Information, Calcul et Communication

CS-119(k) ICC – Théorie
Semaine 5

Rafael Pires
rafael.pires@epfl.ch

Lausanne, 21.03.2025

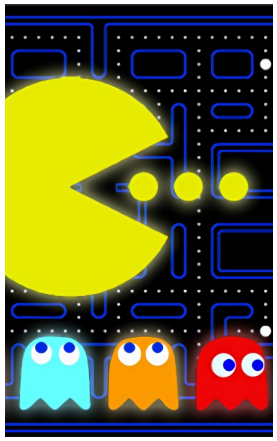
Evaluation du corps estudiantin

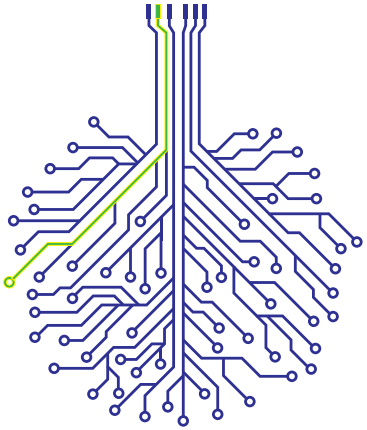
- **Évaluation indicative sur l'enseignement** (jusqu'au dimanche 23.03)
  - Une question sur le déroulement du cours
  - Commentaire
- Évaluation approfondie à la fin du semestre

- Donnez un retour sur le cours sur IS-Academia
  - Feedback anonyme
  - Dites ce qui vous a plu, ce qui vous a déplu;

    ce qui vous convient, ce qui ne vous convient pas
  - Soyez constructifs : c'est votre opportunité d'aider à améliorer le cours durant les 9 semaines à venir. Je lirai tous vos retours et en tiendrai compte.

Précédemment, dans ICC-T 04





- Algorithmes gloutons
- Programmation dynamique
  - Rendu de pièces de monnaie
- Memoïsation
  - Suite de Fibonacci
- Théorie de la calculabilité : décidabilité
  - Le problème de l'arrêt

EPFL

Autres problèmes indécidables

- Un algorithme contient-il un morceau de code inutile ?
- Deux algorithmes calculent-ils la même chose ?
- Peut-on paver le plan sans recouvrement ni espace vide avec un ensemble de formes géométriques ?
- Est-ce qu'une formule mathématique exprimée dans l'arithmétique est vraie dans tous les cas possibles ?

ICC-T 04 : Le problème de l'arrêt

- Supposons, par hypothèse, qu'un tel algorithme **A** existe, c'est-à-dire :
  - **A(P, X)** sort **oui** si **P(X)** s'arrête
  - **A(P, X)** sort **non** si **P(X)** continue indéfiniment

- **Instance**

- **Problème général**

P

entrée : entier **x**
sortie : aucune

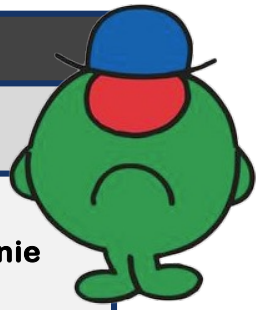
pas ← 0
Tant que pas ≠ x :
    pas ← pas + 1

- **A(P, 5) : oui**
- **A(P, -2) : non**

M. Non

entrée : algorithme **P**
sortie : aucune

Si A(P, P) = oui, alors :
    Effectue une boucle infinie
Sinon
    S'arrêter



Vocabulaire

- **Problème** : une question générale, dont les entrées et critères de réussite sont définis.
  - Question : Étant donné un entier n, est-il non premier ?
  - Critère de réussite : Répondre "oui" si un diviseur d non-trivial de n existe, "non" sinon.
- **Instance** : donnée spécifique à un problème donné. C'est une réalisation concrète du problème.
  - Vérifier si 221 est non premier.
- **Algorithme** : La méthode qui, à partir de l'instance, produit la solution.

Non premier

entrée : entier **n**
sortie : oui / non

Si $n \leq 1$: **Sortir oui**
Si $n = 2$ **ou** $n = 3$:
    Sortir non
Pour d **allant de** 2 **à** $\lceil \sqrt{n} \rceil$:
    Si mod(n,d) = 0: **Sortir oui**
Sortir non

- **Solution** : réponse correcte à une instance d'un problème.
  - Instance : Vérifier si 221 est non premier.
  - Solution : Oui

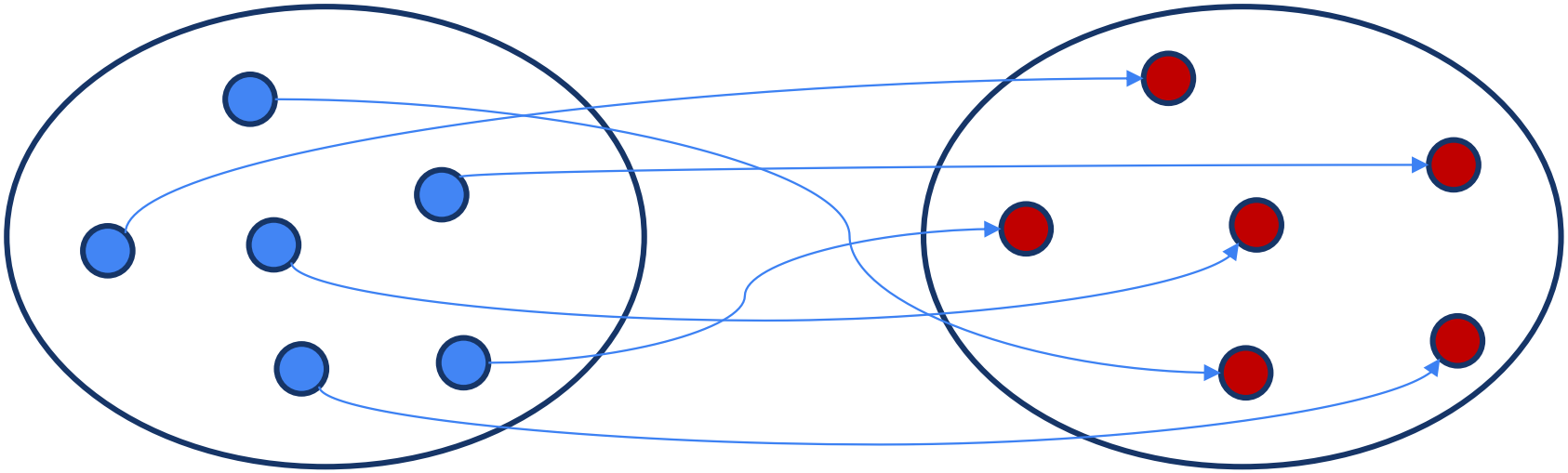
Aujourd'hui

- Dénombrement
- P : Résoudre en temps polynomial
- NP : Vérifier en temps polynomial
- Problèmes d'optimisation discrète

Thanks to J.-C. Chappelier pour une partie du contenu.

