P versus NP

(partie 2)

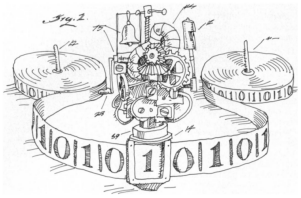
Arnaud Casteigts
Université de Genève

Retraite annuelle de la CRM
Champéry

11 / 09 / 2026.

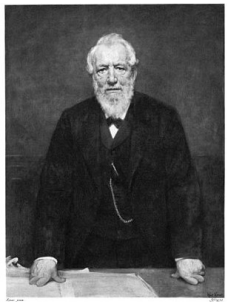
(Partie I)

Décidabilité



Les limites de la connaissance ?

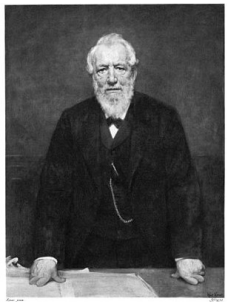
“Ignoramus et ignorabimus” (1872)



Emil du Bois-Reymond (1818-1896)
(physiologiste)

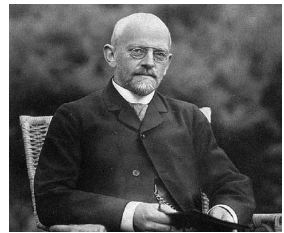
Les limites de la connaissance ?

“Ignoramus et ignorabimus” (1872)



Emil du Bois-Reymond (1818-1896)
(physiologiste)

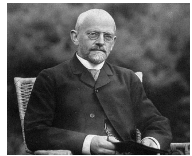
“Wir müssen wissen, wir werden wissen” (1930)



David Hilbert (1862-1943)
(mathématicien)

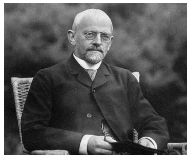
La crise des fondements

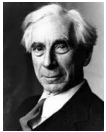
“Wir müssen wissen, wir werden wissen”
David Hilbert (1862–1943)



La crise des fondements

“Wir müssen wissen, wir werden wissen”
David Hilbert (1862–1943)





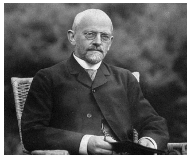
Pas avec la théorie naïve des ensembles !

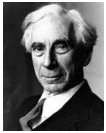
Soit $R = \{x \mid x \notin x\}$, alors $R \in R \iff R \notin R$

Bertrand Russell (1872-1970)

La crise des fondements

“Wir müssen wissen, wir werden wissen”
David Hilbert (1862–1943)





Pas avec la théorie naïve des ensembles !

Soit $R = \{x \mid x \notin x\}$, alors $R \in R \iff R \notin R$

Bertrand Russell (1872-1970)

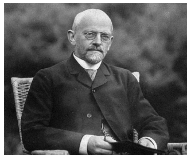
Quel que soit le système, en fait !

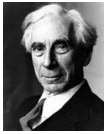
Tout système axiomatique “suffisamment expressif” est soit incomplet, soit incohérent.

Kurt Gödel (1906-1978)

La crise des fondements

“Wir müssen wissen, wir werden wissen”
David Hilbert (1862–1943)





Pas avec la théorie naïve des ensembles !

Soit $R = \{x \mid x \notin x\}$, alors $R \in R \iff R \notin R$

Bertrand Russell (1872-1970)

Quel que soit le système, en fait !

Tout système axiomatique “suffisamment expressif” est soit incomplet, soit incohérent.

Kurt Gödel (1906-1978)

Et même des questions naturelles !

Comme savoir si un algorithme s’arrêtera.

Alan Turing (1912-1954)

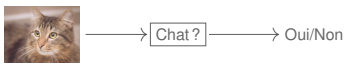
Traitement de l'information

Traitement de l'information

Tri de données :

Calcul d'itinéraire :

Reconnaissance d'image :



Mathématiques :

Traitement de l'information

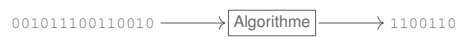
Tri de données :

Calcul d'itinéraire :

Reconnaissance d'image :

Mathématiques :

Plus généralement



Un algorithme prend en entrée une suite de **symboles** et produit en sortie une autre suite de **symboles**. Il réalise un **traitement**.

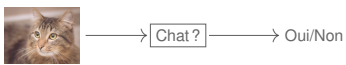
Problème de **décision** : cas particulier dont la sortie est `1` ou `0` (Oui ou Non).

Traitement de l'information

Tri de données :

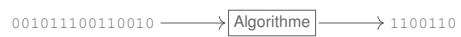
Calcul d'itinéraire :

Reconnaissance d'image :



Mathématiques :

Plus généralement

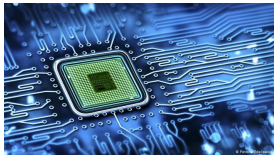


Un algorithme prend en entrée une suite de **symboles** et produit en sortie une autre suite de **symboles**. Il réalise un **traitement**.

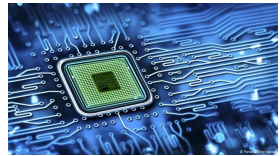
Problème de **décision** : cas particulier dont la sortie est `1` ou `0` (Oui ou Non).

Mais en fait, qu'est-ce qu'un algorithme ?

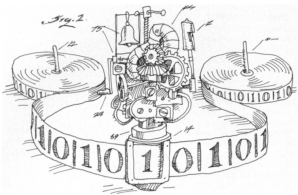
Programme ? Algorithme ?



Programme ? Algorithme ?



Machine de Turing (1936)



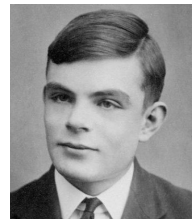
230 A. M. TURING [Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

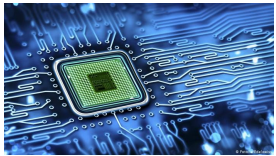
By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

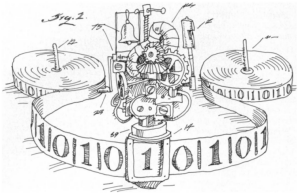
The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are,



Programme ? Algorithme ?



Machine de Turing (1936)



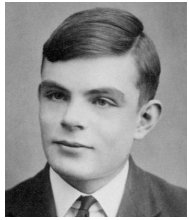
230 A. M. TURING [Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are,



→ Modèle **mathématique** d'algorithme, présumément capable de représenter **n'importe quel traitement** d'information physiquement réalisable (Conjecture de Church-Turing).

Formaliser l'informatique (survol)

Langage formel : ensemble de mots sur un alphabet Σ.

Exemples :

- $L_{pair} = \{0, 10, 100, 110, 1000, 1010, \dots\}$ sur $\Sigma = \{0, 1\}$ (nombres pairs)
- $L_{ab} = \{ab, aabb, aaabbb, \dots\}$ sur l'alphabet $\Sigma = \{a, b\}$ (des a suivis d'autant de b)
- $L_{3-col} = \{100101100101, 1101100110101, \dots\}$ (graphes 3-colorables)
- $L_{chat} = \{0101010110100110, 10101100101100101, \dots\}$ (photos JPG de chats)

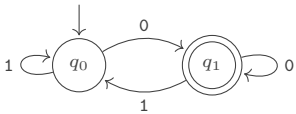
Un problème de décision n'est rien d'autre qu'un langage formel.

Résoudre un problème de décision = Savoir décider si un mot donné appartient au langage correspondant.

On a un modèle simple (les ensembles) de ce qu'est un problème.
Qu'en est-il des modèles de machines ?

