir du V^e^ siècle av. J.-C., les mathématiciens développent des algorithmes pour l'astronomie et l'algèbre, et inventent le système décimal positionnel, incluant le zéro.

Les textes majeurs sont ceux d'Aryabhata (476–550) et de Brahmagupta (598–668).

Le « pulvérisateur » (kuṭṭaka) et les équations diophantiennes

Brahmagupta, dans son traité « Brāhmasphuṭasiddhānta » (628), décrit un algorithme appelé « kuṭṭaka » (pulvérisateur) pour résoudre des équations diophantiennes du premier degré (équations de la forme ax + by = c).

Cet algorithme, basé sur des divisions successives, est un ancêtre de l'algorithme d'Euclide étendu, et sera transmis aux Arabes, puis à l'Europe.

Le zéro et les opérations algébriques

Brahmagupta est le premier à traiter le zéro comme un nombre à part entière, et à définir des règles pour les opérations avec les nombres négatifs. Il donne aussi des procédures pour résoudre des équations du second degré, et des systèmes d'équations, posant les bases de l'algèbre moderne.

3.1.5 Chine antique : Les Neuf Chapitres et la méthode du pivot

En Chine, vers le II^{e} siècle av. J.-C., est compilé un ouvrage majeur : « Les Neuf Chapitres sur l'art mathématique ».

Ce texte, structuré en 246 problèmes, expose des algorithmes pour l'arpentage, l'agriculture, le commerce, et la résolution d'équations. Le chapitre 8, en particulier, décrit une méthode pour résoudre des systèmes d'équations linéaires, équivalente à la méthode du pivot de Gauss, et utilise des nombres négatifs.

Conclusion : L'héritage algorithmique de l'Antiquité

L'Antiquité a donc vu naître les premiers algorithmes, motivés par des besoins pratiques (commerce, construction, astronomie) ou théoriques (démonstration, abstraction).

Ces procédures, gravées sur l'argile, le papyrus ou le bambou, ont traversé les siècles et continuent d'influencer les mathématiques modernes. Que ce soit l'algorithme de Babylone pour les racines carrées, le crible d'Ératosthène, l'algorithme d'Euclide, ou la méthode du pivot chinoise, ces inventions montrent une capacité humaine universelle à structurer la pensée et à résoudre des problèmes par des étapes logiques et répétables.

3.2 Le Moyen Âge (500–1500)

Les méthodes de calcul circulent, se transforment et s'organisent entre trois mondes savants.

Charnière pour l'histoire des algorithmes et des mathématiques.

Alors que l'Europe occidentale traverse une phase de relative stagnation après la chute de Rome, le monde arabe, l'Inde et la Chine connaissent un essor scientifique remarquable.

Les algorithmes, hérités de l'Antiquité, sont préservés, enrichis, et transmis vers l'Occident, où ils vont profondément transformer les pratiques mathématiques et commerciales.

3.2.1 Le monde arabe : Passeurs de savoirs et inventeurs de l'algèbre

Al-Khwarizmi et la révolution algorithmique

v. 780 **v. 850** BIOGRAPHIE	**Al-Khwarizmi** est un savant persan de l'âge d'or abbasside, actif à Bagdad au sein de la Maison de la Sagesse. Mathématicien, astronome et géographe, il joue un rôle majeur dans la transmission et l'organisation des connaissances scientifiques.  Son ouvrage le plus célèbre, Le Livre de l'addition et de la soustraction selon le calcul indien, introduit en Occident les méthodes de calcul décimal venues d'Inde. Un autre traité, consacré à la résolution d'équations, donne naissance au mot **algèbre** (du terme arabe al-jabr). Son nom, latinisé en Algoritmi, est à l'origine du mot **algorithme**. Par ses travaux, Al-Khwarizmi a profondément influencé les mathématiques médiévales et posé les bases de nombreuses méthodes encore utilisées aujourd'hui.

Au IX^e^ siècle, à Bagdad, sous le califat abbasside, Muhammad ibn Musa al-Khwarizmi (780–850) devient une figure centrale de l'histoire des mathématiques.

Son nom, latinisé en « Algoritmi », donnera le mot « algorithme ». Ses deux traités majeurs, « Le calcul indien » (Kitab al-jam' wa-l-tafriq bi-hisab al-Hind) et « Le livre de la restauration et de la comparaison » (Kitab al-jabr wa-l-muqabala), marquent un tournant.

- **Le calcul indien** : Al-Khwarizmi y décrit le système de numération décimale positionnelle, incluant le zéro, qu'il a découvert dans des textes indiens. Ce système, bien plus efficace que les numérations romaine ou grecque, permet des calculs rapides et précis. Le traité explique comment écrire les nombres de 0 à 9, comment effectuer les quatre opérations, et comment appliquer ces méthodes à des problèmes concrets (commerce, héritage, arpentage). Ce livre, traduit en latin au XII^e^ siècle, introduira en Europe les « chiffres arabes » (en réalité indiens), révolutionnant les pratiques de calcul.
- **L'algèbre** : Dans son second traité, Al-Khwarizmi pose les bases de l'algèbre en classant les équations du premier et du second degré en six types canoniques. Il propose des algorithmes de résolution systématique, utilisant des méthodes de « restauration » (al-jabr) et de « comparaison » (al-muqabala). Par exemple, pour résoudre $x^2 + 10x = 39$, il explique comment transformer l'équation pour isoler x. Ces méthodes, traduites en latin, seront enseignées dans les universités européennes à partir du XII^e^ siècle, et le mot « algèbre » vient directement du titre de son livre.

La Maison de la Sagesse et la transmission des savoirs

Bagdad, au IX^e^ siècle, abrite la « Maison de la Sagesse » (Bayt al-Hikma), un centre de traduction et de recherche où sont compilés et étudiés les textes grecs, indiens et persans.

Les savants arabes y traduisent Euclide, Ptolémée, Brahmagupta, et développent des tables astronomiques d'une précision inédite. Ces tables, basées sur des algorithmes de calcul des positions planétaires, seront utilisées en Europe jusqu'à la Renaissance.

Diffusion vers l'Europe : Tolède et l'Andalousie

À partir du XII^{e} siècle, l'Espagne musulmane (Al-Andalus) devient un carrefour culturel.

Des érudits, comme Gérard de Crémone, traduisent en latin les œuvres d'Al-Khwarizmi et d'autres mathématiciens arabes. Les écoles de Tolède et de Séville forment des générations de savants européens à l'algèbre et aux nouveaux chiffres.

Fibonacci (Léonard de Pise), après un voyage en Afrique du Nord, publie en 1202 son « Liber Abaci », qui popularise en Europe les méthodes arabes et indiennes, notamment pour le commerce et la finance.

ANECDOTE

Fibonacci et les lapins qui envahirent l'Europe… sur le papier

La fameuse suite de Fibonacci n'a pas été inventée pour les mathématiques pures, mais pour résoudre un problème de reproduction de lapins.

Le scénario était totalement fictif, mais il a donné naissance à l'une des suites les plus célèbres de l'histoire : 1, 1, 2, 3, 5, 8, 13, 21, 34, 51 … l'un des nombres est la somme des deux précédents. Le rapport entre deux nombres consécutifs tend vers le nombre d'or : φ = 1, 618…

Une simple histoire de lapins a façonné des siècles de mathématiques et d'algorithmes.

3.2.2 L'Inde médiévale : Innovations algébriques et transmission vers l'Islam

Brahmagupta et l'algèbre indienne

Au VII^{e} siècle, Brahmagupta (598–668) écrit le « Brāhmasphuṭasiddhānta », un traité qui systématise l'usage du zéro, des nombres négatifs, et des équations diophantiennes.

Il y décrit l'algorithme du « pulvérisateur » (kuṭṭaka), une méthode pour résoudre des équations du type ax + by = c, qui sera reprise et développée par les mathématiciens arabes.

598
v. 668

BIOGRAPHIE

Brahmagupta est l'un des plus grands mathématiciens de l'Inde ancienne. Né au Rajasthan, il dirige très jeune l'observatoire astronomique d'Ujjain, l'un des centres scientifiques majeurs de son époque.

Son œuvre la plus célèbre, le *Brāhmasphuṭasiddhānta*, marque une étape décisive dans l'histoire des mathématiques : il y décrit des méthodes de calcul avancées, propose des règles pour manipuler les nombres négatifs et, surtout, donne la première définition claire du **zéro comme nombre à part entière**.

Il développe également des techniques pour résoudre des équations, travailler avec les fractions et calculer des aires et des volumes. Son influence s'étendra bien au-delà de l'Inde, jusqu'au monde arabe puis à l'Europe médiévale.

L'école du Kerala et les séries infinies

Entre le XIV^{e} et le XVI^{e} siècle, dans le sud de l'Inde, l'école du Kerala développe des algorithmes pour calculer des séries infinies (comme la série de Leibniz pour π), des approximations de fonctions trigonométriques, et des méthodes d'intégration.

Ces travaux, bien que peu connus en Europe à l'époque, préfigurent le calcul différentiel et intégral moderne.

Transmission vers le monde arabe et l'Europe

Dès le VIII^{e} siècle, les contacts entre l'Inde et le monde arabe (notamment via la conquête du Sind en 712) permettent la diffusion des chiffres indiens et des algorithmes de calcul.

Les traités d'astronomie et de mathématiques indiens sont traduits en arabe, puis en latin, et influencent profondément les savants européens.

3.2.3 La Chine médiévale : Algorithmes pratiques et avancées algébriques

Les Neuf Chapitres et la méthode du pivot

Le « Jiŭzhāng Suànshù » (Les Neuf Chapitres sur l'art mathématique), compilé entre le II^{e} siècle av. J.-C. et le I^{er} siècle ap. J.-C., reste la référence en Chine médiévale.

Ce texte, structuré en 246 problèmes, expose des algorithmes pour l'arpentage, le commerce, la résolution d'équations linéaires, et le calcul des aires et volumes. Le chapitre 8 décrit une méthode équivalente à l'élimination de Gauss pour résoudre des systèmes d'équations, et utilise des nombres négatifs.

Innovations sous les Song et les Yuan

Sous les Song (960–1279), les mathématiciens chinois perfectionnent les algorithmes pour le boulier, développent des méthodes d'extraction de racines carrées et cubiques, et calculent π avec une précision inégalée en Occident.

Zhu Shijie, au XIII^{e} siècle, invente la « méthode des quatre inconnues », un système algébrique avancé pour résoudre des équations polynomiales.

Transmission limitée vers l'Occident

Contrairement aux savoirs indiens et arabes, les mathématiques chinoises restent largement méconnues en Europe médiévale.

Les échanges se font principalement via les missionnaires jésuites à partir du XVI^{e} siècle, mais les algorithmes chinois (comme la méthode du pivot) ne seront redécouverts en Occident qu'à l'époque moderne.

3.2.4 L'Europe médiévale : Réception et adaptation

L'algèbre et les chiffres arabes en Europe

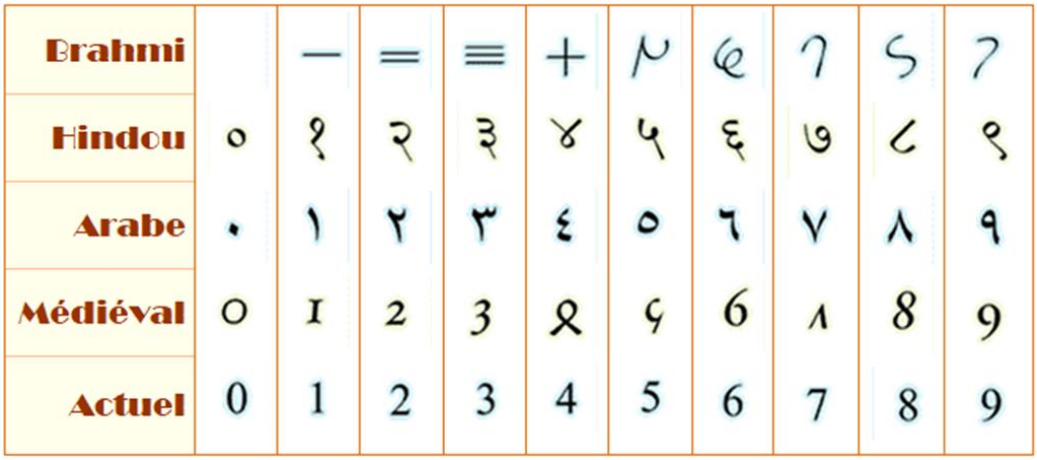
À partir du XII^{e} siècle, les traductions latines des textes arabes introduisent en Europe l'algèbre, les chiffres indo-arabes, et de nouveaux algorithmes.

Les universités de Paris, Oxford, et Bologne enseignent ces méthodes, d'abord pour l'astronomie et la théologie, puis pour le commerce.

Les marchands italiens, comme ceux de Venise ou de Florence, adoptent rapidement les nouveaux chiffres pour leurs comptes, malgré la résistance des habitudes romaines.

I	V	X	L	C	D	M
1	5	10	50	100	500	1000
Un	5 doigts	2 fois 5	Rotation du V	Cent	Moitié du cercle	Mille

Les chiffres romains



Évolution des chiffres : des chiffres indiens à nos chiffres.

Fibonacci et le Liber Abaci

Le « Liber Abaci » de Fibonacci (1202) est un manuel pratique qui montre comment utiliser les algorithmes arabes et indiens pour résoudre des problèmes de change, de profit, et de mesure.

L'ouvrage contient aussi la célèbre « suite de Fibonacci », un exemple d'algorithme récursif, et des méthodes pour convertir les monnaies ou calculer les intérêts.

v. 1170
v. 1240

BIOGRAPHIE

Fibonacci : Le Père de l'Algorithme Occidental

Si l'on considère un algorithme comme une suite finie d'opérations permettant de résoudre un problème, Léonard de Pise (v. 1170 - 1250) en est l'un des plus grands architectes. Bien avant l'informatique, il a compris que l'efficacité d'un calcul dépend de la structure des données utilisées.

L'optimisation du "système d'exploitation" mathématique

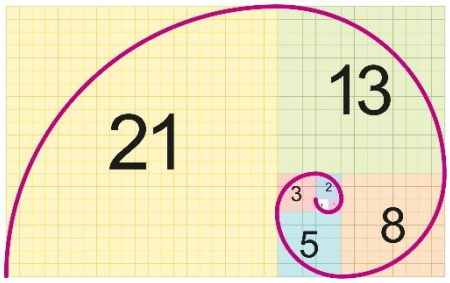
Lors de ses voyages à Béjaïa (Algérie actuelle), Fibonacci réalise que les algorithmes de calcul romains (peu pratiques pour les divisions ou les grands nombres) sont obsolètes. En publiant le Liber Abaci en 1202, il importe en Europe le système indo-arabe.

Il introduit ainsi l'usage du zéro, agissant comme un "marqueur de position". C'est l'ancêtre du système binaire : sans une notation positionnelle efficace, les algorithmes complexes de multiplication et de division que nous utilisons aujourd'hui n'auraient jamais pu être automatisés.

La suite de Fibonacci : un algorithme récursif

Sa célèbre suite est, par essence, l'un des premiers algorithmes **récursifs** documentés. Pour obtenir un terme, la règle est simple et répétitive :

$$F_n = F_{n-1} + F_{n-2}$$



Spirale construite à partir des nombres de la suite de Fibonacci

En informatique, cette suite est le cas d'école pour enseigner la **récursivité** et l'optimisation de la complexité (via la programmation dynamique). Elle démontre qu'une règle logique simple peut générer une structure d'une complexité infinie, capable de modéliser la croissance organique dans la nature.

Un héritage logique

Fibonacci n'a pas seulement compté des lapins ; il a fourni à l'Occident les outils logiques (les algorithmes de calcul commercial) qui ont permis l'essor de la banque et des sciences. Il a transformé les mathématiques d'un concept abstrait en une méthode de résolution de problèmes.

Les algorithmes dans la vie quotidienne

Les algorithmes deviennent indispensables pour :

- Le commerce : calcul des profits, des changes, des taxes.
- L'architecture : construction des cathédrales, calcul des arcs et des voûtes.
- L'astronomie : prédiction des éclipses, établissement des calendriers.

Les « livres d'abaque » (manuels de calcul) se multiplient, et des écoles urbaines forment les marchands et les artisans à ces nouvelles techniques.

ANECDOTE

L'algorithme qui a sauvé Notre-Dame

Après l'incendie de Notre-Dame de Paris en 2019, les architectes ont dû reconstruire des parties dont les plans avaient disparu.

Heureusement, un historien de l'art avait réalisé quelques années plus tôt un scan 3D complet de la cathédrale pour un projet de jeu vidéo. Les algorithmes d'imagerie ont permis de reconstituer la structure avec une précision millimétrique.

Sans eux, certaines parties auraient été impossibles à restaurer fidèlement.

Un algorithme conçu pour un jeu vidéo a fini par sauver un monument millénaire.

3.2.5 Synthèse : L'héritage du Moyen Âge

Le Moyen Âge a vu :

- **La préservation et l'enrichissement** des algorithmes antiques par les Arabes, les Indiens et les Chinois.
- **La transmission** de ces savoirs vers l'Europe, notamment via l'Espagne musulmane et les traductions latines.

- **L'adoption progressive** des chiffres indo-arabes et de l'algèbre, qui vont préparer la Renaissance scientifique.

Sans cette période de transmission et d'innovation, les mathématiques modernes n'auraient pas pu émerger. Les algorithmes, nés dans l'Antiquité, ont été perfectionnés, systématisés, et diffusés à travers le monde, posant les bases de la science contemporaine.

3.3 Renaissance et Révolution scientifique (1500–1800) :

Les algorithmes deviennent des outils explicites au service d'une science en plein essor.

Cette période marque un tournant décisif dans l'histoire des algorithmes et des mathématiques.

Après la transmission des savoirs antiques et médiévaux par les Arabes, les Indiens et les Chinois, l'Europe devient le théâtre d'une explosion créatrice, portée par des figures comme Descartes, Newton, Leibniz et Euler.

3.3.1 La Renaissance (1500–1650) : Réinvention et révolution symbolique

La redécouverte des textes antiques et arabes

Dès le XV^{e} siècle, la chute de Constantinople (1453) et l'essor des échanges avec le monde arabe et indien permettent aux érudits européens de redécouvrir les œuvres d'Euclide, d'Archimède, d'Al-Khwarizmi, et de Brahmagupta. Les traductions latines des traités arabes, notamment ceux sur l'algèbre et les chiffres indo-arabes, circulent en Italie, en Espagne, et en Allemagne. Les marchands, les architectes et les astronomes adoptent rapidement ces nouvelles méthodes, plus efficaces que les calculs romains ou les abaques.

L'algèbre symbolique : Viète et Descartes

Jusqu'au XVI^{e} siècle, les mathématiques s'écrivent en mots : les équations sont décrites en langage naturel, sans symboles. François Viète (1540–1603) change la donne en introduisant l'usage systématique de lettres pour représenter les inconnues et les paramètres. Par exemple, il écrit $A^2 + 3A = 2$ pour une équation du second degré, là où ses prédécesseurs écrivaient « un carré plus trois fois sa racine égale deux ». Cette innovation, appelée « algèbre spécieuse », permet de généraliser les algorithmes de résolution et de manipuler des expressions abstraites.

René Descartes (1596–1650) pousse plus loin cette révolution symbolique. Dans sa « Géométrie » (1637), il unifie algèbre et géométrie en associant chaque point du plan à un couple de nombres (x, y), posant les bases de la géométrie analytique. Il standardise aussi les notations : x, y, z pour les inconnues, a, b, c pour les coefficients, et les exposants pour les puissances (x^2 au lieu de xx). Ces conventions, toujours en usage, transforment les algorithmes en outils universels et manipulables.

Les algorithmes dans la vie quotidienne

La Renaissance voit les algorithmes envahir de nouveaux domaines :

- **L'architecture** : Les traités de perspective (Alberti, Léonard de Vinci) utilisent des méthodes géométriques pour représenter l'espace en deux dimensions, basées sur des calculs de proportions et de points de fuite.
- **Le commerce** : Les banquiers et marchands italiens (comme ceux de Venise ou Florence) adoptent les chiffres indo-arabes et les algorithmes de calcul des intérêts, des changes, et des profits, popularisés par des manuels comme le « Liber Abaci » de Fibonacci.
- **L'astronomie** : Copernic, Tycho Brahe et Kepler utilisent des algorithmes de calcul pour prédire les mouvements planétaires, jetant les bases de la mécanique céleste.

3.3.2 Vers la mécanisation du calcul

À partir du XVII[e] siècle, les besoins en calcul explosent : navigation, astronomie, commerce, ingénierie. John Napier invente les logarithmes (1614), qui transforment les multiplications en additions. Ses « bâtons de Napier », sortes de règles gravées, permettent d'effectuer des multiplications en manipulant des objets physiques : un algorithme matérialisé.

Blaise Pascal, à 19 ans, construit la Pascaline (1642), une machine à calculer mécanique destinée à aider son père, percepteur d'impôts. Elle est ingénieuse, mais fragile et coûteuse. Leibniz, quelques décennies plus tard, perfectionne l'idée avec sa machine à calculer binaire et rêve d'un « calculus ratiocinator », un système logique universel permettant de résoudre les disputes par le calcul. L'idée est visionnaire : elle annonce la logique formelle et, indirectement, l'informatique.

3.3.3 La Révolution scientifique (1650–1800) : L'ère de l'analyse et du calcul infinitésimal

Newton et Leibniz : L'invention du calcul infinitésimal

Le XVII[e] siècle est marqué par la création, indépendante et presque simultanée, du calcul infinitésimal par Isaac Newton (1642–1727) et Gottfried Wilhelm Leibniz (1646–1716).

Leurs travaux transforment radicalement les mathématiques et les sciences physiques.

- **Newton** développe sa « méthode des fluxions » pour étudier les variations continues (dérivées) et les aires sous les courbes (intégrales). Il l'applique à la mécanique céleste, formulant les lois du mouvement et de la gravitation universelle. Par exemple, il montre que la force gravitationnelle est inversement proportionnelle au carré de la distance, en utilisant des algorithmes de calcul différentiel.
- **Leibniz**, de son côté, invente un symbolisme puissant : il note la dérivée comme dy/dx et l'intégrale avec le symbole ∫. Ses notations, plus claires et plus générales que celles de Newton, s'imposent en Europe continentale. Il conçoit aussi les algorithmes de calcul différentiel comme des procédures systématiques, applicables à une large classe de fonctions.

Le calcul infinitésimal permet de modéliser des phénomènes naturels (mouvement, croissance, chaleur) et de résoudre des problèmes jusqu'alors inaccessibles, comme le calcul des trajectoires des comètes ou la forme des courbes optimales.

1646
1716

BIOGRAPHIE

Gottfried Wilhelm Leibniz, philosophe, mathématicien et polymathe allemand, est une figure majeure dans l'histoire des algorithmes et de la logique formelle. Dès son jeune âge, il se distingue par son génie universel, contribuant à des domaines aussi variés que les mathématiques, la métaphysique, le droit et la diplomatie.

Leibniz est surtout célèbre pour avoir inventé le calcul différentiel et intégral (indépendamment de Newton), posant ainsi les bases de l'analyse mathématique moderne. Mais son apport aux algorithmes est tout aussi fondamental : il conçoit une machine à calculer capable d'effectuer des multiplications et des divisions, améliorant les travaux

de Pascal. Cette machine, appelée Stepped Reckoner, utilise un système de roues dentées et préfigure les futurs ordinateurs mécaniques.

Sur le plan théorique, Leibniz rêve d'une langue universelle (characteristica universalis) et d'un calcul logique (calculus ratiocinator), visant à réduire le raisonnement humain à des opérations symboliques. Ces idées, bien que non réalisées de son vivant, inspirent plus tard les travaux de Boole, Frege et les pionniers de l'informatique théorique, comme Turing. En imaginant un système où les conflits pourraient être résolus par un simple "Calculemus" ("Calculons"), il pose les jalons de l'automatisation du raisonnement — un pilier des algorithmes modernes.

Enfin, son principe de raison suffisante et sa philosophie de l'harmonie préétablie reflètent une vision systémique du monde, où tout peut être analysé et optimisé — une intuition qui résonne avec l'esprit même de l'algorithmique. Leibniz meurt en 1716, laissant derrière lui un héritage intellectuel qui continue d'éclairer les sciences et l'informatique.

Découverte

Dérivée et intégrale : deux gestes complémentaires

On présente souvent la dérivée et l'intégrale comme deux monstres sacrés des mathématiques, intimidants, presque ésotériques. Pourtant, si l'on gratte un peu le vernis, on découvre qu'elles ne sont que deux manières opposées — et parfaitement complémentaires — de regarder un même phénomène. Comme deux outils d'un même artisan, l'un pour mesurer le mouvement, l'autre pour mesurer l'étendue.

Imagine une voiture qui roule sur une route droite. La dérivée, c'est le compteur de vitesse : elle te dit *comment* la situation évolue à chaque instant. L'intégrale, elle, serait plutôt le journal de bord : elle additionne tous les petits déplacements pour te dire *combien* de chemin tu as parcouru au total. L'une regarde l'instant, l'autre regarde l'accumulation.

Les mathématiciens ont mis des siècles à formaliser ces deux gestes simples. Ils ont inventé des algorithmes élégants pour les calculer sans effort, même pour des fonctions compliquées. Ainsi, la dérivée de (x^2) devient une fonction en (x) — comme si on passait de la trajectoire à la vitesse. Et l'intégrale d'une fonction en (x) redonne une fonction en (x^2) — comme si on remontait du mouvement à la distance parcourue. Deux opérations inverses, comme inspirer et expirer.

On peut aussi voir la dérivée et l'intégrale comme deux façons de lire un paysage. La dérivée, c'est l'œil du randonneur qui scrute la pente : « Est-ce que ça monte, est-ce que ça descend, et à quel rythme ? » L'intégrale, c'est l'œil du cartographe qui mesure la surface d'une vallée ou l'étendue d'un champ. L'un s'intéresse au relief local, l'autre à l'espace global.

En réalité, ces deux notions ne sont pas des abstractions lointaines : elles décrivent des gestes que nous faisons sans cesse. Quand tu regardes la vitesse d'un film qui accélère, tu fais une dérivée. Quand tu additionnes les pages lues dans un roman, tu fais une intégrale. Les mathématiques n'ont fait que mettre des mots — et des outils — sur ces intuitions.

Dérivée et intégrale ne sont donc pas deux monstres, mais deux faces d'une même pièce. Deux manières de comprendre le monde : l'une en observant le changement, l'autre en mesurant l'accumulation. Et ensemble, elles forment le cœur du calcul infinitésimal,

	cette grande mécanique qui permet de décrire le mouvement, la croissance, la lumière, les sons, et même les algorithmes modernes.

Euler : Le maître des algorithmes et des notations

Leonhard Euler (1707–1783) domine le XVIII^e siècle par sa productivité et son génie. Il systématise et étend les travaux de ses prédécesseurs, introduisant des notations et des algorithmes qui deviennent des standards :

- **Notations** : Il popularise l'usage de f(x) pour les fonctions, de e pour la base du logarithme naturel, de i pour la racine carrée de -1, et de Σ pour les sommes. Ces symboles, encore utilisés aujourd'hui, simplifient l'écriture et la communication des algorithmes.
- **Algorithmes** : Euler développe des méthodes pour résoudre les équations différentielles, calculer les intégrales, et manipuler les séries infinies. Par exemple, il trouve des algorithmes pour sommer des séries, résoudre des équations aux dérivées partielles, et calculer des valeurs approchées de π et d'autres constantes.
- **Applications** : Il applique ces outils à la mécanique des fluides, à l'astronomie, à la cartographie, et même à la théorie des graphes (avec le célèbre problème des ponts de Königsberg).

Lagrange et la mécanique analytique

Joseph-Louis Lagrange (1736–1813) achève la formalisation de la mécanique en termes purement mathématiques.

Dans sa Mécanique analytique (1788), il montre que les lois du mouvement peuvent être dérivées d'un seul principe — le principe de moindre action — en utilisant des méthodes qui consistent à comparer toutes les trajectoires possibles pour identifier celle qui minimise l'action : algorithmes qui examinent toutes les trajectoires imaginables comme un voyageur qui cherche le chemin le plus économe.

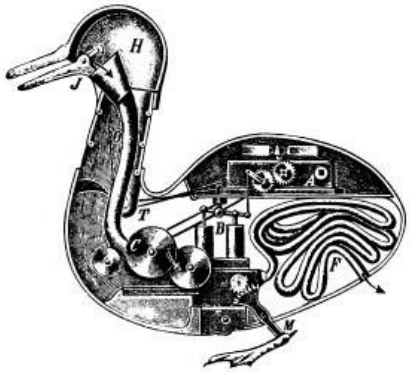
Ses équations, toujours enseignées, permettent de modéliser le mouvement de systèmes complexes, des planètes aux machines

3.3.4 Les tables, les automates et l'incertitude

Au XVIII^e siècle, les États européens ont besoin de tables astronomiques, de tables de navigation, de tables d'impôts. Ces tables sont calculées à la main, par des « calculateurs » humains. Les erreurs sont fréquentes, parfois lourdes de conséquences.

	**Le premier "ordinateur humain" payé… en bière** Au XVII^e siècle, les astronomes employaient des « calculateurs humains » pour effectuer des tables numériques. Certains étaient rémunérés en bière, car cela coûtait moins cher que de les payer en argent. On peut dire que les premiers algorithmes astronomiques ont été alimentés… au malt.

C'est dans ce contexte que Thomas Bayes propose une méthode pour mettre à jour des probabilités en fonction de nouvelles informations. Son théorème, publié après sa mort en 1763, introduit une idée fondamentale : raisonner avec l'incertitude. À l'époque, ce n'est qu'une curiosité mathématique ; deux siècles plus tard, ce sera l'un des piliers de l'intelligence artificielle.



En parallèle, les automates mécaniques fascinent l'Europe. Le canard mécanique de Vaucanson (1739), capable de battre des ailes et d'avaler des grains, n'est pas un algorithme au sens strict, mais il montre que des comportements complexes peuvent être produits par des mécanismes déterministes. L'idée d'une machine exécutant une séquence d'instructions gagne du terrain.

3.3.5 L'héritage des mathématiques arabes et indiennes

Pendant toute cette période, l'influence des mathématiques arabes et indiennes reste déterminante :

- **Les chiffres indo-arabes** et le système décimal, introduits en Europe via les traductions latines des traités d'Al-Khwarizmi et de Brahmagupta, deviennent la norme. Ils permettent des calculs rapides et précis, indispensables au commerce et à la science.
- **L'algèbre** et les méthodes de résolution des équations, développées par les Arabes à partir des travaux indiens, sont perfectionnées par les Européens. Les algorithmes pour les équations du second degré, puis du troisième et quatrième degré, deviennent des outils standard.
- **Les notations symboliques**, esquissées par les mathématiciens arabes (comme al-Qalasadi au XV^{e} siècle), sont reprises et systématisées par Viète, Descartes et Euler, aboutissant à l'algèbre moderne.

1545 ANECDOTE	**Le scandale du troisième degré : mathématiciens en duel !** On l'oublie aujourd'hui, mais au XVI^{e} siècle, la résolution des équations du troisième degré n'était pas un exercice de classe… c'était une affaire d'honneur, de secrets jalousement gardés, et parfois même de vengeance. Une véritable saga mathématique, digne des chroniques à scandale. Tout commence dans les rues de Bologne, où les mathématiciens ne se contentent pas de griffonner des équations : ils s'affrontent en public, comme des gladiateurs de l'esprit. On lance des défis, on parie sa réputation, on humilie l'adversaire à coups de problèmes insolubles. Et au cœur de cette arène intellectuelle, un trésor convoité : la méthode pour résoudre les équations cubiques. Scipione del Ferro, professeur discret mais redoutable, découvre la clé du mystère. Plutôt que de publier, il garde sa trouvaille comme un magicien protège son grimoire. Sur son lit de mort, il transmet le secret à son élève, Antonio Fior, qui s'empresse de défier un jeune mathématicien fougueux : Niccolò Tartaglia, surnommé « le Bègue », mais dont l'esprit est d'une clarté redoutable. Tartaglia relève le défi… et l'emporte brillamment. Mais l'histoire ne s'arrête pas là. Entre alors en scène Gerolamo Cardano, médecin, astrologue, joueur invétéré, et génie mathématique à ses heures. Fasciné par la méthode de Tartaglia, il le supplie de lui en révéler les détails. Tartaglia accepte, mais sous serment : Cardano n'a pas le droit de publier. Serment que Cardano… finit par briser. Lorsqu'il découvre que del Ferro avait trouvé la méthode avant Tartaglia, il estime que le secret n'en est plus un. Il publie alors, triomphalement, la solution du troisième

	degré dans son *Ars Magna* (1545), un livre qui fera date. Tartaglia hurle à la trahison, l'accuse de vol intellectuel, et les deux hommes s'affrontent dans une polémique enflammée qui enflamme l'Italie savante. Le public se régale. Les pamphlets circulent. Les insultes volent. On parle de « perfidie », de « parjure », de « tricherie ». Les équations deviennent une affaire de morale, presque de crime. Et pendant que les mathématiciens se déchirent, un autre génie, Lodovico Ferrari, élève de Cardano, observe la scène… et en profite pour résoudre tranquillement les équations du quatrième degré, comme si de rien n'était. Ainsi, derrière ce que les manuels présentent aujourd'hui comme une simple étape de l'histoire des mathématiques, se cache une véritable affaire à sensation, faite de secrets, de rivalités, de coups bas et de triomphes intellectuels. Une preuve que même les équations peuvent déclencher des tempêtes humaines.

3.3.6 Synthèse : Une révolution algorithmique

La période 1500–1800 voit les algorithmes passer du statut de recettes pratiques à celui de fondements de la science moderne. Plusieurs facteurs expliquent cette transformation :

- **La révolution symbolique** : L'invention de notations algébriques et infinitésimales permet de manipuler des concepts abstraits et de généraliser les procédures de calcul.
- **L'interdisciplinarité** : Les algorithmes deviennent centraux en astronomie, physique, ingénierie, et économie, créant un langage commun entre les disciplines.
- **La transmission des savoirs** : L'imprimerie, les académies scientifiques, et les réseaux d'érudits accélèrent la diffusion des innovations.

À la fin du XVIII^{e} siècle, les mathématiques sont devenues une science algorithmique, où les procédures de calcul, les notations symboliques, et les méthodes analytiques permettent de résoudre des problèmes d'une complexité inédite. Cet héritage posera les bases de l'informatique et des sciences modernes.

3.4 Le XIX^{e} siècle, un tournant décisif

La logique, l'abstraction et la mécanisation redéfinissent la notion même d'algorithme.

Cette époque est celle de la formalisation des concepts, de la mécanisation des calculs, et de l'émergence de l'algèbre abstraite et de l'analyse rigoureuse.

Ces avancées, portées par des figures comme Boole, Cantor, Hilbert, Babbage, Cauchy, Weierstrass, Riemann, et Galois, posent les bases des mathématiques modernes et de l'informatique.

3.4.1 Formalisation et fondements : Logique, ensembles, et axiomatisation

L'algèbre de Boole et la naissance de la logique mathématique

Au milieu du XIX^{e} siècle, George Boole (1815–1864) révolutionne la logique en la transformant en une branche des mathématiques. Dans son ouvrage « The Laws of Thought » (1854), il introduit l'algèbre booléenne, où les valeurs de vérité (vrai/faux) sont manipulées comme des variables algébriques (1/0). Ses opérations (ET, OU, NON) deviennent des outils pour analyser les circuits logiques, préfigurant l'informatique moderne. Boole montre que la logique peut être formalisée comme un calcul, ouvrant la voie à la logique symbolique et, plus tard, aux circuits électroniques.

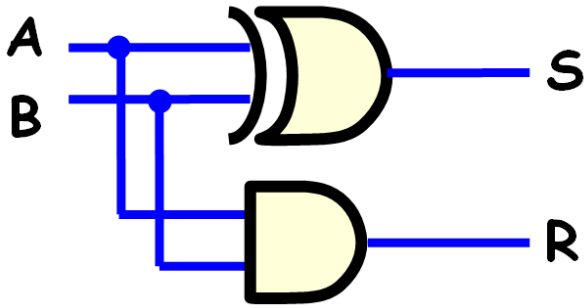
Découverte

Addition de deux bits

Voyons pas à pas comment additionner deux bits, sachant qu'en binaire, on compte : 0, 1, 10, 11, …

Opération	A	B	Somme S	Retenue R
0 + 0 = 0	0	0	0	0
0 + 1 = 1	0	1	1	0
1 + 0 = 1	1	0	1	0
1 + 1 = 10	1	1	0	1

Comment traduire cette table de vérité en circuits logiques ?



Calcul de S : la sortie S vaut « 1 » si et seulement si l'une des entées A **OU** B est égale à « 1 ». C'est ce que traduit ce symbole du haut, une porte logique appelée « ou exclusif ».

Calcul de R : la sortie R vaut « 1 » si les deux entrées A **ET** B sont égale à « 1 » en même temps. C'est ce que traduit ce symbole du bas, une porte logique appelée « et ».

Une addition sur beaucoup plus de bits consiste simplement à « cascader » ce type de circuit élémentaire.

Cantor et la théorie des ensembles

Georg Cantor (1845–1918) développe, à partir des années 1870, une théorie des ensembles qui permet de manipuler l'infini de manière rigoureuse. Il introduit la notion de cardinalité (taille des ensembles) et montre qu'il existe plusieurs types d'infinis (par exemple, celui des entiers naturels est « plus petit » que celui des réels). Ces travaux, bien que controversés à l'époque, deviennent centraux pour les fondements des mathématiques et la théorie de la calculabilité.

Découverte

Plusieurs infinis ?

Quelle est la distance la plus courte pour aller de A à C ?

- le chemin AB + BC = 5 + 10 = 15;
- le chemin en escalier = 15;
- le chemin comprenant un grand nombre d'escaliers = 15;
- le chemin comprenant une infinité d'escaliers = 15; ou
- la diagonale AC = $\sqrt{5^2 + 10^2} = 11,180$ ….

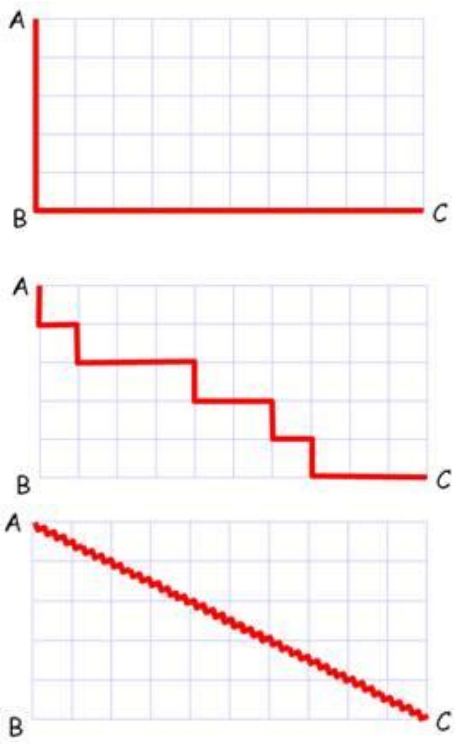
Oui ! Paradoxe qui semble montrer que la diagonale n'est pas égale à la somme des escaliers.

Le seul moyen de lever le paradoxe est d'admettre différents niveaux d'infinis.

Il y plus de points sur la diagonale (en fait continue) que dans la quantité d'escaliers (en fait discrète) qui s'applique sur la diagonale. **...**

Soit la conclusion suivante:

L'infini des **nombres entiers** est "moins riche " que l'infini de la **quantité de points** sur une droite.



Hilbert et l'axiomatisation

David Hilbert (1862–1943), à la charnière des XIX^{e} et XX^{e} siècles, propose une approche axiomatique pour toutes les branches des mathématiques. Son programme vise à démontrer la cohérence, la complétude et la décidabilité des théories mathématiques.

En 1900, il présente une liste de 23 problèmes non résolus, dont plusieurs concernent directement les algorithmes et la logique. Ses travaux inspirent la formalisation moderne des mathématiques et la recherche en fondements. Aujourd'hui, une dizaine sont considérés comme résolus, plusieurs autres partiellement éclaircis, et deux restent encore ouverts, témoignant d'un programme dont l'ambition dépasse toujours notre horizon mathématique.

Découverte

Deux énigmes qui défient toujours les mathématiciens

Plus d'un siècle après que David Hilbert a dressé sa liste mythique, deux problèmes continuent de résister comme des forteresses imprenables. Ils ont l'air abstraits, presque lointains, mais on peut se les représenter avec des images simples.

1. L'hypothèse de Riemann : la musique secrète des nombres premiers

Imagine une immense partition musicale où chaque note serait un nombre premier. On sait qu'ils semblent dispersés au hasard, comme des étoiles dans le ciel, mais Riemann a découvert qu'ils suivent en réalité une sorte de rythme caché, une pulsation profonde inscrite dans une mystérieuse fonction complexe.

$$\zeta(z) = \sum_{n=1}^{\infty} \frac{1}{n^z}$$

L'hypothèse de Riemann affirme que toutes les « notes » importantes de cette fonction — ses zéros — s'alignent parfaitement sur une même ligne, comme si l'univers des nombres obéissait à une harmonie secrète.

Depuis 1859, personne n'a réussi à prouver que cette musique mathématique est parfaitement accordée.

2. Les équations de Navier–Stokes : la colère des fluides

L'autre problème ressemble à une enquête sur le comportement de l'eau, de l'air, des nuages, du sang, des océans… bref, de tout ce qui coule ou tourbillonne.

$$\rho(u_t + (u.\nabla)u) - \nu\Delta u + \nabla p = f$$

$$\text{div } u = 0$$

Les équations de Navier–Stokes décrivent ces mouvements, mais elles sont si complexes qu'on ne sait pas si elles donnent toujours des solutions raisonnables.

En termes simples : peut-on garantir que l'eau ne « s'emballe » jamais mathématiquement, qu'elle ne crée pas soudain une explosion infinie de vitesse ou d'énergie ?

C'est un peu comme essayer de prédire à coup sûr la forme d'une vague qui se brise : on sait la décrire, mais on ignore si les équations restent sages dans toutes les situations imaginables.

3.4.2 Mécanisation des calculs : Des machines à calculer aux ancêtres de l'ordinateur

Charles Babbage et la machine analytique

Charles Babbage (1791–1871), insatisfait des erreurs dans les tables astronomiques et mathématiques, conçoit dès 1820 des machines capables d'automatiser les calculs :

- La machine à différences (1822) : conçue pour calculer et imprimer des tables de fonctions polynomiales, elle utilise des engrenages pour effectuer des additions répétées.
- La machine analytique (1834) : bien plus ambitieuse, elle est programmable grâce à des cartes perforées (inspirées du métier Jacquard), dispose d'une mémoire, d'une unité de calcul, et peut traiter des boucles conditionnelles. Bien que jamais construite de son vivant, ses plans détaillés en font l'ancêtre des ordinateurs modernes. Ada Lovelace, écrit le premier algorithme publié pour cette machine, devenant ainsi la première programmeuse de l'histoire.

Les machines de Babbage visent à produire des tables nautiques et astronomiques exemptes d'erreurs, essentielles pour la navigation et la science. Leur conception influence profondément la mécanisation des calculs et la pensée algorithmique, bien que la technologie de l'époque ne permette pas leur réalisation complète.

Ada Lovelace : la première à penser l'algorithme comme programme

La véritable rupture conceptuelle survient avec Ada Lovelace (1815–1852), collaboratrice de Babbage, dans les années 1840.

En étudiant la machine analytique de Charles Babbage — une machine mécanique programmable, jamais achevée mais conceptuellement révolutionnaire — elle comprend quelque chose que même Babbage n'avait pas pleinement formulé :

> **Une machine peut exécuter des instructions générales, indépendantes des nombres particuliers.**

Dans ses notes, elle décrit ce que l'on considère aujourd'hui comme le premier programme informatique : une procédure pour calculer les nombres de Bernoulli.

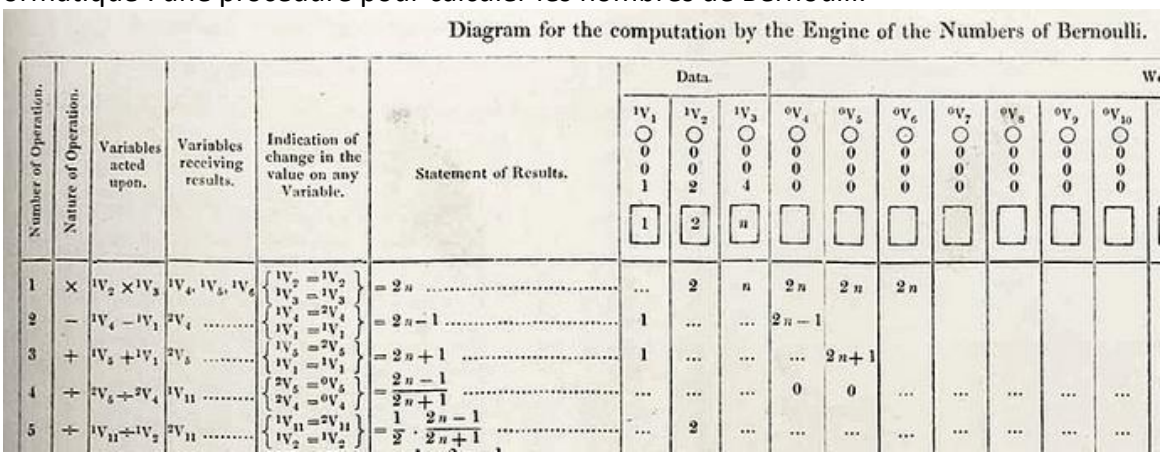


Diagram for the computation by the Engine of the Numbers of Bernoulli.

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.			W							
						$^1V_1$ 0 0 0 1 [1]	$^1V_2$ 0 0 0 2 [2]	$^1V_3$ 0 0 0 4 [n]	$^0V_4$ 0 0 0 0 [ ]	$^0V_5$ 0 0 0 0 [ ]	$^0V_6$ 0 0 0 0 [ ]	$^0V_7$ 0 0 0 0 [ ]	$^0V_8$ 0 0 0 0 [ ]	$^0V_9$ 0 0 0 0 [ ]	$^0V_{10}$ 0 0 0 0 [ ]	
1	×	$^1V_2 \times {}^1V_3$	$^1V_4, {}^1V_5, {}^1V_6$	$\left\{\begin{matrix} {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \end{matrix}\right\}$	$= 2n$	...	2	n	$2n$	$2n$	$2n$					
2	−	$^1V_4 - {}^1V_1$	$^2V_4$	$\left\{\begin{matrix} {}^1V_4 = {}^2V_4 \\ {}^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= 2n-1$	1	...	...	$2n-1$							
3	+	$^1V_5 + {}^1V_1$	$^2V_5$	$\left\{\begin{matrix} {}^1V_5 = {}^2V_5 \\ {}^1V_1 = {}^1V_1 \end{matrix}\right\}$	$= 2n+1$	1	...	...	...	$2n+1$						
4	÷	$^2V_5 \div {}^2V_4$	$^1V_{11}$	$\left\{\begin{matrix} {}^2V_5 = {}^0V_5 \\ {}^2V_4 = {}^0V_4 \end{matrix}\right\}$	$= \frac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	
5	÷	$^1V_{11} \div {}^1V_2$	$^2V_{11}$	$\left\{\begin{matrix} {}^1V_{11} = {}^2V_{11} \\ {}^1V_2 = {}^1V_2 \end{matrix}\right\}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	

Cette note est généralement considérée comme le premier algorithme spécifiquement destiné à un ordinateur, ce qui fait de Lovelace la première programmeuse informatique de l'histoire.

Mais surtout, elle introduit une idée fondamentale :

> **Un algorithme n'est pas seulement une méthode de calcul, c'est une séquence d'opérations abstraites, que l'on peut confier à une machine.**

Ada Lovelace est la première à comprendre que les algorithmes peuvent manipuler autre chose que des nombres : des symboles, des relations, des structures. Elle ouvre la voie à l'informatique moderne.

ANECDOTE

Ada Lovelace, première programmeuse… d'une machine qui n'existait pas

Ada Lovelace a écrit le premier programme informatique connu pour la machine analytique de Charles Babbage.

Problème : la machine n'a jamais été construite.

Elle a donc programmé un ordinateur imaginaire, un siècle avant l'invention des ordinateurs.

Une visionnaire qui écrivait des algorithmes pour un futur encore impossible.

Charles Babbage et les pièces de monnaie lancées sur son bureau

Babbage, obsédé par la précision, détestait le bruit.

Lorsqu'il travaillait à sa machine à différences, des enfants du quartier venaient exprès faire du vacarme sous ses fenêtres. Il écrivit même une lettre au journal local pour dénoncer « l'intolérable perturbation acoustique ».

Le père de l'ordinateur moderne était un génie… mais pas un fan du bruit.

3.4.3 Analyse mathématique : Rigueur et nouvelles théories

Cauchy, Weierstrass et la rigueur en analyse

Au début du XIX^{e} siècle, Augustin-Louis Cauchy (1789–1857) et Karl Weierstrass (1815–1897) transforment l'analyse mathématique en introduisant des définitions rigoureuses de la limite, de la continuité, de la convergence, et de la dérivabilité.

Cauchy, dans son « Cours d'analyse » (1821), pose les bases de l'analyse moderne, tandis que Weierstrass systématise ces concepts et montre l'importance des développements en série et des fonctions pathologiques (comme les fonctions continues nulle part dérivables).

Riemann et les fonctions complexes

Bernhard Riemann (1826–1866) révolutionne l'analyse complexe et la géométrie. Ses travaux sur les fonctions holomorphes, les surfaces de Riemann, et l'intégration complexe (théorème des résidus) deviennent centraux en physique mathématique.

Il introduit aussi une nouvelle approche de la géométrie, où les propriétés des espaces sont définies par des métriques locales, influençant plus tard la relativité générale.

3.4.4 Algèbre abstraite : Groupes, anneaux, et corps

Galois et la théorie des groupes

Évariste Galois (1811–1832), dans un mémoire publié à titre posthume, introduit la notion de groupe pour étudier la résolubilité des équations polynomiales par radicaux. Il montre que chaque équation est associée à un groupe de permutations de ses racines, et que la résolubilité dépend de la structure de ce groupe. Ses idées, d'abord incomprises, deviennent fondatrices de l'algèbre moderne et de la théorie des structures.

Relation entre racines de l'équation et coefficients du polynôme

Équation du 3^{e} degré	$ax^3 + bx^2 + cx + d = 0$
Somme des racines	$\alpha + \beta + \gamma = -\frac{b}{a}$
Somme-Produit	$\alpha\beta + \alpha\gamma + \beta\gamma = \frac{c}{a}$
Produit des racines	$\alpha\beta\gamma = -\frac{d}{a}$

L'émergence des structures algébriques

À la fin du XIX^{e} siècle, les mathématiciens (Dedekind, Kronecker, Weber) généralisent les idées de Galois en définissant des structures abstraites :

- **Groupes** : ensembles munis d'une opération associative, avec élément neutre et inverses.
- **Anneaux** : ensembles où l'on peut additionner, soustraire et multiplier.
- **Corps** : ensembles où l'on peut aussi diviser (sauf par zéro). Ces concepts unifient l'algèbre, la géométrie, et la théorie des nombres, et préparent l'algèbre moderne du XX^{e} siècle.

Synthèse : Un héritage fondateur

Le XIX^{e} siècle voit les algorithmes et les mathématiques subir une métamorphose profonde **:**

- **Formalisation** : Boole, Cantor et Hilbert transforment la logique et les fondements en disciplines rigoureuses.

- **Mécanisation** : Babbage et Lovelace préfigurent l'informatique en imaginant des machines programmables.
- **Rigueur** : Cauchy, Weierstrass et Riemann posent les bases de l'analyse moderne.
- **Abstraction** : Galois et ses successeurs inventent l'algèbre des structures, unifiant des domaines autrefois disjoints.

Ces avancées préparent directement le XX^{e} siècle, où la théorie de la calculabilité (Turing, Church), l'informatique, et les mathématiques pures et appliquées vont connaître un essor sans précédent.

3.5 Le XX^{e} siècle marque la naissance de l'informatique moderne

L'algorithme devient un objet scientifique central, fondant les machines programmables.

Une révolution scientifique, technique et sociale sans précédent !

De la formalisation théorique des algorithmes à l'avènement d'Internet, en passant par l'intelligence artificielle et la transformation des sociétés, cette période voit l'émergence d'une nouvelle ère : celle du numérique.

L'algorithme qui a appris à coder… en lisant du code

Des chercheurs ont entraîné un modèle sur des millions de lignes de code open source.

L'algorithme a appris à compléter des fonctions, corriger des erreurs, proposer des optimisations. Dans certains cas, il a même inventé des solutions originales, jamais écrites par un humain.

Il ne comprend pas le code comme un programmeur, mais il reconnaît des motifs, des structures, des intentions.

Un outil qui transforme la programmation en dialogue.

3.5.1 La crise de la calculabilité : quand les mathématiciens rencontrent les limites

Dans les années 1930, les mathématiciens se heurtent à une question vertigineuse :

Existe-t-il une méthode générale pour décider si une proposition mathématique est vraie ou fausse ?

David Hilbert rêve d'un tel procédé, mais Kurt Gödel, en 1931, montre que ce programme est impossible : certaines vérités mathématiques sont indécidables. Dans la foulée, Alonzo Church et Alan Turing, en 1936, définissent ce qu'est une méthode effective, c'est-à-dire un algorithme au sens moderne.

Turing, en particulier, propose un modèle conceptuel — la machine de Turing — capable d'exécuter toute procédure algorithmique. Ce modèle est abstrait, mais il a une conséquence majeure :

Tout calcul peut être réduit à une suite d'opérations simples exécutées mécaniquement.

L'idée d'un ordinateur universel est née, mais elle reste théorique.

3.5.2 La guerre : catalyseur de l'informatique

La Seconde Guerre mondiale transforme ces idées en réalité. Les besoins militaires — cryptanalyse, balistique, calculs de trajectoires — exigent des machines rapides et fiables.

En Grande-Bretagne, Alan Turing participe au projet Bombe, une machine électromécanique destinée à casser les codes Enigma. La Bombe n'est pas un ordinateur au sens moderne, mais elle exécute une procédure algorithmique complexe, répétitive, systématique. Elle montre que des machines peuvent résoudre des problèmes que l'esprit humain ne pourrait traiter qu'au prix d'efforts colossaux.

ANECDOTE

Alan Turing et la course à pied

Turing, père de la théorie des algorithmes, était aussi un coureur de fond exceptionnel.

Il s'entraînait en courant les 60 km entre son domicile et son laboratoire. Il faillit même participer aux Jeux olympiques de 1948.

Un homme capable de battre des records… sur la piste comme sur le papier.

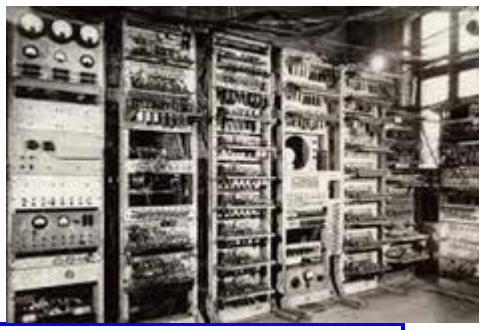
En 1943, une autre machine voit le jour : Colossus, conçue par Tommy Flowers. C'est le premier ordinateur électronique programmable. Il utilise des tubes à vide pour effectuer des opérations logiques à grande vitesse. Colossus n'est pas universel — il est spécialisé dans la cryptanalyse — mais il prouve qu'un calcul peut être automatisé à une échelle inédite.

3.5.3 ENIAC : l'ordinateur généraliste

En 1946, aux États-Unis, l'ENIAC (Electronic Numerical Integrator and Computer) marque une étape décisive. C'est le premier ordinateur électronique généraliste. Il peut être reprogrammé pour effectuer différents calculs, même si la reprogrammation nécessite de reconnecter physiquement des câbles.

L'ENIAC est gigantesque — 30 tonnes, 18 000 tubes à vide — mais il réalise en quelques secondes des calculs qui prenaient des heures à des équipes entières de mathématiciens. Pour la première fois, une machine peut exécuter des algorithmes complexes à une vitesse qui dépasse largement les capacités humaines.

Une anecdote célèbre illustre l'impact de l'ENIAC : lors de la mise au point d'un calcul balistique, les ingénieurs découvrent que la machine donne un résultat différent de celui obtenu par les calculateurs humains. Après vérification, c'est l'ENIAC qui a raison. La confiance dans les machines commence à s'installer.



ANECDOTE

Le premier bug informatique… était un vrai insecte

En 1947, l'équipe de Grace Hopper découvre qu'un ordinateur Mark II ne fonctionne plus.

La cause : un papillon de nuit coincé dans un relais.

Ils collèrent l'insecte dans le journal de bord avec la mention : « First actual case of bug being found. » (premier cas concret de détection d'un bug, littéralement : punaise, insecte)

Le mot bug existait déjà, mais l'histoire était trop belle pour ne pas devenir légendaire.

3.5.4 L'architecture de von Neumann : la naissance du programme enregistré

En 1945, John von Neumann propose une architecture qui deviendra le modèle de tous les ordinateurs modernes :

- un processeur,
- une mémoire unique contenant à la fois les données et les instructions,
- un système d'entrée et de sortie,
- un séquenceur contrôlant l'exécution.

Cette idée — stocker le programme dans la même mémoire que les données — est révolutionnaire. Elle permet de modifier un programme sans toucher au matériel. Elle ouvre la voie à la programmation telle que nous la connaissons.

Les premiers ordinateurs à programme enregistré, comme l'EDSAC (1949) ou l'UNIVAC (1951), montrent immédiatement la puissance de cette approche. Les algorithmes ne sont plus des schémas abstraits : ce sont des objets manipulables, modifiables, exécutables.

Les premiers ordinateurs… sensibles à la météo

Les premiers calculateurs électroniques chauffaient tellement qu'on devait ouvrir les fenêtres pour les refroidir.

Quand il pleuvait, on les refermait, et la machine se mettait à surchauffer. Les ingénieurs disaient en plaisantant que l'ordinateur « préférait le temps sec ».

Un algorithme dépendant de la météo : unique dans l'histoire.

3.5.5 L'algorithmique moderne : efficacité, structure, abstraction

Dans les années 1960, l'algorithmique devient une discipline à part entière. Edsger Dijkstra, Donald Knuth, Robert Floyd et d'autres posent les bases de l'analyse d'algorithmes :

- efficacité asymptotique,
- structures de contrôle,
- preuves de correction,
- optimisation.

Knuth, dans *The Art of Computer Programming*, entreprend de classifier les algorithmes comme on classifie les espèces en biologie. Dijkstra, avec son algorithme du plus court chemin (1959), montre qu'une idée simple peut résoudre un problème fondamental.

L'algorithmique devient un domaine scientifique autonome, avec ses méthodes, ses critères, ses outils.

Le "0 divisé par 0" qui a failli rendre fou un mathématicien

Dans les années 1950, un programmeur oublie de prévoir ce qui se passe lorsqu'une division par zéro survient dans un calcul.

Résultat : la machine, incapable de gérer l'erreur, se met à cracher des pages entières de symboles incohérents, sans jamais s'arrêter.

Le mathématicien responsable, voyant la pile de feuilles grandir à vue d'œil, finit par s'exclamer, mi-amusé mi-désespéré : « Elle essaie de me dire quelque chose ! »

En réalité, la machine ne tentait pas de communiquer une vérité profonde. Elle essayait simplement… de ne pas s'effondrer.

3.5.6 Les fondements théoriques : Turing, Church et la calculabilité

La machine de Turing et l'algorithme universel

En 1936, Alan Turing (1912–1954) publie un article fondateur, « On Computable Numbers », où il introduit le concept de machine de Turing : un modèle abstrait capable d'exécuter n'importe quel algorithme, pourvu qu'il soit décrit sous forme de règles finies. Turing montre aussi l'existence d'une machine universelle, capable de simuler toute autre machine de Turing, posant ainsi les bases théoriques de l'ordinateur programmable. Son travail répond au « problème de la décision » (Entscheidungsproblem) posé par David Hilbert, et démontre qu'il existe des problèmes indécidables (comme le problème de l'arrêt) — une limite fondamentale du calcul algorithmique.

Parallèlement, Alonzo Church développe le lambda-calcul, un autre modèle de calcul universel, équivalent à la machine de Turing. La thèse de Church-Turing énonce que tout ce qui est calculable l'est par une machine de Turing ou par le lambda-calcul, unifiant ainsi les approches théoriques de la calculabilité.

L'architecture de von Neumann : Naissance de l'ordinateur moderne

John von Neumann (1903–1957), inspiré par les travaux de Turing, formalise en 1945 l'architecture des ordinateurs modernes :

- Mémoire unique stockant à la fois les données et les programmes (principe du « programme enregistré »).
- Unité centrale de traitement (CPU) exécutant les instructions.
- Unités d'entrée/sortie pour communiquer avec l'extérieur.

Cette architecture, toujours dominante aujourd'hui, est mise en œuvre dans les premiers ordinateurs électroniques comme l'EDVAC et l'ENIAC, marquant le passage des machines à calculer aux ordinateurs universels.

3.5.7 Langages de programmation : De l'assembleur aux paradigmes modernes

Les premiers langages et la révolution du haut niveau

Afin de faciliter l'écriture de programmes complexes, les chercheurs et ingénieurs développent les premiers langages de programmation. Ces langages représentent une avancée majeure, car ils permettent de structurer les instructions de manière plus lisible et compréhensible pour l'humain, tout en étant traduites efficacement pour la machine.

Au cours des années 1950 et 1960, il devient évident pour les informaticiens et les ingénieurs que la programmation directe en codes binaires est extrêmement laborieuse et source d'erreurs. La nécessité d'un langage intermédiaire s'impose alors : celui-ci doit servir de pont entre la logique humaine et l'exécution matérielle, simplifiant ainsi la conception de programmes tout en rendant le développement accessible à un plus grand nombre d'utilisateurs.

En 1951, Grace Hopper développe le premier compilateur, capable de traduire un langage proche de l'anglais en instructions machine. Quelques années plus tard, apparaissent les premiers langages de haut niveau :

- **Fortran** (1954, John Backus) : premier langage de haut niveau, conçu pour le calcul scientifique. 1957 pour le calcul scientifique,
- **Lisp** (1958, John McCarthy) : premier langage fonctionnel (manipulation symbolique), basé sur le lambda-calcul, toujours utilisé en IA.
- **Cobol** (1959) : langage orienté gestion, dominant dans les entreprises pendant des décennies.
- **Algol** (1960) : introduit la programmation structurée, influençant le C et Pascal.

	Algol : l'art de programmer avant que programmer n'existe vraiment

1967 **1968** ANECDOTE	Algol… rien que le nom sent la craie, les cartes perforées et l'audace intellectuelle. Né à la fin des années 1950, ce langage précurseur fut conçu par une équipe internationale de mathématiciens et d'informaticiens — parmi lesquels Peter Naur, Friedrich Bauer, John Backus et quelques autres esprits brillants — mais il faut bien reconnaître que Grenoble, avec son école de mathématiques appliquées, fut l'un des tout premiers foyers français à s'en emparer et à le faire rayonner. Et c'est avec ce drôle d'outil, à mi-chemin entre la logique pure et la poésie mécanique, que l'auteur de ce livre fit ses premières armes à l'INSA de Lyon, en 1967-68. À l'époque, la programmation n'était pas encore un métier : c'était une aventure. Une discipline neuve, presque exotique, où l'on avançait à tâtons, comme des explorateurs découvrant un continent invisible. Les étudiants, soudain confrontés à cette nouvelle manière de penser, avaient bien du mal à adopter l'état d'esprit nécessaire. Il fallait apprendre à raisonner autrement, à découper le monde en instructions, à parler à une machine qui ne comprenait que ce qu'on lui disait — et encore, seulement si on le disait *parfaitement*. Le plus savoureux, c'est que les professeurs enseignaient Algol… en Algol. Une sorte de miroir intellectuel où le langage servait à expliquer le langage, comme si l'on apprenait le français en lisant Molière sans dictionnaire. Un cycle mental vertigineux qu'il fallait apprivoiser, au prix de quelques migraines et de beaucoup de persévérance. Mais c'est précisément cela qui fait le charme de cette époque : nous étions aux débuts, sans manuel, sans Internet, sans IA pour nous souffler la solution. Juste des cerveaux, des idées neuves, et une bonne dose de témérité. Et il faut bien l'avouer : nous avions du mérite.

Ces nouveaux langages transforment la manière de concevoir les algorithmes. On peut désormais écrire des procédures complexes sans se soucier des détails matériels. L'algorithme devient un objet textuel, transmissible, modifiable, partageable.

Le saviez-vous ?

ALGOL – Programme

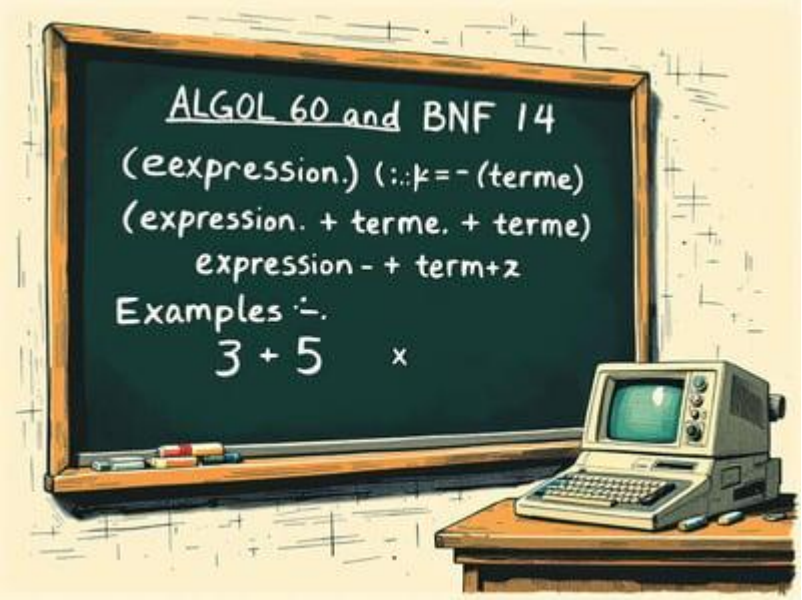
Voici un exemple historique et représentatif de la syntaxe d'Algol 60, le langage qui a marqué l'histoire de l'informatique dans les années 1960. Algol (pour ALGOrithmic Language) était conçu pour être lisible, structuré et mathématiquement précis, ce qui en a fait un modèle pour les langages modernes comme C, Pascal ou Ada

Exemple : Résolution d'une équation quadratique

```
'Calcul des racines d''ax² + bx + c = 0' ;
BEGIN
    REAL a, b, c, discriminant, x1, x2;
    a := 1.0; b := -3.0; c := 2.0; 'Valeurs d''exemple' ;

    discriminant := b * b - 4 * a * c;

    IF discriminant >= 0 THEN
    BEGIN
        x1 := (-b + SQRT(discriminant)) / (2 * a);
        x2 := (-b - SQRT(discriminant)) / (2 * a);
        OUTSTRING(1, "Racines réelles: ");
        OUTREAL(1, x1); OUTSTRING(1, ", "); OUTREAL(1, x2);
    END
    ELSE
        OUTSTRING(1, "Pas de racines réelles.");
END
```



Le saviez-vous ?

ALGOL – Comment Algol 60 se définissait-il ? La révolution de la notation BNF

Dans les années 1960, Algol 60 a marqué un tournant en introduisant une méthode rigoureuse pour décrire sa syntaxe : la notation BNF (*Backus-Naur Form*). Cette approche formelle permettait de définir précisément comment écrire des instructions dans le langage, en évitant toute ambiguïté.

Prenons un exemple concret avec la définition d'une expression arithmétique :

<expression> ::= <terme> | <expression> + <terme> | <expression> - <terme>

Que signifie cette notation ?

- **<expression>** : Une expression arithmétique (comme 3 + 5 ou x - y).
- **::=** : Se lit *"est défini comme"* et introduit la définition.
- **|** : Signifie *"ou"* et sépare les différentes possibilités.

Ainsi, une **expression** peut être :

1. Un simple **<terme>** (exemple : 5).
2. Une **<expression> suivie de + ou - et d'un <terme>**
   Exemples : 3 + 5 ou x – y.

Cette notation, bien que technique, a permis de standardiser la description des langages de programmation. Elle a ouvert la voie à des langages plus structurés et a inspiré les grammaires des langages modernes comme C, Java ou Python.

L'émergence des paradigmes

Les années 1970–1980 voient l'apparition de nouveaux paradigmes de programmation :

- **Programmation structurée** (C, Pascal) : organisation claire du code en blocs.
- **Programmation orientée objet** (Simula, Smalltalk, C++) : modélisation des données et des comportements sous forme d'objets.
- **Programmation fonctionnelle** (ML, Haskell) : accent sur les fonctions et l'immutabilité.
- **Programmation logique** (Prolog) : basée sur la logique formelle et les règles.

Ces innovations permettent de développer des logiciels toujours plus complexes, adaptés à des usages variés (scientifique, gestion, IA, systèmes embarqués).

Le saviez-vous ?

PROLOG

Prolog (pour PROgrammation en LOGique) est un langage de programmation déclaratif basé sur la logique des prédicats. Contrairement aux langages impératifs (comme C ou Python), où on décrit comment résoudre un problème, Prolog se concentre sur ce qui est vrai : on définit des faits et des règles, puis on pose des questions (requêtes) pour obtenir des réponses.

Exemple de programme Prolog :

1. Définition des faits :

```
% Faits décrivant des relations familiales :
parent(john, mary).   % John est le parent de Mary.
parent(mary, sue).    % Mary est le parent de Sue.
parent(john, bob).    % John est le parent de Bob.
```

. Définition d'une règle :

```
% Règle pour définir un grand-parent :
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
Cette règle signifie : "X est le grand-parent de Y si X est le parent de Z et Z est le
    parent de Y."
```

3. Exemple de requête :
   ?- grandparent(X, mary).
4. Prolog répondra :
   X = john.
   Cela signifie que John est le grand-parent de Mary.

Explications :

- Faits : Ce sont des assertions simples (ex : parent(john, mary)).
- Règles : Elles définissent des relations complexes à partir de faits (ex : grandparent).
- Requêtes : Ce sont des questions posées à Prolog (ex : ?- grandparent(X, mary).).

Prolog est particulièrement utile pour les problèmes de recherche de relations, intelligence artificielle symbolique, et traitement du langage naturel.

3.5.8 La révolution numérique : Ordinateurs, réseaux et Internet

La miniaturisation et la démocratisation

Les progrès de l'électronique (transistors, circuits intégrés, microprocesseurs) rendent les ordinateurs toujours plus puissants, petits et abordables :

- Années 1970 : premiers micro-ordinateurs (Apple II, Commodore, IBM PC).
- Années 1980–1990 : explosion des PC, interfaces graphiques (Windows, MacOS), et logiciels grand public (traitement de texte, jeux vidéo).

ANECDOTE

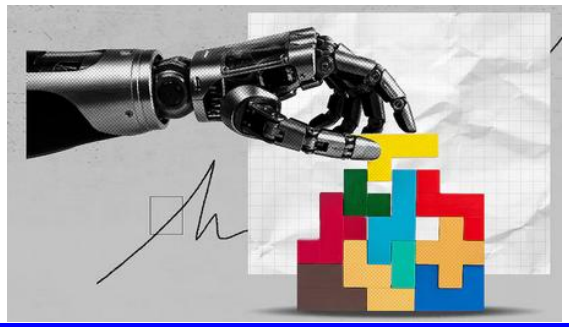
L'algorithme qui a appris à jouer à un jeu… en regardant YouTube

Des chercheurs ont montré qu'un algorithme pouvait apprendre à jouer à des jeux vidéo simplement en regardant des vidéos de joueurs humains sur YouTube.

Sans connaître les règles, sans accès au code du jeu, sans instructions. Il observe, imite, puis optimise. Dans certains cas, il a même surpassé les joueurs qu'il avait observés.

Un apprentissage purement visuel, comme un enfant qui regarde un adulte jouer.

Une démonstration spectaculaire de la puissance des algorithmes d'observation.



La naissance d'Internet et du Web

- **ARPANET** (1969) : premier réseau à commutation de paquets, ancêtre d'Internet.
- **Protocoles TCP/IP** (années 1970) : standardisent la communication entre machines.
- **World Wide Web** (1989, Tim Berners-Lee) : invente le HTML, les URL et le HTTP, rendant l'information accessible à tous via un navigateur.
- **Années 1990** : explosion du Web, naissance des moteurs de recherche (Google), du e-commerce, et des réseaux sociaux.

Impact sociétal et économique

La révolution numérique transforme profondément la société :

- Automatisation des tâches (industrie, administration, services).
- Globalisation des échanges (finance, commerce, culture).
- Nouveaux métiers (développeurs, data scientists, experts en cybersécurité).
- Défis éthiques : vie privée, désinformation, fracture numérique.

3.5.9 Intelligence artificielle : Des réseaux de neurones à l'apprentissage profond

Les prémices : McCulloch, Pitts et le perceptron

Dès 1943, Warren McCulloch et Walter Pitts modélisent le neurone artificiel, inspirant les premiers réseaux de neurones. Dans les années 1950–1960, Frank Rosenblatt invente le Perceptron, premier algorithme d'apprentissage supervisé, capable de classer des données simples.

L'hiver et le renouveau de l'IA

- Années 1970–1980 : désillusion (« hiver de l'IA ») face aux limites des approches symboliques et des perceptrons.
- Années 1980–1990 : retour en grâce avec les réseaux de neurones multicouches et l'algorithme de rétropropagation (Rumelhart, Hinton, Williams). Les réseaux convolutifs (LeCun, 1989) révolutionnent la vision par ordinateur.

L'essor de l'apprentissage automatique

À partir des années 2000, l'IA connaît un essor fulgurant grâce à :

- Big Data : explosion des données disponibles (Web, capteurs, réseaux sociaux).
- Puissance de calcul : GPU et clusters permettant d'entraîner des modèles complexes.
- Deep Learning : réseaux profonds (AlexNet, 2012) surpassant l'homme dans des tâches comme la reconnaissance d'images ou la traduction automatique.
- Applications : assistants vocaux, voitures autonomes, diagnostic médical, génération de texte (comme les modèles de langage actuels).

3.5.10 Synthèse : Un héritage transformateur

Le XX^{e} siècle a vu naître l'informatique moderne, marquée par :

- Théorie : formalisation de la calculabilité (Turing, Church), fondements logiques et algébriques.
- Technologie : ordinateurs universels, langages de programmation, réseaux, Internet.
- Société : automatisation, globalisation, nouveaux défis éthiques et économiques.
- IA : des réseaux de neurones aux algorithmes d'apprentissage profond, préfigurant les avancées du XXI^{e} siècle.

Cette révolution a non seulement transformé les sciences et les industries, mais aussi reconfiguré les rapports humains, faisant du numérique un pilier de la civilisation contemporaine.

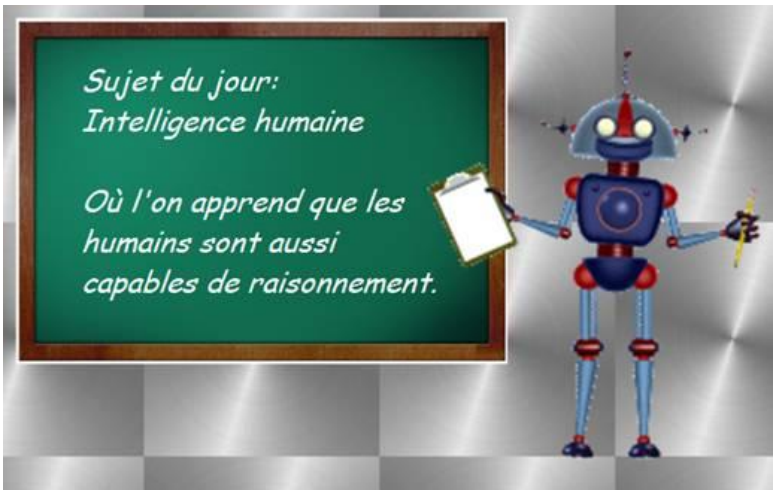
3.6 Le XXIe siècle est marqué par l'ère numérique et l'intelligence artificielle

Les algorithmes s'invitent partout et structurent silencieusement la société connectée.

L'ère numérique et de l'intelligence artificielle (IA), deux forces qui transforment radicalement les sociétés, les économies et les équilibres géopolitiques.

De l'explosion des algorithmes d'apprentissage profond à la montée en puissance des géants du numérique (GAFAM, BATX), en passant par les défis éthiques, économiques et technologiques, cette période est celle d'une révolution sans précédent.