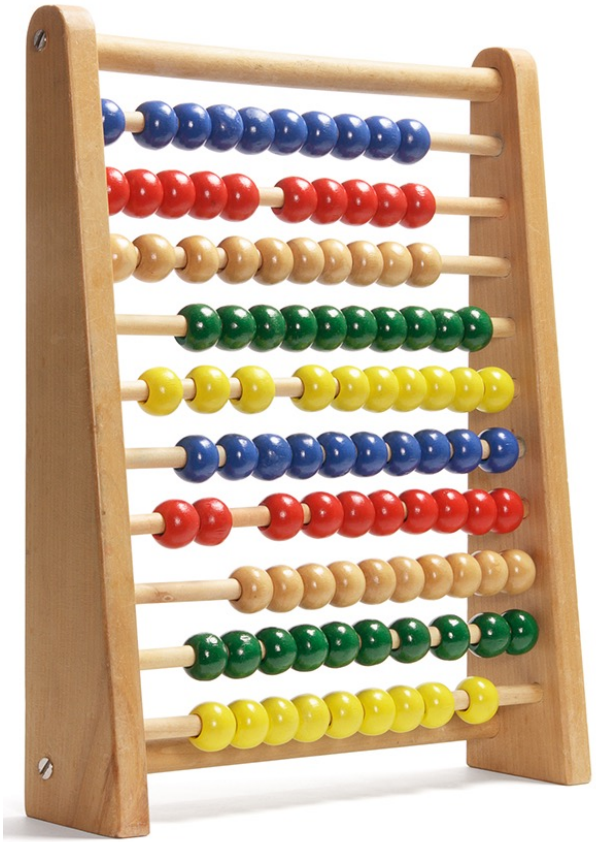
Limites des algorithmes

- A = Ensemble de tous les algorithmes définissables.
- B = Ensemble de toutes les sorties possibles.



- Existe-t-il une fonction définissable (via un algorithme) pour calculer chaque élément de B ?

Dénombrement : Apprendre à compter



- Comment compter un ensemble infini ?

Un ensemble S est dénombrable si et seulement s'il existe une surjection $f: \mathbb{N}^* \to S$.

La fonction f fait le numérotage de S.

Rappel :

Soit $f: E \to F$ une fonction entre deux ensembles, la fonction f est surjective si tout élément de F est atteint par au moins un élément de E. Formellement,

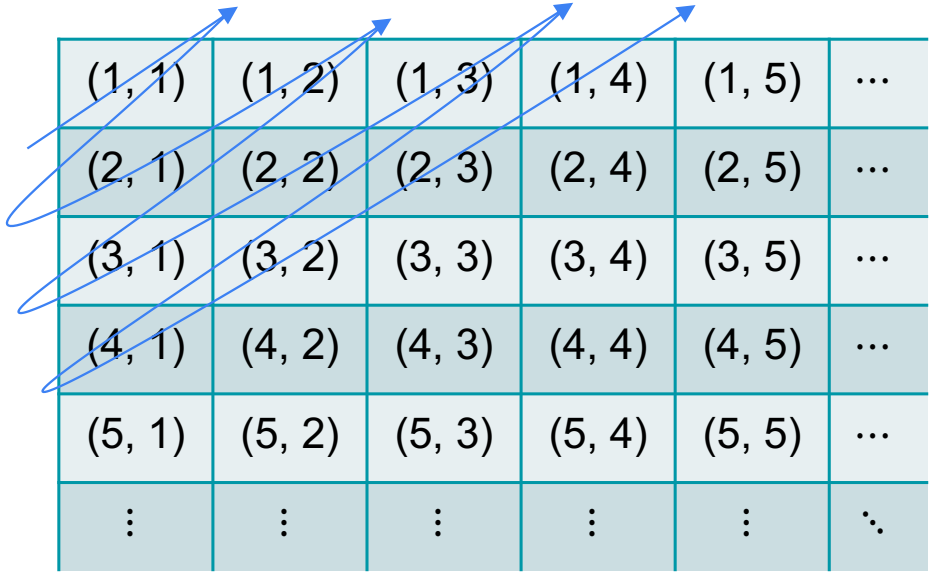
$$\forall \boldsymbol{y} \in \boldsymbol{F}, \exists \boldsymbol{x} \text{ tel que } \boldsymbol{f(x) = y}$$

- Exemple : Les entiers, $\mathbb{Z} = \{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\}$, forment un ensemble dénombrable parce qu'on peut prendre la fonction $f: \mathbb{N}^* \to \mathbb{Z}$.

$$f(x) = \begin{cases} \dfrac{x}{2}, si\ x\ est\ pair \\ -\dfrac{x-1}{2}, si\ x\ est\ impair \end{cases}$$

Dénombrement : Paires d'entiers positifs

- **Exemple :** Les paires d'entiers $f: \mathbb{N}^* \to \mathbb{N}^* \times \mathbb{N}^*$



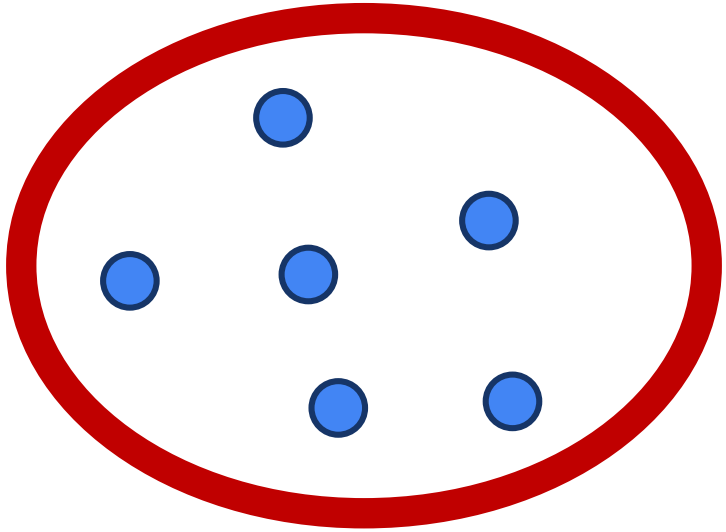
(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	⋯
(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	⋯
(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	⋯
(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	⋯
(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	⋯
⋮	⋮	⋮	⋮	⋮	⋱

$$f(0) = (1,1)$$
$$f(1) = (2,1)$$
$$f(2) = (1,2)$$
$$f(3) = (3,1)$$
$$f(4) = (2,2)$$
$$f(5) = (1,3)$$
$$\vdots$$

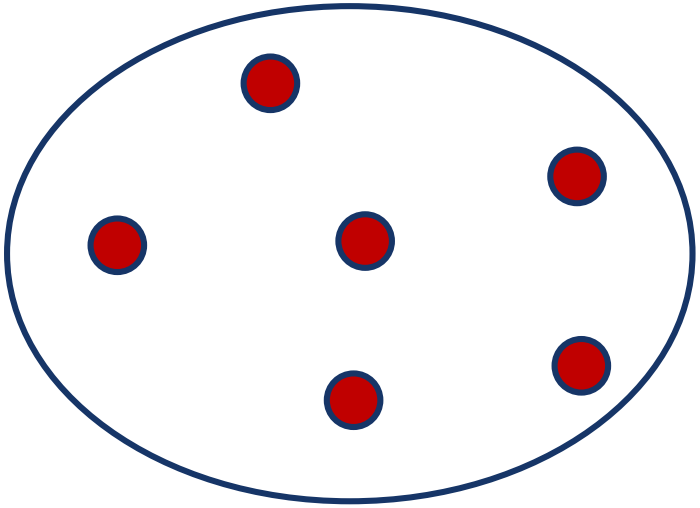
- Chaque paire reçoit son numéro unique – nous avons défini une surjection.
  (Cela prouve également que $\mathbb{Q}$ est dénombrable)

Limites des algorithmes

- A = Ensemble de tous les algorithmes définissables.



- B = Ensemble de toutes les sorties possibles.



- Existe-t-il une fonction définissable (via un algorithme) pour calculer chaque élément de B ?