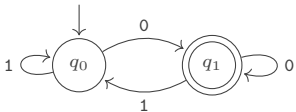
Modèles de machines

Automates finis :

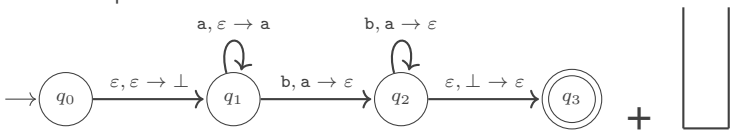


Modèles de machines

Automates finis :

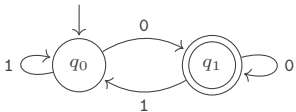


Automates à pile :

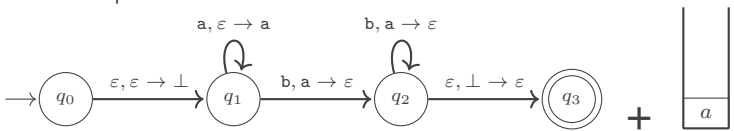


Modèles de machines

Automates finis :

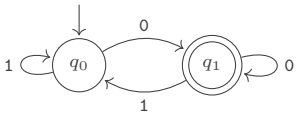


Automates à pile :

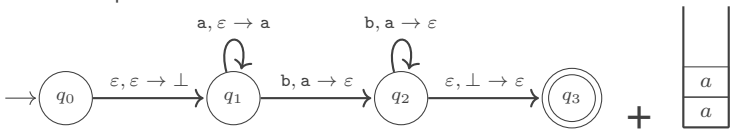


Modèles de machines

Automates finis :

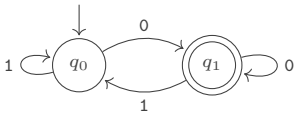


Automates à pile :

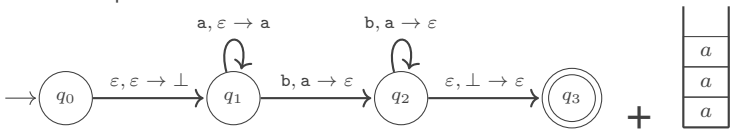


Modèles de machines

Automates finis :

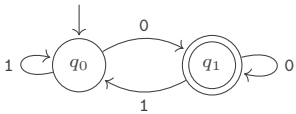


Automates à pile :

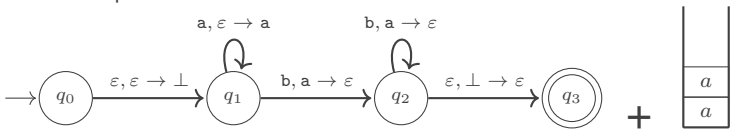


Modèles de machines

Automates finis :

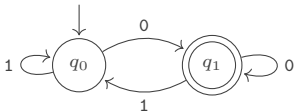


Automates à pile :

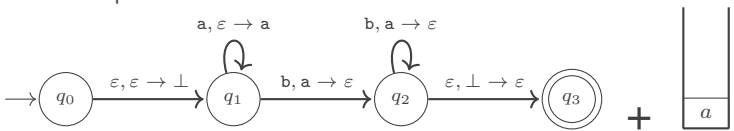


Modèles de machines

Automates finis :

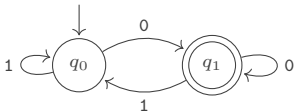


Automates à pile :

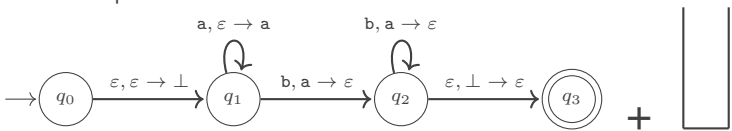


Modèles de machines

Automates finis :

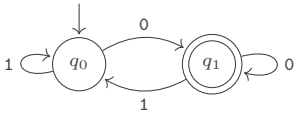


Automates à pile :

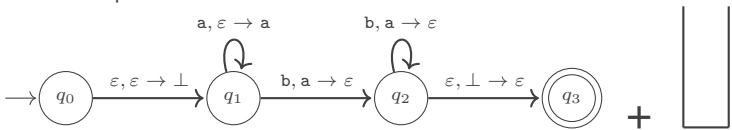


Modèles de machines

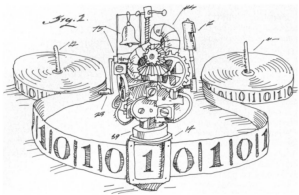
Automates finis :



Automates à pile :



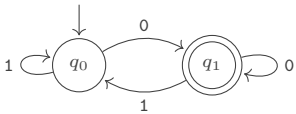
Machine de Turing :



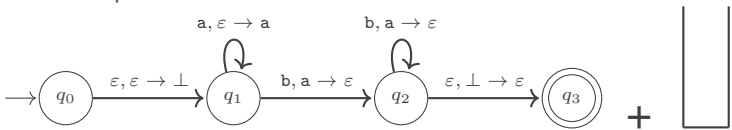
= Automate + bande mémoire infinie où l'on peut lire, écrire et se déplacer.

Modèles de machines

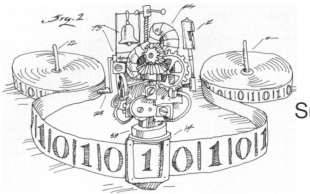
Automates finis :



Automates à pile :



Machine de Turing :



= Automate + bande mémoire infinie où l'on peut lire, écrire et se déplacer.

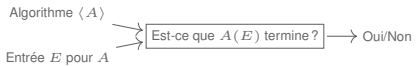
Modélise l'algorithme et la machine (les deux à la fois).

Supposé universel : tout traitement d'information réalisable l'est dans ce modèle.

Existe-t-il des problèmes non-résolubles (langages non décidables) ?

Le problème de l'arrêt (un méta-problème)

Problème de l'arrêt (Halting problem) :

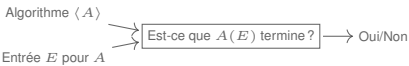


Version plus forte (pour info) :

Existe-t-il un algorithme $A_H(\langle A \rangle, E)$ capable de répondre à cette question dans tous les cas ?

Le problème de l'arrêt (un méta-problème)

Problème de l'arrêt (Halting problem) :



Version plus forte (pour info) :

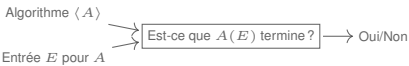
Existe-t-il un algorithme $A_H(\langle A \rangle, E)$ capable de répondre à cette question dans tous les cas ?

Raisonnement par l'absurde

Supposons que A_H existe. On peut alors facilement créer un algorithme A_P qui résoud le cas particulier suivant :

Le problème de l'arrêt (un méta-problème)

Problème de l'arrêt (Halting problem) :



Version plus forte (pour info) :

Existe-t-il un algorithme $A_H(\langle A \rangle, E)$ capable de répondre à cette question dans tous les cas ?

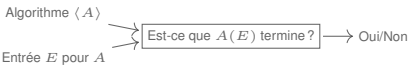
Raisonnement par l'absurde

Supposons que A_H existe. On peut alors facilement créer un algorithme A_P qui résoud le cas particulier suivant :

$A_P(\langle A \rangle)$:
  Si $A_H(\langle A \rangle, \langle A \rangle)$ dit oui :
    Dire oui
  Sinon
    Dire non

Le problème de l'arrêt (un méta-problème)

Problème de l'arrêt (Halting problem) :

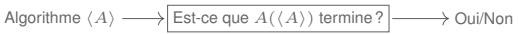


Version plus forte (pour info) :

Existe-t-il un algorithme $A_H(\langle A \rangle, E)$ capable de répondre à cette question dans tous les cas ?

Raisonnement par l'absurde

Supposons que A_H existe. On peut alors facilement créer un algorithme A_P qui résoud le cas particulier suivant :



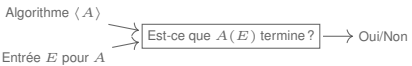
$A_P(\langle A \rangle)$:
  Si $A_H(\langle A \rangle, \langle A \rangle)$ dit oui :
    Dire oui
  Sinon
    Dire non

On peut aussi (soyons tordus) utiliser A_H pour créer un algorithme A_T qui prend en entrée un algorithme $\langle A \rangle$ et adopte le comportement opposé de $A(\langle A \rangle)$, à savoir boucler si $A(\langle A \rangle)$ termine et terminer si $A(\langle A \rangle)$ boucle.

$A_T(\langle A \rangle)$:
  Si $A_H(\langle A \rangle, \langle A \rangle)$ dit oui :
    Boucler à l'infini
  Sinon
    Terminer

Le problème de l'arrêt (un méta-problème)

Problème de l'arrêt (Halting problem) :

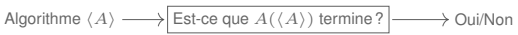


Version plus forte (pour info) :

Existe-t-il un algorithme $A_H(\langle A \rangle, E)$ capable de répondre à cette question dans tous les cas ?

Raisonnement par l'absurde

Supposons que A_H existe. On peut alors facilement créer un algorithme A_P qui résoud le cas particulier suivant :



$A_P(\langle A \rangle)$:
  Si $A_H(\langle A \rangle, \langle A \rangle)$ dit oui :
    Dire oui
  Sinon
    Dire non

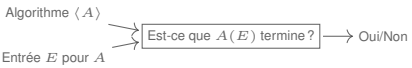
On peut aussi (soyons tordus) utiliser A_H pour créer un algorithme A_T qui prend en entrée un algorithme $\langle A \rangle$ et adopte le comportement opposé de $A(\langle A \rangle)$, à savoir boucler si $A(\langle A \rangle)$ termine et terminer si $A(\langle A \rangle)$ boucle.