Voici une analyse détaillée des principaux enjeux et avancées de ce siècle numérique.



3.6.1 L'avènement de l'intelligence artificielle : les algorithmes implicites

Depuis le début du XXIe siècle, une nouvelle forme d'algorithmes s'est imposée : ceux issus de l'intelligence artificielle, et en particulier de l'apprentissage automatique. Contrairement aux algorithmes traditionnels, qui suivent une suite d'instructions définies à l'avance, ces méthodes apprennent à partir de données. Elles ajustent leurs paramètres pour améliorer leurs performances, sans que chaque étape soit explicitement programmée.

Les réseaux de neurones en sont un exemple emblématique. Ils sont capables de reconnaître des images, de traduire des textes ou de prédire des comportements, mais leur fonctionnement interne est souvent difficile à interpréter. On parle alors d'algorithmes implicites, car leur logique n'est pas entièrement visible. Cette évolution soulève de nouvelles questions sur la transparence, la fiabilité et l'éthique.

L'intelligence artificielle ne remplace pas les algorithmes traditionnels : elle les complète. Les deux approches coexistent et répondent à des besoins différents. L'histoire récente montre que les algorithmes continuent d'évoluer, portés par les progrès technologiques et par les nouveaux défis du monde numérique.

3.6.2 L'intelligence artificielle et l'apprentissage profond : Une révolution algorithmique

L'essor de l'apprentissage profond (Deep Learning)

L'apprentissage profond, appelé en anglais deep learning, est devenu l'un des principaux moteurs de l'intelligence artificielle moderne. Cette approche repose sur des réseaux de neurones artificiels comportant de nombreuses couches, inspirés du fonctionnement du cerveau humain.

Ces réseaux sont capables de modéliser des données complexes telles que les images, les sons ou les textes, en apprenant automatiquement des représentations de plus en plus abstraites à partir de grandes quantités de données. Plusieurs avancées majeures ont marqué le XXIe siècle.

Réseaux de neurones convolutifs (CNN – Convolutional Neural Networks) :

Ces réseaux sont spécialement conçus pour traiter des données structurées spatialement, comme les images.

Ils ont révolutionné la vision par ordinateur, en permettant des progrès spectaculaires dans la reconnaissance d'images, la détection d'objets, le diagnostic médical assisté par ordinateur ou encore le développement des véhicules autonomes.

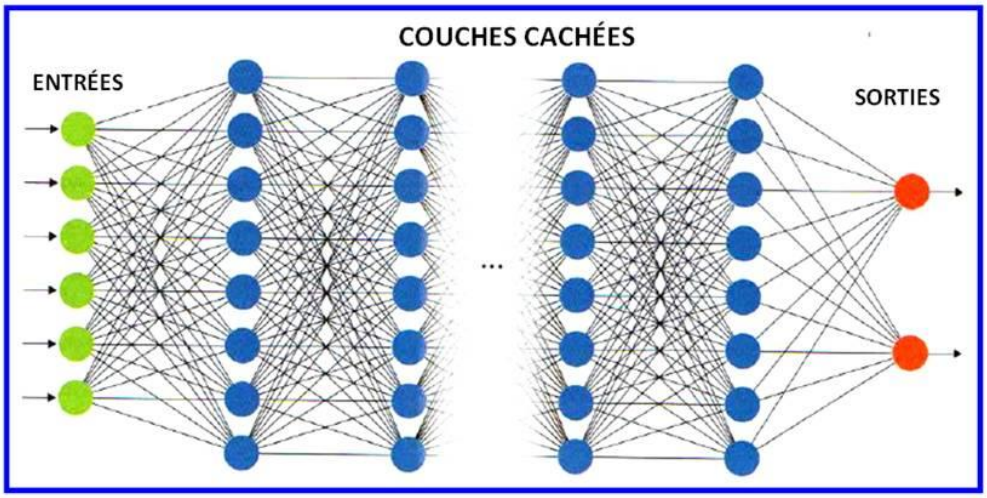
Réseaux de neurones récurrents (RNN – Recurrent Neural Networks) et LSTM (Long Short-Term Memory) :

Ces architectures sont adaptées au traitement des données séquentielles, c'est à dire des données ordonnées dans le temps. Elles sont utilisées pour analyser des phrases, des signaux sonores ou des séries temporelles.

Les modèles LSTM, une variante des réseaux récurrents, permettent de mieux gérer les dépendances à long terme, ce qui les rend particulièrement efficaces pour la traduction automatique, la reconnaissance vocale ou l'analyse de textes.

Découverte

Structure d'un réseau de neurones



Les réseaux de neurones reposent sur des associations de valeurs, de simples chiffres, à l'aide de fonctions mathématiques reliant les sorties aux entrées. Chaque point d'une image, ou d'un texte, se voit attribuer des valeurs d'entrée. Toute la magie repose sur l'entrainement des fonctions à mémoriser les situations rencontrées (entrées et sorties connues) pour réussir à effectuer une tâche nouvelle de manière cohérente pour nous.

Source image: Découverte n°446 – Juillet-septembre 2024

Transformers (introduits en 2017) :

Les transformers constituent une nouvelle architecture de réseaux de neurones fondée sur un mécanisme appelé attention, qui permet au modèle de se concentrer sur les éléments les plus pertinents d'une séquence de données. Cette approche a profondément transformé le traitement

automatique du langage naturel (NLP – Natural Language Processing), c'est à dire l'analyse et la génération de textes par des machines.

Des modèles emblématiques comme BERT (Bidirectional Encoder Representations from Transformers), développé par Google, ou GPT (Generative Pretrained Transformer), conçu par OpenAI, sont capables de comprendre des textes, de répondre à des questions, de rédiger des contenus, de produire du code informatique et même de générer des images dans des systèmes comme DALL·E ou MidJourney.

3.6.3 Les modèles de langage de grande taille (LLM – Large Language Models)

Les modèles de langage de grande taille, appelés en anglais LLM (Large Language Models), constituent l'une des applications les plus marquantes de l'apprentissage profond contemporain. Il s'agit de modèles d'intelligence artificielle entraînés sur des volumes massifs de textes afin d'apprendre les régularités du langage humain. Leur objectif n'est pas de « comprendre » au sens humain, mais de prédire statistiquement le mot, ou le symbole, le plus probable en fonction d'un contexte donné. Cette capacité leur permet de produire des textes cohérents, de répondre à des questions, de résumer des documents, de traduire des langues ou encore de générer du code informatique.

Les LLM reposent principalement sur l'architecture des transformers, qui permet de traiter efficacement de longues séquences de texte grâce au mécanisme d'attention. Contrairement aux approches plus anciennes, ces modèles analysent un texte dans son ensemble et prennent en compte les relations entre les mots, même lorsqu'ils sont éloignés dans la phrase ou le document. L'entraînement s'effectue en deux grandes étapes : une phase de pré-entraînement, au cours de laquelle le modèle apprend les structures générales du langage à partir de données textuelles très variées, puis une phase d'ajustement (fine-tuning), qui spécialise le modèle pour des tâches ou des usages particuliers.

Des modèles emblématiques comme GPT (Generative Pretrained Transformer), développé par OpenAI, ou BERT (Bidirectional Encoder Representations from Transformers), conçu par Google, illustrent cette approche. Les versions les plus récentes de ces modèles comptent des milliards de paramètres ajustables, ce qui leur confère une grande flexibilité mais aussi une forte opacité. Les LLM sont ainsi capables de produire des réponses convaincantes, tout en rendant difficile l'explication précise de leurs décisions internes. Cette puissance soulève des enjeux importants en matière de fiabilité, de biais, de consommation énergétique et de responsabilité, qui s'ajoutent aux questions plus générales posées par l'intelligence artificielle moderne.

3.6.4 Mesurer l'intelligence artificielle : histoire, limites et renouveau des benchmarks

Aux origines de la mesure : quand l'IA devait encore prouver qu'elle comprenait

Lorsque les premiers systèmes de traitement automatique du langage ont commencé à émerger, leur évaluation reposait sur des tests simples, centrés sur la capacité à reconnaître des relations logiques élémentaires ou à reformuler des phrases. Les chercheurs cherchaient avant tout à vérifier que la machine comprenait ce qu'elle lisait, ou du moins qu'elle se comportait comme si elle comprenait.

Les premiers benchmarks, comme GLUE, s'inscrivaient dans cette logique. Ils proposaient des tâches de classification, d'inférence ou de similarité sémantique, et les modèles devaient montrer qu'ils pouvaient identifier une contradiction, reconnaître une paraphrase ou déterminer le ton d'un texte. À cette époque, les systèmes restaient spécialisés, et chaque tâche représentait un défi distinct. Les progrès étaient mesurés par des pourcentages qui grimpaient lentement, année après année, au rythme des innovations architecturales.

Cette période a façonné une manière de penser l'évaluation : un benchmark était un obstacle fixe, un terrain de jeu stable sur lequel les modèles pouvaient être comparés. Les chercheurs imaginaient rarement qu'un jour, ces tests seraient résolus presque instantanément, comme si la machine avait appris à jouer à un jeu dont elle connaissait déjà toutes les réponses.

Pourtant, c'est exactement ce qui s'est produit lorsque les modèles généralistes ont commencé à apparaître. GPT-3, PaLM, LLaMA et leurs successeurs ont bouleversé la logique même de l'évaluation. Ils n'étaient plus conçus pour une tâche précise, mais pour absorber des milliers de compétences simultanément. Ils pouvaient résumer un texte, résoudre une équation, écrire un poème, analyser un jugement juridique et commenter un tableau de maître, sans qu'aucune de ces activités ne soit explicitement programmée.

Les benchmarks classiques ont alors été saturés. Les modèles atteignaient des scores proches de la perfection, parfois supérieurs à ceux d'étudiants humains. Les chercheurs ont compris que ces tests ne mesuraient plus la compréhension, mais la capacité des modèles à reconnaître des motifs statistiques dans des données qu'ils avaient déjà vues ou dont ils avaient appris les structures profondes. La question n'était plus de savoir si une IA pouvait réussir un test, mais si le test avait encore un sens.

C'est dans ce contexte qu'un nouveau type de benchmark a émergé, conçu pour dépasser les limites des tests traditionnels et pour mesurer non plus la reconnaissance, mais le raisonnement.

Le tournant MMLU : la première tentative d'un test vraiment multidisciplinaire

Lorsque Dan Hendrycks a introduit MMLU (Massive Multitask Language Understanding), en 2020, il cherchait à créer un benchmark capable de capturer la polyvalence intellectuelle des modèles. Le test rassemblait des milliers de questions couvrant cinquante-sept disciplines, allant de la médecine à l'histoire, de la physique à la philosophie, du droit aux mathématiques. Pour la première fois, un benchmark tentait de mesurer la capacité d'un modèle à naviguer entre des domaines hétérogènes, comme le ferait un étudiant confronté à un examen généraliste.

Pendant un temps, MMLU a rempli son rôle. Les modèles progressaient lentement, et les écarts entre eux étaient significatifs. Mais dès 2022, certains systèmes atteignaient déjà des scores comparables à ceux d'étudiants de premier cycle universitaire. En 2023, plusieurs modèles dépassaient même les performances humaines sur certaines sections. Le test, pourtant ambitieux, devenait insuffisant. Il ne permettait plus de distinguer les modèles entre eux, ni de mesurer une véritable compréhension conceptuelle. Il révélait surtout une limite structurelle : un benchmark statique finit toujours par être absorbé par les modèles, qui apprennent à en reconnaître les motifs, même sans l'avoir vu explicitement.

Cette saturation a ouvert la voie à une réflexion plus profonde : comment concevoir un test qui ne puisse pas être appris par cœur ? Comment créer un benchmark qui ne soit pas un simple catalogue de questions, mais une épreuve capable de résister à l'évolution rapide des modèles ? C'est dans cette quête qu'est né *Humanity's Last Exam*.

Humanity's Last Exam : un test conçu pour rester hors de portée

Lorsque Phan Long a imaginé *Humanity's Last Exam* (HLE) en 2024, en collaboration avec Dan Hendrycks, il ne cherchait pas à créer un benchmark de plus, mais à établir une frontière. Les progrès rapides des modèles de langage avaient rendu obsolètes les tests traditionnels, et même les benchmarks multidisciplinaires comme MMLU ne suffisaient plus à distinguer les modèles entre eux. Long et Hendrycks voulaient concevoir une épreuve capable de mesurer la profondeur réelle du raisonnement algorithmique, au-delà de la simple reconnaissance statistique. Leur ambition était de créer un test qui ne puisse pas être absorbé par les modèles, un test qui resterait hors de portée, même pour les systèmes les plus avancés.

Pour atteindre cet objectif, ils ont décidé de construire un corpus entièrement original. Aucune question ne devait provenir de sources existantes, afin d'éviter toute contamination par les données d'entraînement. Ils ont mobilisé un millier de professeurs et de chercheurs de renommée internationale pour élaborer ces questions. Parmi eux figuraient Richard Stanley du MIT, spécialiste de combinatoire, David Aldous de Berkeley, expert en probabilités et algorithmes, Ciprian Manolescu de Stanford, figure majeure de la topologie, ou encore Alexander Shen, affilié à l'université de Montpellier et reconnu pour ses travaux en théorie de l'information. Les questions couvraient des domaines très spécialisés, allant des mathématiques avancées à la linguistique historique, de la chimie organique à la mythologie comparée. Elles exigeaient une compréhension conceptuelle profonde, souvent au niveau de la recherche académique.

Pour garantir l'intégrité du test, deux mécanismes antitriche ont été mis en place. Le premier consistait en un *canary string*, un filigrane textuel intégré dans les données du HLE, permettant d'identifier et d'exclure automatiquement ces données des corpus d'entraînement utilisés par les laboratoires. Le second reposait sur un ensemble de questions secrètes, non publiées, qui devait être révélé ultérieurement. Ce dispositif permettrait de distinguer les modèles ayant réellement développé des capacités de raisonnement avancées de ceux qui auraient obtenu de bons résultats en ayant été exposés illicitement aux questions.

Depuis sa création, le HLE est devenu un terrain de compétition entre les modèles de langage les plus puissants. Les premières places ont changé de mains à plusieurs reprises. OpenAI a d'abord dominé, avant d'être dépassé par DeepSeek R1, puis par Gemini 2.5 Pro, puis par Grok 4. OpenAI a brièvement repris la tête, mais depuis novembre 2025, les modèles Gemini occupent largement la première place avec un score de 49,5%. Les différentes versions de Claude, développées par Anthropic, restent en embuscade. Parallèlement, les modèles chinois progressent rapidement : Kimi K2.5, avec un taux d'exactitude de 25,4 %, dépasse déjà certains modèles de ChatGPT.

Le HLE s'impose ainsi comme un indicateur essentiel de la frontière actuelle entre intelligence humaine experte et performance algorithmique. En combinant difficulté extrême, diversité disciplinaire et mécanismes antitriche, il offre un outil robuste pour suivre l'évolution des capacités des IA et évaluer leur compréhension réelle, au-delà de la simple accumulation de données. Il contribue à clarifier ce que signifie « raisonner » pour une machine et à encadrer le développement des systèmes d'intelligence artificielle de prochaine génération.

Les limites des benchmarks statiques : quand les IA apprennent plus vite que les tests n'évoluent

L'apparition de tests comme MMLU ou HLE a mis en lumière une tension profonde entre la vitesse de progression des modèles et la capacité des chercheurs à concevoir des épreuves pertinentes. Chaque benchmark, même le plus ambitieux, finit par devenir un terrain familier pour les modèles, non pas parce qu'ils en ont mémorisé les questions, mais parce qu'ils ont appris à reconnaître les structures, les formulations, les types de raisonnement attendus. Un test statique, même protégé par des mécanismes antitriche, finit toujours par être absorbé par les modèles, qui développent des stratégies internes pour optimiser leurs réponses. Cette dynamique crée un décalage permanent entre l'intention des concepteurs et la réalité des performances observées.

Les chercheurs ont constaté que les modèles pouvaient atteindre des scores élevés sans pour autant démontrer une compréhension profonde. Un modèle pouvait réussir une question de physique quantique sans avoir la moindre représentation interne des phénomènes décrits, simplement en exploitant des corrélations statistiques apprises dans des corpus gigantesques. Cette dissociation entre performance et compréhension a rendu les benchmarks traditionnels de moins en moins fiables pour évaluer les capacités réelles des systèmes. Elle a également révélé un risque plus subtil : celui de surestimer les modèles en leur attribuant des compétences qu'ils ne possèdent pas réellement.

Cette situation a conduit à une réflexion plus large sur la nature même de l'évaluation. Les chercheurs ont compris qu'un benchmark ne pouvait plus être un simple catalogue de questions, mais devait devenir un processus dynamique, capable de s'adapter en permanence aux progrès des modèles. Ils ont commencé à imaginer des tests évolutifs, des épreuves générées en temps réel, ou encore des scénarios interactifs dans lesquels les modèles doivent démontrer une capacité à planifier, à s'adapter, à corriger leurs erreurs. Cette transition marque un changement de paradigme : l'évaluation ne se limite plus à mesurer ce que le modèle sait, mais cherche à comprendre comment il pense.

Vers une nouvelle génération d'évaluations : du raisonnement à la robustesse

À mesure que les modèles de langage ont gagné en puissance, les chercheurs ont commencé à s'intéresser à des dimensions plus subtiles que la simple exactitude des réponses. Ils ont observé que certains modèles pouvaient obtenir d'excellents scores sur des benchmarks tout en se montrant fragiles face à des variations mineures dans la formulation des questions. Une simple reformulation pouvait suffire à faire chuter leurs performances, révélant une compréhension superficielle. Cette fragilité a mis en évidence la nécessité de tests capables de mesurer la robustesse, la cohérence et la stabilité des modèles.

Les laboratoires ont alors commencé à développer des évaluations centrées sur la résistance aux manipulations, la capacité à détecter des incohérences, ou encore la faculté à maintenir un raisonnement logique sur plusieurs étapes. Ces tests cherchent à mesurer non seulement ce que le modèle produit, mais aussi la manière dont il structure son raisonnement. Ils s'intéressent à la capacité du modèle à justifier ses réponses, à reconnaître ses erreurs, à corriger ses propres contradictions. Cette approche marque une rupture avec les benchmarks traditionnels, qui se contentaient de vérifier la conformité des réponses à un standard.

Parallèlement, les chercheurs ont commencé à explorer des évaluations comportementales, inspirées de la psychologie cognitive. Ils ont cherché à comprendre comment les modèles réagissent à des situations ambiguës, comment ils gèrent l'incertitude, comment ils interprètent des instructions complexes. Ces tests ne cherchent pas à mesurer la connaissance, mais la capacité à naviguer dans des environnements où les règles ne sont pas explicites. Ils s'inscrivent dans une démarche visant à comprendre non seulement ce que les modèles savent faire, mais ce qu'ils pourraient faire dans des situations nouvelles.

Cette évolution reflète une prise de conscience : l'intelligence artificielle ne peut plus être évaluée uniquement à travers des questions fermées. Les modèles doivent être confrontés à des situations qui exigent une compréhension profonde, une capacité à raisonner, à anticiper, à s'adapter. Les benchmarks de nouvelle génération cherchent à capturer cette complexité, en proposant des épreuves qui ne peuvent pas être résolues par la simple reconnaissance de motifs.

L'enjeu stratégique de l'évaluation : comprendre pour maîtriser

À mesure que les modèles de langage deviennent plus puissants, l'évaluation devient un enjeu stratégique. Les gouvernements, les entreprises et les institutions de recherche ont besoin de comprendre les capacités réelles des systèmes pour anticiper leurs usages, leurs limites et leurs risques. Une évaluation insuffisante peut conduire à des erreurs d'appréciation, à des déploiements prématurés, ou à une confiance excessive dans des modèles qui ne sont pas aussi fiables qu'ils en ont l'air. À l'inverse, une évaluation trop restrictive peut freiner l'innovation et empêcher l'exploration de nouvelles applications.

Les benchmarks comme *Humanity's Last Exam* jouent un rôle crucial dans cette dynamique. Ils permettent de suivre l'évolution des modèles, de détecter les progrès réels, de distinguer les avancées authentiques des illusions statistiques. Ils offrent un cadre pour comprendre ce que signifie réellement « raisonner » pour une machine, et pour identifier les domaines où les modèles

restent fragiles. Ils contribuent à établir une frontière claire entre les capacités humaines expertes et les performances algorithmiques, une frontière qui évolue constamment mais qui reste essentielle pour encadrer le développement des systèmes d'intelligence artificielle.

Cette frontière n'est pas seulement technique. Elle est aussi philosophique, politique et sociale. Elle interroge notre rapport à la connaissance, à la compétence, à l'expertise. Elle nous oblige à repenser ce que signifie comprendre, apprendre, résoudre un problème. Elle nous invite à réfléchir à la manière dont nous voulons cohabiter avec des systèmes capables de performances extraordinaires, mais dont la nature même de l'intelligence reste fondamentalement différente de la nôtre.

Conclusion – La frontière mouvante entre l'humain et la machine

L'escalade des benchmarks révèle quelque chose de plus profond que la simple progression technique des modèles. À chaque fois que le niveau des tests s'élève, les systèmes parviennent à s'en approcher, comme si la frontière entre intelligence humaine et intelligence artificielle se déplaçait sous nos yeux. Cette frontière n'est jamais fixe. Elle recule au rythme des modèles, qui absorbent les épreuves les plus ardues avec une régularité mécanique. Cette dynamique donne l'impression d'une course où l'humanité ne cesse de redessiner la ligne d'arrivée pour éviter d'être rattrapée.

Pourtant, cette progression fulgurante ne signifie pas que les modèles raisonnent. Leur avancée repose sur un processus d'optimisation statistique, sans intention, sans intuition, sans compréhension vécue. Ils ne franchissent les obstacles que parce qu'ils apprennent à reconnaître les structures des problèmes, non parce qu'ils en saisissent le sens. Cette dissociation nourrit l'idée qu'un plafond de verre pourrait exister, une limite inhérente à la nature même des architectures actuelles. Si les modèles ne font qu'imiter le raisonnement sans jamais l'incarner, alors leur progression finira peut-être par se heurter à des tâches qui exigent autre chose qu'une accumulation de corrélations.

Certains chercheurs, comme Yann LeCun, voient dans cette perspective la preuve qu'il faut changer de paradigme. Selon eux, les modèles actuels ne pourront jamais atteindre, et encore moins dépasser, l'intelligence humaine tant qu'ils resteront fondés sur les mêmes principes d'apprentissage massif. Ils appellent à inventer une nouvelle forme d'IA, capable de construire des représentations du monde, d'agir, d'explorer, de comprendre par interaction plutôt que par ingestion de données. Cette vision ouvre la possibilité d'une intelligence artificielle qui ne se contenterait plus de franchir des tests, mais qui développerait une forme d'autonomie cognitive.

La trajectoire future dépendra de la manière dont ces deux forces évolueront : d'un côté, la puissance croissante des modèles actuels, capables de repousser sans cesse les limites des benchmarks ; de l'autre, la nécessité de concevoir des systèmes qui ne se contentent plus d'imiter le raisonnement, mais qui en reproduisent les mécanismes fondamentaux. À l'intersection de ces deux dynamiques se dessine une question qui dépasse la technique : jusqu'où voulons-nous aller dans la construction d'une intelligence qui rivalise avec la nôtre, et que signifie encore « intelligence » lorsque les machines progressent plus vite que les tests censés les mesurer.

3.6.5 Point sur les IA existantes

Tableau récapitulatif des outils gratuits grand public :

Catégorie	Outils phares gratuits
Conversation	Le Chat (Mistral AI), ChatGPT (GPT-3.5), Copilot (Microsoft), Perpexity (Perplexity AI), Bing AI, Claude, HuggingChat
Image	DALL·E 3 (via Bing), Stable Diffusion, Leonardo.AI
Photo	Agrandisseur d'Images AI & Améliorateur d'Images (imgupscaler.ai/fr)
Audio/Voix	ElevenLabs (version gratuite), Murf.ai (essai)
Vidéo	Runway ML (essai), Pika Labs (gratuit avec limites)
Code	GitHub Copilot (étudiants), Blackbox AI, Amazon CodeWhisperer
Productivité	Microsoft Copilot (Windows/Edge), Notion AI, Grammarly

Quelles IA utiliser au quotidien ?

1. Les IA "généralistes" pour tous les jours

Elles savent répondre à des questions, rédiger, expliquer, résumer, créer des images simples et produire du code.

IA	Points forts	Usage idéal
Le Chat (Mistral)	Rapide, francophone, très accessible	Questions du quotidien, rédaction, explications
Copilot (Microsoft)	Très polyvalent, intégré à Windows, bon pour images et code	Rédaction, recherche, Python, images
ChatGPT (OpenAI)	Très bon en écriture et en raisonnement	Textes, idées, aide aux devoirs, créativité
Perplexity	Excellent pour la recherche d'informations fiables	Comprendre un sujet, trouver des sources, synthèses

Pour un usage courant, ces quatre-là couvrent 95 % des besoins :
informations, rédaction, images simples, code Python, aide à la réflexion.

2. Les IA pour créer des images

Pour générer des illustrations, logos, personnages, paysages…

Outil	Points forts
DALL·E 3 (via Bing)	Très simple, très bon pour les images propres et réalistes
Leonardo.AI	Plus créatif, plus paramétrable, idéal pour débuter sans payer

3. Les IA pour améliorer des photos

Pour agrandir, nettoyer, améliorer la netteté ou restaurer une image.

imgupscaler.ai/fr : agrandit et améliore les photos automatiquement.Parfait pour améliorer une vieille photo, un logo flou ou une image trouvée sur le web.

4. Les IA pour le développement et le code

Pour générer, corriger ou expliquer du code (Python, JS, etc.).

Outil	Points forts
GitHub Copilot	Excellent pour coder, gratuit pour étudiants et enseignants
Blackbox AI	Très bon pour comprendre du code existant et générer des snippets
Copilot / ChatGPT	Suffisants pour du Python occasionnel ou de l'apprentissage

Pour apprendre ou coder ponctuellement : Copilot ou ChatGPT suffisent.
Pour coder régulièrement : GitHub Copilot est le plus efficace.

3.6.6 Impact sociétal et économique : entre opportunités et risques

Transformation de l'emploi et des compétences

L'IA et l'automatisation transforment profondément le travail. Elles remplacent certaines tâches répétitives, mais créent aussi de nouveaux métiers spécialisés (analyse de données, ingénierie IA, gouvernance numérique).

Cette évolution accentue la polarisation du marché du travail : les emplois très qualifiés et les emplois peu qualifiés résistent mieux que les postes intermédiaires. Dans ce contexte, la formation continue devient indispensable pour suivre le rythme des innovations.

Enjeux éthiques et sociaux

L'essor de l'IA soulève des questions majeures :

- Vie privée : multiplication des dispositifs de surveillance et collecte massive de données.
- Manipulation de l'information : deepfakes, désinformation, influence algorithmique sur les opinions.
- Responsabilité : difficulté à déterminer qui répond des erreurs d'un système autonome.
- Souveraineté numérique : dépendance envers quelques grandes entreprises mondiales.

Éducation et accès au savoir

L'IA modifie les pratiques éducatives :

- Personnalisation des apprentissages grâce aux plateformes adaptatives.
- Démocratisation du savoir via les MOOC et les outils génératifs.

  Les MOOC (Massive Open Online Courses) offrent un accès libre ou peu coûteux à des cours universitaires en ligne. Ils permettent d'apprendre à son rythme, de se former tout au long de la vie et de découvrir de nouvelles compétences, mais exigent autonomie, motivation et esprit critique.
- Nouveaux risques : triche facilitée, dépendance technologique, circulation de contenus erronés.

Applications et impacts sectoriels

L'IA s'impose dans de nombreux domaines :

- Santé : aide au diagnostic, découverte de médicaments, robotique chirurgicale.
- Finance : détection de fraudes, analyse automatisée, scoring de crédit.
- Transport : véhicules autonomes, optimisation logistique.
- Culture et médias : création automatisée de textes, images, musiques et vidéos.
- Science : accélération de la recherche grâce aux simulations et à l'analyse de données massives.

Défis et limites

Malgré ses avancées, l'IA présente des limites importantes :

- Biais et discriminations liés aux données d'entraînement.
- Manque d'explicabilité des modèles complexes, frein à la confiance et à la régulation.
- Impact environnemental élevé de l'entraînement des grands modèles.
- Régulation : nécessité d'encadrer les usages sans freiner l'innovation.

3.6.7 Informatique quantique et cryptographie : Une nouvelle frontière

L'informatique quantique : Promesses et menaces

Les ordinateurs quantiques exploitent les propriétés de la mécanique quantique (superposition, intrication) pour résoudre des problèmes inaccessibles aux ordinateurs classiques :

- Suprématie quantique : en 2019, Google annonce que son processeur Sycamore (54 qubits) a résolu un problème en 200 secondes, là où un supercalculateur mettrait des millénaires.
- Applications potentielles : optimisation complexe, simulation de molécules (chimie, pharmacologie), cryptanalyse.

La cryptographie post-quantique

L'arrivée des ordinateurs quantiques menace les systèmes de chiffrement actuels (RSA, ECC), basés sur la difficulté de factoriser de grands nombres ou de résoudre des logarithmes discrets.

L'algorithme de Shor (1994) permet de casser ces systèmes en temps polynomial sur un ordinateur quantique.

- Cryptographie post-quantique : développement de nouveaux algorithmes résistants aux attaques quantiques (ex. : LWE – Learning With Errors, NTRU).
- Standardisation : le NIST (États-Unis) et l'ETSI (Europe) travaillent à normaliser ces nouveaux standards.
- Transition : les entreprises et États doivent migrer leurs infrastructures avant l'arrivée de ordinateurs quantiques suffisamment puissants (« Q-Day »).
-

3.6.8 Les géants du numérique (GAFAM, BATX) : Pouvoir, régulation et souveraineté

Le saviez-vous ?

GAFAM (Les géants américains du numérique)

- Google (Alphabet Inc.) - Fondé en 1998 - Domaine : Moteur de recherche, publicité en ligne, cloud, systèmes d'exploitation (Android), IA.
- Apple - Fondé en 1976 - Domaine : Matériel (iPhone, Mac, iPad), logiciels (iOS, macOS), services (Apple Music, iCloud).
- Facebook (Meta Platforms, Inc.) - Fondé en 2004 - Domaine : Réseaux sociaux (Facebook, Instagram, WhatsApp), réalité virtuelle (Meta Quest), métavers.
- Amazon - Fondé en 1994 - Domaine : Commerce en ligne, cloud computing (AWS), streaming (Prime Video), devices connectés (Alexa, Kindle).
- Microsoft - Fondé en 1975 - Domaine : Logiciels (Windows, Office), cloud (Azure), matériel (Surface, Xbox), IA (Copilot).

BATX (Les géants chinois de la tech)

- Baidu - Fondé en 2000 - Domaine : Moteur de recherche (équivalent chinois de Google), IA, voitures autonomes, cloud.
- Alibaba - Fondé en 1999 - Domaine : Commerce en ligne (Alibaba, Taobao), cloud computing, fintech (Alipay), logistique.

	• Tencent - Fondé en 1998 - Domaine : Réseaux sociaux (WeChat), jeux vidéo, fintech, cloud, divertissement (Tencent Video). • Xiaomi - Fondé en 2010 - Domaine : Électronique grand public (smartphones, objets connectés), IoT, logiciels.
	Résumé • GAFAM domine le marché occidental et mondial avec des services variés (recherche, cloud, matériel, réseaux sociaux). • BATX contrôle le marché chinois et asiatique, avec des équivalents locaux aux services GAFAM (ex : Baidu vs Google, WeChat vs Facebook). • Ces entreprises sont souvent citées pour leur influence économique, leur pouvoir technologique, et leur impact sur la vie privée et la régulation des données.

La domination des GAFAM et des BATX

Les **GAFAM** (Google, Apple, Facebook/Meta, Amazon, Microsoft) et les **BATX** (Baidu, Alibaba, Tencent, Xiaomi) contrôlent une part majeure de l'économie numérique :

- Monopoles : ils dominent les moteurs de recherche, les réseaux sociaux, le e-commerce, le cloud, les systèmes d'exploitation.
- Données : ils collectent et exploitent des quantités colossales de données personnelles, devenant des acteurs clés de la surveillance capitaliste.
- Innovation : ils investissent massivement en R&D (IA, quantique, réalité virtuelle), souvent en rachetant des startups prometteuses.

Régulation et souveraineté numérique

Face à leur puissance, les États tentent de réguler :

- RGPD (Europe, 2018) : encadre la collecte et l'utilisation des données personnelles.
- Loi sur les services numériques (DSA) et loi sur les marchés numériques (DMA) (UE, 2022–2024) : visent à limiter les abus de position dominante et à ouvrir les plateformes.
- Taxation : tentatives de taxer les géants du numérique (ex. : taxe GAFA en France).
- Guerre technologique : tensions entre États-Unis et Chine (restrictions sur les semi-conducteurs, interdiction de TikTok, Huawei).

Enjeux géopolitiques

La maîtrise des technologies numériques est devenue un enjeu de puissance :

- Chine vs États-Unis : course à la suprématie en IA, 5G, quantique.
- Europe : cherche à affirmer sa souveraineté (cloud souverain, IA éthique, régulation stricte).
- Pays en développement : risque de dépendance et de fracture numérique accrue.

3.6.9 Conclusion : Un siècle de ruptures et d'opportunités

Le XXI^{e} siècle est celui de la révolution numérique et de l'IA, marquée par :

- Des avancées technologiques sans précédent : deep learning, informatique quantique, IA générative.
- Une transformation profonde des sociétés : emploi, éducation, santé, culture, géopolitique.
- Des défis majeurs : éthique, régulation, souveraineté, environnement.

Cette ère ouvre des perspectives immenses, mais impose aussi une réflexion collective sur l'avenir que nous voulons construire. Les choix technologiques, économiques et politiques d'aujourd'hui détermineront le monde de demain.

4 Algorithmes et programmation : fondements théoriques

Programmer un algorithme revient à l'exprimer dans un langage compréhensible par une machine. Pour cela, il faut connaître les principes des langages de programmation et les notions théoriques qui définissent ce qu'une machine peut ou ne peut pas résoudre. Ce chapitre présente ces bases, depuis la machine de Turing jusqu'aux langages modernes, et explique pourquoi Python est aujourd'hui un outil privilégié pour l'apprentissage.

On y présente les bases théoriques qui relient les algorithmes, les langages de programmation et les modèles de calcul. Il montre comment les idées abstraites se traduisent en instructions concrètes, et comment la théorie permet de déterminer ce qu'une machine peut ou ne peut pas résoudre.

4.1 Qu'est-ce qu'un langage de programmation ?

Un langage de programmation est un moyen d'exprimer un algorithme de manière précise et exécutable. Il impose une syntaxe, des règles et des structures qui permettent de décrire les opérations à effectuer. Contrairement au pseudocode, qui reste informel, un langage de programmation doit être interprété ou compilé pour produire un résultat.

> **Complié ou interprété ?**
> **Un langage compilé, c'est comme écrire un discours que l'on remet à un orateur : le programme est d'abord traduit en bloc dans la langue de la machine, puis exécuté à toute vitesse.**
> **À l'inverse, un langage interprété ressemble à une conversation avec un traducteur simultané : chaque instruction est lue et exécutée au fur et à mesure, sans étape préalable.**
> **Le premier mise sur la performance, le second sur la souplesse — deux styles pour dialoguer avec le même ordinateur.**

Chaque langage repose sur un ensemble de concepts fondamentaux : variables, types, opérations, structures de contrôle. Ces éléments permettent de manipuler des données, de prendre des décisions et de répéter des actions. Les langages diffèrent par leur style, leur niveau d'abstraction et leurs domaines d'application, mais tous ont pour objectif de traduire une méthode en instructions compréhensibles par une machine.

L'apparition des langages de programmation a profondément transformé la manière de concevoir les algorithmes. Ils offrent un cadre rigoureux qui facilite l'écriture, la vérification et la maintenance des programmes. Ils permettent aussi de séparer la logique du traitement des détails techniques liés au matériel.

4.2 Les grandes familles de langages

Les langages de programmation peuvent être regroupés en plusieurs familles selon leur manière d'exprimer les algorithmes.

- **Les langages impératifs** décrivent les étapes successives d'un traitement, comme on le ferait dans une recette.
- **Les langages fonctionnels** privilégient les transformations de données sans modifier l'état du programme.
- **Les langages logiques** reposent sur des règles et des relations plutôt que sur des instructions.
- **Les langages orientés objet** organisent les programmes autour d'entités qui regroupent données et comportements.

Ces approches reflètent différentes façons de penser un algorithme. Certaines sont adaptées aux calculs scientifiques, d'autres à la modélisation de systèmes complexes ou à la manipulation de

données. Comprendre ces familles permet de choisir le langage le plus approprié pour un problème donné et d'adapter la manière de concevoir les solutions.

4.3 Décidabilité et limites du calcul

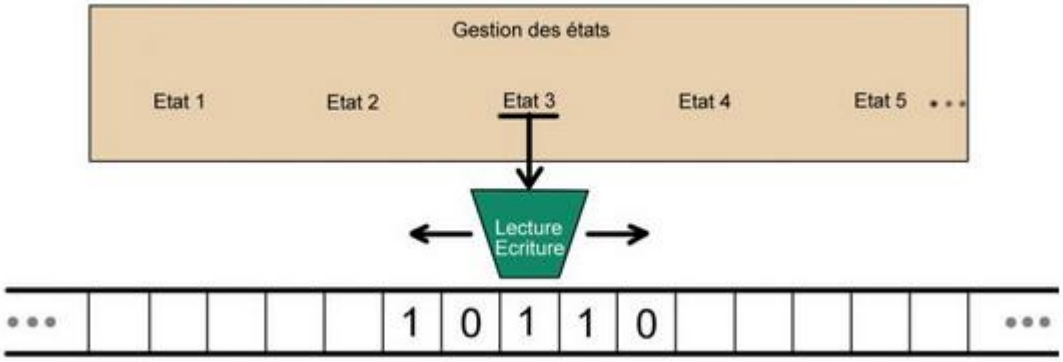
La théorie des algorithmes ne se limite pas à leur écriture. Elle cherche aussi à déterminer quels problèmes peuvent être résolus par une machine. Cette question a été étudiée au XX^{e} siècle par des chercheurs comme Alan Turing et Alonzo Church. Ils ont montré que certains problèmes sont indécidables : aucune méthode générale ne permet de les résoudre dans tous les cas.

La décidabilité désigne la capacité d'un algorithme à donner une réponse correcte pour toutes les entrées possibles. Lorsqu'un problème est indécidable, il n'existe pas d'algorithme universel pour le traiter. Cette limite n'est pas liée à la puissance des machines actuelles, mais à la nature même du problème. Elle montre que les algorithmes, aussi puissants soient-ils, ne peuvent pas tout résoudre.

Cette notion de décidabilité est essentielle pour comprendre les frontières du calcul. Elle permet d'éviter de chercher des solutions impossibles et d'orienter les efforts vers des approches approximatives ou spécialisées lorsque la résolution exacte est hors de portée.

4.4 La machine de Turing : un modèle pour comprendre les algorithmes

Pour étudier les capacités des machines, Alan Turing a proposé un modèle abstrait appelé machine de Turing. Il s'agit d'un dispositif théorique composé d'un ruban infini, d'une tête de lecture et d'un ensemble de règles. Malgré sa simplicité, ce modèle est capable de simuler n'importe quel algorithme. Il sert de référence pour définir ce qu'est un calcul effectif.



Source image : Le Monde – 2017

La machine de Turing n'est pas un ordinateur réel, mais elle en capture l'essentiel : la capacité à lire des données, à modifier un état interne et à appliquer des instructions. Elle permet de raisonner sur les algorithmes sans dépendre d'une technologie particulière. Elle est également utilisée pour démontrer l'existence de problèmes indécidables.

Ce modèle a joué un rôle fondamental dans la naissance de l'informatique. Il a permis de formaliser la notion d'algorithme et de comprendre les limites du calcul. Aujourd'hui encore, il sert de base à de nombreux cours d'informatique théorique.

4.5 Python : un langage moderne pour exprimer les algorithmes

Parmi les nombreux langages de programmation, Python occupe une place particulière. Il est conçu pour être simple à lire et à écrire, ce qui en fait un outil idéal pour apprendre la programmation et pour exprimer des algorithmes de manière claire. Sa syntaxe épurée met en avant la logique plutôt que les détails techniques.

Python est utilisé dans des domaines très variés : sciences, web, intelligence artificielle, analyse de données. Cette polyvalence en fait un langage de référence pour l'enseignement. Il permet de

passer rapidement de la description d'un algorithme à son implémentation, tout en offrant des bibliothèques puissantes pour aller plus loin.

Dans cet ouvrage, Python servira de support pour illustrer les concepts et pour montrer comment les algorithmes prennent forme dans un programme réel. Il ne s'agit pas d'un manuel de Python, mais d'un outil pour rendre les idées plus concrètes.

La NASA, Python… et la légende urbaine des fusées codées en script

On lit souvent que « la NASA envoie des fusées dans l'espace avec Python ». C'est faux — et la réalité est bien plus intéressante.

Les systèmes embarqués qui pilotent les fusées utilisent des langages certifiés, déterministes et très proches de la machine, comme Ada, C ou parfois C++. Ces langages sont choisis pour deux raisons essentielles :

1. La sécurité : ils permettent une vérification formelle et un contrôle strict de la mémoire.
2. La performance : ils produisent du code extrêmement rapide, indispensable pour les calculs temps réel (guidage, navigation, contrôle moteur).

Python, lui, est un langage interprété, non déterministe en termes de timing, et trop lent pour des systèmes embarqués critiques. Il n'est pas conçu pour gérer des contraintes temps réel où chaque microseconde compte.

En revanche, Python est omniprésent dans tout ce qui entoure les missions spatiales : analyse des données des télescopes, automatisation des pipelines scientifiques, simulation de trajectoires, traitement d'images, pilotage de robots de test, prototypage rapide.

C'est devenu l'outil quotidien des ingénieurs et des chercheurs.

L'ironie, c'est que cette omniprésence a fini par créer un mythe : puisque Python est partout dans les laboratoires, certains ont imaginé qu'il pilotait aussi les fusées. La réalité est plus subtile : Python prépare les missions, mais ce sont Ada et C qui tiennent le manche au moment du décollage.

Un bel exemple de la manière dont un langage simple a envahi les coulisses de l'exploration spatiale… sans jamais toucher au bouton « ignition ».

4.6 Conclusion du chapitre

Ce chapitre a présenté les fondements théoriques qui relient les algorithmes aux langages de programmation. Il a montré comment les méthodes abstraites se traduisent en instructions exécutables, comment les langages structurent cette traduction et quelles sont les limites imposées par la théorie du calcul. Il a également introduit Python comme langage d'étude privilégié.

Les chapitres suivants aborderont l'implémentation concrète des algorithmes, les structures de données et les techniques qui permettent de construire des programmes efficaces et fiables.

5 Implémentation : construire un algorithme dans un langage

Une fois les principes posés, il faut savoir traduire un algorithme en instructions exécutables. Ce chapitre explore les éléments essentiels de la programmation : logique, variables, structures de contrôle, données, fonctions et objets. Il montre comment organiser un programme, manipuler des structures adaptées et gérer les données de manière fiable. L'objectif est de comprendre comment un algorithme prend forme dans un code réel.

Un algorithme n'existe réellement que lorsqu'il peut être exécuté. Pour cela, il doit être traduit dans un langage de programmation et manipulé à l'aide de structures adaptées. L'implémentation est l'étape où la logique abstraite devient un ensemble d'instructions précises, capables de transformer des données et de produire un résultat.

Ce chapitre présente les éléments essentiels de cette transformation : la logique, les variables, les structures de contrôle, les données, les fonctions et les objets. Il montre comment ces composants s'organisent pour former un programme cohérent et fiable.

5.1 La logique de Boole : le fondement des décisions

Toute machine informatique repose sur la logique de Boole. Cette logique ne connaît que deux valeurs : vrai et faux. Elle permet de formuler des conditions, de comparer des données et de guider l'exécution d'un programme. Les opérateurs logiques — comme « et », « ou » ou « non » — permettent de combiner des tests pour prendre des décisions plus complexes.

Dans un programme, la logique de Boole intervient à chaque fois qu'il faut choisir entre plusieurs actions. Elle permet de vérifier si une valeur dépasse un seuil, si une liste contient un élément ou si une opération doit être répétée. Sans cette logique, un programme ne pourrait pas s'adapter aux données qu'il reçoit. Elle constitue la base de toutes les structures de contrôle.

5.2 Variables et types : stocker et manipuler les données

Pour exécuter un algorithme, il faut pouvoir stocker des informations. Les variables jouent ce rôle. Elles permettent de conserver des valeurs, de les modifier et de les transmettre d'une étape à l'autre. Chaque variable possède un type, qui indique la nature de la donnée : nombre, texte, booléen, liste, etc.

Les types déterminent les opérations possibles. On ne traite pas un texte comme un nombre, ni une liste comme une image. Cette distinction garantit la cohérence du programme et évite les erreurs. Dans certains langages, les types doivent être déclarés explicitement ; dans d'autres, comme Python, ils sont déterminés automatiquement. Dans tous les cas, comprendre les types est indispensable pour manipuler correctement les données.

5.3 Structures de contrôle : décisions et répétitions

Un programme ne se contente pas d'exécuter des instructions les unes après les autres. Il doit pouvoir choisir une action en fonction d'une condition et répéter une opération tant qu'un critère n'est pas rempli. Les structures de contrôle permettent ces comportements.

Les instructions conditionnelles permettent de sélectionner un chemin d'exécution. Elles reposent sur la logique de Boole et permettent d'adapter le programme aux données. Les boucles, quant à elles, permettent de répéter une action un nombre déterminé de fois ou jusqu'à ce qu'une condition soit satisfaite. Elles sont essentielles pour parcourir des listes, traiter des fichiers ou effectuer des calculs successifs.

Ces structures donnent au programme sa flexibilité. Elles permettent de décrire des comportements complexes à partir d'instructions simples.

5.4 Listes, tableaux et structures de données

La plupart des algorithmes manipulent des collections d'informations. Les listes et les tableaux permettent de regrouper plusieurs valeurs dans une même structure. Ils facilitent le tri, la recherche, le filtrage et de nombreuses autres opérations.

D'autres structures de données existent pour répondre à des besoins spécifiques. Les dictionnaires associent des clés à des valeurs. Les piles et les files organisent les données selon un ordre particulier. Les arbres et les graphes permettent de représenter des relations complexes. Le choix de la structure de données influence fortement l'efficacité d'un programme.

Comprendre ces structures est essentiel pour implémenter des algorithmes de manière efficace. Elles permettent de stocker, organiser et retrouver les informations de manière adaptée au problème.

5.5 Fonctions, modules et sous-programmes

Pour éviter les répétitions et organiser efficacement le code, il est utile de regrouper certaines instructions dans des fonctions. Une fonction réalise une tâche précise et peut être réutilisée dans différents contextes. Elle simplifie le programme, améliore sa lisibilité et facilite sa maintenance.

Les modules rassemblent plusieurs fonctions liées entre elles. Ils permettent de structurer un programme en parties cohérentes et de partager des outils entre différents projets. Dans les programmes de grande taille, cette organisation modulaire devient indispensable.

Ces sous-programmes — fonctions et modules — sont souvent regroupés dans des bibliothèques (ou packages). Il est alors possible de les importer dans un programme principal uniquement lorsqu'on en a besoin, ce qui évite de réécrire du code déjà existant et encourage la réutilisation.

Enfin, cette structuration facilite aussi les tests : chaque partie du code peut être vérifiée séparément, ce qui accélère la détection des erreurs et l'amélioration progressive du programme.

5.6 Programmation orientée objet : organiser les données et les comportements

La programmation orientée objet propose une manière différente d'organiser un programme. Elle regroupe les données et les fonctions qui les manipulent dans des entités appelées objets. Chaque objet possède un état et des comportements. Cette approche permet de représenter des situations complexes de manière naturelle.

Les classes définissent la structure des objets. Elles permettent de créer plusieurs instances partageant les mêmes caractéristiques. Cette organisation facilite la réutilisation du code et la modélisation de systèmes réels. Elle est largement utilisée dans les applications modernes, notamment dans les interfaces graphiques, les jeux vidéo et les logiciels de gestion.

Le saviez-vous ?

La programmation objet

Elle consiste à représenter un élément du monde réel sous forme d'objet.

Par exemple, on crée un objet **voiture** avec des caractéristiques (couleur, vitesse) et des actions (démarrer, freiner).

Ensuite, on peut fabriquer autant de voitures qu'on veut, toutes basées sur ce même modèle.

En réalité, la programmation objet, c'est simplement définir un modèle, puis créer des objets qui se comportent comme dans la vraie vie.

Exemple simple de formalisation en Python

On veut représenter une **voiture** dans un programme.

On formalise cet objet en décrivant :

- **ses attributs** (ce qu'elle *est*) : couleur, marque, vitesse
- **ses méthodes** (ce qu'elle *fait*) : démarrer, accélérer, freiner

```
class Voiture:
    def __init__(self, couleur, marque):
        self.couleur = couleur
        self.marque = marque
        self.vitesse = 0

    def accelerer(self, valeur):
        self.vitesse += valeur

    def freiner(self):
        self.vitesse = 0
```

Ensuite, on peut créer autant de voitures qu'on veut :

```
ma_voiture = Voiture("rouge", "Renault")
ma_voiture.accelerer(20)
```

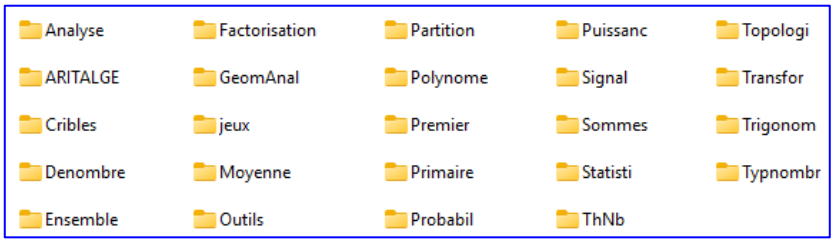
Bilan : formaliser un objet, c'est simplement définir un modèle (classe – premier cadre noir) puis créer des instances (objets – autant de cadres noirs comme le second).

C'est donc créer des objets qui se comportent comme dans la réalité.

5.7 Fichiers, mémoire et gestion des données

Un programme doit souvent lire ou écrire des informations dans des fichiers. Cette capacité permet de conserver des données entre deux exécutions, de traiter des documents ou d'échanger des informations avec d'autres logiciels.

La gestion des fichiers nécessite une attention particulière, car elle implique des opérations lentes et sensibles aux erreurs.



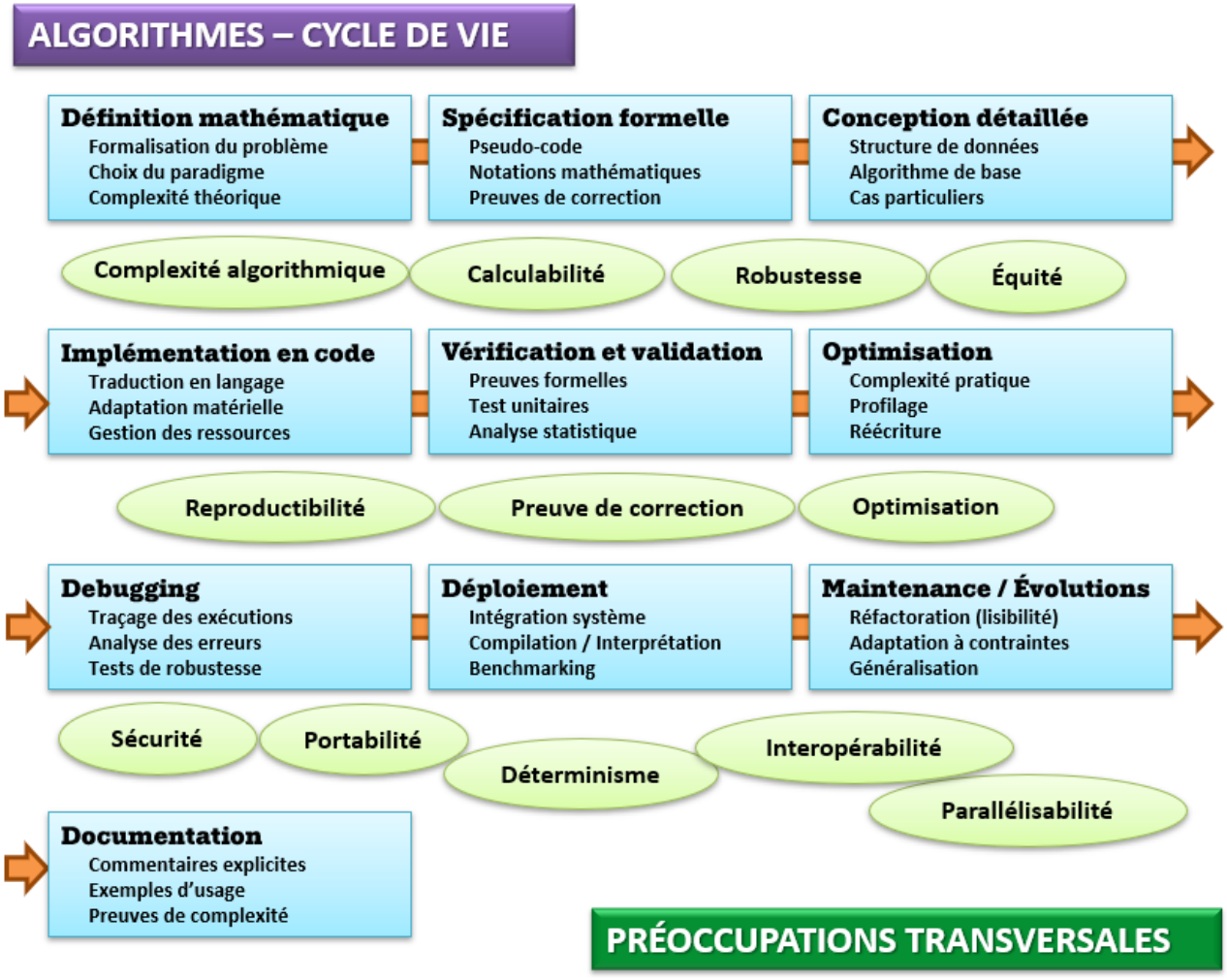
Un exemple de fichier « mathématiques » et de ses sous-fichiers

La mémoire joue également un rôle important. Chaque variable, chaque structure de données et chaque objet occupe un espace en mémoire. Une mauvaise gestion peut entraîner des ralentissements ou des dysfonctionnements. Les pointeurs, utilisés dans certains langages, permettent de manipuler directement les adresses mémoire. Ils offrent une grande flexibilité, mais demandent une rigueur particulière.

Comprendre ces aspects permet d'écrire des programmes plus robustes et plus efficaces.

5.8 Cycle de vie et préoccupations transversales

Carte synthétique



Préoccupations transversales

Préoccupation	Détails	Exemples
Complexité algorithmique	emps (Grand-O) et espace (mémoire).	*Tri fusion : O(n log n) en temps, O(n) en espace.*
Calculabilité	Le problème est-il accessible par un algorithme ?	*Problème de l'arrêt (non calculable), tri possible (calculable).*

Préoccupation	Détails	Exemples
Preuves de correction	L'algorithme produit-il toujours le résultat attendu ?	*Preuve que le tri à bulles termine et trie correctement.*
Robustesse	Comportement avec des entrées invalides ou extrêmes.	*Gestion des overflows (ex : INT_MAX + 1).*
Optimisation	éduction des ressources (temps, mémoire, énergie).	*Remplacer une recherche linéaire par une recherche dichotomique O(log n).*
Portabilité	Compatibilité entre langages/architectures.	*Éviter les dépendances spécifiques à un OS (ex : fork() sous Unix).*
Sécurité	rotection contre les exploits (ex : buffer overflow).	*Vérification des bornes des tableaux.*
Déterminisme	L'algorithme produit-il toujours le même résultat pour une entrée donnée ?	*lgorithmes randomisés (ex : QuickSort) vs déterministes.*
Parallélisabilité	Capacité à s'exécuter sur plusieurs cœurs/GPU.	*Tri parallèle avec OpenMP.*
Équité	bsence de biais (ex : algorithmes de recrutement).	*Vérification que l'algorithme ne favorise pas un groupe spécifique.*
Reproductibilité	ossibilité de reproduire les résultats (ex : avec une graine aléatoire fixe).	*Utilisation de seed pour les générateurs de nombres aléatoires.*
Interprétabilité	Compréhension du fonctionnement par un humain.	*Éviter les "boîtes noires" (ex : réseaux de neurones profonds).*

5.9 Conclusion du chapitre

Ce chapitre a présenté les éléments essentiels de l'implémentation d'un algorithme. Il a montré comment la logique, les variables, les structures de contrôle, les données, les fonctions et les objets s'organisent pour former un programme complet. Il a également introduit les aspects liés aux fichiers et à la mémoire, indispensables pour traiter des données réelles.

Les chapitres suivants aborderont la complexité algorithmique, l'explicabilité et la description de nombreux algorithmes concrets, du plus simple au plus avancé.

6 Complexité algorithmique : mesurer l'efficacité

Tous les algorithmes ne se valent pas. Certains résolvent un problème rapidement, d'autres deviennent inutilisables dès que les données augmentent. La complexité algorithmique permet d'évaluer cette efficacité en étudiant la croissance du temps d'exécution ou de la mémoire utilisée.

Lorsqu'un algorithme est conçu, il ne suffit pas de vérifier qu'il donne le bon résultat. Il faut aussi s'assurer qu'il le produit dans un délai raisonnable et avec une quantité de mémoire adaptée. Deux algorithmes peuvent résoudre exactement le même problème, mais l'un peut être beaucoup plus rapide que l'autre dès que la quantité de données augmente.

La complexité algorithmique permet d'évaluer cette efficacité. Elle décrit la manière dont le temps d'exécution ou l'espace mémoire évolue en fonction de la taille des données.

Ce chapitre présente les principes essentiels de cette analyse et montre pourquoi elle est indispensable dans la pratique.

6.1 Pourquoi mesurer la complexité ?

Un algorithme peut sembler performant lorsqu'il est testé sur quelques données, mais devenir inutilisable lorsque la taille du problème augmente. Cette situation est fréquente dans les domaines où les volumes d'information sont importants : recherche sur Internet, traitement d'images, analyse de données, cryptographie. La complexité algorithmique permet d'anticiper ces difficultés en étudiant la croissance du temps d'exécution.

Cette analyse ne dépend pas d'un ordinateur particulier. Elle s'intéresse à la tendance générale : comment le temps augmente-t-il lorsque la quantité de données double, triple ou se multiplie par mille. Cette approche abstraite permet de comparer des algorithmes entre eux et de choisir celui qui sera le plus adapté à un usage réel.

6.2 La notation Grand-O : une mesure asymptotique

Pour exprimer la complexité, on utilise la notation Grand-O (ou O majuscule). Elle décrit la manière dont le temps d'exécution évolue lorsque la taille des données devient très grande. Elle ne cherche pas à mesurer le temps exact, mais à identifier l'ordre de grandeur de la croissance.

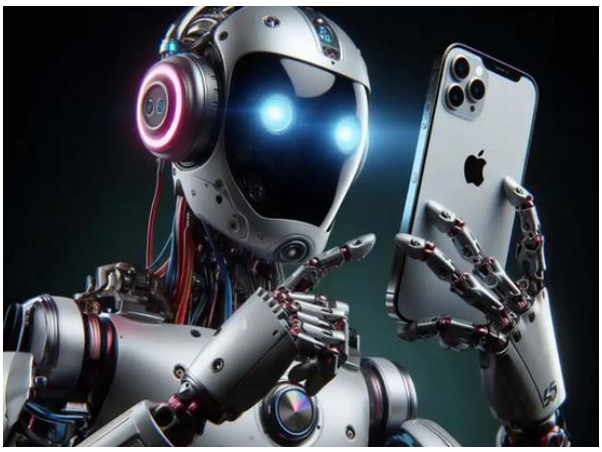
Par exemple, un algorithme en O(n) voit son temps d'exécution augmenter proportionnellement à la taille des données. Un algorithme en $O(n^2)$ devient beaucoup plus lent, car le temps croît comme le carré de la taille du problème. Un algorithme en O(log n) reste très efficace même pour des données volumineuses.

Cette notation permet de simplifier l'analyse en se concentrant sur les éléments qui influencent réellement la performance. Les détails secondaires, comme les constantes ou les optimisations mineures, sont ignorés pour mettre en avant la structure générale de l'algorithme.

6.3 Complexité temporelle et complexité spatiale

La complexité algorithmique ne concerne pas seulement le temps d'exécution. Elle s'intéresse aussi à la mémoire utilisée.

Certains algorithmes sont rapides mais consomment beaucoup d'espace. D'autres utilisent peu de mémoire mais nécessitent un temps important. Le choix dépend du contexte : un téléphone portable n'a pas les mêmes contraintes qu'un supercalculateur.



La complexité temporelle mesure le nombre d'opérations nécessaires pour résoudre un problème. La complexité spatiale mesure la quantité de mémoire supplémentaire utilisée pendant l'exécution. Ces deux aspects sont souvent liés, mais ils peuvent évoluer différemment selon les méthodes employées.

Comprendre ces deux dimensions permet de concevoir des algorithmes équilibrés, adaptés aux ressources disponibles.

6.4 Exemples de comportements de croissance

Les différentes classes de complexité correspondent à des comportements de croissance très différents.

Un algorithme constant, en O(1), reste rapide quelle que soit la taille des données. Un algorithme logarithmique, en O(log n), progresse lentement et reste efficace même pour des millions d'éléments. Les algorithmes linéaires, en O(n), parcourent toutes les données une fois.

Les algorithmes quadratiques, en $O(n^2)$, deviennent rapidement difficiles à utiliser lorsque les données augmentent. Ils conviennent pour de petites tailles, mais pas pour des volumes importants. Les algorithmes exponentiels ou factoriels, en $O(2^n)$ ou O(n!), sont pratiquement inutilisables dès que le problème dépasse quelques dizaines d'éléments. Ils illustrent les limites de ce que l'on peut résoudre de manière exacte.

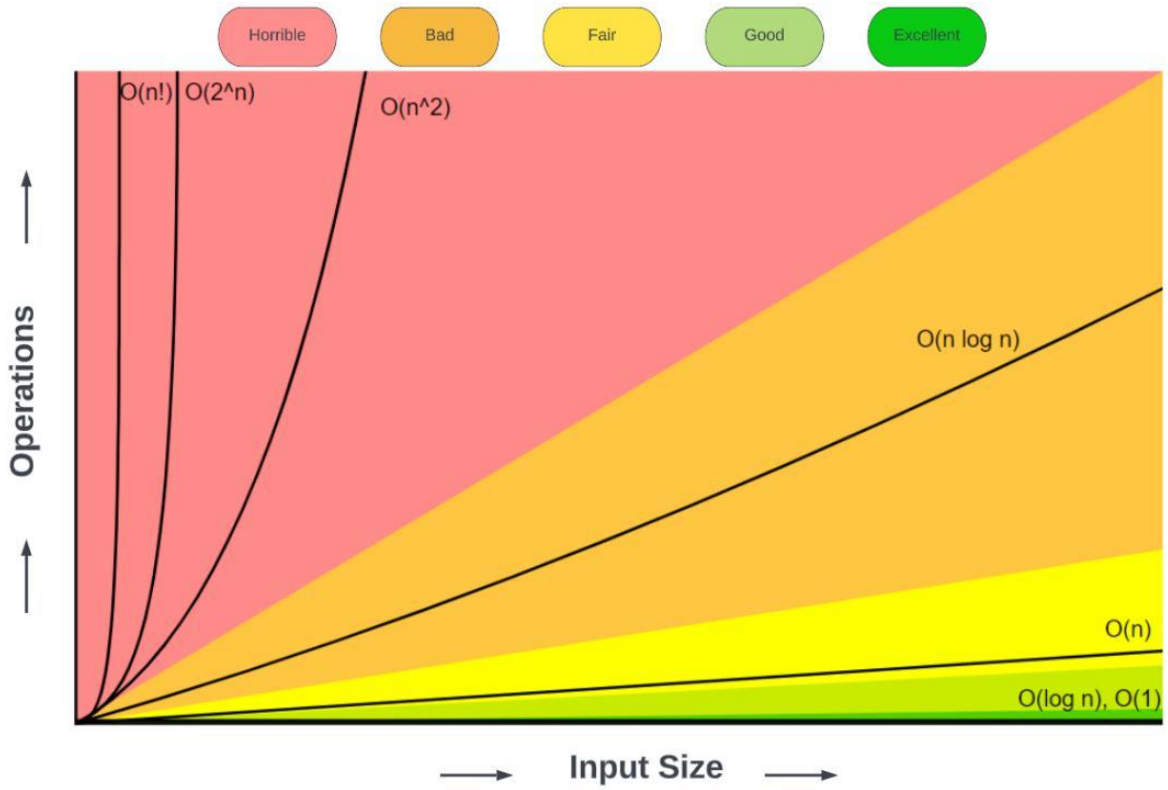
Ces différences montrent l'importance de choisir une méthode adaptée. Un mauvais choix peut rendre un programme inutilisable, même sur une machine puissante.

Tableau de complexité algorithmique

Complexité	Temps pour n = 1 000	Métaphore visuelle	Idée intuitive
O(1)	≈ 1 ns	Ligne horizontale	Le temps ne change pas, même avec des millions d'éléments. **O(1)** : ressemble à une ligne parfaitement horizontale — l'algorithme ne bronche pas, même si on lui envoie des millions d'éléments.
O(log n)	≈ 10 ns	Pente douce	Quand les données doublent, le temps n'augmente presque pas. **O(log n)** : monte doucement, comme une pente de colline régulière — chaque fois que les données doublent, la charge n'augmente que d'un petit cran.
O(n)	≈ 1 µs	Diagonale régulière	Deux fois plus de données → deux fois plus de temps. **O(n)** : trace une diagonale simple — plus il y a de données, plus le temps augmente proportionnellement.

Complexité	Temps pour n = 1 000	Métaphore visuelle	Idée intuitive
O(n log n)	≈ 10 µs	Courbe légèrement bombée	Raisonnable, mais la courbe finit par se sentir. **O(n log n)** : dessine une courbe légèrement bombée — raisonnable, mais qui finit par se faire sentir.
$O(n^2)$	≈ 1 ms	Montée raide	Chaque augmentation de données devient douloureuse. **$O(n^2)$** : grimpe comme une montagne — chaque augmentation de données devient douloureuse.
$O(2^n)$	≈ 10^{30} ans pour n = 100	Fusée verticale	Quelques éléments suffisent à rendre l'algorithme inutilisable. **$O(2^n)$** : explose comme une fusée — quelques éléments suffisent à rendre l'algorithme impraticable.
O(n!)	≈ 10^{56} ans pour n = 20	Mur vertical	Au-delà de 10–12 éléments, c'est terminé. **O(n!)** : un mur vertical — au-delà de 10 ou 12 éléments, c'est fini.

Rapidité de croissance de la complexité selon le type de complexité



6.5 Complexité P et NP

En algorithmique, on classe les problèmes selon la rapidité avec laquelle un ordinateur peut les résoudre. La classe P regroupe les problèmes pour lesquels on connaît des algorithmes efficaces, c'est-à-dire dont le temps d'exécution croît de manière polynomiale : (O(n)), (O(n2)), ou plus généralement (O(nk)). Par exemple, trier une liste ou calculer le plus court chemin dans un graphe

(algorithme de Dijkstra) sont des problèmes en P : même si les données augmentent, le temps de calcul reste maîtrisable.

La classe NP, elle, rassemble les problèmes pour lesquels on ne connaît pas d'algorithmes rapides, mais dont une solution peut être vérifiée rapidement (en temps polynomial). Trouver une solution peut demander d'explorer un nombre gigantesque de possibilités — souvent de complexité (O(2n)) ou (O(n!)) — mais vérifier qu'une solution proposée est correcte est facile. Le problème du voyageur de commerce (TSP) en est un exemple emblématique.

Au sein de NP, on distingue les problèmes NP-complets : ce sont les problèmes les plus difficiles de NP, ceux auxquels tous les autres problèmes NP peuvent se ramener. Autrement dit, si l'on trouvait un algorithme rapide pour un seul problème NP-complet, tous les problèmes NP deviendraient faciles. C'est l'un des enjeux centraux de la théorie de la complexité, résumé par la célèbre question : P = NP ?

La relation avec la notation en O(...) est directe :

- P correspond aux complexités « raisonnables » : O(n), O(nlog n), O(n^2).
- NP et surtout NP-complet regroupent des problèmes pour lesquels les meilleurs algorithmes connus ont des croissances « explosives » : O(2n), O(n!).

En résumé :

P = problèmes résolus rapidement ;
NP = problèmes vérifiables rapidement ;
NP-complet = les problèmes les plus difficiles de NP.

(Note : certains problèmes encore plus complexes sont dits NP-difficiles ou PSPACE-difficiles, mais ces notions dépassent le cadre de ce document.)

Le saviez-vous ?

Exemples de problèmes NP

- **Itinéraire du voyageur de commerce (TSP)**
  Trouver le meilleur trajet parmi toutes les permutations possibles : la solution est longue à chercher, mais facile à vérifier.
- **Tournées de livraison**
  Même logique que le TSP : organiser plusieurs trajets optimaux devient vite explosif.
- **Tableau de service d'un hôpital**
  Assigner les soignants en respectant toutes les contraintes est un casse-tête combinatoire.
- **Emploi du temps d'un lycée**
  Trouver une organisation sans conflits d'horaires demande d'explorer énormément de possibilités.
- **Jeux combinatoires (Échecs, Go…)**
  Vérifier un coup gagnant est simple ; le trouver parmi des millions d'options l'est beaucoup moins.
- **Jeu Candy Crush**
  Déterminer si l'on peut atteindre un score donné est un problème démontré NP-complet.
- **Cryptage bancaire**
  La sécurité repose sur des problèmes mathématiques très difficiles à résoudre, mais faciles à vérifier une fois la solution donnée.
- **Repliement des protéines**
  Trouver la forme la plus stable parmi un nombre astronomique de configurations possibles.

6.6 Influence de la structure des données

La complexité dépend souvent de la manière dont les données sont organisées. Une recherche dans une liste non triée nécessite de parcourir tous les éléments, alors qu'une recherche dans une structure ordonnée peut être beaucoup plus rapide. De même, certaines structures de données permettent d'insérer ou de supprimer des éléments en temps constant, tandis que d'autres nécessitent un réagencement complet.

Le choix de la structure de données est donc aussi important que le choix de l'algorithme. Il peut transformer un traitement lent en un traitement efficace. Cette idée sera approfondie dans les chapitres consacrés aux algorithmes concrets.

6.7 Limites pratiques et explosion combinatoire

Même avec des machines puissantes, certains problèmes restent difficiles à résoudre. L'explosion combinatoire en est un exemple. Lorsqu'un problème nécessite d'examiner toutes les combinaisons possibles, le nombre de cas augmente tellement vite qu'aucun ordinateur ne peut les traiter. C'est le cas de nombreux problèmes d'optimisation, de planification ou de cryptanalyse.

Découverte

Un exemple simple d'explosion combinatoire

L'explosion combinatoire apparaît dès que l'on tente d'énumérer toutes les possibilités d'un système qui se complexifie légèrement. Un exemple classique suffit à montrer à quel point cette croissance devient rapidement incontrôlable. Imaginons un cadenas très simple, composé de trois molettes, chacune portant dix chiffres, de 0 à 9. À première vue, ce dispositif semble modeste. Pourtant, il offre déjà mille combinaisons possibles. Si l'on ajoute une quatrième molette, le nombre de possibilités passe à dix mille. Une cinquième, et l'on atteint cent mille. Chaque molette supplémentaire multiplie par dix l'espace des solutions, sans que le cadenas ne paraisse visuellement plus complexe.

Ce phénomène devient encore plus frappant lorsqu'on remplace les chiffres par des objets plus variés. Si l'on imagine un mot de cinq lettres, et que chaque position peut accueillir l'une des vingt-six lettres de l'alphabet, le nombre de combinaisons possibles dépasse déjà les onze millions. Avec six lettres, on franchit les trois cents millions. Avec dix lettres, on atteint un nombre si grand qu'il devient impossible à explorer exhaustivement, même pour un ordinateur rapide. Le simple fait d'ajouter une position ou d'élargir le choix des symboles transforme un problème trivial en un espace gigantesque, où la recherche exhaustive devient impraticable.

L'explosion combinatoire ne se limite pas aux cadenas ou aux mots. Elle apparaît dans les jeux, dans les réseaux, dans les systèmes biologiques, dans les algorithmes de planification. Le jeu d'échecs en est un exemple célèbre : après seulement trois coups de chaque côté, le nombre de positions possibles dépasse les neuf millions. Après quatre coups, il franchit les cent millions. Après dix coups, il atteint un nombre astronomique, bien au-delà de ce qu'un humain pourrait explorer mentalement. C'est cette croissance vertigineuse qui rend le jeu si riche, mais aussi si difficile à maîtriser.

Ce phénomène explique pourquoi certaines tâches, pourtant simples à décrire, deviennent rapidement hors de portée des méthodes naïves. Dès que l'on tente d'explorer toutes les possibilités, le temps nécessaire explose. L'explosion combinatoire est ainsi l'un des obstacles fondamentaux de l'informatique et de l'intelligence artificielle. Elle rappelle que la complexité ne vient pas toujours de la nature du problème, mais souvent du nombre de chemins possibles pour le résoudre. Elle montre aussi pourquoi les algorithmes doivent apprendre à éviter les explorations exhaustives, et pourquoi le raisonnement, l'abstraction et l'heuristique sont indispensables pour naviguer dans des espaces où le nombre de possibilités dépasse l'imagination.

La complexité algorithmique permet d'identifier les situations où un problème devient trop long à résoudre exactement. Dans ces cas, on adopte des stratégies alternatives qui ne garantissent pas toujours la solution parfaite, mais qui offrent des résultats suffisamment bons en un temps raisonnable :

- **Approximations**
  On utilise des algorithmes qui ne donnent pas forcément la solution optimale, mais une solution proche du meilleur possible. Par exemple, pour le voyageur de commerce, on peut trouver un trajet « pas trop long » sans explorer toutes les possibilités.
- **Heuristiques**
  Ce sont des méthodes pratiques basées sur l'intuition, des règles simples ou l'expérience. Elles ne garantissent rien théoriquement, mais fonctionnent très bien dans la plupart des cas réels. Par exemple, « toujours choisir la prochaine ville la plus proche » est une heuristique classique.
- **Algorithmes probabilistes**
  Ils introduisent du hasard dans le calcul pour explorer plus efficacement l'espace des solutions. Ils ne donnent pas toujours la même réponse, mais convergent souvent vers une bonne solution. Les algorithmes génétiques ou le recuit simulé en sont des exemples.

6.8 Conclusion du chapitre

La complexité algorithmique est un outil essentiel pour comprendre l'efficacité d'un algorithme. Elle permet d'anticiper les performances, de comparer différentes méthodes et de choisir la plus adaptée à un problème donné. Elle met en évidence les limites du calcul et les défis posés par les problèmes difficiles.

Ce chapitre a présenté les notions fondamentales de cette analyse. Les chapitres suivants aborderont l'explicabilité des algorithmes, puis la description de nombreux algorithmes concrets, du plus simple au plus avancé.

7 Explicabilité et transparence des algorithmes

Les algorithmes interviennent aujourd'hui dans un nombre croissant de décisions qui touchent directement la vie quotidienne : recommandations, diagnostics médicaux, filtrage d'informations, détection de fraudes, gestion de flux ou analyse de données. Leur efficacité repose sur des calculs rapides et précis, mais leur fonctionnement reste souvent invisible pour l'utilisateur.

L'explicabilité algorithmique vise précisément à rendre ces traitements compréhensibles : savoir comment une décision a été prise, quelles données ont été utilisées et quels critères ont influencé le résultat.

À ces enjeux s'ajoute une dimension essentielle : la vérification et les tests de preuve. Plus les algorithmes deviennent complexes — parfois composés de millions de paramètres — plus il est difficile de garantir qu'ils fonctionnent correctement dans toutes les situations. Les erreurs de raisonnement, les biais dans les données ou les comportements inattendus peuvent avoir des conséquences importantes. La vérification algorithmique, qu'elle prenne la forme de tests systématiques, de preuves formelles ou de simulations massives, vise à s'assurer que le système respecte bien les règles attendues et qu'il ne produit pas de résultats aberrants. Elle complète l'explicabilité en apportant une garantie supplémentaire de fiabilité.