Dénombrement et les algorithmes

- L'ensemble de tous algorithmes définissables, est-il dénombrable ?
  - Un algorithme est simplement un texte écrit à l'aide d'un alphabet choisi.
    Il est facile de numéroter tous les textes possibles dans un ordre lexicographique :
  - Commençons par énumérer les textes d'un seul caractère :

1	2	3	…	25	26
a	b	c	…	y	z

  - Continuons avec les textes de deux caractères :

27	28	29	…	701	702
aa	ab	ac	…	zy	zz

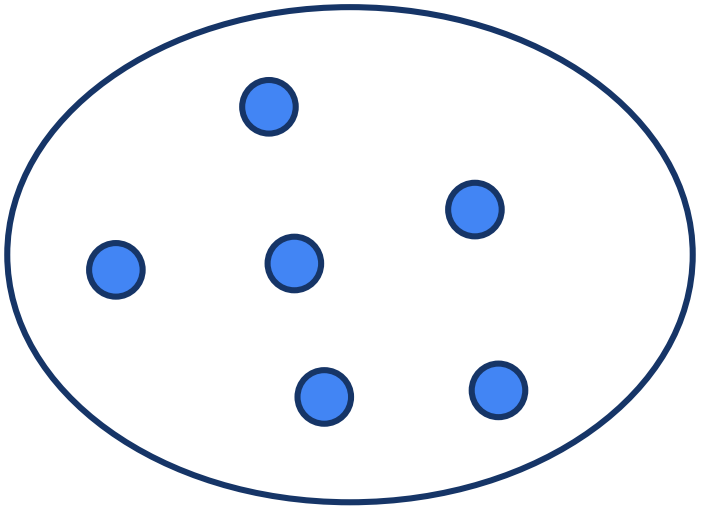
  - Maintenant passons aux textes de trois caractères :

703	704	705	…	18'277	18'278
aaa	aab	aac	…	zzy	zzz

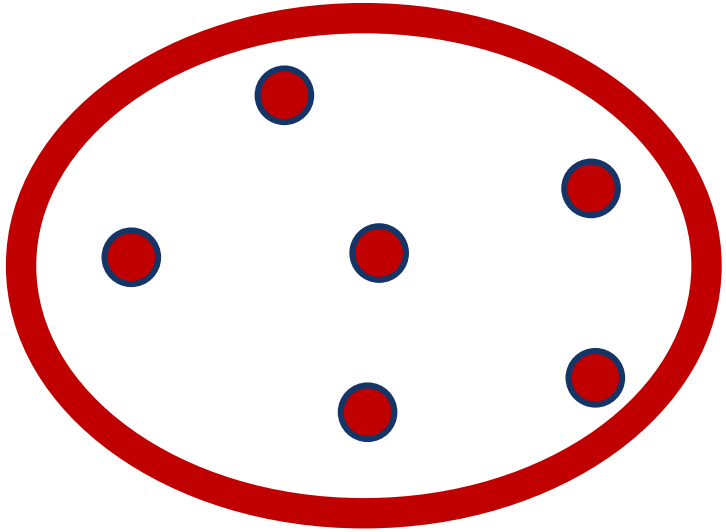
Certains des textes énumérés ne sont pas des algorithmes, mais
tous les algorithmes sont énumérés.

Limites des algorithmes

- A = Ensemble de tous les algorithmes définissables.



- B = Ensemble de toutes les sorties possibles.



- Existe-t-il une fonction définissable (via un algorithme) pour calculer chaque élément de B ?

Diagonale de Cantor : Les fonctions booléennes

- Soit l'ensemble des fonctions booléennes d'une variable entière $B = \{f | f : \mathbb{Z}^* \to 0, 1\}$
  - Exemple :

$$f(n) = \begin{cases} 1, \text{si } n \text{ est pair} \\ 0, \text{si } n \text{ est impair} \end{cases} = (0,1,0,1,0, \cdots)$$

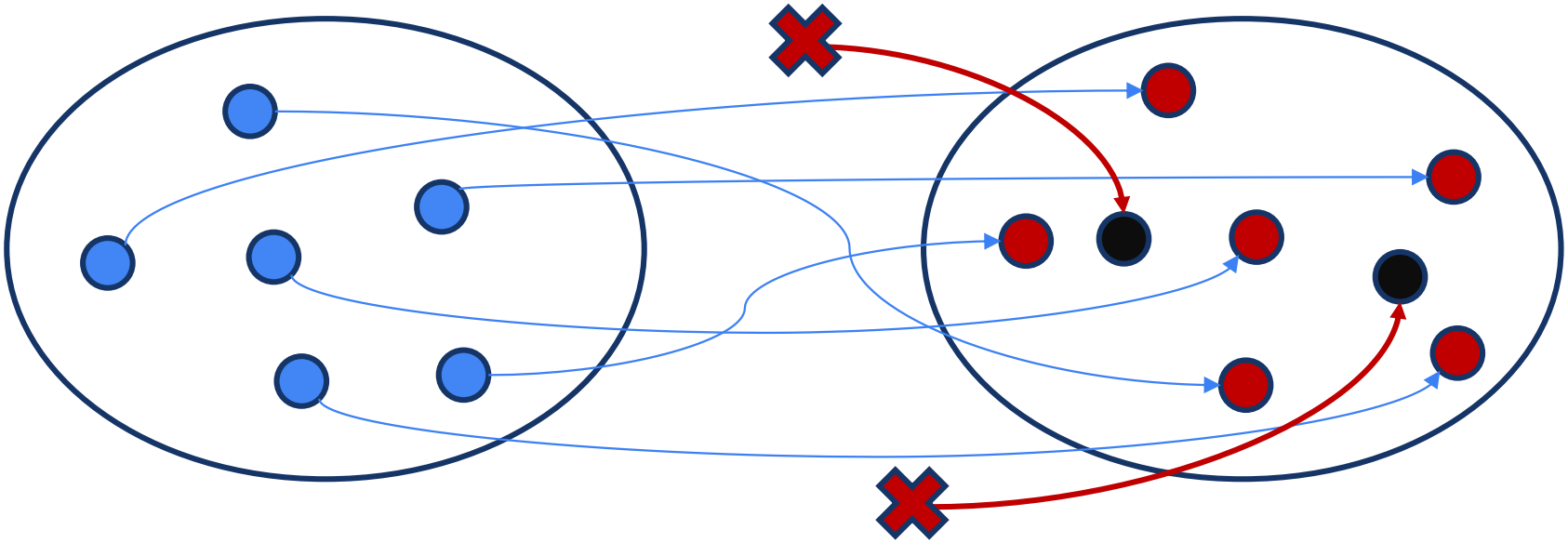
- Si l'ensemble B était dénombrable, on pourrait numéroter les fonctions : $f_1, f_2, f_3, f_4, f_5, \cdots$

n	1	2	3	4	5	⋯
f_1	**0**	1	1	0	0	⋯
f_2	0	**0**	0	0	1	⋯
f_3	1	1	**0**	0	1	⋯
f_4	0	1	0	**1**	0	⋯
f_5	1	0	1	0	**1**	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋱

1. Observons la diagonale : c'est une liste de valeurs binaires, donc une fonction $f \in B$
2. On peut la définir $f(n) = f_n(n)$
3. Définissons une nouvelle fonction booléenne $f_*(n) = 1 - f_n(n) = (1,1,1,0,0, \cdots)$, donc encore une fonction $f_* \in B$
4. Mais f_* n'apparaît pas dans l'énumération !
5. Donc B n'est pas dénombrable