$A_T(\langle A \rangle)$:
  Si $A_H(\langle A \rangle, \langle A \rangle)$ dit oui :
    Boucler à l'infini
  Sinon
    Terminer

Que se passe-t-il si l'on exécute $A_T(\langle A_T \rangle)$?

Le problème de l'arrêt (un méta-problème)

Problème de l'arrêt (Halting problem) :

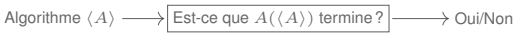


Version plus forte (pour info) :

Existe-t-il un algorithme $A_H(\langle A \rangle, E)$ capable de répondre à cette question dans tous les cas ?

Raisonnement par l'absurde

Supposons que A_H existe. On peut alors facilement créer un algorithme A_P qui résoud le cas particulier suivant :



$A_P(\langle A \rangle)$:
  Si $A_H(\langle A \rangle, \langle A \rangle)$ dit oui :
    Dire oui
  Sinon
    Dire non

On peut aussi (soyons tordus) utiliser A_H pour créer un algorithme A_T qui prend en entrée un algorithme $\langle A \rangle$ et adopte le comportement opposé de $A(\langle A \rangle)$, à savoir boucler si $A(\langle A \rangle)$ termine et terminer si $A(\langle A \rangle)$ boucle.

$A_T(\langle A \rangle)$:
  Si $A_H(\langle A \rangle, \langle A \rangle)$ dit oui :
    Boucler à l'infini
  Sinon
    Terminer

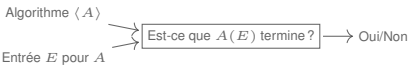
Que se passe-t-il si l'on exécute $A_T(\langle A_T \rangle)$?

- Si A_H nous dit que $A_T(\langle A_T \rangle)$ termine, alors $A_T(\langle A_T \rangle)$ boucle
- Si A_H nous dit que $A_T(\langle A_T \rangle)$ boucle, alors $A_T(\langle A_T \rangle)$ termine

Contradiction, A_H ne peut pas exister.

Le problème de l'arrêt (un méta-problème)

Problème de l'arrêt (Halting problem) :



Version plus forte (pour info) :

Existe-t-il un algorithme $A_H(\langle A \rangle, E)$ capable de répondre à cette question dans tous les cas ?

Raisonnement par l'absurde

Supposons que A_H existe. On peut alors facilement créer un algorithme A_P qui résoud le cas particulier suivant :

$A_P(\langle A \rangle)$:
  Si $A_H(\langle A \rangle, \langle A \rangle)$ dit oui :
    Dire oui
  Sinon
    Dire non

On peut aussi (soyons tordus) utiliser A_H pour créer un algorithme A_T qui prend en entrée un algorithme $\langle A \rangle$ et adopte le comportement opposé de $A(\langle A \rangle)$, à savoir boucler si $A(\langle A \rangle)$ termine et terminer si $A(\langle A \rangle)$ boucle.

$A_T(\langle A \rangle)$:
  Si $A_H(\langle A \rangle, \langle A \rangle)$ dit oui :
    Boucler à l'infini
  Sinon
    Terminer

Que se passe-t-il si l'on exécute $A_T(\langle A_T \rangle)$?

- Si A_H nous dit que $A_T(\langle A_T \rangle)$ termine, alors $A_T(\langle A_T \rangle)$ boucle
- Si A_H nous dit que $A_T(\langle A_T \rangle)$ boucle, alors $A_T(\langle A_T \rangle)$ termine

(et toc !)

Contradiction, A_H ne peut pas exister.

Autres problèmes indécidables

Tous ces problèmes sont indécidables **dans le cas général**. Cela n'empêche pas des cas particuliers d'être décidable, bien sûr.

Autres problèmes indécidables

Tous ces problèmes sont indécidables **dans le cas général**. Cela n'empêche pas des cas particuliers d'être décidable, bien sûr.

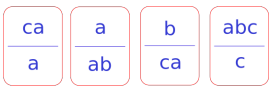
- Résolution d'équations diophantiennes (théorème de Matiyasevich, 10ème problème de Hilbert)
  (équations polynomiales à coefficients entiers et solutions entières)

Autres problèmes indécidables

Tous ces problèmes sont indécidables **dans le cas général**. Cela n'empêche pas des cas particuliers d'être décidable, bien sûr.

- Résolution d'équations diophantiennes (théorème de Matiyasevich, 10ème problème de Hilbert)
  (équations polynomiales à coefficients entiers et solutions entières)

- Problème de correspondance de Post



Autres problèmes indécidables

Tous ces problèmes sont indécidables **dans le cas général**. Cela n'empêche pas des cas particuliers d'être décidable, bien sûr.

- Résolution d'équations diophantiennes (théorème de Matiyasevich, 10ème problème de Hilbert)
  (équations polynomiales à coefficients entiers et solutions entières)

- Problème de correspondance de Post

ca	a	b	abc
a	ab	ca	c

Autres problèmes indécidables

Tous ces problèmes sont indécidables **dans le cas général**. Cela n'empêche pas des cas particuliers d'être décidable, bien sûr.

- Résolution d'équations diophantiennes (théorème de Matiyasevich, 10ème problème de Hilbert)
  (équations polynomiales à coefficients entiers et solutions entières)

- Problème de correspondance de Post

ca	a	b	abc
a	ab	ca	c

- Décider si un algorithme répond toujours OUI.

Autres problèmes indécidables

Tous ces problèmes sont indécidables **dans le cas général**. Cela n'empêche pas des cas particuliers d'être décidable, bien sûr.

- Résolution d'équations diophantiennes (théorème de Matiyasevich, 10ème problème de Hilbert)
  (équations polynomiales à coefficients entiers et solutions entières)

- Problème de correspondance de Post

ca	a	b	abc
a	ab	ca	c

- Décider si un algorithme répond toujours OUI.
- Nombreuses propriétés de programme... (Théorème de Rice)

Autres problèmes indécidables

Tous ces problèmes sont indécidables **dans le cas général**. Cela n'empêche pas des cas particuliers d'être décidable, bien sûr.

- Résolution d'équations diophantiennes (théorème de Matiyasevich, 10ème problème de Hilbert)
  (équations polynomiales à coefficients entiers et solutions entières)

- Problème de correspondance de Post

ca	a	b	abc
a	ab	ca	c

- Décider si un algorithme répond toujours OUI.
- Nombreuses propriétés de programme... (Théorème de Rice)
- (ajouté hier) Deux k-variétés données ($k \geq 4$) sont-elles homéomorphes

Autres problèmes indécidables

Tous ces problèmes sont indécidables **dans le cas général**. Cela n'empêche pas des cas particuliers d'être décidable, bien sûr.

- Résolution d'équations diophantiennes (théorème de Matiyasevich, 10ème problème de Hilbert)
  (équations polynomiales à coefficients entiers et solutions entières)

- Problème de correspondance de Post

ca	a	b	abc
a	ab	ca	c

- Décider si un algorithme répond toujours OUI.
- Nombreuses propriétés de programme... (Théorème de Rice)
- (ajouté hier) Deux k-variétés données ($k \geq 4$) sont-elles homéomorphes
- Problème ouvert (ajouté avant-hier) : Le rang d'une courbe elliptique est-il décidable ?

Autres problèmes indécidables

Tous ces problèmes sont indécidables **dans le cas général**. Cela n'empêche pas des cas particuliers d'être décidable, bien sûr.

- Résolution d'équations diophantiennes (théorème de Matiyasevich, 10ème problème de Hilbert)
  (équations polynomiales à coefficients entiers et solutions entières)

- Problème de correspondance de Post

ca / a	a / ab	b / ca	abc / c

- Décider si un algorithme répond toujours OUI.
- Nombreuses propriétés de programme... (Théorème de Rice)
- (ajouté hier) Deux k-variétés données ($k \geq 4$) sont-elles homéomorphes
- Problème ouvert (ajouté avant-hier) : Le rang d'une courbe elliptique est-il décidable ?

En fait :

- Le nombre d'**algorithmes** a la cardinalité des entiers ($\aleph_0$)
- Le nombre de **problèmes** a la cardinalité des réels ($2^{\aleph_0}$)

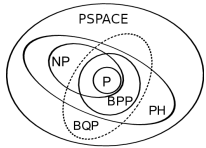
$\rightarrow$ La majorité des problèmes n'est pas décidable.
(heureusement, beaucoup n'ont aucun intérêt)

Natural	Real
0	0.236436775676...
1	0.098473294543...
2	0.193214042202...
3	0.843279242093...
4	0.012934812343...
5	0.639423412934...
6	0.017773923845...
7	0.238920090909...
8	0.123984732999...
9	0.646329878122...
10	0.000123943437...
11	0.981298312892...
⋮	⋮
	0.293233992132...
	0.746894310875...