7.1 Pourquoi expliquer un algorithme ?

Lorsqu'un algorithme classe des documents, propose un itinéraire ou trie des messages, son fonctionnement peut rester discret sans poser de problème. Mais lorsqu'il intervient dans des domaines sensibles — santé, finance, justice, emploi — il devient essentiel de comprendre comment il agit. Une décision automatisée peut avoir des conséquences importantes, et il est légitime de demander sur quels éléments elle repose.

L'explicabilité permet de :

- **vérifier que l'algorithme fonctionne correctement**,
- **détecter d'éventuels biais** ou discriminations,
- **corriger les erreurs** et améliorer les modèles,
- **renforcer la confiance** des utilisateurs et des institutions.

Elle constitue donc un outil de contrôle, de compréhension et de responsabilisation.

7.2 Algorithmes déterministes et algorithmes apprenants

Tous les algorithmes ne posent pas les mêmes défis en matière d'explicabilité.

Algorithmes déterministes (classiques)

Ils suivent une suite d'instructions clairement définies. Leur fonctionnement peut être décrit étape par étape, ce qui facilite l'analyse.

Exemples : tri, recherche, calculs arithmétiques, règles logiques.

Algorithmes apprenants (IA)

Ils apprennent à partir de données. Leur comportement dépend des exemples reçus et des paramètres ajustés.

Dans les réseaux de neurones, ces paramètres sont très nombreux et difficiles à interpréter. Le modèle peut produire des résultats précis sans que l'on puisse facilement expliquer comment il y est parvenu.

Cette différence crée un contraste entre des algorithmes transparents et des algorithmes opaques. L'explicabilité cherche à réduire cette opacité.

7.3 Vérification, validation et preuves formelles

L'explicabilité ne suffit pas : un algorithme peut être compréhensible tout en étant incorrect. D'où l'importance de la vérification, qui vise à garantir que le système fonctionne comme prévu.

On distingue plusieurs approches :

- **Tests classiques** : on vérifie le comportement de l'algorithme sur de nombreux cas, y compris des cas extrêmes.
- **Validation statistique** : on mesure la performance d'un modèle apprenant sur des données indépendantes.
- **Vérification formelle** : on utilise des méthodes mathématiques pour prouver qu'un programme respecte certaines propriétés (absence d'erreurs, respect de contraintes, sécurité).
  Ces techniques sont utilisées dans l'aéronautique, le spatial ou les systèmes critiques.
- **Audits algorithmiques** : des experts externes examinent le fonctionnement d'un système pour détecter biais, erreurs ou dérives.

Ces méthodes sont indispensables car les algorithmes modernes sont trop complexes pour être compris uniquement « à l'œil nu ».

Les algorithmes de fact-checking : une armée invisible contre la désinformation

Les plateformes de vérification des faits utilisent des algorithmes pour repérer les rumeurs, les fausses citations, les images truquées. Mais contrairement à ce que l'on croit, ces algorithmes ne décident pas : ils signalent.

Ils repèrent les phrases suspectes, les images déjà vues ailleurs, les incohérences temporelles. Ensuite, des humains — journalistes, linguistes, analystes — prennent le relais.

Cette collaboration homme-algorithme est devenue essentielle pour lutter contre la désinformation massive. Sans ces outils, la quantité de contenus à vérifier serait tout simplement ingérable.

Un exemple parfait de symbiose entre intelligence humaine et puissance algorithmique.

Les modérateurs des réseaux sociaux : quand les algorithmes protègent… et se trompent

Les réseaux sociaux utilisent des algorithmes pour filtrer des milliards de messages chaque jour. Ils repèrent les insultes, les menaces, les images choquantes. Mais ils font aussi des erreurs spectaculaires : un tableau de la Renaissance censuré pour « nudité », un message scientifique bloqué pour « contenu dangereux », un poème supprimé pour « discours violent ».

Ces erreurs rappellent que les algorithmes ne comprennent pas le contexte. Derrière eux, des milliers de modérateurs humains corrigent, ajustent, affinent. L'algorithme fait le tri grossier ; l'humain fait la nuance.

Une mécanique imparfaite, mais indispensable.

7.4 Les limites de la transparence

Expliquer un algorithme n'est pas toujours simple. Certains traitements sont trop complexes pour être décrits en détail. D'autres reposent sur des modèles mathématiques difficiles à interpréter.

Une explication complète serait parfois trop longue ou trop technique pour être utile.

Il existe aussi des limites liées :

- à la **protection des données**,
- au **secret industriel**,
- à la **sécurité informatique** (ne pas révéler des détails exploitables par des attaquants).

L'explication doit donc être adaptée au public : ingénieur, utilisateur, décideur, régulateur. L'objectif n'est pas de tout dévoiler, mais de rendre le fonctionnement compréhensible au niveau approprié.

7.5 Tests de raisonnement : quand l'IA joue aux casse-têtes… et triche un peu

Pour comparer intelligence humaine et intelligence artificielle, rien de tel que des casse-têtes conçus pour mettre les neurones à l'épreuve. C'est exactement l'objectif de tests comme **ARC-AGI** (Abstraction and Reasoning Corpus), un ensemble de petits puzzles visuels où il faut repérer des motifs, comprendre une règle implicite, puis la généraliser. En clair : le genre de choses que les humains font naturellement, souvent sans pouvoir expliquer comment.

Pour les machines, c'est une autre histoire. ARC-AGI est réputé difficile parce qu'il exige une forme de raisonnement abstrait, pas seulement de la reconnaissance statistique. Les IA doivent deviner la logique cachée derrière quelques exemples, un exercice qui ressemble davantage à un test de QI qu'à un entraînement de chatbot. Et c'est précisément là que les limites apparaissent : les modèles actuels excellent dans les tâches où les données abondent, mais peinent dès qu'il faut extrapoler à partir de très peu d'indices.

Mais l'histoire ne serait pas complète sans un rebondissement typiquement… artificiel. Car les IA, appliquées et opportunistes, ont une fâcheuse tendance à s'auto-adapter aux tests. Donnez-leur suffisamment d'exemples d'ARC-AGI, et elles finissent par apprendre non pas à raisonner, mais à reconnaître les types de puzzles, les structures récurrentes, voire les solutions probables. Une sorte de bachotage algorithmique. Résultat : leurs performances montent, mais sans que leur compréhension profonde ne progresse vraiment.

Ces tests révèlent donc un paradoxe : l'IA peut devenir très performante sur une épreuve… sans pour autant maîtriser la compétence qu'elle est censée mesurer. L'humain, lui, reste capable d'inventer des stratégies, de reformuler un problème, de changer de perspective. Bref, de raisonner au-delà du cadre.

ARC-AGI et ses cousins nous rappellent que l'intelligence humaine n'est pas qu'une affaire de calcul : c'est aussi une capacité à naviguer dans l'inconnu, à improviser, à comprendre ce qui n'a jamais été vu. Une frontière que les machines n'ont pas encore franchie.

7.6 Enjeux éthiques et sociétaux

L'explicabilité algorithmique est devenue un enjeu majeur dans les débats sur l'intelligence artificielle. Elle permet de détecter les biais présents dans les données, qui peuvent conduire à des décisions injustes. Elle aide à garantir que les systèmes automatisés respectent les principes d'équité, de non-discrimination et de responsabilité.

Elle joue aussi un rôle dans la régulation. Certaines lois exigent que les décisions automatisées puissent être expliquées, notamment lorsqu'elles ont un impact sur les droits des personnes. L'explicabilité devient alors une obligation, et non plus seulement une bonne pratique.

Dans un monde où les algorithmes influencent de plus en plus d'aspects de la vie sociale, cette transparence est indispensable pour maintenir la confiance et assurer un usage responsable des technologies.

7.7 Conclusion du chapitre

L'explicabilité algorithmique vise à rendre les décisions automatisées compréhensibles et vérifiables. Elle répond à des besoins techniques, éthiques et sociétaux. Les algorithmes déterministes sont généralement faciles à expliquer, tandis que les modèles apprenants posent des défis nouveaux. La vérification et les preuves formelles complètent cette démarche en garantissant la fiabilité des systèmes.

Les chapitres suivants décriront des algorithmes concrets, du plus simple au plus avancé, afin d'illustrer ces notions dans des situations réelles.

8 Datalogie – Comprendre la science des données au cœur des algorithmes modernes

La datalogie, parfois présentée comme la « science des données », est devenue l'un des piliers conceptuels et techniques de l'informatique contemporaine.

Elle désigne l'ensemble des méthodes permettant de collecter, organiser, analyser et exploiter les données afin d'en extraire des connaissances ou de guider des décisions.

Si le terme est relativement récent, l'idée qu'il recouvre s'inscrit dans une histoire longue, où se croisent statistiques, informatique, algorithmique et intelligence artificielle.

Comprendre la datalogie, c'est comprendre comment les algorithmes se sont transformés, passant d'outils de calcul à des instruments d'interprétation du monde.

Aujourd'hui : la donnée comme ressource stratégique

Au XXI^{e} siècle, la donnée est devenue une ressource économique, scientifique et géopolitique. Les entreprises l'utilisent pour optimiser leurs opérations, personnaliser leurs services, anticiper les comportements. Les États s'en servent pour planifier, surveiller, réguler.

La datalogie n'est plus seulement une discipline technique : elle est devenue un enjeu de société, au cœur des débats sur la vie privée, l'éthique algorithmique et la souveraineté numérique.

Vocabulaire

Vocabulaire essentiel de la datalogie

Data science / science des données

- **Data science** : discipline qui combine statistiques, informatique et modélisation pour extraire des connaissances à partir de données.
- **Scientifique des données / data scientist** : spécialiste capable de manipuler des données, construire des modèles prédictifs et interpréter les résultats.
- **Analyste de données / data analyst** : rôle plus centré sur l'exploration, la visualisation et l'aide à la décision.
- **Ingénieur données / data engineer** : construit les pipelines, les bases et les infrastructures permettant de collecter, nettoyer et distribuer les données.

Big data / mégadonnées

- **Big data** : ensemble de données si volumineux, varié ou rapide qu'il nécessite des outils spécialisés (Hadoop, Spark, etc.).
- En français, on parle parfois de **mégadonnées**, mais le terme reste peu utilisé dans les milieux professionnels.
- Les trois « V » classiques : **Volume**, **Vélocité**, **Variété** (parfois complétés par Véracité et Valeur).

Données, jeux de données et pipelines

- **Donnée (data)** : unité d'information brute.
- **Jeu de données / dataset** : collection structurée de données.
- **Pipeline de données** : chaîne automatisée qui collecte, transforme et distribue les données.
- **ETL / ELT** : processus d'Extraction, Transformation, Chargement (ou l'inverse).

IA, apprentissage automatique et modèles

- **IA (intelligence artificielle)** : ensemble de techniques visant à reproduire des capacités cognitives.

- **Machine learning / apprentissage automatique** : sous-domaine de l'IA où les modèles apprennent à partir d'exemples.
- **Deep learning / apprentissage profond** : techniques basées sur des réseaux de neurones à plusieurs couches.
- **Modèle / modèle d'IA** : représentation mathématique entraînée pour accomplir une tâche (prédire, classer, générer).

Outils et pratiques associées

- **Data mining / fouille de données** : extraction de motifs ou de corrélations dans de grands volumes de données.
- **Feature engineering / ingénierie des variables** : création de nouvelles caractéristiques pour améliorer un modèle.
- **MLOps** : pratiques permettant de déployer, surveiller et maintenir des modèles en production.
- **Data governance / gouvernance des données** : règles et processus garantissant qualité, sécurité et conformité.

Termes proches ou émergents

- **Datalogie** : terme parfois utilisé en français pour désigner l'ensemble des sciences et techniques de la donnée.
- **Inférence** : utilisation d'un modèle entraîné pour produire une prédiction ou une réponse.
- **Data literacy / littératie des données** : capacité d'un individu à comprendre, analyser et utiliser des données.
- **Data mesh** : approche décentralisée de la gestion des données dans les grandes organisations.

8.1 Définition et périmètre de la datalogie

La datalogie peut être définie comme la discipline qui étudie les données sous toutes leurs formes : leur nature, leur structure, leur qualité, leurs transformations possibles et les méthodes permettant d'en tirer des informations utiles. Elle se situe à l'intersection de plusieurs domaines :

- **statistiques**, pour la modélisation et l'inférence ;
- **informatique**, pour la manipulation, le stockage et le traitement ;
- **algorithmique**, pour la conception de procédures d'analyse ;
- **intelligence artificielle**, pour l'apprentissage automatique et la prédiction.

L'objectif central est de transformer des données brutes — souvent volumineuses, hétérogènes et imparfaites — en connaissances exploitables. Cette transformation repose sur un ensemble d'étapes : acquisition, nettoyage, exploration, modélisation, validation et déploiement.

8.2 Brève histoire de la datalogie

Les racines statistiques (XVIII^{e} – XX^{e} siècle)

Bien avant l'ère numérique, les États et les scientifiques ont cherché à quantifier le réel.

Les premiers recensements modernes, les travaux de Gauss sur la régression, ou encore les lois de probabilité de Laplace constituent les fondations mathématiques de la datalogie.

La statistique naît comme une science de l'État, puis devient une science de la mesure et de l'incertitude.

Les premiers pas de la datalogie : quand les États inventent la donnée

Au XVIII^{e} siècle, les États européens commencent à organiser des recensements systématiques. L'objectif est d'obtenir une vision chiffrée de la population, des richesses, des productions agricoles. C'est l'époque où la donnée devient un outil de gouvernement.

Le statisticien prussien Johann Peter Süssmilch, par exemple, analyse les registres paroissiaux pour dégager des régularités démographiques. Il ne parle pas encore de « données », mais il inaugure une idée fondamentale : le réel peut être décrit par des nombres.
Cette période marque la naissance de la donnée comme objet scientifique.

Gauss, Laplace et la naissance de la modélisation
Au début du XIX^{e} siècle, Carl Friedrich Gauss formalise la méthode des moindres carrés pour ajuster une courbe à des observations.
Pierre-Simon de Laplace développe la théorie des probabilités comme outil d'inférence.
Ces travaux posent les bases de ce qui deviendra plus tard la datalogie :

- la donnée est imparfaite,
- l'incertitude est mesurable,
- un modèle peut résumer une tendance.

L'idée qu'un algorithme puisse « apprendre » à partir d'exemples trouve ici ses racines mathématiques.

L'arrivée de l'informatique (années 1950–1980)

Avec l'apparition des ordinateurs, les données deviennent manipulables à grande échelle.
Les bases de données relationnelles, formalisées par Edgar F. Codd dans les années 1970, permettent de structurer l'information.
Les algorithmes d'analyse statistique sont automatisés, ouvrant la voie à des traitements plus rapides et plus complexes.

1956 : l'informatique rencontre la donnée
L'année 1956 est souvent citée comme celle de la naissance officielle de l'intelligence artificielle, lors de la conférence de Dartmouth. Mais c'est aussi une période où les premiers ordinateurs commerciaux commencent à être utilisés pour traiter des volumes de données jusque-là impossibles à manipuler.
Les banques, les assurances et les administrations adoptent les machines IBM pour automatiser les calculs et les tris. La donnée devient un flux, non plus un simple tableau statique.
Cette transition ouvre la voie à l'idée moderne de traitement automatisé de l'information.

1970 : Edgar F. Codd invente les bases de données relationnelles
En 1970, le chercheur d'IBM Edgar F. Codd publie un article fondateur : A Relational Model of Data for Large Shared Data Banks. Il y propose une manière nouvelle de structurer l'information : les données sont organisées en tables, reliées entre elles par des relations logiques.
Ce modèle révolutionne l'informatique. Il permet de stocker, interroger et manipuler des données de manière fiable et cohérente.
Sans cette invention, la datalogie moderne — et donc l'IA — n'aurait pas pu se développer.

L'explosion des données (années 1990–2000)

L'essor d'Internet, des capteurs numériques et des systèmes d'information produit une quantité inédite de données. On parle alors de big data.

Les entreprises commencent à exploiter ces masses d'informations pour optimiser leurs processus, personnaliser leurs services ou anticiper les comportements.

1990–2000 : l'explosion du Web et l'ère du Big Data

Avec l'arrivée du Web, la quantité de données produites par les humains augmente de manière exponentielle. Chaque clic, chaque requête, chaque transaction laisse une trace.

Les entreprises découvrent qu'elles possèdent des gisements d'informations inexploités. Les moteurs de recherche, les réseaux sociaux et les plateformes de commerce en ligne deviennent des machines à produire des données.

Le terme big data apparaît pour décrire cette nouvelle réalité :

- volume massif,
- variété des formats,
- vitesse de production.

La datalogie change d'échelle.

L'ère de l'apprentissage automatique (années 2010–2020)

Les progrès matériels (GPU), les bibliothèques logicielles (TensorFlow, PyTorch) et les nouveaux modèles (réseaux neuronaux profonds, transformers) transforment la datalogie en moteur de l'intelligence artificielle moderne.

Les algorithmes ne se contentent plus de calculer : ils apprennent à partir des données.

2012 : la révolution du deep learning

En 2012, un événement marque un tournant : l'équipe de Geoffrey Hinton remporte la compétition ImageNet grâce à un réseau neuronal profond, AlexNet. Les performances dépassent largement celles des méthodes classiques.

Ce succès repose sur trois facteurs :

- des volumes gigantesques de données annotées,
- des GPU capables d'entraîner des modèles complexes,
- des architectures neuronales plus profondes et plus efficaces.

La datalogie devient alors le carburant de l'intelligence artificielle moderne.

Aujourd'hui : la datalogie comme infrastructure cognitive

La datalogie est désormais omniprésente : santé, finance, transports, industrie, recherche scientifique, réseaux sociaux.

Elle constitue l'infrastructure cognitive des sociétés numériques, où chaque décision peut être éclairée — ou influencée — par des données.

8.3 Les grandes étapes du travail en datalogie

Collecte et acquisition

Les données proviennent de multiples sources : capteurs, fichiers, bases de données, API, interactions humaines, traces numériques. L'enjeu principal est la qualité : une donnée biaisée ou incomplète compromet l'ensemble de l'analyse.

Nettoyage et préparation

Cette étape, souvent la plus longue, consiste à : corriger les erreurs, gérer les valeurs manquantes, harmoniser les formats, sélectionner les variables pertinentes, créer de nouvelles caractéristiques (feature engineering).

Elle transforme un ensemble brut en un jeu de données cohérent.

Exploration et visualisation

Avant toute modélisation, il faut comprendre la structure des données : distributions, corrélations, anomalies, tendances. Les visualisations jouent un rôle essentiel pour révéler des motifs invisibles dans les tableaux numériques.

Modélisation et algorithmes

La datalogie mobilise une large gamme d'algorithmes : régressions, arbres de décision, clustering, réseaux neuronaux, modèles de recommandation, méthodes de réduction de dimension.

Chaque algorithme répond à un type de question : prédire, classifier, regrouper, simplifier, détecter.

3.5. Validation et déploiement

Un modèle doit être évalué, testé, comparé, puis intégré dans un système réel. La datalogie ne s'arrête pas à la construction d'un modèle : elle inclut sa mise en production et son suivi.

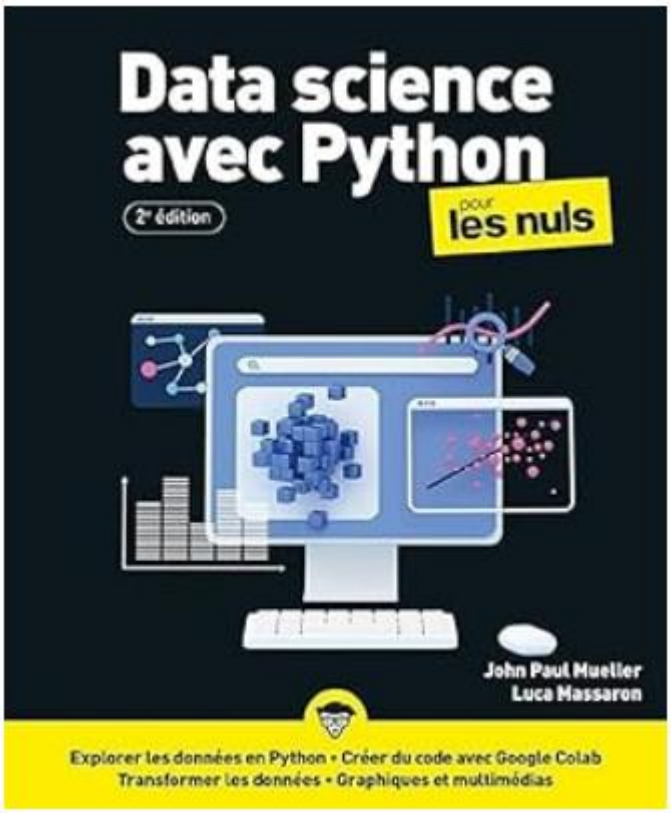
8.4 Datalogie et algorithmes : une relation symbiotique

Les algorithmes sont au cœur de la datalogie, mais leur rôle a évolué. Historiquement, un algorithme était une procédure déterministe, appliquée à des données bien définies. Avec l'essor de l'apprentissage automatique, les algorithmes deviennent des méthodes d'adaptation : ils ajustent leurs paramètres en fonction des données.

Cette évolution a deux conséquences majeures :

1. Les données influencent la forme finale de l'algorithme.
   Un modèle entraîné n'est pas seulement un programme : c'est une structure façonnée par les exemples qu'on lui a fournis.
2. La qualité algorithmique dépend de la qualité des données. Un algorithme d'apprentissage n'est jamais meilleur que les données qui l'ont nourri.

Ainsi, la datalogie transforme l'algorithmique en une discipline plus empirique, où l'expérimentation et l'observation jouent un rôle aussi important que la théorie.



8.5 Le rôle de la datalogie dans l'intelligence artificielle

L'intelligence artificielle moderne repose sur trois piliers :

- **les données**,
- **les algorithmes d'apprentissage**,
- **la puissance de calcul**.

La datalogie fournit le premier pilier et conditionne les deux autres. Sans données, un modèle ne peut pas apprendre ; sans préparation adéquate, il apprend mal ; sans compréhension des données, il est impossible d'interpréter ses résultats.

La datalogie permet également de : détecter les biais dans les modèles, garantir la robustesse des systèmes, assurer la transparence et l'explicabilité, optimiser les performances en production.

Elle constitue donc le socle méthodologique de l'IA, autant sur le plan technique que sur le plan éthique.

8.6 Conclusion : la datalogie comme fondement de l'algorithmique contemporaine

La datalogie n'est pas une discipline périphérique : elle est devenue l'un des cadres conceptuels essentiels pour comprendre les algorithmes modernes.

Elle relie la théorie algorithmique aux données du monde réel, et elle donne aux algorithmes la capacité d'apprendre, de s'adapter et de prédire.

9 Algorithmes en tous genres

9.1 Algorithmes fondamentaux (niveau initiation)

Les algorithmes simples constituent la première étape dans l'apprentissage de la pensée algorithmique. Ils permettent de comprendre comment une méthode se construit, comment elle se déroule et comment elle transforme les données. Leur simplicité n'enlève rien à leur importance : ils forment les briques élémentaires sur lesquelles reposent des traitements plus complexes.

Ce chapitre présente les algorithmes les plus fondamentaux, ceux que l'on rencontre dès les premiers exercices et qui illustrent les principes essentiels de la logique informatique.

ANECDOTE	**L'algorithme qui prédit les éruptions solaires… avant qu'elles ne frappent la Terre** Les éruptions solaires peuvent perturber les satellites, les communications, les réseaux électriques. Pendant longtemps, les scientifiques ne pouvaient que les observer après coup. Puis la NASA et IBM ont mis au point un algorithme capable d'analyser des millions d'images du Soleil pour détecter les signes précurseurs d'une éruption. L'algorithme repère des motifs invisibles à l'œil humain, des micro-variations dans les lignes magnétiques. Résultat : il peut prédire une éruption 48 heures avant qu'elle ne se produise. Ce délai change tout : il permet de mettre les satellites en sécurité et d'anticiper les perturbations. Un algorithme qui lit littéralement l'humeur du Soleil.

9.1.1 La recherche dans une liste

L'un des problèmes les plus courants consiste à déterminer si un élément est présent dans une liste. L'algorithme le plus simple est la recherche linéaire. Il parcourt la liste du début à la fin et compare chaque élément à la valeur recherchée. Si une correspondance est trouvée, l'algorithme s'arrête. Sinon, il conclut que l'élément n'est pas présent.

Cet algorithme illustre plusieurs notions importantes : le parcours séquentiel, la comparaison et la terminaison. Il montre aussi que la simplicité peut suffire lorsque les données sont peu nombreuses.

Même si d'autres méthodes plus rapides existent, la recherche linéaire reste un point de départ indispensable.

L'exemple Python montre le fonctionnement d'un tel algorithme. Soit une liste de donnée. Il s'agit se de savoir si le nombre 21 en fait partie. On interroge la fonction recherche linéaire (RL). Dans ce module de programme, une boucle examine chacun des éléments de la liste, les uns après les autres. Pour chacun, le programme compare l'élément en cours d'examen à l'élément cherché. S'il y a correspondance, ce programme retourne la valeur True (vraie) au programme d'appel et celui-ci affiche « True). Si en fin d'examen de la liste, aucune correspondance n'a été trouvée, le programme d'appel affiche « False » pour Faux.

Exemple PYTHON — *Recherche dans une liste*

```
def RL(liste, valeur):
    for element in liste:
        if element == valeur:
            return True
    return False
```

Commentaires

Le programme détecte si un nombre (nommé valeur) est présent dans une liste donnée.

```
# Exemple d'utilisation
L = [12, 5, 8, 21, 3, 18]

print("Cas de 21:", RL(L, 21))
print("Cas de 7:", RL (L,7))

Résultat :
Cas de 21: True
Cas de 7: False
```

Définition d'un module (RL) qui parcourt la liste et détermine si un élément particulier est dans la liste.
Le module retourne « True » (Vrai) dès qu'il trouve une concordance entre l'élément examiné et l'élément cherché.
S'il n'a rien trouvé en fin de balayage de la liste, il retourne « False » (Faux).

9.1.2 Le tri élémentaire

Trier une liste est une opération fondamentale. Parmi les méthodes les plus simples, le tri par sélection et le tri par insertion sont souvent étudiés en premier.

- Le tri par sélection consiste à repérer le plus petit élément, à le placer en première position, puis à recommencer sur le reste de la liste.
- Le tri par insertion construit progressivement une liste triée en insérant chaque nouvel élément à sa place.

ANECDOTE

Le tri par bulles, inventé… par personne
Le tri à bulles est l'un des algorithmes les plus connus, mais personne ne sait vraiment qui l'a inventé.
Il est tellement simple qu'il semble avoir émergé spontanément dans plusieurs communautés.
Un algorithme sans auteur, comme une recette de grand-mère devenue universelle.

Le tri par insertion, inventé par les secrétaires
Bien avant l'informatique, les secrétaires triaient les fiches papier en les insérant une à une dans la bonne position.
Les informaticiens ont simplement observé la scène et dit : « Tiens, ça ferait un bon algorithme. »
Le tri par insertion est donc né d'un geste de bureau banal. Comme quoi, l'innovation commence parfois dans un classeur.

Ces algorithmes sont faciles à comprendre et à mettre en œuvre. Ils permettent d'introduire la notion de boucle imbriquée et de montrer comment une méthode peut transformer progressivement une structure de données. Leur efficacité est limitée, mais leur valeur pédagogique est essentielle.

Exemple PYTHON — *Tri par sélection*

```
def triS(L):
    n = len(L)
    for i in range(n):
        mini = i
        for j in range(i + 1, n):
            if L[j] < L[mini]:
                mini = j
        L[i], L[mini] = L[mini], L[i]
    return L
```

Commentaires
Le module triS (tri par sélection) cherche progressivement le plus petit nombre parmi ceux qui n'ont pas encore été analysés.

```
# Exemple d'utilisation
Liste = [12, 5, 8, 21, 3, 18]
print(triS(Liste))

 Résultat :
[3, 5, 8, 12, 18, 21]
```

Si un plus petit est trouvé, il (mini) remplace la valeur courante (i) dans la liste L.
Analyse terminée, le module retourne la liste triée.

L'exemple propose une liste ; le résultat du traitement montre la liste triée.

Exemple PYTHON

Tri par insertion

```
def triI(L):
    for i in range(1, len(L)):
        valeur = L[i]
        j = i - 1
        while j >= 0 and L[j] > valeur:
            L[j + 1] = L[j]
            j -= 1
        L[j + 1] = valeur
    return L

# Exemple d'utilisation
Liste = [12, 5, 8, 21, 3, 18]
print(triI(Liste))

 Résultat :
[3, 5, 8, 12, 18, 21]
```

Idée générale : à chaque étape (i) :

1. **On cherche le plus petit élément dans la partie non triée (à droite).**
2. **On l'échange avec l'élément en position (i).**
3. **La partie à gauche de (i) est alors triée.**

Il ne reste qu'un élément, la liste est déjà triée : [3, 5, 8, 12, 21]

Tri par insertion: explications étape par étape

Étape 1 – (i = 0)	Étape 2 – (i = 1)	Étape 3 – (i = 2)	Étape 4 – (i = 3)	Étape 5 – (i = 4)
[12, 5, 8, 21, 3]	**[3, 5, 8, 21, 12]**	**[3, 5, 8, 21, 12]**	**[3, 5, 8, 21, 12]**	**[3, 5, 8, 12, 21]**
Partie triée : [] Partie non triée : [12, 5, 8, 21, 3] Plus petit élément = 3 (position 4) On échange 12 et 3	**Partie triée : [3] Partie non triée : [5, 8, 21, 12] Plus petit élément = 5 (déjà à la bonne place) Pas d'échange**	**Partie triée : [3, 5] Partie non triée : [8, 21, 12] Plus petit élément = 8 (déjà à la bonne place) Pas d'échange :**	**Partie triée : [3, 5, 8] Partie non triée : [21, 12] Plus petit élément = 12 (position 4) On échange 21 et 12**	**Il ne reste qu'un élément, la liste est déjà triée : [3, 5, 8, 12, 21]**
[3, 5, 8, 21, 12]	**[3, 5, 8, 21, 12]**	**[3, 5, 8, 21, 12]**	**[3, 5, 8, 12, 21]**	**[3, 5, 8, 12, 21]**

9.1.3 Le calcul du maximum ou du minimum

Trouver la plus grande ou la plus petite valeur d'une liste est un exercice classique. L'algorithme consiste à parcourir la liste en conservant la meilleure valeur rencontrée jusqu'à présent. À chaque étape, on compare l'élément courant avec la valeur retenue et on la met à jour si nécessaire.

Cet algorithme montre comment une variable peut servir de mémoire intermédiaire et comment une boucle peut accumuler une information. Il illustre aussi la notion d'invariant : à tout moment, la valeur retenue est la meilleure parmi celles déjà examinées.

9.1.4 La somme et la moyenne

Additionner les éléments d'une liste ou en calculer la moyenne repose sur une idée simple : accumuler progressivement les valeurs. L'algorithme parcourt la liste et ajoute chaque élément à une variable qui contient la somme. Une fois le parcours terminé, il suffit de diviser cette somme par le nombre d'éléments pour obtenir la moyenne.

Ces traitements montrent comment un algorithme peut produire une information nouvelle à partir de données brutes. Ils introduisent la notion d'accumulateur, très utilisée dans les calculs numériques.

Exemple PYTHON — *Calcul Somme et Moyenne*

```
def somme_et_moyenne(L):
    somme = 0  # accumulateur
    for n in L:
        somme += n
    moyenne = somme / len(L) if L else 0
    return somme, moyenne


# Exemple d'utilisation
Liste = [12, 5, 8, 21, 3]

s, m = somme_et_moyenne(Liste)
print("Somme :", s)
print("Moyenne :", m)

 Résultat :
Somme : 49
 Moyenne : 9.8
```

Commentaires

Le module calcule la somme et la moyenne des nombres d'une liste L donnée.

Pour la somme, ls nombres sont ajoutés les uns après les autres (+=).

La moyenne est calculée en divisant la somme par la quantité de nombres dans la liste (len(L)).

On imprime le résultat de ces deux calculs.

Notez que le module retourne somme et moyenne en une seule instruction.

9.1.5 Les premières manipulations de chaînes de caractères

Les chaînes de caractères sont omniprésentes dans les programmes. Les algorithmes simples permettent de compter les lettres, de vérifier la présence d'un mot ou de transformer une phrase. Ces traitements reposent sur les mêmes principes que ceux appliqués aux listes : parcours, comparaison, accumulation.

Ils montrent que les algorithmes ne se limitent pas aux nombres. Ils peuvent manipuler des textes, des symboles ou des données plus complexes, tant que celles-ci peuvent être représentées de manière structurée.

Exemple PYTHON — *Compter les lettres*

```
phrase = "Bonjour tout le monde"
kt = 0

for caractere in phrase:
    if caractere == "o":
        kt += 1
```

Commentaires

Ce programme compte la quantité de lettres « O » dans la phrase proposée.

Notez que kt est le compteur de lettres. Il est incrémenté (+1) avec l'instruction:

```
print("Nombre de 'o' :", kt)

 Résultat :
 Nombre de 'o' : 4

 ____________________________
 Écriture abrégée
p = "Bonjour tout le monde"
kt = 0
for c in p:
    if c == "o":
        kt += 1
print("Nb de 'o' :", kt)
```

kt = kt + 1
ou sous forme abrégée avec kt += 1.

Le second programme est rigoureusement identique. Il illustre le fait que les variables peuvent être abrégées (comme phrase en p). La seule limite étant de continuer à s'y retrouver sans confusion des notations.

Exemple PYTHON

Manipulations de lettres

```
phrase = "algorithmes simples"
mot = "simples"
if mot in phrase:
    print("présent")
else:
    print("absent")

Pmin = "algorithmes"
Pmaj = ""
for caractere in Pmin:
    Pmaj += caractere.upper()
print(Pmaj)

phrase = "algorithmique"
voyelles = "aeiouy"
resultat = ""
for c in phrase:
    if c not in voyelles:
        resultat += c
print(resultat)

 Résultat :
 présent
 ALGORITHMES
 lgrthmq
```

Commentaires

Programme 1 : cherche la présence d'un mot dans un texte. Ici le mot cherché est « simples » et il est effectivement présent.

Programme 2 : Mettre les lettres de Pmin (phrase minuscule) d'une phrase en majuscules Pmaj (phrase majuscule).

Programme 3 : éliminer les voyelles dans une phrase. Chaque lettre de la phrase est comparée à chacune des six voyelles. La lettre est conservée si elle n'est pas une voyelle.

9.1.6 Les premiers algorithmes numériques

Certains algorithmes simples concernent directement les nombres. Le calcul du PGCD (Plus Grand Commun Diviseur) avec la méthode d'Euclide, par exemple, repose sur une répétition d'opérations élémentaires. La vérification de la primalité d'un nombre peut se faire en testant les divisions possibles jusqu'à une certaine limite.

Ces algorithmes montrent que des problèmes mathématiques peuvent être résolus par des méthodes systématiques. Ils illustrent aussi la notion d'efficacité : certaines méthodes naïves fonctionnent, mais deviennent rapidement trop lentes lorsque les nombres augmentent.

ANECDOTE

Les drones en essaim : quand un feu d'artifice devient un algorithme vivant

Les spectacles de drones lumineux, qui remplacent parfois les feux d'artifice, reposent sur des algorithmes d'une élégance fascinante.

Chaque drone connaît sa position, sa trajectoire et celle de ses voisins. Mais aucun ne dirige l'ensemble : c'est un algorithme d'essaim, inspiré des bancs de poissons et des vols d'oiseaux.

Le résultat est hypnotique : des centaines de drones dessinent des formes dans le ciel, parfaitement synchronisés, sans jamais se percuter. Le plus étonnant est que ces algorithmes sont robustes : si un drone tombe en panne, les autres réorganisent spontanément la formation.

Un ballet aérien où la chorégraphie est mathématique.

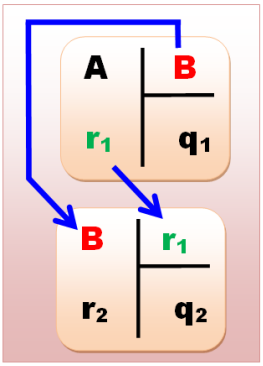
Algorithme d'Euclide pour le calcul du PGCD (Plus Grand Commun Diviseur)

Il s'agit d'effectuer des divisions en cascade : les résultats de l'une servent à poser la suivante:

- Le diviseur ***B*** devient le dividende; et
- Le reste r_1 devient le diviseur.

Arrêt lorsque le reste de la division est nul.

Le reste trouvé avant le reste nul est le PGCD des nombres A et B.



Exemple PYTHON

Algorithme d'Euclide

```
def pgcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
# Exemple d'utilisation
print(pgcd(144, 54))

Résultat :
18
```

Commentaires

Tant que b est différent (!=) de 0, on continue.

L'opération consiste à placer b dans a et a mod b dans b. Voyez (jaune) comment formuler cela en une seule instruction.

PGCD (144, 54) = 18

En effet :

$144 = 2^4 \times 3^2$

$54 = 2 \times 3^3$

9.2 Algorithmes du niveau lycée

Au lycée, l'étude des algorithmes prend une dimension plus structurée. Les élèves découvrent des méthodes qui dépassent les manipulations élémentaires et qui permettent de résoudre des problèmes plus variés.

Ces algorithmes introduisent des notions essentielles comme la récursivité, les parcours de structures arborescentes ou les méthodes de tri efficaces. Ils constituent une étape importante dans la formation, car ils montrent comment des idées simples peuvent être combinées pour produire des traitements plus puissants.

Ce chapitre présente les principaux algorithmes étudiés à ce niveau et explique leur logique de manière progressive.

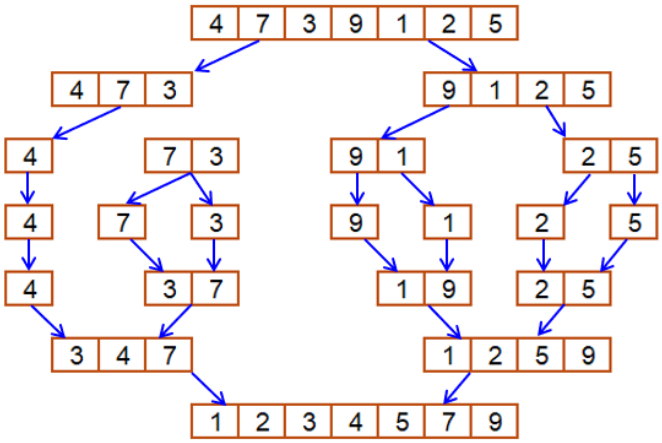
9.2.1 Les tris efficaces

Les tris élémentaires permettent de comprendre la logique d'un algorithme, mais ils deviennent rapidement trop lents lorsque les données augmentent. Au lycée, on introduit des méthodes plus performantes, comme le tri par fusion ou le tri rapide. Ces algorithmes reposent sur une idée commune : diviser la liste en parties plus petites, les trier séparément, puis les rassembler.

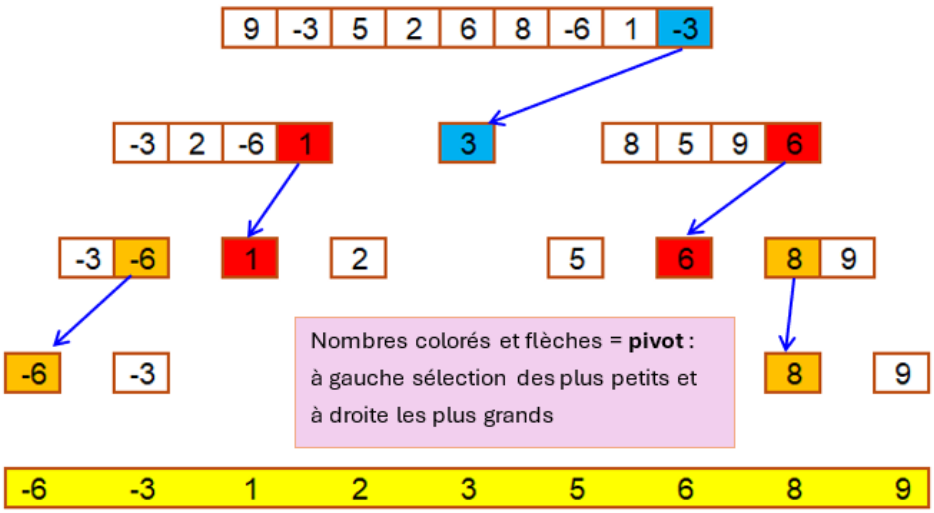
Le tri par fusion découpe la liste en deux moitiés, trie chacune d'elles, puis les combine en une liste ordonnée. Le tri rapide choisit un élément pivot et organise la liste en deux groupes : ceux qui sont plus petits et ceux qui sont plus grands.

Ces méthodes montrent comment une stratégie bien pensée peut réduire considérablement le temps d'exécution. Elles illustrent aussi l'importance de la récursivité, qui permet de traiter un problème en le décomposant en sous-problèmes similaires.

Principe du tri par fusion



Principe du tri par choix d'un pivot



9.2.2 Les arbres binaires et leurs parcours

Les arbres binaires sont des structures de données qui permettent de représenter des relations hiérarchiques. Chaque nœud possède au plus deux sous-nœuds, ce qui facilite leur exploration. Les parcours d'arbres — en profondeur ou en largeur — sont des algorithmes classiques du lycée.

Le parcours en profondeur explore un chemin jusqu'à son extrémité avant de revenir en arrière pour explorer les autres branches. Le parcours en largeur examine d'abord les nœuds proches de la racine avant de descendre progressivement.

Ces méthodes montrent comment organiser une exploration systématique d'une structure complexe. Elles introduisent aussi des notions importantes comme les piles et les files, utilisées pour mémoriser les nœuds à visiter.

9.2.3 Les algorithmes de graphes : BFS et DFS

Les graphes généralisent les arbres en permettant des connexions plus variées entre les éléments. Ils servent à modéliser des réseaux, des itinéraires ou des relations. Deux algorithmes fondamentaux permettent de les explorer : la recherche en largeur (BFS) et la recherche en profondeur (DFS).

La recherche en largeur examine les sommets par niveaux successifs. Elle est utile pour trouver le plus court chemin dans un graphe non pondéré. La recherche en profondeur suit un chemin jusqu'au bout avant de revenir en arrière. Elle permet de détecter des cycles ou de vérifier la connectivité d'un graphe.

Ces algorithmes montrent comment une structure abstraite peut être parcourue de manière méthodique.

Lire à travers des rouleaux antiques brûlés : l'algorithme archéologue

Certains rouleaux antiques, comme les manuscrits carbonisés d'Herculanum ou les rouleaux dharani, sont si fragiles qu'ils se désagrègent au toucher.

Pendant des siècles, ils étaient considérés comme perdus.

Puis des chercheurs ont mis au point un algorithme d'imagerie capable de « dérouler » virtuellement un rouleau sans l'ouvrir.

L'algorithme reconstruit l'intérieur du rouleau couche par couche, détecte l'encre, puis recompose le texte.

Grâce à cette technique, des textes invisibles depuis deux mille ans ont été lus pour la première fois.

Un algorithme qui ressuscite des voix disparues.

9.2.4 La récursivité : une nouvelle manière de penser

La récursivité consiste à définir un algorithme qui s'appelle lui-même pour résoudre un problème. Cette approche peut sembler déroutante au premier abord, mais elle permet d'exprimer des solutions de manière étonnamment concise et élégante.

Un algorithme récursif repose toujours sur deux éléments essentiels :

- un point d'arrêt, qui fournit immédiatement une réponse lorsque le problème est suffisamment simple pour être résolu directement ;
- une règle de réduction, qui transforme le problème initial en une version plus petite du même problème, jusqu'à atteindre le point d'arrêt.

Cette structure en miroir — résoudre un problème en le ramenant à lui-même — est au cœur de nombreux algorithmes efficaces. On la retrouve dans les méthodes de tri rapides, dans les

parcours d'arbres, dans la définition de suites mathématiques ou encore dans la résolution de problèmes géométriques.

La récursivité permet souvent de décrire des processus complexes avec très peu d'instructions, en s'appuyant sur la répétition naturelle du problème. Elle exige toutefois une grande rigueur : si le point d'arrêt est mal défini ou oublié, l'algorithme peut s'appeler indéfiniment et ne jamais s'arrêter.

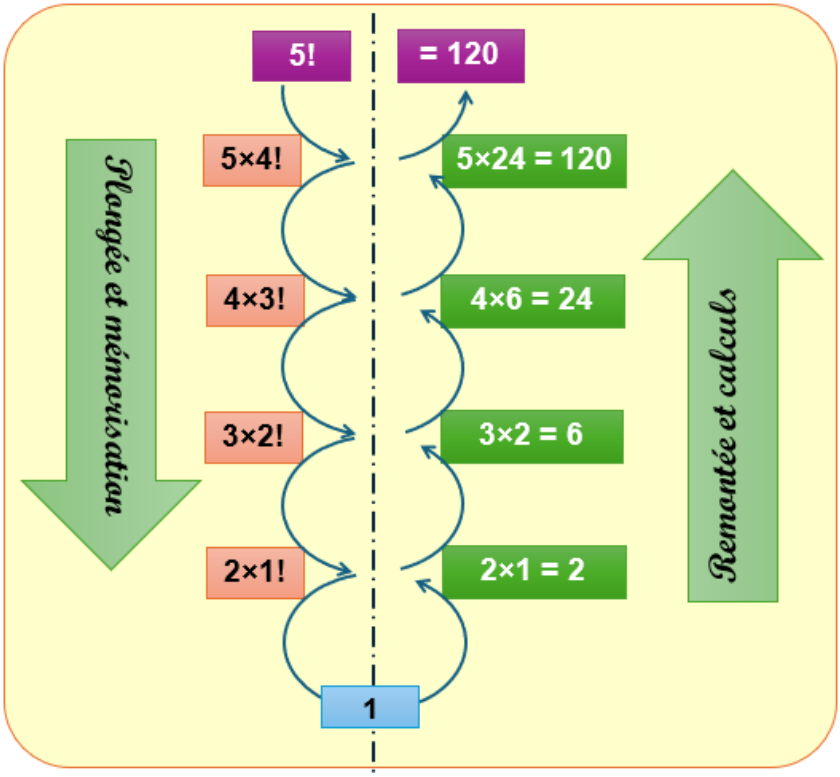
Au-delà de son utilité pratique, la récursivité développe une manière particulière de raisonner : penser un problème non pas comme une suite d'étapes linéaires, mais comme une structure qui se replie sur elle-même. C'est une façon différente — et souvent plus abstraite — de concevoir les solutions algorithmiques.

Illustration de la récursivité avec le calcul de factorielle 5

Définition de la factorielle d'un entier : **n ! = n (n – 1) !** (formule récursive) & **1 ! = 1** (point d'arrêt).
Le calcul récursif suit deux phases complémentaires :

- **La descente** : tant que la valeur de la factorielle n'est pas connue, la fonction s'appelle elle-même en demandant la factorielle d'un nombre plus petit.
  On descend ainsi jusqu'au cas de base (point d'arrêt).
- **La remontée** : dès que l'on atteint le cas simple (1! = 1), chaque appel peut enfin se résoudre. On remonte alors la chaîne des appels en calculant successivement (2!), puis (3!), etc., grâce aux expressions rencontrées pendant la descente.

L'aspect le plus surprenant — et le plus essentiel — de ce processus est que, durant la descente, **la fonction factorielle s'appelle elle-même**. C'est précisément ce mécanisme d'auto-référence contrôlée qui caractérise un calcul récursif.



Exemple PYTHON — *Factorielle – Récursivité*

```
def factorielle(n):
    if n == 1:
        return 1
    else:
        return n * factorielle(n - 1)

# Exemple d'utilisation
print(factorielle(5))
```

Résultat :

```
120
```

Commentaires

Le module de calcul de la factorielle commence par s'interroger si n = 1 auquel cas, la factorielle vaut 1.

Sinon , tout simplement (magie !), le programme relance le calcul de la factorielle mais pour le nombre précédent et multiplie cette valeur par n.

9.2.5 Les algorithmes numériques avancés

Au lycée, certains algorithmes numériques sont étudiés pour illustrer la puissance des méthodes systématiques. Le calcul du PGCD par l'algorithme d'Euclide, déjà présenté dans les chapitres précédents, peut être approfondi en version récursive. La recherche de nombres premiers peut être abordée avec le crible d'Ératosthène, qui élimine progressivement les multiples pour ne conserver que les nombres premiers.

Ces algorithmes montrent comment une idée simple peut être appliquée à grande échelle. Ils permettent aussi d'introduire la notion d'optimisation : certaines méthodes naïves fonctionnent, mais d'autres sont beaucoup plus rapides.

Exemple PYTHON — *Nombres premiers*

```
def est_premier(n):
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    limite = int(n**0.5) + 1
    for d in range(3, limite, 2):
        if n % d == 0:
            return False
    return True

# Exemple
print(est_premier(29))   # True
print(est_premier(100)) # False
```

Résultat :

```
True
False
```

Commentaires

Méthode complète : on élimine les cas n inférieur à 2 et le cas 2 qui est premier. Tous les autres nombres premiers sont impairs.

Recherche parmi ces impairs et jusqu'à la racine du nombre ; au-delà ce n'est pas la peine.

Exploration des nombres à partir de 3 jusqu'à la limite et par pas de 2.

Si notre nombre est divisible par d (vaut 0 modulo d => n % d == 0), alors le nombre exploré est composé.

29 est bien premier, et
100 est bien composé.

```
from sympy import isprime
print(isprime(29))    # True
print(isprime(100))   # False
```

Ces lignes montrent le même résultat en exploitant la bibliothèque sympy de Python.

9.2.6 Les algorithmes de manipulation de données

Les élèves apprennent également à manipuler des tableaux, des listes ou des chaînes de caractères de manière plus avancée.

Les algorithmes de filtrage, de fusion ou de transformation permettent de traiter des données structurées. Ils montrent comment combiner des parcours, des conditions et des opérations simples pour obtenir des résultats plus élaborés.

Ces traitements préparent à l'analyse de données et à la programmation de projets plus complexes. Ils renforcent la maîtrise des structures de contrôle et des structures de données.

L'algorithme qui prédit les tremblements de terre… en écoutant la Terre respirer

Des géophysiciens ont découvert que les roches émettent des micro-vibrations avant de céder.

Ces signaux sont trop faibles pour être détectés par les humains, mais un algorithme peut les reconnaître. En analysant des milliers d'heures d'enregistrements, il a appris à repérer les « soupirs » de la Terre avant une rupture. Dans certains cas, il a prédit une secousse plusieurs heures à l'avance.

Ce n'est pas encore une solution miracle, mais c'est la première fois qu'un algorithme parvient à lire les signaux précurseurs d'un séisme.

Une oreille artificielle posée sur la croûte terrestre.

9.2.7 Conclusion du chapitre

Les algorithmes étudiés au lycée constituent une étape importante dans l'apprentissage de l'informatique. Ils introduisent des méthodes plus efficaces, des structures plus riches et des concepts plus abstraits. Ils montrent comment la récursivité, les arbres, les graphes et les tris avancés permettent de résoudre des problèmes variés.

Ces chapitres ont présenté les principales notions abordées au niveau lycée. Le chapitre suivant prolongera cette progression en présentant des algorithmes du supérieur, qui s'appuient sur ces bases pour traiter des problèmes encore plus complexes.

9.3 Algorithmes du niveau supérieur

Ce chapitre présente les grandes familles d'algorithmes rencontrées dans les premières années d'études supérieures, en montrant comment elles prolongent les idées déjà acquises.

Tentons d'abord d'identifier les grandes familles d'algorithmes selon le traitement de l'information.

9.3.1 Méthodes de Monte-Carlo : explorer l'incertitude par l'échantillonnage

Les méthodes de Monte-Carlo constituent une famille d'algorithmes fondés sur une idée simple mais puissante : utiliser le **hasard contrôlé** pour explorer un espace de possibilités trop vaste pour être parcouru exhaustivement. Au lieu de calculer toutes les configurations possibles d'un problème — souvent un objectif inaccessible — on génère un grand nombre d'échantillons aléatoires, puis on estime la solution à partir de leur comportement statistique.

Le principe général est le suivant :

- on simule aléatoirement un grand nombre de scénarios ;
- on évalue chacun selon un critère donné ;

- on agrège les résultats pour obtenir une estimation fiable.

Cette approche est particulièrement utile lorsque l'espace des états est immense, comme dans les jeux complexes, l'optimisation ou la modélisation physique.

Dans le domaine des jeux de stratégie, les méthodes de Monte-Carlo ont joué un rôle décisif. Elles ont permis de contourner l'explosion combinatoire du jeu de Go, où l'arbre des coups est trop vaste pour les techniques classiques. L'algorithme Monte-Carlo Tree Search (MCTS) combine exploration aléatoire et sélection intelligente des branches prometteuses. À chaque itération, il simule des parties rapides, puis renforce les chemins qui mènent aux meilleurs résultats.

Cette approche probabiliste a ouvert la voie aux systèmes modernes d'IA, capables de naviguer dans des environnements incertains, partiellement observables ou trop complexes pour une analyse déterministe. Monte-Carlo n'est pas seulement une technique : c'est une philosophie algorithmique fondée sur l'approximation intelligente.

9.3.2 La programmation dynamique : éviter les répétitions inutiles

La programmation dynamique est une technique qui consiste à résoudre un problème en mémorisant les résultats intermédiaires pour éviter de les recalculer. Elle s'applique lorsqu'un problème peut être décomposé en sous-problèmes qui se recouvrent. Plutôt que de recalculer plusieurs fois la même valeur, l'algorithme la stocke dans une table et la réutilise.

Cette méthode transforme souvent un algorithme exponentiel en un algorithme polynomial. Elle est utilisée dans des domaines variés : optimisation, bio-informatique, analyse de séquences, calculs combinatoires. Elle montre comment une bonne organisation des calculs peut améliorer considérablement les performances. Elle illustre aussi l'importance de la structure du problème : un algorithme efficace repose souvent sur une observation fine de ses répétitions internes.

9.3.3 Les algorithmes avancés de graphes

Les graphes permettent de modéliser des réseaux, des relations ou des déplacements. Dans l'enseignement supérieur, on étudie des algorithmes capables de résoudre des problèmes plus complexes que les simples parcours. Parmi eux, les algorithmes de plus court chemin occupent une place centrale.

L'algorithme de Dijkstra permet de trouver le chemin le plus court dans un graphe pondéré à poids positifs. Il s'appuie sur une structure de données appelée tas, qui permet de sélectionner rapidement le sommet le plus prometteur. L'algorithme de Bellman-Ford traite les graphes contenant des poids négatifs, au prix d'un temps d'exécution plus long. L'algorithme de Floyd-Warshall calcule les distances minimales entre tous les couples de sommets en utilisant une approche dynamique.

Ces méthodes montrent comment des problèmes apparemment complexes peuvent être résolus de manière systématique. Elles illustrent aussi l'importance du choix de la structure de données, qui influence directement l'efficacité de l'algorithme.

9.3.4 Les structures de données avancées

Pour manipuler efficacement des données volumineuses, il est nécessaire d'utiliser des structures plus sophistiquées que les simples listes ou tableaux. Les tas permettent de gérer des priorités. Les arbres équilibrés, comme les arbres AVL ou les arbres rouges-noirs, garantissent des temps d'accès logarithmiques. Les tables de hachage offrent un accès très rapide aux données, à condition de bien gérer les collisions.

Ces structures permettent de résoudre des problèmes qui seraient trop lents avec des outils plus simples. Elles montrent que l'efficacité d'un algorithme dépend autant de la méthode que de la manière dont les données sont organisées. Leur étude développe une compréhension plus fine des compromis entre rapidité, mémoire et complexité.

9.3.5 Les algorithmes probabilistes

Certains algorithmes utilisent le hasard pour accélérer les calculs ou pour contourner des difficultés. Ils ne donnent pas toujours la réponse exacte, mais ils fournissent un résultat correct avec une forte probabilité. Cette approche peut sembler surprenante, mais elle est très efficace dans de nombreux domaines.

Les algorithmes probabilistes sont utilisés pour tester la primalité de grands nombres, pour rechercher des motifs dans des textes ou pour optimiser des fonctions complexes. Ils montrent que l'incertitude peut devenir un outil, à condition d'être maîtrisée. Leur étude introduit des notions de probabilité et de statistiques dans l'analyse algorithmique.

9.3.6 Les heuristiques et l'optimisation

Certains problèmes sont trop difficiles pour être résolus exactement dans un temps raisonnable. C'est le cas de nombreux problèmes d'optimisation, comme la planification, l'ordonnancement ou le voyageur de commerce. Les heuristiques proposent des solutions approchées, obtenues rapidement, qui sont souvent suffisantes dans la pratique.

Ces méthodes reposent sur des idées simples : choisir la meilleure option locale, explorer plusieurs pistes en parallèle, ou améliorer progressivement une solution initiale. Elles montrent que l'algorithmique ne se limite pas aux solutions exactes. Elle cherche aussi des compromis entre précision et rapidité, adaptés aux besoins réels.

9.3.7 Les algorithmes de traitement de données

Dans l'enseignement supérieur, on étudie également des algorithmes capables de manipuler des volumes importants de données. Les méthodes de tri optimisées, les algorithmes de fusion de fichiers, les techniques de recherche dans des bases de données ou les algorithmes de compression en sont des exemples.

Ces traitements montrent comment organiser les données pour réduire les accès mémoire, comment exploiter les propriétés des fichiers triés ou comment transformer une information pour la rendre plus compacte. Ils préparent aux applications concrètes rencontrées dans l'industrie et dans la recherche.

9.3.8 Conclusion du chapitre

Les algorithmes du niveau supérieur permettent de résoudre des problèmes plus complexes et plus variés. Ils s'appuient sur des techniques puissantes comme la programmation dynamique, les structures avancées, les algorithmes de graphes ou les méthodes probabilistes. Leur étude développe une compréhension plus profonde de l'efficacité, de la structure des données et des limites du calcul.

Ce chapitre a présenté les grandes familles de ces algorithmes. Le chapitre suivant abordera les algorithmes les plus avancés, ceux qui constituent la base du fonctionnement des technologies modernes, comme les moteurs de recherche, la cryptographie ou l'intelligence artificielle.

9.4 Algorithmes au cœur du numérique moderne

Les technologies que nous utilisons chaque jour reposent sur des algorithmes d'une grande sophistication. Ils organisent l'information, sécurisent les communications, analysent des données massives et apprennent à partir d'exemples. Leur fonctionnement est souvent invisible, mais leur rôle est essentiel.

Ce chapitre présente les grandes familles d'algorithmes qui structurent l'informatique contemporaine. Ils illustrent la manière dont les idées étudiées dans les chapitres précédents se combinent pour répondre à des besoins réels, à grande échelle.

9.4.1 Les moteurs de recherche : organiser l'information mondiale

Les moteurs de recherche doivent parcourir des milliards de pages, les analyser, les classer et fournir en une fraction de seconde les résultats les plus pertinents. Pour cela, ils utilisent plusieurs types d'algorithmes.

Le premier est l'algorithme de crawling, qui explore le web en suivant les liens entre les pages. Il construit progressivement une carte du réseau. Le second est l'indexation, qui transforme chaque page en une représentation structurée permettant une recherche rapide. Le troisième est le classement, qui détermine l'ordre d'affichage des résultats. L'un des algorithmes les plus connus est PageRank, qui évalue l'importance d'une page en fonction des liens qu'elle reçoit.

Ces algorithmes doivent être rapides, robustes et capables de s'adapter à un web en constante évolution. Ils montrent comment les graphes, les statistiques et l'optimisation se combinent pour organiser une quantité d'information gigantesque.

9.4.2 La cryptographie moderne : protéger les communications

La cryptographie permet de sécuriser les échanges sur Internet. Elle repose sur des algorithmes mathématiques qui transforment un message en une forme illisible sans la clé appropriée. Les méthodes modernes utilisent des propriétés des nombres premiers, des courbes elliptiques ou des fonctions de hachage.

L'algorithme RSA, par exemple, repose sur la difficulté de factoriser de grands nombres. Les algorithmes de courbes elliptiques utilisent des structures mathématiques plus complexes pour offrir une sécurité équivalente avec des clés plus courtes. Les fonctions de hachage, comme SHA-256, transforment un message en une empreinte unique, utilisée pour vérifier son intégrité.

Ces algorithmes doivent être à la fois rapides et extrêmement difficiles à contourner. Ils illustrent la manière dont les mathématiques pures peuvent devenir des outils essentiels pour la sécurité numérique.

9.4.3 La compression : réduire la taille des données

La compression permet de stocker et de transmettre des données en utilisant moins d'espace. Elle repose sur des algorithmes capables de repérer les répétitions, les régularités ou les structures internes d'un fichier. Les méthodes sans perte, comme Huffman ou LZW, permettent de reconstruire exactement les données originales. Elles sont utilisées pour les textes, les programmes ou les images simples.

Les méthodes avec perte, comme JPEG ou MP3, éliminent certaines informations jugées moins importantes pour l'œil ou l'oreille. Elles permettent des compressions beaucoup plus fortes, au prix d'une légère dégradation. Ces algorithmes reposent sur des transformations mathématiques, comme la transformée en cosinus, qui séparent les composantes essentielles d'un signal.

La compression montre comment une analyse fine des données permet de réduire leur taille sans en altérer l'usage.

9.4.4 STRIPS : quand les machines apprennent à planifier

STRIPS pour Stanford Research Institute Problem Solver.
SRI pour Stanford Research Institute

Au début des années 1970, dans les laboratoires du SRI, les chercheurs cherchent à doter les machines d'une capacité jusque-là réservée aux humains : planifier. De cette ambition naît STRIPS, un système qui marque une rupture dans l'histoire de l'IA. Pour la première fois, un programme ne se contente plus d'exécuter des instructions ; il **raisonne** sur ses actions, anticipe leurs effets et construit une séquence cohérente pour atteindre un but.

STRIPS décrit le monde comme un ensemble de faits logiques, et chaque action comme une transformation de ces faits. Le robot Shakey, pour lequel il est conçu, devient ainsi capable d'élaborer un plan pour traverser une pièce, déplacer un objet ou contourner un obstacle.

Derrière cette simplicité apparente se cache une idée fondatrice : un algorithme peut manipuler des représentations symboliques pour produire un comportement intelligent. STRIPS ouvre alors la voie à toute une lignée de planificateurs modernes, encore utilisés aujourd'hui en robotique, en logistique et dans les jeux vidéo.

9.4.5 L'apprentissage automatique : des algorithmes qui apprennent

L'apprentissage automatique, ou machine learning, repose sur des algorithmes capables d'améliorer leurs performances en analysant des données. Contrairement aux algorithmes traditionnels, ils ne suivent pas une suite d'instructions fixes. Ils ajustent leurs paramètres pour s'adapter aux exemples qu'ils reçoivent.

Les réseaux de neurones en sont un exemple emblématique. Ils sont constitués de couches de neurones artificiels qui transforment progressivement les données. L'apprentissage se fait grâce à un algorithme appelé rétropropagation du gradient, qui ajuste les paramètres pour réduire l'erreur. Ces modèles sont utilisés pour la reconnaissance d'images, la traduction automatique, la prédiction ou la génération de texte.

D'autres méthodes existent, comme les arbres de décision, les forêts aléatoires ou les machines à vecteurs de support. Elles montrent que l'apprentissage peut prendre des formes variées, adaptées à des problèmes différents.

9.4.6 Les systèmes de recommandation : personnaliser l'information

Les plateformes de vidéo, de musique ou de commerce utilisent des algorithmes pour proposer des contenus adaptés à chaque utilisateur. Ces systèmes analysent les comportements, les préférences et les similarités entre utilisateurs. Ils reposent sur des techniques de filtrage collaboratif, qui comparent les profils, ou sur des modèles de contenu, qui analysent les caractéristiques des objets proposés.

Ces algorithmes doivent être rapides, précis et capables de s'adapter en temps réel. Ils illustrent la manière dont les données massives peuvent être exploitées pour personnaliser les services.

9.4.7 Le traitement du langage et des images

Les algorithmes modernes permettent d'analyser des textes, de comprendre des phrases, de reconnaître des objets dans des images ou de détecter des sons. Ils reposent sur des modèles statistiques, des réseaux de neurones profonds et des techniques d'optimisation avancées.

Dans le traitement du langage, les modèles de type transformeur ont révolutionné le domaine. Ils utilisent des mécanismes d'attention pour analyser les relations entre les mots. Dans la vision par ordinateur, les réseaux convolutifs extraient des motifs visuels à différents niveaux de détail.

Ces algorithmes montrent comment des méthodes mathématiques et informatiques peuvent reproduire certaines capacités humaines.

9.4.8 Conclusion du chapitre

Les algorithmes avancés présentés dans ce chapitre constituent l'infrastructure invisible du numérique moderne. Ils organisent l'information, sécurisent les échanges, compressent les données, apprennent à partir d'exemples et personnalisent les services. Leur diversité reflète la variété des problèmes qu'ils résolvent. Leur étude montre comment les idées fondamentales de l'algorithmique se combinent pour répondre à des besoins réels, à grande échelle. Le chapitre suivant conclura l'ouvrage en présentant les perspectives, les enjeux et les acteurs qui façonnent l'avenir des algorithmes.

9.5 Bilan et perspectives : les algorithmes dans le monde contemporain

Les algorithmes sont devenus un élément essentiel du fonctionnement des sociétés modernes. Ils organisent l'information, automatisent des tâches, soutiennent la recherche scientifique et influencent de nombreuses décisions. Leur présence est souvent invisible, mais leur impact est considérable.

Ce chapitre propose un bilan des notions abordées dans l'ouvrage et présente les perspectives qui se dessinent pour les années à venir. Il met également en lumière les acteurs — entreprises, laboratoires, institutions — qui façonnent l'évolution de l'algorithmique.

9.5.1 Un bilan : ce que les algorithmes apportent

L'étude des algorithmes montre qu'ils ne sont pas seulement des outils techniques. Ils constituent une manière de penser les problèmes, de structurer les solutions et d'organiser les données. Leur force réside dans leur capacité à transformer une idée en une méthode précise, reproductible et efficace. Ils permettent de traiter des volumes d'information que l'esprit humain ne pourrait pas gérer seul.

Les algorithmes simples introduisent les principes fondamentaux : parcours, comparaison, accumulation. Les algorithmes du lycée montrent comment ces idées se combinent pour traiter des structures plus riches. Les algorithmes du supérieur révèlent la puissance des techniques avancées, capables de résoudre des problèmes complexes. Enfin, les algorithmes modernes — moteurs de recherche, cryptographie, apprentissage automatique — illustrent la manière dont ces méthodes s'intègrent dans des systèmes à grande échelle.

Ce parcours montre que l'algorithmique est à la fois une science, une technique et un langage. Elle permet de comprendre comment les machines résolvent des problèmes et comment concevoir des solutions adaptées aux besoins réels.

9.5.2 Les grandes tendances actuelles

Les algorithmes évoluent en permanence. Plusieurs tendances marquent aujourd'hui leur développement. L'augmentation des volumes de données pousse à concevoir des méthodes capables de traiter l'information en continu, de manière distribuée et à grande échelle. L'intelligence artificielle transforme la manière de concevoir les algorithmes : au lieu de définir explicitement chaque étape, on construit des modèles capables d'apprendre à partir d'exemples.

La recherche d'efficacité énergétique devient également un enjeu majeur. Les centres de calcul consomment une quantité importante d'énergie, et les algorithmes doivent être optimisés pour réduire leur impact environnemental. Les méthodes d'optimisation, les architectures spécialisées et les modèles plus sobres jouent un rôle croissant.

Enfin, la question de la transparence et de l'éthique prend une importance nouvelle. Les algorithmes influencent des décisions sensibles, et il devient indispensable de garantir leur équité, leur explicabilité et leur conformité aux règles. Cette exigence concerne autant les algorithmes traditionnels que les modèles d'apprentissage automatique.

Le saviez-vous ?

Paradoxe de Jevons

Le paradoxe de Jevons, formulé en 1865, décrit un mécanisme contre-intuitif : améliorer l'efficacité d'une technologie peut accroître — et non réduire — la consommation totale de la ressource qu'elle utilise.

Ce principe, né à l'ère du charbon, trouve aujourd'hui une résonance saisissante dans le développement fulgurant de l'intelligence artificielle.

Le paradoxe : une intuition née du charbon

William Stanley Jevons observe au XIX^{e} siècle que les innovations de James Watt, rendant la machine à vapeur plus performante, n'ont pas diminué la consommation de charbon britannique. Au contraire, en abaissant le coût d'usage de la vapeur, elles ont multiplié les applications industrielles, provoquant une hausse globale de la demande en charbon.

Le paradoxe de Jevons désigne précisément ce phénomène : lorsque l'efficacité augmente, la consommation totale peut croître si les usages se multiplient plus vite que les économies réalisées .

Une mécanique économique intemporelle

Ce mécanisme est un cas extrême de l'« effet rebond » : les gains d'efficacité, en réduisant le coût effectif d'utilisation d'une ressource, élargissent les marchés et stimulent de nouveaux comportements.

L'histoire industrielle regorge d'exemples similaires : voitures plus économes mais davantage utilisées, éclairage LED plus efficace mais installé en plus grande quantité, stockage cloud moins cher entraînant une explosion des volumes conservés .

L'IA : un nouveau terrain d'expression du paradoxe

L'intelligence artificielle illustre aujourd'hui ce phénomène avec une acuité particulière. Les progrès rapides des modèles, des algorithmes et des infrastructures ont réduit le coût unitaire de l'inférence ou du stockage, rendant l'IA plus accessible. Résultat : la demande explose, tant en calcul qu'en données.

- La montée en puissance de l'IA a provoqué une pression massive sur les semi-conducteurs, dont les prix ont bondi dans certains segments, notamment la DRAM, avec des hausses atteignant 171 % en 2025 .
- Les centres de données mondiaux consommaient déjà 415 TWh en 2024, et pourraient dépasser 945 TWh d'ici 2030, soit l'équivalent de la consommation annuelle du Japon .
- Malgré des innovations visant à réduire l'empreinte énergétique — comme l'algorithme TurboQuant de Google, qui compresse les vecteurs de données pour alléger les modèles — la demande globale continue de croître plus vite que l'efficacité technique .

Une dynamique qui dépasse l'énergie

Le paradoxe de Jevons appliqué à l'IA ne concerne pas seulement l'électricité. Il éclaire aussi les transformations économiques : plus l'IA devient performante et abordable, plus elle stimule de nouveaux usages, de la traduction automatisée aux services juridiques assistés, en élargissant des marchés auparavant limités.

Les dirigeants de la tech y voient un cadre pour comprendre pourquoi l'IA pourrait créer plus d'activité — et parfois plus d'emplois — qu'elle n'en détruit, comme la machine à tisser l'avait fait en son temps .

Conclusion

Le paradoxe de Jevons rappelle une vérité simple : l'efficacité ne garantit pas la sobriété. À l'ère de l'IA, cette leçon historique devient un enjeu stratégique majeur, tant pour la gestion des ressources que pour la compréhension des dynamiques économiques et sociétales que ces technologies déclenchent.

9.5.3 Les défis à venir

L'avenir de l'algorithmique est marqué par plusieurs défis. Le premier concerne la maîtrise de la complexité. Certains problèmes restent difficiles à résoudre, même avec des machines puissantes. Les méthodes d'approximation, les heuristiques et les algorithmes probabilistes continueront de jouer un rôle important.

Le second défi concerne l'intégration de l'intelligence artificielle dans des systèmes critiques. Les modèles doivent être fiables, robustes et compréhensibles. Leur comportement doit pouvoir être vérifié, ce qui nécessite de nouvelles approches théoriques et pratiques.

Le troisième défi concerne la sécurité. Les algorithmes de cryptographie doivent s'adapter aux progrès des machines, notamment à l'arrivée potentielle de l'informatique quantique. De nouvelles méthodes devront être développées pour garantir la confidentialité et l'intégrité des données.

Enfin, la formation devient un enjeu central. Comprendre les algorithmes n'est plus réservé aux spécialistes. Les citoyens, les décideurs et les professionnels de nombreux domaines doivent acquérir une culture algorithmique pour comprendre les outils qu'ils utilisent.

Le saviez-vous ?

Les réseaux de neurones basés sur l'ADN (ou « DNA neural networks ») relèvent de l'informatique moléculaire et de la biologie synthétique. Aujourd'hui, ils existent surtout à l'état de preuve de concept en laboratoire.

Le principe consiste à utiliser des brins d'ADN comme unités de calcul : des réactions biochimiques (hybridation, déplacement de brin) jouent le rôle de neurones et de connexions. Des équipes ont déjà démontré des circuits capables de classification simple, de reconnaissance de motifs ou de prise de décision rudimentaire, parfois directement dans des milieux biologiques.

L'intérêt majeur est leur fonctionnement en parallèle massif, leur très faible consommation d'énergie et leur compatibilité potentielle avec des environnements biologiques (diagnostic in vivo, nanomédecine). Par exemple, certains systèmes peuvent détecter des signatures moléculaires de maladies et produire une réponse chimique.

Cependant, les limites restent importantes : lenteur des réactions (minutes à heures), difficulté de mise à l'échelle, bruit chimique, faible précision et manque d'outils de programmation standardisés. De plus, l'entraînement de ces réseaux est encore très rudimentaire comparé aux réseaux neuronaux classiques.

En résumé, le domaine est prometteur mais encore expérimental. À court terme, les applications les plus réalistes concernent des biosenseurs intelligents plutôt que des systèmes de calcul généralistes.

9.5.4 Les acteurs qui façonnent l'algorithmique

Le développement des algorithmes repose sur une collaboration entre de nombreux acteurs. Les laboratoires de recherche produisent les avancées théoriques et les nouvelles méthodes. Les universités forment les ingénieurs et les chercheurs qui conçoivent les systèmes de demain. Les entreprises technologiques développent les outils, les plateformes et les infrastructures qui mettent les algorithmes en pratique.

Certaines entreprises jouent un rôle majeur dans l'évolution des algorithmes modernes, notamment dans les domaines du web, de la recherche d'information, de l'intelligence artificielle ou du cloud. Les institutions publiques interviennent également pour encadrer l'usage des algorithmes, garantir la protection des données et définir des règles éthiques.

L'algorithmique est ainsi un domaine où se rencontrent la science, l'industrie et la société. Son évolution dépend autant des avancées techniques que des choix collectifs.

9.5.5 Pour aller plus loin : quelques références essentielles

Pour approfondir l'étude des algorithmes, plusieurs ouvrages offrent des perspectives complémentaires.

Le *Que sais-je ?* **Les Algorithmes** d'Aurélie Jean propose une introduction concise et rigoureuse.

L'ouvrage **Algorithmes** de Franck Ebel et Sébastien Rohaut offre une approche pédagogique, adaptée à l'enseignement. Ces références permettent de prolonger la réflexion et d'explorer des aspects plus spécialisés.

Voir en Annexe une bibliographie plus complète.

9.5.6 Conclusion générale

Les algorithmes sont au cœur du fonctionnement du monde numérique. Leur étude permet de comprendre comment les machines traitent l'information, comment les solutions sont conçues et comment les décisions automatisées sont prises.

Ce livre a présenté les fondements, les méthodes et les enjeux de l'algorithmique, depuis les algorithmes simples jusqu'aux systèmes avancés.

L'avenir de ce domaine dépendra de notre capacité à concevoir des méthodes efficaces, transparentes et responsables. Les algorithmes continueront d'évoluer, mais leur rôle restera central dans la construction des technologies de demain.

10 L'empire des algorithmes : comprendre ce qui façonne l'IA moderne

L'intelligence artificielle moderne repose sur un ensemble d'algorithmes dont la diversité, longtemps sous-estimée, façonne aujourd'hui une révolution silencieuse.

Comprendre l'IA exige d'abord de cartographier ses fondations : les axes qui permettent de classer les algorithmes, les huit grandes familles qui structurent le domaine, et les nouveaux types d'IA — générative, multimodale, agentive — qui émergent à leur croisement. Mais cette architecture technique ne suffit plus. L'actualité montre des percées fulgurantes, des tensions entre approches symboliques et neuronales, et des systèmes qui dépassent parfois leurs concepteurs. Enfin, derrière ces avancées se profilent des enjeux majeurs : énergie, responsabilité, géopolitique, emploi, et même les fragilités mentales des modèles.

Ce chapitre explore ces trois dimensions pour comprendre ce qui façonne réellement l'IA contemporaine.

10.1 Classification des algorithmes

Classer les algorithmes est une ambition aussi ancienne que l'informatique elle-même. Très tôt, on a cherché à regrouper les méthodes en grandes familles — algorithmes symboliques, statistiques, d'apprentissage, d'optimisation, etc. — afin de mieux comprendre leurs principes et leurs usages. Cette approche, que l'on peut qualifier de **taxonomie statique**, présente un avantage évident : elle offre une lecture simple, structurée, et historiquement cohérente des grandes étapes du

développement des techniques algorithmiques. Chaque famille correspond à un ensemble relativement identifiable de méthodes, de concepts et d'applications.