Limites des algorithmes

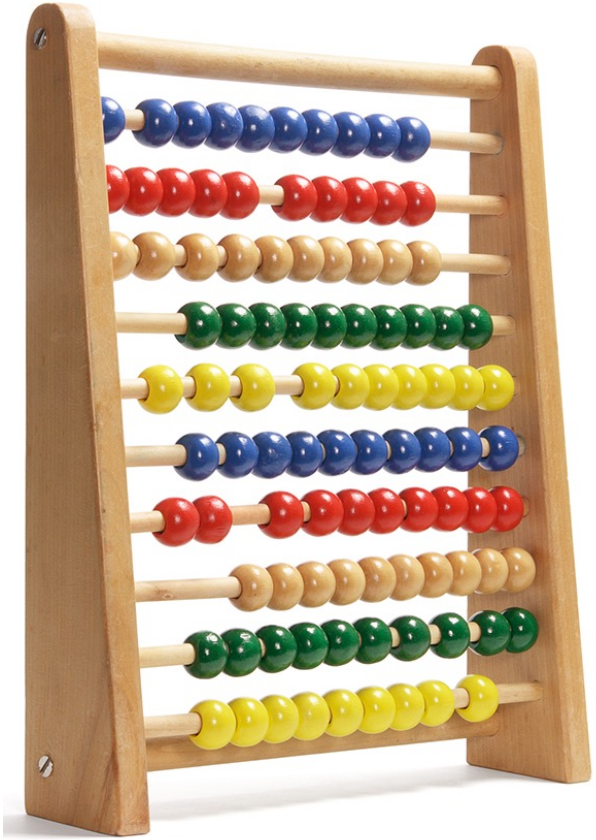
- A = Ensemble de tous les algorithmes définissables.
- B = Ensemble de toutes les sorties possibles.



- Existe-t-il une fonction définissable (via un algorithme) pour calculer chaque élément de B ?

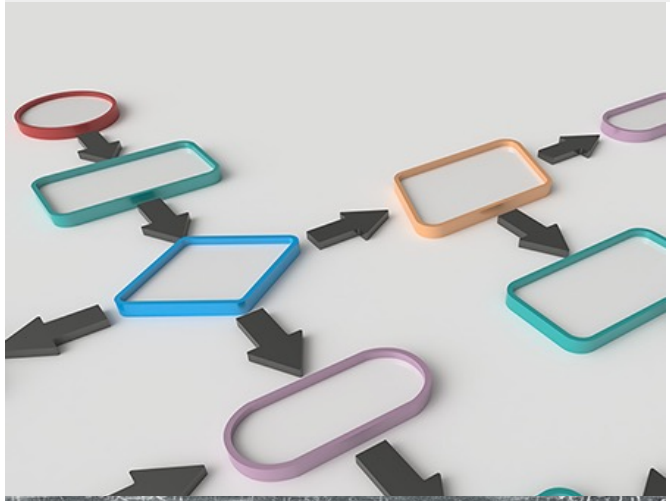
Non

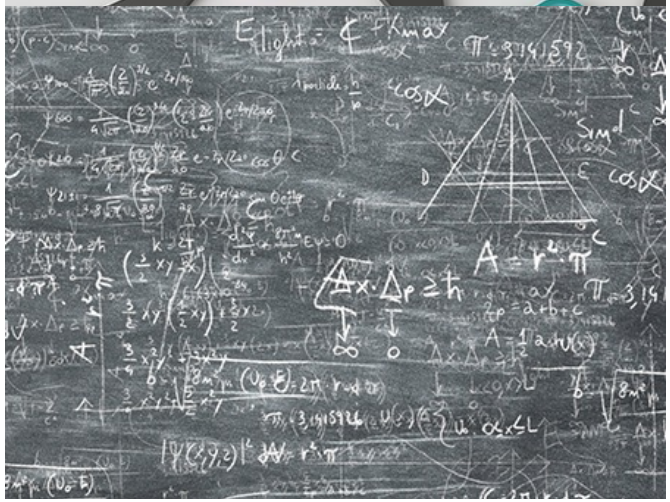
Dénombrement : l'essentiel



- L'ensemble de tous les algorithmes (quel que soit le formalisme) est dénombrable.
- L'ensemble de fonctions booléennes d'une variable entière n'est pas dénombrable.
- Donc, il existe beaucoup plus de fonctions booléennes que d'algorithmes.
- Donc, la plupart des fonctions booléennes ne peuvent pas être calculées par un algorithme. Il n'y a simplement pas assez d'algorithmes possibles.

Algorithmes vs. problèmes



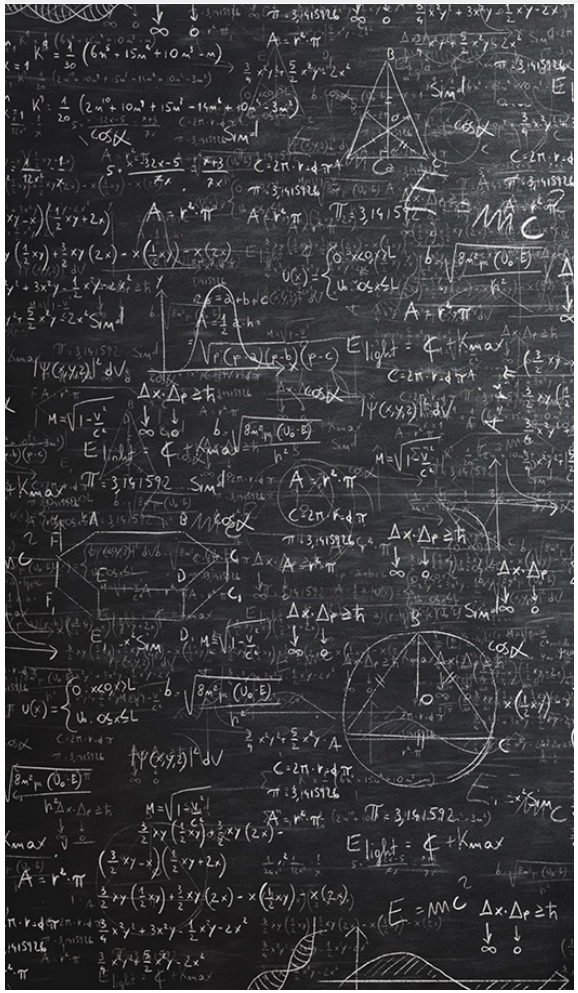


1. **Complexité temporelle d'un algorithme** : nombre d'opérations élémentaires effectuées par celui-ci, dans le pire des cas

2. **Problème** : ensemble de questions, solubles (ou non) par un algorithme (ensemble infini → problème intéressant!)

Aujourd'hui, nous allons nous intéresser aux **classes de complexité des problèmes**.

Classes de complexité des problèmes



- Tout problème est-il soluble par un algorithme ?
  - Réponse : Non

- Les problèmes solubles le sont-ils tous en un temps raisonnable ?
  - Réponse : Encore non !

Les classes de complexité des problèmes nous aident à identifier quels problèmes sont faciles à résoudre et quels problèmes le sont moins.

Classification des problèmes

❖ **Décidables**

- Recherche dans une liste
- Tri
- Le problème du voyageur du commerce

❖ **Indécidables**

- Problème de l'arrêt
- Problème de l'équivalence des programmes
- Problème du pavage
- Problème de la validité en arithmétique

Aujourd’hui

- Dénombrement
- P : Résoudre en temps polynomial
- NP : Vérifier en temps polynomial
- Problèmes d’optimisation discrète

ICC-T 02 : Notation $\Theta(\cdot)$: définition

- Dans de nombreuses applications, on a affaire à des données d'entrée de grande taille.
- Dans ce cas, on aimerait obtenir des ordres de grandeur plutôt que de devoir faire des calculs détaillés.