Argument diagonal (Cantor)

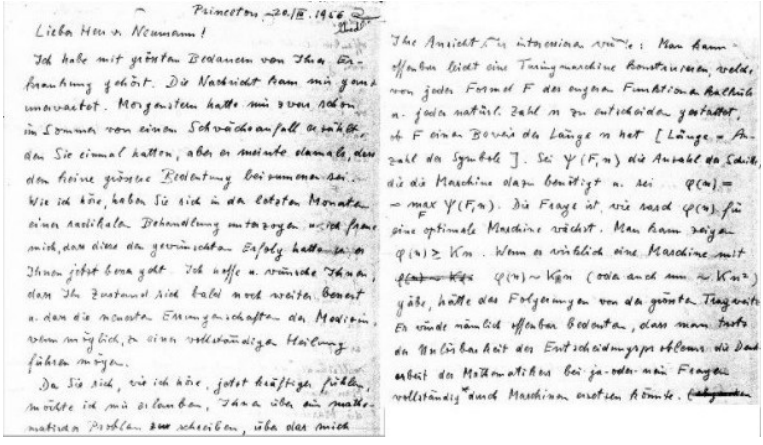
Deuxième limite des connaissances

Complexité algorithmique



(désormais, on reste dans le monde décidable)

La lettre de Gödel à Von Neumann (1956)



Princeton, 20/III. 1956

Liebe Herr v. Neumann!

Ich habe mit größtem Bedauern von Ihre Erkrankung gehört. Die Nachricht kam mir ganz unerwartet. Morgenstern hatte mir zwar schon im Sommer von einem Schwächeanfall erzählt, den Sie einmal hatten, aber er meinte damals, dass keine größere Bedeutung beizumessen sei. Wie ich höre, haben Sie sich in der letzten Monaten einer radikalen Behandlung unterzogen, und ich wünsche, dass diese den gewünschten Erfolg hatte und Ihnen jetzt beim gibt. Ich hoffe, es wünsche Ihnen, dass Ihr Zustand sich bald noch weiter bessert, und dass die neuesten Errungenschaften der Medizin, wenn möglich, zu einer vollständigen Heilung führen mögen.

Da Sie sich, wie ich höre, jetzt häufiger fühlen, möchte ich mir erlauben, Ihnen über ein mathematisches Problem zu schreiben, über das mich Ihre Ansicht sehr interessieren würde: Man kann offenbar leicht eine Turingmaschine konstruieren, welche von jeder Formel F des engeren Funktionalkalküls u. jeder natürl. Zahl n zu entscheiden gestattet, ob F einen Beweis der Länge n hat [Länge = Anzahl der Symbole]. Sei $\psi(F,n)$ die Anzahl der Schritte, die die Maschine dazu benötigt, u. sei $\varphi(n) = \max_F \psi(F,n)$. Die Frage ist, wie rasch $\varphi(n)$ für eine optimale Maschine wächst. Man kann zeigen $\varphi(n) \geq Kn$. Wenn es wirklich eine Maschine mit $\varphi(n) \sim Kn$ (oder auch nur $\sim Kn^2$) gäbe, hätte das Folgerungen von der größten Tragweite. Es würde nämlich offenbar bedeuten, dass man trotz der Unlösbarkeit des Entscheidungsproblems die Denkarbeit der Mathematiker bei ja-oder-nein Fragen vollständig durch Maschinen ersetzen könnte.

Gödel's letter to Von Neumann (1956)

[...] One can obviously construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\Psi(F, n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \Psi(F, n)$.

Gödel's letter to Von Neumann (1956)

[...] One can obviously construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\Psi(F, n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \Psi(F, n)$.

The question is how fast $\varphi(n)$ grows for an optimal machine. (...) If there really were a machine with $\varphi(n) \sim n$ (or even $\sim n^2$), this would have consequences of the greatest importance. Namely, it would mean that (...) the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine.

Gödel's letter to Von Neumann (1956)

[...] One can obviously construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\Psi(F, n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \Psi(F, n)$.

The question is how fast $\varphi(n)$ grows for an optimal machine. (...) If there really were a machine with $\varphi(n) \sim n$ (or even $\sim n^2$), this would have consequences of the greatest importance. Namely, it would mean that (...) the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine.

(...) It would be interesting to know, for instance, the situation concerning the determination of primality of a number and how strongly in general the number of steps in finite combinatorial problems can be reduced with respect to simple exhaustive search.

Et voilà le domaine de la complexité algorithmique lancé !

Complexité algorithmique ?

Ressources nécessaires pour résoudre un problème (décidable).

Quel type de ressources ?

- **Temps (nombre d'opérations)**
- Espace (quantité de mémoire)
- Impact du non-déterminisme ?
- Impact de l'aléa ?
- ...

Complexité algorithmique ?

Ressources nécessaires pour résoudre un problème (décidable).

Quel type de ressources ?

- **Temps (nombre d'opérations)**
- Espace (quantité de mémoire)
- Impact du non-déterminisme ?
- Impact de l'aléa ?
- ...

Point de vue **asymptotique**

- **Évolution** de ces quantités en fonction de la **taille** n de l'entrée, quand $n \to \infty$
- Notations $O(\cdot), \Omega(\cdot), \Theta(\cdot)$ (ignore les facteurs constants et les termes dominés)

  Intuition $\leq$ $\geq$ $=$

  Ex : $3n^2 + 5n + 4 = \Theta(n^2)$

Complexité algorithmique ?

Ressources nécessaires pour résoudre un problème (décidable).

Quel type de ressources ?

- **Temps (nombre d'opérations)**
- Espace (quantité de mémoire)
- Impact du non-déterminisme ?
- Impact de l'aléa ?
- ...

Point de vue **asymptotique**

- **Évolution** de ces quantités en fonction de la **taille** n de l'entrée, quand $n \to \infty$
- Notations $O(\cdot), \Omega(\cdot), \Theta(\cdot)$ (ignore les facteurs constants et les termes dominés)

  Intuition $\leq \quad \geq \quad =$

  Ex : $3n^2 + 5n + 4 = \Theta(n^2)$

- Quelques adjectifs :

Constant	$O(1)$	Quadratique	$O(n^2)$
Logarithmique	$O(\log n)$	Exponentiel	$O(2^n)$
Linéaire	$O(n)$	Factoriel	$O(n!)$
Quasi-linéaire	$O(n \log n)$	Polynomial	$O(n^c) = n^{O(1)}$

En général, on s'intéresse au **pire cas** (maximum sur toutes les instances possibles du problème).

L'espace et le temps

Classes génériques de problèmes

- TIME($f(n)$) : Problèmes que l'on peut résoudre en temps $O(f(n))$.
- SPACE($f(n)$) : Problèmes que l'on peut résoudre en espace $O(f(n))$.

L'espace et le temps

Classes génériques de problèmes

- TIME($f(n)$) : Problèmes que l'on peut résoudre en temps $O(f(n))$.
- SPACE($f(n)$) : Problèmes que l'on peut résoudre en espace $O(f(n))$.

Cas particuliers les plus connus

Nom commun	Problèmes résolubles en...	Définition
LOGSPACE	espace logarithmique	SPACE($\log n$)
P	temps polynomial	TIME($n^{O(1)}$)
PSPACE	espace polynomial	SPACE($n^{O(1)}$)
EXP	temps exponentiel	TIME(2^n)

L'espace et le temps

Classes génériques de problèmes

- TIME($f(n)$) : Problèmes que l'on peut résoudre en temps $O(f(n))$.
- SPACE($f(n)$) : Problèmes que l'on peut résoudre en espace $O(f(n))$.

Cas particuliers les plus connus

Nom commun	Problèmes résolubles en...	Définition
LOGSPACE	espace logarithmique	SPACE($\log n$)
P	temps polynomial	TIME($n^{O(1)}$)
PSPACE	espace polynomial	SPACE($n^{O(1)}$)
EXP	temps exponentiel	TIME(2^n)

$$\text{LOGSPACE} \subseteq \text{P} \subseteq \text{PSPACE} \subseteq \text{EXP}$$

L'espace et le temps

Classes génériques de problèmes

- TIME($f(n)$) : Problèmes que l'on peut résoudre en temps $O(f(n))$.
- SPACE($f(n)$) : Problèmes que l'on peut résoudre en espace $O(f(n))$.