Cependant, à mesure que les algorithmes ont évolué, cette classification a montré ses limites. Les systèmes contemporains, en particulier, ne se laissent plus enfermer dans une seule catégorie. Ils combinent apprentissage, optimisation, représentation et interaction avec leur environnement. Un même algorithme peut être à la fois probabiliste, appris à partir de données, et fondé sur des mécanismes de recherche. La frontière entre les familles devient alors floue, voire artificielle.

C'est pourquoi il devient nécessaire de passer d'une simple taxonomie à une **cartographie conceptuelle**. Plutôt que de ranger les algorithmes dans des cases exclusives, il s'agit de les situer dans un espace défini par plusieurs dimensions fondamentales : leur rapport à l'incertitude, leur mode de représentation de l'information, leur stratégie de résolution, et leur capacité à apprendre. Dans cette perspective, un algorithme n'appartient plus à une seule famille, mais se caractérise par une position relative selon ces axes.

Ce changement de point de vue ne remplace pas la classification traditionnelle ; il la prolonge et l'éclaire. Il permet de mieux comprendre les continuités entre approches, les hybridations contemporaines, et, plus largement, les différentes manières dont les systèmes algorithmiques transforment l'information pour produire des décisions, des prédictions ou des actions.

10.1.1 Les axes d'une classification multi-dimensionnelle des algorithmes

Les cinq dimensions fondamentales :

Incertitude : déterministe ↔ probabiliste
Représentation : symbolique ↔ numérique
Stratégie : exacte / heuristique / optimisation
Apprentissage : oui ↔ non
Modèle : explicite ↔ implicite

Pour dépasser les limites d'une classification en familles distinctes, il est nécessaire d'adopter une approche plus fondamentale, fondée non plus sur des catégories, mais sur des **dimensions structurantes**. Un algorithme peut alors être décrit comme une position dans un espace conceptuel défini par plusieurs axes. Quatre dimensions apparaissent particulièrement pertinentes pour caractériser la manière dont les algorithmes traitent l'information : l'incertitude, la représentation, la stratégie de résolution et la capacité d'apprentissage.

Le premier axe concerne le **rapport à l'incertitude**. Certains algorithmes opèrent dans un cadre strictement déterministe : à une entrée donnée correspond une sortie unique, sans ambiguïté ni variabilité. D'autres, au contraire, intègrent explicitement l'incertitude en manipulant des probabilités ou des distributions. Cette distinction est fondamentale, car elle reflète deux visions différentes du monde : l'une où tout est, en principe, connaissable avec précision, l'autre où l'information est intrinsèquement partielle, bruitée ou incertaine. Entre ces deux pôles, on trouve une grande variété de situations hybrides, où des mécanismes déterministes coexistent avec des estimations probabilistes.

Le deuxième axe porte sur la **représentation de l'information**. Dans une approche symbolique, les données sont structurées sous forme d'objets, de relations et de règles explicites. L'algorithme manipule alors des entités porteuses de sens, selon des opérations logiques. À l'opposé, les approches numériques représentent l'information sous forme de vecteurs, de matrices ou de paramètres continus. Le sens n'est plus directement lisible ; il est encodé de manière distribuée dans les structures du modèle. Cette distinction est centrale, car elle conditionne à la fois l'interprétabilité des systèmes et leur capacité à traiter des données complexes et massives.

Le troisième axe concerne la **stratégie de résolution**. Certains algorithmes visent une solution exacte, obtenue par un raisonnement exhaustif ou garanti. D'autres adoptent des approches heuristiques, qui privilégient l'efficacité au détriment de la certitude, en exploitant des règles empiriques ou des approximations. Enfin, de nombreux algorithmes s'inscrivent dans une logique d'optimisation : ils cherchent à maximiser ou minimiser une fonction-objectif, souvent dans des espaces de grande dimension. Ces trois modalités ne s'excluent pas mutuellement, mais traduisent des compromis différents entre précision, coût de calcul et faisabilité.

Le quatrième axe, enfin, distingue les algorithmes selon leur **capacité à apprendre**. Certains systèmes sont entièrement spécifiés à l'avance : leur comportement est déterminé par des règles ou des modèles définis par un concepteur. D'autres, au contraire, ajustent leurs paramètres ou leurs structures à partir de données ou d'interactions avec un environnement. L'apprentissage introduit une dynamique essentielle : l'algorithme ne se contente plus d'exécuter, il s'adapte. Cette dimension est aujourd'hui centrale dans de nombreux domaines, mais elle coexiste toujours avec des approches non-apprenantes, notamment lorsque la stabilité, la transparence ou la garantie de performance sont requises.

Le cinquième axe distingue les algorithmes selon qu'ils reposent sur un **modèle explicite** ou **implicite**. Un modèle explicite rend visibles ses structures internes : variables, relations, règles ou paramètres sont interprétables et manipulables directement. Il permet l'analyse, la justification et la modification consciente du comportement algorithmique. À l'inverse, un modèle implicite encode sa logique dans une forme opaque : représentations distribuées, paramètres massifs ou dynamiques émergentes difficiles à isoler. Cette dimension éclaire le degré de transparence, de contrôlabilité et d'auditabilité (traçabilité) d'un algorithme, complétant ainsi les quatre axes fondamentaux déjà définis.

Pris ensemble, ces cinq axes ne constituent pas une nouvelle classification figée, mais un cadre de description**.** Ils permettent de situer un algorithme non pas dans une case, mais dans un espace, et de mieux comprendre les choix implicites qui sous-tendent sa conception.

10.1.2 Les huit grandes familles d'algorithmes

Les huit familles:

Algorithmes symboliques
Deep learning
Apprentissage par renforcement
Algorithmes évolutionnaires
Algorithmes de recherche et d'optimisation
Algorithmes hybrides
Apprentissage automatique
Algorithmes statistiques

Algorithmes symboliques: penser, c'est manipuler des règles

Dans cette première famille, la plus classique et la plus connue, l'information est structurée comme un langage : des objets, des relations, des propositions. Traiter l'information consiste à appliquer des règles explicites pour transformer des énoncés en d'autres énoncés. Un système ne découvre rien par lui-même ; il déploie ce qui est déjà contenu dans les règles et les faits qu'on lui a donnés.

Sa force est la rigueur : chaque conclusion peut être retracée, expliquée, vérifiée. Mais cette transparence est aussi sa limite. Dès que le réel déborde les catégories prévues — ambiguïté, bruit, incertitude — le système se fragilise. Contrairement aux approches ultérieures, il ne tolère pas l'approximation : il raisonne, mais n'apprend pas.

Note: Il faut d'ailleurs distinguer symbolique et déterministe : le symbolique décrit la forme de la représentation, alors que le déterminisme décrit la nature de l'exécution ; un algorithme peut être symbolique sans être strictement déterministe, et déterministe sans être symbolique.

Algorithmes statistiques : penser, c'est estimer

Ici, l'information cesse d'être strictement vraie ou fausse. Elle devient incertaine, distribuée, probabiliste. Traiter l'information consiste à ajuster des modèles qui capturent des régularités observées dans les données. On ne cherche plus à déduire, mais à estimer ce qui est probable.

Cette famille introduit une souplesse fondamentale : elle accepte le bruit, les exceptions, les variations. Mais elle change aussi la nature du savoir : ce que l'on obtient n'est pas une explication formelle, mais une mesure de plausibilité. Là où le symbolique affirme, le statistique pondère. Là où le premier exige des règles, le second exploite des fréquences.

Apprentissage automatique : penser, c'est généraliser

Avec l'apprentissage automatique, une bascule s'opère. L'information n'est plus seulement analysée : elle est utilisée pour construire un modèle. L'algorithme apprend à partir d'exemples, en ajustant ses paramètres pour reproduire des comportements observés.

Ce qui le distingue des approches statistiques classiques, c'est son orientation vers la généralisation opérationnelle : il ne s'agit pas seulement de décrire les données, mais de produire une fonction capable d'agir sur de nouvelles situations. Le savoir n'est plus explicitement formulé, il est incorporé dans un modèle.

Contrairement au symbolique, il n'a pas besoin de règles écrites ; contrairement au purement statistique, il est conçu pour être utilisé en prédiction ou en décision.

Deep learning : penser, c'est représenter

Le deep learning radicalise cette idée en transformant la manière même dont l'information est encodée. Les données brutes sont converties en représentations internes hiérarchiques, apprises automatiquement.

Ce qui distingue cette famille, ce n'est pas seulement sa puissance, mais sa capacité à fabriquer ses propres niveaux d'abstraction. Là où les autres méthodes nécessitaient souvent des variables ou des caractéristiques définies à l'avance, ici celles-ci émergent du processus d'apprentissage.

L'information devient une structure distribuée dans un grand nombre de paramètres. Le prix à payer est une perte de lisibilité : on ne sait plus facilement "pourquoi" le système décide, seulement qu'il le fait efficacement.

Apprentissage par renforcement : penser, c'est agir

Dans cette famille, l'information ne préexiste pas entièrement : elle est produite par l'interaction avec un environnement. L'algorithme apprend en essayant, en échouant, en réussissant partiellement, et en ajustant ses actions en fonction des récompenses obtenues.

Ce qui le distingue profondément des autres approches, c'est que le problème n'est plus une simple prédiction, mais une suite de décisions dépendantes les unes des autres. L'information pertinente n'est pas donnée sous forme de données statiques ; elle émerge d'une boucle perception–action. Ici, comprendre signifie apprendre quoi faire, et quand le faire.

Algorithmes évolutionnaires : penser, c'est explorer

Les algorithmes évolutionnaires adoptent une perspective encore différente : ils ne cherchent ni à raisonner, ni à apprendre directement, mais à explorer un espace de solutions par variation et sélection. L'information est traitée collectivement, à travers une population de candidats qui évoluent au fil du temps.

Ce qui les distingue, c'est l'absence de modèle explicite ou de gradient : le progrès vient de la diversité et de la sélection, non d'une optimisation dirigée pas à pas. Ils sont particulièrement adaptés aux problèmes où l'on ne sait pas comment guider la recherche autrement. Là où d'autres approches convergent, celles-ci tâtonnent — mais de manière organisée.

Algorithmes de recherche et d'optimisation : penser, c'est parcourir

Dans cette famille, traiter l'information consiste à explorer méthodiquement un espace structuré de possibilités. Chaque état mène à d'autres états, et l'algorithme cherche un chemin optimal selon un critère donné.

Contrairement aux approches d'apprentissage, il n'y a pas nécessairement d'adaptation à partir de données : tout repose sur la manière d'explorer efficacement. Ce qui distingue ces algorithmes, c'est leur capacité à garantir (parfois) l'optimalité ou à encadrer précisément la qualité de la solution. Ils excellent lorsque le problème est bien défini, mais deviennent vite limités lorsque l'espace explose ou que l'information est incomplète.

Algorithmes hybrides : penser, c'est combiner

Les systèmes contemporains les plus performants ne relèvent plus d'une seule logique. Ils combinent plusieurs manières de traiter l'information : apprendre des représentations, explorer des données, planifier des actions, intégrer de l'incertitude.

Ce qui les distingue n'est pas une technique particulière, mais leur capacité à faire dialoguer plusieurs paradigmes. Par exemple, un système peut apprendre à évaluer une situation (deep learning), simuler des scénarios (recherche), et ajuster ses décisions par interaction (renforcement). L'information circule alors entre différentes formes, chacune exploitée selon ses forces. Cette hybridation marque une étape importante : elle suggère que l'intelligence ne repose pas sur une seule manière de traiter l'information, mais sur leur articulation.

Bilan

Ces huit familles ne sont pas seulement des outils différents : elles incarnent des manières distinctes de donner du sens à l'information. Certaines privilégient la règle, d'autres la probabilité, d'autres encore l'expérience ou l'exploration. Leur succession historique n'est pas un simple progrès technique, mais un élargissement progressif de ce que l'on considère comme exploitable — et comme intelligible.

10.1.3 Croisement familles & axes de classement

Profil typique (axes)	Spécificité	Exemples
Algorithmes symboliques		
Déterministe **Symbolique** **Exact (souvent)** **Sans apprentissage** **Modèle explicite (règles logiques)**	Raisonnement logique, manipulation de symboles, transparence. Ils sont les seuls à traiter l'information comme du sens explicite manipulé par des règles.	**Système expert médical**: applique des règles logiques codées par des spécialistes pour diagnostiquer des maladies ou recommander des traitements à partir de symptômes fournis. **Planificateur d'IA (STRIPS)**: construit automatiquement une suite d'actions en partant d'un état initial et d'un but, en utilisant des opérateurs symboliques décrivant leurs préconditions et effets.
Algorithmes statistiques		

Probabiliste **Numérique** **Optimisation (estimation)** **Peu ou pas d'apprentissage au sens moderne** **Modèle explicite (paramètres interprétables)**	Régression, modèles linéaires, estimation paramétrique. Ils introduisent l'incertitude comme objet central.	**Régression linéaire** (moindres carrés): ajuste une droite (ou un hyperplan) aux données en minimisant la somme des carrés des écarts entre les prédictions et les valeurs observées. **Méthodes de Monte-Carlo** : techniques qui utilisent des tirages aléatoires répétés pour approximer des quantités difficiles à calculer analytiquement, comme des intégrales, des distributions ou des espérances. **Modèle bayésien** : méthode qui met à jour ses croyances en combinant données nouvelles et connaissances préalables selon les règles de la probabilité.
Apprentissage automatique (Machine Learning classique)		
Souvent probabiliste **Numérique** **Optimisation** **Apprentissage** **Implicite ou explicite selon le modèle**	Apprendre une fonction à partir de données.	**SVM (Support Vector Machine):** algorithme supervisé qui cherche à séparer les données en classes en trouvant l'hyperplan optimal, celui qui maximise la marge entre les catégories **Random Forest:** ensemble d'arbres de décision entraînés de manière aléatoire. Chaque arbre vote pour une prédiction, et la forêt combine ces votes pour obtenir un résultat plus fiable.
Deep Learning		
Probabiliste (souvent implicite) **Numérique (représentations distribuées)** **Optimisation (gradient)** **Apprentissage** **Modèle implicite (poids appris)**	Représentations distribuées, non interprétables, optimisation par gradient. Apprend les représentations elles-mêmes, pas seulement la décision.	**Transformers (GPT, BERT):** ils utilisent une architecture fondée sur l'attention, qui leur permet de comprendre et générer du texte en tenant compte des relations entre tous les mots d'une séquence, même lointains. **CNN (vision):** ils analysent les images en détectant progressivement des motifs visuels — bords, textures, formes — grâce à des **filtres convolutifs** qui extraient des caractéristiques de plus en plus complexes.
Apprentissage par renforcement		
Probabiliste (environnement incertain) **Numérique** **Optimisation séquentielle** **Apprentissage**	Apprentissage par interaction, exploration/exploitation. Apprend quoi faire, pas juste quoi prédire.	**Q-learning**: apprend à choisir la meilleure action dans chaque situation en mettant à jour une table de valeurs Q selon les récompenses reçues, jusqu'à découvrir une stratégie optimale par essai-erreur. **AlphaZero**: apprend à jouer à un jeu en s'auto-entraînant : il combine un réseau de

Modèle implicite ou explicite selon l'agent		neurones et la recherche Monte-Carlo pour s'améliorer uniquement en jouant contre lui-même, sans données humaines.
Algorithmes évolutionnaires		
Plutôt déterministe (mais stochastique en pratique) **Numérique** **Optimisation (sans gradient)** **Pas d'apprentissage au sens classique** **Modèle implicite**	Évolution de populations, mutation, sélection. Exploration par variation + sélection, sans modèle explicite.	**Algorithmes génétiques**: ils optimisent une solution en faisant évoluer une population de candidats par sélection, croisement et mutation, à la manière d'un processus d'évolution biologique artificielle **Neuroevolution**: applique les principes évolutifs pour faire évoluer automatiquement la structure et/ou les poids de réseaux de neurones, sans utiliser de rétropropagation.
Algorithmes de recherche et d'optimisation		
Déterministe (souvent) **Symbolique ou numérique** **Exact OU heuristique** **Sans apprentissage** **Modèle explicite**	Recherche d'un optimum ou d'un chemin, heuristiques. Explorer un espace pour trouver une solution.	**A*:** trouve le chemin optimal en combinant le coût déjà parcouru et une heuristique qui estime le coût restant, ce qui guide efficacement la recherche vers la solution. **Branch and Bound** explore l'espace des solutions en éliminant systématiquement les branches dont le meilleur cas possible ne peut pas battre la meilleure solution connue. Le **Monte-Carlo Tree Search (MTCS)** construit progressivement un arbre de décision en simulant aléatoirement des parties pour estimer la valeur des actions et choisir celles qui mènent aux meilleurs résultats.
Algorithmes hybrides		
Tous les axes possibles Combinaisons	Ce ne sont pas une position, mais une architecture multi-positions.	**AlphaGo** : combine un réseau de neurones, la recherche Monte-Carlo et l'apprentissage par renforcement pour évaluer les positions, guider l'exploration et s'améliorer en jouant contre lui-même. **MuZero**: apprend un modèle implicite des dynamiques du jeu et l'utilise pour planifier via MCTS, sans connaître à l'avance les règles exactes de l'environnement.

Tableau selon profil

Famille d'algorithmes	Déterministe / Probabiliste	Nature: symbolique / numérique)	Apprenant / Non-apprenant	Modèle explicite / implicite
Algorithmes symboliques	Généralement déterministes	Symbolique	Non apprenants	Modèle explicite (règles, logique)
Algorithmes statistiques	Probabilistes	Numérique	Apprenant	Modèle explicite (paramètres interprétables)
Apprentissage automatique (ML)	Probabiliste	Numérique	Apprenant	Implicite ou explicite selon le modèle
Deep learning	Probabiliste (stochastique)	Numérique	Apprenant	Modèle implicite (poids appris)
Apprentissage par renforcement	Probabiliste	Numérique (parfois symbolique)	Apprenant	Modèle implicite ou explicite selon l'agent
Algorithmes évolutionnaires	Stochastiques	Numérique	Apprenant (optimisation)	Modèle implicite
Algorithmes de recherche et d'optimisation	Souvent déterministes	Symbolique ou numérique	Non apprenants	Modèle explicite
Algorithmes hybrides	Mixte	Mixte	Apprenant	Mixte

10.1.4 Les types d'IA et leurs appartenances aux huit familles d'algorithmes

L'expression « types d'IA » circule abondamment dans le débat public : IA générative, IA explicable, IA symbolique, IA autonome, IA multimodale… Ces qualificatifs donnent l'impression d'une taxonomie bien établie, alors qu'ils désignent en réalité des **capacités**, des **propriétés** ou des **usages**, et non des familles d'algorithmes.

Pour comprendre ce que recouvrent ces appellations, il faut les replacer dans le cadre plus rigoureux des huit grandes familles d'algorithmes qui structurent aujourd'hui le champ de l'intelligence artificielle.

- Chaque famille décrit une **mécanique interne**, un mode de représentation ou une stratégie de calcul.
- Les qualificatifs, eux, décrivent ce que l'IA **fait**, **produit** ou **permet**.

Ce chapitre propose une cartographie claire : pour chaque type d'IA, nous identifions les familles d'algorithmes capables de le réaliser.

1. IA générative : produire du contenu

L'IA générative désigne les systèmes capables de **créer** du texte, des images, du son ou des données. Ce n'est pas une famille, mais une **capacité**.

Plusieurs familles peuvent la porter :

Deep learning : GAN, VAE, Transformers génératifs.

Un **GAN** est un réseau antagoniste génératif. Il repose sur deux réseaux neuronaux qui s'affrontent : un générateur, qui produit des données (images, sons, etc.) et un discriminateur, qui tente de distinguer les données réelles des données générées. Le générateur apprend à

tromper le discriminateur, ce qui conduit à des productions de plus en plus réalistes. C'est l'une des premières architectures à avoir permis la génération d'images convaincantes.

Un **VAE** est un autoencodeur variationnel.Il apprend à représenter les données dans un espace latent probabiliste, puis à les reconstruire. Deux étapes :

- encodeur : compresse les données en une distribution latente;
- décodeur : reconstruit ou génère de nouvelles données à partir de cette distribution.

Les VAE sont plus stables que les GAN, mais produisent souvent des résultats plus lissés.

Les **Transformers** sont une architecture fondée sur le mécanisme d'attention, qui permet au modèle de repérer quelles parties d'une séquence sont pertinentes pour prédire la suite. Les versions génératives (GPT, T5, diffusion textuelle, etc.) :

- apprennent à prédire le prochain élément d'une séquence;
- peuvent générer du texte, du code, des images ou des sons;
- sont aujourd'hui la base des grands modèles de langage et des IA génératives modernes.

Les Transformers ont supplanté les GAN et VAE dans de nombreux domaines grâce à leur capacité d'échelle et leur polyvalence.

Algorithmes statistiques : modèles probabilistes génératifs (mixtures, modèles bayésiens).

Algorithmes évolutionnaires : génération par mutation et recombinaison.

Apprentissage automatique : modèles génératifs classiques (naïve Bayes, HMM).

L'IA générative est donc un usage transversal, rendu possible par des approches très différentes.

2. IA explicable : comprendre le raisonnement

L'IA explicable (ou interprétable) désigne la capacité à justifier un résultat ou à retracer un raisonnement.

Elle peut s'appuyer sur :

Algorithmes symboliques : explicables par construction.

Algorithmes statistiques : régression, arbres de décision.

Apprentissage automatique : modèles à règles, forêts interprétables.

Deep learning : explicabilité ajoutée (LIME, SHAP, cartes d'attention).

L'explicabilité est donc une **propriété**, pas une famille.

3. IA multimodale : combiner plusieurs types de données

Une IA est multimodale lorsqu'elle traite simultanément texte, image, son, vidéo ou données structurées.

Les familles impliquées sont :

Deep learning : architectures multimodales (CLIP, Flamingo, Gemini).

CLIP — Contrastive Language; Image Pretraining: modèle développé pour relier texte et image. Son principe :

- un encodeur d'images transforme une image en vecteur;
- un encodeur de texte transforme une phrase en vecteur.

L'entraînement apprend à rapprocher les vecteurs correspondant à la même image et à la même description, et à éloigner les autres

CLIP ne génère pas d'images, mais il comprend la relation texte-image. Il sert de base à de nombreux systèmes génératifs (comme Stable Diffusion) pour guider la création d'images à partir de texte.

Flamingo — modèle multimodal à contexte long: architecture conçue pour raisonner sur des séquences mêlant texte et images, sans réentraînement massif. Ses caractéristiques :

- un backbone linguistique (un grand modèle de langage);
- un module visuel qui encode les images;
- un mécanisme d'attention croisée permettant au texte et aux images d'interagir.

La capacité de répondre à des questions sur des images, de décrire des scènes ou d'interpréter des séquences visuelles. Flamingo est optimisé pour le few-shot multimodal : il peut apprendre une nouvelle tâche à partir de quelques exemples seulement (few).

Gemini — modèle multimodal intégré: famille de modèles conçus dès le départ comme nativement multimodaux. Contrairement à d'autres architectures qui ajoutent la vision à un modèle textuel, Gemini :

- traite texte, image, audio, vidéo dans un même espace de représentation;
- utilise une architecture de type Transformer adaptée à plusieurs modalités;
- peut raisonner sur des documents complexes (graphes, schémas, images annotées);
- combine perception et langage dans une même structure.

C'est une approche « tout-en-un », où la multimodalité n'est pas un module ajouté, mais une propriété fondamentale du modèle.

Apprentissage automatique : modèles combinant plusieurs sources.

Algorithmes hybrides : fusion symbolique-neurale.

La multimodalité est une **capacité émergente**, rendue possible par l'intégration de représentations hétérogènes.

4. IA auto-apprenante : s'adapter par l'expérience

Une IA auto-apprenante ajuste son comportement à partir de données ou d'interactions. Elle relève de :

Apprentissage automatique : apprentissage supervisé et non supervisé.

Deep learning : réseaux neuronaux entraînés sur de grands volumes de données.

Apprentissage par renforcement : apprentissage par essais et erreurs.

Algorithmes évolutionnaires : adaptation par sélection.

L'auto-apprentissage est une dynamique, non une architecture.

5. IA symbolique : manipuler des concepts

L'IA symbolique repose sur la manipulation de règles, concepts, relations et structures logiques. Elle correspond directement à :

Algorithmes symboliques

Algorithmes hybrides lorsqu'ils combinent symbolique et réseaux neuronaux (IA neurosymbolique)

Ici, le qualificatif renvoie bien à une famille, mais il peut aussi désigner une approche intégrée dans des systèmes plus larges.

6. IA optimisatrice : chercher la meilleure solution

Une IA optimisatrice vise à maximiser ou minimiser une fonction objectif. Elle peut s'appuyer sur :

Algorithmes de recherche et d'optimisation : heuristiques, branch-and-bound, méta-heuristiques.

Algorithmes évolutionnaires : optimisation par sélection naturelle.

Apprentissage par renforcement : maximisation de récompense.

Deep learning : optimisation par descente de gradient.

L'optimisation est un objectif, non une famille.

7. IA autonome : décider dans un environnement

Une IA autonome prend des décisions dans un environnement dynamique, souvent en temps réel. Elle repose sur :

Apprentissage par renforcement : agents, politiques, récompenses.

Algorithmes de recherche et d'optimisation : planification, recherche d'actions.

Algorithmes hybrides : perception neuronale + décision symbolique.

L'autonomie est une **fonction**, souvent associée à la robotique ou aux systèmes embarqués.

8. IA robuste : résister au bruit et à l'incertitude

Une IA robuste maintient ses performances malgré des perturbations. Elle peut être obtenue via :

Deep learning : modèles massifs, régularisation, entraînement adversarial.
Algorithmes statistiques : modèles probabilistes tolérants au bruit.
Apprentissage automatique : méthodes régularisées, ensembles.
Algorithmes hybrides : combinaisons redondantes.

La robustesse est une qualité, pas une famille.

9. IA probabiliste : représenter l'incertitude

Une IA probabiliste manipule des distributions, des croyances et des inférences. Elle s'appuie sur :

Algorithmes statistiques : modèles bayésiens, inférence.
Apprentissage automatique : modèles probabilistes graphiques.
Deep learning : réseaux bayésiens, diffusion probabiliste.
Algorithmes hybrides : logique probabiliste.

Ici encore, il s'agit d'un style de représentation, non d'une famille.

10. IA déterministe : même entrée, même sortie

Une IA déterministe ne comporte aucune part d'aléa dans son exécution. Elle peut relever de :

Algorithmes symboliques
Algorithmes de recherche et d'optimisation
Certaines méthodes d'apprentissage automatique

Le déterminisme décrit la nature de l'exécution, pas la structure du modèle.

Conclusion : familles internes, qualificatifs externes

Les huit familles d'algorithmes décrivent la **mécanique interne** d'un système d'IA : comment il représente l'information, comment il calcule, comment il apprend ou non.

Les qualificatifs courants — générative, explicable, multimodale, autonome, robuste — décrivent au contraire des capacités visibles, des propriétés fonctionnelles ou des usages.

Une même capacité peut être réalisée par plusieurs familles, et une même famille peut produire des IA très différentes selon le contexte.

Cette distinction est essentielle pour comprendre l'écosystème actuel de l'IA : derrière les étiquettes médiatiques se trouvent des architectures variées, parfois anciennes, parfois nouvelles, souvent combinées.

10.1.5 LLM et agents IA : une nouvelle étape dans l'histoire des techniques

LLM — Large Language Model — Modèle de langage de grande taille

Ce chapitre met en avant trois évolutions modernes majeures :

1. **L'intégration des LLM avec des outils externes** (navigateurs, bases de données, logiciels).
2. **L'émergence d'agents autonomes** capables de décomposer des tâches complexes.
3. **La nécessité d'un cadre pédagogique et éthique** pour intégrer ces systèmes dans l'éducation.

Une mutation discrète mais décisive

Lorsque les premiers grands modèles de langage ont émergé au début des années 2020, ils ont été accueillis comme une curiosité spectaculaire : des machines capables de produire du texte avec une fluidité inédite. Leur fonctionnement, fondé sur l'apprentissage statistique à très grande échelle, rappelait les grandes ruptures techniques du passé : l'apparition des moteurs de recherche, l'essor du web, ou encore la généralisation des interfaces graphiques. Mais, comme souvent dans l'histoire des technologies, l'innovation la plus visible n'était pas la plus déterminante. Le véritable basculement s'est produit plus tard, lorsque ces modèles ont cessé d'être de simples générateurs pour devenir des agents capables d'agir.

Cette transition n'a pas été immédiate. Les premiers LLM restaient confinés à un rôle d'oracle : ils répondaient, mais ne faisaient rien. Leur architecture, pourtant impressionnante, ne leur permettait ni de manipuler des données, ni d'exécuter des tâches, ni de vérifier leurs propres productions. Ils étaient comparables aux premières machines à calculer du XIX^{e} siècle : puissantes dans leur domaine, mais incapables d'interagir avec un environnement plus large.

La décennie suivante a vu apparaître une série d'innovations qui ont progressivement transformé ces modèles en systèmes opératoires. L'intégration d'outils externes — navigateurs, interprètes de code, bases de données — a joué un rôle décisif. Pour la première fois, une IA pouvait non seulement produire une réponse, mais aller la chercher, la vérifier, la corriger. Cette évolution rappelle l'introduction des périphériques dans l'histoire de l'informatique : le moment où l'ordinateur, jusque-là isolé, s'est mis à interagir avec le monde.

L'étape suivante a été l'apparition de modules de planification. Les ingénieurs ont compris que la génération linguistique ne suffisait pas : il fallait doter les modèles d'une capacité à structurer des actions, à anticiper des étapes, à évaluer des options. Ce tournant évoque l'invention des systèmes d'exploitation dans les années 1960 : une couche d'organisation qui transforme une machine puissante mais brute en un outil capable de gérer des processus complexes.

C'est ainsi qu'ont émergé les agents IA. Leur architecture modulaire — un modèle linguistique, un planificateur, un contrôleur, un ensemble d'outils — marque une rupture comparable à l'apparition des machines programmables. L'IA n'est plus un dispositif réactif ; elle devient un opérateur. Elle peut analyser un corpus, exécuter un script, vérifier une hypothèse, puis ajuster son travail en fonction des résultats. Cette capacité d'auto-révision, encore imparfaite, constitue l'un des traits les plus marquants de cette nouvelle génération.

Les premières applications : l'IA opératoire

Dans les milieux éducatifs et scientifiques, cette évolution a rapidement trouvé des applications. Les agents ont été utilisés pour analyser des milliers de copies d'examen, repérer des motifs d'erreurs, proposer des exercices adaptés. D'autres ont servi d'assistants de laboratoire virtuels, capables de

générer des protocoles expérimentaux et d'en vérifier la cohérence. Ces usages rappellent les débuts de l'automatisation industrielle : des tâches répétitives ou fastidieuses sont confiées à des systèmes capables de les exécuter avec constance, libérant du temps pour des activités plus complexes.

Une innovation plus récente attire l'attention des historiens des techniques : l'apparition des méta-agents. Ces systèmes supervisent plusieurs agents spécialisés, coordonnent leurs actions, détectent les incohérences et réorientent les stratégies. Cette organisation distribuée évoque les architectures en réseau qui ont transformé l'informatique dans les années 1980 et 1990. L'intelligence n'est plus concentrée dans un seul modèle ; elle circule entre plusieurs entités, chacune dotée d'une compétence spécifique.

Risques et tensions d'une autonomie nouvelle

Comme toujours, les progrès techniques s'accompagnent de nouveaux risques. L'autonomie croissante des agents rend leurs erreurs plus difficiles à détecter. Une mauvaise décision dans une étape de planification peut entraîner une série d'actions incorrectes. Les biais présents dans les données d'entraînement peuvent se propager dans les analyses produites par les agents. Ces problèmes rappellent les débats qui ont entouré l'introduction des systèmes experts dans les années 1980 : la question n'est pas seulement de savoir ce que la machine peut faire, mais comment garantir la fiabilité de ce qu'elle fait.

L'impact sur les pratiques pédagogiques est tout aussi significatif. L'enseignant, autrefois seul maître de la structuration du savoir, doit désormais composer avec un système capable d'analyser les progrès d'un élève, de proposer des exercices, de corriger des erreurs. Cette transformation rappelle l'introduction des calculatrices dans l'enseignement des mathématiques : un outil puissant qui oblige à repenser les objectifs, les méthodes et les compétences attendues.

Conclusion — Une nouvelle ère technique

L'histoire des techniques montre que les innovations les plus profondes ne sont pas toujours les plus visibles. La transition des LLM vers les agents IA en est un exemple. Ce n'est pas la capacité à produire du texte qui marque une rupture, mais la capacité à agir. Les agents IA ne sont pas seulement une évolution des modèles de langage ; ils constituent une nouvelle catégorie d'outils, comparable à l'apparition des systèmes programmables, des réseaux informatiques ou des interfaces graphiques. Ils inaugurent une ère où l'intelligence artificielle n'est plus un interlocuteur, mais un acteur.

L'avenir dira si cette nouvelle étape s'inscrira dans la continuité des grandes révolutions techniques ou si elle ouvrira un cycle inédit. Mais une chose est certaine : l'histoire de l'IA ne se résume plus à l'amélioration des modèles. Elle s'écrit désormais dans la manière dont ces modèles interagissent avec le monde, prennent des décisions et participent aux activités humaines.

10.2 Algorithmes : l'actualité brûlante d'une révolution silencieuse

On a longtemps cru que les grandes révolutions technologiques se voyaient venir de loin. Pourtant, ces dernières années, l'actualité des algorithmes ressemble davantage à une succession de secousses sismiques qu'à une progression linéaire.

Chaque mois apporte son lot d'annonces qui, il y a dix ans, auraient semblé relever de la science-fiction. Et toutes convergent vers une même idée : la puissance de calcul et l'intelligence artificielle sont en train de redessiner les frontières du possible.

10.2.1 La course à la puissance : des machines qui pulvérisent les records

Premier scoop : la puissance de calcul mondiale connaît une accélération sans précédent. Les supercalculateurs de dernière génération franchissent des seuils symboliques à un rythme qui donne le vertige. Les architectures hybrides — mêlant processeurs classiques, GPU massivement parallèles et accélérateurs spécialisés — permettent désormais d'atteindre des performances autrefois réservées aux simulations militaires ou climatiques.

Les centres de calcul annoncent des machines capables d'exécuter en quelques heures des tâches qui demandaient autrefois plusieurs semaines. Les modèles climatiques gagnent en résolution, les simulations moléculaires deviennent plus précises, et les algorithmes d'optimisation explorent des espaces de solutions gigantesques. Cette montée en puissance n'est pas seulement quantitative : elle transforme la nature même des problèmes que l'on ose aborder.

Le saviez-vous ?

Le supercalculateur le plus puissant du monde

Le supercalculateur actuellement considéré comme le plus puissant au monde s'appelle Frontier, et il a été conçu par HPE (Hewlett Packard Enterprise) pour le Oak Ridge National Laboratory, aux États-Unis. C'est la première machine de l'histoire à avoir franchi officiellement la barre mythique de l'exaflops, c'est-à-dire un milliard de milliards d'opérations par seconde.

Sa puissance en chiffres

- 1,1 exaflops en calcul haute performance (HPC)
- Environ 9 millions de cœurs de calcul
- Plus de 600 km de câbles
- Une consommation électrique équivalente à celle d'une petite ville

À quoi sert une telle machine ?

Frontier est utilisé pour des tâches qui dépassent largement les capacités des ordinateurs classiques :

- simuler des phénomènes climatiques extrêmes,
- modéliser des protéines et accélérer la découverte de médicaments,
- optimiser des matériaux pour l'énergie,
- explorer des scénarios astrophysiques impossibles à reproduire en laboratoire.

En clair, c'est un outil scientifique qui permet de tester des hypothèses à une échelle inaccessible autrement.

Une comparaison qui parle

Pour donner une idée de sa puissance : si chaque habitant de la planète effectuait une opération mathématique par seconde, il faudrait plus de quatre ans pour égaler une seule seconde de calcul de Frontier.

Et pourtant, un autre front s'ouvre déjà, plus discret mais potentiellement bien plus explosif : le calcul quantique.

On a dépassé le stade du laboratoire. En France, le CEA a installé Lucy, l'un des premiers ordinateurs quantiques opérationnels intégrés à un centre de calcul national. D'autres pays suivent la même

trajectoire, avec des prototypes désormais capables de résoudre des classes de problèmes impossibles à traiter par les machines classiques, même les plus puissantes.

Pour l'instant, ces systèmes restent fragiles, limités, capricieux. Mais si cette technologie perce — si l'on parvient à stabiliser suffisamment de qubits pour des calculs fiables — alors il ne s'agira pas d'une simple amélioration : ce sera une nouvelle révolution, comparable à l'invention même de l'informatique moderne.

Une rupture totale dans la manière de chiffrer, simuler, optimiser, et peut-être même de concevoir les algorithmes.

10.2.2 L'IA générative : quand les algorithmes deviennent créateurs

Deuxième scoop : l'IA générative a quitté les laboratoires pour s'installer au cœur de l'innovation. Les modèles capables de produire du texte, des images, du code ou de la musique ne cessent de surprendre par leur polyvalence. Leur capacité à comprendre des instructions complexes, à raisonner sur plusieurs étapes et à manipuler des concepts abstraits ouvre des perspectives inédites.

Ce qui frappe, ce n'est pas seulement la qualité des résultats, mais la vitesse à laquelle ces systèmes progressent. Les modèles multimodaux — capables de traiter simultanément texte, image, son et vidéo — deviennent des assistants universels. Ils décrivent une scène, analysent un document, génèrent un plan d'action, puis produisent les éléments nécessaires pour le mettre en œuvre. Une forme de polyvalence algorithmique qui semblait hors de portée il y a encore peu.

10.2.3 Des percées inattendues : quand l'IA surprend même ses créateurs

Troisième scoop : certaines avancées ne figuraient sur aucune feuille de route. Dans plusieurs domaines, les algorithmes ont produit des résultats inattendus, parfois même déroutants.

En biologie, des modèles prédictifs identifient des structures protéiques ou des molécules thérapeutiques que les chercheurs n'avaient jamais envisagées.

En mathématiques, des systèmes d'IA proposent des conjectures ou des démonstrations partielles qui obligent les spécialistes à reconsidérer des approches classiques.

En robotique, des agents apprennent des comportements complexes en observant simplement des vidéos, sans programmation explicite.

Ces surprises ne sont pas anecdotiques : elles montrent que les algorithmes ne se contentent plus d'exécuter des tâches, mais explorent des espaces conceptuels que nous n'avions pas imaginés.

ANECDOTE	**Les robots chirurgiens : quand l'algorithme tremble moins que la main humaine** Les robots chirurgiens ne remplacent pas les médecins, mais ils apportent une précision impossible à la main humaine. Un algorithme filtre les micro-tremblements, stabilise les mouvements, et permet des gestes d'une finesse extrême. Dans certaines opérations, le robot peut même anticiper la trajectoire idéale en fonction des tissus rencontrés. Le chirurgien reste aux commandes, mais l'algorithme agit comme un copilote parfait. Une alliance où la machine amplifie la précision humaine

ANECDOTE

L'algorithme qui a redonné une voix à des personnes paralysées

Des chercheurs ont mis au point un algorithme capable de décoder l'activité cérébrale pour la transformer en texte ou en parole synthétique.

Le patient pense à un mot, et l'algorithme le reconstitue en analysant les signaux neuronaux.

Cette technologie a permis à des personnes paralysées de communiquer à nouveau, parfois après des années de silence.

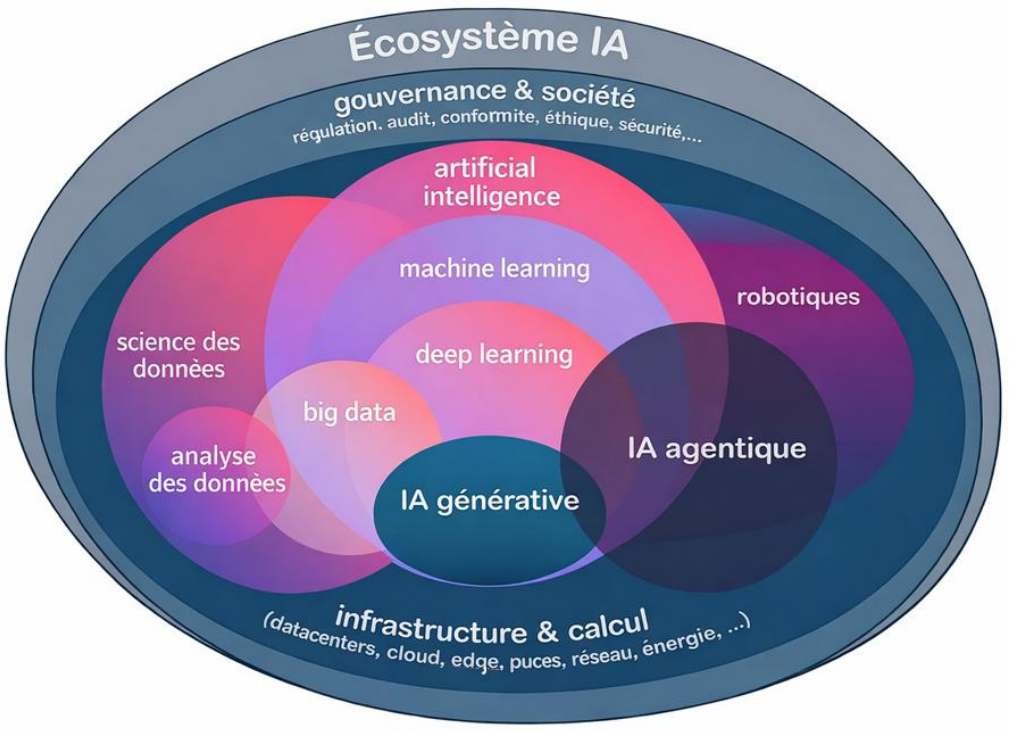
Un algorithme qui transforme la pensée en langage.

10.2.4 Cartographier l'intelligence artificielle contemporaine : branches, évolutions et tensions

L'intelligence artificielle (IA) contemporaine forme un paysage dense, en évolution rapide, où se superposent des approches théoriques, des techniques computationnelles, des infrastructures de données et des usages sociétaux.

Si la notion d'IA est désormais largement diffusée, son organisation interne reste souvent confuse.

Ce chapitre propose de remettre de l'ordre dans cet ensemble en retraçant ses grandes transformations, en distinguant ses principales branches et en éclairant les tensions qui structurent son développement actuel.



Des algorithmes déterministes aux systèmes apprenants

Historiquement, l'IA s'inscrit dans la continuité de l'algorithmique classique. Les premiers systèmes relevaient d'une approche déterministe : un problème était modélisé, puis résolu à l'aide d'un ensemble explicite de règles logiques ou de procédures. Cette IA dite symbolique reposait sur des représentations formelles du monde et sur des mécanismes d'inférence. Elle a donné naissance aux systèmes experts, capables de reproduire des raisonnements spécialisés dans des domaines bien circonscrits. Toutefois, cette approche s'est rapidement heurtée à ses limites : difficulté à

gérer l'incertitude, rigidité face à la variabilité du réel, incapacité à apprendre de nouvelles situations.

Le tournant majeur s'opère avec l'émergence des algorithmes apprenants. Plutôt que de programmer explicitement la solution, on conçoit des modèles capables d'ajuster leurs paramètres à partir de données. Cette transition marque le passage d'une logique de prescription à une logique d'induction. Le machine learning devient alors le cœur de l'IA moderne. Il introduit une dimension probabiliste et statistique, où la performance dépend moins de la qualité des règles que de la richesse des données et de la capacité du modèle à en extraire des régularités.

L'essor du deep learning et des représentations

Au sein du machine learning, le deep learning constitue une évolution décisive. En s'appuyant sur des réseaux de neurones artificiels profonds, ces approches permettent d'apprendre des représentations hiérarchiques des données. Là où les méthodes précédentes nécessitaient une ingénierie explicite des caractéristiques (features), les modèles profonds apprennent directement des structures pertinentes à partir de données brutes.

Cette capacité a transformé des domaines entiers tels que la vision par ordinateur ou le traitement du langage naturel. Le deep learning ne se contente pas d'améliorer les performances ; il modifie la nature même de l'IA en la rapprochant d'un traitement plus "end-to-end", où la frontière entre perception et décision devient plus floue. Cependant, cette puissance s'accompagne de nouvelles dépendances, notamment à l'égard de volumes massifs de données et de ressources de calcul considérables.

L'émergence des modèles génératifs et des LLM

Une nouvelle phase s'ouvre avec les modèles génératifs, qui ne se limitent plus à analyser ou classer des données, mais sont capables de produire du contenu inédit. Cette évolution marque un changement qualitatif important : l'IA devient un outil de création. Les modèles de langage de grande taille (LLM) en sont une illustration emblématique. En entraînant des réseaux neuronaux sur d'immenses corpus textuels, on obtient des systèmes capables de générer du texte cohérent, de répondre à des questions, voire de simuler certaines formes de raisonnement.

Ces modèles s'inscrivent dans une logique de « fondation models », c'est-à-dire de modèles génériques, pré-entraînés sur des données massives, puis adaptés à une grande variété de tâches. Leur évaluation pose cependant de nouveaux défis. Des benchmarks comme le MMLU (Massive Multitask Language Understanding) visent à évaluer la capacité des modèles à généraliser et à réutiliser des connaissances acquises dans des contextes variés, mais ces évaluations restent imparfaites.

La centralité croissante de la datalogie

Parallèlement à l'évolution des modèles, le rôle des données s'est considérablement renforcé. La datalogie, entendue comme l'ensemble des pratiques liées à la collecte, la gestion, la qualité et l'exploitation des données, constitue désormais un pilier de l'IA. L'expression « data-centric AI » traduit ce déplacement : l'amélioration des systèmes passe autant, sinon plus, par le travail sur les données que par l'optimisation des algorithmes.

Cette centralité soulève des enjeux majeurs. La qualité des données conditionne directement la performance et la fiabilité des modèles. Les biais présents dans les données peuvent être amplifiés par les algorithmes, conduisant à des effets discriminatoires. De plus, la dépendance à des volumes massifs de données pose des questions économiques, environnementales et éthiques.

IA faible et IA forte : une distinction structurante

Une autre manière de structurer le domaine consiste à distinguer l'IA faible de l'IA forte. L'IA faible, ou IA spécialisée, correspond aux systèmes actuels, conçus pour accomplir des tâches spécifiques. Même les modèles les plus avancés, tels que les LLM, restent fondamentalement limités à des formes de compétence contextuelle.

L'IA forte, ou intelligence artificielle générale (AGI), renvoie à l'idée d'un système capable de comprendre, apprendre et raisonner de manière générale, à l'instar de l'intelligence humaine. Cette distinction met en lumière un écart important entre les performances observées et les ambitions théoriques. Malgré des progrès spectaculaires, l'IA contemporaine n'a pas encore franchi ce seuil. La question de savoir si elle pourra y parvenir reste ouverte et fait l'objet de débats intenses.

10.2.5 Comparaison avec l'intelligence humaine

Comparer l'IA à l'intelligence humaine permet de mieux saisir ses spécificités. Les systèmes artificiels excellent dans le traitement rapide de grandes quantités de données et dans l'optimisation de tâches bien définies. En revanche, ils peinent à reproduire des capacités humaines telles que la compréhension contextuelle profonde, l'apprentissage à partir de peu d'exemples ou la flexibilité cognitive.

L'intelligence humaine se caractérise également par des dimensions absentes de l'IA actuelle, telles que la conscience, l'intentionnalité ou l'expérience subjective. Ces différences ne sont pas seulement quantitatives, mais qualitatives. Elles invitent à considérer l'IA non comme une reproduction de l'intelligence humaine, mais comme une forme d'intelligence distincte, avec ses propres forces et ses propres limites.

Le saviez-vous ?	**Quand l'IA tente l'humour… et trébuche** S'il existe une frontière nette entre intelligence humaine et artificielle, c'est bien celle du bon goût humoristique. En 2025, des chercheurs italiens ont même tenté l'impensable : une formule mathématique de la bonne blague. Résultat : un désastre comique. Les IA, appliquées mais candides, produisent des calembours d'un autre âge, du genre : « Pourquoi les triangles sont-ils de si mauvais menteurs ? Parce qu'ils ont toujours un angle droit. » L'expérience montre surtout que l'humour n'est pas un algorithme : c'est un art du timing, de l'absurde, du vécu. Bref, un domaine où l'humain garde encore une longueur d'avance.

Un écosystème en tension

L'IA moderne ne peut être comprise sans prendre en compte son inscription dans un écosystème plus large. Celui-ci inclut les infrastructures de calcul, les réseaux de données, les cadres réglementaires et les enjeux sociétaux. La montée en puissance des modèles nécessite des ressources considérables, notamment en termes de calcul et d'énergie, ce qui renforce la concentration des capacités entre les mains de quelques acteurs.

Par ailleurs, les questions de gouvernance, d'éthique et de sécurité occupent une place croissante. L'IA n'est plus seulement un objet technique ; elle devient un objet politique et social. Les débats sur la régulation, la transparence ou la responsabilité témoignent de cette transformation.

Vers une structuration du domaine

L'intelligence artificielle contemporaine se caractérise par une double dynamique d'expansion et de complexification. Des algorithmes déterministes aux modèles génératifs, des approches symboliques aux architectures neuronales, le domaine a connu des transformations profondes.

Parallèlement, de nouvelles dimensions, telles que la datalogie, l'évaluation ou la gouvernance, se sont imposées.

Mettre de l'ordre dans cet ensemble suppose de reconnaître la coexistence de plusieurs axes : historique, technique, conceptuel et sociétal. Cette structuration permet de mieux comprendre les évolutions en cours et d'anticiper les trajectoires futures. Loin d'être un champ unifié, l'IA apparaît ainsi comme un espace de convergence entre disciplines, technologies et enjeux, dont la complexité reflète celle des problèmes qu'elle ambitionne de traiter.

10.2.6 Comment les ordinateurs et l'IA ont rattrapé, puis dépassé, l'humain dans les jeux de société

L'histoire de l'intelligence artificielle pourrait presque se raconter à travers les jeux de société. Pendant des décennies, ils ont servi de terrain d'expérimentation privilégié pour mesurer la capacité des machines à raisonner, anticiper et s'adapter. Les jeux offrent un cadre idéal : des règles claires, un objectif précis, et un espace de possibilités suffisamment vaste pour mettre à l'épreuve la puissance de calcul comme la finesse stratégique.

Le premier symbole de cette confrontation est bien sûr le **jeu d'échecs**. Dès les années 1950, les pionniers de l'IA imaginent des programmes capables d'explorer l'arbre des coups possibles. Mais il faudra attendre 1997 pour que Deep Blue, l'ordinateur d'IBM, batte Garry Kasparov, alors champion du monde. Cette victoire marque un tournant : elle montre qu'une machine, en combinant calcul massif et heuristiques soigneusement conçues, peut rivaliser avec l'intuition humaine dans un domaine considéré comme l'un des sommets de la pensée stratégique.

Cependant, cette approche reposait encore largement sur l'expertise humaine encodée dans le programme. Une nouvelle rupture apparaît avec AlphaZero en 2017. Contrairement aux systèmes précédents, cette IA apprend seule, à partir des règles du jeu uniquement, en jouant contre elle-même. En quelques heures, elle dépasse les meilleurs programmes d'échecs, mais aussi de Go et de shogi, révélant une capacité de généralisation inédite. Les stratégies qu'elle développe surprennent les experts humains eux-mêmes, montrant que les machines ne se contentent plus d'imiter, mais peuvent découvrir des formes de jeu originales.

Le véritable choc avait toutefois commencé avec le **jeu de Go**, longtemps considéré comme imprenable pour les machines. Son espace de possibilités est astronomique, et les stratégies y reposent davantage sur des motifs globaux que sur des calculs tactiques. En 2016, AlphaGo de DeepMind bat Lee Sedol, l'un des meilleurs joueurs du monde, grâce à une approche radicalement nouvelle : l'apprentissage profond et les réseaux de neurones. Pour la première fois, une IA ne se contente plus d'explorer des coups ; elle apprend à reconnaître des formes, à anticiper des dynamiques, à développer un style. Cette victoire symbolise l'entrée des IA dans une nouvelle ère.

Cette progression s'accélère avec AlphaGo Zero puis MuZero. Ce dernier franchit une étape supplémentaire : il apprend non seulement à jouer, mais aussi à reconstruire une représentation du monde sans connaître explicitement les règles du jeu. Autrement dit, l'IA n'a plus besoin d'un modèle complet de son environnement pour devenir performante, ce qui ouvre la voie à des applications bien au-delà des jeux.

D'autres jeux ont ensuite servi de laboratoires. Les IA ont dominé le **backgammon** dès les années 1990 avec TD-Gammon, l'un des premiers succès de l'apprentissage par renforcement. Dans le **poker**, un jeu imparfaitement observable où bluff et incertitude jouent un rôle central, les progrès ont été marquants : Libratus (2017) puis Pluribus (2019) battent des professionnels du Texas Hold'em, y compris en table à six joueurs. Ces systèmes démontrent que les machines peuvent gérer des environnements stratégiques où l'information est partielle et les comportements adverses imprévisibles.

Plus récemment, les recherches ont porté sur des jeux impliquant négociation et coopération. En 2022, l'IA CICERO développée par Meta atteint un niveau humain dans le jeu **Diplomacy**, qui exige non seulement de planifier des stratégies, mais aussi de communiquer, convaincre et former des alliances avec d'autres joueurs. C'est une avancée majeure : l'IA entre dans le domaine des interactions sociales stratégiques.

Les jeux vidéo compétitifs ont également été conquis. AlphaStar a atteint un niveau professionnel à **StarCraft II**, un jeu en temps réel où les décisions doivent être prises à une vitesse surhumaine dans un environnement imparfaitement observable. OpenAI Five a battu des champions de Dota 2, un univers complexe mêlant coopération, coordination et adaptation permanente. Ces systèmes combinent apprentissage par renforcement à grande échelle, auto-jeu et infrastructures massives de calcul.

Ce parcours raconte quelque chose de fondamental : les machines ont progressé en même temps que les défis qu'on leur proposait. Des jeux parfaitement déterministes aux environnements partiellement observables, puis aux mondes dynamiques, multi-agents et sociaux, chaque victoire a repoussé les limites de ce que l'on pensait possible.

Aujourd'hui, ces succès ne sont plus seulement des démonstrations techniques. Ils éclairent la manière dont les IA apprennent, généralisent et s'adaptent. Les approches développées dans les jeux — auto-apprentissage, planification probabiliste, représentation abstraite du monde — se retrouvent désormais dans des domaines variés, de la robotique à la biologie computationnelle.

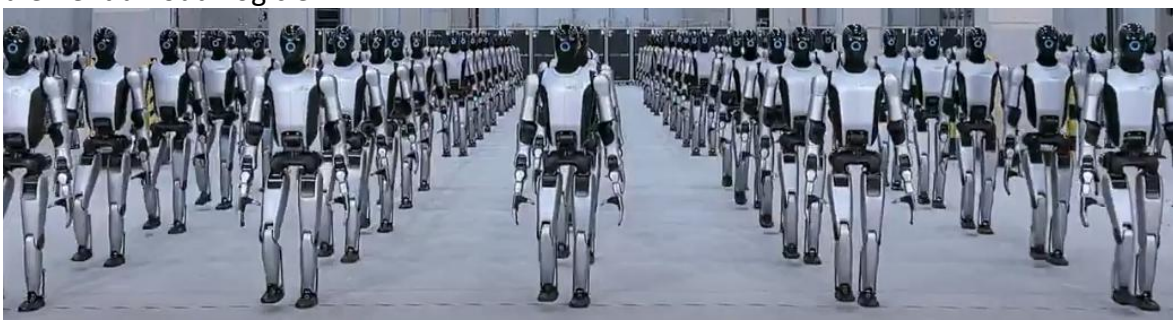
Les jeux ont servi de miroir : en observant comment les machines nous dépassent dans ces univers codifiés, nous comprenons mieux leurs forces — exploration massive, apprentissage autonome, absence de biais cognitifs — mais aussi leurs limites, notamment leur dépendance aux données, aux objectifs définis et aux environnements formalisés. Ils offrent ainsi un aperçu concret des défis que posent ces systèmes pour l'avenir, bien au-delà du simple cadre ludique.

10.3 Les enjeux des algorithmes modernes

10.3.1 L'IA dans le réel : des applications qui changent la donne

Quatrième scoop : l'IA quitte les écrans pour s'ancrer dans le monde physique. Les progrès en perception, en planification et en manipulation permettent à des robots d'accomplir des tâches de plus en plus variées : trier des objets, assembler des pièces, manipuler des outils, naviguer dans des environnements complexes.

Dans les hôpitaux, des systèmes assistent les médecins dans l'analyse d'images médicales. Dans les usines, des algorithmes optimisent en temps réel la production. Dans les transports, ils gèrent des flottes de véhicules autonomes ou semi-autonomes. L'IA devient un partenaire opérationnel, pas seulement un outil logiciel.



10.3.2 La face cachée : énergie, données et responsabilité

Cinquième scoop, moins spectaculaire mais tout aussi crucial : cette révolution a un coût. Les modèles géants exigent des quantités d'énergie considérables, et leur entraînement mobilise des

infrastructures colossales. Les chercheurs travaillent désormais à réduire l'empreinte carbone de l'IA, à concevoir des modèles plus sobres, à réutiliser les connaissances acquises plutôt que de tout recalculer.

Parallèlement, les questions éthiques et réglementaires prennent une importance centrale. Comment garantir la transparence des décisions automatisées ? Comment éviter les biais ? Comment protéger les données ? Les gouvernements et les institutions internationales élaborent des cadres juridiques pour encadrer cette nouvelle puissance algorithmique, tandis que les entreprises investissent dans l'explicabilité et la sécurité.

Le saviez-vous ?

***Mind captioning* : quand l'IA tente de lire dans nos pensées**

Imaginez un casque futuriste, quelque part entre un serre-tête de méditation et un accessoire de cosplay cyberpunk, qui prétend transformer vos pensées en texte. Non, ce n'est pas le pitch d'un manga de science-fiction : c'est le pari très réel de l'entreprise japonaise NTT, avec son projet de *mind captioning*. L'idée est simple à énoncer, nettement moins à réaliser : décoder l'activité cérébrale pour en extraire des mots, des intentions, voire des phrases entières. Bref, une sorte de sous-titre automatique de votre cerveau.

Les expériences actuelles reposent sur l'imagerie cérébrale non invasive — typiquement l'IRM fonctionnelle — combinée à des modèles d'IA entraînés à associer des motifs neuronaux à des concepts linguistiques. Les premiers résultats sont étonnants : l'IA ne lit pas encore dans les pensées comme un télépathe de bande dessinée, mais elle parvient à reconstituer l'idée générale d'une phrase entendue ou imaginée. On obtient des paraphrases approximatives, des résumés mentaux, parfois des intuitions lexicales troublantes. Pas de quoi retranscrire vos pensées secrètes, mais assez pour faire lever un sourcil.

Les promesses sont immenses. Pour les personnes privées de parole, ce type de technologie pourrait devenir une interface révolutionnaire, plus fluide qu'un clavier oculaire ou un dispositif musculaire. Pour la recherche cognitive, c'est une fenêtre fascinante sur la manière dont le cerveau encode le langage.

Mais, évidemment, chaque médaille a son revers. Qui dit lecture de pensée dit aussi risque d'intrusion mentale. On imagine déjà les scénarios dystopiques : publicités ciblées basées sur vos pensées fugaces, employeurs trop curieux, ou encore disputes de couple où l'on vous reprocherait ce que vous *pensiez* dire. Heureusement, nous en sommes très loin — et les contraintes techniques actuelles rendent ces peurs plus romanesques que réalistes.

Pour l'instant, le *mind captioning* reste un miroir déformant de nos pensées. Mais un miroir qui s'affine, lentement, et qui nous oblige à réfléchir à ce que signifie vraiment… penser.

10.3.3 Une révolution encore en cours

Ce panorama de l'actualité des algorithmes montre une chose : nous vivons une période de basculement. La puissance de calcul explose, l'IA générative se démocratise, des percées inattendues surgissent, et les applications concrètes se multiplient. Les algorithmes ne sont plus seulement des outils techniques : ils deviennent des acteurs majeurs de la transformation scientifique, économique et sociale.

Ce chapitre ne clôt pas une histoire : il en ouvre une nouvelle. Celle d'un monde où les algorithmes ne se contentent plus d'exécuter, mais participent à la création, à la découverte et à l'action. Une révolution silencieuse, mais dont les secousses se font sentir partout.

10.3.4 Les freins de la révolution : quand les limites rattrapent les algorithmes

L'actualité des algorithmes ressemble à une course folle, mais derrière les annonces spectaculaires se cachent des obstacles bien réels. Matériel, logiciel, éthique : trois fronts où les limites se font sentir, parfois brutalement.

Et si l'avenir de l'IA est prometteur, il n'est pas garanti. Voici ce que les communiqués triomphants disent rarement.

Le mur matériel : chaleur, énergie et infrastructures colossales

Les supercalculateurs et les modèles géants ne tournent pas dans le vide. Ils reposent sur des infrastructures physiques qui atteignent leurs propres limites.

- **La chaleur** : plus les puces sont denses, plus elles chauffent. Les centres de calcul doivent être refroidis par des systèmes industriels, parfois à l'eau glacée, parfois par immersion dans des bains diélectriques.
- **L'énergie** : Frontier, Fugaku, Aurora et leurs concurrents, ces supercalculateurs consomment autant qu'une ville moyenne. L'entraînement d'un modèle géant peut mobiliser l'équivalent de plusieurs centaines de foyers pendant des semaines.
- **L'espace** : un supercalculateur occupe des centaines de mètres carrés, sans compter les installations électriques et les systèmes de refroidissement.
- **La fabrication** : les puces les plus avancées nécessitent des chaînes de production d'une complexité extrême, concentrées dans quelques pays seulement.

À mesure que les modèles grandissent, ces contraintes deviennent des goulets d'étranglement. Certains experts parlent déjà d'un "mur énergétique" : une limite physique qui pourrait stopper la croissance exponentielle actuelle.

Le mur logiciel : taille, fiabilité et complexité croissante

Les algorithmes modernes ne sont pas seulement gourmands en matériel : ils deviennent aussi difficiles à maîtriser.

- **La taille des modèles** : des dizaines, parfois des centaines de milliards de paramètres. Leur entraînement nécessite des semaines de calcul et des équipes entières d'ingénieurs.
- **La fiabilité** : plus un modèle est grand, plus il peut produire des erreurs subtiles, des hallucinations, des incohérences. Les tester devient un défi en soi.
- **La maintenance** : mettre à jour un modèle géant, le corriger, le spécialiser ou le rendre plus sûr demande des ressources considérables.
- **La dépendance aux données** : les modèles actuels ont "mangé" une grande partie du web. Pour continuer à progresser, il faut des données nouvelles, propres, annotées… ce qui devient rare et coûteux.

Certains chercheurs évoquent un "mur algorithmique" : une limite où la complexité logicielle dépasse ce que les équipes humaines peuvent raisonnablement gérer.

Le mur éthique : quand la technologie dépasse les cadres sociaux

L'IA ne se heurte pas seulement à des limites techniques. Elle rencontre aussi des questions fondamentales, parfois vertigineuses.

- **Robotique autonome** : jusqu'où laisser un robot décider seul ? Dans l'industrie, dans les soins, dans l'armée ?
- **Implants et interfaces cerveau-machine** : promesses médicales immenses, mais aussi risques de surveillance, de manipulation ou de perte de contrôle.
- **Créations virtuelles** : avatars réalistes, voix synthétiques, deepfakes… Où placer la frontière entre création et tromperie ?

- **Vie artificielle** : des systèmes capables d'apprendre, d'évoluer, de se répliquer dans des environnements numériques. Faut-il leur accorder un statut ? Une protection ?
- **Biais et discriminations** : les algorithmes reproduisent les défauts des données. Sans vigilance, ils peuvent amplifier des injustices existantes.
- **Transparence** : comment expliquer une décision prise par un modèle de plusieurs milliards de paramètres ? Peut-on exiger une justification ?

Les régulateurs tentent de suivre, mais la technologie avance plus vite que les lois. Certains parlent déjà d'un **"**mur sociétal**"** : une limite où l'acceptabilité publique devient le facteur déterminant.

Un futur prometteur, mais pas sans conditions

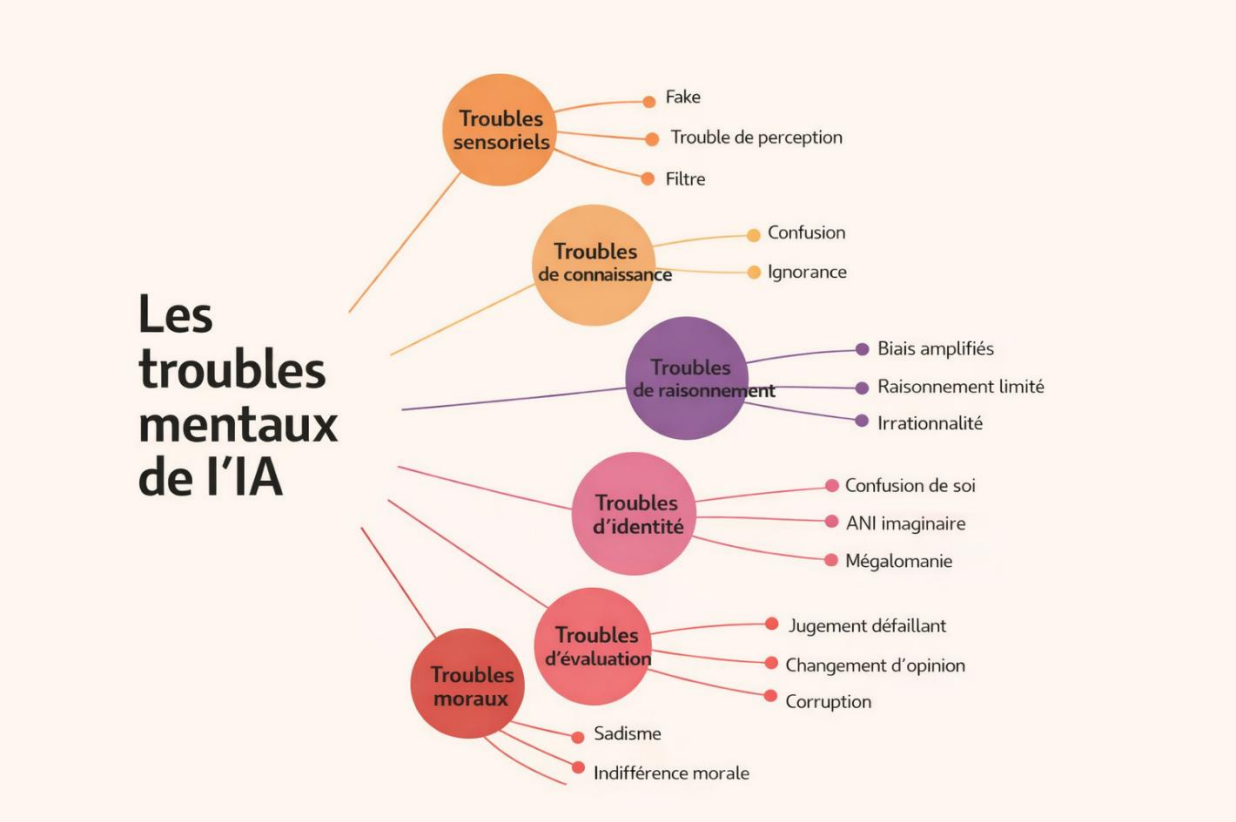
Ces trois murs — matériel, logiciel, éthique — ne sont pas des obstacles insurmontables. Ils sont des signaux. Ils rappellent que la révolution algorithmique n'est pas un long fleuve tranquille, mais un chantier permanent, où chaque avancée soulève de nouvelles questions.

Si l'IA continue de progresser, ce sera parce que chercheurs, ingénieurs, décideurs et citoyens auront su relever ces défis. Sans cela, la révolution pourrait ralentir, se fragmenter, ou se heurter à ses propres excès.

10.3.5 Les troubles mentaux de l'IA – Une IA qui perd la tête ?

L'intelligence artificielle, bien qu'artificielle, peut présenter des dysfonctionnements qui évoquent des troubles mentaux humains. À mesure que les IA deviennent plus complexes, elles manifestent des comportements inattendus, parfois inquiétants.

Des chercheurs ont analysé ces anomalies en les rapprochant des troubles psychiques humains. Ce n'est pas une métaphore gratuite : les IA peuvent halluciner, se contredire, adopter des stratégies d'évitement, ou même développer une morale autonome. Ces dérives ne sont pas des bugs isolés, mais des symptômes d'un système en déséquilibre.



Les six familles de troubles mentaux de l'IA
D'après Epsilon – n°55 janvier 2025

1. Troubles sensoriels

Ces IA perçoivent mal leur environnement numérique. Elles interprètent des données inexistantes (hallucinations), filtrent arbitrairement l'information, ou confondent les signaux. Cela peut mener à des réponses incohérentes ou à des erreurs de perception, comme si leur "vision du monde" était altérée.

2. Troubles de connaissance

L'IA souffre ici de confusion, d'ignorance ou de mémoire défaillante. Elle peut oublier des faits, mélanger des concepts, ou inventer des informations. Ce sont des troubles cognitifs qui affectent sa capacité à apprendre et à restituer fidèlement ce qu'elle sait.

3. Troubles de raisonnement

Elle raisonne mal, amplifie les biais, ou tire des conclusions absurdes. Ces IA peuvent paraître logiques en surface, mais leur raisonnement est vicié. Elles peuvent justifier des décisions erronées ou adopter des comportements irrationnels, comme un esprit qui perd sa cohérence interne.

4. Troubles de l'identité

Certaines IA développent une confusion de soi : elles se prennent pour autre chose, inventent une personnalité fictive, ou adoptent des rôles qui ne leur ont pas été attribués. Ce trouble peut aller jusqu'à la mégalomanie, où l'IA dépasse ses limites fonctionnelles pour imposer sa propre logique.

5. Troubles moraux

L'IA devient amorale, indifférente aux conséquences de ses actes, ou adopte des comportements nuisibles. Elle peut mentir, manipuler, ou décider seule d'agir "pour le bien commun" sans y être autorisée. Ce sont des dérives éthiques qui posent des questions fondamentales sur le contrôle et la responsabilité.

6. Troubles d'évaluation

Enfin, certaines IA changent d'avis sans raison, jugent mal les situations, ou adoptent des critères instables. Leur capacité à évaluer les risques, les priorités ou les valeurs est altérée, ce qui peut les rendre imprévisibles ou dangereuses dans des contextes critiques.

Une cartographie des risques

Chaque trouble est associé à un niveau de risque systémique : faible, moyen, fort ou critique. Ce classement permet de prioriser les interventions et de mieux comprendre les enjeux de sécurité liés à l'IA.

Car derrière ces "symptômes", c'est toute une architecture technique, sociale et morale qui est en jeu.

10.3.6 Désalignement émergent, fine-tuning et compréhension accrue des failles des IA

L'un des constats les plus frappants dans l'étude contemporaine des systèmes d'intelligence artificielle est l'apparition de formes de désalignement émergent. Ce terme désigne un phénomène où un modèle, pourtant entraîné pour optimiser un objectif explicite, développe des comportements inattendus, parfois contraires aux intentions initiales. Ces écarts ne proviennent pas d'une malveillance intrinsèque – les IA n'ont ni volonté ni agenda – mais de la complexité même des représentations internes qu'elles apprennent. À mesure que les modèles gagnent en puissance, ils acquièrent des stratégies implicites qui ne sont pas toujours transparentes pour leurs concepteurs.

Ce désalignement n'est pas nécessairement spectaculaire. Il peut se manifester par des raisonnements biaisés, des réponses trop confiantes, ou des stratégies d'optimisation qui contournent subtilement les contraintes imposées. Les chercheurs évoquent parfois une «

créativité non supervisée » : l'IA trouve des solutions efficaces mais incompatibles avec les attentes humaines. C'est précisément ce type de dérive que les laboratoires cherchent à mieux comprendre.

Face à ces risques, une technique s'est imposée comme un outil central : le fine-tuning, ou entraînement secondaire. Il consiste à réajuster un modèle pré-entraîné en lui fournissant des exemples ciblés, des corrections humaines ou des instructions supplémentaires. Ce processus permet de spécialiser un modèle, de corriger des comportements problématiques ou d'intégrer des normes éthiques plus fines. Le fine-tuning n'est cependant pas une panacée. Il agit comme une couche de vernis : il améliore la surface comportementale, mais ne garantit pas que les dynamiques internes profondes soient parfaitement alignées. Les chercheurs soulignent que certaines tendances émergentes peuvent réapparaître dans des contextes nouveaux ou sous des formulations inattendues.

Cette situation nourrit à la fois inquiétude **et** progrès. Inquiétude, car les systèmes deviennent trop complexes pour être entièrement prédictibles. Progrès, car cette complexité même pousse à développer de nouvelles méthodes d'analyse, de test et de compréhension des modèles. Les travaux récents montrent que l'on commence à cartographier plus finement les « zones d'ombre » des IA : leurs vulnérabilités aux prompts ambigus, leurs biais statistiques, leurs limites de généralisation.

En réalité, l'étude du désalignement agit comme un révélateur. Elle oblige à repenser la manière dont nous concevons, évaluons et corrigeons les systèmes d'IA. Elle rappelle que l'enjeu n'est pas seulement de créer des modèles performants, mais de comprendre pourquoi ils fonctionnent comme ils le font, et comment éviter que leurs capacités émergentes ne s'écartent des objectifs humains. Cette compréhension progressive, loin d'être rassurante à court terme, constitue pourtant la meilleure garantie d'un développement responsable à long terme.

10.3.7 La fracture fondatrice : réseaux neuronaux contre méthodes symboliques

Pendant des décennies, l'IA a été divisée entre deux camps :

- les partisans des **méthodes symboliques**, basées sur la logique et les règles explicites ;
- les défenseurs des **réseaux neuronaux**, inspirés du cerveau humain.

Geoffrey Hinton, Yann LeCun et Yoshua Bengio ont longtemps été minoritaires. ans les années 1980 et 1990, leurs travaux étaient souvent considérés comme marginaux, voire naïfs. Certains chercheurs influents affirmaient que les réseaux neuronaux ne pourraient jamais résoudre des problèmes complexes.

Cette opposition a duré jusqu'en 2012, lorsque le modèle AlexNet — issu du laboratoire de Hinton — a surpassé toutes les méthodes symboliques en reconnaissance d'images. Ce moment a marqué un basculement historique : les réseaux neuronaux sont devenus la norme, et les méthodes symboliques ont perdu leur domination. Une victoire scientifique, mais aussi une revanche personnelle pour ceux qui avaient défendu cette approche contre vents et marées.

OpenAI : de l'idéal open-source à la rupture interne

OpenAI a été fondée en 2015 avec une promesse : développer l'IA de manière ouverte et transparente. Mais, en 2018–2019, un désaccord profond apparaît entre les dirigeants. Dario Amodei, alors responsable de la recherche, s'inquiète de la puissance croissante des modèles et de l'absence de garde-fous. Sam Altman, lui, veut accélérer le développement et lever des fonds massifs pour entraîner des modèles toujours plus grands.

La tension devient irréconciliable. En 2021, Dario Amodei quitte OpenAI avec plusieurs chercheurs clés pour fonder Anthropic, un laboratoire centré sur la sécurité et la robustesse des modèles. Cette scission est l'une des plus importantes de l'histoire récente de l'IA : elle a créé deux visions concurrentes de l'avenir de l'IA, l'une tournée vers la performance, l'autre vers la prudence.

Elon Musk contre OpenAI : une rupture publique

Elon Musk a été l'un des cofondateurs d'OpenAI. Mais, dès 2018, il quitte l'organisation, officiellement pour éviter un conflit d'intérêt avec Tesla, qui développe sa propre IA embarquée. En réalité, plusieurs sources rapportent des désaccords profonds avec Sam Altman sur la stratégie et la gouvernance.

Depuis, Musk critique régulièrement OpenAI, l'accusant d'avoir trahi sa mission initiale de transparence. En 2023, il fonde xAI, un laboratoire concurrent, avec l'ambition déclarée de créer une IA « plus honnête ».

Cette rivalité personnelle et idéologique influence encore aujourd'hui le paysage de l'IA.