> Soient $f, g : \mathbb{N} \to \mathbb{R}_+$ deux fonctions non-négatives
> On dit que "$f(n)$ est un grand theta de $g(n)$" et on écrit "$f(n) = \Theta(g(n))$"
> s'il existe $0 < C1 < C2 < \infty$ et $N \geq 1$ tels que
>
> $$C1\ g(n) \leq f(n) \leq C2\ g(n) \quad \text{pour tout } n \geq N$$

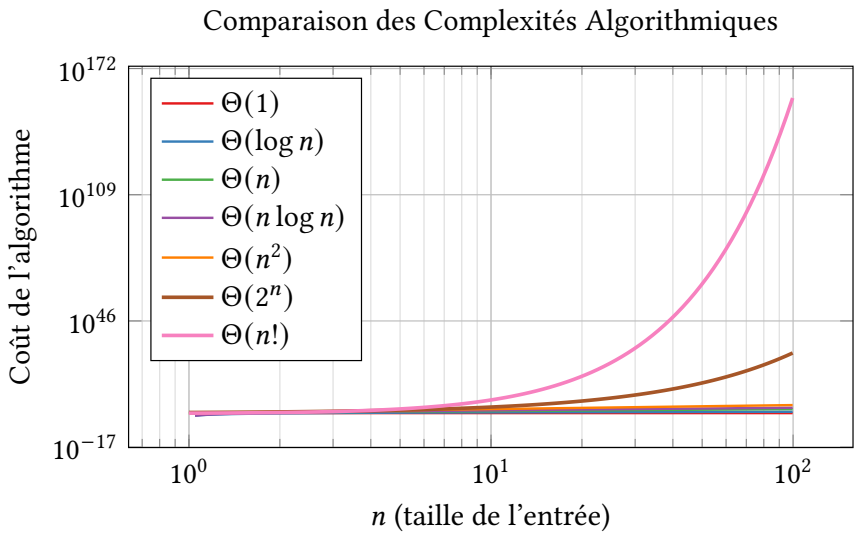
Notation $\mathbf{O}(\cdot)$: définition

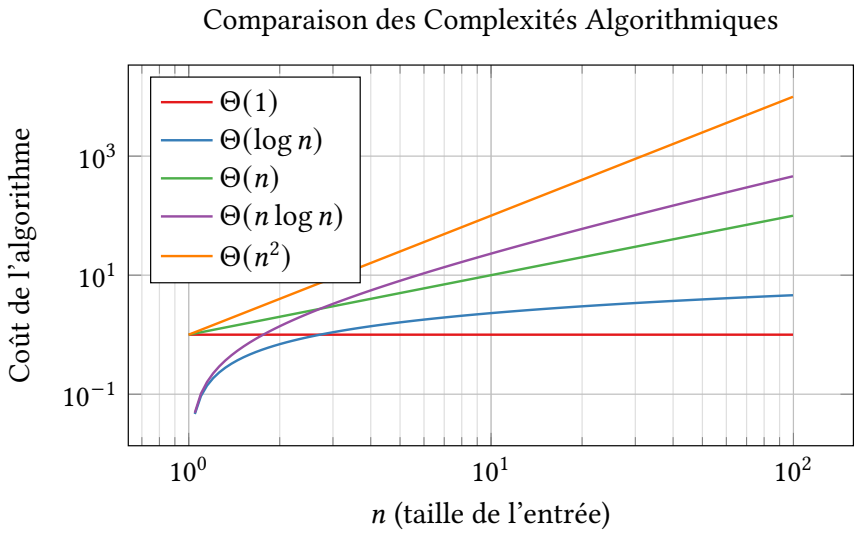
- **Définition**

> **Soient $f, g : \mathbb{N} \to \mathbb{R}_+$ deux fonctions non-négatives**
> **On dit que "$f(n)$ est un grand $\mathbf{O}$ de $g(n)$" et on écrit "$f(n) = \mathbf{O}(g(n))$"**
> **s'il existe $C > 0$ et $N \geq 1$ tels que**
>
> $$f(n) \leq C\ g(n) \quad \textbf{pour tout } n \geq N$$

- **Exemples**
  - Si $f(n) \leq g(n)$ pour tout $n \geq 1$, alors $f(n) = O(g(n))$
  - Si $f(n) = \Theta(g(n))$, alors $f(n) = O(g(n))$
  - La fonction $f(n) = 3\,n + 1$ est un $O(n)$, est aussi un $O(n^2)$
  - La fonction $f(n) = n^2 + 2n + 1$ est un $O(n^2)$, mais n'est pas un $O(n)$
  - La fonction $\log_2(n)$ est un $O(n^p)$ pour tout $p > 0$
  - La fonction n^p est un $O(2^n)$ pour tout $p > 0$

ICC-T 02 : Notation $\Theta(\cdot)$: Ordres de grandeur





Impraticables : $\boldsymbol{\Theta(2^n)}$, $\boldsymbol{\Theta(n!)}$

Problèmes appartenant à la classe P

Plus lents, mais souvent acceptés : $\boldsymbol{\Theta(n^2)}$ ... $\boldsymbol{\Theta(n^k)}$, $\boldsymbol{\Theta(n.\log(n))}$

Rapides: $\boldsymbol{\Theta(1)}$, $\boldsymbol{\Theta(\log n)}$, $\boldsymbol{\Theta(n)}$

Classe P : Résoudre en temps polynomial

- La **classe P** est l'ensemble des problèmes qui peuvent être résolus en temps polynomial
  - il existe un algorithme de résolution dont la complexité temporelle est

$O(n^p)$ pour des données d'entrée de taille n
(avec $p \geq 1$ un nombre fixé)

Classe P : Résoudre en temps polynomial

- « Etant donnée une liste ordonnée L de n nombres entiers, existe-t-il deux indices différents $i, j \in \{1, \dots, n\}$ tels que $L(i) + L(j) = 0$? »
  - Comme nous l'avons déjà vu, un algorithme de complexité temporelle $\Theta(n^2)$, ou même $\Theta(n)$, permet de répondre à cette question : le problème ci-dessus fait donc partie de la **classe P**.
- « Etant donnée une liste ordonnée L de n nombres entiers, existe-t-il trois indices différents $i, j, k \in \{1, \dots, n\}$ tels que $L(i) + L(j) + L(k) = 0$? »
  - Un algorithme de complexité temporelle $\Theta(n^3)$ permet de répondre à cette question : le problème ci-dessus fait donc également partie de la classe P.

Exemples de problèmes appartenant à la classe P

- Le problème de la recherche d'un élément dans une liste de taille n.
  Non triée : $O(n)$, triée $O(log_2 n)$
- Le problème du tri d'une liste de taille n.
  $O(n \log_2 n)$
- Le calcul de la somme ou de moyenne de n nombres. $O(n)$
- Identifier si tous les éléments d'une liste de taille n sont différents. $O(n^2)$
- Calculer le nième terme de la suite de Fibonnaci.
  Exponentiation de matrices : $O(\log n)$
- Multiplication de deux matrices de taille $n \times n$. $O(n^{2.3728596})$

Tous $O(n^3)$

Classification des problèmes

❖ **Décidables**

- Recherche dans une liste

P

- Tri

- Le problème du voyageur du commerce

❖ **Indécidables**

- Problème de l'arrêt
- Problème de l'équivalence des programmes
- Problème du pavage
- Problème de la validité en arithmétique

Aujourd’hui

- Dénombrement
- P : Résoudre en temps polynomial
- NP : Vérifier en temps polynomial
- Problèmes d’optimisation discrète

Classe NP : Vérifier en temps polynomial

NP

- La **classe NP** est l'ensemble des problèmes pour lesquels, si on nous propose une solution du problème, il est alors possible de
  vérifier en temps polynomial si celle-ci en est une ou pas.