Cas particuliers les plus connus

Nom commun	Problèmes résolubles en...	Définition
LOGSPACE	espace logarithmique	SPACE($\log n$)
P	temps polynomial	TIME($n^{O(1)}$)
PSPACE	espace polynomial	SPACE($n^{O(1)}$)
EXP	temps exponentiel	TIME(2^n)

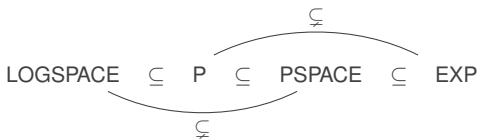
L'espace et le temps

Classes génériques de problèmes

- TIME$(f(n))$: Problèmes que l'on peut résoudre en temps $O(f(n))$.
- SPACE$(f(n))$: Problèmes que l'on peut résoudre en espace $O(f(n))$.

Cas particuliers les plus connus

Nom commun	Problèmes résolubles en...	Définition
LOGSPACE	espace logarithmique	SPACE$(\log n)$
P	temps polynomial	TIME$(n^{O(1)})$
PSPACE	espace polynomial	SPACE$(n^{O(1)})$
EXP	temps exponentiel	TIME(2^n)



L'espace et le temps

Classes génériques de problèmes

- TIME($f(n)$) : Problèmes que l'on peut résoudre en temps $O(f(n))$.
- SPACE($f(n)$) : Problèmes que l'on peut résoudre en espace $O(f(n))$.

Cas particuliers les plus connus

Nom commun	Problèmes résolubles en...	Définition
LOGSPACE	espace logarithmique	SPACE($\log n$)
P	temps polynomial	TIME($n^{O(1)}$)
PSPACE	espace polynomial	SPACE($n^{O(1)}$)
EXP	temps exponentiel	TIME(2^n)

$$\text{LOGSPACE} \subseteq \text{P} \subseteq \text{PSPACE} \subseteq \text{EXP}$$

La plus importante est la classe P

Problèmes que l'on peut résoudre "rapidement" (en temps $n^{O(1)}$).

(+ robuste / composable / souvent réaliste)

Classes de problèmes NP et coNP

Plusieurs définitions équivalentes, les plus simples :

NP : ∃ **preuve courte** que la réponse est OUI (si la réponse est OUI) – certificat positif

coNP : ∃ **preuve courte** que la réponse est NON (si la réponse est NON) – certificat négatif

Preuve courte = **vérifiable** en temps polynomial

Classes de problèmes NP et coNP

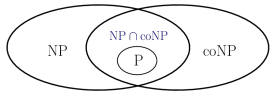
Plusieurs définitions équivalentes, les plus simples :

NP : ∃ **preuve courte** que la réponse est OUI (si la réponse est OUI) – certificat positif

coNP : ∃ **preuve courte** que la réponse est NON (si la réponse est NON) – certificat négatif

Preuve courte = **vérifiable** en temps polynomial

Observation : P ⊆ NP et P ⊆ coNP (l'algorithme lui-même permet de vérifier, sans autre certificat)



Classes de problèmes NP et coNP

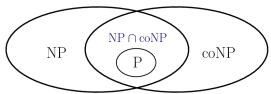
Plusieurs définitions équivalentes, les plus simples :

NP : ∃ **preuve courte** que la réponse est OUI (si la réponse est OUI) – certificat positif

coNP : ∃ **preuve courte** que la réponse est NON (si la réponse est NON) – certificat négatif

Preuve courte = **vérifiable** en temps polynomial

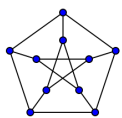
Observation : P $\subseteq$ NP et P $\subseteq$ coNP (l'algorithme lui-même permet de vérifier, sans autre certificat)



Quelques problèmes dans NP (présumément pas dans P) : 3-COLORING, CLIQUE, TSP, FACTORISATION, SAT, ...

Exemple : 3-COLORING

Ce graphe peut-il être colorié avec 3 couleurs ?



Classes de problèmes NP et coNP

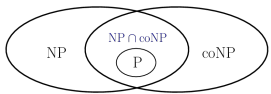
Plusieurs définitions équivalentes, les plus simples :

NP : ∃ **preuve courte** que la réponse est OUI (si la réponse est OUI) – certificat positif

coNP : ∃ **preuve courte** que la réponse est NON (si la réponse est NON) – certificat négatif

Preuve courte = **vérifiable** en temps polynomial

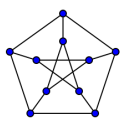
Observation : P ⊆ NP et P ⊆ coNP (l'algorithme lui-même permet de vérifier, sans autre certificat)

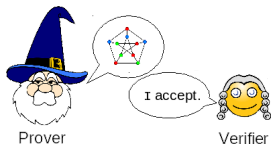


Quelques problèmes dans NP (présumément pas dans P) : 3-COLORING, CLIQUE, TSP, FACTORISATION, SAT, ...

Exemple : 3-COLORING

Ce graphe peut-il être colorié avec 3 couleurs ?





(certificat = la coloration elle-même)

Classes de problèmes NP et coNP

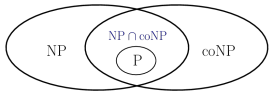
Plusieurs définitions équivalentes, les plus simples :

NP : ∃ **preuve courte** que la réponse est OUI (si la réponse est OUI) – certificat positif

coNP : ∃ **preuve courte** que la réponse est NON (si la réponse est NON) – certificat négatif

Preuve courte = **vérifiable** en temps polynomial

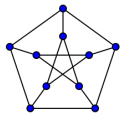
Observation : P ⊆ NP et P ⊆ coNP (l'algorithme lui-même permet de vérifier, sans autre certificat)

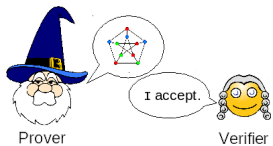


Quelques problèmes dans NP (présumément pas dans P) : 3-COLORING, CLIQUE, TSP, FACTORISATION, SAT, ...

Exemple : 3-COLORING

Ce graphe peut-il être colorié avec 3 couleurs ?



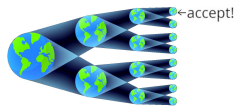


(certificat = la coloration elle-même)

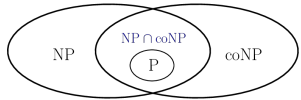
Définition historique

NP = Non-deterministic Polynomial time

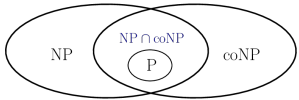
Intuition : capacité à "**deviner**" le certificat (qu'il suffit alors de vérifier).



Intermède (exercices)



Intermède (exercices)

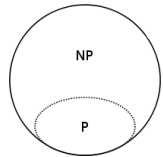


Classer les problèmes de décision suivants

- Entrée : Une liste de mot.
  Question : est-elle triée alphabétiquement ?
- Entrée : Un graphe.
  Question : est-il connexe ?
- Entrée : Un nombre N.
  Question : est-il premier ?
- Entrée : Deux nombres N et k.
  Question : N admet-il un facteur $\leq k$?
- Entrée : Deux graphes G_1 et G_2.
  Question : sont-ils isomorphes ?
- Entrée : Un graphe G et un entier k.
  Question : G contient-t-il un sous-graphe complet de taille k ?
- Entrée : Un plateau d'échecs de taille $N \times N$ (règles généralisées).
  Question : Y a-t-il une stratégie gagnante pour le joueur 1 ?
- Entrée : Un énoncé mathématique.
  Question : est-il vrai ?

P versus NP

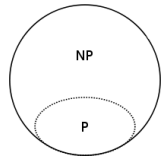
Les questions faciles à vérifier sont-elles faciles à résoudre ?
(Est-ce que P $=$ NP ?)



P versus NP

Les questions faciles à vérifier sont-elles faciles à résoudre ?

(Est-ce que $P = NP$?)



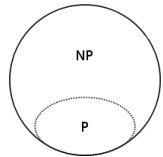
Importance de la question

- La majorité des problèmes informatiques de la vie courante sont dans NP.
  Si $P = NP$, on peut résoudre tous ces problèmes rapidement.
- Serait-ce une bonne nouvelle ? Oui et non (cryptographie).
- Un des 7 “problèmes du millénaire” (fondation Clay, $1 M / pb), au même titre que la conjecture de Riemann.

P versus NP

Les questions faciles à vérifier sont-elles faciles à résoudre ?

(Est-ce que P $=$ NP ?)



Importance de la question

- La majorité des problèmes informatiques de la vie courante sont dans NP.
  Si P $=$ NP, on peut résoudre tous ces problèmes rapidement.
- Serait-ce une bonne nouvelle ? Oui et non (cryptographie).
- Un des 7 “problèmes du millénaire” (fondation Clay, $1 M / pb), au même titre que la conjecture de Riemann.

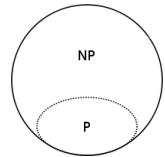
Portée philosophique ?