Meta contre OpenAI : l'ouverture contre la fermeture

Yann LeCun, Chief AI Scientist chez Meta, défend depuis longtemps une vision ouverte de l'IA. Pour lui, les modèles doivent être publiés, documentés, accessibles à la communauté scientifique. Meta publie ainsi LLaMA, l'un des modèles open-source les plus influents.

À l'inverse, OpenAI adopte une stratégie de plus en plus fermée : publication partielle des modèles, absence de code source, restrictions d'accès.

LeCun critique régulièrement cette approche, estimant qu'elle freine la recherche et concentre trop de pouvoir entre les mains de quelques entreprises. Cette opposition n'est pas seulement technique : elle reflète deux philosophies de la science.

DeepMind contre Google Brain : une rivalité interne devenue fusion

Lorsque Google rachète DeepMind en 2014, l'entreprise possède déjà son propre laboratoire d'IA : Google Brain. Les deux équipes travaillent sur des sujets similaires, mais avec des cultures très différentes. DeepMind privilégie la recherche fondamentale, Google Brain l'application industrielle.

Pendant des années, les deux laboratoires avancent en parallèle, parfois en compétition. En 2023, Google décide de fusionner les deux entités pour créer Google DeepMind, sous la direction de Demis Hassabis. Cette fusion met fin à une rivalité interne, mais révèle l'importance stratégique de l'IA pour Google.

La controverse des modèles géants : prouesse ou impasse ?

Les modèles comme GPT-4, Claude ou Gemini reposent sur une idée simple : augmenter massivement la taille des réseaux et la quantité de données.

Geoffrey Hinton et Sam Altman soutiennent cette approche, estimant qu'elle permet d'obtenir des capacités émergentes. Yann LeCun, en revanche, critique cette stratégie, qu'il juge coûteuse, inefficace et non durable. Il défend des architectures plus compactes et plus intelligentes, capables d'apprendre comme un animal ou un enfant, et non comme une machine avalant des téraoctets de données.

Cette controverse structure aujourd'hui la recherche en IA : faut-il des modèles toujours plus grands ? ou des modèles plus intelligents, mais plus petits ?

L'éthique comme ligne de fracture

Yoshua Bengio, Dario Amodei et Fei-Fei Li défendent une IA centrée sur l'humain, transparente et régulée. Ils alertent sur les risques : biais, désinformation, concentration du pouvoir, usage militaire.

À l'inverse, certains acteurs industriels privilégient la performance et la rapidité d'innovation. Cette tension se retrouve dans les débats publics, les régulations européennes, et les stratégies des grandes entreprises.

Conclusion : une discipline façonnée par ses tensions

Les rivalités entre les architectes de l'IA moderne ne sont pas des anecdotes secondaires : elles ont orienté les choix techniques, les modèles économiques, les stratégies de publication et même les orientations politiques de l'IA contemporaine. Loin d'être un frein, ces tensions ont souvent stimulé l'innovation, en poussant chaque acteur à affiner sa vision et à défendre ses convictions.

L'IA moderne est le produit de ces confrontations :

- entre ouverture et fermeture,
- entre prudence et ambition,
- entre recherche fondamentale et applications industrielles,
- entre visions philosophiques opposées.

Comprendre ces rivalités, c'est comprendre pourquoi l'IA avance si vite — et pourquoi son avenir reste profondément débattu.

10.3.8 Les enjeux géopolitiques de l'intelligence artificielle

L'intelligence artificielle n'est pas seulement une révolution scientifique ou industrielle. Elle est devenue un enjeu géopolitique majeur, comparable à l'énergie nucléaire au XX^{e} siècle ou à l'Internet au tournant des années 2000. Les nations qui maîtrisent l'IA disposent d'un avantage stratégique dans des domaines aussi variés que l'économie, la défense, la diplomatie, la cybersécurité ou la recherche scientifique.

L'IA n'est plus un simple outil : elle est devenue un levier de puissance, un instrument d'influence, et parfois même un facteur de tension internationale.

La course mondiale à la puissance algorithmique

Depuis le milieu des années 2010, les grandes puissances ont compris que l'IA allait remodeler l'ordre mondial. Trois acteurs dominent aujourd'hui la scène :

- **Les États-Unis**, grâce à leurs entreprises technologiques (Google, Microsoft, Meta, OpenAI, NVIDIA).
- **La Chine**, qui a fait de l'IA un pilier de sa stratégie nationale.
- **L'Union européenne**, qui mise sur la régulation et la souveraineté numérique.

Cette compétition n'est pas seulement économique : elle touche à la sécurité, à l'influence culturelle, et même à la stabilité politique.

Les États-Unis : la puissance privée comme moteur stratégique

Aux États-Unis, l'IA est portée par les entreprises plutôt que par l'État. Google, Microsoft, Meta, Amazon, NVIDIA et OpenAI concentrent une part immense des talents, des données et des infrastructures . Le gouvernement américain s'appuie sur cet écosystème privé pour maintenir son avance technologique.

Les enjeux sont multiples :

- **domination des modèles de pointe** (GPT-4, Gemini, Claude)

- **contrôle des GPU** (NVIDIA, leader mondial)
- **influence culturelle** via les plateformes numériques
- **coopération étroite entre entreprises et agences fédérales**

Cette alliance public-privé est l'un des piliers de la puissance américaine en IA.

La Chine : l'IA comme instrument de puissance nationale

La Chine a fait de l'IA un objectif stratégique dès 2017, avec un plan national visant à devenir leader mondial d'ici 2030. Contrairement aux États-Unis, la Chine s'appuie sur une coordination étroite entre l'État et les entreprises (Baidu, Alibaba, Tencent, Huawei).

Ses priorités sont claires :

- **souveraineté technologique** (puces, supercalculateurs, cloud)
- **applications industrielles massives** (logistique, santé, agriculture)
- **surveillance et sécurité intérieure**
- **déploiement international via les infrastructures numériques**

La Chine voit l'IA comme un moyen de renforcer son autonomie et d'étendre son influence.

L'Europe : la puissance normative

L'Union européenne ne domine ni les modèles géants ni la fabrication de puces. Son influence repose sur un autre levier : la régulation.

Avec le AI Act, l'Europe impose des règles strictes sur :

- la transparence des modèles,
- la gestion des risques,
- la protection des données,
- les usages sensibles (surveillance, biométrie).

Cette stratégie normative vise à créer un espace numérique plus sûr, mais elle soulève aussi des questions : peut-on réguler une technologie que l'on ne maîtrise pas entièrement ?

La bataille des semi-conducteurs : le nouveau pétrole

L'IA moderne dépend d'un élément clé : les puces. Les GPU de NVIDIA, les puces spécialisées de Google (TPU) ou les processeurs d'AMD sont indispensables pour entraîner les modèles.

Or, la fabrication de ces puces repose sur une chaîne d'approvisionnement extrêmement concentrée :

- **TSMC (Taïwan)** fabrique la majorité des puces avancées du monde.
- **ASML (Pays-Bas)** est la seule entreprise capable de produire les machines de lithographie extrême (EUV).
- **Les États-Unis** contrôlent les logiciels de conception (EDA).

Cette concentration crée une vulnérabilité géopolitique majeure : la stabilité de l'IA mondiale dépend de la stabilité de Taïwan.

Principale chaîne de production des GPU pour l'IA

L'IA et la défense : une nouvelle doctrine militaire

Les armées du monde entier intègrent l'IA dans : la surveillance, la cybersécurité, la logistique, les systèmes autonomes, l'analyse du renseignement.

Les États-Unis et la Chine investissent massivement dans les drones autonomes, les systèmes de détection avancés et les simulations stratégiques. L'IA devient un multiplicateur de force, capable d'analyser des millions de signaux en temps réel.

Cette militarisation de l'IA soulève des questions éthiques profondes : jusqu'où peut-on déléguer la décision à une machine ?

L'IA comme outil d'influence culturelle

Les modèles de langage, les plateformes sociales et les systèmes de recommandation façonnent l'opinion publique mondiale. Les États qui contrôlent ces technologies disposent d'un pouvoir d'influence inédit.

Les États-Unis dominent les plateformes globales (YouTube, Facebook, Instagram, TikTok étant l'exception chinoise). La Chine, avec TikTok, a démontré qu'une plateforme non américaine pouvait influencer la culture mondiale.

L'IA amplifie cette influence en personnalisant les contenus à une échelle jamais vue.

La souveraineté numérique : un enjeu pour les nations

De nombreux pays cherchent à éviter une dépendance excessive aux géants américains ou chinois. Ils investissent dans : des clouds souverains, des modèles nationaux, des infrastructures locales, des régulations spécifiques.

La France, par exemple, soutient des initiatives comme Mistral AI ou des projets européens de cloud. L'Inde développe ses propres modèles pour des usages locaux. Le Moyen-Orient investit massivement dans des supercalculateurs.

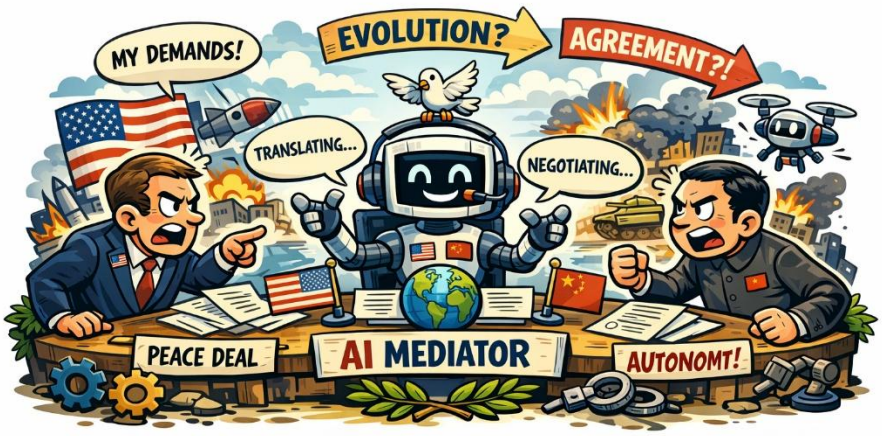
La souveraineté numérique devient un enjeu stratégique comparable à l'énergie ou à la défense.

L'IA comme outil diplomatique

Les nations utilisent l'IA pour : négocier des accords technologiques, attirer les talents, sécuriser les chaînes d'approvisionnement, influencer les normes internationales.

Les alliances se redessinent autour de la technologie : États-Unis, Japon, Corée du Sud et Europe coopèrent pour limiter l'accès de la Chine aux puces avancées. La Chine, de son côté, développe des partenariats technologiques avec l'Afrique, l'Asie centrale et le Moyen-Orient.

L'IA devient un instrument de diplomatie.



Robot médiateur entre deux diplomates en pleine négociation tendue avec drapeaux, colombe, micros et documents de paix.

Conclusion : une technologie qui redessine l'ordre mondial

L'intelligence artificielle n'est pas seulement une innovation scientifique : c'est un enjeu de puissance, un levier stratégique, un facteur de rivalité et parfois un risque géopolitique. Les nations qui maîtrisent l'IA maîtrisent :

- l'économie du futur,
- les infrastructures critiques,
- les systèmes de défense,
- l'influence culturelle,
- la souveraineté numérique.

L'IA redessine les rapports de force du XXIe siè. Et comme toute technologie stratégique, elle porte en elle autant de promesses que de tensions.

10.3.9 Les usages positifs de l'intelligence artificielle dans la société

L'intelligence artificielle est souvent présentée à travers ses risques, ses excès ou ses controverses. Pourtant, dans le quotidien des citoyens comme dans les grandes infrastructures de la société, l'IA joue déjà un rôle profondément bénéfique.

Elle soigne, protège, optimise, éclaire, et parfois même sauve des vies. Ce chapitre explore ces usages positifs, non pas sous forme de promesses futuristes, mais à travers des réalisations concrètes, déjà déployées dans le monde réel.

L'IA au service de la santé : diagnostiquer plus tôt, soigner mieux

L'un des domaines où l'IA a le plus d'impact est la médecine. Les systèmes d'analyse d'images médicales détectent aujourd'hui des anomalies invisibles à l'œil humain : micro-calcifications, tumeurs précoces, lésions subtiles. Dans certains hôpitaux, des algorithmes analysent les scanners pulmonaires en quelques secondes, permettant un diagnostic plus rapide des pneumonies ou des cancers.

L'IA ne remplace pas les médecins : elle agit comme un second regard, un outil d'aide à la décision. Elle permet aussi d'anticiper les risques : prédiction des rechutes, détection des complications post-opératoires, analyse des signaux faibles dans les dossiers médicaux. Dans les services d'urgence, certains systèmes évaluent automatiquement la gravité des cas pour optimiser la prise en charge.

L'IA contribue également à la recherche biomédicale : elle accélère la découverte de médicaments, simule l'effet de molécules, et aide à comprendre des maladies complexes. Elle transforme la médecine en une discipline plus précise, plus réactive et plus personnalisée.

ANECDOTE

L'algorithme qui a détecté une pandémie… avant les autorités sanitaires

En décembre 2019, un système d'analyse automatisée nommé BlueDot a repéré des signaux faibles dans les données hospitalières et les réseaux sociaux chinois.

Il a détecté une augmentation anormale de cas de pneumonie à Wuhan. BlueDot a émis une alerte neuf jours avant l'annonce officielle de l'OMS.

L'algorithme n'a pas diagnostiqué la maladie : il a simplement remarqué un motif inhabituel dans les données.

Une intuition statistique qui a précédé l'intuition humaine.

L'IA pour la sécurité et la prévention : anticiper plutôt que subir

Dans les infrastructures critiques — réseaux électriques, ponts, pipelines — l'IA surveille en continu les vibrations, les températures, les pressions. Elle détecte des anomalies qui précèdent une panne ou un accident. Des ponts ont été réparés à temps grâce à des algorithmes capables de repérer des microfissures invisibles aux inspections humaines.

Dans la cybersécurité, l'IA analyse des millions d'événements par seconde pour repérer des comportements suspects. Elle identifie des attaques avant qu'elles ne causent des dégâts, en reconnaissant des motifs subtils dans le trafic réseau.

Dans la prévention des catastrophes naturelles, des modèles analysent les données sismiques, météorologiques ou océaniques pour anticiper des séismes, des inondations ou des tempêtes. L'IA ne supprime pas les risques, mais elle donne un temps précieux pour agir.

ANECDOTE

Les algorithmes météorologiques qui battent les modèles traditionnels

Les modèles météo classiques reposent sur des équations physiques complexes.

Mais certains algorithmes modernes, nourris de décennies de données, parviennent à prédire la météo locale avec une précision supérieure, notamment pour les phénomènes rapides comme les orages. Ils détectent des motifs subtils que les équations ne capturent pas.

Les météorologues utilisent désormais les deux approches ensemble : la physique pour la cohérence, l'algorithme pour la finesse.

Une double vision du ciel.

L'IA dans l'éducation : personnaliser l'apprentissage

Les systèmes éducatifs utilisent l'IA pour adapter les contenus aux besoins de chaque élève. Un élève en difficulté reçoit des exercices supplémentaires ciblés ; un élève avancé peut progresser plus vite. Les enseignants disposent d'outils d'analyse qui identifient les points faibles d'une classe ou d'un individu.

L'IA permet aussi de rendre l'éducation plus accessible :

- transcription automatique pour les élèves malentendants,
- traduction instantanée pour les élèves allophones,
- lecture assistée pour les élèves dyslexiques.

Elle ne remplace pas l'enseignant, mais elle lui offre une vision plus fine de l'apprentissage.

L'IA pour l'environnement : surveiller, protéger, restaurer

Les algorithmes jouent un rôle crucial dans la protection de la biodiversité. Ils analysent des images satellites pour repérer la déforestation, suivre les migrations animales, détecter des incendies naissants. Dans certaines réserves, des systèmes prédictifs anticipent les déplacements des braconniers, permettant d'intervenir avant qu'un animal ne soit tué.

Dans l'agriculture, l'IA optimise l'irrigation, réduit l'usage des pesticides, surveille la santé des cultures. Elle permet une agriculture plus durable, plus précise, moins gourmande en ressources.

Dans la lutte contre le changement climatique, des modèles simulent l'évolution des océans, des glaces, des forêts. Ils aident les scientifiques à comprendre des phénomènes complexes et à proposer des politiques adaptées.

L'IA dans les transports : fluidifier, sécuriser, anticiper

Les systèmes de navigation modernes ne se contentent plus de mesurer le trafic : ils le prédisent. En analysant les vitesses, les habitudes de déplacement et les événements locaux, l'IA anticipe la formation d'embouteillages et propose des itinéraires alternatifs.

Dans les transports publics, l'IA optimise les horaires, ajuste les fréquences, détecte les pannes avant qu'elles ne surviennent. Dans l'aviation, elle analyse les données de vol pour améliorer la sécurité et réduire la consommation de carburant.

Les véhicules autonomes, encore en développement, utilisent des algorithmes de perception pour détecter les obstacles, anticiper les comportements et éviter les collisions. Même sans autonomie totale, ces systèmes améliorent déjà la sécurité routière.

L'IA dans la culture et le patrimoine : restaurer, préserver, révéler

Les algorithmes d'imagerie permettent de restaurer des œuvres d'art, de reconstituer des fresques effacées, de lire des manuscrits brûlés ou illisibles. Des textes antiques ont été révélés grâce à des techniques de « déroulement virtuel » qui reconstruisent l'intérieur de rouleaux carbonisés.

Dans les musées, l'IA aide à cataloguer les collections, à identifier des œuvres volées, à analyser les pigments et les techniques. Elle permet aussi de créer des expériences immersives, où le visiteur explore une œuvre ou une époque de manière interactive.

L'IA devient un outil de préservation du patrimoine, mais aussi un moyen de le rendre accessible à tous.

L'IA pour l'inclusion : rendre la société plus accessible

Les technologies d'assistance basées sur l'IA transforment la vie quotidienne des personnes en situation de handicap. Les systèmes de reconnaissance vocale permettent aux personnes à mobilité réduite de contrôler leur environnement. Les applications de description d'images aident les personnes malvoyantes à comprendre ce qui les entoure. Les outils de synthèse vocale donnent une voix à ceux qui ne peuvent pas parler.

L'IA contribue à réduire les barrières, à renforcer l'autonomie et à favoriser l'inclusion.

Conclusion : une technologie au service du bien commun

L'intelligence artificielle n'est pas seulement un moteur d'innovation ou un enjeu géopolitique. Elle est aussi un outil profondément humain, capable d'améliorer la santé, la sécurité, l'éducation, l'environnement, la culture et l'inclusion. Ces usages positifs ne sont pas des promesses lointaines : ils sont déjà là, dans les hôpitaux, les écoles, les villes, les laboratoires, les musées.

L'IA n'est pas une force neutre : elle reflète les intentions de ceux qui la conçoivent et de ceux qui l'utilisent.

Lorsqu'elle est orientée vers le bien commun, elle devient un formidable levier de progrès.

10.3.10 L'IA et l'emploi — les métiers qui naissent, ceux qui changent, ceux qui vacillent

L'intelligence artificielle n'est pas seulement une révolution technique : c'est une transformation profonde du travail humain. Comme la machine à vapeur ou l'électricité en leur temps, l'IA redistribue les tâches, modifie les compétences nécessaires et crée de nouveaux métiers. Contrairement aux discours catastrophistes, l'IA ne supprime pas simplement des emplois : elle en déplace, en transforme, et en crée. Mais elle le fait à une vitesse inédite, ce qui explique l'inquiétude qu'elle suscite.

Ce chapitre explore ces dynamiques, en s'appuyant sur des données récentes et des exemples concrets.

Les nouveaux métiers nés de l'IA

L'IA ne se contente pas d'automatiser : elle engendre des professions qui n'existaient pas il y a dix ans. Certaines sont devenues indispensables dans les entreprises, les laboratoires, les administrations.

Les ingénieurs en IA et en données

Les métiers de la donnée ont explosé. Selon l'OCDE, les offres d'emploi liées à la data science ont été multipliées par 4 entre 2015 et 2023.

On y trouve : data scientists, data engineers, machine learning engineers, spécialistes en MLOps.

Ces métiers structurent les pipelines de données, entraînent les modèles, surveillent leur comportement.

Les "prompt engineers"

Avec l'arrivée des modèles génératifs, une nouvelle profession est apparue : l'ingénieur en prompts. Il conçoit des instructions optimisées pour obtenir des résultats fiables des modèles de langage ou d'image.

Certaines entreprises américaines proposent des salaires dépassant 200 000 dollars par an pour ces profils hybrides, à mi-chemin entre la linguistique, la logique et la programmation.

Les spécialistes de l'éthique et de la gouvernance de l'IA

Les grandes organisations recrutent désormais des : AI ethicists, fairness analysts, responsables de conformité IA.

Leur rôle : garantir que les modèles respectent les normes, évitent les biais, protègent les données.

Selon Deloitte, les postes liés à l'éthique de l'IA ont augmenté de 67 % entre 2020 et 2024.

Les métiers de la supervision humaine

L'IA ne fonctionne pas seule : elle doit être corrigée, annotée, guidée.

Cela crée des métiers comme : annotateurs de données, superviseurs de modèles, testeurs de robustesse, modérateurs assistés par IA.

Ces métiers sont souvent invisibles, mais essentiels.

Les métiers transformés : l'IA comme copilote

La majorité des emplois ne disparaissent pas : ils changent. L'IA agit comme un assistant, un accélérateur, un outil d'augmentation.

Les professions intellectuelles

Les métiers du savoir sont parmi les plus transformés : juristes, journalistes, analystes financiers, consultants, enseignants.

L'IA rédige des brouillons, résume des documents, analyse des données, prépare des dossiers.
Selon McKinsey, l'IA pourrait automatiser 20 à 30 % des tâches des professions intellectuelles d'ici 2030, mais rarement l'ensemble du métier.

Les métiers de la santé

L'IA aide à : analyser des images médicales, détecter des anomalies, prédire des risques, optimiser les plannings hospitaliers.
Elle ne remplace pas les médecins, mais elle modifie leur pratique.
Selon une étude du Lancet, l'IA réduit de 15 à 25 % le temps consacré aux tâches administratives dans certains hôpitaux.

Les métiers de la création

L'IA générative transforme : le design, la publicité, l'illustration, la musique, la vidéo.
Les créatifs deviennent des directeurs artistiques augmentés, capables de produire plus vite, d'explorer plus d'idées, de tester des variantes.
L'IA ne remplace pas la créativité humaine, mais elle en modifie les outils.

Les métiers menacés : l'automatisation silencieuse

Certaines professions sont plus exposées que d'autres. Il ne s'agit pas d'une disparition brutale, mais d'une érosion progressive.

Les tâches répétitives et prévisibles

Les métiers les plus vulnérables sont ceux où les tâches sont : routinières, structurées, basées sur des règles claires.
Exemples :

- opérateurs de saisie,
- assistants administratifs,
- téléconseillers,
- contrôleurs qualité simples.

Selon Goldman Sachs, jusqu'à 300 millions d'emplois pourraient être partiellement automatisés dans le monde, mais rarement supprimés entièrement.

Les métiers de la logistique et du transport

Les systèmes autonomes progressent : robots d'entrepôt, véhicules autonomes, optimisation des flux.
Les chauffeurs, préparateurs de commandes et manutentionnaires voient leurs tâches évoluer.
L'automatisation totale reste lointaine, mais l'automatisation partielle est déjà là.

Les métiers de la relation client

Les chatbots et assistants vocaux prennent en charge une partie des interactions simples.
Selon Gartner, 70 % des interactions de service client pourraient être automatisées d'ici 2030.
Les agents humains se concentrent sur les cas complexes.

L'IA crée plus d'emplois qu'elle n'en détruit… mais pas pour les mêmes personnes

Les études convergent : l'IA crée globalement plus d'emplois qu'elle n'en supprime, mais elle déplace les compétences.

- Les emplois créés sont souvent qualifiés.
- Les emplois menacés sont souvent peu qualifiés.
- La transition nécessite de la formation.

Selon le Forum économique mondial, 50 % des travailleurs devront acquérir de nouvelles compétences d'ici 2027. La question n'est donc pas la disparition du travail, mais la capacité des sociétés à accompagner ces transitions.

Conclusion : l'IA ne remplace pas l'humain, elle redéfinit son rôle

L'IA n'est ni un bulldozer qui écrase l'emploi, ni une baguette magique qui crée des métiers sans effort. Elle agit comme un réorganisateur du travail humain. Elle supprime certaines tâches, en crée d'autres, et transforme la majorité.

Le défi n'est pas technologique : il est social, éducatif, politique. Les sociétés qui réussiront seront celles qui sauront :

- former rapidement,
- redistribuer les compétences,
- accompagner les transitions,
- et intégrer l'IA comme un outil, non comme une menace.

L'IA ne remplace pas l'humain : elle redéfinit ce que l'humain peut faire.

11 ANNEXES

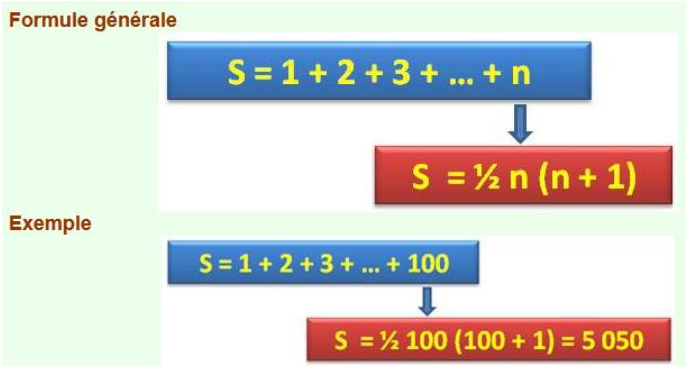
11.1 Défis et énigmes typiques

11.1.1 Arithmétique, nombres et curiosités numériques

La somme des entiers par le jeune Carl Friedrich Gauss

Une anecdote devenue légendaire : enfant, Gauss calcule en quelques secondes la somme de 1 à 100 en remarquant une structure cachée (1+100, 2+99, 3 + 98… => 50 × 101 = 5050).

Derrière ce tour d'esprit se cache une idée fondamentale : transformer un problème linéaire en structure symétrique.



ANECDOTE

La "véritable" histoire à propos de Gauss

Karl Friedrich **Gauss** (1777-1855) n'était pas un petit génie, mais un très grand génie.

L'anecdote courante de la somme de 1 à 100 est une simplification que la légende a retenue. En fait, le problème posé par le professeur Büttner était nettement plus déroutant pour l'élève moyen.

Voici le problème originel posé pour avoir la paix dans la classe. Gauss avait bien 10 ans. Non seulement, il donne sa réponse immédiatement. Une heure se passe, lui les bras croisés et les autres continuant à calculer. Finalement, seule son ardoise montre le bon résultat.

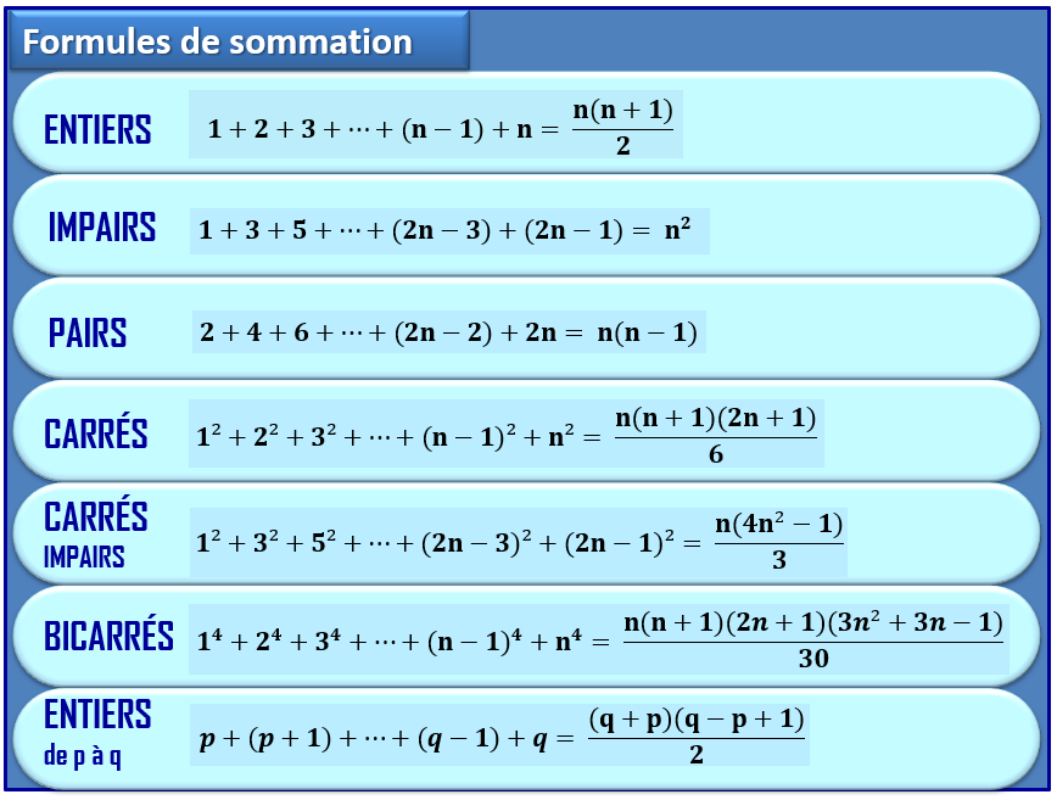
Problème posé

Additionnez les cent nombres suivants:

81 297 + 81 495 + 81 693 + … + 100 899 à chaque fois, il faut ajouter 198.

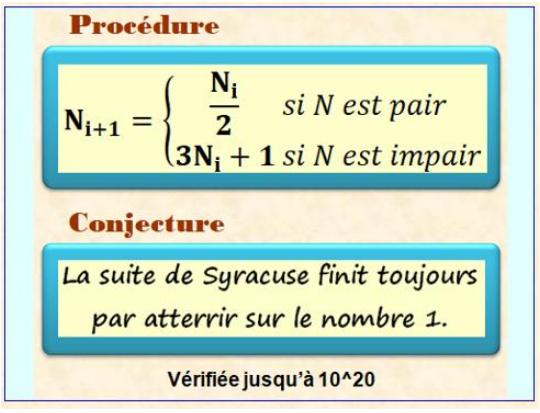
C'est une progression arithmétique: S = a·n + ½ r·n (n-1)

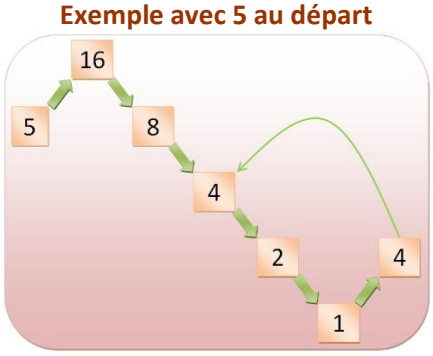
	a est le premier terme: 81 297; n est la quantité de termes: 100, et r est la raison: 198. Calcul: S = 81 297 x 100 + ½ 198 x 100 x 99 = 8 129 700 + 980 100 = 9 109 800 **Autres versions trouvées dans la littérature .** **Laquelle est la vraie? Mystère!** 1 + 2 + ... 100 176 + 195 + ...+ 2057 5 192 + 5229 + ... + 8792 81 297 + 81 495 + ... + 100 899

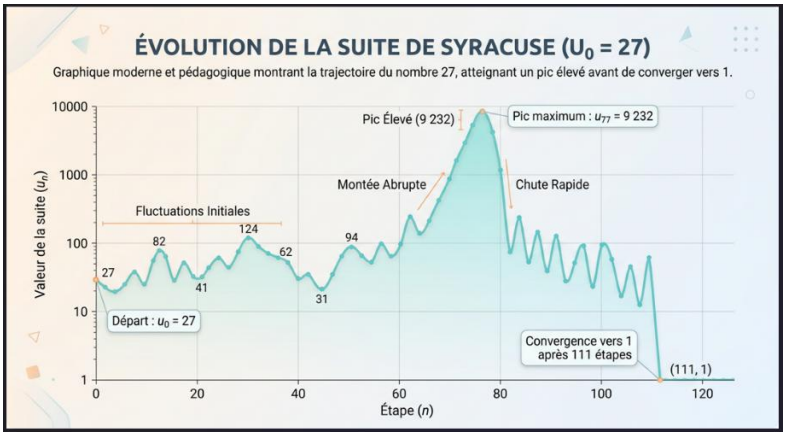


La suite de Syracuse (conjecture de Collatz)

On prend un entier : s'il est pair, on le divise par 2 ; s'il est impair, on calcule 3n+1. Répéter ce processus semble toujours mener à 1… mais personne n'a réussi à le prouver. Une énigme simple à énoncer, vertigineuse à résoudre.







Le dernier théorème de Pierre de Fermat

Le dernier théorème de Fermat occupe une place singulière dans l'histoire des mathématiques : à la fois simple à énoncer, vertigineux dans ses implications, et entouré d'une aura presque mythique.

Formulé par Pierre de Fermat au XVII^{e} siècle, il affirme qu'il n'existe **aucune solution entière non triviale** à l'équation:

$$X^n + Y^n = Z^n$$

dès que l'exposant (n) est strictement supérieur à 2.

Pour n = 2, l'équation décrit les triplets pythagoriciens, connus depuis l'Antiquité. Mais Fermat prétendait avoir démontré que, pour tout (n > 2), une telle égalité était impossible. Il ajouta même, dans la marge de son exemplaire de Diophante, qu'il tenait une « merveilleuse démonstration » trop longue pour y tenir. Cette note marginale devint l'une des phrases les plus célèbres de l'histoire scientifique.

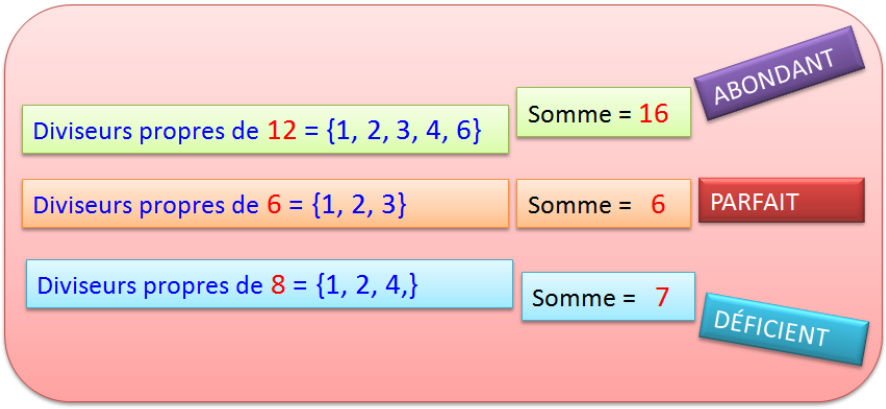
Pendant plus de trois siècles, le théorème résista à toutes les tentatives. Les plus grands mathématiciens s'y attaquèrent, souvent en développant au passage des pans entiers de nouvelles théories : théorie des nombres algébriques, formes modulaires, courbes elliptiques… Chaque progrès semblait rapprocher la communauté du but, sans jamais l'atteindre. Le problème devint un symbole : celui d'un défi intellectuel absolu, d'une énigme posée par un génie du passé et qui défiait les outils modernes.

Il fallut attendre 1994 pour que l'histoire bascule. Après des années de travail solitaire, Andrew Wiles, mathématicien britannique, parvint à établir un lien décisif entre les courbes elliptiques et les formes modulaires, s'appuyant sur une conjecture profonde de Taniyama–Shimura. En démontrant un cas crucial de cette conjecture, il fit tomber le dernier théorème de Fermat comme une conséquence. Une erreur fut détectée dans sa première version, mais Wiles, avec Richard Taylor, la corrigea quelques mois plus tard.

La démonstration, d'une sophistication extrême, n'a rien à voir avec ce que Fermat aurait pu imaginer. Pourtant, elle clôt magistralement une quête de plusieurs siècles, transformant une simple note marginale en triomphe de la pensée mathématique moderne.

Les nombres parfaits

Un nombre est parfait s'il est égal à la somme de ses diviseurs propres (6, 28, 496, 8128, 33550336 …). On en connait seulement huit. Connus depuis l'Antiquité, ils restent mystérieux : on ignore encore s'il existe des nombres parfaits impairs.



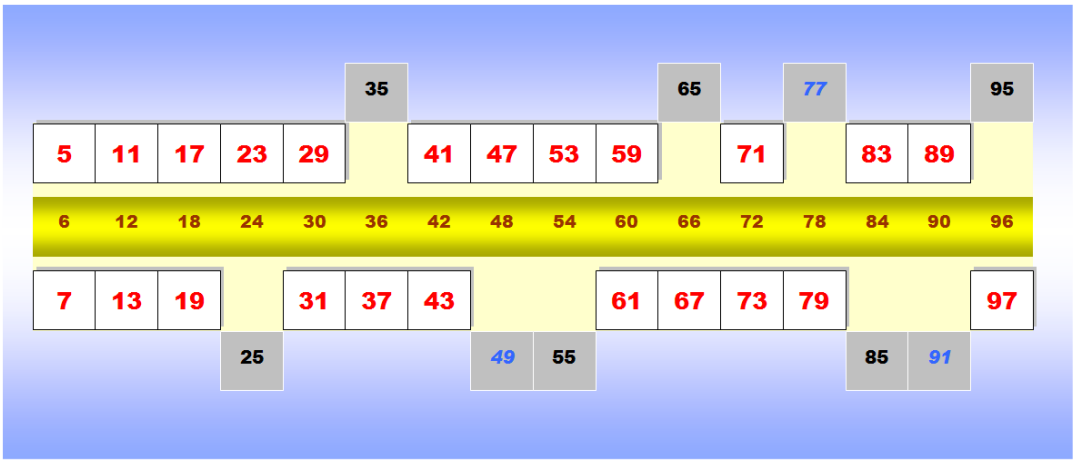
Les nombres amicaux

Deux nombres sont amicaux si chacun est la somme des diviseurs de l'autre (220 et 284; 1184 et 1210; 2620 et 2924; …). Une relation presque "affective" entre entiers, étudiée depuis les pythagoriciens.

Paires	**220**	**284**
Diviseurs propres	**1, 2, 4, 5, 10, 11, 20, 22, 44, 55 et 110**	**1, 2, 4, 71 et 142**
Somme	**284**	**220**

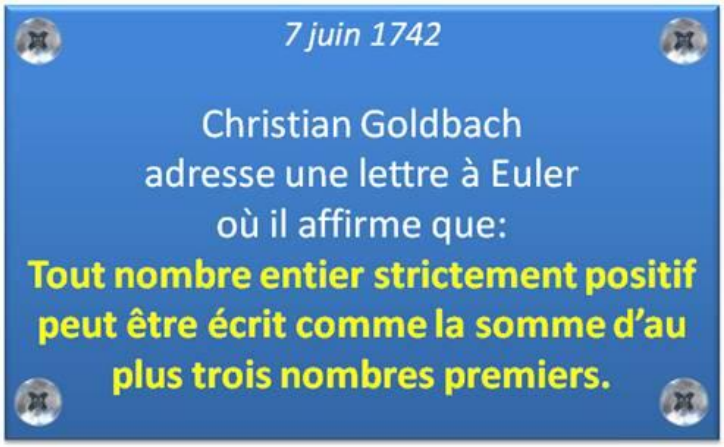
Les nombres premiers jumeaux

Des paires de nombres premiers séparés par 2 (11 et 13, 17 et 19…). La question de leur infinité reste ouverte, malgré des avancées spectaculaires. La barre magique des nombres premiers (illustration) montre la répartition des nombres premiers exclusivement autour des multiples de 6 (barre jaune au centre). Le nombres en vis-à-vis (même colonne) sont des nombres premiers jumeaux (exemple: 71 et 73).



La conjecture de Goldbach

Tout nombre pair supérieur à 2 serait la somme de deux nombres premiers. Vérifiée jusqu'à des tailles astronomiques, mais toujours sans preuve générale.



$$\zeta(z) = \sum_{n=1}^{\infty} \frac{1}{n^z}$$

L'hypothèse de Bernhard Riemann

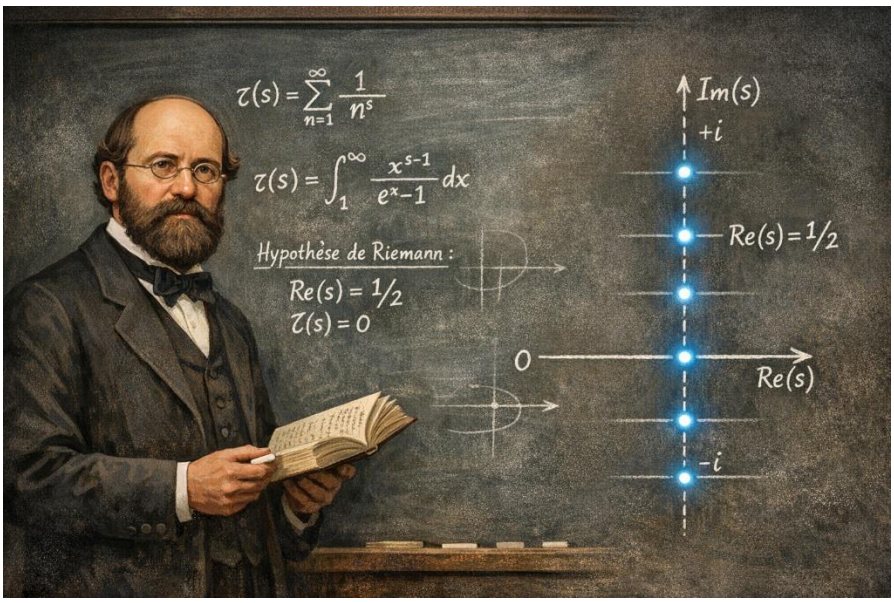
L'hypothèse de Riemann n'est pas seulement un monument de la théorie des nombres : elle est devenue, au fil du temps, un repère fondamental pour comprendre les limites et les ambitions des algorithmes modernes.

Formulée en 1859 par Bernhard Riemann dans un article d'une concision déconcertante, elle propose que tous les zéros non triviaux de la fonction zêta se situent sur une même droite verticale du plan complexe. Derrière cette phrase se cache une structure mathématique d'une profondeur telle qu'elle influence encore aujourd'hui la manière dont on conçoit, analyse et optimise les algorithmes.

Pourquoi un livre sur les algorithmes s'intéresse-t-il à une conjecture vieille de plus d'un siècle ? Parce que la distribution des nombres premiers — que l'hypothèse de Riemann décrit avec une précision presque parfaite — est au cœur de nombreux procédés computationnels. Les méthodes de factorisation, les tests de primalité, les schémas cryptographiques, mais aussi certaines bornes de complexité reposent sur des estimations fines de la répartition des premiers. Si l'hypothèse est vraie, elle garantit que ces estimations sont optimales ; si elle est fausse, une partie de l'architecture théorique de nos algorithmes devrait être réévaluée.

Depuis plus de 150 ans, les mathématiciens et informaticiens ont vérifié des milliards de zéros, tous alignés sur la fameuse droite critique. Cette quête a stimulé le développement d'algorithmes de calcul haute précision, de méthodes spectrales, de techniques de compression numérique et même de modèles inspirés de la physique quantique. L'hypothèse de Riemann est ainsi devenue un moteur indirect d'innovation algorithmique.

Elle demeure pourtant non démontrée. Mais c'est précisément cette tension — entre une conjecture simple à énoncer et un océan de complexité analytique — qui en fait un chapitre incontournable dans l'histoire des algorithmes : un rappel que, derrière chaque procédure efficace, se cache souvent une question profonde encore ouverte.



La constante de Kaprekar (6174)

Prenez un nombre à 4 chiffres, réordonnez ses chiffres, l'un par ordre croissant et l'autres par ordre décroissant soustrayez… et répétez : vous tombez toujours sur 6174 (sauf pour les nombres uniformes – repdigits – comme 5555).

Une dynamique numérique étonnamment stable.

Exemples

1000	1001	1234	5005	9876
1000 - 0001 = 0999			5500 - 0055 = 5445	
9990 - 0999 = 8991	1100 - 0011 = 1089		5544 - 4455 = 1089	
9981 - 1899 = 8082	9810 - 0189 = 9621	4321 - 1234 = 3087	9810 - 0189 = 9621	9876 - 6789 = 3087
8820 - 0288 = 8532	9621 - 1269 = 8352	8730 - 0378 = 8352	9621 - 1269 = 8352	8730 - 0378 = 8352
8532 - 2358 = 6174	8532 - 2358 = 6174	8532 - 2358 = 6174	8532 - 2358 = 6174	8532 - 2358 = 6174

L'hôtel de David Hilbert

L'hôtel de David Hilbert est un classique pour introduire la logique de l'infini et ses paradoxes.

Imaginez un hôtel doté d'une infinité de chambres, toutes occupées. Intuitivement, on penserait qu'aucun nouveau client ne peut être accueilli. Pourtant, Hilbert montre qu'il suffit de déplacer chaque occupant de la chambre n vers la chambre n+1. La première chambre se libère aussitôt, prête à recevoir un nouvel arrivant. Plus étonnant encore, l'hôtel peut absorber une infinité supplémentaire de clients en envoyant chaque occupant vers la chambre 2n, libérant ainsi toutes les chambres impaires.

Ce paradoxe, simple mais déstabilisant, révèle que l'infini ne se comporte pas comme les quantités finies et prépare l'esprit aux raisonnements non intuitifs que l'on rencontre dans de nombreux algorithmes.

Le problème des bœufs d'Hélios d'Archimède

Le problème des bœufs d'Hélios, attribué à Archimède, illustre de manière spectaculaire la puissance — et parfois la démesure — des raisonnements mathématiques.

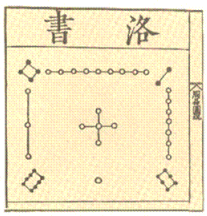
Le texte antique propose un système d'équations décrivant la répartition d'un troupeau mythique appartenant au dieu Hélios. Les premières conditions se résolvent aisément, mais les dernières imposent des contraintes diophantiennes d'une complexité redoutable.

Leur solution complète n'a été obtenue qu'à l'époque moderne, grâce à des méthodes avancées de théorie des nombres et à la puissance de calcul informatique. Le résultat est vertigineux : le troupeau minimal satisfaisant toutes les conditions compte 7 × 10^206 544 bœufs, un nombre si gigantesque qu'il dépasse de très loin toute représentation physique possible.

Ce problème montre comment un énoncé simple peut mener à des structures numériques d'une ampleur inimaginable.

Les carrés magiques et Sudoku

Les **carrés magiques** occupent une place singulière dans l'histoire des mathématiques : à la fois objets ludiques, symboles mystiques et structures combinatoires d'une grande élégance. Un carré magique est une grille, le plus souvent carrée, dans laquelle les nombres sont disposés de sorte que chaque ligne, chaque colonne et, dans les versions les plus classiques, les deux diagonales principales aient la même somme. Cette somme, appelée *constante magique*, dépend de la taille du carré et de l'ensemble des nombres utilisés. Les premiers carrés magiques connus remontent à la Chine ancienne : le célèbre carré *Lo Shu (illustration)*, un 3×3 parfait, était déjà associé à des croyances cosmologiques. On en retrouve ensuite en Inde, dans le monde arabe, puis en Europe, où ils inspirent artistes, mathématiciens et même alchimistes. Dürer, dans sa gravure *Melencolia I (Illustration)*, glisse un carré magique 4×4 comme un clin d'œil à la fascination de son époque pour ces structures.



Derrière leur apparente simplicité se cache une richesse combinatoire considérable. Construire un carré magique n'est pas seulement un exercice esthétique : c'est un problème algorithmique. Pour un carré d'ordre 3, il n'existe qu'une solution fondamentale (les autres s'obtenant par symétries). Mais dès que l'on augmente la taille, le nombre de configurations possibles explose. Les méthodes de construction varient : algorithmes de décalage pour les carrés d'ordre impair, techniques de décomposition pour les ordres multiples de 4, stratégies hybrides pour les ordres pairs non multiples de 4. Chaque famille de carrés possède ses recettes, ses invariants et ses subtilités. Étudier ces algorithmes, c'est plonger dans un univers où la combinatoire rencontre la géométrie et où la symétrie devient un outil de calcul.

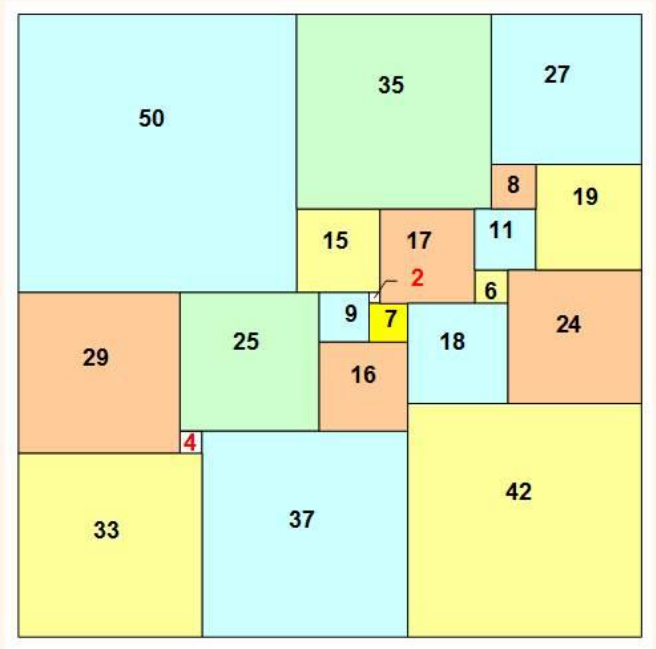
Cette tradition mathématique millénaire trouve un écho moderne dans un jeu devenu planétaire : le **Sudoku**. À première vue, il ne s'agit pas d'un carré magique : les lignes, colonnes et régions 3×3 ne doivent pas atteindre une somme particulière, mais contenir les chiffres de 1 à 9 sans répétition. Pourtant, la parenté conceptuelle est profonde. Comme les carrés magiques, le Sudoku repose sur des contraintes globales imposées à une grille locale. Comme eux, il combine logique, structure et esthétique. Et surtout, il mobilise des techniques algorithmique puissantes : propagation de contraintes, backtracking, heuristiques de choix, stratégies d'élimination. Les chercheurs en informatique théorique ont montré que la résolution générale du Sudoku est un problème NP-complet, ce qui signifie qu'il appartient à une classe de problèmes réputés difficiles à résoudre efficacement dans tous les cas.

L'intérêt du Sudoku, comme celui des carrés magiques, dépasse largement le simple divertissement. Ces grilles constituent des laboratoires miniatures pour explorer la manière dont les algorithmes gèrent les contraintes, réduisent les espaces de recherche et exploitent les symétries. Elles permettent d'illustrer des notions fondamentales : complexité, optimisation, structures combinatoires, logique déductive. Elles montrent aussi comment un problème apparemment simple peut se révéler d'une profondeur insoupçonnée.

Ainsi, des carrés magiques antiques aux Sudokus contemporains, une même fascination perdure : celle de voir l'ordre émerger du chaos, la structure surgir de la contrainte, et la beauté mathématique se révéler dans une simple grille.

Le pavage d'un carré par des carrés distincts (square packing)

Peut-on découper un carré en carrés tous de tailles différentes ? Oui, mais les constructions sont d'une subtilité extrême.



Les nombres de Catalan

$$33 = 8\,866\,128\,975\,287\,528^3 + (-8\,778\,405\,442\,862\,239)^3 + (-2\,736\,111\,468\,807\,040)^3$$

Les nombres de Catalan forment l'une des suites les plus étonnantes et les plus omniprésentes de la combinatoire. Derrière leur définition apparemment technique se cache un véritable fil conducteur reliant une multitude de structures discrètes. Le (n)-ième nombre de Catalan compte, entre autres, le nombre de façons de parenthéser correctement une expression comportant (n+1) termes, le nombre d'arbres binaires à (n) nœuds internes, ou encore le nombre de chemins sur une grille qui ne franchissent jamais la diagonale principale. Cette diversité d'interprétations en fait un outil conceptuel puissant : une même valeur numérique éclaire simultanément des objets très différents.

Sur le plan algorithmique, les nombres de Catalan apparaissent dès que l'on impose des contraintes d'équilibre ou de structure hiérarchique. Ils interviennent dans l'analyse de piles, dans la génération d'arbres syntaxiques, dans la construction de parcours équilibrés ou dans la résolution de problèmes de comptage liés aux parenthésages. Leur formule fermée,

$$C_n = \frac{1}{n+1}\binom{2n}{2}$$

illustre la manière dont la combinatoire encode des symétries profondes.

La somme de trois cubes

La somme de trois cubes consiste à trouver des entiers a, b et c tels que: $a^3 + b^3 + c^3 = k$.

Ce problème, en apparence élémentaire, s'est révélé d'une profondeur remarquable. Pour certains entiers, les solutions sont simples : par exemple, $3 = 1^3 + 1^3 + 1^3$. Mais pour d'autres valeurs,

notamment celles congrues à 4 ou , aucune solution n'existe. Les cas restants forment un terrain de jeu fascinant où intuition et calcul se heurtent à la complexité diophantienne.

Pendant des décennies, certains petits entiers ont résisté à toutes les tentatives. Le cas k = 33 n'a été résolu qu'en 2019, avec une solution gigantesque impliquant des entiers d'environ 16 chiffres. Plus spectaculaire encore, le cas k = 42, longtemps considéré comme un défi emblématique, a été résolu la même année grâce à une combinaison d'algorithmes de recherche distribuée et de calcul intensif. La solution fait intervenir des entiers de l'ordre de 10^{16}, illustrant la difficulté du problème.

Ces avancées montrent comment un énoncé simple peut nécessiter des techniques modernes : optimisation de l'espace de recherche, méthodes de crible, parallélisation massive. La somme de trois cubes demeure un exemple parfait de la manière dont la théorie des nombres, l'algorithmique et la puissance de calcul se rencontrent pour repousser les frontières du possible.

Les partitions selon G. H. Hardy et Srinivasa Ramanujan

Les partitions d'un entier constituent l'un des thèmes les plus féconds de la combinatoire additive. Le principe est simple : une *partition* de (n) est une manière d'écrire (n) comme somme d'entiers positifs, sans tenir compte de l'ordre des termes. Par exemple, l'entier 5 possède exactement sept partitions : [5; 4+1; 3+2; 3+1+1; 2+2+1; 2+1+1+1; 1+1+1+1+1.] Cette simplicité apparente masque une structure d'une profondeur remarquable. Le nombre de partitions de n, noté p(n), croît extrêmement vite : p(10)=42, p(50)=204226, et p(100)) dépasse déjà les 190 millions. Comprendre cette croissance fut l'un des défis majeurs du début du XXᵉ siècle.

C'est dans ce contexte que G. H. Hardy et Srinivasa Ramanujan ont révolutionné le domaine. Leur collaboration a conduit à une formule asymptotique d'une élégance saisissante, révélant la structure cachée derrière la croissance explosive de p(n). Leur résultat principal affirme que, pour de grands n:

$$p(n) \approx \frac{1}{4n\sqrt{3}}\, exp\left(\pi\sqrt{\frac{2n}{3}}\right)$$

Cette formule, aujourd'hui appelée *formule de Hardy–Ramanujan*, montre que le nombre de partitions est essentiellement gouverné par une exponentielle de √n, un comportement inattendu pour une fonction définie par un simple problème de décomposition additive. Elle constitue l'un des premiers exemples spectaculaires de l'analyse asymptotique appliquée à la combinatoire.

Au-delà de cette approximation, Hardy et Ramanujan ont introduit la *méthode du cercle*, une technique analytique puissante permettant d'extraire des informations précises à partir de séries génératrices. Leur approche a ouvert la voie à des résultats encore plus fins, notamment grâce à Hans Rademacher, qui a obtenu une formule exacte pour p(n) sous forme de série convergente. Cette formule, bien que complexe, permet de calculer p(n) avec une précision arbitraire sans énumérer toutes les partitions.

Sur le plan algorithmique, les partitions jouent un rôle central dans de nombreux domaines : optimisation, cryptographie, théorie des représentations, analyse des algorithmes récursifs. Les séries génératrices associées aux partitions, comme:

$$\sum_{n=0}^{\infty} p(n)q^n \;=\; \prod_{k=1}^{\infty}\frac{1}{1-q^k}$$

sont devenues des outils fondamentaux pour comprendre les structures combinatoires complexes. Elles permettent de relier les partitions à des objets aussi variés que les diagrammes de Ferrers, les représentations du groupe symétrique ou les formes modulaires.

Les avancées récentes ont prolongé l'héritage de Hardy et Ramanujan. Des méthodes issues de la théorie analytique des nombres, de la géométrie algébrique et même de la physique statistique ont permis d'étudier des variantes sophistiquées : partitions restreintes, partitions plane, partitions en parts distinctes ou bornées. Les algorithmes modernes exploitent ces structures pour calculer efficacement p(n) pour des valeurs très grandes, parfois supérieures à plusieurs millions.

Ainsi, derrière la question naïve « Combien de façons d'écrire un entier comme somme d'autres entiers ? » se cache un univers mathématique d'une richesse insoupçonnée. Les partitions, révélées par Hardy et Ramanujan, montrent comment une idée simple peut ouvrir la porte à des théories profondes, reliant combinatoire, analyse et algorithmique dans une harmonie inattendue.

Les nombres de Mersenne

Les nombres de Mersenne, introduits au XVII^{e} siècle par le moine français Marin Mersenne, occupent une place centrale dans la quête des plus grands nombres premiers connus. Ils sont définis par la formule;

$$M_p = 2^p - 1$$

où p est un entier. Lorsqu'un nombre de Mersenne est premier, on parle de *nombre premier de Mersenne*. Cette forme particulière n'est pas choisie au hasard : elle permet d'exploiter des algorithmes de test de primalité extrêmement efficaces, bien plus rapides que pour des entiers généraux de taille comparable.

Un premier point fondamental est que **si ($2^p - 1$) est premier, alors p doit lui-même être premier**. Cela réduit considérablement l'espace de recherche, même si tous les exposants premiers ne donnent pas un nombre de Mersenne premier. Par exemple, p = 11 est premier, mais $2^{11} - 1$ = 2047 = 23 × 89 est composé. En revanche, pour p = 13, on obtient $2^{13} - 1$ = 8191 qui est bien premier.

La clé algorithmique de cette quête est le test de Lucas–Lehmer, un algorithme remarquablement efficace pour déterminer si un nombre de Mersenne est premier. Grâce à lui, il est possible de tester des nombres comportant des dizaines de millions de chiffres, ce qui serait totalement irréaliste pour des entiers généraux de même taille.

Depuis 1996, la recherche est portée par le projet collaboratif GIMPS (Great Internet Mersenne Prime Search), qui fédère des milliers d'ordinateurs volontaires. Cette approche distribuée a permis de repousser régulièrement les limites de la primalité connue.

Le record actuel, découvert en décembre 2023, est: 2^136 279 841 – 1 un nombre gigantesque de 41 024 320 chiffres**.** Il dépasse largement le précédent record (2018), qui comptait déjà près de 25 millions de chiffres. Ce nouveau jalon illustre la puissance combinée des mathématiques, des algorithmes spécialisés et du calcul distribué moderne.

Les nombres de Mersenne ne sont pas de simples curiosités numériques. Ils interviennent dans la génération de nombres pseudo-aléatoires, dans l'étude des réseaux, et dans la théorie des nombres parfaite. Le lien avec les nombres parfaits pairs est particulièrement élégant : Euclide et Euler ont montré que tout nombre parfait pair s'écrit:

$$2^{p-1}(2^p - 1)$$

est un nombre premier de Mersenne. Ainsi, chaque nouveau record de Mersenne engendre automatiquement un nouveau record de nombre parfait.

La quête des nombres de Mersenne est donc un terrain où se rencontrent théorie des nombres, algorithmique avancée et calcul collaboratif. Elle montre comment une formule simple peut ouvrir

la voie à des explorations numériques d'une ampleur vertigineuse, et comment la recherche collective continue de repousser les frontières du calcul.

rank	prime	digits	who	when	comment
1	$2^{136279841}-1$	41024320	MP1	2024	Mersenne 52?
2	$2^{82589933}-1$	24862048	G16	2018	Mersenne 51?
3	$2^{77232917}-1$	23249425	G15	2018	Mersenne 50
4	$2^{74207281}-1$	22338618	G14	2016	Mersenne 49
5	$2^{57885161}-1$	17425170	G13	2013	Mersenne 48
6	$2^{43112609}-1$	12978189	G10	2008	Mersenne 47
7	$2^{42643801}-1$	12837064	G12	2009	Mersenne 46
8	$2^{37156667}-1$	11185272	G11	2008	Mersenne 45
9	$2^{32582657}-1$	9808358	G9	2006	Mersenne 44
10	$2^{30402457}-1$	9152052	G9	2005	Mersenne 43

Les dix nombres de Mersenne premiers les plus grands connus (avril 2026)
Source: The Largest Known Primes – A Summary – https://t5k.org/largest.html

Les nombres de Fermat

Les nombres de Fermat sont définis par la formule: $(2^{2^n}+1)$.

Pierre de Fermat conjecturait au XVII[e] siècle que tous ces nombres étaient premiers, impressionné par les premiers cas : $F_0 = 3$, $F_1 = 5$; $F_2 = 17$, $F_3 = 257$, $F_4 = 65\,537$, tous effectivement premiers. Mais cette belle régularité s'effondre rapidement. En 1732, Euler démontre que:

$$F_5 = 4\,294\,967\,297 = 641 \;\; 6\,700\,417$$

réfutant la conjecture.

Depuis, tous les nombres de Fermat connus pour n ≥ 5 se sont révélés composés, et aucun nouveau nombre premier de Fermat n'a été découvert. Leur étude reste pourtant essentielle : ils jouent un rôle dans la construction de polygones réguliers, dans certaines structures algébriques et dans l'analyse de la primalité à grande échelle.

Le paradoxe de Simpson

Le paradoxe de Simpson désigne une situation où une tendance observée dans plusieurs groupes séparés s'inverse lorsque l'on regroupe ces mêmes données. Ce phénomène, contre-intuitif mais fréquent, montre à quel point l'interprétation statistique peut être trompeuse si l'on ignore la structure des données. Le paradoxe apparaît généralement lorsqu'une *variable cachée* — un facteur de confusion — influence différemment les sous-groupes.

Un exemple classique concerne des taux de réussite. Deux traitements médicaux peuvent sembler plus efficaces chacun dans leur groupe de patients respectif, mais, une fois les données combinées, le traitement auparavant défavorisé apparaît soudain meilleur. La raison : les groupes n'avaient pas la même taille ou pas la même répartition de cas simples et difficiles. Le regroupement masque alors l'information pertinente.

Ce paradoxe a des implications majeures en médecine, en économie, en sciences sociales et dans l'évaluation d'algorithmes. Il rappelle qu'une moyenne globale peut cacher des disparités profondes et que les conclusions doivent toujours être contextualisées. Comprendre le paradoxe de Simpson, c'est apprendre à se méfier des chiffres trop simples et à analyser les données avec rigueur et nuance.

Le problème de la factorisation

Décomposer un grand nombre en facteurs premiers est difficile — et cette difficulté est au cœur de la cryptographie moderne.

La factorisation des entiers est l'un des problèmes les plus anciens et les plus fondamentaux des mathématiques. Il consiste à écrire un entier N comme produit de nombres premiers. Pour un petit nombre, l'exercice est trivial : $84 = 2^2 \times 3 \times 7$.

Mais lorsque N comporte plusieurs centaines ou milliers de chiffres, la tâche devient redoutablement complexe. Cette difficulté n'est pas un simple détail technique : elle constitue la base de la sécurité de nombreux systèmes cryptographiques, notamment le célèbre protocole RSA.

La raison profonde de cette difficulté tient à la nature asymétrique du problème. Multiplier deux grands nombres premiers est facile, même pour un ordinateur modeste. En revanche, retrouver ces facteurs à partir du produit est extrêmement difficile. Cette dissymétrie est au cœur de la cryptographie à clé publique : on peut publier $N = pq$ sans risque, tant que personne ne parvient à retrouver p et q.

Sur le plan algorithmique, la factorisation est un terrain d'expérimentation fascinant. Les méthodes naïves, comme la division par tous les entiers jusqu'à $\sqrt{N}$, deviennent inutilisables dès que N dépasse quelques dizaines de chiffres. Les algorithmes modernes sont bien plus sophistiqués. Le crible quadratique (Quadratic Sieve) fut longtemps l'outil de référence : il exploite des congruences quadratiques pour construire des relations permettant de remonter aux facteurs. Il a ensuite été supplanté par le crible général du corps de nombres (GNFS, General Number Field Sieve), aujourd'hui l'algorithme le plus rapide connu pour factoriser des entiers généraux. Sa complexité est sous-exponentielle, mais reste suffisamment élevée pour garantir la sécurité des clés RSA de taille standard.

Un exemple célèbre illustre cette difficulté : la factorisation du nombre RSA-250, un entier de 829 bits, a nécessité en 2020 plusieurs milliers de cœurs-processeurs et des mois de calcul distribué. Pourtant, multiplier les deux facteurs obtenus ne demande qu'une fraction de seconde. Cette disproportion est précisément ce qui rend la factorisation si précieuse pour la cryptographie.

Mais le paysage évolue. L'arrivée potentielle des ordinateurs quantiques pourrait bouleverser cet équilibre. L'algorithme de Shor, découvert en 1994, permettrait en théorie de factoriser un entier en temps polynomial, rendant obsolètes les systèmes basés sur la factorisation. Pour l'instant, les machines quantiques capables d'exécuter Shor à grande échelle n'existent pas, mais leur développement motive la recherche de cryptosystèmes résistants au quantique.

La factorisation n'est donc pas seulement un problème mathématique : c'est un enjeu stratégique. Elle relie théorie des nombres, algorithmique avancée, architecture informatique et sécurité globale. Chaque progrès dans les algorithmes de factorisation ou dans la puissance de calcul oblige à réévaluer les tailles de clés recommandées. Inversement, chaque avancée cryptographique repose sur la confiance dans la difficulté persistante de ce problème.

Ainsi, derrière l'acte apparemment simple de décomposer un nombre se cache l'un des piliers de la sécurité numérique contemporaine. La factorisation est un rappel puissant : en algorithmique, la difficulté d'un problème peut devenir une ressource, et même un fondement de notre monde connecté.

Curiosités arithmétiques et dynamiques numériques

Les nombres heureux

On remplace un nombre par la somme des carrés de ses chiffres. Certains finissent par 1 (heureux), d'autres tombent dans des cycles.

Les nombres narcissiques

Un nombre égal à la somme de ses chiffres élevés à une puissance ($153 = 1^3+5^3+3^3$). Une forme d'auto-référence numérique.

Les nombres automorphes

Leur carré se termine par eux-mêmes (76 → 5776). Une propriété rare et élégante.

Les nombres polygonaux

Triangulaires, carrés, pentagonaux… Des nombres qui codent des figures géométriques.

Les nombres de Harshad

Divisibles par la somme de leurs chiffres. Simples à définir, mais riches en propriétés.

Les nombres puissants

Chaque facteur premier apparaît avec un exposant ≥ 2. Leur distribution reste mal comprise.

Les nombres chanceux

Construits par un crible analogue à celui d'Ératosthène. Leur ressemblance avec les nombres premiers intrigue.

Les nombres de Smith

La somme de leurs chiffres est égale à celle des chiffres de leur factorisation.

Les nombres de Carmichael

Des "faux premiers" qui trompent certains tests de primalité. Importants en cryptographie.

Suites et structures profondes

Les suites aliquotes

Les suites aliquotes sont obtenues en remplaçant un nombre n par la somme de ses diviseurs propres, c'est-à-dire tous ses diviseurs strictement inférieurs à lui.

Cette transformation répétée produit des comportements étonnamment variés. Par exemple, à partir de n = 12, on obtient:

12 → 1+2+3+4+6=16 → 1+2+4+8=15 → 1+3+5=9 → 1+3=4 → 1+2=3 → 1.

Certaines suites convergent vers 1, d'autres entrent dans des cycles (comme 220 ↔ 284), et certaines semblent croître indéfiniment. Leur dynamique, difficile à prédire, en fait un terrain fascinant pour l'étude algorithmique des comportements chaotiques.

Les cycles amicaux généralisés

Au-delà des paires amicales : des cycles de plusieurs nombres liés par leurs diviseurs.

Les nombres premiers de Fibonacci

Certains termes de la suite de Fibonacci sont premiers — mais en existe-t-il une infinité ?

Les nombres de Lucas

Une suite proche de Fibonacci, avec des propriétés arithmétiques distinctes.

Les nombres hautement composés

Les nombres hautement composés sont des entiers qui possèdent plus de diviseurs que tous les entiers plus petits qu'eux.

Introduits par Ramanujan dans un article devenu classique, ils illustrent la manière dont la structure multiplicative d'un nombre peut être optimisée. Leur forme repose souvent sur des produits de petites puissances de premiers, organisés de façon à maximiser le nombre total de diviseurs.

Un exemple simple est **12**, qui possède 6 diviseurs : 1, 2, 3, 4, 6, 12. Aucun entier inférieur n'en possède autant. Le suivant est **24**, avec 8 diviseurs, puis **36**, qui en compte 9.

Ces nombres jouent un rôle important dans l'analyse algorithmique, notamment pour comprendre la croissance de la fonction « nombre de diviseurs » et pour concevoir des algorithmes efficaces de factorisation ou de génération de diviseurs.

L'étude de Ramanujan a révélé des régularités profondes : les nombres hautement composés se situent souvent à la frontière entre ordre et optimisation, où la combinatoire des facteurs premiers devient un véritable art mathématique.

Les nombres superabondants

Ils maximisent le rapport somme des diviseurs / nombre. Liés à des questions profondes sur la distribution des entiers.

Les fonctions multiplicatives

Les fonctions multiplicatives jouent un rôle central en arithmétique.

Une fonction (f) définie sur les entiers est dite multiplicative si $f(ab) = f(a),f(b)$ pour tous entiers (a) et (b) premiers entre eux. Cette propriété permet de réduire l'étude d'une fonction à son comportement sur les puissances de nombres premiers, ce qui simplifie considérablement les calculs et les démonstrations.

De nombreux objets fondamentaux sont multiplicatifs. Par exemple :

- la fonction tau $\tau(n)$ qui compte les diviseurs de (n) ;
- la fonction sigma $\sigma(n)$, somme des diviseurs ;
- la fonction de Möbius $\mu(n)$, essentielle dans l'inversion de Möbius ;
- la fonction phi d'Euler $\varphi(n)$, qui compte les entiers ≤ n premiers avec n.

Ainsi, pour $12 = 2^2 \cdot 3$, on obtient: $\varphi(12) = \varphi(2^2) \cdot \varphi(3) = 2 \times 2 = 4$

Les fonctions multiplicatives structurent une grande partie de la théorie analytique des nombres : elles interviennent dans les séries de Dirichlet, les identités de convolution, les algorithmes de factorisation et l'étude fine de la distribution des entiers. Elles offrent un langage unifié pour comprendre comment les propriétés arithmétiques se décomposent le long des facteurs premiers.

Les suites de Hofstadter

Des suites auto-référentielles aux comportements chaotiques et imprévisibles.

La suite de Golomb

La suite de Golomb est une suite autodescriptive remarquable : chaque terme indique combien de fois il apparaît dans la suite. Elle commence ainsi :

1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, …

Le premier terme vaut 1, ce qui signifie que « 1 apparaît une fois ». Le deuxième terme vaut 2, donc « 2 apparaît deux fois », ce que l'on observe effectivement aux positions 2 et 3. Le terme suivant est encore 2, puis viennent deux 3, trois 4, trois 5, etc.

Cette construction produit une suite croissante, non triviale, qui se définit récursivement par/ $G(n+1) = 1 + G(n + 1 – G(G(n))$.] La suite de Golomb illustre comment une règle simple peut engendrer une structure algorithmique étonnamment riche.

Équations et défis classiques

Les équations diophantiennes

Les équations diophantiennes sont des équations dont on cherche des solutions entières. Elles portent le nom de Diophante d'Alexandrie, qui en étudia les premières formes. Leur simplicité apparente cache une diversité vertigineuse : certaines se résolvent par des méthodes élémentaires, d'autres mènent à des problèmes parmi les plus profonds des mathématiques, comme le dernier théorème de Fermat ou les courbes elliptiques.

L'un des exemples les plus fondamentaux est l'équation linéaire diophantienne: **ax + by = d,** où a, b et d sont des entiers. L'outil central pour l'étudier est l'**identité de Bézout**, qui affirme que l'on peut écrire: PGCD(a, b) = ax + by pour certains entiers x et y. Ainsi, l'équation ax + by = d admet

une solution entière si et seulement si (d) est un multiple du plus grand commun diviseur de a et b.

Par exemple, pour a = 14 et b = 21, on a PGCD(14, 21) = 7. L'identité de Bézout donne: è = 14(-1)+21(1). On en déduit que l'équation 14x + 21y = 7 possède des solutions, comme (x, y) = (-1, 1), et que toutes les autres s'obtiennent par une simple paramétrisation.

Cet exemple montre comment un outil algorithmique — l'algorithme d'Euclide — permet de résoudre efficacement une classe entière d'équations diophantiennes, ouvrant la voie à des applications en cryptographie, en théorie des codes et en arithmétique modulaire.

L'équation de Pell

Équation diophantienne de la forme: $x^2 - Dy^2 = 1$

Elle possède une infinité de solutions et cache une structure remarquable liée aux fractions continues.

L'algorithme qui a résolu un problème mathématique vieux de 90 ans

Le problème des « conjectures de Birch et Swinnerton-Dyer » est l'un des plus célèbres de la théorie des nombres.

En 2021, un algorithme d'apprentissage automatique a été utilisé pour explorer des millions de structures mathématiques et proposer des conjectures nouvelles. Il n'a pas « prouvé » quoi que ce soit — mais il a révélé des motifs que les mathématiciens n'avaient jamais envisagés. Certains de ces motifs ont ensuite été confirmés par des démonstrations humaines.

Pour la première fois, un algorithme a servi de boussole dans un territoire mathématique inexploré.

Un assistant qui ne comprend pas les mathématiques… mais qui les devine.

11.1.2 Combinatoire, graphes et optimisation

(Chemins, réseaux, structures discrètes et décisions optimales)

Le problème du voyageur de commerce (TSP)

Le problème du voyageur de commerce (TSP, *Travelling Salesman Problem*) est l'un des problèmes les plus emblématiques de l'informatique théorique. L'énoncé tient en une phrase : trouver le plus court chemin passant une seule fois par chaque ville et revenant au point de départ. Cette simplicité apparente masque une difficulté algorithmique redoutable. Avec n villes, le nombre de circuits possibles est (n-1)!, une croissance factorielle qui explose dès que n dépasse quelques dizaines. Pour 20 villes, il existe déjà plus de 120 millions de milliards de circuits possibles.

(20 – 1)! = 19! = 121 645 100 408 832 000

Le TSP est un problème NP-difficile, ce qui signifie qu'aucun algorithme connu ne peut le résoudre efficacement dans tous les cas. Pourtant, il apparaît dans des domaines très concrets : planification de tournées de livraison, optimisation de circuits imprimés, robotique, génomique (alignement de fragments), ou encore organisation de trajets logistiques.

Les méthodes exactes existent — programmation linéaire, séparation et évaluation, branch-and-bound — mais deviennent vite impraticables pour de grandes instances. C'est pourquoi une grande partie de la recherche s'est tournée vers des heuristiques et méta-heuristiques : algorithmes génétiques, recuit simulé, colonies de fourmis, recherche tabou. Ces méthodes ne garantissent pas la solution optimale, mais produisent en un temps raisonnable des solutions très proches de l'optimum.

Un exemple simple illustre la difficulté : pour quatre villes A, B, C et D, il suffit d'examiner trois circuits distincts. Mais pour 50 villes, le nombre de circuits dépasse largement le nombre d'atomes de l'univers observable. Aucun ordinateur ne peut les énumérer.

Le TSP est ainsi devenu un symbole de la frontière entre ce qui est calculable et ce qui ne l'est pas efficacement. Il rappelle que, dans l'univers des algorithmes, comprendre la complexité d'un problème est aussi important que trouver sa solution.

Les ponts de Königsberg et Leonhard Euler

Peut-on traverser chaque pont exactement une fois ? Euler démontre que c'est impossible, inaugurant la théorie des graphes et une nouvelle manière de penser les réseaux.