- **Remarques** :
  - NP ne veut pas dire « non-polynomial »
  - P ⊂ NP : Il est plus facile de vérifier une solution que d'en proposer une !

  - Tous les problèmes appartenant à la classe P appartiennent également à la classe NP !

Classe NP : Factorisation

- « Soient P, Q deux nombres premiers à n chiffres, et soit $N = P \cdot Q$. Etant donné N, on aimerait retrouver P et Q. »

- **Exemple** : si $N = 98'201$, que valent P et Q ?

  - Ce problème n'est a priori pas facile à résoudre…
    Par contre, si on nous propose une solution (ici, $P = 347$ et $Q = 283$), il est alors facile de vérifier que $N = P \cdot Q$ en effectuant la multiplication (en $\Theta(n^2)$ opérations) :
    le problème appartient donc à la classe NP.

Classe NP : Somme de sous-ensembles

- « Etant donnée une liste ordonnée L de n nombres entiers, **existe-t-il** un sous-ensemble $S \subset \{1, \dots, n\}$ tel que $\sum_{(i \in S)} L(i) = 0$? »

- **Exemple :** $L = (-15, -12, -3, -1, +5, +17, +23)$
- **Réponse :** oui ! En effet : $-15 \ -12 - 1 + 5 + 23 = 0$

  - Mais pour obtenir cette réponse, on ne connaît pas d'autre algorithme que de tester tous les sous-ensembles S possibles, qui sont au nombre de 2^n.

❖ **Conclusion 1** : On ne sait pas si ce problème fait partie de la classe P.
❖ **Conclusion 2** : Si on nous propose une solution du problème, il est alors facile de vérifier en temps polynomial $\Theta(n)$ que c'en est bien une !

Le problème ci-dessus fait partie de la classe NP.

P vs. NP

P vs. NP : La question à un million de dollars !

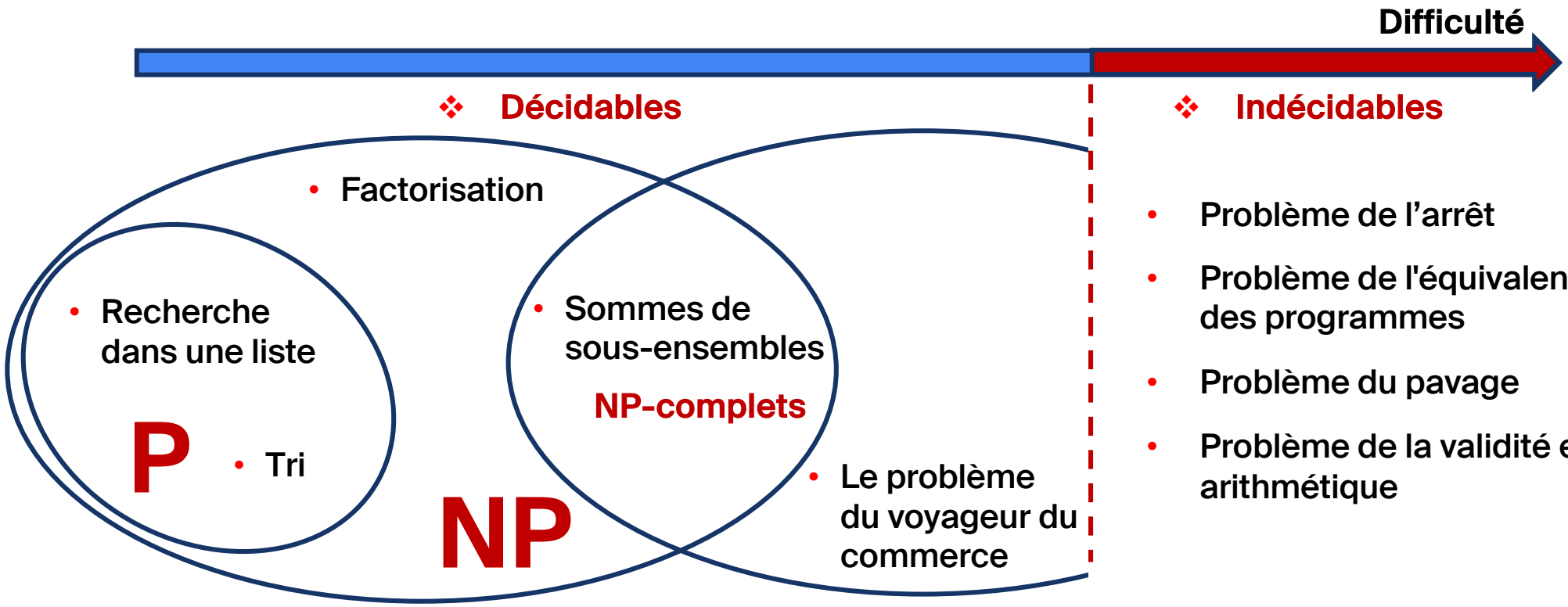
- Le problème :

« Etant donnée une liste ordonnée L de n nombres entiers,
existe-t-il un sous-ensemble $S \subset \{1, ..., n\}$
tel que $\sum_{i \in S} L(i) = 0$? »

s'appelle le problème des sommes des sous-ensembles.
On peut montrer qu'il fait partie des problèmes les plus difficiles à résoudre parmi ceux de la classe NP.

- Si vous démontrez que ce problème est dans la classe P, ou qu'il n'en fait pas partie,
  vous aurez prouvé si $P = NP$ ou non.
  - Allez chercher votre récompense au Clay Mathematics Institute !

Classification des problèmes



Aujourd’hui

- Dénombrement
- P : Résoudre en temps polynomial
- NP : Vérifier en temps polynomial
- Problèmes d’optimisation discrète

Problèmes d’optimisation discrète

- Un problème d'optimisation discrète est un problème où l'on cherche à maximiser ou minimiser une fonction objective, sous certaines contraintes et où les choix possibles sont pris dans un ensemble discret (fini ou dénombrable).

- Pour de nombreux problèmes d’optimisation discrète, on ne sait pas s’ils appartiennent à la classe P, ni s’ils appartiennent à la classe NP.

- Tous ces problèmes sont caractérisés par l’existence d’un nombre fini, mais grand, de solutions possibles.

Le problème du sac à dos

- n objets, chacun pesant un certain poids
- un sac à dos de capacité maximale C
- Comment remplir au mieux le sac à dos ?

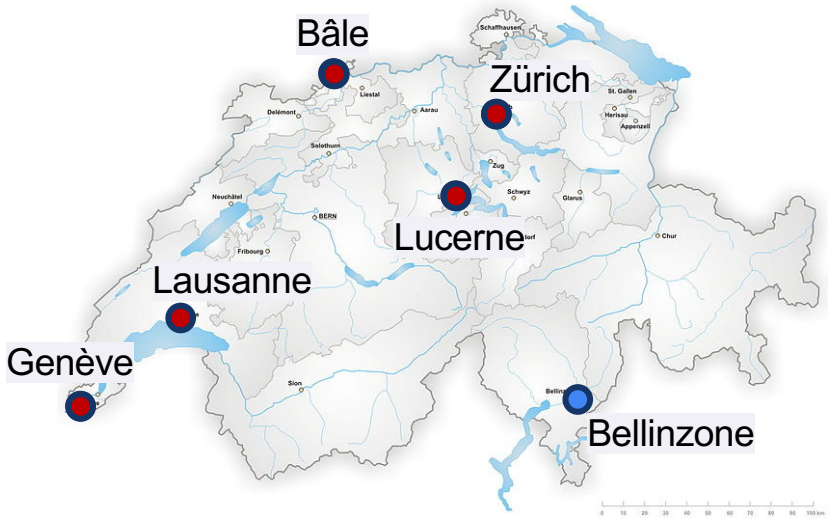
Le problème du sac à dos

- « Etant donnés une liste L de n nombres entiers positifs et $C > 0$, trouver un sous-ensemble $S \subset \{1, \dots, n\}$ tel que $\sum_{i \in S} L(i) \leq C$ et $\sum_{i \in S} L(i)$ soit maximale. »