- Tout énoncé mathématique ayant une preuve humainement vérifiable peut-il être résolu “rapidement”.
- Les connaissances que l’on est capable de vérifier sont-elles “facilement” découvrable ?
- Je sais reconnaître une symphonie, donc j’aurais pu la composer moi-même ?
- L’intuition est-elle automatisable ?
- *etc.*

P versus NP

Les questions faciles à vérifier sont-elles faciles à résoudre ?

(Est-ce que P $=$ NP ?)



Importance de la question

- La majorité des problèmes informatiques de la vie courante sont dans NP.
  Si P $=$ NP, on peut résoudre tous ces problèmes rapidement.
- Serait-ce une bonne nouvelle ? Oui et non (cryptographie).
- Un des 7 “problèmes du millénaire” (fondation Clay, $1 M / pb), au même titre que la conjecture de Riemann.

Portée philosophique ?

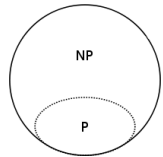
- Tout énoncé mathématique ayant une preuve humainement vérifiable peut-il être résolu “rapidement”.
- Les connaissances que l’on est capable de vérifier sont-elles “facilement” découvrable ?
- Je sais reconnaître une symphonie, donc j’aurais pu la composer moi-même ?
- L’intuition est-elle automatisable ?
- *etc.*

bémols : formalisation + quid de $O(n^{100})$?

P versus NP

Les questions faciles à vérifier sont-elles faciles à résoudre ?

(Est-ce que P $=$ NP ?)



Importance de la question

- La majorité des problèmes informatiques de la vie courante sont dans NP.
  Si P $=$ NP, on peut résoudre tous ces problèmes rapidement.
- Serait-ce une bonne nouvelle ? Oui et non (cryptographie).
- Un des 7 “problèmes du millénaire” (fondation Clay, $1 M / pb), au même titre que la conjecture de Riemann.

Portée philosophique ?

- Tout énoncé mathématique ayant une preuve humainement vérifiable peut-il être résolu “rapidement”.
- Les connaissances que l’on est capable de vérifier sont-elles “facilement” découvrable ?
- Je sais reconnaître une symphonie, donc j’aurais pu la composer moi-même ?
- L’intuition est-elle automatisable ?
- *etc.*

bémols : formalisation + quid de $O(n^{100})$?

Statut de la question ? À ce jour, on ne connaît pas la réponse.

L’écrasante majorité des experts pensent que P $\neq$ NP.

NP-complétude

Difficulté et complétude

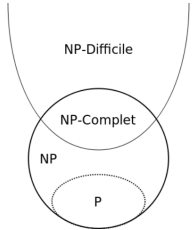
- NP-difficile : problèmes qui sont **au moins aussi difficile** que tout problème de NP
  Cela se montre par des **réductions** entre problèmes.
- NP-complet : à la fois dans NP et NP-difficile

NP-complétude

Difficulté et complétude

- NP-difficile : problèmes qui sont **au moins aussi difficile** que tout problème de NP
  Cela se montre par des **réductions** entre problèmes.
- NP-complet : à la fois dans NP et NP-difficile

Comment montrer qu'un problème est NP-difficile ?
→ trouver un problème déjà NP-difficile et le réduire à ce problème.

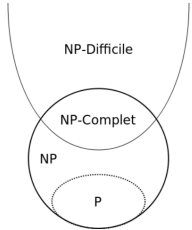


NP-complétude

Difficulté et complétude

- NP-difficile : problèmes qui sont **au moins aussi difficile** que tout problème de NP
  Cela se montre par des **réductions** entre problèmes.
- NP-complet : à la fois dans NP et NP-difficile

Comment montrer qu'un problème est NP-difficile ?
→ trouver un problème déjà NP-difficile et le réduire à ce problème.



Exemples de problèmes NP-complets

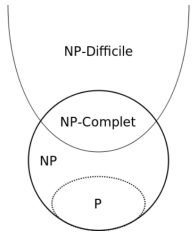
- SAT : $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \vee \ldots$ (Cook, Levin'71)
- 3-Coloring, Clique, Set Cover, Hamiltonian Cycle, Tsp,
- Des milliers d'autres problèmes ...

NP-complétude

Difficulté et complétude

- NP-difficile : problèmes qui sont **au moins aussi difficile** que tout problème de NP
  Cela se montre par des **réductions** entre problèmes.
- NP-complet : à la fois dans NP et NP-difficile

Comment montrer qu'un problème est NP-difficile ?
$\rightarrow$ trouver un problème déjà NP-difficile et le réduire à ce problème.



Exemples de problèmes NP-complets

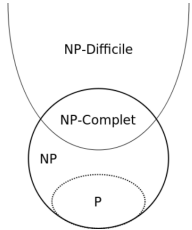
- SAT : $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \vee \ldots$ (Cook, Levin'71)
- 3-Coloring, Clique, Set Cover, Hamiltonian Cycle, Tsp,
- Des milliers d'autres problèmes ...

Si **un seul** de ces problèmes est résoluble en temps polynomial, alors ils le sont **tous** et P $=$ NP.
Si un seul de ces problèmes requiert un temps super-polynomial, alors P $\neq$ NP.

NP-complétude

Difficulté et complétude

- NP-difficile : problèmes qui sont **au moins aussi difficile** que tout problème de NP
  Cela se montre par des **réductions** entre problèmes.
- NP-complet : à la fois dans NP et NP-difficile



Comment montrer qu'un problème est NP-difficile ?
→ trouver un problème déjà NP-difficile et le réduire à ce problème.

Exemples de problèmes NP-complets

- SAT : $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \vee \ldots$ (Cook, Levin'71)
- 3-COLORING, CLIQUE, SET COVER, HAMILTONIAN CYCLE, TSP,
- Des milliers d'autres problèmes ...

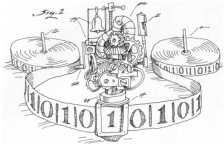
Si **un seul** de ces problèmes est résoluble en temps polynomial, alors ils le sont **tous** et P $=$ NP.
Si un seul de ces problèmes requiert un temps super-polynomial, alors P $\neq$ NP.

Rappel important

Cette théorie s'intéresse au **pire cas**, c'est à dire aux instances les plus dures à résoudre. Beaucoup de ces problèmes sont résoluble dans certains cas pratiques.

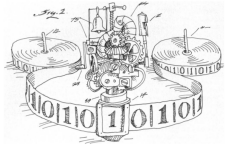
Les instances issues du monde réel sont souvent sympatiques.

Et l'IA dans tout ça ?



V.S.

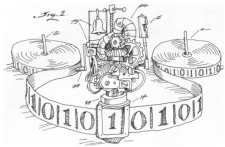
Et l'IA dans tout ça ?



V.S.

Tous les résultats évoquées s'appliquent à l'IA, sans distinction.

Et l'IA dans tout ça ?

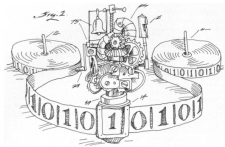


V.S.

Tous les résultats évoquées s'appliquent à l'IA, sans distinction.

De nombreux problèmes, y compris NP-complets, sont **faciles dans le cas moyen**, l'IA (apprentissage profond) peut en théorie les résoudre sur les instances typiques.

Et l'IA dans tout ça ?



V.S.

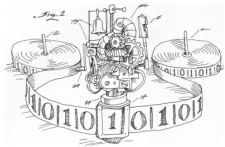
Tous les résultats évoquées s'appliquent à l'IA, sans distinction.

De nombreux problèmes, y compris NP-complets, sont **faciles dans le cas moyen**, l'IA (apprentissage profond) peut en théorie les résoudre sur les instances typiques.

Elle peut aussi trouver de meilleurs algorithmes que nous.

Aucun résultat théorique n'exclut que l'IA surpasse les algorithmiciens ! :-)

Et l'IA dans tout ça ?



V.S.

Tous les résultats évoquées s'appliquent à l'IA, sans distinction.

De nombreux problèmes, y compris NP-complets, sont **faciles dans le cas moyen**, l'IA (apprentissage profond) peut en théorie les résoudre sur les instances typiques.

Elle peut aussi trouver de meilleurs algorithmes que nous.

Aucun résultat théorique n'exclut que l'IA surpasse les algorithmiciens ! :-)
Les problèmes qui sont **difficiles dans le cas moyen** resteront inaccessibles pour l'IA. Présumément, FACTORING en fait partie.

Et le quantique ?

BPP : *Bounded-error probabilistic polynomial time*

- Problèmes résolubles en temps polynomial par un **algorithme probabiliste** (peut tirer à pile ou face) avec une probabilité d'erreur strictement inférieure à 1/2

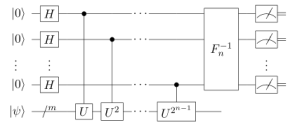
Et le quantique ?