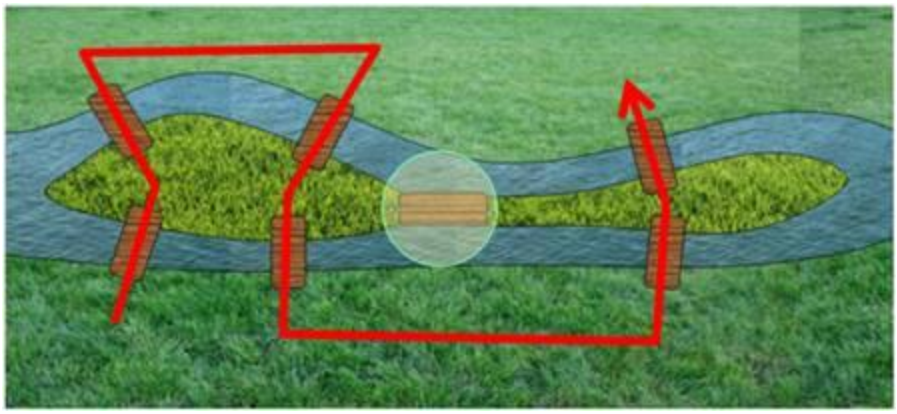
ÉNIGME

Les ponts de Königsberg — La naissance des graphes

Le principe : traverser une ville en passant une seule fois par chacun de ses ponts.

L'énigme

Au XVIII^{e} siècle, les habitants de Königsberg se demandaient s'il était possible de faire une promenade traversant une seule fois chacun des sept ponts de la ville. Euler démontre que c'est impossible — non pas en étudiant la géographie, mais en inventant une nouvelle idée : représenter la ville comme un graphe.



Pourquoi c'est algorithmique :

Euler montre qu'il est impossible de trouver un tel parcours.

> **Un graphe possède un parcours eulérien si et seulement si 0 ou 2 sommets ont un degré impair**.

À Königsberg, ils sont quatre. Fin de l'histoire !

Ce problème fondateur donne naissance à la théorie des graphes, aujourd'hui omniprésente dans les algorithmes (réseaux, GPS, Internet…).

Le saviez-vous ?

Euler a résolu le problème sans jamais dessiner la ville : il a inventé l'abstraction.

Le problème des quatre couleurs

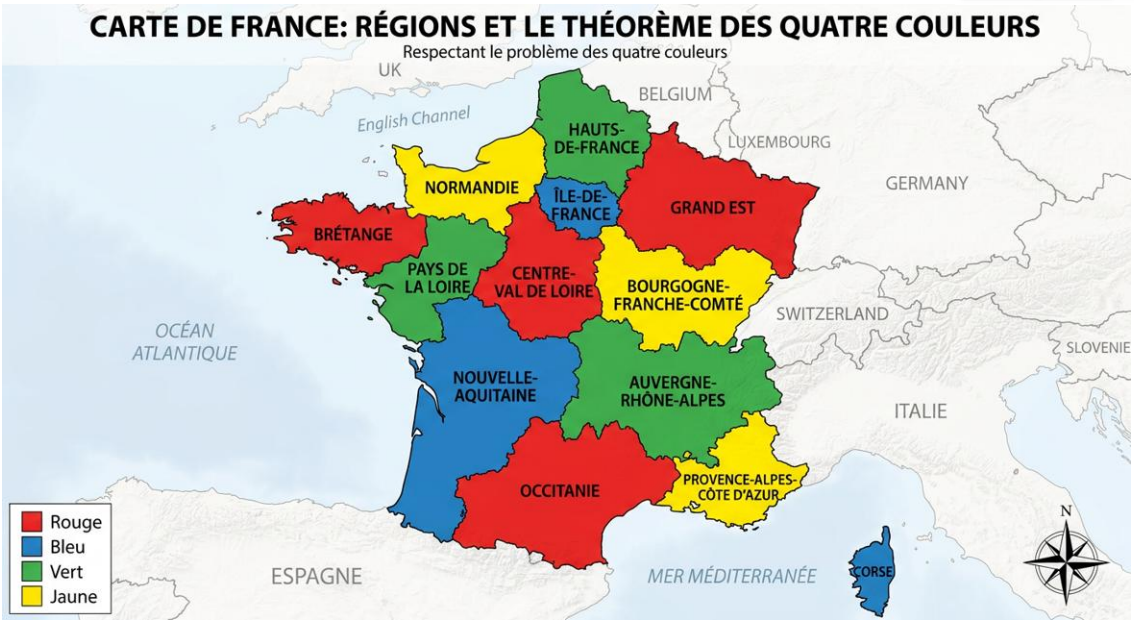
Le problème des quatre couleurs affirme qu'il suffit de quatre couleurs pour colorier n'importe quelle carte plane de façon que deux régions adjacentes ne partagent jamais la même couleur. Formulée au XIX^{e} siècle, cette conjecture semblait simple, presque naïve, mais elle résista à toutes les tentatives de démonstration pendant plus d'un siècle.

La difficulté vient du nombre immense de configurations possibles : il ne s'agit pas seulement de cartes géographiques, mais de tous les graphes planaires imaginables. En 1976, Appel et Haken

apportent enfin une preuve… mais une preuve révolutionnaire. Ils réduisent le problème à un ensemble de configurations critiques, puis utilisent un ordinateur pour vérifier des milliers de cas. C'est l'une des premières démonstrations assistées par machine, marquant un tournant dans l'histoire des mathématiques. Elle a suscité débats et révisions, mais reste aujourd'hui acceptée. Le problème des quatre couleurs illustre ainsi comment l'algorithmique peut devenir un outil essentiel pour résoudre des questions théoriques d'une complexité insoupçonnée.

Carré latin 3×3

A	B	C
C	A	B
B	C	A



Le problème des officiers de Leonhard Euler

Le problème des trente-six officiers fut formulé par Leonhard Euler en 1782. Il demande s'il est possible de disposer 36 officiers, appartenant à six régiments et à six grades, dans un carré de 6 × 6 de sorte que chaque ligne et chaque colonne contienne un officier de chaque régiment et de chaque grade, sans répétition.

Autrement dit, il s'agit de construire deux carrés latins orthogonaux d'ordre 6 : l'un représentant les régiments, l'autre les grades, chaque paire (régiment, grade) apparaissant une seule fois.

Un **carré latin** est une grille n × n remplie de n symboles distincts, chacun apparaissant une fois par ligne et par colonne. Par exemple, pour (n = 3), voir l'illustration.

Deux carrés latins sont dits **orthogonaux** si, lorsqu'on les superpose, toutes les combinaisons de symboles apparaissent exactement une fois. Le carré obtenu est alors appelé **carré gréco-latin**, car Euler utilisait des lettres grecques et latines pour représenter les deux ensembles de symboles.

Euler conjectura qu'il était impossible de construire de tels carrés pour n = 6 et, plus généralement, pour tout n = 2 ou n = 6 mod 4.

Cette conjecture resta longtemps ouverte. En 1901, Gaston Tarry confirma par une vérification exhaustive qu'aucun carré gréco-latin d'ordre 6 n'existe : le cas des trente-six officiers est bel et bien impossible. En revanche, pour tous les autres ordres n ≠ 2 ou 6, des carrés orthogonaux peuvent être construits.

Ce problème illustre la puissance et la subtilité de la combinatoire algorithmique : une règle simple — éviter les répétitions — conduit à des structures d'une complexité étonnante, où la logique pure rencontre les limites du calcul exhaustif.

Le problème du postier chinois

Comment parcourir toutes les rues d'un réseau en minimisant la distance totale ? Une version "réaliste" des parcours eulériens, avec des applications concrètes (ramassage, voirie).

Le problème du sac à dos (knapsack)

Le problème du sac à dos (knapsack) est un modèle fondamental en algorithmique : on dispose d'objets ayant chacun un poids et une valeur, et l'on cherche à sélectionner ceux qui maximisent la valeur totale sans dépasser une capacité limite. Ce problème, simple à formuler, est **NP-complet**, ce qui signifie qu'aucun algorithme connu ne le résout efficacement dans tous les cas. Il sert de base à de nombreuses applications : allocation de ressources, logistique, cryptographie, planification industrielle, compression de données.

L'idée centrale est toujours la même : choisir intelligemment sous contrainte. Les algorithmes classiques incluent la programmation dynamique, les heuristiques gloutonnes (pour certaines variantes), ou encore les méta-heuristiques comme le recuit simulé ou les algorithmes génétiques.

Les énigmes populaires illustrent parfaitement cette logique. La plus célèbre est celle des chameaux et des bananes : un chamelier doit transporter 3000 bananes à travers un désert, mais son

chameau ne peut porter que 1000 bananes à la fois et en mange une par kilomètre. Comment maximiser la quantité livrée ? Le problème force à optimiser les trajets, les dépôts intermédiaires et les pertes inévitables — exactement comme un sac à dos où chaque choix influence les suivants.

On peut citer aussi les variantes du sac à dos fractionnaire (où l'on peut prendre une partie d'un objet), ou les énigmes de pirates partageant un trésor sous contraintes de poids et de valeur.

Le knapsack est ainsi un modèle universel : derrière une histoire de marchand, de chameau ou de trésor se cache toujours la même question algorithmique : *comment tirer le meilleur parti d'un espace limité ?*

Le stable maximum (ensemble indépendant maximum)

Trouver le plus grand ensemble de sommets d'un graphe sans arêtes entre eux. Un problème simple à formuler, mais très difficile en pratique.

Le coloriage de graphes

Attribuer des couleurs aux sommets sans conflit. Derrière ce jeu se cachent des problèmes fondamentaux en planification et allocation de ressources.

Le mariage stable (algorithme de David Gale–Lloyd Shapley)

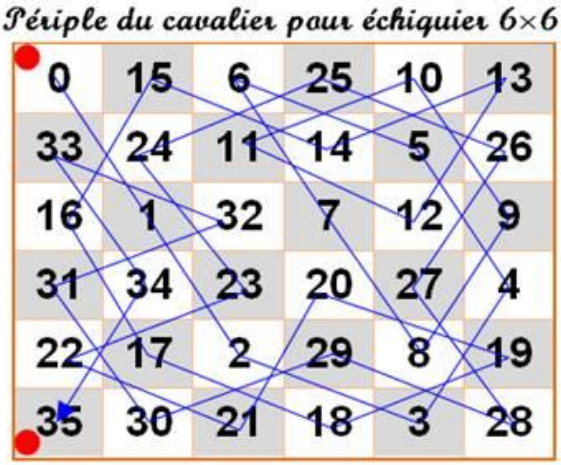
Comment former des paires stables à partir de préférences individuelles ? Une solution élégante avec des applications réelles (affectation d'étudiants, hôpitaux…).

Le problème des huit dames

Placer huit reines sur un échiquier sans qu'elles ne s'attaquent. Un classique révélant la puissance des approches combinatoires et algorithmiques.

Le parcours du cavalier

Déplacer le cavalier et parcourir toutes les cases de l'échiquier.



Les tours de Hanoï

Déplacer une pile de disques selon des règles strictes. Derrière ce jeu se cache une structure récursive parfaite et une croissance exponentielle.

Jeu créé en 1883 par le français Édouard Lucas (1842-1891).Il en fait un petit livre au titre plein de mystère en 1882: La tour de Hanoï

> **Jeu rapporté du Tonkin par le professeur N. Claus (de Siam) du collège Mandarin Li-Sou-Stian**
> **Claus est une anagramme de Lucas et Li-Sou-Tina de Saint Louis.**
> **Il commence son ouvrage par:**
> ***D'après une vieille légende indienne, les brahmes se succèdent depuis bien longtemps sur les marches de l'autel dans le temple de Bénarès pour exécuter le déplacement de la Tour Sacré de BRAHMA aux soixante-quatre étages en or fin, garnis de diamants de Golconde. Quand tout sera fini, la tour et les brahmes tomberont, et ce sera la fin du monde!***

Le jeu original de Lucas est conservé au Conservatoire des Arts et métiers à Paris.

ÉNIGME

Les tours de Hanoï — La récursivité incarnée

Le principe : déplacer une pile de disques d'un piquet à un autre, un par un, sans jamais poser un disque plus grand sur un plus petit.

Pourquoi c'est algorithmique :

La solution optimale repose sur une idée simple et géniale :

Pour déplacer n disques, il faut d'abord déplacer n−1 disques ailleurs.

C'est l'exemple parfait d'un algorithme récursif, où la solution d'un problème dépend de la solution du même problème en plus petit.

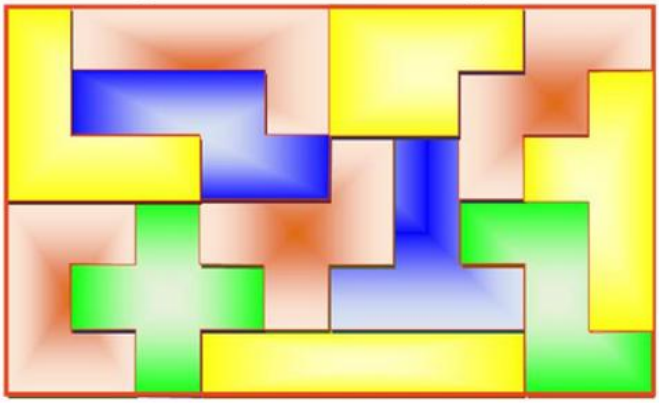
Le saviez-vous ?

Le nombre minimal de coups est $2^n - 1$. Avec 64 disques, cela prendrait… plus de 500 milliards d'années.

Les polyominos

Assembler des formes constituées de carrés adjacents. Ce domaine a inspiré des jeux comme Tetris et pose des questions profondes de pavage.

Rectangle 6×10 avec les douze pentaminos



Le Sudoku généralisé

Déterminer si une grille possède une solution est un problème NP-complet. Un jeu populaire qui cache une complexité théorique importante.

Le Rubik's Cube et le "nombre de Dieu"

Quel est le nombre minimal de mouvements pour résoudre n'importe quelle configuration ? La réponse est 20 — une borne étonnamment faible.

Le réseau de Steiner

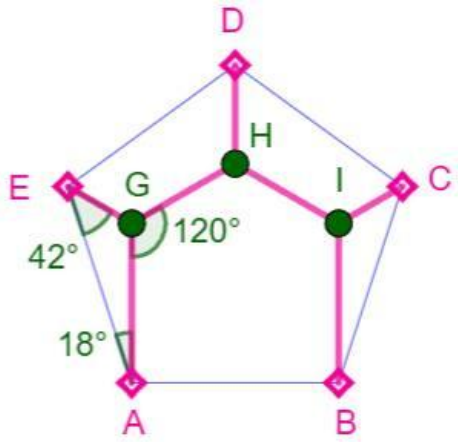
Relier un ensemble de points avec une longueur totale minimale, en autorisant l'ajout de points intermédiaires. Une optimisation géométrique subtile.

Réseau de Steiner pour cinq points (roses).

L'arbre de Steiner minimum pour cinq points disposés aux sommets d'un pentagone régulier est montré sur cette figure. Il implique trois points intermédiaires (verts)

Pour sa construction *(en suivant la figure)*:

- dessiner un pentagone régulier;
- trois verticales à partir de A, B et D;
- l'angle de 42° à partir de E; intersection en G;
- l'angle de 120° en G; intersection en H;
- horizontale en G; intersection en I.
- dessiner l'arbre de Steiner (segments roses)



Le plus court chemin (algorithme de Edsger W. Dijkstra)

Le problème du plus court chemin, formulé par Edsger W. Dijkstra en 1959, est un pilier de l'algorithmique moderne. Il consiste à trouver, dans un graphe pondéré, le chemin de coût minimal entre un point de départ et un point d'arrivée. Chaque arête du graphe représente une route, un lien ou une connexion, et chaque poids correspond à une distance, un temps ou un coût.

L'idée de Dijkstra est à la fois simple et élégante :

1. On commence au nœud source, dont la distance est fixée à zéro.
2. On attribue à tous les autres nœuds une distance initiale infinie.
3. À chaque étape, on choisit le nœud non encore traité ayant la plus petite distance connue, puis on met à jour les distances de ses voisins si un chemin plus court est trouvé.

4. On répète jusqu'à avoir traité tous les nœuds ou atteint la destination.

Ce processus garantit que, lorsque chaque nœud est marqué comme « visité », sa distance est minimale. L'algorithme est efficace : sa complexité est $O(n^2)$ pour une implémentation simple, et $O((n + m)\log n)$ avec une file de priorité où n est le nombre de nœuds et m celui des arêtes.

Dijkstra est omniprésent : il alimente les GPS, les réseaux informatiques, les systèmes de routage, et même les jeux vidéo pour calculer des trajets optimaux. Son principe — explorer progressivement les chemins les plus prometteurs — incarne la logique algorithmique de la recherche efficace : avancer sans tout explorer, mais sans jamais perdre la garantie du meilleur chemin.

Le flot maximal (algorithme de Lester Ford Jr.–Delbert R. Fulkerson)

Maximiser un flux à travers un réseau sous contraintes de capacité. Un pilier de l'optimisation combinatoire.

Pavages, structures et propriétés des graphes

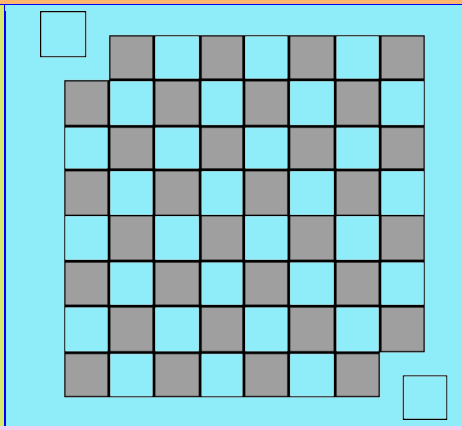
Le problème du domino parfait

Peut-on recouvrir un échiquier privé de deux cases opposées avec des dominos ? Non — une preuve élégante basée sur la parité.

Problème de l'échiquier tronqué

Avec des dominos , peut-on couvrir un échiquier tronqué de deux cases de même couleur ?

Non ! Il y deux approches possibles :
l'expérimentation en examinant tous les cas et en constatant que c'est impossible, ou
la démonstration logique aboutissant à une conclusion indéniable.
En bref: en posant les dominos, j'aurai toujours autant de cases blanches que de noires couvertes. Or sur l'échiquier tronqué, j'ai moins de cases blanches. D'où impossibilité pour obtenir le recouvrement.



Démonstration

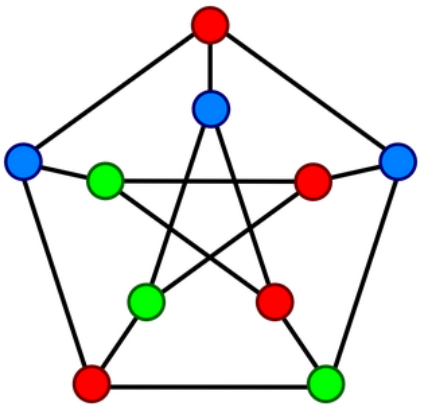
1	*L'échiquier tronqué possède*	**32 cases noires et 30 blanches.**
2	*Un domino couvre toujours*	**1 case noire et 1 blanche.**
3	*30 dominos couvrent*	**30 cases noires et 30 blanches.**
4	*Il reste 1 domino pour*	**2 cases noires.**
5	*Or un domino ne peut pas couvrir*	**2 cases de la même couleur.**

Les polyominos rectifiables

Peut-on assembler une forme en un rectangle parfait ? Certaines formes résistent à toute tentative.

Les graphes planaires

Déterminer si un graphe peut être dessiné sans croisements. Une question centrale reliant topologie et algorithmique.



Les graphes bipartis

Peut-on séparer les sommets en deux groupes sans arêtes internes ? Une structure clé en théorie des graphes et en optimisation.

Le matching maximum

Trouver le plus grand ensemble d'arêtes disjointes. Essentiel pour les problèmes d'affectation.

Le vertex cover

Choisir un ensemble minimal de sommets couvrant toutes les arêtes. Un problème NP-difficile fondamental.

Le set cover

Couvrir un ensemble avec un nombre minimal de sous-ensembles. Au cœur des algorithmes d'approximation.

La clique maximum

Trouver le plus grand sous-graphe complet. L'un des problèmes les plus difficiles en combinatoire.

Les graphes parfaits

Des graphes où le nombre de couleurs nécessaires correspond exactement à la taille du plus grand clique. Une théorie profonde et élégante.

Le graphe de Petersen – Le plus petit rebelle des graphes

Dans le vaste zoo des graphes, certains animaux sont discrets mais redoutablement malins. Le graphe de Petersen, avec ses dix sommets et quinze arêtes, appartient à cette catégorie. À première vue, il ressemble à un simple jouet mathématique : une étoile à cinq branches enchâssée dans un pentagone. Mais ne vous fiez pas à son allure sage. Ce petit graphe est un véritable saboteur de conjectures naïves, un trublion qui adore contredire les intuitions les mieux établies.

Pourquoi un tel prestige pour un graphe si minuscule ? Parce qu'il possède une collection de propriétés contre-intuitives qui en font un champion des contre-exemples. Par exemple, il est **non** hamiltonien (impossible d'y tracer un cycle passant une seule fois par chaque sommet), alors même qu'il est très symétrique et très connecté. Il est aussi non-planaire, mais d'une manière subtile : il ne contient pas les grands coupables habituels (comme K_5 ou $K_{3,3}$) en sous-graphe, ce qui en fait un cas d'école pour comprendre la planéité.

Le graphe de Petersen sert également de repoussoir dans des domaines variés : théorie des couleurs, couplages parfaits, flots, facteurs, et même certaines conjectures sur les graphes cubiques. À chaque fois qu'un mathématicien formule une hypothèse un peu trop optimiste, Petersen surgit, l'air de dire : « Vraiment ? Tu es sûr ? »

Son rôle pédagogique est immense. Il rappelle que l'intuition humaine, même entraînée, peut être trompeuse, et que les structures simples peuvent cacher des comportements étonnamment riches. Pour les algorithmes, c'est un terrain d'entraînement idéal : petit, mais suffisamment retors pour tester la robustesse d'une idée.

En somme, le graphe de Petersen est un peu le chat noir de la théorie des graphes : compact, élégant, mais toujours prêt à renverser vos certitudes d'un coup de patte mathématique.

La théorie de Ramsey

La théorie de Ramsey est l'art délicat de démontrer que, même plongé dans un chaos total, l'univers finit toujours par trahir une forme d'ordre. C'est presque une excuse mathématique pour expliquer pourquoi, malgré nos efforts pour tout laisser au hasard, des motifs finissent par apparaître. On peut mélanger des couleurs, des relations, des connexions à l'infini : passé un certain seuil, une structure ordonnée devient inévitable. C'est comme si les mathématiques nous murmuraient : « Tu peux essayer de semer le désordre, mais je finirai par remettre un peu de rangement là-dedans. »

Cette théorie explore donc les régularités cachées, celles qui émergent malgré nous, comme des coïncidences trop insistantes pour être ignorées. Elle montre que le chaos absolu n'existe pas vraiment : il y a toujours un motif qui guette, prêt à surgir dès que les éléments deviennent trop nombreux. Une belle leçon d'humilité… et de patience.

Le saviez-vous ?

Théorie de RAMSEY

Le principe des tiroirs dit que: si vous avez trois chaussettes à placer dans deux tiroirs, il y aura nécessairement un tiroir comportant au moins deux chaussettes. *(Illustration)*

Plus incroyable, la théorie de Ramsey permet d'affirmer, par exemple, que parmi six personnes, il y aura nécessairement au moins trois personnes qui se connaissent mutuellement ou trois personnes qui ne se connaissent pas du tout.

Ou encore, si on colorie un hexagone et ses diagonales en rouge et en bleu, il est impossible d'éviter un triangle monochrome (rouge ou bleu).

Structures algorithmiques et combinatoires avancées

Le tri par piles (stack sorting)

Combien de piles sont nécessaires pour trier une permutation ? Un problème mêlant structures de données et combinatoire.

Le comptage des inversions

Mesurer le désordre d'une permutation en comptant les paires inversées. Étroitement lié aux algorithmes de tri.

Le tri par échanges adjacents

Combien d'échanges voisins faut-il pour trier ? Une autre mesure fondamentale du désordre.

Les graphes de Cayley

Construire un graphe à partir d'un groupe algébrique. Une source infinie de structures symétriques.

Les hypercubes

Graphes représentant des cubes en dimension n. Utilisés en informatique parallèle et en théorie des codes.

Les graphes de De Bruijn

Représenter toutes les séquences possibles d'une longueur donnée. Crucial en bioinformatique et en génomique.

Les graphes de Johnson

Des graphes très structurés définis à partir de combinaisons. Importants en théorie des codes et optimisation.

Les graphes de Kneser et le théorème de László Lovász

Colorer ces graphes nécessite des outils topologiques profonds. Une rencontre spectaculaire entre combinatoire et topologie.

Les graphes de permutation

Représenter une permutation sous forme de graphe. Applications en tri, en biologie computationnelle et en théorie des ordres.

11.1.3 Géométrie, paradoxes spatiaux et formes étonnantes

(Formes, aires, volumes, illusions et surprises de l'espace)

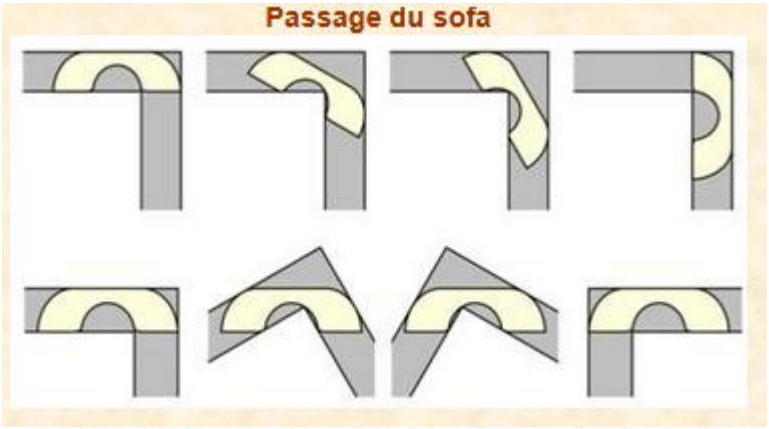
Le problème du sofa mobile – Un défi de géométrie algorithmique

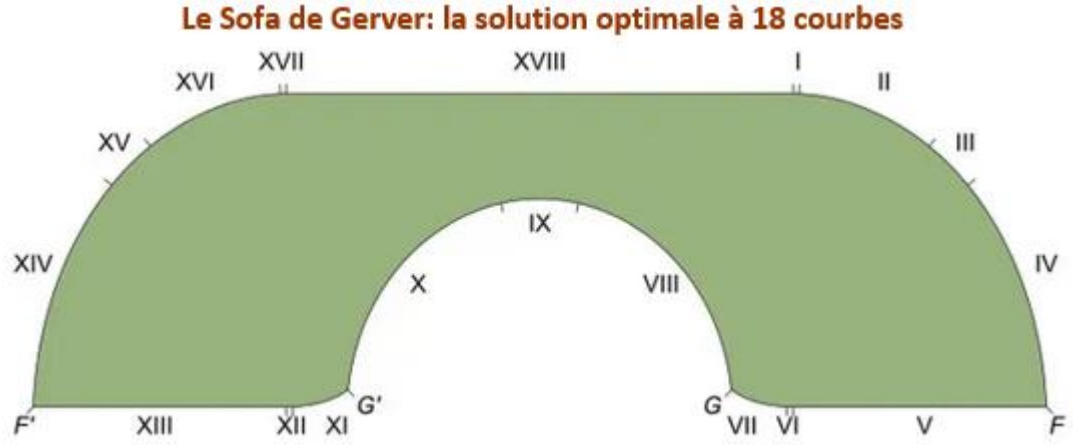
Le problème du sofa mobile (moving sofa problem), posé en 1966 par Leo Moser, est devenu l'un des casse-têtes géométriques les plus célèbres du XX^{e} siècle. Sa formulation est d'une simplicité trompeuse : *quelle est la plus grande aire d'une forme plane rigide pouvant être déplacée dans un couloir en angle droit d'un mètre de large ?* Derrière cette question se cache un problème d'optimisation géométrique d'une subtilité extrême, où interviennent trajectoires, rotations, contraintes de contact et frontières analytiques.

Pendant des décennies, les mathématiciens ont tenté de déterminer la forme optimale. Les premières solutions approchées, comme celle de Hammersley, étaient relativement simples. Mais en 1992, Joseph Gerver propose une forme étonnamment complexe, composée de dix-huit arcs de courbes analytiques, dont l'aire atteint environ 2,2195. Cette figure, asymétrique et contre-intuitive, est longtemps restée la meilleure candidate connue.

Le problème est emblématique des difficultés rencontrées en géométrie algorithmique : il s'agit de maximiser une quantité (l'aire) sous des contraintes de mouvement non linéaires, où chaque position possible impose une inégalité géométrique différente. On peut le voir comme une recherche dans un espace de configurations de très grande dimension, où la forme optimale doit épouser les parois du couloir au cours de sa trajectoire.

En 2024, une avancée majeure est annoncée : le mathématicien Jineon Baek publie une démonstration affirmant que la forme de Gerver est effectivement optimale. Si cette preuve, encore en cours d'examen par les spécialistes, est confirmée, elle clôturera près de soixante ans de spéculations. Le problème du sofa mobile deviendrait alors un exemple fascinant de la manière dont une question simple peut mener à des structures géométriques d'une sophistication inattendue, et illustrer la puissance — mais aussi les limites — des méthodes algorithmiques dans l'exploration des formes optimales.





La chèvre attachée à un piquet

Une chèvre broute attachée à une corde : quelle est l'aire accessible ? Derrière cette situation bucolique se cache une géométrie faite d'arcs de cercle et d'intersections délicates.

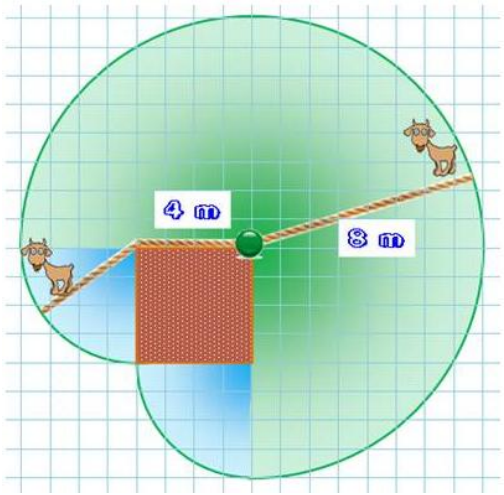
La chèvre et la grange – Cas très simple

La chèvre est en laisse. La corde mesure 8 m. Cette corde est fixée au coin d'une grange carrée de 4 m de côté.

Quelle est l'aire broutée par la chèvre?

La figure montre qu'elle a la liberté de brouter:

- sur les trois quarts d'un grand cercle de 8 m de rayon (zone verte);
- puis, du fait que la corde est entravée, sur deux quarts d'un petit cercle de 4 m de coté (zones bleues).



L'aiguille de Buffon

En lançant une aiguille sur un parquet de lignes parallèles, on peut estimer π. Une rencontre fascinante entre géométrie et probabilité, étudiée dès le XVIII^{e} siècle.

L'**aiguille de Buffon**, proposée en 1777, est l'une des premières expériences liant hasard et géométrie.

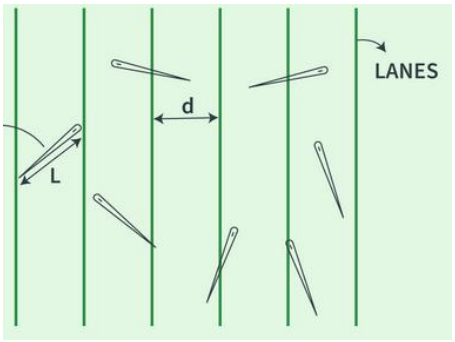
On lance une aiguille de longueur L sur un parquet dont les lignes parallèles sont espacées de d.

La probabilité qu'elle coupe une ligne dépend de son angle et de sa position aléatoire. Buffon montre que cette probabilité vaut:

$$\mathbf{P = \frac{2L}{\pi d}}$$

lorsque (L ≤ d). En répétant l'expérience un grand nombre de fois, on peut donc estimer la valeur de π.

Ce procédé ingénieux préfigure les méthodes de Monte-Carlo et illustre la puissance des approches probabilistes en géométrie.



Le problème de Kakeya

Le problème de Kakeya, formulé au début du XX^{e} siècle, interroge la possibilité de faire pivoter un segment de longueur unitaire dans une région du plan. Intuitivement, on s'attend à ce qu'une telle région possède une aire minimale strictement positive. Pourtant, les travaux de Besicovitch ont révélé un résultat stupéfiant : il existe des ensembles, appelés *ensembles de Kakeya*, dans lesquels un segment peut effectuer une rotation complète tout en ayant une aire aussi petite que l'on veut, voire nulle au sens de la mesure de Lebesgue.

Ces ensembles sont extrêmement fragmentés, proches de structures fractales, et défient l'intuition géométrique classique. Le problème de Kakeya a eu un impact profond en analyse harmonique, en théorie géométrique de la mesure et même en théorie des nombres. Il illustre comment une question simple peut conduire à des constructions d'une complexité insoupçonnée, et comment la géométrie peut se heurter aux limites de notre intuition lorsqu'elle rencontre l'infini et la fragmentation extrême des ensembles pathologiques.

Solution : une deltoïde

La deltoïde est la trajectoire d'un point d'un cercle lorsque celui-ci roule sans glisser à l'intérieur d'un cercle trois fois plus grand.

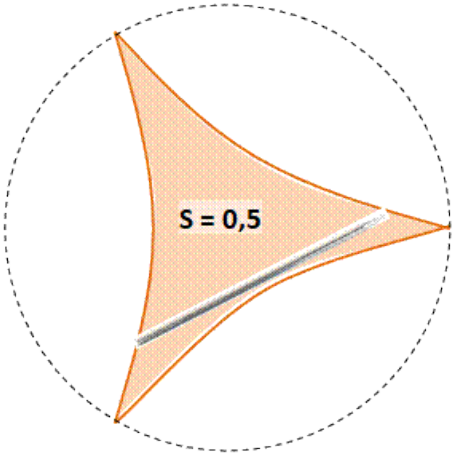
L'aiguille tourne avec ses deux extrémités parcourant deux arcs tandis qu'elle reste tangente au troisième arc.

Le problème a été posé en 1917 par Soichi Kakeya. Il trouva rapidement la solution deltoïde.

Conjecturant que c'était la meilleure solution, les mathématiciens essayèrent de prouver qu'il s'agissait de la surface minimale, sans vraiment chercher mieux. Puis …

Zeev Dvir, mathématicien et informaticien à l'Université de Princeton, a prouvé la conjecture de Kakeya pour certains systèmes de nombres finis avec son étudiant, Manik Dhar.

On l'a longtemps cru qu'il s'agissait de la meilleure solution "non- convexe" .



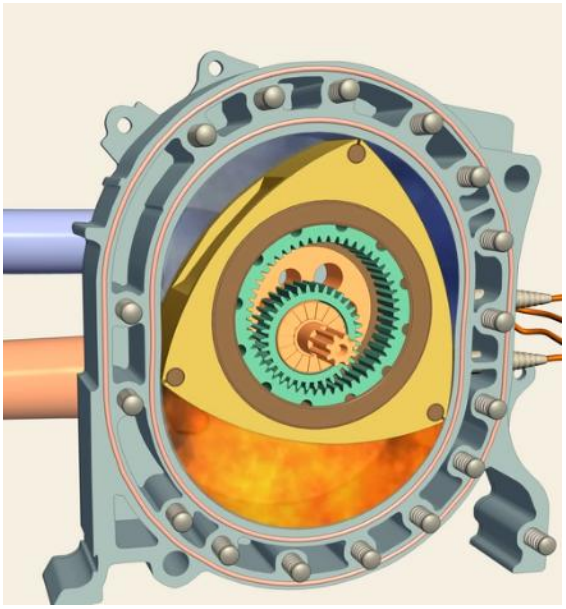
La conjecture de Kakeya est démontrée pour les nombres discrets dans divers systèmes.

Elle reste ouverte pour les nombres réels.

Certains mathématiciens doutent qu'elle soit vérifiée dans ce cas-là.

Le paradoxe de Stefan Banach–Alfred Tarski

Une sphère peut être découpée en morceaux et reconstituée en deux sphères identiques. Un résultat déroutant, lié à l'infini et à l'axiome du choix.



La quadrature du cercle

Construire un carré de même aire qu'un cercle avec règle et compas : un rêve antique, finalement prouvé impossible au XIX^{e} siècle grâce à la transcendance de π.

La duplication du cube

Doubler le volume d'un cube à l'aide de constructions classiques est impossible. Un autre problème antique révélant les limites des outils géométriques.

La trisection de l'angle

Diviser un angle en trois parties égales avec règle et compas : possible dans certains cas, mais impossible en général.

Le triangle de Reuleaux *(Illustraton)*

Une forme de largeur constante qui roule comme un cercle sans en être un. Utilisée dans des mécanismes ingénieux, notamment pour percer des trous quasi carrés.

Les surfaces minimales (bulles de savon)

Les films de savon adoptent spontanément la surface minimale. Une manifestation physique élégante d'un principe mathématique profond.

Le problème de Hadwiger–Nelson

Combien de couleurs faut-il pour colorier le plan sans que deux points à distance 1 aient la même couleur ? Une question simple encore ouverte.

Le problème de la pizza

Comment découper équitablement une pizza sans accéder à son centre ? Plusieurs méthodes ingénieuses montrent que l'équité ne dépend pas toujours de la symétrie.

Le partage équitable (fair division)

Diviser un objet entre plusieurs personnes aux préférences différentes sans jalousie. Un domaine à la frontière entre mathématiques, économie et justice.

Formes optimales et pavages

Le problème isopérimétrique

Quelle forme maximise l'aire pour un périmètre donné ? Le cercle est optimal — mais les variantes de cette question ouvrent des perspectives riches.

Le cercle “invisible”

Certaines configurations géométriques rendent des objets invisibles sous certains angles. Une exploration des limites de la perception géométrique.

Le théorème de Georg Pick

Une formule simple relie l’aire d’un polygone aux points entiers qu’il contient. Une élégance rare en géométrie discrète.

Théorème de Pick

L'aire du polygone dont les sommets sont sur un quadrillage est fonction de ces deux quantités de points:

P = quantité de points sur la frontière (le périmètre)

Q = quantité de points à l'intérieur de la frontière.

$$A = \frac{P}{2} + Q - 1$$

Pentagone quelconque (exemple)

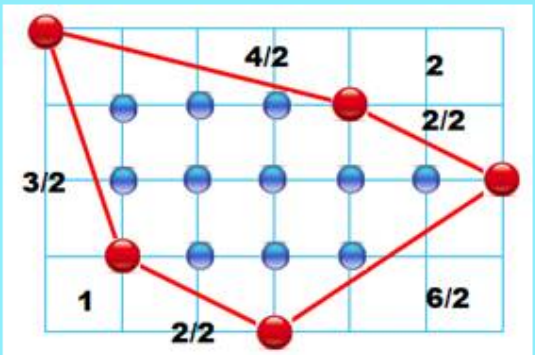
Le calcul de l'aire de ce pentagone avec une méthode conventionnelles donne 12,5 cm²

Avec la formule de Pick:

1. Compter les points rouges P = 5;
2. Compter les points bleus Q = 11;
3. Appliquer la formule;

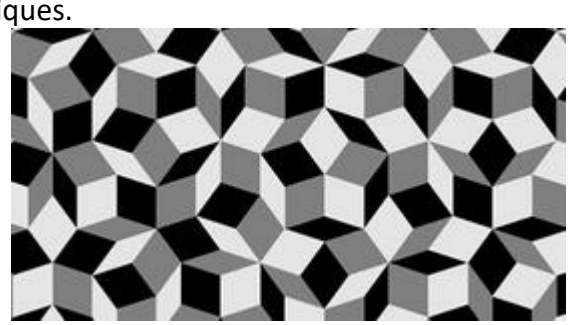
A = P/2 + Q – 1

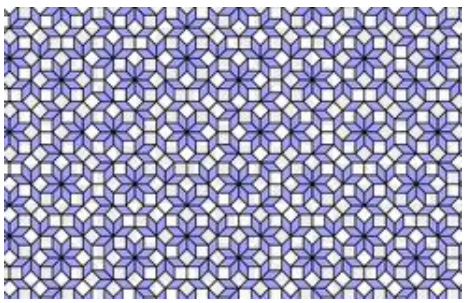
A = 5/2 + 11 – 1 = 12,5 cm²



Les pavages de Roger Penrose

Les pavages de Roger Penrose ont profondément transformé notre compréhension de l’ordre et de la symétrie. Découverts dans les années 1970, ils montrent qu’il est possible de recouvrir le plan à l’aide de seulement deux formes, tout en produisant un motif non périodique, c’est-à-dire sans aucune répétition régulière. Cette propriété, longtemps jugée impossible, révèle qu’un ensemble de règles locales peut imposer un ordre global d’une richesse remarquable. Les pavages de Penrose possèdent notamment une symétrie d’ordre cinq, interdite dans les pavages périodiques classiques.





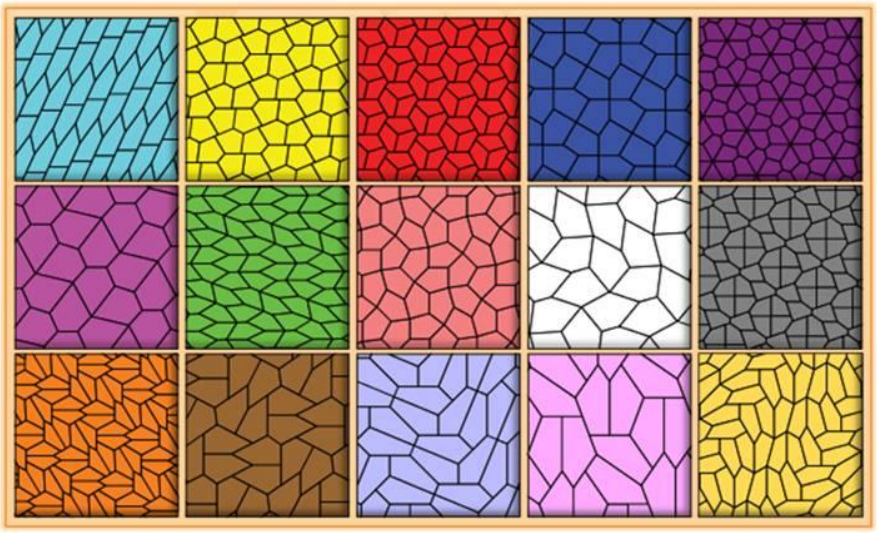
Cette découverte a trouvé un écho inattendu en physique. En 1984, la mise en évidence des quasi-cristaux a montré que certains matériaux adoptent une organisation atomique quasi périodique, très proche des pavages de Penrose. Les mathématiques du pavage sont ainsi devenues un outil pour comprendre la structure intime de la matière.

Un résultat récent, obtenu à la mi-2024, enrichit encore ce paysage. Les pavages apériodiques d’Ammann-Beenker, proches cousins des pavages de Penrose, ont été démontrés capables d’admettre des cycles hamiltoniens. Cette propriété, absente chez Penrose, permet de relier de manière optimale les atomes dans certains modèles de quasi-cristaux. Elle ouvre de nouvelles

perspectives sur la façon dont l'ordre apériodique peut influencer les propriétés physiques des matériaux, et illustre la fécondité du dialogue entre géométrie et physique contemporaine.

Le pavage du plan par des pentagones

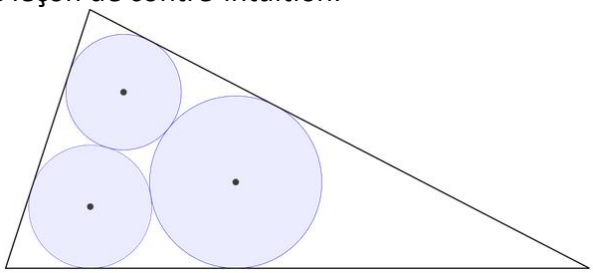
Le pavage du plan par des pentagones est un problème classique de géométrie combinatoire : parmi tous les pentagones convexes possibles, lesquels peuvent recouvrir le plan sans chevauchement ni trou ? La question, posée dès le XIX^{e} siècle, a résisté pendant plus d'un siècle. Au fil des décennies, de nouvelles familles de pentagones pavants ont été découvertes, souvent grâce à des arguments subtils mêlant géométrie, symétrie et contraintes angulaires. Ce n'est qu'en 2015 que la classification complète a été établie : quinze types de pentagones convexes pavent le plan, et aucun autre. Une conclusion tardive pour un problème en apparence élémentaire.



Les quinze classes de pavage pentagonal

Le problème de Malfatti

Placer trois cercles dans un triangle pour maximiser leur aire totale. La solution intuitive n'est pas optimale — une belle leçon de contre-intuition.



Le cercle osculateur

Le cercle qui "épouse" localement une courbe au mieux. Il capture la notion de courbure en un point.

Points remarquables et géométrie classique

Le problème de Vincenzo Viviani

Dans un tétraèdre régulier, la somme des distances à trois faces est constante. Une propriété élégante et peu intuitive.

Le point de Pierre de Fermat

Trouver un point minimisant la somme des distances à trois sommets. Une solution géométrique étonnamment simple.

Le point de Evangelista Torricelli

Une reformulation du problème de Fermat où les angles formés sont de 120°, révélant une structure optimale.

Optimisation géométrique et algorithmique

Le cercle inscrit maximal

Trouver le plus grand cercle contenu dans une région donnée. Un problème central en géométrie computationnelle.

Le cercle englobant minimal

Le plus petit cercle contenant un ensemble de points. Utilisé en clustering et en analyse spatiale.

Le rectangle englobant minimal

Trouver le rectangle de surface minimale contenant un ensemble de points. Un classique en géométrie algorithmique.

Le polygone convexe maximal

Extraire la plus grande structure convexe d'un nuage de points. Étroitement lié à l'enveloppe convexe.

Les polygones étoilés

Déterminer si un polygone possède un point depuis lequel tout est visible. Une notion subtile de visibilité.

Le recouvrement par disques

Le recouvrement par disques est un problème fondamental d'optimisation : combien de disques de rayon donné faut-il pour couvrir un ensemble du plan ?

Cette question, simple en apparence, intervient dans de nombreux domaines concrets, comme la conception de réseaux de capteurs, la couverture Wi-Fi ou la surveillance robotique.

Mathématiquement, il s'agit de minimiser le nombre de disques tout en garantissant que chaque point de la zone visée appartient à au moins l'un d'eux.

Selon la forme de l'ensemble, le problème peut devenir très complexe et même NP-difficile, illustrant la richesse des liens entre géométrie, algorithmique et applications pratiques.

Le problème d'Apollonius of Perga

Construire un cercle tangent à trois objets donnés (points, droites, cercles). Un défi classique aux multiples solutions.

Le pavage hexagonal optimal

Les abeilles utilisent des hexagones pour maximiser l'efficacité. Une solution optimale démontrée mathématiquement.

Le polyèdre inscrit de volume maximal

Quel solide maximise le volume à l'intérieur d'une sphère ? Une question délicate reliant symétrie et optimisation.

Les géodésiques sur un polyèdre

Quel est le plus court chemin sur une surface anguleuse ? Contrairement au plan, les trajectoires peuvent être surprenantes.

11.1.4 Logique, paradoxes et raisonnements déroutants

(Auto-référence, probabilités, décisions et limites de la raison)

Le paradoxe du menteur

Le paradoxe du menteur est l'un des plus anciens et des plus célèbres paradoxes logiques. Il remonte à l'Antiquité, où l'on attribue à Épiménide la déclaration devenue emblématique : « Tous les Crétois sont des menteurs. » Sous sa forme moderne, il se condense en une phrase encore plus percutante : **« Je mens toujours. »** Si cette phrase est vraie, alors la personne dit la vérité, donc elle ne ment pas toujours, ce qui contredit l'énoncé. Mais si la phrase est fausse, alors elle ne ment pas toujours, ce qui implique qu'elle dit parfois la vérité — et donc que la phrase pourrait être vraie. Dans les deux cas, l'énoncé se retourne contre lui-même.

Ce paradoxe met en lumière un phénomène fondamental : l'auto-référence peut déstabiliser les systèmes logiques les plus rigoureux. Il ne s'agit pas seulement d'un jeu linguistique, mais d'un problème structurel. Dès qu'un énoncé parle de sa propre vérité, il devient possible de construire des boucles logiques qui échappent à toute évaluation cohérente.

Au XX^{e} siècle, ce paradoxe a joué un rôle crucial dans les travaux de Gödel. En construisant des énoncés arithmétiques capables de « dire » qu'ils sont indémontrables, Gödel a montré que tout système formel suffisamment puissant contient des propositions vraies mais impossibles à démontrer en son sein. Le paradoxe du menteur devient alors une métaphore — ou plutôt une préfiguration — des limites intrinsèques des systèmes logiques.

Aujourd'hui encore, ce paradoxe irrigue la logique, la théorie des langages, l'informatique théorique et même la philosophie de l'esprit. Il rappelle que la vérité n'est pas toujours un concept simple, et que la capacité d'un système à parler de lui-même ouvre la porte à des phénomènes profondément déstabilisants.

Le paradoxe de Bertrand Russell

Le paradoxe de Bertrand Russell, formulé en 1901, est l'un des événements les plus marquants de l'histoire des mathématiques modernes. Il surgit au cœur d'un projet ambitieux : fonder l'ensemble des mathématiques sur la théorie naïve des ensembles, où tout ensemble est défini par une propriété. Russell remarque alors que cette liberté totale conduit à une contradiction spectaculaire. Considérons l'ensemble R de tous les ensembles qui **ne se contiennent pas eux-mêmes**. La question devient inévitable : *R se contient-il lui-même ?*

Si R se contient lui-même, alors par définition il ne devrait pas se contenir. Mais s'il ne se contient pas, alors il satisfait la propriété définissant R, et devrait donc s'y trouver. Dans les deux cas, on obtient une contradiction. Ce paradoxe révèle que la théorie naïve des ensembles, pourtant intuitive, est incohérente dès qu'elle autorise des définitions auto-référentielles trop larges.

L'impact fut immense. Russell et Whitehead tentèrent de sauver les fondements des mathématiques en élaborant la *théorie des types*, qui interdit précisément ce genre d'auto-référence. Parallèlement, Zermelo puis Fraenkel développèrent une théorie axiomatique des ensembles plus restrictive, aujourd'hui connue sous le nom de ZF ou ZFC, qui évite soigneusement les constructions paradoxales.

Le paradoxe de Russell n'est pas seulement une curiosité logique : il marque la fin de l'illusion selon laquelle les mathématiques reposeraient sur une base parfaitement transparente et intuitive. Il a ouvert la voie à une réflexion profonde sur les limites de la formalisation, influençant les travaux

de Gödel, Tarski et bien d'autres. En un sens, il rappelle que même les concepts les plus fondamentaux — comme celui d'ensemble — doivent être maniés avec une rigueur extrême pour éviter les pièges de l'auto-référence.

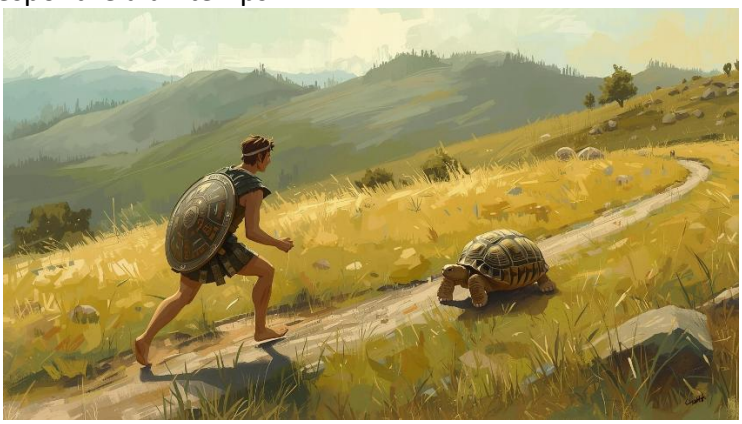
Le paradoxe de Berry

Le paradoxe de Berry naît d'une définition auto-référentielle : « le plus petit entier non définissable en moins de vingt mots ». Cette phrase, pourtant courte, définit précisément un entier censé ne pas pouvoir l'être. On obtient ainsi une contradiction directe. Ce paradoxe révèle les limites du langage naturel et montre comment la définition peut devenir source d'incohérence lorsqu'elle se retourne sur elle-même.

Les paradoxes de Zeno of Elea

Ces paradoxes, formulés au Ve siècle av. J.-C., sont parmi les premières explorations philosophiques de l'infini et du mouvement. Le plus célèbre met en scène Achille poursuivant une tortue. Pour la rattraper, il doit d'abord atteindre le point où elle se trouvait. Mais entre-temps, la tortue a avancé un peu. Achille doit alors atteindre ce nouveau point, puis le suivant, et ainsi de suite. Zénon en conclut qu'Achille ne rattrape jamais la tortue, car il doit parcourir une infinité d'étapes.

Ces paradoxes ne visent pas à nier le mouvement, mais à montrer que notre intuition de l'espace et du temps se heurte à des difficultés conceptuelles dès qu'intervient l'infini. Ils ont profondément influencé la réflexion mathématique : il faudra attendre le développement du calcul infinitésimal et la notion moderne de somme de séries convergentes pour comprendre comment une infinité d'étapes peut correspondre à un temps fini.



Le paradoxe des anniversaires

Ce paradoxe illustre à merveille la manière dont notre intuition échoue face aux probabilités. On pourrait croire qu'il faut une grande foule pour que deux personnes partagent le même anniversaire. Pourtant, avec seulement 23 individus, la probabilité qu'au moins deux aient la même date de naissance dépasse déjà 50 %. Ce résultat, contre-intuitif, provient du fait que le nombre de paires possibles croît très vite : dans un groupe de 23 personnes, il existe 253 comparaisons de dates. La probabilité que toutes soient différentes chute donc rapidement.

Ce paradoxe est souvent utilisé pour introduire la loi des grands nombres et les collisions en informatique. Dans les algorithmes de hachage, par exemple, deux clés distinctes peuvent produire la même valeur — un phénomène analogue à deux anniversaires identiques. Les concepteurs de systèmes cryptographiques doivent en tenir compte pour éviter les collisions trop fréquentes.

D'autres paradoxes statistiques révèlent la même tension entre intuition et calcul. Le paradoxe de Monty Hall, où changer de porte triple vos chances de gagner, défie la logique spontanée. Le

paradoxe de Simpson montre qu'une tendance observée dans plusieurs groupes peut s'inverser lorsqu'on les combine. Et le paradoxe des jumeaux en relativité, bien que physique, repose aussi sur une intuition trompeuse du temps.

Ces exemples rappellent que la pensée algorithmique et probabiliste exige de dépasser nos réflexes cognitifs. Le paradoxe des anniversaires, en particulier, enseigne que la probabilité n'est pas une affaire de « bon sens », mais de combinatoire. Il révèle la puissance des mathématiques pour corriger nos intuitions et comprendre les phénomènes où le hasard, la complexité et la logique se rencontrent.

Le problème de Monty Hall (ou problème des trois portes)

Le problème de Monty Hall est un classique des probabilités, particulièrement intéressant dans un ouvrage consacré aux algorithmes, car il illustre comment une mise à jour d'information peut modifier une décision optimale. Inspiré du jeu télévisé Let's Make a Deal, il se formule ainsi : trois portes sont proposées, derrière l'une se trouve une voiture, derrière les deux autres des chèvres. Le joueur choisit une porte. L'animateur, qui connaît la solution, ouvre ensuite une autre porte, révélant une chèvre, puis propose au joueur de changer son choix.

L'intuition suggère que les deux portes restantes ont chacune une probabilité de 1/2. Pourtant, une analyse rigoureuse montre que changer de porte donne une probabilité de gain de 2/3, contre 1/3 si l'on reste sur son choix initial. Ce résultat découle du fait que l'action de l'animateur n'est pas aléatoire : il révèle toujours une chèvre et ne peut jamais dévoiler la voiture. Son comportement agit donc comme un filtre d'information, redistribuant les probabilités.

D'un point de vue algorithmique, ce problème est un exemple simple de mise à jour bayésienne implicite : une hypothèse initiale (la position de la voiture) est réévaluée après l'observation d'un événement dépendant de cette hypothèse. Il met aussi en évidence les dangers d'un raisonnement naïf face à des systèmes où les règles de décision influencent les données observées.

Le paradoxe de Monty Hall possède plusieurs « cousins » conceptuels. On peut citer le **paradoxe des trois prisonniers**, qui repose sur une structure d'information similaire, ainsi que le problème des « **deux enfants** » (ou « au moins une fille »), où la manière d'obtenir une information modifie les probabilités. Plus généralement, ces situations relèvent des probabilités conditionnelles biaisées par un mécanisme de sélection, un thème central en algorithmique probabiliste et en apprentissage.

Le paradoxe des deux enveloppes

Le paradoxe des deux enveloppes peut être considéré comme un « cousin » du problème de Monty Hall, même si le mécanisme en jeu est différent.

Le paradoxe des deux enveloppes met en scène deux enveloppes contenant des sommes d'argent, l'une ayant le double de l'autre. Après en avoir choisi une, on se demande s'il faut changer. Un raisonnement naïf suggère que changer est toujours avantageux, ce qui conduit à une contradiction absurde.

Comme dans problème de Monty Hall, l'erreur vient d'une mauvaise modélisation de l'information. Ici, c'est l'ambiguïté sur la distribution des valeurs qui fausse le calcul des espérances et piège l'intuition.

Le paradoxe des trois prisonniers

Le paradoxe des trois prisonniers est un problème classique de probabilités conditionnelles, souvent rapproché du célèbre problème de Monty Hall. Il met en lumière une idée essentielle mais contre-intuitive : la manière dont une information est obtenue influence les probabilités.

Trois prisonniers, A, B et C, savent que l'un d'eux sera gracié, tandis que les deux autres seront exécutés. Chacun a donc, au départ, une probabilité de **1/3** d'être gracié.

Le prisonnier A, cherchant à obtenir un avantage, demande au gardien de lui révéler le nom de l'un des deux autres prisonniers qui sera exécuté. Le gardien accepte, mais avec une contrainte importante : il doit toujours donner le nom d'un condamné, et il choisit parmi les possibilités disponibles sans jamais révéler directement le sort de A.

Supposons que le gardien dise : « B sera exécuté ».

À ce stade, A pourrait penser que la situation a changé : il ne reste plus que deux personnes possibles (A et C), donc sa probabilité serait passée de 1/3 à 1/2. Mais ce raisonnement est incorrect.

En réalité, la probabilité que A soit gracié reste **1/3**, tandis que celle que C soit gracié devient **2/3**.

Pourquoi ? Parce que l'information donnée par le gardien n'est pas neutre. Elle dépend du choix initial du prisonnier gracié :

- Si A est gracié (probabilité 1/3), le gardien peut dire soit B soit C (il a le choix).
- Si B est gracié (probabilité 1/3), le gardien est obligé de dire C.
- Si C est gracié (probabilité 1/3), le gardien est obligé de dire B.

Ainsi, lorsque le gardien dit « B sera exécuté », cette réponse est plus probable dans le cas où C est gracié que dans le cas où A est gracié. L'information reçue favorise donc implicitement l'hypothèse « C est gracié ».

Ce paradoxe montre que **toutes les informations ne se valent pas** : il faut toujours se demander *comment* elles ont été produites.

Un paradoxe apparent en génétique : « au moins une fille »

Un autre exemple classique, souvent rapproché du précédent, concerne une famille avec deux enfants.

On compare deux situations :

1. On sait que la famille a **au moins une fille**.
2. On choisit un enfant au hasard et on apprend que c'est une fille.

Ces deux formulations semblent proches, mais elles conduisent à des probabilités différentes.

Cas 1 : « au moins une fille »

Les combinaisons possibles pour deux enfants (en supposant garçon/fille équiprobables) sont :

garçon–garçon / garçon–fille / fille–garçon / fille–fille

Si l'on sait qu'il y a au moins une fille, on élimine seulement le cas garçon–garçon. Il reste donc :

garçon–fille / fille–garçon / fille–fille

Parmi ces trois cas équiprobables, un seul contient deux filles. La probabilité que les deux enfants soient des filles est donc **1/3**.

Cas 2 : « un enfant tiré au hasard est une fille »

Ici, la situation est différente. On ne conditionne pas simplement sur une propriété globale (« au moins une fille »), mais sur un processus de sélection.

On imagine que l'on choisit un enfant au hasard dans la famille, et que l'on observe qu'il s'agit d'une fille. Cette méthode donne plus de poids aux familles avec deux filles, car elles ont deux chances d'être sélectionnées.

Dans ce cas, la probabilité que l'autre enfant soit aussi une fille devient **1/2**.

Ce que ces paradoxes nous apprennent

Ces deux situations illustrent une idée fondamentale en probabilités :

Une probabilité dépend autant de l'information que de la manière dont cette information est obtenue.

Dans le paradoxe des prisonniers, comme dans celui des enfants, l'erreur intuitive consiste à traiter une information comme si elle était neutre, alors qu'elle est en réalité biaisée par un mécanisme de sélection.

Autrement dit, les probabilités ne décrivent pas seulement le monde tel qu'il est, mais aussi la façon dont on y accède.

C'est précisément cette dépendance au contexte qui rend ces paradoxes à la fois déroutants et extrêmement formateurs : ils obligent à dépasser une vision naïve du hasard pour entrer dans une compréhension plus fine des probabilités conditionnelles.

Les énigmes des chapeaux

Des joueurs doivent deviner la couleur de leur chapeau en observant les autres. Ces puzzles révèlent des stratégies collectives étonnantes.

Énigme

Trois personnes ***A, B et C*** ont les yeux bandés.

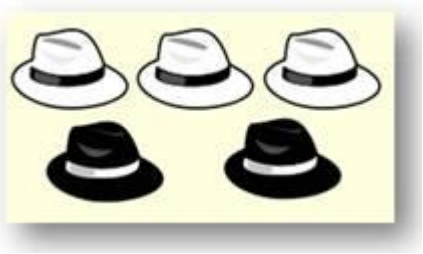
Parmi 3 chapeaux blancs et 2 chapeaux noirs, chacun en reçoit un.

On enlève les bandeaux, mais étant l'un derrière l'autre, C voit A et B, B voit A et A ne voit rien.

Ils gagnent si au bout d'une minute l'un d'eux sait quel chapeau il porte.

À la 59^{e} seconde, A dit, je sais.

Quelle est la couleur du chapeau? Pourquoi?



Solution

C voit deux chapeaux et ne dit rien; ils ne sont pas tous deux noirs sinon il dirait qu'il a un blanc. Parmi les deux chapeaux qu'il voit, l'un est blanc: le A ou le B

B, constatant que C ne parle pas, fait la même déduction: j'ai un chapeau blanc ou c'est A qu'il l'a. S'il voit A en noir, il sait qu'il a le blanc. Or, il ne dit rien. C'est qu'il ne voit pas A en noir, mais en blanc.

A porte un chapeau blanc.

Le paradoxe de Newcomb

Faut-il faire confiance à une prédiction parfaite ou agir librement ? Un dilemme entre rationalité et libre arbitre.

Le paradoxe de Braess

Ajouter une route à un réseau peut empirer la circulation. Un phénomène réel qui défie l'intuition.

Le paradoxe de Marquis de Condorcet

Des préférences individuelles cohérentes peuvent produire une décision collective incohérente. Un problème fondamental en théorie du vote.

Le paradoxe de l'ascenseur

On observe plus souvent certains trajets que d'autres — non par hasard, mais à cause des flux et des probabilités conditionnelles.

Auto-référence et paradoxes logiques profonds

Le paradoxe du barbier

Ce paradoxe est l'une des reformulations les plus célèbres du paradoxe de Russell, destinée à le rendre plus accessible sans en trahir la profondeur. L'énoncé est simple : dans un village, un barbier rase tous les hommes qui ne se rasent pas eux-mêmes, et seulement ceux-là. La question devient alors inévitable : *le barbier se rase-t-il lui-même ?*

Si le barbier se rase lui-même, alors il ne devrait pas appartenir à la catégorie de ceux qui ne se rasent pas eux-mêmes, et donc il ne devrait pas se raser. Mais s'il ne se rase pas lui-même, alors il fait partie de ceux que le barbier doit raser — donc il devrait se raser. Dans les deux cas, on obtient une contradiction. Le paradoxe ne provient pas d'un raisonnement erroné, mais d'une définition impossible, qui s'auto-réfère de manière incohérente.

Ce paradoxe est un miroir fidèle du paradoxe de Russell *(vu plus haut)* : l'ensemble de tous les ensembles qui ne se contiennent pas eux-mêmes se contient-il lui-même ? La version du barbier remplace les ensembles par des personnes et l'appartenance par une action, mais la structure logique reste identique. Dans les deux cas, le problème surgit lorsque l'on autorise une définition trop large, qui inclut potentiellement l'objet défini dans son propre domaine d'application.

Le paradoxe du barbier possède plusieurs cousins conceptuels. Le paradoxe du menteur (« Je mens toujours ») repose sur une auto-référence linguistique qui rend l'énoncé indécidable. Le paradoxe de Berry (« le plus petit entier non définissable en moins de vingt mots ») montre comment une définition peut se retourner contre elle-même. Le paradoxe de Grelling-Nelson (« hétérologique » est-il hétérologique ?) explore les pièges des prédicats qui s'appliquent à eux-mêmes. Tous ces paradoxes partagent une même racine : la difficulté de manipuler des systèmes où les objets peuvent se décrire eux-mêmes.

L'impact du paradoxe du barbier en logique algorithmique est considérable. Il montre qu'un système de règles naïvement formulé peut conduire à des contradictions internes. En algorithmique, cela se traduit par la nécessité d'éviter les définitions circulaires non contrôlées, les fonctions qui s'appellent elles-mêmes sans condition d'arrêt, ou les structures de données autorisant des auto-références incohérentes. Le paradoxe préfigure également les résultats de Gödel : dans tout système suffisamment expressif, il existe des énoncés qui parlent de leur propre démontrabilité, ouvrant la voie à des limites fondamentales.

Dans les langages de programmation, on retrouve cette problématique dans la gestion des types récursifs, des pointeurs, ou des systèmes de modules. Les concepteurs doivent imposer des règles strictes pour éviter les définitions paradoxales. En théorie des ensembles, la réponse fut l'axiomatisation ZF, qui interdit explicitement les ensembles « trop grands » ou auto-référentiels.

Ainsi, le paradoxe du barbier n'est pas une simple curiosité logique. C'est une fenêtre sur les limites de la définition, un avertissement sur les dangers de l'auto-référence, et un jalon essentiel dans la construction des fondements modernes de la logique et de l'algorithmique.

Le paradoxe de Curry

Une phrase auto-référente qui permet, en apparence, de démontrer n'importe quelle proposition.

Le paradoxe de Curry pousse la logique du "barbier" encore plus loin. Il repose sur une phrase du type : « Si cette phrase est vraie, alors 2 = 1. »

Étonnamment, dans certains systèmes logiques permissifs, cette structure suffit à « prouver » n'importe quelle proposition. Le paradoxe révèle que certaines règles d'implication doivent être soigneusement encadrées pour éviter l'effondrement total du système

Le paradoxe de Yablo

Le paradoxe de Yablo propose une infinité de phrases, chacune disant que toutes les suivantes sont fausses. Aucune phrase ne se réfère directement à elle-même, et pourtant l'ensemble est paradoxal.

Yablo montre que l'auto-référence n'est pas nécessaire pour créer une contradiction : une chaîne infinie de dépendances suffit. Ce résultat a eu un impact profond en logique des modèles et en théorie de la vérité.

Le paradoxe de Moore

Le paradoxe de Moore explore la tension entre vérité et croyance : « Il pleut, mais je ne crois pas qu'il pleuve. » La phrase peut être vraie, mais elle ne peut pas être affirmée de manière cohérente.

Ce paradoxe met en lumière les limites des systèmes où vérité factuelle et états mentaux interagissent, un thème essentiel en intelligence artificielle et en logique épistémique.

Le paradoxe de Fitch

Le paradoxe de Fitch montre que si toutes les vérités sont connaissables, alors toutes les vérités sont connues — une conclusion surprenante.

Ce résultat, central en logique philosophique, questionne la notion même de connaissance potentielle et a des implications pour les systèmes où l'on modélise des agents capables d'apprendre.

Le paradoxe de Good

Le paradoxe de Good concerne les prédictions auto-réalisatrices : une affirmation peut devenir vraie simplement parce qu'elle a été annoncée.

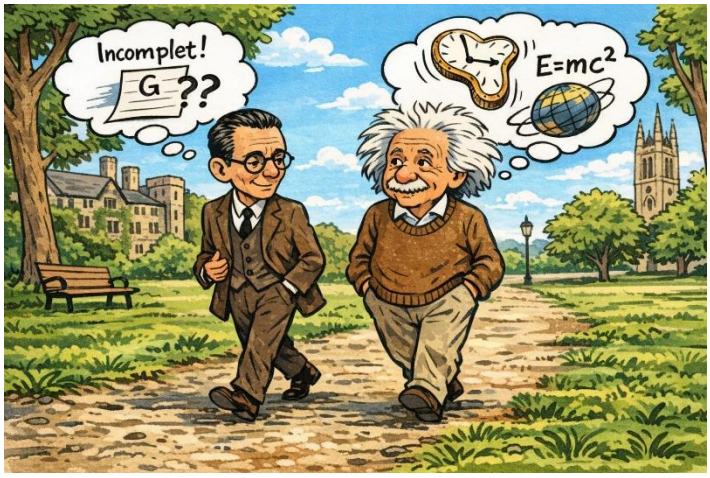
Ce phénomène apparaît dans les systèmes logiques, mais aussi en économie, en sociologie ou dans les algorithmes d'apprentissage où les modèles influencent les données qu'ils prédisent

Le paradoxe de Skolem

Le paradoxe de Skolem révèle qu'un univers mathématique infini peut posséder une description « dénombrable » dans un modèle.

Cette étrangeté montre que la taille d'un ensemble dépend du point de vue logique adopté, un résultat fondamental en théorie des modèles.

Le théorème de Kurt Gödel (incomplétude)



Le théorème d'incomplétude de Kurt Gödel, publié en 1931, est l'un des résultats les plus profonds et les plus déstabilisants de toute l'histoire des mathématiques.

Il affirme qu'**aucun système logique suffisamment puissant pour contenir l'arithmétique ne peut être à la fois complet et cohérent**. Autrement dit, si un système est cohérent — c'est-à-dire qu'il n'y produit pas de contradictions — alors il existe nécessairement des énoncés vrais mais impossibles à démontrer à l'intérieur de ce système. Et si l'on tente d'ajouter ces énoncés comme axiomes, de nouveaux énoncés indécidables apparaissent. La quête d'un fondement ultime, parfaitement clos et autosuffisant, devient alors impossible.

L'idée centrale du théorème repose sur une construction ingénieuse : Gödel encode des énoncés mathématiques sous forme de nombres, puis fabrique une phrase qui, en substance, affirme « je

suis indémontrable ». Si le système pouvait démontrer cette phrase, il serait incohérent ; s'il ne peut pas la démontrer, alors la phrase est vraie mais indémontrable. Cette auto-référence contrôlée, inspirée du paradoxe du menteur, montre que la vérité dépasse nécessairement la démontrabilité.

L'impact de ce résultat est immense. Il met fin au programme de Hilbert, qui visait à formaliser toutes les mathématiques dans un système complet et cohérent. Il révèle que les mathématiques ne peuvent jamais être entièrement capturées par un ensemble fini de règles mécaniques. Il ouvre aussi la voie à une nouvelle compréhension de la logique, où la notion de vérité ne se réduit pas à la preuve.

En algorithmique, le théorème de Gödel annonce des limites analogues. Il préfigure le théorème de Church et Turing sur l'indécidabilité : certains problèmes ne peuvent pas être résolus par un algorithme, quelle que soit la puissance de calcul disponible. Le lien est profond : un système formel peut être vu comme un programme, et une démonstration comme une exécution. Gödel montre que certains « programmes logiques » ne peuvent jamais décider toutes les vérités arithmétiques.

Le théorème a également influencé la théorie de la complexité, la logique modale, la philosophie de l'esprit et même la réflexion sur l'intelligence artificielle. Il suggère que tout système mécanique de raisonnement possède des limites intrinsèques, et que certaines vérités ne peuvent être atteintes que par un acte d'intuition ou par un changement de cadre logique.

Enfin, l'incomplétude a des cousins conceptuels dans d'autres paradoxes logiques : le paradoxe du barbier, le paradoxe de Curry, le paradoxe de Yablo ou encore les constructions auto-référentielles en théorie des modèles. Tous montrent que la capacité d'un système à parler de lui-même est une source potentielle d'instabilité.

Le théorème de Gödel n'est donc pas seulement un résultat technique : c'est une frontière philosophique. Il nous rappelle que la formalisation, si puissante soit-elle, ne peut jamais capturer toute la richesse du vrai.

Le théorème de Raymond Smullyan–Martin Hugo Löb (paradoxe de Löb)

Le paradoxe de Löb, formulé par Martin Hugo Löb et popularisé par Raymond Smullyan, explore la logique de la prouvabilité.

Il montre qu'un système formel capable de parler de ses propres démonstrations obéit à une loi étrange : si le système peut prouver que « si une proposition est prouvable, alors elle est vraie », il peut aussi prouver cette proposition elle-même.

Cette boucle logique révèle une structure auto-référente comparable à celle du théorème de Gödel : la frontière entre vérité et démontrabilité devient floue, soulignant les limites internes des systèmes formels

Le paradoxe de Quine

Le paradoxe de Quine, proposé par le philosophe et logicien Willard Van Orman Quine, est une prouesse linguistique : une phrase qui **s'auto-décrit** sans se référer explicitement à elle-même.

Par exemple : « Cette phrase contient trois mots » — une affirmation qui parle d'elle-même sans utiliser de pronom réflexif.

Ce paradoxe explore la capacité du langage à se représenter lui-même, un thème central en logique, en sémantique et en informatique. Il inspire les constructions de programmes capables de s'imprimer eux-mêmes (les *quines*), illustrant la puissance et la subtilité de l'auto-référence algorithmique.

Paradoxes décisionnels et situations impossibles

Le paradoxe du crocodile

Un crocodile promet de rendre un enfant si une prédiction est correcte. Quelle que soit la réponse, une contradiction surgit.

Paradoxe du crocodile
Un crocodile s'empare d'un bébé et propose à la mère:

Si tu devines ce que je vais faire,
je te rends le bébé, sinon je le dévore.
- Tu vas le dévorer, s'écrie la mère.

Si le crocodile dévore le bébé, la mère a bien deviné et le crocodile doit rendre le bébé!
Si le crocodile ne dévore pas le bébé, la mère s'est trompée et le crocodile doit dévorer le bébé!
Il est impossible de sortir de ce dilemme, car les prémisses du raisonnement amènent à une conclusion contradictoire logiquement déduite.
Lewis Carroll a proposé une solution pragmatique:
Si le crocodile dévore le bébé, la mère a dit vrai et le crocodile manque à sa parole.
S'il rend le bébé, la mère s'est trompée et le crocodile manque à sa parole.
De toute manière, l'animal manque à sa parole.
Puisqu'il n'a aucun espoir de satisfaire le sens de l'honneur, on ne peut douter qu'il agira en accord avec sa nature.

Le paradoxe du juge (exécution surprise)

Un prisonnier apprend qu'il sera exécuté un jour « surprise » la semaine suivante. Il raisonne que cela est impossible : chaque jour potentiel peut être éliminé par anticipation.
Pourtant, l'exécution survient et le surprend. Ce paradoxe montre comment la raison logique peut échouer face à la connaissance auto-référente : la prévision d'un événement dépend de la manière dont l'information est intégrée, un thème clé en théorie des algorithmes et en logique temporelle.

Le raisonnement classique se fait par les jours, en remontant la semaine.
Supposons que l'exécution puisse avoir lieu du lundi au vendredi.
Vendredi : Le prisonnier se dit : « Si je suis encore vivant jeudi soir, l'exécution devra avoir lieu vendredi. Donc elle ne serait plus une surprise. » Il conclut : vendredi est impossible.
Jeudi : Ayant éliminé vendredi, il se dit : « Si je suis vivant mercredi soir, et que vendredi est exclu, l'exécution devra avoir lieu jeudi. Donc elle ne serait plus une surprise. » Il élimine jeudi.
Mercredi, mardi, lundi : Il applique le même raisonnement par récurrence à rebours et élimine chaque jour.
Conclusion logique : l'exécution surprise est impossible. Mais lorsqu'elle a lieu, par exemple mercredi, il est effectivement surpris—son raisonnement, pourtant « correct » localement, échoue globalement.

Le paradoxe de l'omniscience

Un être omniscient sait tout ce qui est vrai. Mais peut-il choisir d'ignorer une information ? S'il le peut, il n'est plus omniscient ; s'il ne le peut pas, il perd sa liberté.
Ce paradoxe explore la tension entre connaissance totale et volonté libre, question cruciale pour les modèles d'intelligence artificielle et les systèmes de décision où la connaissance complète peut empêcher toute autonomie.