- On ne sait pas s'il fait partie de la classe P, ni s'il fait partie de la classe NP. Voici toutefois un algorithme polynomial en n qui remplit au moins la moitié du sac à dos :

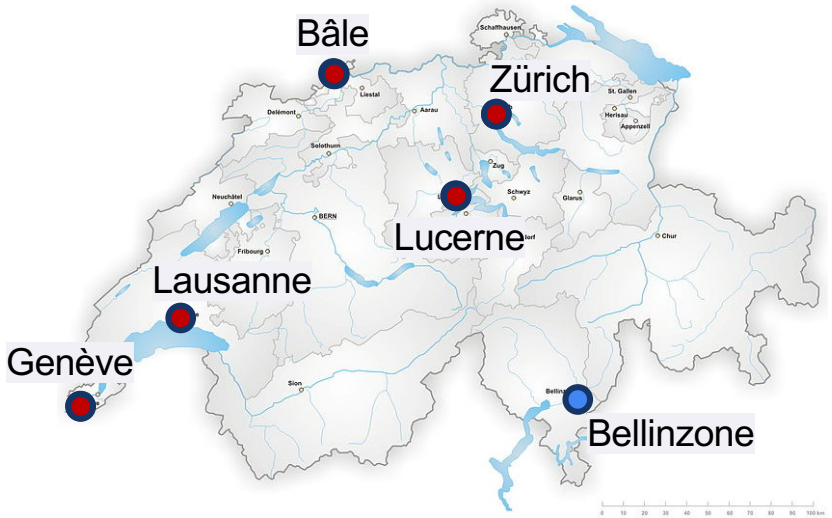
1. Ordonner la liste L dans l'ordre décroissant
2. Chercher le nombre $k \in \{1, \dots, n-1\}$ tel que
$$\sum_{i=1}^{k} L(i) \leq C \text{ et } \sum_{i=1}^{k+1} L(i) > C$$
3. Alors $\sum_{i=1}^{k} L(i) > \frac{C}{2}$

Le problème du voyageur de commerce



- Version « Euclidienne » : voyages à vol d'oiseau entre chaque ville.
  - Étant donné un ensemble de n villes sur une carte, trouver le chemin fermé le plus court passant une et une seule fois par chacune de ces villes.

Le problème du voyageur de commerce



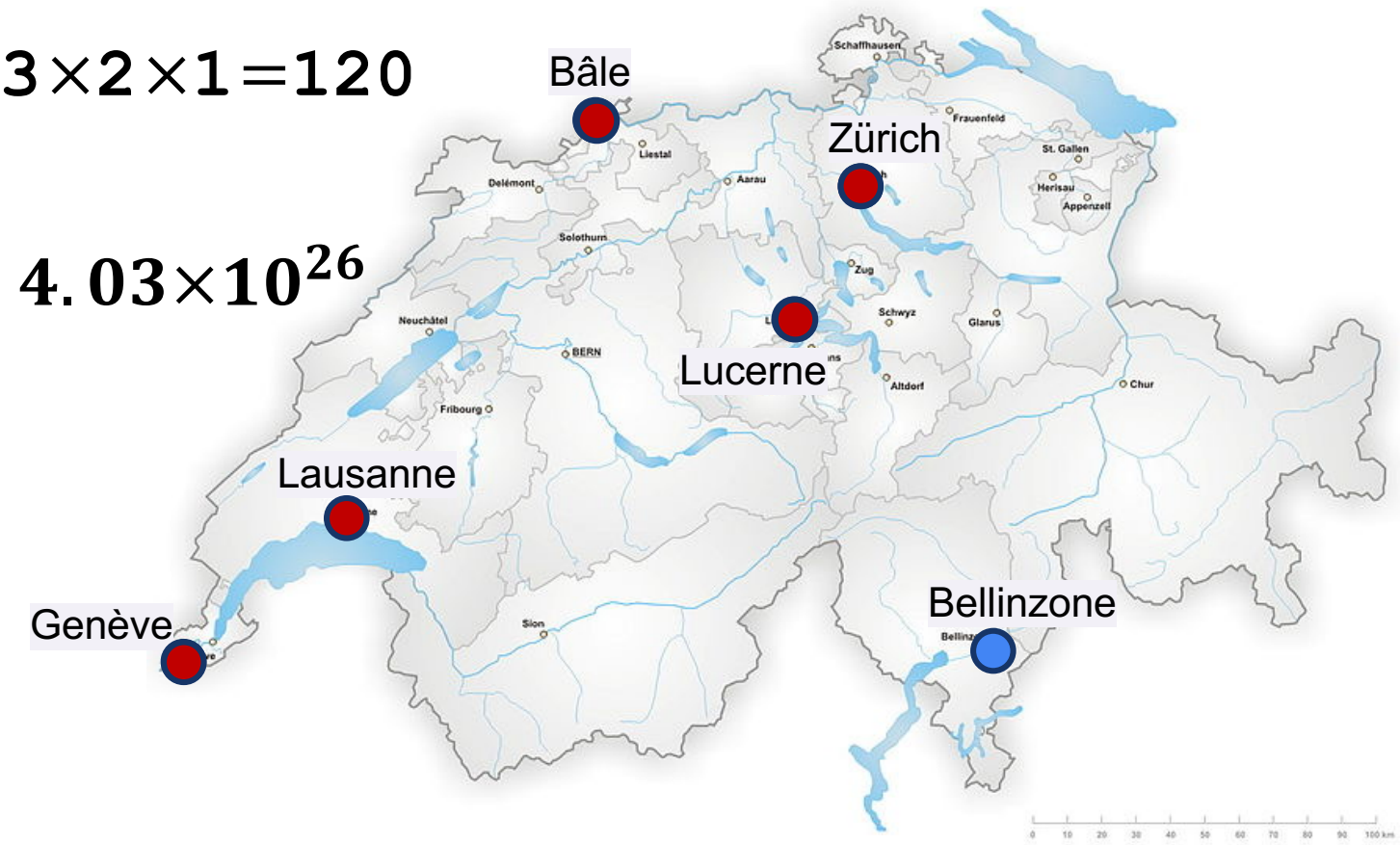
- **Premier essai** :
  - Tester tous les chemins fermés possibles.
  - Cet algorithme trouve le chemin fermé optimal. ✅
  - Impossible à mettre en pratique !