BPP : *Bounded-error probabilistic polynomial time*

- Problèmes résolubles en temps polynomial par un **algorithme probabiliste** (peut tirer à pile ou face) avec une probabilité d'erreur strictement inférieure à 1/2

BQP : *Bounded error quantum polynomial time*

- Problèmes résolubles en temps polynomial par un **ordinateur quantique** avec une probabilité d'erreur strictement inférieure à 1/2



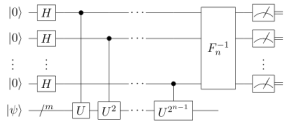
Et le quantique ?

BPP : *Bounded-error probabilistic polynomial time*

- Problèmes résolubles en temps polynomial par un **algorithme probabiliste** (peut tirer à pile ou face) avec une probabilité d'erreur strictement inférieure à 1/2

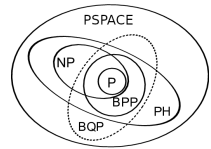
BQP : *Bounded error quantum polynomial time*

- Problèmes résolubles en temps polynomial par un **ordinateur quantique** avec une probabilité d'erreur strictement inférieure à 1/2



Que sait-on ?

- P $\subseteq$ BPP — $(0 < 1/2)$
- BPP $\subseteq$ BQP — (non-réversibilité simulable en temps poly)
- BQP $\subseteq$ PSPACE — (Bernstein et Vazirani'97)
- FACTORING $\in$ BQP — (Shor'94)
- Quid de BQP *versus* NP ? — (pressenties incomparables)



(structure pressentie)

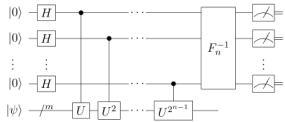
Et le quantique ?

BPP : *Bounded-error probabilistic polynomial time*

- Problèmes résolubles en temps polynomial par un **algorithme probabiliste** (peut tirer à pile ou face) avec une probabilité d'erreur strictement inférieure à 1/2

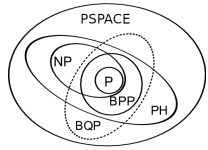
BQP : *Bounded error quantum polynomial time*

- Problèmes résolubles en temps polynomial par un **ordinateur quantique** avec une probabilité d'erreur strictement inférieure à 1/2



Que sait-on ?

- P $\subseteq$ BPP $(0 < 1/2)$
- BPP $\subseteq$ BQP (non-réversibilité simulable en temps poly)
- BQP $\subseteq$ PSPACE (Bernstein et Vazirani'97)
- FACTORING $\in$ BQP (Shor'94)
- Quid de BQP *versus* NP ? (pressenties incomparables)



(structure pressentie)

Un ordinateur quantique peut-il résoudre des problèmes NP-complets ?

$\rightarrow$ Jugé peu probable par les spécialistes.

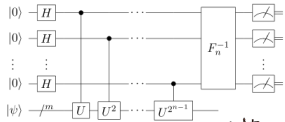
Et le quantique ?

BPP : *Bounded-error probabilistic polynomial time*

- Problèmes résolubles en temps polynomial par un **algorithme probabiliste** (peut tirer à pile ou face) avec une probabilité d'erreur strictement inférieure à 1/2

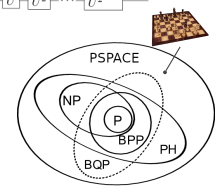
BQP : *Bounded error quantum polynomial time*

- Problèmes résolubles en temps polynomial par un **ordinateur quantique** avec une probabilité d'erreur strictement inférieure à 1/2



Que sait-on ?

- P $\subseteq$ BPP $(0 < 1/2)$
- BPP $\subseteq$ BQP (non-réversibilité simulable en temps poly)
- BQP $\subseteq$ PSPACE (Bernstein et Vazirani'97)
- FACTORING $\in$ BQP (Shor'94)
- Quid de BQP *versus* NP ? (pressenties incomparables)



(structure pressentie)

Un ordinateur quantique peut-il résoudre des problèmes NP-complets ?

$\rightarrow$ Jugé peu probable par les spécialistes.

Conclusion

L'informatique peut-elle tout faire ?

Conclusion

L'informatique peut-elle tout faire ?

- Problèmes indécidables
- Problèmes dont la complexité est trop importante
- L'IA hérite des mêmes limitations
- Le quantique va possiblement plus loin (mais a des limitations aussi)
- De nombreux problèmes (mêmes difficiles) sont résolubles dans le cas moyen

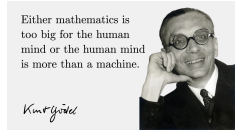
Conclusion

L'informatique peut-elle tout faire ?

- Problèmes indécidables
- Problèmes dont la complexité est trop importante
- L'IA hérite des mêmes limitations
- Le quantique va possiblement plus loin (mais a des limitations aussi)
- De nombreux problèmes (mêmes difficiles) sont résolubles dans le cas moyen

Autres pistes de discussions

- Les humains sont-ils soumis à la thèse de Church-Turing ?
- Peuvent-ils puiser dans d'autres sources ? (machines à oracles / théorie de Penrose)
- Ces résultats sont plausiblement les mêmes à l'autre bout de l'univers



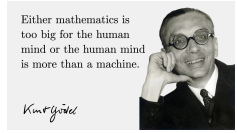
Conclusion

L'informatique peut-elle tout faire ?

- Problèmes indécidables
- Problèmes dont la complexité est trop importante
- L'IA hérite des mêmes limitations
- Le quantique va possiblement plus loin (mais a des limitations aussi)
- De nombreux problèmes (mêmes difficiles) sont résolubles dans le cas moyen

Autres pistes de discussions

- Les humains sont-ils soumis à la thèse de Church-Turing ?
- Peuvent-ils puiser dans d'autres sources ? (machines à oracles / théorie de Penrose)
- Ces résultats sont plausiblement les mêmes à l'autre bout de l'univers



Merci !

Backup slides...

Chercher une démonstration

Vérifier une démonstration est “facile” :
il suffit de contrôler chaque étape.

Donc « existe-t-il une démonstration de taille $\leq n$? »
est un problème de **NP**.

Si **P** $=$ **NP**, **trouver** une démonstration
n’est pas significativement plus difficile que de la vérifier.

Chercher une démonstration

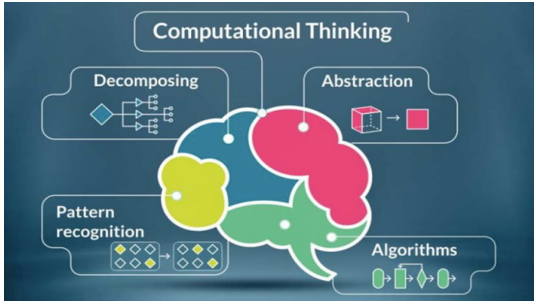
Vérifier une démonstration est “facile” :
il suffit de contrôler chaque étape.

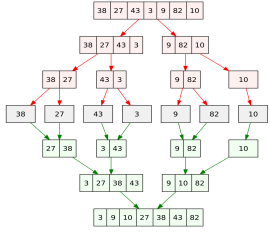
Donc « existe-t-il une démonstration de taille $\leq n$? »
est un problème de **NP**.

Si **P** $=$ **NP**, **trouver** une démonstration
n’est pas significativement plus difficile que de la vérifier.

$$L_{maths} = \{(S, \underbrace{11\cdots1}_{n \text{ times}}) \mid S \text{ has a proof of length at most } n \text{ in ZFC}\}.$$

La pensée algorithmique





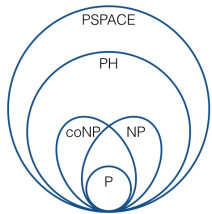
Machines à oracles et hiérarchie polynomiale

Classes définies par oracles

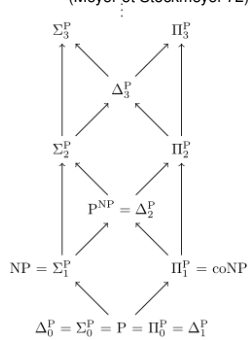
- Soient deux classes X et Y, on note X^Y la classe des problèmes résolubles par un algorithme de X ayant accès à un oracle pour un problème Y-complet.
- Par exemple, P^{NP} = problèmes résolubles en temps polynomial avec un oracle pour SAT.
- Les oracles ne sont pas forcément des pbs de décisions, par exemple, #P (dénombrement).