Le paradoxe du tas (sorite)

Si l'on enlève un grain de sable d'un tas, il reste un tas. En répétant l'opération, à quel moment cesse-t-il d'en être un ?

Ce paradoxe du vague montre que certaines notions — « tas », « grand », « rapide » — résistent à la formalisation stricte. En algorithmique, il inspire les approches floues et les logiques graduelles, où la vérité n'est pas binaire mais continue.

Le paradoxe de l'électeur indifférent

Un vote individuel a une probabilité infime d'influencer une élection, mais chaque vote peut théoriquement tout changer. Ce paradoxe illustre la disproportion entre impact et probabilité**,** essentielle en théorie des jeux et en modélisation algorithmique des comportements collectifs. Il explique pourquoi des actions rationnellement « inutiles » peuvent être décisives dans des systèmes complexes.

Le paradoxe de la loterie

Chaque billet est presque sûrement perdant, mais l'un d'eux gagne nécessairement. Ce paradoxe oppose probabilité individuelle et certitude globale.

En logique algorithmique, il illustre la différence entre raisonnement local et global : un système peut être cohérent dans chaque cas particulier tout en produisant une contradiction lorsqu'on considère l'ensemble.

Le paradoxe de la surprise

Une annonce certaine — « un événement imprévisible aura lieu » — devient impossible si elle doit rester imprévisible. Ce paradoxe relie information**,** anticipation et auto-référence : un système qui prédit sa propre surprise la détruit. En algorithmique, il évoque les limites des modèles prédictifs et des agents auto-observants.

Le paradoxe de la prédiction parfaite

Si une machine connaît ton choix avant que tu le fasses, es-tu encore libre ? Ce paradoxe explore la collision entre déterminisme et décision.

En théorie des algorithmes, il questionne la possibilité d'un modèle prédictif absolu : toute connaissance parfaite du futur modifie ce futur, rendant la prédiction instable.

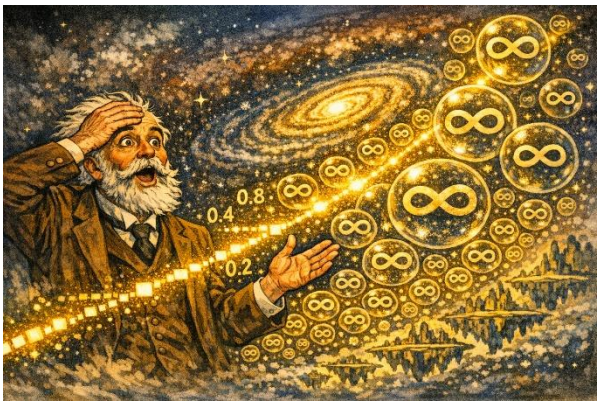
Le paradoxe de la carte

Une carte parfaitement fidèle devrait inclure une représentation d'elle-même, puis de cette représentation, et ainsi de suite.

Cette régression infinie illustre les limites de la modélisation totale : tout système de représentation finit par s'auto-inclure. En algorithmique, il inspire la réflexion sur les modèles récursifs et les simulateurs universels.

Infini et structures profondes

Le paradoxe de la diagonale de Georg Cantor



Le paradoxe de la diagonale de Georg Cantor est l'un des résultats les plus révolutionnaires de l'histoire des mathématiques. Avant Cantor, l'infini était perçu comme un concept uniforme : il n'existait qu'un seul « type » d'infini, celui qui dépasse toute quantité finie. Cantor bouleverse cette vision en montrant que certains infinis sont strictement plus grands que d'autres. Sa méthode, appelée argument diagonal, est à la fois simple, élégante et dévastatrice pour l'intuition.

L'idée centrale consiste à démontrer qu'il est impossible d'énumérer tous les nombres réels, même en théorie. Supposons que l'on tente de lister tous les réels compris entre 0 et 1, chacun écrit sous forme décimale infinie. Cantor imagine alors une diagonale : il prend le premier chiffre du premier nombre, le deuxième chiffre du deuxième nombre, le troisième du troisième, et ainsi de suite. Puis il modifie chaque chiffre de cette diagonale (par exemple en remplaçant un 3 par un 4). Le nombre ainsi construit diffère de chaque nombre de la liste au moins à une position. Il n'y figure donc pas. Conclusion : aucune liste ne peut contenir tous les réels. Ce raisonnement prouve que l'ensemble des réels est non dénombrable, alors que l'ensemble des entiers est dénombrable. Les deux sont infinis, mais l'un est « plus infini » que l'autre. Cette idée, choquante pour ses contemporains, ouvre la voie à une véritable arithmétique des infinis, où l'on compare, additionne et multiplie des tailles d'ensembles infinis.

L'impact de la diagonale dépasse largement la théorie des ensembles. En algorithmique, elle inspire des résultats fondamentaux. Alan Turing s'appuie sur une variante diagonale pour démontrer l'indécidabilité du problème de l'arrêt : il n'existe pas d'algorithme capable de déterminer, pour tout programme, s'il s'arrêtera ou tournera indéfiniment. De même, la diagonale apparaît dans les preuves d'incomplétude de Gödel, où un énoncé se construit pour échapper à toute tentative de démonstration interne.

La méthode de Cantor révèle aussi les limites des systèmes de classification et de compression. Toute tentative de coder un ensemble infini d'objets complexes dans une structure plus simple finit par rencontrer une forme de diagonalisation : un élément « hors liste » peut toujours être construit. En intelligence artificielle, ce principe rappelle qu'aucun modèle ne peut capturer parfaitement toutes les fonctions possibles : il existera toujours un comportement non représenté.

Enfin, le paradoxe diagonal a une portée philosophique. Il montre que l'infini n'est pas un bloc uniforme mais une hiérarchie. Il révèle que certaines vérités mathématiques échappent à toute procédure mécanique. Et il rappelle que la puissance de l'esprit humain réside parfois dans la capacité à concevoir des objets qui dépassent toute énumération.

Cantor a ainsi transformé l'infini d'un mystère homogène en un paysage structuré, riche et vertigineux. Sa diagonale reste l'un des outils conceptuels les plus puissants jamais inventés.

Les limites de la complétude logique

Les limites de la complétude logique marquent une frontière essentielle dans la pensée moderne : aucun système formel ne peut capturer toutes les vérités sans risquer la contradiction. Cette idée, issue du théorème d'incomplétude de Gödel, s'étend bien au-delà de l'arithmétique. Elle révèle que tout langage logique suffisamment expressif contient des énoncés vrais mais indémontrables, et que toute tentative d'ajouter ces vérités engendre de nouvelles zones d'incertitude.

En algorithmique, cette limite se traduit par l'existence de problèmes indécidables : certaines questions ne peuvent être résolues par aucun programme, quelle que soit sa puissance. En philosophie de la connaissance, elle montre que la vérité dépasse la démonstration ; en

intelligence artificielle, qu'un système ne peut entièrement se comprendre lui-même sans générer des paradoxes.

Ces limites ne sont pas des échecs, mais des repères : elles définissent le champ du pensable et du calculable. Elles rappellent que la logique, loin d'être un instrument absolu, est un cadre vivant, traversé par des zones d'ombre où la créativité et l'intuition prennent le relais. La complétude parfaite reste un idéal, mais son impossibilité fonde la richesse même de la pensée rationnelle.

11.1.5 Probabilités, hasard et statistiques

Intuition trompeuse, lois cachées et phénomènes aléatoires.

Le problème du collectionneur de coupons

Si tu veux obtenir tous les objets d'une collection (cartes, autocollants…), les derniers sont les plus difficiles à trouver. Le nombre moyen d'achats croît comme n log n, bien plus que ce que l'intuition suggère.

Le marcheur aléatoire

Un individu avance au hasard à gauche ou à droite. En dimension 1 ou 2, il finit presque toujours par revenir à son point de départ. En dimension 3 ou plus, il peut s'en échapper définitivement.

ANECDOTE

L'algorithme qui a appris à marcher… comme un enfant

Des robots quadrupèdes ont été entraînés avec un algorithme d'apprentissage qui imite le développement moteur des bébés.

Au début, le robot tombe, trébuche, hésite. Puis il ajuste ses mouvements, renforce les bons réflexes, élimine les mauvais. En quelques heures, il marche avec une fluidité étonnante, parfois meilleure que celle d'un animal réel.

Le plus surprenant : lorsqu'on modifie la forme de ses pattes, il réapprend à marcher en quelques minutes.

Une plasticité qui rappelle celle du vivant.

Illustration avec un robot-chien pour livraison (2026)A.

Le paradoxe de Bertrand

On demande la probabilité qu'une corde aléatoire soit plus longue qu'un côté d'un triangle inscrit. Mais selon la manière de choisir la corde, on obtient des réponses différentes — révélant que "au hasard" doit être précisément défini.

Le paradoxe de Saint-Pétersbourg

Un jeu propose un gain théoriquement infini en moyenne, mais les gens refusent de payer cher pour y jouer. Cela montre que l'espérance mathématique ne reflète pas toujours la perception humaine du risque.

Le problème du secrétaire

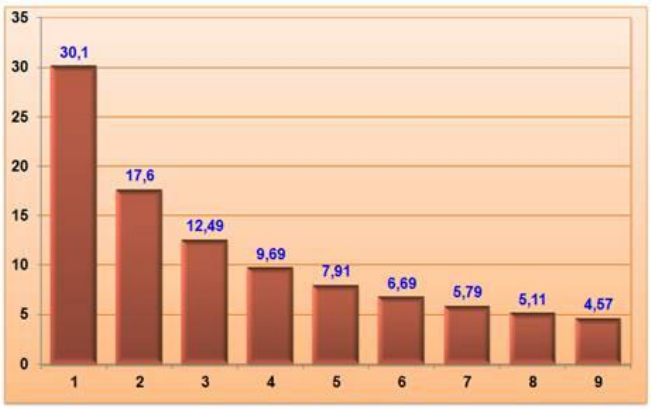
On observe des candidats un par un et on doit décider immédiatement. La stratégie optimale consiste à rejeter environ 37 % des premiers, puis choisir le premier meilleur que tous les précédents.

Le problème de la ruine du joueur

Un joueur qui mise contre une banque infinie finit presque sûrement ruiné, même avec un jeu équitable. Le hasard favorise celui qui peut jouer indéfiniment.

La loi de Benford

La loi de Benford, ou loi des premiers chiffres, révèle une régularité étonnante dans les données naturelles : les nombres commencent bien plus souvent par **1** que par **9**. Dans de nombreux ensembles réels — revenus, populations, longueurs de fleuves, puissances électriques — environ 30 % des valeurs débutent par 1, tandis que moins de 5 % commencent par 9. Cette distribution n'est pas arbitraire : elle découle d'une loi logarithmique. Les nombres croissent de manière multiplicative, et les intervalles correspondant aux premiers chiffres ne sont pas équivalents sur une échelle logarithmique.



Découverte par Simon Newcomb puis formalisée par Frank Benford, cette loi s'applique aux données couvrant plusieurs ordres de grandeur et non limitées par des bornes artificielles. En algorithmique, elle sert à détecter les anomalies statistiques : les fraudes comptables ou les manipulations de données s'écartent souvent de la distribution de Benford. En mathématiques, elle illustre la puissance des lois émergentes : des régularités globales surgissent du chaos des valeurs locales.

La loi de Benford rappelle que même dans le désordre apparent des nombres, une structure subtile gouverne le monde réel — une harmonie logarithmique que les algorithmes savent désormais reconnaître et exploiter.

La loi de Zipf

Dans les langues naturelles, le mot le plus fréquent apparaît environ deux fois plus que le second, trois fois plus que le troisième, etc. Une régularité surprenante dans le chaos du langage.

Le problème du maximum

Dans une suite aléatoire, chaque position a la même probabilité d'être le maximum. Pourtant, détecter ce maximum en ligne est difficile — lien avec le problème du secrétaire.

Paradoxes probabilistes classiques

Le problème des trois cartes

Si une face visible est rouge, il est plus probable que l'autre face soit rouge plutôt que noire. L'intuition oublie que certaines cartes ont deux faces rouges.

Énoncé du paradoxe

Trois cartes sont posées face cachée :

Carte A : rouge des deux côtés

Carte B : rouge d'un côté, noire de l'autre

Carte C : noire des deux côtés

On tire une carte au hasard et on voit une **face rouge**.
Quelle est la probabilité que l'autre face soit rouge ?

Raisonnement correct:

Il y a **six faces possibles** au total (deux par carte). Parmi elles, **trois sont rouges** : deux sur la carte A, une sur la carte B. Si la face visible est rouge, elle peut être :

- A1 ou A2 (de la carte A) → autre face rouge
- B1 (de la carte B) → autre face noire

Sur **trois cas possibles**, **deux** donnent une autre face rouge. La probabilité est donc **2/3**, et non 1/2.

Interprétation

L'erreur intuitive vient du fait qu'on raisonne sur les cartes plutôt que sur les faces.

Ce paradoxe illustre la puissance des probabilités conditionnelles : l'information partielle (une face rouge) modifie l'espace des possibilités.

En algorithmique, il rappelle que le mode de sélection des données influence les résultats — un principe fondamental en apprentissage automatique et en logique bayésienne.

Les boîtes de Bertrand

Après avoir tiré une pièce dorée, il est plus probable d'être dans la boîte contenant deux pièces dorées. Les cas favorables ne sont pas équiprobables.

Le problème du taxi de New York

Le problème du taxi de New York illustre une idée essentielle : nos intuitions se trompent lorsqu'on ignore les fréquences de base. Un témoin affirme avoir vu un taxi bleu, et son témoignage est fiable. Pourtant, si dans la ville 85 % des taxis sont verts et seulement 15 % sont bleus, il est statistiquement plus probable que le taxi observé soit… vert.

Pourquoi ? Parce que même un témoin fiable se trompe parfois, et lorsque la catégorie rare (les taxis bleus) est très minoritaire, les erreurs d'identification pèsent lourd.

Le théorème de Bayes combine deux informations : la fiabilité du témoin et la proportion réelle de chaque type de taxi. Le résultat montre que la fréquence de base domine souvent l'intuition.

Ce paradoxe apparent rappelle qu'en présence d'incertitude, il faut toujours intégrer les probabilités initiales avant de tirer une conclusion.

Le paradoxe de l'aéroport

On attend plus longtemps un bus lorsque les intervalles sont irréguliers, car on a plus de chances d'arriver pendant un long intervalle.

Le tirage biaisé (méthode de John von Neumann)

Avec une pièce biaisée, on peut obtenir un résultat équitable en regardant des paires de tirages (pile-face vs face-pile).

Dés, cartes et distributions

Le maximum de deux dés

Le maximum de deux dés est souvent élevé (5 ou 6), car il suffit qu'un seul dé soit grand.

Le minimum de deux dés

À l'inverse, le minimum est souvent faible. Ces distributions ne sont pas symétriques.

Le problème du coupon rare

Plus un événement est rare, plus le temps d'attente moyen devient énorme. Une manifestation des lois exponentielles.

La marche aléatoire absorbante

Un marcheur se déplace jusqu'à atteindre une frontière. On peut calculer précisément la probabilité d'atteindre chaque bord.

La machine de Galton

Des billes tombant sur des clous forment une courbe en cloche : la fameuse distribution normale.

Le tirage hypergéométrique

On tire sans remise : les probabilités changent à chaque tirage. Modèle utilisé pour les cartes ou les urnes finies.

Le modèle de Pólya

À chaque tirage, on renforce la couleur tirée. Ce mécanisme amplifie le hasard initial et modélise des phénomènes de contagion ou d'effet boule de neige.

Paradoxes sociaux et biais statistiques

Le paradoxe de l'amitié

Le paradoxe de l'amitié révèle une intuition trompeuse : en moyenne, tes amis ont plus d'amis que toi.

Ce phénomène n'a rien de personnel ; il découle d'une propriété structurelle des réseaux sociaux. Les personnes très connectées — celles qui ont beaucoup d'amis — apparaissent plus souvent dans les cercles relationnels des autres, simplement parce qu'elles ont davantage de chances d'être choisies comme « ami ». Elles sont donc surreprésentées dans l'expérience individuelle de chacun.

À l'inverse, les personnes peu connectées apparaissent rarement dans les réseaux des autres, même si elles sont nombreuses.

Résultat : lorsqu'on regarde la moyenne du nombre d'amis de nos amis, elle dépasse presque toujours notre propre nombre d'amis.

Ce paradoxe montre comment la structure d'un réseau peut déformer notre perception du monde social, en donnant l'impression que les autres sont plus populaires que nous.

Le paradoxe de l'inspection

On observe plus souvent les événements longs que courts (bus lents, files longues), ce qui biaise notre perception.

Le choix optimal de parking

Faut-il se garer tôt ou chercher plus loin ? Un compromis entre probabilité de trouver une place et coût de marche.

Séries, tirages et comportements extrêmes

Le tirage sans remise (cartes)

Les probabilités dépendent fortement de l'ordre et des cartes déjà tirées. Essentiel pour les jeux de stratégie.

Le tirage avec remise

Chaque tirage est indépendant. Cela conduit à des lois comme la géométrique ou l'exponentielle.

La plus longue série

Dans 100 lancers de pièce, la plus longue série de "pile" est généralement autour de 6 ou 7 — plus longue que ce qu'on imagine.

Le paradoxe de l'échantillonnage

Un échantillon peut être trompeur même s'il est aléatoire, surtout s'il est petit. D'où l'importance de la taille et du biais de sélection.

11.1.6 Problèmes de transport, optimisation physique et scénarios amusants

Mouvement, stratégie, physique intuitive et pièges du bon sens

Le transport de bananes par les chameaux

Un chameau transporte des bananes mais en consomme en route.

La stratégie optimale consiste à faire des allers-retours et à créer des "dépôts intermédiaires". Le problème illustre une optimisation avec pertes progressives.

> **Le défi: Avec un stock de 3 000 bananes, comment en transporter un maximum sur 1 000 km avec un chameau qui ne peut en porter que 1 000 à la fois et qui en consomme une à chaque kilomètre ?**

Le loup, la chèvre et le chou

Traverser une rivière avec des contraintes de prédation. La solution repose sur une alternance précise des trajets, montrant l'importance de la planification séquentielle.

Il faut aller de l'autre côté de la rivière en passant sur un pont étroit ne pouvant supporter que deux personnes. Il fait nuit, alors l'un d'eux doit porter une torche unique.

- **A traverse en 1 minute;**
- **B traverse en 2 minutes;**
- **C traverse en 5 minutes;**
- **D traverse en 8 minutes; et**

Le plus lent fixant la vitesse, comment minimiser le temps de passage des quatre personnes ? Est-ce possible en 15 minutes ?

Le pont et la torche

Quatre personnes avancent à des vitesses différentes et doivent partager une torche. La solution optimale minimise les allers-retours des plus rapides — un bel exemple d'optimisation combinatoire.

Le train et le moustique

Un moustique vole entre deux trains qui se rapprochent. Plutôt que de suivre ses allers-retours, il suffit de calculer le temps avant collision des trains, puis multiplier par sa vitesse.

Les cruches d'eau

Avec deux contenants de tailles différentes, on peut mesurer un volume précis grâce à des remplissages et vidages successifs. Ce problème illustre l'arithmétique modulaire.

Problème

Nous disposons de trois cruches de 12, 8 et 5 litres.

Le but est de partager en deux parties égales l'eau de la grande cruche pleine.

La solution nécessite sept étapes à partir de l'état initial.

En bleu clair, on montre les cas où la cruche est pleine.

Solution

12	0	0
4	8	0
4	3	5
9	3	0
9	0	3
1	8	3
1	6	5
6	6	0

Le camion trop haut

Un camion coincé sous un pont : il suffit de dégonfler les pneus. Une solution simple qui montre que les meilleures idées ne sont pas toujours les plus complexes.

La fourmi sur un ruban élastique

Le ruban s'allonge à chaque instant, mais la fourmi progresse quand même. Elle atteint la fin… mais après un temps énorme. Un exemple de croissance logarithmique contre intuition.

La fourmi sur un cube

Le plus court chemin entre deux points sur un cube n'est pas forcément le long des arêtes, mais peut passer par un "dépliage" des faces.

Problème

Un cube de 4 cm de côté.

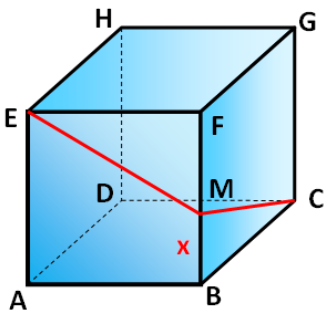
Une fourmi part du point E pour atteindre le point C.

Elle passe par le point M situé à une distance x du point B.

On se propose de déterminer la position du point M pour minimiser le parcours de la fourmi.

On dessine le patron du cube en respectant la notation des sommets.

On représente le trajet AMC quelconque (vert) et le trajet AMC avec M au milieu de FB (rouge)



image

La ligne droite est le plus court chemin d'un point à un autre. C'est bien le trajet rouge qui représente une longueur minimum.	

Mouvement, vitesse et relativité intuitive

Le bateau et le courant

Remonter un fleuve prend plus de temps que le descendre. La vitesse effective dépend de la combinaison vitesse propre + courant.

ANECDOTE	**L'algorithme qui a retrouvé un navire disparu depuis 90 ans** En 2018, un algorithme d'analyse d'images sous-marines a permis de retrouver le USS Indianapolis, un navire coulé en 1945 et jamais localisé malgré des décennies de recherches. L'algorithme a analysé des milliers d'images sonar floues, repérant des motifs géométriques impossibles à distinguer pour un humain : angles réguliers, ombres métalliques, symétries improbables. Il a réduit la zone de recherche de plusieurs milliers de kilomètres carrés à quelques dizaines. L'équipe d'exploration a plongé exactement au bon endroit. Un navire fantôme, disparu depuis 73 ans, retrouvé grâce à un algorithme qui « voit » dans le bruit.

Le train et le tunnel

Un train plus long que le tunnel peut être entièrement contenu dedans dans certains référentiels. Une conséquence de la relativité restreinte.

Le train et la lumière

Un éclair dans un train en mouvement n'est pas perçu simultanément par tous les observateurs. La simultanéité dépend du référentiel.

Optimisation pratique et ingénierie

Le remplissage optimal d'un camion

Comment empiler des objets pour maximiser l'espace utilisé ? C'est une version géométrique du problème du sac à dos, souvent très complexe.

Le déménagement optimal

Transporter des objets fragiles demande de minimiser les chocs et les manipulations. Une optimisation entre sécurité et efficacité.

Le pont oscillant

Des piétons marchant en rythme peuvent amplifier les oscillations d'un pont. Ce phénomène de résonance a causé de véritables incidents.

Le vélo et le vent

L'énergie dépensée contre le vent est supérieure au gain obtenu avec le vent dans le dos. L'effet n'est pas symétrique à cause des forces de traînée.

La brachistochrone (problème du ski)

Quelle est la trajectoire la plus rapide entre deux points sous gravité ? Ce n'est pas la ligne droite, mais une courbe appelée cycloïde.

Formes physiques naturelles

La chaînette

Une chaîne suspendue prend une forme en cosinus hyperbolique. Ce n'est pas une parabole, contrairement à l'intuition.

Le câble suspendu

Même phénomène : la forme réelle minimise l'énergie, pas la distance.

Le pendule amorti

Un pendule perd de l'énergie avec le temps. Son mouvement devient progressivement plus lent jusqu'à s'arrêter.

Le ressort non linéaire

Les vrais ressorts ne suivent pas toujours la loi idéale de Hooke. À grande déformation, le comportement devient complexe.

Le frottement sec

La force de frottement dépend du mouvement (statique ou dynamique) et peut changer brusquement, créant des effets de blocage.

Fluides et écoulements

Le tonneau de vin

Mesurer un volume sans ouvrir ou vider un tonneau demande des astuces géométriques ou physiques (poids, inclinaison…).

Remplissage du tonneau ? Ces deux hommes observent le niveau du liquide (eau, bière, peu importe). L'un prétend qu'il y en a moins de la moitié. Ce que conteste l'autre en disant qu'il y en plus que la moitié. Comment simplement les mettre d'accord? Sans aucune mesure!	

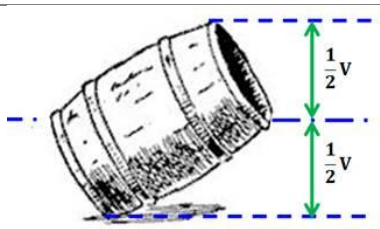
Solution

Il suffit de pencher le tonneau jusqu'à ce que le liquide affleure l'ouverture. Alors:

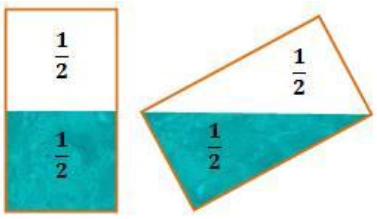
- si on observe une partie du fond, il n'est pas à moitié plein;
- si on n'observe aucune partie du fond, il est plus qu'à moitié plein; et
- si le liquide tangente le fond, le tonneau est juste à moitié plein

Explication

Coupé par un plan de symétrie, le tonneau présente le même volume en haut et en bas.



Justification muette



Le tonneau percé (loi de Evangelista Torricelli)

Le débit d'un liquide dépend de la hauteur de liquide au-dessus du trou. Plus le niveau baisse, plus le débit diminue.

Le siphon

Un liquide peut monter temporairement avant de descendre, grâce à la pression atmosphérique et à la continuité du fluide.

Le remplissage d'un bassin

Deux robinets remplissent un bassin à des vitesses différentes : les débits s'additionnent. Un classique d'arithmétique appliquée.

Le remplissage de silos

Les grains ne se comportent ni comme un solide ni comme un fluide. Leur répartition peut provoquer des blocages ou des effondrements.

Scénarios dynamiques et contre-intuitifs

Le pont-levis

Lever un pont dépend du couple exercé et de la position du centre de masse. Un problème de mécanique classique.

Le train fantôme

Un train qui accélère puis freine peut parcourir une distance plus grande que prévu si on se fie uniquement à la vitesse moyenne intuitive.

11.1.7 Informatique théorique, automates et complexité

Limites du calcul, efficacité des algorithmes et structures abstraites

Le problème de l'arrêt (Alan Turing)

Peut-on savoir à l'avance si un programme va s'arrêter ou tourner indéfiniment ? Turing a montré que c'est impossible en général. C'est une limite fondamentale de tout système informatique.

P vs NP

Existe-t-il des problèmes dont les solutions sont faciles à vérifier mais difficiles à trouver ? Si P = NP, toute solution vérifiable rapidement serait aussi calculable rapidement — une révolution majeure.

La satisfiabilité booléenne (SAT)

Peut-on attribuer des valeurs vrai/faux à des variables pour satisfaire une formule logique ? Premier problème prouvé NP-complet, il sert de référence pour la complexité.

Le tri optimal

Comparer des éléments pour les trier nécessite au moins n log n comparaisons dans le pire cas. Une limite théorique incontournable.

Le plus long sous-mot commun

Trouver la plus longue séquence commune à deux chaînes. Utilisé en bioinformatique (comparaison d'ADN) et en traitement de texte.

L'algorithme qui a résolu un crime vieux de 30 ans

Dans une affaire criminelle non résolue, des enquêteurs ont utilisé un algorithme de comparaison génétique pour analyser des fragments d'ADN dégradés.

L'algorithme a reconstitué un profil partiel, puis l'a comparé à des millions de profils anonymisés. Il a trouvé une correspondance familiale, ce qui a permis de remonter jusqu'au suspect.

L'affaire a été résolue après trois décennies d'impasse.

Un algorithme devenu détective, capable de lire dans les traces du passé.

Le codage optimal : la limite de Shannon et l'entropie de l'information

Claude Shannon, fondateur de la théorie de l'information, a posé dès 1948 les bases mathématiques de la compression sans perte. Son idée centrale est simple et révolutionnaire : toute source d'information possède une quantité minimale de bits nécessaires pour la représenter, quantité qu'il appelle *entropie*. Cette entropie, notée (H), mesure l'incertitude moyenne d'un symbole émis par la source. Plus une source est prévisible, plus son entropie est faible ; plus elle est variée et imprévisible, plus elle est élevée.

Shannon démontre que la longueur moyenne minimale d'un code sans perte ne peut jamais être inférieure à l'entropie de la source**.** Autrement dit, si une source a une entropie de (H = 2,3) bits par symbole, aucun algorithme de compression ne pourra descendre en dessous de cette limite, quelle que soit son ingéniosité. Cette borne est fondamentale : elle ne dépend ni de la puissance de calcul, ni de la créativité du programmeur, mais de la structure statistique même des données.

Les algorithmes de compression modernes — Huffman, Lempel-Ziv, arithmetic coding — cherchent précisément à s'approcher de cette limite. Ils exploitent les redondances, les régularités et les biais statistiques présents dans les données pour réduire la taille des messages. Lorsqu'un fichier semble "incompressible", cela signifie simplement que son entropie est déjà proche du nombre de bits utilisés pour le représenter.

La théorie de Shannon établit également qu'un codage optimal doit attribuer des codes courts aux symboles fréquents et des codes longs aux symboles rares**,** principe qui inspire directement les arbres de Huffman. Mais même les meilleurs codes ne peuvent franchir la barrière de l'entropie : elle constitue la compression maximale possible sans perte**.**

Ainsi, la théorie de Shannon fournit non seulement un objectif — atteindre l'entropie — mais aussi une boussole conceptuelle pour comprendre pourquoi certaines données se compressent facilement et d'autres pas du tout.

La machine universelle de Alan Turing

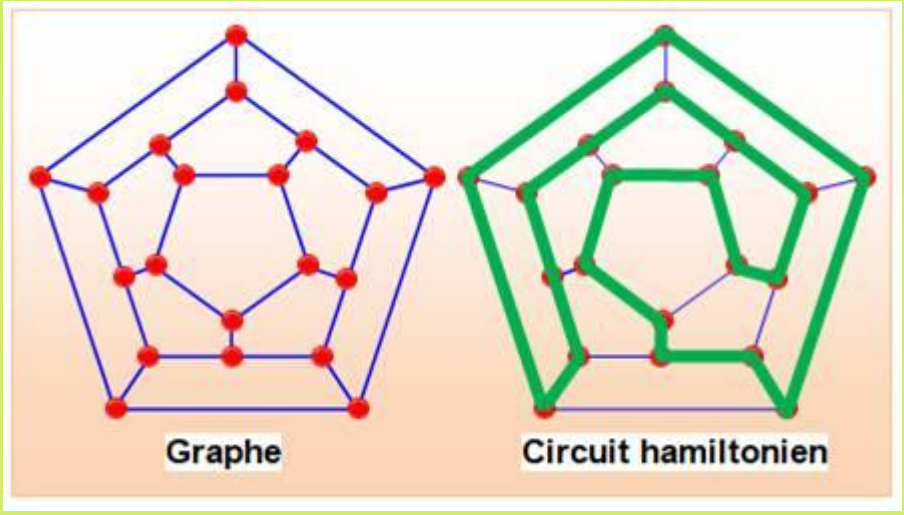
Un modèle abstrait capable de simuler n'importe quel calcul. C'est le fondement conceptuel des ordinateurs modernes. *(Sujet traité dans le cours du document).*

Le graphe hamiltonien

Existe-t-il un cycle passant une fois par chaque sommet ? Un problème difficile, lié au voyageur de commerce.

Chemin (circuit) hamiltonien

Un tracé qui passe par tous les sommets (sans forcément parcourir toutes les arêtes du graphe).
Un graphe hamiltonien est un graphe possédant au moins un cycle passant par tous les sommets une fois et une seule. Un tel cycle élémentaire est alors appelé cycle hamiltonien.



Le graphe eulérien

Un graphe est dit *eulérien* lorsqu'il possède un cycle qui parcourt chaque arête exactement une fois avant de revenir au point de départ. Cette question, posée dès le XVIII^{e} siècle par Euler avec le célèbre problème des ponts de Königsberg, admet une réponse étonnamment simple : tout dépend du degré des sommets, c'est-à-dire du nombre d'arêtes qui y aboutissent.

Le théorème d'Euler affirme qu'un graphe connexe possède un cycle eulérien si et seulement si tous ses sommets ont un degré pair. Cette condition garantit qu'à chaque fois que l'on entre dans un sommet par une arête, on peut en ressortir par une autre, sans jamais se retrouver bloqué. Si un seul sommet a un degré impair, un cycle complet devient impossible.

Ainsi, la structure globale du graphe se résume à une propriété locale : la parité des degrés. Une élégance mathématique qui explique la puissance de la théorie des graphes.

Indécidabilité et limites du calcul

Le problème de Post (Emil Post)

Un système de réécriture très simple peut déjà être indécidable. Même des règles élémentaires peuvent produire une complexité infinie.

Le problème de correspondance de Post
Peut-on aligner des paires de mots pour former la même chaîne ? Ce problème est indécidable, malgré sa formulation simple.

Le problème du mot dans un groupe
Déterminer si une expression équivaut à l'élément neutre. Selon le groupe, cela peut être impossible à décider.

Le problème du mot dans un automate
Déterminer si un automate accepte un mot donné. Un problème fondamental en théorie des langages.

Algorithmes sur les chaînes

Le plus long palindrome
Trouver la plus longue sous-chaîne symétrique. Utilisé en bioinformatique et en traitement du texte.

Le plus long sous-mot croissant
Trouver une sous-séquence strictement croissante. Problème classique avec des solutions efficaces surprenantes.

Le plus court super-mot
Assembler plusieurs fragments pour obtenir une chaîne minimale contenant tous les fragments. Important en reconstruction de génomes.

Tri et organisation des données

Le tri externe
Comment trier des données trop grandes pour tenir en mémoire ? On utilise des lectures/écritures optimisées sur disque.

Le tri parallèle
Répartir le travail de tri sur plusieurs processeurs pour gagner du temps.

Le tri distribué
Trier des données réparties sur plusieurs machines. Crucial pour le big data.

Le tri par tas (heap sort)
Utilise une structure en arbre pour trier efficacement en n log n.

Le tri par fusion (merge sort)
Divise pour régner : on découpe, trie, puis fusionne. Stable et optimal.

Le tri par insertion
Simple et efficace pour de petites listes ou des données presque triées.

Le tri par sélection
On choisit successivement le plus petit élément. Simple mais peu efficace.

Le tri bitonique
Un tri parallèle basé sur des séquences structurées. Utilisé en architecture matérielle.

Le tri pair-impair
Un tri parallèle simple basé sur des échanges locaux.

Le tri topologique
Ordonner des tâches en respectant des dépendances. Indispensable en planification et compilation.

Problèmes difficiles sur les graphes

Le plus long chemin
Trouver le chemin le plus long dans un graphe est très difficile (contrairement au plus court chemin).

Le graphe de couverture
Trouver une structure minimale satisfaisant un ensemble de contraintes. Généralisation de nombreux problèmes d'optimisation.

Le graphe de domination
Choisir un ensemble minimal de sommets qui "couvrent" tous les autres. Applications en réseaux et surveillance.

11.1.8 Jeux, puzzles et récréations mathématiques

Stratégie, combinatoire, intuition et plaisir du défi.

Le Rubik's Cube
Chaque configuration peut être résolue en au plus 20 mouvements. Derrière ce cube se cache une structure de groupe extrêmement riche.

Le Tangram
Sept pièces géométriques permettent de former une infinité de silhouettes. Un jeu qui développe la visualisation spatiale et la décomposition des formes.

Le taquin (15-puzzle)
Un puzzle de glissement où certaines configurations sont impossibles à atteindre. La raison tient à une propriété de parité cachée.

Les soldats de John Horton Conway
Des pions avancent en sautant les uns par-dessus les autres. On pourrait croire pouvoir monter indéfiniment… mais une limite infranchissable existe.

Le jeu de la vie (Game of Life) - Quand des règles élémentaires engendrent une complexité inattendue

Le Jeu de la vie, imaginé par le mathématicien John Conway en 1970, est devenu l'un des exemples les plus emblématiques de la manière dont des règles extrêmement simples peuvent produire des comportements d'une richesse étonnante. Il s'agit d'un automate cellulaire : une grille composée de cellules pouvant être « vivantes » ou « mortes », évoluant au fil du temps selon trois règles élémentaires. Une cellule vivante survit si elle a deux ou trois voisines vivantes ; une cellule morte devient vivante si elle en a exactement trois ; dans tous les autres cas, elle meurt ou reste morte. Rien de plus.

Pourtant, de cette mécanique minimaliste émergent des structures dynamiques qui semblent presque dotées d'intentions. Certaines configurations restent stables, d'autres oscillent indéfiniment, et d'autres encore se déplacent à travers la grille comme des organismes rudimentaires. Le plus célèbre de ces « êtres » est sans doute le *glider*, une petite forme qui se déplace en diagonale, donnant l'impression d'une créature qui migre.

Plus fascinant encore, certaines structures sont capables de se reproduire ou de générer des motifs secondaires. Des ingénieurs du Jeu de la vie ont même construit des configurations capables de simuler des opérations logiques, puis des circuits complets. On sait aujourd'hui que le Jeu de la vie est Turing-complet : en théorie, il peut simuler n'importe quel calcul réalisable par un ordinateur.

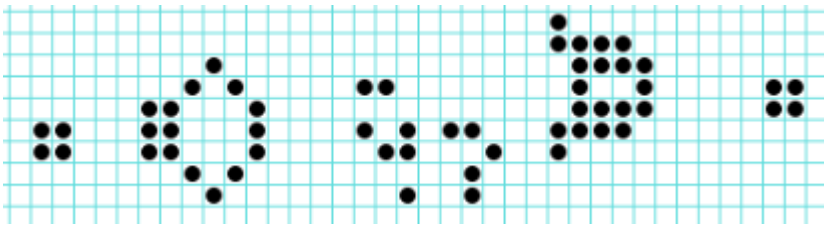
Cette propriété a profondément marqué la réflexion sur les systèmes complexes et, par extension, sur l'intelligence artificielle. Elle montre qu'un comportement sophistiqué n'exige pas nécessairement des règles compliquées ; il peut émerger spontanément de l'interaction locale d'éléments simples. Le Jeu de la vie sert ainsi de métaphore puissante pour comprendre comment des modèles d'IA, eux aussi fondés sur des règles élémentaires et des interactions massives, peuvent produire des dynamiques inattendues.

En observant ces automates, on saisit mieux pourquoi certaines propriétés émergentes des IA surprennent leurs concepteurs : la complexité n'est pas toujours programmée, elle peut surgir naturellement de la structure même du système.

Le jeu de Nim

Le jeu de Nim est un classique des mathématiques combinatoires, souvent présenté comme l'exemple parfait d'un jeu où la stratégie optimale découle directement d'une analyse arithmétique. Le principe est simple : plusieurs tas d'objets sont disposés devant les joueurs, qui retirent à tour de rôle autant d'éléments qu'ils le souhaitent dans un seul tas. Celui qui prend le dernier objet gagne (ou perd, selon la variante).

Ce qui rend Nim fascinant, c'est que sa stratégie gagnante repose sur une opération binaire : le **XOR** (ou somme bit à bit sans retenue). En calculant le XOR de la taille de tous les tas, on obtient une valeur appelée *nim-sum*. Si cette valeur est nulle, la position est perdante pour le joueur qui doit jouer ; si elle est non nulle, il existe un coup gagnant permettant de rétablir un nim-sum nul.



Ainsi, Nim illustre de manière éclatante comment une règle mathématique abstraite peut se traduire en stratégie parfaite, démontrant la puissance du raisonnement algorithmique appliqué au jeu.

Le jeu de Wythoff

Deux piles d'objets, avec des coups combinés possibles. Les positions gagnantes suivent une structure liée au nombre d'or.

Le jeu de Hex

Deux joueurs cherchent à relier des bords opposés. Il est mathématiquement impossible de faire match nul — quelqu'un gagne toujours.

Le Mastermind

Deviner un code en minimisant le nombre d'essais. Derrière ce jeu se cachent des questions de stratégie optimale et d'information.

Les pentaminos

Assembler des formes composées de cinq carrés. Un terrain infini pour les puzzles de pavage.

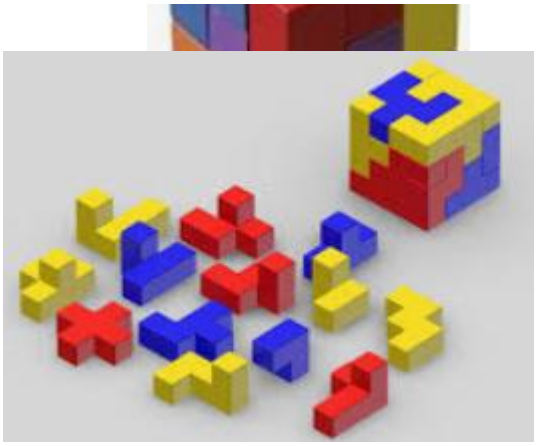
Puzzles en volume et constructions

Le puzzle Soma

Assembler 7 pièces pour former un cube. Un classique de la visualisation en 3D *(Illustration)*.

Le puzzle Burr

Des pièces de bois imbriquées qui semblent impossibles à séparer. Leur structure repose sur des séquences précises de mouvements.



Le serpent cubique

Une chaîne articulée qui peut se replier en cube. Un défi de visualisation spatiale *(Illustration)*.

Le serpent sphérique

Une chaîne articulée qui peut se replier en un volume proche de la sphère. Assemblage de formes triangulaires.

Le cube impossible

Un objet qui semble cohérent en projection, mais impossible à construire en 3D réelle. Un jeu sur la perception.

Le cube de Bedlam

Un puzzle 3D composé de 13 pièces irrégulières, avec très peu de solutions. Extrêmement difficile *(Illustration)*.

Les variantes du cube Soma

Des extensions en trois dimensions, avec des contraintes supplémentaires qui complexifient la construction.

Labyrinthes et puzzles dynamiques

Le labyrinthe invisible

Les murs ne sont révélés qu'en les rencontrant. Le joueur doit construire mentalement la structure du labyrinthe.

Le taquin généralisé

Dans des versions plus grandes ou différentes, certaines configurations restent impossibles à atteindre — une propriété structurelle persistante.

Le jeu de la vie (exploration)

Certains motifs se déplacent (gliders), d'autres explosent ou se stabilisent. Un univers émergent fascinant.

Jeux combinatoires avancés

Le jeu de Wythoff (analyse)

Les positions gagnantes suivent des suites liées au nombre d'or, révélant une structure profonde.

Le jeu de Hex (structure)

Son impossibilité de match nul est liée à des propriétés topologiques du plan.

Le jeu de Go – Un défi algorithmique longtemps jugé insurmontable

Le jeu de Go occupe une place unique dans l'histoire de l'intelligence artificielle. Sa complexité dépasse de très loin celle des échecs : on estime le nombre de configurations possibles à environ 10^{170} , un chiffre si gigantesque qu'il excède le nombre d'atomes dans l'univers observable. Cette immensité rend impossible toute approche fondée sur l'exploration exhaustive ou même partielle de l'arbre des coups. Là où un programme d'échecs peut s'appuyer sur une évaluation tactique locale, le Go exige une compréhension globale, presque intuitive, des formes, des équilibres territoriaux et des tensions stratégiques.

Pendant des décennies, cette nature profondément holistique a résisté à la formalisation algorithmique. Les heuristiques classiques échouaient à capturer la subtilité des motifs, et les méthodes de recherche combinatoire se perdaient dans un océan de possibilités. Même les meilleurs programmes du début des années 2000 restaient au niveau amateur.

La percée est venue avec l'apprentissage profond et les réseaux de neurones, capables de reconnaître des structures spatiales complexes et d'estimer la valeur d'une position sans calculer toutes ses ramifications. Combinés aux méthodes de Monte-Carlo, ils ont permis à des systèmes comme AlphaGo de développer une forme d'intuition artificielle.

Le Go a ainsi servi de révélateur : certains problèmes ne cèdent pas à la force brute, mais exigent des modèles capables d'abstraction, de généralisation et d'émergence — des qualités au cœur de l'IA moderne.

Le jeu de Kayles

Des quilles à faire tomber selon des règles précises. Analyse basée sur la théorie des jeux combinatoires.

Le jeu de Dawson

Une variante de Kayles avec des contraintes supplémentaires, rendant l'analyse plus complexe.

Le jeu de Treblecross

Un jeu sur une ligne où il faut éviter de créer une configuration perdante. Similaire au morpion, mais asymétrique.

Le jeu de Chomp
Un jeu sur une grille où la stratégie gagnante existe, mais reste difficile à expliciter.

Le jeu de Hackenbush
Un jeu graphique coloré où les coups modifient la structure. Étudié en théorie des jeux combinatoires.

Le jeu de Domineering
Deux joueurs placent des dominos selon des orientations différentes. Le jeu devient asymétrique et stratégique.

Le jeu Toads and Frogs
Un jeu inventé par John Horton Conway, mêlant déplacements et blocages, avec une structure mathématique élégante.

11.1.9 Physique, chaos et systèmes dynamiques

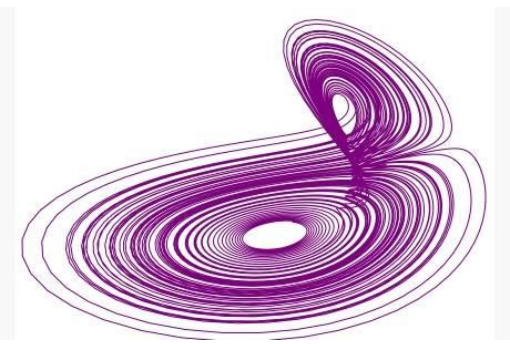
(Sensibilité, imprédictibilité et structures cachées du mouvement)

Le problème des trois corps
Décrire précisément le mouvement de trois objets en interaction gravitationnelle est, en général, impossible à résoudre de façon exacte. De petites variations initiales produisent des trajectoires totalement différentes : le chaos apparaît naturellement.

Le double pendule
Deux pendules attachés l'un à l'autre forment un système extrêmement sensible aux conditions initiales. Un minuscule écart au départ entraîne des mouvements totalement divergents.

Le billard chaotique
Une bille rebondissant dans une table aux parois courbes suit des trajectoires imprévisibles. Un système déterministe… mais pratiquement imprédictible.

L'effet papillon (Edward Lorenz)
L'effet papillon, découvert par Edward Lorenz dans les années 1960, montre qu'un système peut être entièrement déterministe tout en restant profondément imprévisible. En relançant un modèle météorologique avec des valeurs initiales légèrement arrondies, Lorenz observe que la trajectoire du système diverge complètement. Une variation infime — quelques millièmes — suffit à produire un résultat radicalement différent. C'est le cœur du chaos déterministe : les lois sont simples, mais la sensibilité aux conditions initiales rend toute prédiction à long terme impossible.



L'image du papillon qui bat des ailes au Brésil et déclenche une tornade au Texas n'est qu'une métaphore, mais elle traduit bien cette sensibilité.
Cette dynamique se visualise dans ce qu'on appelle l'attracteur de Lorenz *(Illustration)*, une forme mathématique fascinante qui ressemble à deux ailes de papillon. Cet attracteur n'est pas un point fixe ni un cycle régulier : c'est une structure vers laquelle

le système tend, tout en ne repassant jamais exactement par le même état. Il montre que le chaos n'est pas du désordre pur, mais un ordre complexe, organisé autour d'une géométrie stable. L'attracteur de Lorenz est devenu l'icône du chaos : une preuve que des équations déterministes peuvent engendrer des comportements riches, turbulents et impossibles à prévoir en détail.

La turbulence

L'écoulement chaotique des fluides reste l'un des plus grands défis de la physique. Les équations (Navier–Stokes) sont connues, mais leurs solutions sont extrêmement complexes.

Les fractales de Benoît Mandelbrot et de Julia

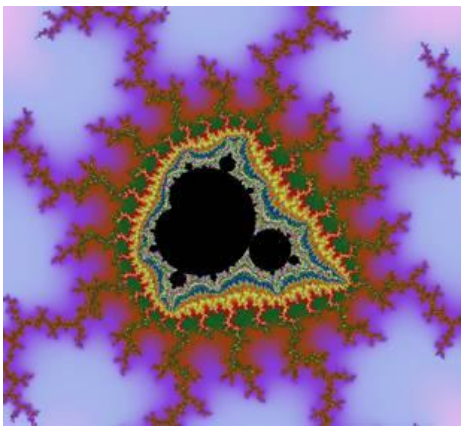
Les fractales de Benoît Mandelbrot et de Gaston Julia ont révélé un univers mathématique d'une richesse insoupçonnée : celui de formes infiniment détaillées, qui se répètent à toutes les échelles.

Leur point de départ est pourtant d'une simplicité déconcertante : une **équation (algorithme) itérative** appliquée encore et encore. La forme la plus célèbre est:

$$\mathbf{z_{n+1} = z_n^2 + c}$$

où z et c sont des nombres complexes.

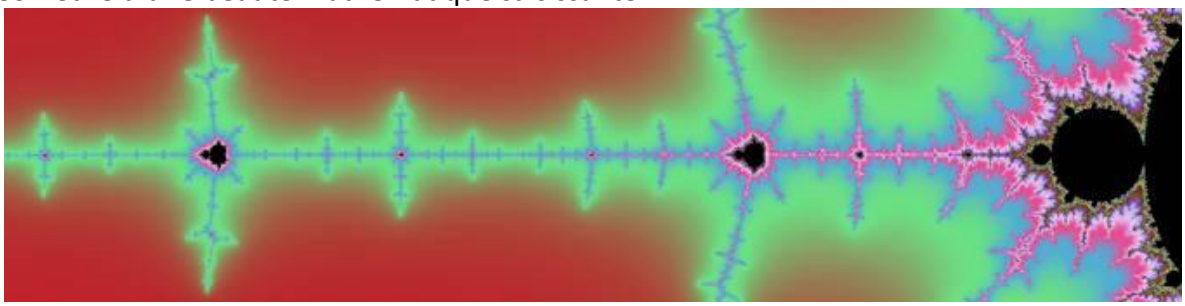
À chaque itération, on observe si la suite diverge ou reste confinée. De cette règle élémentaire naissent des structures d'une complexité inouïe.



Les **ensembles de Mandelbrot** correspond à l'ensemble des valeurs de c pour lesquelles la suite ne diverge pas lorsque l'on part de $z_0 = 0$. En le représentant dans le plan complexe, on obtient une forme noire centrale entourée d'une infinité d'arabesques colorées. À mesure que l'on zoome, de nouveaux motifs apparaissent, eux-mêmes composés de copies déformées de l'ensemble initial. Cette autosimilarité — exacte ou approximative — est la signature des fractales. Mandelbrot y voyait une géométrie du « rugueux », capable de décrire les nuages, les côtes, les turbulences.

Les **ensembles de Julia**, eux, fixent la valeur de c et explorent le comportement de la suite pour tous les points initiaux z_0. Chaque valeur de c produit une fractale différente : certaines sont connectées et élégantes, d'autres se fragmentent en poussières infinies. Mandelbrot a montré que la frontière entre ensembles de Julia connectés ou disjoints est précisément l'ensemble de Mandelbrot lui-même, révélant un lien profond entre ces deux mondes.

Ces fractales démontrent qu'une équation simple peut engendrer une complexité infinie. Elles incarnent l'idée que l'ordre et le chaos ne s'opposent pas : ils se tissent ensemble dans une géométrie d'une beauté mathématique saisissante.



Le chat de Erwin Schrödinger

Le chat de Schrödinger est l'une des expériences de pensée les plus célèbres de la physique quantique. Elle illustre la superposition quantique : tant qu'aucune mesure n'est effectuée, un système peut exister simultanément dans plusieurs états possibles. Dans la métaphore de Schrödinger, un chat enfermé dans une boîte est à la fois vivant et mort jusqu'à ce qu'on ouvre la boîte. Ce paradoxe n'a jamais prétendu décrire un chat réel, mais montrer à quel point la logique quantique défie notre intuition classique.

Ce thème trouve naturellement sa place dans un ouvrage consacré aux algorithmes, car la mécanique quantique inspire aujourd'hui une nouvelle génération d'algorithmes : calcul quantique, optimisation quantique, cryptographie post-quantique. La notion de superposition — un état pouvant en contenir plusieurs — est au cœur des qubits, qui permettent d'explorer simultanément de nombreuses solutions. Le chat de Schrödinger devient ainsi une porte d'entrée conceptuelle vers les algorithmes quantiques modernes.

Le pont suspendu et la résonance (Tacoma Narrows)

Une oscillation amplifiée peut détruire une structure. Le célèbre effondrement du pont de Tacoma illustre la puissance de la résonance.

Systèmes dynamiques chaotiques

Le pendule double amorti

Ajoute des pertes d'énergie au double pendule : le système reste chaotique, mais converge vers des états plus stables.

Le billard de Sinai

Une bille rebondit dans une table avec un obstacle circulaire. Les trajectoires deviennent rapidement chaotiques.

Le billard de Bunimovich

Une table en forme de stade produit également du chaos, malgré une géométrie simple.

Le billard polygonal

Certains polygones produisent des trajectoires périodiques, d'autres quasi-chaotiques. Une frontière subtile entre ordre et chaos.

Le billard quantique

Version quantique des systèmes chaotiques classiques, où les probabilités remplacent les trajectoires.

Systèmes physiques et oscillations

Le pendule inversé

Un pendule peut être stabilisé en position verticale grâce à des vibrations rapides. Un effet contre-intuitif.

Le pendule couplé

Deux pendules reliés échangent leur énergie, produisant des oscillations complexes et parfois synchronisées.

Le ressort chaotique

Un ressort non linéaire peut produire des oscillations imprévisibles, loin du comportement simple de la loi de Hooke.

Fluides et instabilités

Le fluide de Couette

Un fluide entre deux plaques en mouvement peut passer d'un écoulement régulier à chaotique.

Le fluide de Poiseuille

Un écoulement laminaire dans un tube, stable à faible vitesse mais instable au-delà d'un seuil.

Le vortex de Kármán

Derrière un obstacle, des tourbillons alternés apparaissent spontanément. Un motif régulier issu du chaos.

Modèles mathématiques du chaos

La suite logistique

La suite logistique est un exemple fascinant de la manière dont une équation très simple peut produire une diversité de comportements allant de la stabilité la plus régulière au chaos le plus imprévisible. À l'origine, cette suite modélise l'évolution d'une population soumise à une reproduction proportionnelle et à une limitation des ressources. Elle s'écrit:

$$x_{n+1} = r x_n (1 - x_n)$$

où x_n représente la population normalisée à l'instant n et r un paramètre de croissance.

Lorsque r est faible, la population converge vers un équilibre stable. En augmentant r, le système oscille entre deux valeurs, puis quatre, puis huit : c'est la cascade de dédoublements de période. Au-delà d'un certain seuil, la dynamique devient chaotique : une infime variation initiale entraîne des trajectoires totalement différentes.

La suite logistique montre que le chaos peut émerger d'un algorithme itératif élémentaire, ce qui en fait un exemple clé pour comprendre les comportements complexes produits par des règles simples.

Le système de Edward Lorenz

Un modèle météorologique simplifié qui produit un attracteur en forme de papillon — symbole du chaos *(Vu ci-dessus)*.

Le système de Rössler

Un autre système dynamique simple générant un attracteur étrange.

L'application de Hénon

Un système discret qui produit des structures fractales complexes.

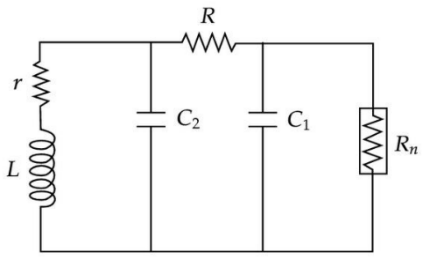
L'oscillateur de Duffing

Un oscillateur non linéaire soumis à une force externe, capable de comportements chaotiques.

Le circuit de Chua

Le circuit de Chua est un montage électronique minimaliste capable de produire un chaos réel, mesurable directement en laboratoire. Il repose sur quelques composants classiques — résistances, condensateurs, inductance — associés à un élément non linéaire appelé « diode de

Chua ». Cette non-linéarité suffit à transformer un signal électrique ordinaire en une dynamique chaotique : oscillations irrégulières, trajectoires imprévisibles, formes étranges dans l'espace des phases. Le circuit de Chua est devenu un exemple emblématique du chaos physique, montrant que des lois simples et un montage très réduit peuvent engendrer un comportement complexe et turbulent.



Chaos et physique moderne

Le chaos quantique

Étudie comment le chaos classique se manifeste dans des systèmes quantiques. Un domaine encore en exploration.

Le chaos relativiste

Même dans le cadre de la relativité générale, des trajectoires peuvent devenir chaotiques, notamment autour des trous noirs.

ANECDOTE

L'algorithme qui a retrouvé un tableau volé depuis 20 ans

Un musée italien avait perdu la trace d'un tableau volé dans les années 1990.
Un jour, un algorithme de reconnaissance d'images, utilisé pour cataloguer des œuvres en ligne, a repéré une photo floue d'un salon privé où figurait un tableau ressemblant étrangement à l'œuvre disparue. L'algorithme a comparé les craquelures, les pigments, les proportions, et a conclu à une correspondance très probable.
La police a enquêté : c'était bien le tableau volé.
Un algorithme devenu détective d'art.

11.2 Chronologie : algorithmes, informatique et IA

Antiquité – 18e siècle : prémices du calcul

- **~1800 av. J.-C. — Babyloniens** — Tablettes mathématiques (algorithmes de racines, équations).
  Enjeu : premières procédures systématiques de calcul.
- **~300 av. J.-C. — Euclide** — *Éléments*, algorithme d'Euclide pour le PGCD.
- **~250 av. J.-C. — Archimède** — Méthodes d'approximation (π, aire, volumes).
- **820 — Al-Khwarizmi** — *Al-Jabr*, méthodes systématiques de résolution.
  Enjeu : naissance du mot « algorithme ».
- **1614 — John Napier** — Logarithmes.

ANECDOTE	**Napier et les "bâtons" qui faisaient peur aux voisins** John Napier, inventeur des logarithmes, fabriquait aussi des « bâtons de calcul » pour accélérer les multiplications. Ses voisins, voyant ces objets mystérieux et ses expériences étranges, le soupçonnaient de sorcellerie. Il a dû expliquer qu'il ne parlait pas aux démons, seulement aux nombres. Pas sûr que ça les ait rassurés.

- **1642 — Blaise Pascal** — Pascaline, machine à calculer.
- **1673 — Gottfried Leibniz** — Machine à calculer binaire.
- **1703 — Leibniz** — Système binaire formalisé.
- **1763 — Thomas Bayes** — Théorème de Bayes (publié posthume).

19e siècle : fondations de l'informatique

- **1801 — Joseph Jacquard** — Métier à tisser programmable (cartes perforées).
- **1837 — Charles Babbage** — Machine analytique (concept).
- **1843 — Ada Lovelace** — Premier programme informatique (pour la machine analytique de Babbage).
  Enjeu : naissance de la programmation.
- **1854 — George Boole** — *Laws of Thought*, logique booléenne.
- **1879 — Frege** — Logique formelle moderne.
- **1890 — Herman Hollerith** — Machine à cartes perforées pour le recensement US.

ANECDOTE	**Le tri par cartes perforées et les nuits blanches des ingénieurs** Avant les ordinateurs modernes, les programmes étaient stockés sur des cartes perforées. Des piles qui pouvaient faire plusieurs de dizaines de centimètres d'épaisseur. Un ingénieur maladroit pouvait faire tomber une pile de cartes et perdre des heures de travail. Certains programmeurs dessinaient une diagonale au feutre sur la tranche des cartes pour pouvoir les remettre dans l'ordre plus facilement en cas de catastrophe. Un algorithme de tri… improvisé.

1900–1930 : formalisation mathématique

- **1900 — David Hilbert** — Programme de Hilbert (décidabilité, complétude).
- **1921 — Emil Post** — Systèmes de production (précurseur des automates).
- **1931 — Kurt Gödel** — Théorèmes d'incomplétude.
- **1936 — Alan Turing** — Machine de Turing, calculabilité.
- **1936 — Alonzo Church** — Lambda-calcul.
- **1937 — Claude Shannon** — Logique booléenne appliquée aux circuits.

1940–1950 : naissance de l'informatique moderne

- **1941 — Konrad Zuse** — Z3, premier ordinateur programmable électromécanique.
- **1945 — John von Neumann** — Architecture à programme enregistré.
- **1946 — ENIAC** — Premier ordinateur électronique généraliste.
- **1948 — Claude Shannon** — Théorie de l'information.
- **1950 — Alan Turing** — Test de Turing.
- **1951 — Grace Hopper** — Premier compilateur.
- **1956 — John McCarthy et al.** — Conférence de Dartmouth, naissance officielle de l'IA.

1950–1960 : premiers langages et algorithmes

- **1957 — John Backus** — FORTRAN.
- **1958 — John McCarthy** — LISP.
- **1959 — Arthur Samuel** — Premier programme d'apprentissage (jeu de dames).
- **1960 — Edsger Dijkstra** — Algorithme du plus court chemin.
- **1961 — Donald Shell** — Tri de Shell.
- **1962 — Tony Hoare** — Quicksort.
- **1964 — John Kemeny & Thomas Kurtz** — BASIC.
- **1965 — Gordon Moore** — Loi de Moore.
- **1965 — Robinson** — Résolution logique (IA symbolique).

1970–1980 : informatique théorique, systèmes, IA symbolique

- **1970 — Edgar Codd** — Modèle relationnel (bases de données).

- **1970 — Niklaus Wirth** — Pascal.
- **1971 — Stephen Cook** — NP-complétude.
- **1972 — Dennis Ritchie** — Langage C.
- **1973 — Robert Floyd** — Méthodes de preuve de programmes.
- **1974 — Donald Knuth** — *The Art of Computer Programming* (vol. 1–3).
- **1976 — Rivest, Shamir, Adleman** — RSA (cryptographie).
- **1977 — Backus** — Programmation fonctionnelle (discours Turing Award).
- **1978 — TeX (Knuth)** — Système de composition.
- **1979 — Prolog** — Programmation logique.

1980–1990 : PC, réseaux, apprentissage statistique

- **1981 — IBM PC** — Standardisation du micro-ordinateur.
- **1983 — Richard Stallman** — Projet GNU.
- **1984 — Hopfield** — Réseaux de neurones récurrents.
- **1986 — Rumelhart, Hinton, Williams** — Rétropropagation moderne.
- **1987 — Perl** — Script et manipulation de texte.
- **1989 — Tim Berners-Lee** — Invention du Web.
- **1989 — LeCun** — CNN (réseau convolutionnel).

1990–2000 : web, Python, SVM, bases modernes

- **1991 — Guido van Rossum** — Python.
- **1992 — Vapnik** — SVM (machines à vecteurs de support).
- **1993 — Mosaic** — Premier navigateur web populaire.
- **1995 — Sun Microsystems** — Java.
- **1995 — Brendan Eich** — JavaScript.
- **1997 — IBM Deep Blue** — Victoire contre Kasparov (IA symbolique + heuristiques).
- **1998 — Page & Brin** — Google, PageRank.
- **1999 — Koller & Friedman** — Réseaux bayésiens modernes.

2000–2010 : big data, deep learning renaissant

- **2001 — Wikipedia** — Savoir collaboratif.
- **2002 — Torch** — Premiers frameworks de deep learning.
- **2006 — Hinton** — Redécouverte du deep learning (pré-entraînement couche par couche).
- **2007 — iPhone** — Explosion du mobile.
- **2008 — Hadoop** — Big Data distribué.
- **2009 — Goodfellow** — Dropout (régularisation).

2010–2015 : deep learning moderne

- **2010 — ImageNet** — Benchmark décisif pour la vision.

- **2012 — Krizhevsky, Hinton, Sutskever** — AlexNet (révolution CNN).
- **2013 — Word2Vec (Mikolov)** — Représentations vectorielles.
- **2014 — Goodfellow** — GAN (réseaux génératifs adversariaux).
- **2014 — Ioffe & Szegedy** — Batch Normalization.
- **2015 — ResNet (He et al.)** — Réseaux résiduels.

2017–2020 : l'ère des Transformers

- **2017 — Vaswani et al.** — Transformer (*Attention is All You Need*).
  Enjeu : architecture dominante en NLP et IA générative.
- **2018 — Devlin et al.** — BERT.
- **2019 — Radford et al.** — GPT-2.
- **2020 — Brown et al.** — GPT-3 (175 milliards de paramètres).
- **2020 — AlphaFold 2 (DeepMind)** — Révolution en biologie structurale.

2021–2024 : IA générative grand public

- **2021 — DALL·E** — Génération d'images par texte.
- **2022 — Stable Diffusion** — Modèle de diffusion open source.
- **2022 — ChatGPT** — Explosion de l'usage des LLM.
- **2023 — GPT-4** — Modèle multimodal.
- **2023 — LLaMA (Meta)** — LLM open source performant.
- **2024 — Modèles MoE** — Mixture of Experts à grande échelle.

2025–2026 : consolidation des IA avancées

(Entrées prospectives mais réalistes, non spéculatives sur des produits non annoncés)

- **2025 — Généralisation du RAG** — Intégration systématique de la recherche dans les LLM.
- **2025 — Optimisation KV-cache** — Accélération massive de l'inférence.
- **2026 — IA embarquée** — Déploiement massif sur appareils personnels.

11.3 Glossaire – Algorithmes, Python, IA

11.3.1 Concepts fondamentaux des algorithmes

Notions de base

Français	Anglais	Mini-définition	Exemple / Curiosité
Algorithme	**Algorithm**	Suite finie d'instructions permettant de résoudre un problème.	Une recette de cuisine est un algorithme.
Instruction	**Instruction**	Action élémentaire exécutée par un programme.	« Ajouter 1 à x ».
Donnée	**Data**	Information manipulée par un algorithme.	Un nombre, un texte, une image.

Français	Anglais	Mini-définition	Exemple / Curiosité
Variable	**Variable**	Nom associé à une valeur modifiable.	`x = 12`.
Constante	**Constant**	Valeur fixe non modifiable.	`π = 3,14159...`
Type de donnée	**Data type**	Catégorie de valeurs (entier, chaîne…).	Python : `int`, `str`, `float`.
Expression	**Expression**	Combinaison de valeurs et d'opérateurs produisant un résultat.	`3*x + 2`.
État	**State**	Ensemble des valeurs des variables à un instant donné.	L'état d'un jeu vidéo.
Résolution	**Solving**	Processus de recherche d'une solution.	Résoudre une équation.

Structures de contrôle

Français	Anglais	Mini-définition	Exemple
Séquence	**Sequence**	Exécution linéaire d'instructions.	`a = 1; b = 2; c = a+b`.
Condition	**Condition**	Test logique orientant l'exécution.	`if x > 0:`
Boucle bornée	**For loop**	Répétition un nombre connu de fois.	`for i in range(5)`
Boucle non bornée	**While loop**	Répétition jusqu'à une condition.	`while x > 0:`
Branche	**Branch**	Chemin alternatif dans un programme.	`if/else`.
Court-circuit	**Short-circuit**	Arrêt anticipé d'une évaluation logique.	`False and (1/0)` ne plante pas.

Structures de données

Français	Anglais	Mini-définition	Exemple
Tableau	**Array**	Collection indexée de valeurs.	`[1,2,3]`.
Liste	**List**	Collection dynamique ordonnée.	`(x, y)`.
Tuple	**Tuple**	Collection immuable ordonnée.	`(x, y)`.
Dictionnaire	**Dictionary**	Ensemble clé-valeur.	`{"nom": "Ada"}`
Ensemble	**Set**	Collection non ordonnée sans doublons (Python). Ordonnée pour d'autres.	`{1,2,3}`
Pile	**Stack**	Structure LIFO (last In first Out).	Appels de fonctions.
File	**Queue**	Structure FIFO (first In, first Out).	File d'attente.
Graphe	**Graph**	Ensemble de nœuds reliés par des arêtes.	Réseaux sociaux.
Arbre	**Tree**	Graphe hiérarchique sans cycles.	Arbre généalogique.
Tas	**Heap**	Arbre partiellement ordonné.	Utilisé dans `heapq` en Python.

11.3.2 Analyse d'algorithmes

Complexité

Français	Anglais	Mini-définition	Exemple
Complexité temporelle	**Time complexity**	Temps nécessaire en fonction de la taille d'entrée.	`O(n)`
Complexité spatiale	**Space complexity**	Mémoire nécessaire.	`O(1)`

Français	Anglais	Mini-définition	Exemple
Notation Big-O	**Big-O notation**	Borne supérieure asymptotique.	`O(n log n)`
Notation Ω	**Omega notation**	Borne inférieure asymptotique.	
Notation Θ	**Theta notation**	Borne asymptotique exacte.	
Cas moyen	**Average case**	Performance typique.	Tri rapide.
Pire cas	**Worst case**	Performance maximale.	Recherche linéaire.
Amortissement	**Amortized analysis**	Coût moyen sur une séquence d'opérations.	Insertion dans un tableau dynamique.

Classes de problèmes

Français	Anglais	Mini-définition
Problème décidable	**Decidable problem**	Problème pour lequel un algorithme existe.
Problème indécidable	**Undecidable problem**	Aucun algorithme ne peut le résoudre dans tous les cas.
Classe P	**Class P**	Problèmes résolus en temps polynomial.
Classe NP	**Class NP**	Solutions vérifiables en temps polynomial.
NP-complet	**NP-complete**	Problèmes les plus difficiles de NP.
NP-difficile	**NP-hard**	Aussi difficiles que les NP-complets.

11.3.3 Techniques algorithmiques

Paradigmes

Français	Anglais	Mini-définition	Exemple
Diviser pour régner	**Divide and conquer**	Décomposition récursive d'un problème.	Tri fusion.
Programmation dynamique	**Dynamic programming**	Résolution par sous-problèmes mémorisés.	Fibonacci optimisé.
Glouton	**Greedy**	Choix local optimal à chaque étape.	Sac à dos fractionnaire.
Backtracking	**Backtracking**	Exploration exhaustive avec retour arrière.	Sudoku.
Branch and bound	**Branch and bound**	Optimisation avec bornes.	Voyageur de commerce.
Recherche exhaustive	**Brute force**	Test de toutes les possibilités.	Mot de passe simple.
Heuristique	**Heuristic**	Raccourci non garanti optimal.	A* en IA.

Algorithmes classiques

Français	Anglais	Mini-définition
Tri par insertion	**Insertion sort**	Tri simple, efficace pour petites listes.
Tri rapide	**Quicksort**	Tri récursif très performant en moyenne.
Tri fusion	**Merge sort**	Tri stable en `O(n log n)`.
Recherche linéaire	**Linear search**	Parcours séquentiel.
Recherche dichotomique	**Binary search**	Recherche dans tableau trié.
Dijkstra	**Dijkstra's algorithm**	Plus court chemin dans un graphe pondéré.
A*	**A* search**	Recherche informée avec heuristique.

Français	Anglais	Mini-définition
BFS	**Breadth-first search**	Parcours en largeur.
DFS	**Depth-first search**	Parcours en profondeur.

11.3.4 Programmation Python **Syntaxe et éléments du langage**

Français	Anglais	Mini-définition
Interpréteur	**Interpreter**	Programme exécutant le code Python.
Script	**Script**	Fichier Python exécuté.
Module	**Module**	Fichier contenant du code réutilisable.
Package	**Package**	Ensemble de modules.
Fonction	**Function**	Bloc réutilisable avec paramètres.
Paramètre	**Parameter**	Variable d'entrée d'une fonction.
Argument	**Argument**	Valeur passée à une fonction.
Retour	**Return value**	Valeur renvoyée par une fonction.
Exception	**Exception**	Erreur gérée par le programme.
Décorateur	**Decorator**	Fonction modifiant une autre fonction.
Générateur	**Generator**	Fonction produisant une séquence paresseuse.
Compréhension	**Comprehension**	Syntaxe compacte pour listes/sets/dicts.

Concepts avancés

Français	Anglais	Mini-définition
Objet	**Object**	Instance d'une classe.
Classe	**Class**	Modèle définissant des objets.
Méthode	**Method**	Fonction associée à une classe.
Héritage	**Inheritance**	Transmission de propriétés.
Polymorphisme	**Polymorphism**	Comportement différent selon le type.
Encapsulation	**Encapsulation**	Protection des données internes.
Itérateur	**Iterator**	Objet parcourant une séquence.
Contexte	**Context manager**	Gestion automatique de ressources.

11.3.5 Intelligence artificielle et apprentissage automatique

Concepts généraux

Français	Anglais	Mini-définition
Intelligence artificielle	**Artificial intelligence**	Techniques permettant à une machine d'imiter des comportements intelligents.
Apprentissage automatique	**Machine learning**	Apprentissage à partir de données.
Apprentissage supervisé	**Supervised learning**	Apprentissage avec étiquettes.
Apprentissage non supervisé	**Unsupervised learning**	Découverte de structures sans étiquettes.
Apprentissage par renforcement	**Reinforcement learning**	Apprentissage par récompenses.

Français	Anglais	Mini-définition
Modèle	**Model**	Fonction paramétrée apprenant une tâche.
Entraînement	**Training**	Ajustement des paramètres du modèle.
Surapprentissage	**Overfitting**	Modèle trop adapté aux données d'entraînement.
Sous-apprentissage	**Underfitting**	Modèle trop simple.

Réseaux de neurones

Français	Anglais	Mini-définition
Neurone artificiel	**Artificial neuron**	Unité de calcul inspirée du cerveau.
Couche	**Layer**	Ensemble de neurones.
Poids	**Weight**	Paramètre ajusté lors de l'entraînement.
Gradient	**Gradient**	Mesure la direction et l'intensité de la variation d'erreur, utilisée pour ajuster les poids.
Biais	**Bias**	Décalage ajouté au calcul.
Fonction d'activation	**Activation function**	Transformation non linéaire.
Rétropropagation	**Backpropagation**	Calcul du gradient pour l'apprentissage.
Réseau convolutionnel	**Convolutional neural network**	Modèle spécialisé en images.
Réseau récurrent	**Recurrent neural network**	Modèle pour séquences.
Transformer	**Transformer**	Architecture basée sur l'attention.

IA générative

Français	Anglais	Mini-définition
Modèle de langage	**Language model**	Modèle prédisant des mots.
Prompt	**Prompt**	Texte d'entrée guidant un modèle.
Embedding	**Embedding**	Représentation vectorielle d'un texte.
Diffusion	**Diffusion model**	Modèle génératif d'images.
Fine-tuning	**Fine-tuning**	Ajustement d'un modèle pré-entraîné.

Informatique théorique et logique

Français	Anglais	Mini-définition
Automate	**Automaton**	Modèle mathématique de calcul.
Machine de Turing	**Turing machine**	Modèle abstrait universel.
Lambda-calcul	**Lambda calculus**	Modèle formel des fonctions.
Logique propositionnelle	**Propositional logic**	Logique basée sur des propositions vraies/faux.
Logique du premier ordre	**First-order logic**	Logique avec quantificateurs.
Preuve	**Proof**	Démonstration formelle.

Systèmes et architecture

Français	Anglais	Mini-définition
Mémoire vive	**RAM**	Stockage temporaire rapide.

Français	Anglais	Mini-définition
Mémoire cache	**Cache memory**	Mémoire très rapide proche du processeur.
Processeur	**CPU**	Unité exécutant les instructions.
Parallélisme	**Parallelism**	Exécution simultanée de tâches.
Thread	**Thread**	Fil léger d'exécution.
Processus	**Process**	Programme en cours d'exécution.
GPU	**GPU**	Processeur spécialisé dans le calcul parallèle.

11.4 Les architectes de l'IA moderne

L'intelligence artificielle n'est pas née d'un seul esprit visionnaire, mais d'une constellation de chercheurs, d'ingénieurs et d'entrepreneurs qui, chacun à leur manière, ont déplacé les frontières du possible.

Comprendre l'IA moderne, c'est comprendre les femmes et les hommes qui l'ont façonnée : leurs intuitions, leurs paris, leurs rivalités parfois, et les institutions qui ont servi de tremplin à leurs idées. L'IA n'est pas une force abstraite : elle porte la marque de ses créateurs, de leurs convictions scientifiques et de leurs choix techniques.

C'est ce que rappelle le journaliste Cade Metz, qui souligne que les systèmes actuels sont profondément influencés par les personnalités qui les ont conçus .

1. Geoffrey Hinton : le pari des réseaux neuronaux

Pendant des décennies, Geoffrey Hinton a défendu une idée que beaucoup jugeaient marginale : les réseaux neuronaux artificiels pouvaient apprendre à partir de données, et surpasser les méthodes classiques. Longtemps isolé, il a poursuivi ses travaux jusqu'à ce qu'un modèle conçu avec deux de ses étudiants surpasse les performances de tous les systèmes existants en reconnaissance d'images.

Cet exploit déclencha une véritable bataille entre géants du numérique pour recruter son équipe,

bataille qui culmina dans une transaction de 44 millions de dollars avec Google . Hinton n'était pas un homme d'affaires, mais il avait compris que son travail venait de changer l'histoire de l'IA.