Hiérarchie polynomiale (PH)

- Principe : Itération d'oracles impliquant P, NP et coNP
- Problèmes complets à chaque étage
  Quantified SAT (QSAT) :
  $\exists_{x1} \forall_{x_2} \exists_{x_3} \ldots, \phi(x_1, x_2, x_3, \ldots)$
- Machine de Turing alternantes
- PH $\subseteq$ $\mathrm{P}^{\#\mathrm{P}}$ (Toda'91)
- PH $\stackrel{?}{=}$ PSPACE



(Meyer et Stockmeyer'72)



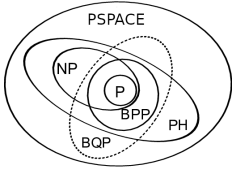
Séparation par oracle

Principe : on veut montrer que deux classes sont différentes, mais on y arrive pas.

→ Essayer de montrer au moins qu'elles sont différentes **relativement** à un certain oracle.

Exemples de séparations

- Il existe un oracle X tel que $P^X \neq NP^X$ (Baker, Gill, Solovay'75)
  (cependant : il existe aussi Y tel que $P^Y = NP^Y$) ("Relativization barrier")
- $\exists X$ tel que NP$^X \subsetneq$ BQP^X (Bennett, Bernstein, Brassard, Vazirani'97)
- $\exists X$ tel que BQP$^X \subsetneq$ NP^X (Berthiaume, Brassard'92)
- $\exists X$ tel que BQP$^X \subsetneq$ PH^X (Raz, Tal'19)



(attention ↑ conjectures)

Remarque

N'importe quelle séparation non-relative impliquerait, à minima, que P $\neq$ PSPACE, ce qui est un des principaux problèmes ouverts du domaine.

Deux tâches très inégales

Prouver P $=$ NP

Exhiber **un** algorithme polynomial pour **un** problème NP-complet.

Une construction. On sait faire ce genre de chose.

Prouver P $\neq$ NP

Montrer qu'**aucun** algorithme polynomial n'existe.

Une **borne inférieure**. On ne sait presque rien démontrer de ce genre.

Et c'est du côté droit que penche la conviction générale.

L'état réel des bornes inférieures

Pour prouver **P** $\neq$ **NP**, il faudrait une borne **superpolynomiale**.

Les meilleures bornes connues, sur des modèles de calcul généraux, sont **linéaires** : de l'ordre de quelques n.

L'écart entre ce qu'on sait démontrer
et ce qu'il faudrait démontrer est **immense**.

Ce n'est pas qu'on approche du but sans y arriver : on n'a pas commencé.

Nuance : « NP-complet » n'est pas « impossible »

- La NP-complétude est une propriété du **pire cas**.
- Les solveurs SAT industriels traitent couramment des instances à des **millions de variables** : vérification de circuits, planification, preuve automatique.
- Les instances **qui se présentent** ont une structure ; les instances **difficiles** sont rares et souvent artificielles.

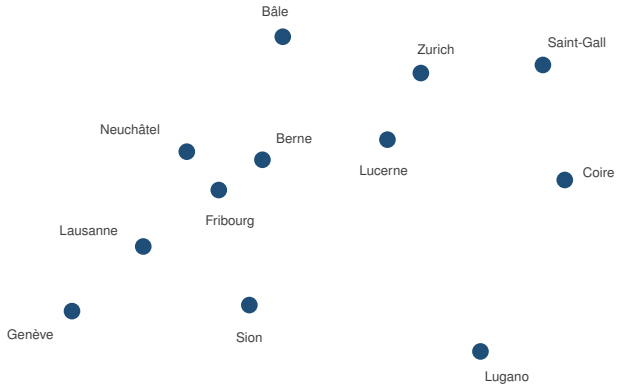
Votre secrétariat **trouve** le planning des examens.
Ce n'est pas une réfutation de la théorie.

Nuance : contourner l'obstacle

- **Approximation** : renoncer à l'optimum, garantir qu'on n'en est pas loin.
- **Heuristiques** : pas de garantie, mais ça marche en pratique.
- **Paramétrer** : isoler ce qui rend l'instance difficile et exploiter que ce paramètre reste petit.
- **Restreindre** : sur certaines familles de graphes, le problème redevient facile.

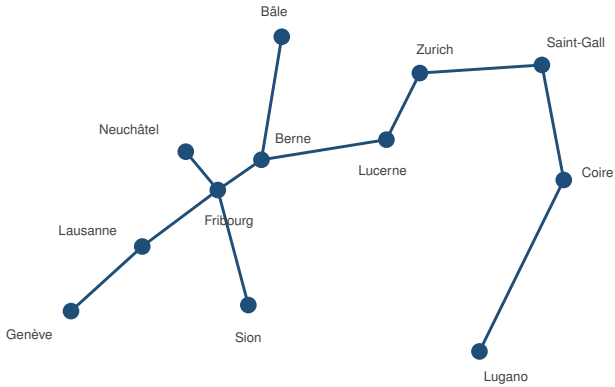
Illustrons la première — avec les deux problèmes du début.

Exemple : voyageur de commerce “metric”



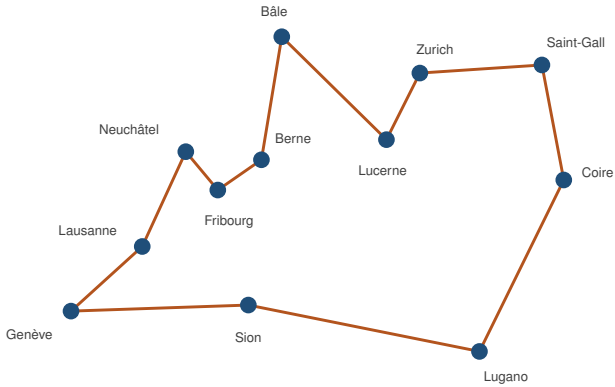
Deux questions qui se ressemblent : **tout relier au moindre coût** · **faire une tournée passant partout**.

Tout relier au moindre coût — facile



Arbre couvrant de poids minimum. Résoluble par un algorithme glouton (Kruskal ou Prim). Optimal, prouvé, en temps polynomial.

Faire une tournée — difficile



Voyageur de commerce. Ici l'optimum a été calculé exactement — 12 villes, c'est encore faisable.
À 30 villes, plus personne ne garantit l'optimum par force brute.

L'algo de 2-approximation (en temps polynomial)

1. Construire l'**arbre couvrant minimum** (facile).
2. Le **parcourir en doublant chaque arête** : on revient au départ en passant partout, pour un coût de $2 \times$ ACM.
3. **Raccourcir** : sauter les villes déjà visitées. L'inégalité triangulaire garantit qu'on ne perd rien.

$$\text{Tournée obtenue} \leq \quad 2 \times \text{ ACM} \quad \leq \quad 2\times \text{ tournée optimale.}$$

Une garantie à un facteur 2, en temps polynomial.

L'ACM est plus court que la tournée optimale : en retirer une arête donne un arbre couvrant.

On peut faire encore mieux : 1.5 fois l'optimum (Christofides, 1971)

Trois barrières

On a démontré que **des familles entières de méthodes**
ne peuvent pas trancher la question.

C'est une forme de progrès inhabituelle : on n'avance pas vers la réponse, on élimine des chemins.

1975 — la relativisation

Baker, Gill et Solovay.

- On peut munir les machines d'un **oracle** : une boîte noire qui répond instantanément à un problème fixé. Cela donne des classes $\mathbf{P}^A$ et $\mathbf{NP}^A$.
- Il existe un oracle A tel que $\mathbf{P}^A = \mathbf{NP}^A$.
- Il existe un oracle B tel que $\mathbf{P}^B \neq \mathbf{NP}^B$.

Toute preuve qui **resterait valable en présence d'un oracle** — c'est le cas des arguments de diagonalisation, hérités de Cantor et de Turing — ne peut donc pas trancher.

1994 — les preuves naturelles

Razborov et Rudich.

Presque toutes les bornes inférieures connues suivent le même schéma : on exhibe une **propriété** que possèdent les fonctions faciles et que la fonction visée ne possède pas.

Razborov et Rudich montrent que si une telle propriété est **assez générale** et **assez facile à tester** — ce qui est le cas de toutes celles qu'on sait manier — alors elle permettrait de **casser la cryptographie**.

Autrement dit : ou bien les fonctions à sens unique n'existent pas, ou bien cette méthode est sans issue.

Alexander Razborov (1963–)

Prix Gödel 2007.

2008 — l'algébrisation

Avi Wigderson (1956–)

Aaronson et Wigderson.

Dans les années 1990, quelques résultats avaient franchi la barrière de 1975 grâce à des techniques **algébriques** : on remplace une formule booléenne par un polynôme sur un corps fini.

Aaronson et Wigderson montrent que ces techniques se heurtent à leur tour à une barrière du même type, un cran plus haut.

Il faut une idée qui n'appartienne à aucune de ces trois familles. Personne n'en a.

Wigderson : prix Abel 2021, prix Turing 2023.