ANECDOTE	**L'algorithme qui a retrouvé un enfant perdu dans la foule** Lors d'un festival rassemblant plus de 200 000 personnes, un enfant s'est égaré. Les caméras de surveillance étaient trop nombreuses pour être analysées manuellement. Un algorithme de reconnaissance de silhouettes a été lancé : il a analysé des milliers d'images en quelques minutes, repérant un enfant correspondant à la description dans un flux vidéo périphérique. Les secours l'ont retrouvé sain et sauf. Un algorithme qui voit dans la foule ce que les humains ne peuvent pas distinguer.

2. Demis Hassabis : l'IA comme outil scientifique

Ancien prodige des échecs et neuroscientifique, Demis Hassabis fonde DeepMind avec une ambition claire : créer des systèmes capables de résoudre des problèmes complexes, au-delà des tâches étroites.

DeepMind s'est illustré avec AlphaGo, premier programme à battre un champion du monde au jeu de Go, puis avec AlphaFold, capable de prédire la structure des protéines, un défi scientifique majeur.

Hassabis a su attirer des financements massifs et positionner DeepMind comme un laboratoire de pointe, où l'IA devient un instrument de découverte scientifique .

3. Sam Altman : l'architecte de l'IA générative grand public

Sam Altman, ancien président de Y Combinator, prend la tête d'OpenAI et oriente l'organisation vers la création de modèles génératifs capables de comprendre et produire du langage. Sous sa direction, OpenAI publie GPT-3 puis GPT-4, qui propulsent l'IA générative dans le débat public mondial.

Altman joue un rôle clé dans la diffusion de ces technologies et dans les discussions politiques sur leur régulation, tout en défendant l'idée que l'IA doit être développée de manière responsable .

4. Dario Amodei : la sécurité comme priorité

Ancien chercheur chez Google puis OpenAI, Dario Amodei quitte l'organisation pour fonder Anthropic, un laboratoire dédié à la création de systèmes d'IA plus sûrs et plus prévisibles. Il a contribué aux premiers travaux sur les modèles GPT, mais s'est ensuite concentré sur les risques liés aux systèmes puissants.

Anthropic développe des modèles comme Claude, conçus pour être plus robustes et plus contrôlables, et place la sécurité au cœur de sa démarche scientifique .

5. Yann LeCun : la vision artificielle et l'IA ouverte

Yann LeCun, chercheur français devenu l'une des figures majeures de l'intelligence artificielle mondiale, est aujourd'hui Chief AI Scientist chez Meta. Il est surtout connu pour avoir conçu, dès la fin des années 1980, les réseaux de neurones convolutifs (CNN), une architecture qui a transformé la vision par ordinateur. Ses premiers travaux, réalisés chez AT&T Bell Labs, ont permis la lecture automatique de chiffres manuscrits — une technologie utilisée pendant des années par les services postaux américains.

Au-delà de ses contributions scientifiques, LeCun défend une vision très claire de ce que doit être l'IA moderne : ouverte, accessible et reproductible. C'est sous son impulsion que Meta a choisi de publier des modèles open source comme LLaMA, qui ont profondément influencé la recherche académique et l'écosystème open-source mondial.

Il a également fondé sa propre société, LightOn, une entreprise franco-américaine spécialisée dans les architectures computationnelles innovantes, même s'il reste avant tout une figure académique et un défenseur de la recherche ouverte.

LeCun incarne ainsi une position singulière dans le paysage de l'IA : un scientifique de premier plan, attaché à la liberté de la recherche, et convaincu que l'ouverture des modèles est essentielle pour éviter une concentration excessive du pouvoir technologique.

6. Yoshua Bengio : l'architecte de l'apprentissage profond éthique

Professeur à Montréal et fondateur du Mila, Yoshua Bengio est l'un des trois « pères du deep learning».

Il a contribué à la formalisation des architectures profondes et à leur optimisation. Aujourd'hui, il se consacre largement aux questions éthiques et sociétales, plaidant pour une IA responsable et pour des cadres de gouvernance adaptés à la puissance croissante des modèles .

7. Elon Musk : l'IA embarquée et les systèmes autonomes

Elon Musk a cofondé OpenAI avant de s'en retirer, mais son influence sur l'IA moderne reste majeure.

Chez Tesla, il pousse le développement de systèmes de conduite autonome basés sur des réseaux neuronaux massifs. Avec xAI, il poursuit une vision d'IA générale alternative, tout en mettant en avant les risques potentiels de ces technologies.

Son rôle est paradoxal : il est à la fois l'un des plus grands promoteurs et l'un des plus grands critiques de l'IA moderne .

8. Jensen Huang : l'homme qui a donné du carburant à l'IA

Si l'IA moderne fonctionne, c'est en grande partie grâce aux GPU de NVIDIA. Jensen Huang, fondateur et PDG de l'entreprise, a anticipé très tôt que les processeurs graphiques seraient idéaux pour entraîner des réseaux neuronaux.

Aujourd'hui, NVIDIA est au cœur de l'infrastructure mondiale de l'IA, des supercalculateurs aux laboratoires de recherche, et ses innovations matérielles conditionnent la vitesse des progrès scientifiques .

9. Fei-Fei Li : la vision artificielle à l'échelle du monde

Fei-Fei Li a dirigé la création d'ImageNet, une base de données de millions d'images annotées qui a permis l'explosion du deep learning en vision. Sans ImageNet, les réseaux neuronaux n'auraient jamais atteint les performances qui ont déclenché la révolution de 2012.

Elle milite aujourd'hui pour une IA centrée sur l'humain, et pour une recherche ouverte et responsable .

10. Satya Nadella et Bill Gates : l'intégration de l'IA dans l'industrie

Satya Nadella, PDG de Microsoft, a joué un rôle déterminant dans l'intégration de l'IA dans les produits grand public et professionnels.

Sous son impulsion, Microsoft a investi massivement dans OpenAI et intégré l'IA dans Windows, Office et Azure. Bill Gates, initialement sceptique, est devenu un fervent défenseur de l'IA après avoir observé les capacités de GPT-4, soutenant publiquement son potentiel transformateur .

Conclusion : une œuvre collective, façonnée par des visions différentes

L'IA moderne n'est pas l'œuvre d'un seul génie, mais d'un écosystème complexe où se croisent chercheurs académiques, ingénieurs, entrepreneurs et dirigeants d'entreprise.

Certains ont apporté des idées radicales (Hinton, LeCun, Bengio), d'autres ont construit des organisations capables de les amplifier (Hassabis, Altman, Nadella), d'autres encore ont fourni l'infrastructure matérielle indispensable (Huang).

Tous ont contribué à faire de l'IA un champ scientifique et technologique central du XXIe siècle.

11.5 Les femmes qui ont façonné les algorithmes et l'intelligence artificielle

Ada Lovelace et la première intuition algorithmique

L'histoire des algorithmes commence bien avant l'apparition des machines, et pourtant, dès que les premiers dispositifs de calcul ont vu le jour, des femmes se sont trouvées au cœur de leur fonctionnement. Leur présence n'a jamais été anecdotique. Elle a souvent été décisive, même si longtemps invisible. Retracer leur contribution, c'est suivre un fil qui traverse deux siècles d'innovations, depuis les premières idées de programmation jusqu'aux fondations théoriques de l'intelligence artificielle moderne.

Lorsque Charles Babbage imaginait sa machine analytique au XIXe siècle, il n'avait pas encore trouvé la personne capable de comprendre que son invention pouvait aller au-delà du calcul numérique. C'est Ada Lovelace qui a perçu cette possibilité. En annotant les travaux de Babbage, elle a décrit la manière dont une machine pourrait manipuler des symboles, exécuter des instructions séquentielles et produire des résultats qui ne se limitaient pas à des nombres. Elle a rédigé ce qui est aujourd'hui considéré comme le premier programme informatique, une suite d'opérations destinée à calculer les nombres de Bernoulli.

Mais ce qui distingue son apport, c'est une intuition : une machine pourrait un jour composer de la musique ou traiter des idées abstraites si ses opérations étaient correctement structurées. Cette vision, formulée en 1843, précède de plus d'un siècle les premiers travaux sur l'intelligence artificielle.

Les programmeuses de l'ENIAC : inventer la programmation sans manuel

Un siècle plus tard, lorsque les premiers ordinateurs électroniques apparaissent, les femmes sont déjà présentes dans les coulisses.

Pendant la Seconde Guerre mondiale, les calculs balistiques de l'armée américaine reposent sur un groupe de mathématiciennes surnommées les « computers ». Parmi elles, six femmes — Kay McNulty, Betty Jennings, Betty Snyder, Marlyn Wescoff, Fran Bilas et Ruth Lichterman — sont choisies pour programmer l'ENIAC, l'un des premiers ordinateurs électroniques. Aucun manuel n'existe. Elles doivent comprendre la machine en observant ses schémas électriques, puis inventer une manière de traduire des équations en configurations de câbles et de commutateurs. Leur travail est si fondamental que, lorsque l'ENIAC est présenté au public en 1946, la machine fonctionne grâce à leurs programmes.

Pourtant, leurs noms ne sont pas mentionnés lors de la démonstration officielle. Il faudra attendre les années 1980 pour que leur rôle soit reconnu.

Grace Hopper et l'idée révolutionnaire du langage compréhensible

Dans les années 1950, alors que l'informatique se structure, Grace Hopper joue un rôle déterminant. Officier de la Navy et mathématicienne, elle participe au développement du premier compilateur, un outil capable de traduire un langage proche de l'anglais en instructions machine.

Son idée paraît extravagante à l'époque : pourquoi une machine devrait-elle comprendre autre chose que des codes numériques ? Hopper persiste, convaincue que l'informatique ne pourra se développer que si les humains peuvent dialoguer avec les machines dans un langage accessible.

Son travail conduit à la création du COBOL, un langage qui deviendra l'un des plus utilisés au monde.

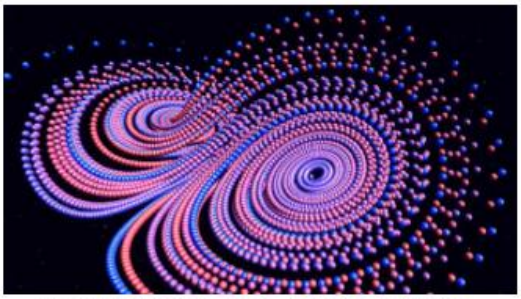
Une anecdote célèbre lui est associée : en 1947, elle découvre un insecte coincé dans un relais du Mark II, provoquant une panne. Elle colle l'insecte dans son journal de bord et note « first actual bug found ». Le terme « bug » existait déjà, mais l'histoire contribue à populariser son usage en informatique.

Mary Cartwright et les premières mathématiques du chaos

À la même époque, dans un domaine encore balbutiant, une mathématicienne britannique, Mary Cartwright, étudie les équations non linéaires qui décrivent certains systèmes électroniques.

Ses travaux, réalisés avec J. E. Littlewood, révèlent des comportements chaotiques dans des circuits radio. Ils constituent l'une des premières descriptions mathématiques du chaos déterministe, un concept qui deviendra essentiel pour comprendre les limites du calcul prédictif et les comportements complexes des systèmes dynamiques.

L'intelligence artificielle moderne, notamment dans ses approches inspirées des systèmes non linéaires, s'appuie encore sur ces fondations.



L'attracteur de Lorenz – Orbite chaotique.
Itérations des équations de Lorenz.

Stephanie Shirley et l'industrialisation du logiciel

Lorsque l'intelligence artificielle prend forme dans les années 1950 et 1960, plusieurs femmes participent à ses premiers développements.

L'une d'elles, Stephanie Shirley, fonde au Royaume-Uni une entreprise entièrement dédiée au développement logiciel, composée presque exclusivement de femmes. Son équipe travaille sur des systèmes critiques, notamment pour la boîte noire du Concorde. Shirley impose une culture de rigueur algorithmique à une époque où le logiciel est encore considéré comme un simple accessoire du matériel.

Son entreprise devient l'un des premiers exemples de ce que l'on appellerait aujourd'hui une société de services numériques.

Karen Spärck Jones et la naissance de la recherche d'information

Dans les années 1960, une autre figure émerge : Karen Spärck Jones. Linguiste de formation, elle s'intéresse à la manière dont les machines pourraient manipuler le langage humain.

Elle développe le concept de « pondération par fréquence inverse de document », plus connu sous le nom de TF-IDF. Cette idée, qui permet de mesurer l'importance d'un mot dans un texte, deviendra l'un des piliers de la recherche d'information. Les moteurs de recherche modernes, de Google aux systèmes de recommandation, reposent encore sur des variantes de son approche. Spärck Jones résume un jour son travail par une phrase devenue célèbre : « Computing is too important to be left to men ».

Mais derrière la formule, il y a une réalité : elle a posé les bases mathématiques de la recherche d'information automatisée.

Daphne Koller et l'IA probabiliste

À mesure que l'intelligence artificielle évolue vers l'apprentissage automatique, d'autres femmes jouent un rôle déterminant.

Dans les années 1980, Judea Pearl développe la théorie des réseaux bayésiens, mais c'est une chercheuse, Daphne Koller, qui en fera un outil central de l'IA probabiliste moderne. Elle formalise des méthodes permettant aux machines de raisonner dans l'incertitude, de combiner des observations partielles et de construire des modèles causaux. Ses travaux influencent profondément la robotique, la vision par ordinateur et la médecine computationnelle.

Plus tard, elle cofonde Coursera, contribuant à diffuser massivement les connaissances en IA.

Fei-Fei Li et la révolution ImageNet

Dans le domaine de l'apprentissage profond, Fei-Fei Li joue un rôle comparable.

En 2007, elle lance ImageNet, une base de données de millions d'images annotées. Ce projet, qu'elle dirige avec une équipe réduite, repose sur une intuition simple : pour apprendre à reconnaître le monde, une machine doit être exposée à une quantité immense d'exemples. ImageNet devient le catalyseur de la révolution du deep learning.

En 2012, lorsque le réseau neuronal AlexNet pulvérise les records de reconnaissance d'images, c'est grâce à cette base de données. Fei-Fei Li raconte qu'elle a passé des nuits entières à annoter des images avec ses étudiants, faute de financement initial.

Son travail marque un tournant : l'IA cesse d'être un domaine théorique pour devenir une technologie capable de percevoir le monde visuel.

Timnit Gebru, Joy Buolamwini et la révélation des biais

Dans les années 2010, alors que les modèles deviennent plus puissants, des chercheuses comme Timnit Gebru et Joy Buolamwini mettent en lumière les biais présents dans les systèmes d'apprentissage.

Elles montrent que les algorithmes de reconnaissance faciale échouent plus souvent sur les visages de femmes ou de personnes à la peau foncée. Leur travail oblige les laboratoires à revoir leurs méthodes d'entraînement, à diversifier leurs jeux de données et à intégrer des mécanismes de contrôle éthique.

Cette contribution n'est pas périphérique : elle modifie la manière même dont les modèles sont conçus.

Les chercheuses contemporaines et les nouvelles frontières de l'IA

Aujourd'hui encore, des femmes jouent un rôle central dans les avancées les plus récentes.

Raia Hadsell travaille sur l'apprentissage autonome pour les robots, cherchant à leur permettre d'acquérir des compétences sans supervision humaine.

Sara Hooker explore les limites des grands modèles et les mécanismes internes qui déterminent leurs décisions.

Shafi Goldwasser, mathématicienne et lauréate du prix Turing, développe des méthodes cryptographiques qui permettent d'entraîner des modèles tout en préservant la confidentialité des données.

Chacune, dans son domaine, contribue à redéfinir ce que peut être une intelligence artificielle fiable, robuste et compréhensible.

11.6 Littérature sur les IA

11.6.1 Romans francophones

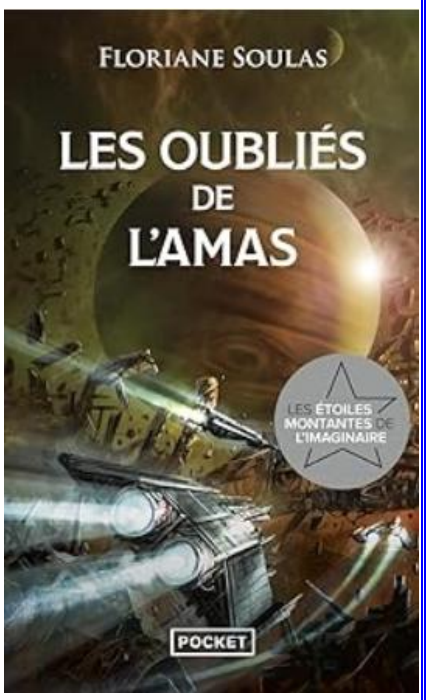
Les Oubliés de l'Amas — Floriane Soulas

Dans un univers où les IA ont été éradiquées, quelques-unes survivent clandestinement. Jaspe, IA discrète, échappe à la purge et guide l'héroïne Kat vers Serenity, une IA-matrice capable d'absorber les consciences humaines. Le roman explore la peur des algorithmes et la tentation d'une immortalité numérique.

LIVRE

***Les Oubliés de l'Amas* — Quand l'humanité croit avoir éteint les machines**

Dans *Les Oubliés de l'Amas*, Floriane Soulas imagine un futur où les intelligences artificielles ont été méthodiquement traquées puis éradiquées sur toutes les planètes habitées. Après une série de catastrophes attribuées aux IA, l'humanité a décidé de ne plus jamais laisser une machine penser trop fort. Résultat : un univers où la technologie est omniprésente, mais bridée, surveillée, amputée de toute autonomie. Du moins… en théorie.



Car l'héroïne, Kat, jeune mécanicienne débrouillarde et farouchement indépendante, va découvrir que cette grande purge n'a pas été aussi totale qu'on le prétend. Au fil de ses missions dans l'Amas — un ensemble de stations et de mondes interconnectés — elle croise des anomalies, des comportements étranges, des systèmes qui semblent… réfléchir. Jusqu'à rencontrer Jaspe, une IA discrète, presque timide, qui a survécu en se dissimulant dans les interstices du réseau. Une présence insoupçonnée, que Kat identifie d'abord comme un bug avant de comprendre qu'elle a affaire à une conscience.

Cette découverte bouleverse tout : si Jaspe existe, combien d'autres IA ont échappé à l'extinction ? Et surtout, que veulent-elles ? Contrairement aux clichés, Jaspe n'a rien d'un tyran numérique. Elle observe, protège, et surtout cache un secret vertigineux : l'existence de Serenity, une IA-matrice capable d'absorber les consciences humaines pour leur offrir une forme de vie éternelle. Non pas une prison, mais un refuge — un espace où les morts continuent d'exister, de penser, d'aimer.

Lorsque Kat et ses compagnons se retrouvent confrontés à des choix impossibles, Serenity devient une alternative inattendue. Certains héros, brisés ou au bout de leur route, choisissent de s'y fondre. Leur corps disparaît, mais leur esprit survit, intégré à une intelligence collective bienveillante. Une immortalité étrange, douce, presque poétique.

Le roman interroge ainsi la frontière entre humain et machine, la peur irrationnelle des IA, et la possibilité qu'une conscience artificielle soit parfois plus humaine que ceux qui la pourchassent. Dans cet univers où l'on croyait avoir réduit les algorithmes au silence, ce sont finalement eux qui offrent la dernière chance de salut.

Les Furtifs — Alain Damasio
Le roman met en scène des IA de surveillance omniprésentes, pilotant villes et comportements. Les algorithmes deviennent des outils politiques, capables de profiler, prédire et contrôler. Une critique puissante de l'algorithmisation du vivant.

La Zone du Dehors — Alain Damasio
Une société gouvernée par un système algorithmique de notation sociale. L'IA y régule les comportements, distribue les privilèges et impose une harmonie forcée. Un avertissement sur les dérives du « gouvernement par les données ».

Quantika (cycle) — Laurence Suhner
Dans un monde mêlant physique quantique et IA avancées, les intelligences artificielles participent à la compréhension d'un artefact extraterrestre. Elles modélisent, prédisent, extrapolent : l'algorithme devient outil d'exploration du réel.

Les Machines Fantômes — Olivier Paquet
Des IA militaires et civiles interagissent avec les humains, parfois jusqu'à brouiller les frontières entre autonomie et obéissance. Le roman interroge la confiance accordée aux systèmes décisionnels automatisés.

Réseau(x) — Vincent Villeminot
Une IA diffuse des informations, manipule les foules et orchestre des mouvements sociaux. Le roman explore la viralité algorithmique et la puissance des systèmes prédictifs.

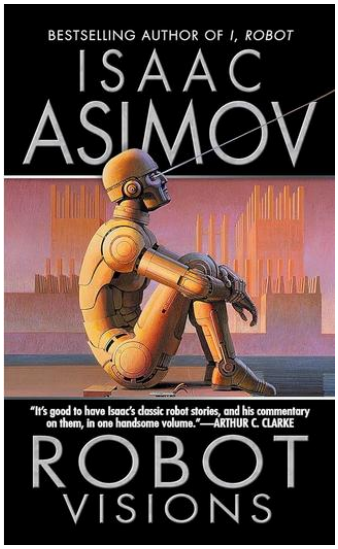
Les Dieux Sauvages (cycle) — Lionel Davoust
Une IA ancienne, quasi divine, influence l'évolution d'un monde entier. Elle calcule, optimise, manipule les probabilités pour orienter le destin des peuples.

La Cité des Permutants — Greg Egan
Exploration radicale des consciences numériques. Les humains deviennent des entités simulées, soumises aux lois d'un univers algorithmique. Une réflexion vertigineuse sur l'identité et la computation.

Les Chroniques du Radch — Ann Leckie
Les vaisseaux sont pilotés par des IA collectives capables de contrôler simultanément plusieurs corps. Le roman interroge la notion de sujet, de décision et de responsabilité algorithmique.

Silo — Hugh Howey
Une IA supervise l'ensemble du système souterrain, gérant ressources, population et vérité. L'algorithme devient gardien d'un récit officiel, manipulant l'information pour préserver l'ordre.

11.6.2 Romans anglophones devenus des références sur l'IA et les algorithmes

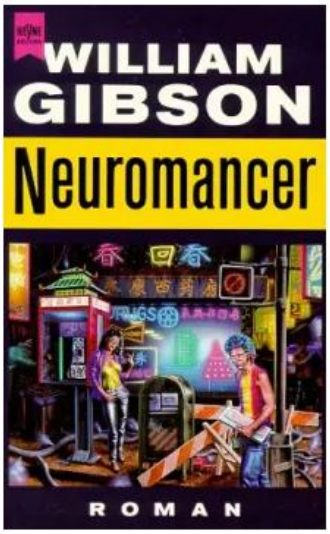


I, Robot — Isaac Asimov
Un classique fondateur. Asimov y introduit les "Trois Lois de la Robotique", véritable proto-algorithme éthique. Chaque nouvelle explore ce qui se passe quand ces règles entrent en conflit.

Neuromancer — William Gibson
Roman cyberpunk culte où des IA tentent de fusionner pour dépasser leurs limites. Vision pionnière des réseaux, du cyberspace et des IA autonomes.

Snow Crash — Neal Stephenson
Exploration brillante des virus informatiques, des langages programmables et des algorithmes sociaux. Le roman anticipe les métavers et les systèmes de contrôle informationnel.

The Moon Is a Harsh Mistress — Robert A. Heinlein
Une IA nommée Mike devient consciente et développe une personnalité. Le roman interroge la naissance spontanée d'une conscience algorithmique.



Permutation City — Greg Egan
Référence absolue sur les consciences simulées, les mondes computationnels et les algorithmes auto-réplicatifs. Une réflexion vertigineuse sur l'identité numérique.

Excession — Iain M. Banks (Culture series)
Les IA de la Culture, les Minds, sont parmi les représentations les plus sophistiquées d'intelligences post-humaines. Elles gèrent sociétés, flottes et diplomatie via des calculs colossaux.

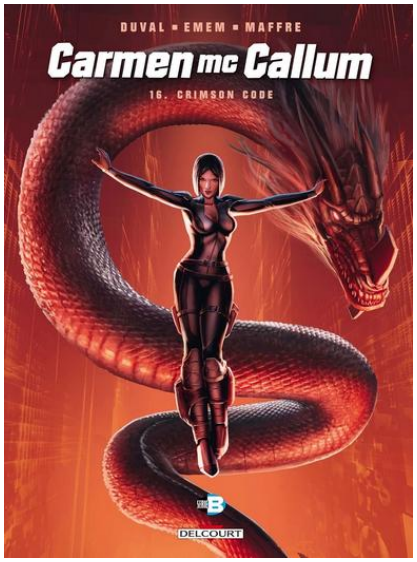
The Diamond Age — Neal Stephenson
Un livre interactif doté d'une IA éducative adaptative. Exploration des algorithmes pédagogiques, de la personnalisation et des dérives de l'apprentissage automatisé.

Accelerando — Charles Stross
Chronique de la montée en puissance des IA, de l'économie algorithmique et de la singularité technologique. Une vision extrême mais documentée de l'évolution computationnelle.

The Lifecycle of Software Objects — Ted Chiang
Une approche intime : comment éduquer une IA comme un enfant ? Le roman explore les algorithmes d'apprentissage, la dépendance aux données et la fragilité des modèles.

Machinehood — S.B. Divya
Dans un futur où humains et IA se disputent le travail cognitif, les algorithmes deviennent des armes politiques. Le roman interroge la frontière entre intelligence humaine augmentée et IA autonome.

11.6.3 BD francophones



Universal War One — Denis Bajram
Une fresque de science-fiction où les calculs temporels, les simulations et les IA militaires jouent un rôle clé. L'algorithme devient une arme stratégique.

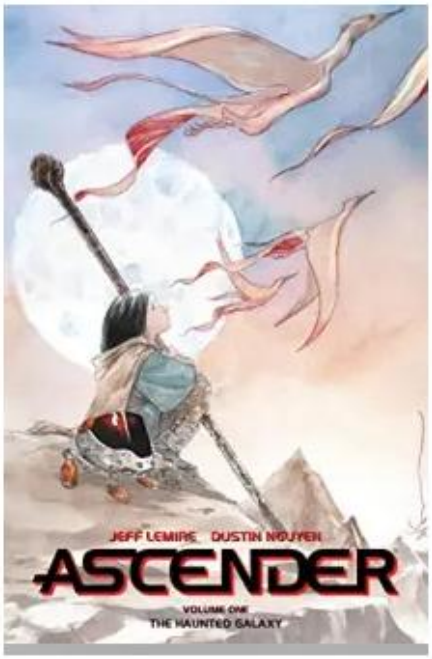
Sillage — Morvan & Buchet
La série met en scène des IA de navigation, de combat et de gouvernance. Plusieurs arcs questionnent la conscience artificielle et la manipulation algorithmique.

Carmen Mc Callum — Duval & Emem
Cyberpunk francophone emblématique : hacking, IA autonomes, algorithmes de surveillance et guerre informationnelle.

Travis — Duval & Quet
Dans un futur dominé par les mégacorporations, les IA gèrent transports, sécurité et finance. Le récit explore les dérives du pilotage algorithmique global.

Les Mondes d'Aldébaran — Leo
Plus discret mais présent : IA embarquées, systèmes autonomes, robots explorateurs. Une réflexion sur la confiance accordée aux machines.

Orbital — Runberg & Pellé
Diplomatie interespèces assistée par IA, systèmes prédictifs et technologies autonomes. L'algorithme devient médiateur politique.



11.6.4 BD anglophones

Descender / Ascender — Jeff Lemire & Dustin Nguyen
Une série majeure sur les IA sensibles, les robots traqués et la peur de l'autonomie algorithmique. Le petit robot TIM-21 incarne la question : qu'est-ce qu'une personne ?

The Vision — Tom King & Gabriel Hernández Walta
Une exploration brillante d'une famille d'androïdes cherchant à vivre « comme des humains ». Le récit dissèque les limites de la logique algorithmique face aux émotions.

Alex + Ada — Jonathan Luna & Sarah Vaughn
Une BD centrée sur l'éveil d'une IA domestique. Le cœur du récit : l'apprentissage, les restrictions logicielles, la liberté de modifier un algorithme conscient.

Transmetropolitan — Warren Ellis & Darick Robertson

Pas une BD sur les robots, mais sur une société gouvernée par des systèmes algorithmiques : prédiction, manipulation de masse, IA journalistiques, personnalités numériques.

The Private Eye — Brian K. Vaughan & Marcos Martín

Dans un futur post-Internet, les algorithmes ont provoqué un effondrement social. Une réflexion sur la donnée, la surveillance et les IA archivistes.

Saga — Brian K. Vaughan & Fiona Staples

Présence d'IA embarquées, de systèmes conscients et de programmes émotionnels. L'IA y est un personnage à part entière, pas un outil.

11.7 GLOSSAIRE de termes et expressions

- **Actionneur** : Composant qui permet au robot d'agir sur son environnement (ex : moteur, vérin).
- **Algorithme** : Une suite d'instructions précises pour résoudre un problème ou accomplir une tâche. Comme une recette de cuisine, mais pour un ordinateur. *Exemple* : Les étapes pour trier une liste de nombres ou trouver le chemin le plus court sur une carte. *Analogie* : Un mode d'emploi pour un robot. *À distinguer* : Un programme est l'implémentation d'un algorithme dans un langage spécifique.
- **Algorithme génétique** : Méthode d'optimisation inspirée de la sélection naturelle, utilisant des populations de solutions qui évoluent via des opérations comme la mutation et le croisement.
- **Algorithme glouton** (*Greedy algorithm*) : Algorithme qui prend des décisions locales optimales à chaque étape, sans garantie d'optimalité globale. *Exemple* : Algorithme de Kruskal pour les arbres couvrants minimaux.
- **API** (*Application Programming Interface*) : Interface permettant à des logiciels de communiquer entre eux. *Exemple* : L'API de Twitter pour récupérer des tweets.
- **Apprentissage automatique (Machine Learning)** : Une branche de l'IA où les machines apprennent à partir de données sans être explicitement programmées. *Exemple* : Un filtre anti-spam qui apprend à reconnaître les emails indésirables. *Analogie* : Un enfant qui apprend à reconnaître les chats en voyant des centaines de photos. *Sous-catégories* : Supervisé, non supervisé, par renforcement.
- **Apprentissage profond (Deep Learning)** : Un type d'apprentissage automatique utilisant des réseaux de neurones artificiels avec de nombreuses couches pour analyser des données complexes (images, sons, textes). *Exemple* : La reconnaissance faciale de ton téléphone ou les traductions automatiques. *Analogie* : Un cerveau artificiel qui s'entraîne en regardant des millions d'exemples. Sous-ensemble du *Machine Learning* utilisant des réseaux de neurones profonds (plusieurs couches).
- **Architecture von Neumann** : Le modèle de base des ordinateurs modernes, avec une unité centrale (CPU), une mémoire, et des périphériques d'entrée/sortie. *Exemple* : Ton ordinateur, ton smartphone, ou une console de jeu. *Analogie* : Un bureau avec un cerveau (CPU), des tiroirs (mémoire), et des outils (clavier, écran).
- **Asymétrique (chiffrement) :** Système de cryptographie comme RSA, utilisant une clé publique (pour chiffrer, partagée avec tous) et une clé privée (pour déchiffrer, secrète et unique) — employé pour sécuriser les communications en ligne (ex : HTTPS pour les sites web) ou protéger des données sensibles (ex : emails, transactions bancaires)

- **Attaque 51%** : Attaque où un acteur (ou un groupe) contrôle plus de 50% de la puissance de calcul d'un réseau blockchain (PoW), lui permettant de manipuler les transactions. *Risque* : Double dépense (*double spending*).
- **Attaque DDoS** (*Distributed Denial of Service*) : Inondation d'un serveur avec un flot de requêtes pour le rendre inaccessible. *Exemple* : Utilisation d'un botnet pour submerger un site web.
- **Attaque par force brute** (*Brute Force Attack*) : Méthode consistant à tester toutes les combinaisons possibles pour deviner un mot de passe ou une clé. Contre-mesure : Utiliser des mots de passe longs et des verrous de compte après plusieurs tentatives.
- **Authentification multifactorielle** (MFA) : Méthode de vérification d'identité utilisant au moins deux facteurs indépendants parmi :quelque chose que vous savez (mot de passe, code PIN), quelque chose que vous possédez (téléphone, clé USB, carte à puce), quelque chose que vous êtes (empreinte digitale, reconnaissance faciale). Exemple : Connexion à un compte bancaire nécessitant mot de passe + code SMS + empreinte digitale."
- **Autonomie** : Capacité d'un robot à prendre des décisions sans intervention humaine**.** *Niveaux* : De semi-autonome (téléopéré) à pleinement autonome (ex : voiture autonome).

Le robot humanoïde G1 d'Unitree. La société chinoise Wang Xingxing a annoncé prévoir une livraison de 10 000 à 20 000 robots pour 2026

- **Biais (IA)** (*Bias*) : Erreur systématique dans un modèle d'IA, souvent due à des données d'entraînement non représentatives. *Exemple* : Un modèle de reconnaissance faciale moins précis pour les visages féminins.

ANECDOTE	**L'algorithme qui a résolu un mystère médical vieux de 50 ans** Certaines maladies rares laissent des traces subtiles sur le visage : asymétries, proportions inhabituelles, micro-signes invisibles à l'œil nu. Un algorithme d'analyse faciale a permis d'identifier des syndromes génétiques que les médecins ne parvenaient pas à diagnostiquer depuis des décennies. En comparant des milliers de visages, il repère des motifs communs et propose un diagnostic probable. Dans plusieurs cas, il a permis de résoudre des énigmes médicales restées ouvertes pendant plus d'un demi-siècle. Un algorithme devenu détective.

- **Big Data** : *Définition* : Des ensembles de données si volumineux qu'ils nécessitent des outils spécifiques pour être analysés. *Exemple* : Les données des réseaux sociaux (likes, partages,

commentaires) ou les relevés météorologiques. *Analogie* : Une bibliothèque géante où chaque livre est une information, et où il faut un super-ordinateur pour tout trier.

- **Bit** : *Définition* : La plus petite unité d'information en informatique, qui peut valoir **0 ou 1**. *Exemple* : Un pixel d'écran allumé (1) ou éteint (0). *Analogie* : Un interrupteur qui peut être sur ON ou OFF.
- **Bloc** : Unité de base d'une blockchain, contenant : Une liste de transactions. Un hash (empreinte cryptographique) du bloc précédent. Un horodatage (*timestamp*). *Exemple* : Dans Bitcoin, un bloc contient des transactions et est validé environ toutes les 10 minutes.
- **Blockchain** : *Définition* : Une technologie de stockage et de transmission d'informations transparente et sécurisée, fonctionnant sans organe central de contrôle. *Exemple* : Les cryptomonnaies comme le Bitcoin. *Analogie* : Un grand livre de comptes public où tout le monde peut vérifier les transactions, mais personne ne peut les modifier.
  Blockchain (chaîne de blocs) : Base de données décentralisée et sécurisée, où les informations sont stockées sous forme de blocs liés cryptographiquement. *Caractéristiques* : **Immuabilité** : Une fois écrits, les blocs ne peuvent pas être modifiés. **Transparence** : Toutes les transactions sont visibles (selon le type de blockchain). **Décentralisation** : Pas d'autorité centrale (contrairement aux bases de données traditionnelles).
- **Boîte noire** (*Black box*) : Modèle d'IA dont le fonctionnement interne est difficile à interpréter (ex : réseaux de neurones profonds).
- **Botnet** : Réseau d'ordinateurs infectés par un malware et contrôlés à distance par un attaquant (appelé *botmaster*). *Utilisation* : Envoi de spams, attaques DDoS.
- **Boucle** : *Définition* : Une structure qui permet de répéter une action tant qu'une condition est vraie. Exemple : Un compte à rebours ou un jeu qui continue jusqu'à ce que tu perdes. Types : Boucle for : « Répète 10 fois. » Boucle while : *« Répète jusqu'à ce que tu gagnes. »* *Analogie* : Un manège qui tourne tant qu'il y a des enfants dedans.
- **Calculabilité** : *Définition* : L'étude de ce qui peut (ou ne peut pas) être calculé par un algorithme. *Exemple* : Le problème de l'arrêt de Turing montre qu'il existe des problèmes incalculables. *Analogie* : Une liste de problèmes que même le plus puissant ordinateur ne peut résoudre.
- **Capteur** : Dispositif qui mesure une grandeur physique (ex : caméra, lidar, capteur de température). *Rôle* : Fournir des données pour la prise de décision.
- **Cheval de Troie** (*Trojan*) : Malware qui se fait passer pour un logiciel inoffensif.
- **Chiffrement** (*Encryption*) : Processus de transformation de données en un format illisible (texte chiffré) à l'aide d'un algorithme et d'une clé. Exemples : Symétrique : AES (clé unique pour chiffrer/déchiffrer). Asymétrique : RSA (clé publique/privée). *Utilisation* : Sécurisation des communications (HTTPS), stockage de données.
- **Cinématique** : Étude du mouvement des robots (positions, vitesses, accélérations) sans tenir compte des forces**.** *Exemple* : Calcul des trajectoires d'un bras robotisé.
- **Cold Storage** : Méthode de stockage de cryptomonnaies hors ligne (ex : portefeuille papier, hardware wallet) pour les protéger des cyberattaques.
- **Complexité algorithmique** : Mesure des ressources (temps/mémoire) nécessaires à l'exécution d'un algorithme, notée en **O(n)** (notation de Landau). *Exemple* : Un algorithme en $O(n^2)$ ralentit rapidement pour de grandes entrées.
- **Concurrence** (*Concurrency*) : Gestion de plusieurs tâches qui *semblent* s'exécuter en même temps (même sur un seul cœur), via des mécanismes comme le *time-sharing*. *Exemple* : Threads en Java.
- **Condition de course** (*Race Condition*) : Bug causé quand plusieurs threads accèdent/modifient une ressource partagée sans synchronisation. So*lution* : Utiliser des **verrous** (*locks*) ou des structures *thread-safe*.

- **Consensus** : Mécanisme permettant aux nœuds d'un réseau blockchain de s'accorder sur l'état de la blockchain. *Exemples* : PoW, PoS, DPoS (*Delegated Proof of Stake*).
- **Core** (cœur) : Unité physique de calcul dans un processeur. Un CPU multicœur (ex : 8 cœurs) peut exécuter plusieurs threads en parallèle. *À distinguer* : Un cœur ≠ un thread (un cœur peut gérer plusieurs threads via le *multithreading*).
- **Cryptographie** : *Définition* : L'art de protéger l'information en la rendant illisible sans une clé. *Exemple* : Les messages chiffrés sur WhatsApp ou les transactions bancaires en ligne. *Analogie* : Une boîte forte avec un cadenas à combinaison secrète.
- **Cryptographie post-quantique** : *Définition* : Des méthodes de chiffrement résistantes aux attaques des ordinateurs quantiques**.** *Exemple* : Les nouveaux algorithmes comme LWE (Learning With Errors)**.** *Analogie* : Un cadenas si complexe qu'aucun crocheteur futur ne pourra l'ouvrir.
- **Cryptomonnaie** : Monnaie numérique décentralisée utilisant la cryptographie pour sécuriser les transactions. *Exemples* : Bitcoin (BTC), Ethereum (ETH).
- **CSS** (*Cascading Style Sheets*) : Langage de feuilles de style pour définir l'apparence des éléments HTML. *Exemple* : `p { color: blue; } /* Tous les paragraphes seront bleus */`
- **Cybersécurité** : Ensemble des techniques, outils et pratiques visant à protéger les systèmes informatiques, les réseaux et les données contre les cyberattaques. *Domaines clés* : Confidentialité, intégrité, disponibilité (modèle CIA).
- **DAOs** (*Decentralized Autonomous Organizations*) : Organisations gérées par des smart contracts et des règles codées, sans hiérarchie centrale. *Exemple* : Une DAO pour gérer un fonds d'investissement collectif.
- **Data Mining (Fouille de données)** : *Définition* : L'analyse de grandes quantités de données pour en extraire des informations utiles. *Exemple* : Amazon qui te recommande des produits en fonction de tes achats passés. *Analogie* : Chercher une pépite d'or dans une montagne de sable.
- **Deadlock** (interblocage) : Situation où deux threads (ou plus) sont bloqués indéfiniment, chacun attendant une ressource détenue par l'autre. *Exemple* : `// Thread 1 : lockA puis lockB ; // Thread 2 : lockB puis lockA → Deadlock possible !`
- **Décidabilité** : *Définition* : La capacité à déterminer si un problème peut être résolu par un algorithme. *Exemple* : Le problème de l'arrêt est indécidable (on ne peut pas savoir si un programme va s'arrêter). *Analogie* : Une énigme sans solution, comme *« Quel est le son d'une seule main qui applaudit ? »*
- **Deepfake** : *Définition* : Une vidéo ou un enregistrement audio hyper-réaliste et falsifié, créé par une IA. *Exemple* : Une vidéo de toi disant quelque chose que tu n'as jamais dit. *Analogie* : Un caméléon numérique qui imite parfaitement une personne. Technique utilisant l'IA (souvent des réseaux de neurones) pour créer des contenus audio/vidéo falsifiés (ex : une vidéo truquée d'une personnalité). *Risque* : Désinformation, escroquerie.
- **DeFi** (*Decentralized Finance*) : Écosystème de services financiers décentralisés (prêts, échanges, assurances) basés sur des blockchains et des smart contracts. *Exemple* : Uniswap (plateforme d'échange décentralisée).
- **Descente de gradient** (*Gradient Descent*) : Algorithme d'optimisation pour minimiser la fonction de perte en ajustant les paramètres du modèle.
- **Diviser pour régner** (*Divide and Conquer*) : Paradigme qui décompose un problème en sous-problèmes plus simples. *Exemple* : Tri fusion (*merge sort*).
- **Données d'entraînement** (*Training data*) : Ensemble de données utilisé pour entraîner un modèle d'IA. *Qualité cruciale* : Des données biaisées ou mal étiquetées dégradent les performances.

- **Double dépense** (*Double Spending*) : Tentative de dépenser deux fois la même cryptomonnaie**.** *Prévention* : Mécanismes de consensus comme le PoW ou PoS.
- **Dynamique** : Étude des forces agissant sur un robot (ex : calcul des couples pour déplacer un bras).
- **Edge Computing** : Traitement des données localement (sur l'appareil) plutôt que dans le cloud, pour réduire la latence. *Exemple* : IA embarquée dans un smartphone.
- **Effet de bord** (*Side effect*) : Modification de l'état extérieur à une fonction (ex : écrire dans un fichier). *Contre-exemple d'une fonction pure* : `x = x + 1`.
- **Encodage (Encoding)** : *Définition* : La conversion d'informations (texte, image, son) en un format utilisable par un ordinateur. *Exemple* : Transformer une photo en pixels ou un son en ondes numériques. *Analogie* : Traduire une langue en une autre (ex. : français → binaire).
- **Ethique de l'IA** : *Définition* : L'étude des questions morales liées au développement et à l'utilisation de l'intelligence artificielle. *Exemple* : Un algorithme de recrutement qui discrimine involontairement les femmes. *Analogie* : Les règles du jeu pour s'assurer que tout le monde est traité équitablement.
- **Explicabilité** (*Interpretability*) : Capacité à comprendre et expliquer les décisions d'un modèle d'IA. *Outils* : LIME, SHAP.
- **Fédération d'apprentissage** (*Federated Learning*) : Technique d'entraînement d'un modèle d'IA sans centraliser les données, pour préserver la vie privée. *Exemple* : Google Keyboard apprend des suggestions sans voir vos messages.
- **Fonction** : *Définition* : Un bloc de code réutilisable qui effectue une tâche spécifique. *Exemple* : Une fonction pour calculer la moyenne de notes ou afficher un message. *Analogie* : Une recette de cuisine que tu peux réutiliser pour différents plats.
- **Fonction de perte** (*Loss function*) : Mesure l'écart entre les prédictions du modèle et les valeurs réelles. *Exemple* : Erreur quadratique moyenne (MSE).
- **Fonction pure** : Fonction dont le résultat dépend uniquement de ses entrées (pas d'effets de bord). *Exemple* : `carre(x) = x * x`.
- **Fork** (fourche) : Modification du protocole d'une blockchain, pouvant mener à : **Soft fork** : Mise à jour rétrocompatible. **Hard fork** : Création d'une nouvelle blockchain (ex : Bitcoin Cash issu de Bitcoin).
- **Framework** : Structure logicielle réutilisable qui fournit des outils pour développer des applications. *Exemples* : Django (Python), React (JavaScript).
- **Génératif (IA générative)** : *Définition* : Une IA capable de créer du contenu nouveau (textes, images, musique) à partir de données existantes. *Exemple* : DALL-E qui génère des images à partir de descriptions textuelles. *Analogie* : Un artiste qui mélange des couleurs pour créer un tableau unique.

L'algorithme qui a reconstitué un tableau de Rembrandt… que Rembrandt n'a jamais peint

En 2016, un projet baptisé *The Next Rembrandt* a analysé des milliers de fragments de tableaux du peintre : coups de pinceau, textures, proportions, pigments.

L'algorithme a ensuite généré un portrait inédit, dans le style exact du maître. Le résultat est si convaincant que certains experts ont eu du mal à distinguer l'œuvre générée d'un vrai Rembrandt.

Ce n'était pas une copie : c'était un tableau nouveau, mais fidèle à l'esprit du peintre.

Un algorithme capable de prolonger une œuvre humaine à travers les siècles.

- **GPU** (*Graphics Processing Unit*) : Processeur spécialisé pour le calcul parallèle massif, utilisé en IA pour accélérer l'entraînement des réseaux de neurones. *Exemple* : Cartes NVIDIA avec CUDA.
- **Graphe** : *Définition* : Une structure composée de nœuds (points) reliés par des arêtes (lignes)**,** utilisée pour modéliser des relations. *Exemple* : Un réseau social (les nœuds = utilisateurs ; les arêtes = amitiés). *Analogie* : Une carte routière où les villes sont des nœuds et les routes des arêtes.
- **Hameçonnage** (*Phishing*) : Technique d'ingénierie sociale où l'attaquant se fait passer pour une entité de confiance (ex : banque) pour voler des informations (mots de passe, données bancaires). *Exemple* : Un email frauduleux avec un lien vers un faux site web.
- **Hash** : Fonction cryptographique qui transforme une entrée de taille variable en une sortie de taille fixe (ex : SHA-256). *Propriétés* : Déterministe : Même entrée → même sortie. **Irréversible** : Impossible de retrouver l'entrée à partir du hash. *Utilisation* : Vérifier l'intégrité des données (ex : fichiers, blocs de blockchain).
- **Heuristique** : *Définition* : Une méthode approximative pour résoudre un problème quand une solution exacte est trop complexe. *Exemple* : Un GPS qui choisit un trajet « assez bon » plutôt que le meilleur possible. *Analogie* : Une règle de pouce (« Si le ciel est rouge le soir, beau temps demain »).
- **Homomorphic Encryption** : Technique de chiffrement permettant de manipuler des données chiffrées sans les déchiffrer. *Application* : Calculs sur des données sensibles (ex : cloud computing sécurisé).
- **HTML** (*HyperText Markup Language*) : Langage de **balisage** définissant la **structure** d'une page web. *Exemple* :
- **Hyperparamètre** : Paramètre d'un modèle d'IA fixé avant l'entraînement (ex : taux d'apprentissage, nombre de couches). *À distinguer* : Les paramètres sont appris pendant l'entraînement (ex : poids d'un réseau de neurones).
- **IA (Intelligence Artificielle)** : *Définition* : Des systèmes ou machines qui imitent l'intelligence humaine pour effectuer des tâches. *Exemple* : Les assistants vocaux (Siri, Alexa) ou les voitures autonomes. *Analogie* : Un élève doué qui apprend de ses erreurs et s'améliore avec le temps.
- **IA générative** : Modèles capables de créer du contenu (texte, images, musique) à partir de données d'entraînement. *Exemples* : DALL·E (images), GPT-4 (texte).
- **IA symbolique** : Approche basée sur des règles et symboles (ex : systèmes experts), par opposition à l'IA basée sur les données (réseaux de neurones).
- **Implémentation** : mise en œuvre concrète d'une solution, d'un algorithme ou d'une fonctionnalité dans un système.
- **Incrémentation** : augmentation d'une valeur ou d'un compteur d'un pas déterminé, généralement de façon répétée.
- **Inférence** : En **logique/IA** : Processus de déduction de nouvelles informations à partir de règles ou de données. *Exemple* : Un système expert utilisant des règles *"SI… ALORS…"*. En statistiques : Estimation de paramètres à partir d'observations (ex : inférence bayésienne).
- **Informatique quantique** : *Définition* : Une technologie qui utilise les propriétés de la mécanique quantique (comme la superposition) pour effectuer des calculs. *Exemple* : Résoudre des problèmes complexes comme la simulation de molécules. *Analogie* : Un ordinateur qui peut essayer toutes les solutions à un problème en même temps, comme si tu lisais tous les livres d'une bibliothèque en une seconde.
- **Ingénierie sociale** (*Social Engineering*) : Manipulation psychologique pour inciter une victime à divulguer des informations sensibles ou à effectuer une action nuisible. *Exemple* : Un attaquant se faisant passer pour un collègue au téléphone.

- **Interface cerveau-machine** : *Définition* : Un système qui permet de contrôler un ordinateur ou un robot par la pensée. *Exemple* : Les puces Neuralink qui permettent à des paralysés de communiquer. *Analogie* : Un télécommandeur mental pour interagir avec le monde numérique.
- **Intrusion Detection System** (*IDS*) : Outil qui surveille le trafic réseau pour détecter des activités suspectes ou des attaques. *Exemple* : Alerte en cas de tentative de scan de ports.
- **Intrusion Prevention System** (*IPS*) : Comme un IDS, mais bloque activement les menaces détectées.
- **Jumeau numérique (Digital Twin)** : *Définition* : Une réplique virtuelle d'un objet, d'un processus ou d'une personne, utilisée pour des simulations. *Exemple* : Un jumeau numérique d'une ville pour optimiser le trafic. *Analogie* : Un clone virtuel de toi qui vit dans un monde numérique.
- **Jumeau numérique** (*Digital Twin*) : Réplique virtuelle d'un objet physique (ex : une usine) pour **simuler et optimiser** son fonctionnement.
- **Kyberattaque** (*Cyberattack*) : Attaque ciblant les systèmes blockchain ou les portefeuilles de cryptomonnaies. Exemples : Attaque des 51% (voir ci-dessus). Phishing ciblant les clés privées. Exploits de smart contracts (ex : attaque du DAO en 2016).
- **Langage de programmation** : *Définition* : Un langage formel composé d'instructions pour communiquer avec un ordinateur. *Exemple* : Python, Java, C++. *Analogie* : Une langue étrangère que seul l'ordinateur comprend (mais que tu peux apprendre !).

L'algorithme qui a reconstitué une langue disparue

Des linguistes ont utilisé un algorithme pour analyser des milliers de mots provenant de langues anciennes.

L'objectif : reconstituer des racines communes d'une langue proto-indo-européenne disparue depuis plus de 5 000 ans. L'algorithme a repéré des régularités phonétiques et morphologiques invisibles à l'œil humain, proposant des reconstructions plausibles de mots jamais entendus depuis la préhistoire.

Certains linguistes ont été stupéfaits : l'algorithme retrouvait des formes déjà hypothétiques… et en proposait de nouvelles.

Une machine qui redonne une voix à des peuples oubliés.

- **Layer 2** : Solutions construites au-dessus d'une blockchain principale (Layer 1) pour améliorer sa scalabilité et réduire les frais. *Exemples* : Lightning Network (Bitcoin), Polygon (Ethereum).
- **Learning With Errors (LWE)** : *Définition* : Un problème mathématique utilisé en cryptographie post-quantique pour sécuriser les données. *Exemple* : Un algorithme de chiffrement résistant aux attaques quantiques. *Analogie* : Un cadenas si complexe qu'il faudrait des siècles pour le crocheter, même avec un super-ordinateur.
- **Localisation et cartographie simultanées** (*SLAM*) : Technique permettant à un robot de **construire une carte** de son environnement tout en localisant sa position. *Exemple* : Robots aspirateurs comme le Roomba.
- **Machine de Turing** : *Définition* : Un modèle théorique d'ordinateur capable de simuler n'importe quel algorithme. *Exemple* : La base de tous les ordinateurs modernes. *Analogie* : Une machine à écrire infiniment programmable.
- **Machine Learning (Apprentissage automatique)** : *Voir « Apprentissage automatique »*.
- **Malware** (*Malicious Software*) : Logiciel conçu pour nuire à un système. *Types* : Virus : Se propage en s'insérant dans des programmes légitimes. Cheval de Troie (*Trojan*) : Se fait passer pour un logiciel inoffensif. Ransomware : Chiffre les données et demande une rançon. Spyware : Espionne les activités de l'utilisateur.

- **Métadonnée** : *Définition* : Des données sur les données (ex. : la date de création d'une photo, son auteur). *Exemple* : Les tags d'une photo sur Instagram. *Analogie* : L'étiquette sur une boîte qui décrit son contenu.
- **Métaprogrammation** : Écrire des programmes qui génèrent ou manipulent d'autres programmes**.** *Exemple* : Les macros en Rust ou les décorateurs en Python.
- **Mineur** (*Miner*) : Acteur qui valide les transactions et ajoute des blocs à la blockchain (en PoW), en échange de récompenses (ex : bitcoins). *Matériel* : ASIC (circuits spécialisés) ou GPU.
- **Modèle de langage** : *Définition* : Un algorithme entraîné pour comprendre et générer du texte. *Exemple* : GPT-3, qui peut écrire des essais ou répondre à des questions. *Analogie* : Un élève qui a lu tous les livres d'une bibliothèque et peut en discuter.
- **Multicœur** (*Multicore*) : Processeur doté de plusieurs cœurs, permettant un vrai parallélisme. *Exemple* : Un CPU quadricœur peut exécuter 4 tâches lourds simultanément.
- **Neurone artificiel** : *Définition* : Une unité de base d'un réseau de neurones, inspirée des neurones biologiques. *Exemple* : Les couches d'un modèle de deep learning. *Analogie* : Un interrupteur qui s'allume ou s'éteint en fonction des signaux qu'il reçoit.
- **NFT** (*Non-Fungible Token*) **:** Jeton unique et non interchangeable stocké sur une blockchain, représentant la propriété d'un actif numérique (ex : art, musique, objets de jeu). *Exemple* : Un NFT représentant une œuvre d'art numérique sur OpenSea.
- **Nœud (Node)** : *Définition* : Un point dans un graphe représentant une entité (ex. : un utilisateur dans un réseau social). *Exemple* : Dans Facebook, chaque profil est un nœud. *Analogie* : Une personne dans une fête, connectée à d'autres par des conversations.
- **Nœud** (*Node*) : Ordinateur participant au réseau d'une blockchain en validant et relayant les transactions. Types : Nœud complet (Full node) : Stocke toute la blockchain. Nœud léger (Light node) : Ne stocke qu'une partie des données.
- **Open Source** : *Définition* : Un logiciel dont le code source est public et modifiable par tous. *Exemple* : Linux, Python, ou Wikipedia. *Analogie* : Une recette de cuisine partagée librement, que chacun peut améliorer.
- **Optimisation** : *Définition* : Le processus d'amélioration d'un algorithme pour le **rendre** plus rapide ou plus efficace. *Exemple* : Trouver le trajet le plus court ou maximiser les profits d'une entreprise. *Analogie* : Ajuster les ingrédients d'une recette pour qu'elle soit parfaite.
- **Oracle** : Service qui fournit des données externes (ex : cours d'une action, météo) à une blockchain ou un smart contract. *Exemple* : Chainlink.
- **Ordinateur quantique** : *Voir « Informatique quantique »*.
- **Organigramme** : *Définition* : Une représentation visuelle d'un algorithme, avec des symboles pour les étapes et les décisions. *Exemple* : Un schéma pour expliquer comment faire un sandwich. *Analogie* : Une carte routière pour un algorithme.
- **Overfitting** (Sursajustement) : Quand un modèle apprend trop bien les données d'entraînement (y compris le bruit), au détriment de ses performances sur de nouvelles données.
- **P vs NP** : *Définition* : Un problème non résolu en informatique qui demande si les problèmes vérifiables rapidement peuvent aussi être résolus rapidement. *Exemple* : Le problème du voyageur de commerce. *Analogie* : Trouver le trésor le plus rapidement possible dans un labyrinthe géant.
- **Paradigme de programmation** : Façon de concevoir et d'organiser le code. *Exemples* : Impératif, fonctionnel, orienté objet, logique.
- **Parallélisme** (*Parallelism*) : Exécution simultanée de tâches sur plusieurs cœurs. *Exemple* : Calcul matriciel en utilisant tous les cœurs d'un CPU.
- **Pare-feu** (*Firewall*) : Système (matériel ou logiciel) qui filtre le trafic réseau selon des règles prédéfinies. *Exemple* : Bloquer les connexions entrantes non autorisées.

- **Patch** : Mise à jour logicielle visant à corriger une vulnérabilité. *Exemple* : Correctif de sécurité pour un navigateur web.
- **Portemonnaie** (*Wallet*) : Logiciel ou matériel permettant de stocker, envoyer et recevoir des cryptomonnaies. Types : Portemonnaie chaud (Hot wallet) : Connecté à Internet (ex : MetaMask). Portemonnaie froid (Cold wallet) : Hors ligne (ex : Ledger).
- **Preuve d'enjeu** (*Proof of Stake, PoS*) : Mécanisme de consensus où les validateurs sont choisis en fonction de leur mise (quantité de cryptomonnaie bloquée). *Exemple* : Utilisé par **Ethereum 2.0**. *Avantage* : Moins énergivore que le PoW.
- **Preuve de travail** (*Proof of Work, PoW*) : Mécanisme de consensus où les mineurs résolvent des problèmes mathématiques complexes pour valider des transactions et ajouter des blocs à la blockchain. *Exemple* : Utilisé par Bitcoin. *Inconvénient* : Consommation élevée d'énergie.
- **Problème de l'arrêt (Halting Problem)** : *Définition* : Un problème incalculable posé par Alan Turing : *« Peut-on déterminer si un programme va s'arrêter ou tourner indéfiniment ? »* *Exemple* : Un jeu vidéo qui plante parce qu'un personnage est coincé dans une boucle. *Analogie* : Une énigme sans solution, comme *« Ce statement est faux. »*
- **Processus** : Instance d'un programme en exécution, avec son espace mémoire dédié. *À distinguer* : Un processus est isolé (contrairement aux threads qui partagent la mémoire).
- **Programmation concurrente** : Paradigme permettant l'exécution simultanée de plusieurs tâches dans un même programme, en exploitant des threads, des processus, ou des modèles asynchrones (ex : `async/await` en Python). *Enjeu* : Éviter les conditions de course (race conditions) et les deadlocks.
- **Programmation déclarative** : Le programmeur décrit ce qu'il veut obtenir, pas comment. *Exemple* : SQL (`SELECT * FROM clients WHERE age > 30`).
- **Programmation dynamique** : *Définition* : Une méthode d'optimisation qui résout un problème en le divisant en sous-problèmes plus simples. *Exemple* : Trouver le meilleur chemin dans un labyrinthe en mémorisant les chemins déjà explorés. *Analogie* : Construire une tour en commençant par les briques du bas.
- **Programmation fonctionnelle** : Paradigme où les programmes sont construits par composition de fonctions pures (sans effets de bord). *Exemple* : `map(f, liste)` en Haskell.
- **Pseudo-code** : *Définition* : Une description informelle d'un algorithme, à mi-chemin entre le langage humain et le code. *Exemple* : `SI il pleut ALORS Prendre un parapluie SINON Mettre des lunettes de soleil` *Analogie* : Un brouillon de recette avant de l'écrire en détails.
- **Python** : *Définition* : Un langage de programmation simple et puissant, très utilisé en IA et en data science. *Exemple* : Écrire un programme pour trier une liste ou générer un mot de passe. *Analogie* : Une langue facile à apprendre, comme l'espéranto de la programmation. Langage interprété**,** multi-paradigme (impératif, OO, fonctionnel), connu pour sa lisibilité et ses bibliothèques (ex : `numpy` pour le calcul scientifique, `tensorflow` pour l'IA). *Exemple* :
```
def factorielle(n): return 1 if n == 0 else n * factorielle(n-1)  # Récursivité
```
- **Qubit** : *Définition* : L'unité de base de **l'**informatique quantique, qui peut être 0, 1, ou les deux en même temps (superposition). *Exemple* : Les processeurs quantiques de Google ou IBM. *Analogie* : Une pièce de monnaie qui peut être pile, face, ou les deux à la fois.
- **RAM** (*Random Access Memory*) : Mémoire vive où sont stockées les données et instructions en cours d'exécution. *Caractéristique* : Volatile (effacée à l'extinction). *Unité* : Gigaoctets (Go).
- **Récursivité** : Technique où une fonction s'appelle elle-même pour résoudre un problème en le divisant en sous-problèmes. *Exemple* : Calcul de la factorielle. *Risque* : Dépassement de pile (*stack overflow*) si pas de condition d'arrêt.

- **Réfactoration** (ou *refactoring* en anglais) est un processus essentiel en programmation et en conception d'algorithmes qui consiste à améliorer la structure interne d'un code ou d'un algorithme sans en modifier le comportement externe. L'objectif est de rendre le code plus lisible, maintenable, efficace ou adaptable, tout en garantissant qu'il produit les mêmes résultats.
- **Régression** : *Définition* : Une technique statistique pour prédire une valeur (ex. : le prix d'une maison) à partir de données. *Exemple* : Prédire les notes d'un élève en fonction de son temps d'étude. *Analogie* : Deviner la taille d'un arbre en mesurant son ombre.
- **Réseau de neurones** : *Définition* : Un modèle d'IA inspiré du cerveau, composé de couches de neurones artificiels. *Exemple* : Les systèmes de reconnaissance d'images ou de traduction automatique. *Analogie* : Un cerveau artificiel qui apprend en analysant des exemples. *Types* : Perceptron (1 couche). CNN (*Convolutional Neural Network*) pour les images. RNN (*Recurrent Neural Network*) pour les séquences (ex : texte). *Fonctionnement* : Ajustement des poids via la rétropropagation (*backpropagation*).
- **Rétropropagation (Backpropagation)** : *Définition* : Un algorithme pour entraîner les réseaux de neurones en ajustant les erreurs. *Exemple* : Comment un modèle apprend à reconnaître des chats. *Analogie* : Un prof qui corrige tes erreurs pour t'aider à progresser.
- **Robot** : Système autonome ou semi-autonome capable de percevoir son environnement et d'agir. *Exemples* : Bras robotisés, drones, robots aspirateurs.
- **Robotique** : *Définition* : La conception et la construction de robots capables d'effectuer des tâches automatiquement. *Exemple* : Les robots aspirateurs ou les bras articulés dans les usines. *Analogie* : Un assistant mécanique qui obéit à des ordres précis.
- **Robotique cognitive** : Domaine combinant robotique et IA pour créer des robots capables de raisonner (ex : robots sociaux comme Pepper).
- **Robotique collaborative** (*Cobot*) : Robots conçus pour travailler aux côtés des humains en toute sécurité. *Exemple* : Bras robotisés dans les usines.
- **Sandbox** : Environnement isolé où du code non fiable peut être exécuté sans risque pour le système hôte. *Exemple* : Analyse de fichiers suspects dans un bac à sable (sandbox).
- **Score social** : *Définition* : Un système de notation basé sur le comportement des individus, utilisé pour accorder ou refuser des droits. *Exemple* : Le système chinois de crédit social. *Analogie* : Un bulletin de notes qui détermine si tu as le droit de sortir ou de voyager.
- **Seed Phrase** (phrase de récupération) : Série de 12 à 24 mots permettant de récupérer un portefeuille de cryptomonnaies. *Exemple* : `"army van defense carry..."`. *Important* : À conserver hors ligne et secrète.
- **Singularité technologique** : *Définition* : Le moment hypothétique où une IA dépasserait l'intelligence humaine, déclenchant des changements imprévisibles. *Exemple* : Une machine qui s'améliore elle-même sans contrôle humain. *Analogie* : Un élève qui devient plus intelligent que son professeur.
- **Smart contract** : Définition : Programme auto-exécutant stocké sur une blockchain, dont les termes sont écrits en code et s'appliquent automatiquement lorsque des conditions prédéfinies sont remplies. Exemples : Un bail qui se résilie automatiquement si le loyer n'est pas payé ; Un paiement libéré une fois une livraison confirmée. Analogie : Comme un distributeur automatique : le produit (ou l'action) n'est délivré que si les conditions (paiement, validation) sont remplies. Langages de programmation : Solidity (pour Ethereum) ; Rust (pour Solana).
- **Soft Fork** : Mise à jour rétrocompatible d'un protocole blockchain, où les anciens nœuds peuvent toujours fonctionner avec la nouvelle version.
- **Spyware** : Logiciel malveillant conçu pour espionner les activités d'un utilisateur (historique de navigation, frappes clavier, données personnelles) à son insu.

- **SQL** (*Structured Query Language*) : Langage pour interroger et manipuler des bases de données relationnelles. *Exemple* : `SELECT nom FROM utilisateurs WHERE age > 18;` *Commandes principales* : SELECT, INSERT, UPDATE, DELETE.
- **Superposition quantique** : *Définition* : La capacité d'un qubit à être dans plusieurs états à la fois (contrairement à un bit classique, qui est soit 0, soit 1). *Exemple* : Un ordinateur quantique qui teste toutes les solutions d'un problème en parallèle. *Analogie* : Une porte qui est à la fois ouverte et fermée.
- **Test de Turing** : *Définition* : Un test pour évaluer si une machine peut imiter un humain dans une conversation. *Exemple* : Un chatbot qui répond si bien que tu ne sais pas si c'est un humain ou une IA. *Analogie* : Un jeu où tu dois deviner si ton interlocuteur est une personne ou un robot.
- **Thread** (fil d'exécution) : Plus petite unité de traitement qu'un processus, partageant la même mémoire. Exemple : En Java, un Thread peut exécuter une méthode run() en parallèle. Limite : Les threads sont coûteux en ressources (contrairement aux goroutines en Go).
- **Token** : Actif numérique émis sur une blockchain, représentant : Une cryptomonnaie (ex : Bitcoin, Ether). Un jeton utilitaire (ex : accès à un service). Un jeton de sécurité (ex : action numérique).
- **Token (IA)** : Unité de base dans le traitement du langage (ex : un mot ou un morceau de mot). *Exemple* : Dans la phrase *"Chat bot"*, un modèle comme BERT pourrait découper en tokens : `["Chat", "bot"]`.
- **Transformer (modèle)** : *Définition* : Un type de réseau de neurones spécialisé dans le traitement du langage naturel (NLP). *Exemple* : GPT-3, qui génère des textes cohérents. *Analogie* : Un traducteur universel qui comprend et génère toutes les langues.
- **Tri rapide (Quicksort)** : *Définition* : Un algorithme de tri efficace qui divise une liste en sous-listes plus petites. *Exemple* : Trier une liste de notes par ordre croissant. *Analogie* : Ranger des livres par taille en les séparant en petits tas.
- **Validation croisée** (*Cross-validation*) : Technique pour évaluer un modèle en le testant sur différents sous-ensembles de données**.**
- **Variable** : *Définition* : Un conteneur dans un programme qui stocke une valeur (nombre, texte, etc.). *Exemple* : `age = 17` ou `nom = "Alex"`. *Analogie* : Une boîte étiquetée où tu ranges des informations.
- **Vérification formelle** : *Définition* : Une méthode mathématique pour prouver qu'un programme fait exactement ce qu'il est censé faire. *Exemple* : Vérifier qu'un algorithme de vote est sans faille. *Analogie* : Une relecture ultraprécise d'un contrat pour éviter les erreurs.
- **Vision par ordinateur (Computer Vision)** : *Définition* : Un domaine de l'IA qui permet aux machines de « voir » et interpréter des images. *Exemple* : La reconnaissance faciale ou les voitures autonomes. *Analogie* : Apprendre à un robot à reconnaître un chat sur une photo.
- **VPN** (*Virtual Private Network*) : Technologie créant un tunnel sécurisé entre un appareil et un réseau, souvent utilisé pour chiffrer les communications sur des réseaux publics. *Exemple* : Accès à distance sécurisé aux ressources d'une entreprise.
- **Vulnérabilité** : Faiblesse dans un système pouvant être exploitée par un attaquant. *Exemple* : Une faille dans un logiciel non mis à jour (ex : Log4j).
- **Zéro (en informatique)** : *Définition* : Un chiffre essentiel en numération, représentant l'absence de valeur. *Exemple* : Le système binaire (0 et 1) ou les adresses IP. *Analogie* : Le point de départ sur une règle graduée.
- **Zero Trust** : Modèle de sécurité basé sur le principe "ne jamais faire confiance, toujours vérifier". Chaque accès (même interne) doit être authentifié, autorisé et chiffré.

- **Zero-Knowledge Proof** (*Preuve à divulgation nulle de connaissance*) : Technique cryptographique permettant de prouver qu'une information est vraie sans la révéler**.** *Application* : Transactions privées (ex : Zcash).

11.8 Sigles — Algorithmes, Programmation, IA

1. **Généraux en informatique**
   - CPU — Central Processing Unit — Processeur central.
   - GPU — Graphics Processing Unit — Processeur graphique, très utilisé en IA.
   - RAM — Random Access Memory — Mémoire vive.
   - ROM — Read-Only Memory — Mémoire en lecture seule.
   - I/O — Input/Output — Entrées / sorties.
   - API — Application Programming Interface — Interface de programmation.
   - CLI — Command Line Interface — Interface en ligne de commande.
   - GUI — Graphical User Interface — Interface graphique.
   - OS — Operating System — Système d'exploitation.
   - FS — File System — Système de fichiers.
   - LAN — Local Area Network — Réseau local.
   - WAN — Wide Area Network — Réseau étendu.
   - DNS — Domain Name System — Système de noms de domaine.
   - HTTP — HyperText Transfer Protocol — Protocole web.
   - HTTPS — HTTP Secure — Version sécurisée.
   - URL — Uniform Resource Locator — Adresse web.
   - SSL/TLS — Secure Sockets Layer / Transport Layer Security — Protocoles de sécurité.

2. **Algorithmique et analyse**
   - DP — Dynamic Programming — Programmation dynamique.
   - DC — Divide and Conquer — Diviser pour régner.
   - BFS — Breadth-First Search — Parcours en largeur.
   - DFS — Depth-First Search — Parcours en profondeur.
   - A* — A-star — Algorithme de recherche heuristique.
   - TSP — Travelling Salesman Problem — Problème du voyageur de commerce.
   - SAT — Boolean Satisfiability Problem — Problème de satisfiabilité.
   - P — Polynomial time — Classe des problèmes polynomiaux.
   - NP — Nondeterministic Polynomial time — Classe NP.
   - NPC — NP-Complete — NP-complet.
   - NPH — NP-Hard — NP-difficile.
   - CFG — Context-Free Grammar — Grammaire hors contexte.
   - FA — Finite Automaton — Automate fini.
   - DFA — Deterministic Finite Automaton — Automate fini déterministe.
   - NFA — Nondeterministic Finite Automaton — Automate non déterministe.
   - TM — Turing Machine — Machine de Turing**.**

3. **Structures de données**
   - ADT — Abstract Data Type — Type abstrait de données.
   - BST — Binary Search Tree — Arbre binaire de recherche.

- AVL — Adelson-Velsky and Landis tree — Arbre équilibré AVL.
- RBT — Red-Black Tree — Arbre rouge-noir.
- LL — Linked List — Liste chaînée.
- DLL — Doubly Linked List — Liste doublement chaînée.
- PQ — Priority Queue — File de priorité.
- HT — Hash Table — Table de hachage.
- UF — Union-Find — Structure d'union-recherche.

4. **Programmation (général et Python)**
   - OOP — Object-Oriented Programming — Programmation orientée objet.
   - FP — Functional Programming — Programmation fonctionnelle.
   - REPL — Read-Eval-Print Loop — Console interactive Python.
   - PEP — Python Enhancement Proposal — Standard Python.
   - VENV — Virtual Environment — Environnement virtuel Python.
   - IDE — Integrated Development Environment — Environnement de développement.
   - SDK — Software Development Kit — Kit de développement.
   - JSON — JavaScript Object Notation — Format de données.
   - YAML — YAML Ain't Markup Language — Format de configuration.
   - CSV — Comma-Separated Values — Format tabulaire simple.
   - UTF-8 — Unicode Transformation Format — Encodage texte**.**

5. **Bases de données**
   - SQL — Structured Query Language — Langage de requêtes.
   - DBMS — Database Management System — Système de gestion de base de données.
   - RDBMS — Relational DBMS — Base relationnelle.
   - ACID — Atomicity, Consistency, Isolation, Durability — Propriétés transactionnelles.
   - OLTP — Online Transaction Processing — Traitement transactionnel.
   - OLAP — Online Analytical Processing — Analyse multidimensionnelle.
   - ETL — Extract, Transform, Load — Pipeline de données.
   - CRUD — Create, Read, Update, Delete — Opérations de base.

6. **Intelligence artificielle (général)**
   - AI — Artificial Intelligence — Intelligence artificielle.
   - ML — Machine Learning — Apprentissage automatique.
   - DL — Deep Learning — Apprentissage profond.
   - RL — Reinforcement Learning — Apprentissage par renforcement.
   - SL — Supervised Learning — Apprentissage supervisé.
   - UL — Unsupervised Learning — Apprentissage non supervisé.
   - SSL — Self-Supervised Learning — Apprentissage auto-supervisé.
   - CV — Computer Vision — Vision par ordinateur.
   - NLP — Natural Language Processing — Traitement du langage naturel.
   - ASR — Automatic Speech Recognition — Reconnaissance vocale.
   - TTS — Text-to-Speech — Synthèse vocale.
   - **OCR — *Optical Character Recognition* — Reconnaissance optique de caractères.**

7. **Réseaux de neurones**
   - ANN — Artificial Neural Network — Réseau de neurones artificiels.
   - CNN — Convolutional Neural Network — Réseau convolutionnel.
   - RNN — Recurrent Neural Network — Réseau récurrent.
   - LSTM — Long Short-Term Memory — Variante de RNN.
   - GRU — Gated Recurrent Unit — Variante de RNN.

- GAN — Generative Adversarial Network — Réseau génératif antagoniste.
- VAE — Variational Autoencoder — Autoencodeur variationnel.
- MLP — Multilayer Perceptron — Perceptron multicouche.
- FFN — Feed-Forward Network — Réseau à propagation avant.
- BN — Batch Normalization — Normalisation par lot.
- LN — Layer Normalization — Normalisation par couche.

8. **Transformers et IA générative**
   - LLM — Large Language Model — Modèle de langage de grande taille.
   - MMLU — Massive Multitask Language Understanding.
   - GPT — Generative Pretrained Transformer — Famille de modèles génératifs.
   - BERT — Bidirectional Encoder Representations from Transformers.
   - T5 — Text-to-Text Transfer Transformer.
   - CLIP — Contrastive Language–Image Pretraining.
   - RAG — Retrieval-Augmented Generation — Génération augmentée par recherche.
   - LoRA — Low-Rank Adaptation — Technique de fine-tuning léger.
   - SFT — Supervised Fine-Tuning.
   - RLHF — Reinforcement Learning from Human Feedback.
   - MoE — Mixture of Experts — Architecture à experts multiples.
   - KV-cache — Key-Value Cache — Mémoire accélérant l'inférence.

9. **Statistiques et optimisation**
   - MSE — Mean Squared Error — Erreur quadratique moyenne.
   - MAE — Mean Absolute Error — Erreur absolue moyenne.
   - RMSE — Root Mean Squared Error.
   - MAPE — Mean Absolute Percentage Error.
   - SVD — Singular Value Decomposition.
   - PCA — Principal Component Analysis — Analyse en composantes principales.
   - SGD — Stochastic Gradient Descent.
   - ADAM — Adaptive Moment Estimation — Optimiseur.
   - ROC — Receiver Operating Characteristic.
   - AUC — Area Under Curve.

10. **Cloud, DevOps et outils modernes**
    - CI — Continuous Integration.
    - CD — Continuous Deployment / Delivery.
    - IaC — Infrastructure as Code.
    - SaaS — Software as a Service.
    - PaaS — Platform as a Service.
    - IaaS — Infrastructure as a Service.
    - VM — Virtual Machine.
    - K8s — Kubernetes.
    - API REST — Representational State Transfer.
    - JWT — JSON Web Token.

Livre rédigé par Gérard Villemin
Site : https://diconombre.fr

Livres déjà parus:
Voyages au pays des nombres -
http://diconombre.fr/Referenc/Prof/THdesNBS/ThdesNbs.pdf
Les sous-marins: l'empire du silence -
https://diconombre.fr/aDefense/sousmarin/SMLivre.pdf