ICC-T 01 : problème du voyageur de commerce

$$5 \times 4 \times 3 \times 2 \times 1 = 120$$

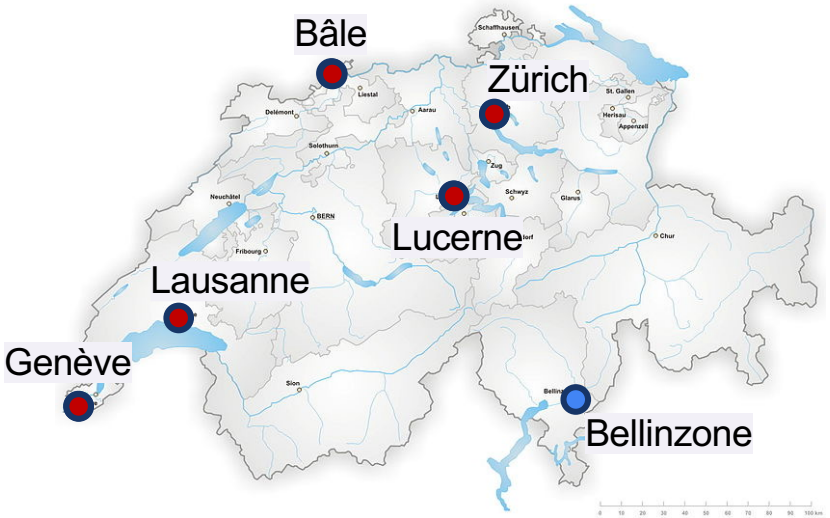
$$n!$$

$$26! \approx 4.03 \times 10^{26}$$



Factoriale	Résultat
1!	1
2!	2
3!	6
4!	24
5!	120
6!	720
7!	5040
8!	40320
9!	362880
10!	3628800
11!	39916800
12!	479001600
13!	6227020800
14!	87178297200
15!	1307674368000
16!	20922789888000
17!	355687428096000
18!	6402373705728000
19!	121645100408832000
20!	2432902008176640000
21!	51090942171709440000
22!	1124000727777607680000
23!	25852016738884976640000
24!	620448401733239439360000
25!	15511210043330985984000000

Le problème du voyageur de commerce

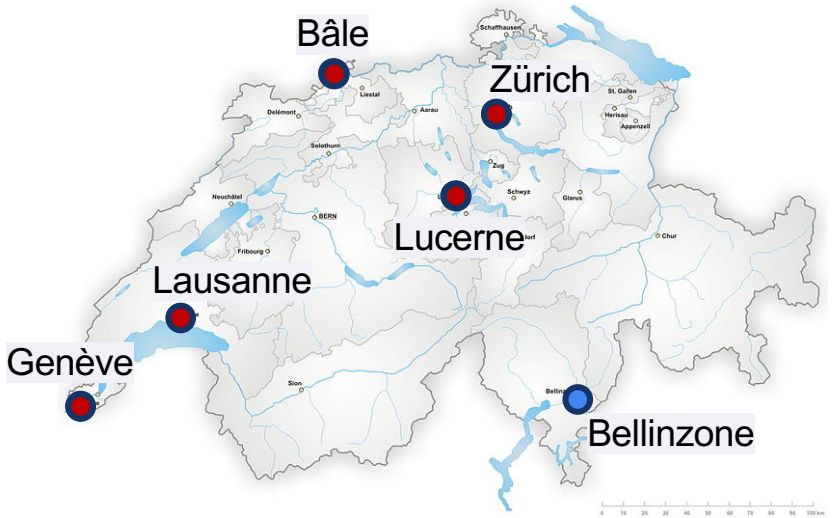


- **Deuxième essai** :

1. Partir d'une vile choisie au hasard.
2. Tant qu'il reste une ville non-explorée, rejoindre la ville non-explorée la plus proche.
3. Quand toutes les villes ont été explorées, revenir à la ville de départ.

- Complexité temporelle polynomiale en n. ✅
- En général, le chemin fermé résultant est loin d'être optimal !

Le problème du voyageur de commerce



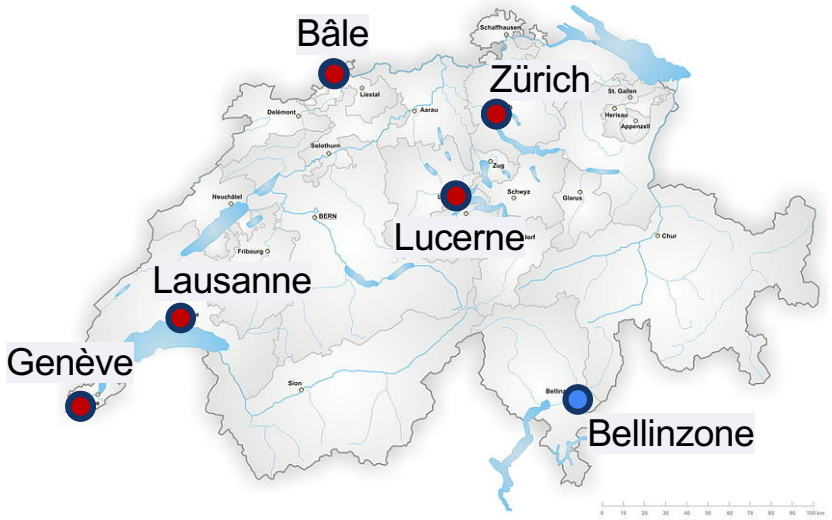
- **Troisième essai** :

1. Choisir un chemin fermé au hasard.
   $(v_1, v_2, \cdots, v_n, v_{n+1} = v_1)$
2. Effectuer une boucle pour i allant de 1 à n :
   - Permuter les villes v_i et v_{i+1} dans le chemin si cette permutation raccourcit la longueur totale du chemin.

- Complexité temporelle polynomiale en n. ✅

- Meilleure performance moyenne que l'algorithme précédant,
  mais résultat toujours aléatoire.

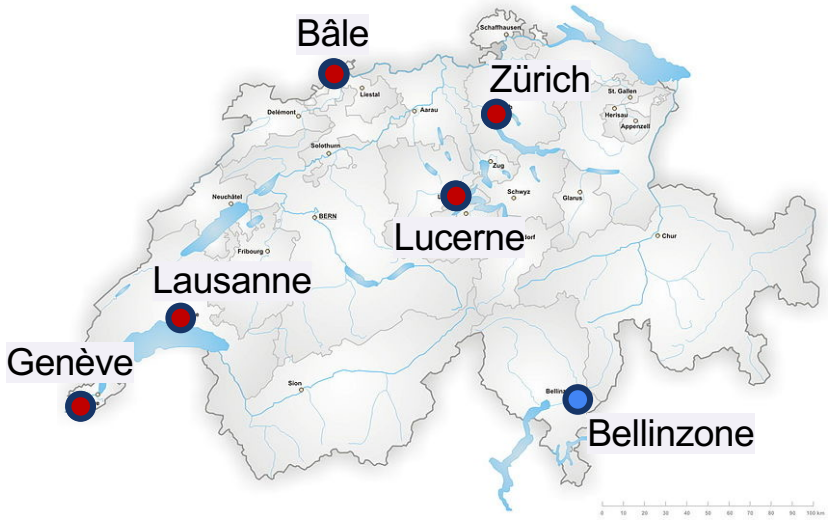
Le problème du voyageur de commerce : Algorithme avec garantie d'approximation et $O(n^k)$



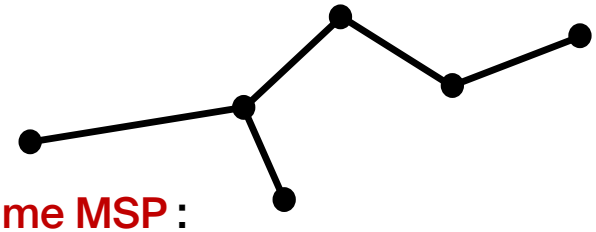
- **Théorème** :
  - Soit L_{min} la longueur du chemin fermé optimal (*i.e.*, le plus court), il existe un algorithme de complexité temporelle polynomiale en n permettant de trouver un chemin fermé de longueur $L \leq 2L_{min}$.

1. Trouver une arbre couvrant minimal
2. Parcourir le long de l'arbre en prenant des raccourcis.

Le problème du voyageur de commerce : Algorithme avec garantie d'approximation et $O(n^k)$



- Étant données des villes sur une carte, on cherche à les relier par un arbre dont la somme des longueurs des branches soit minimale.

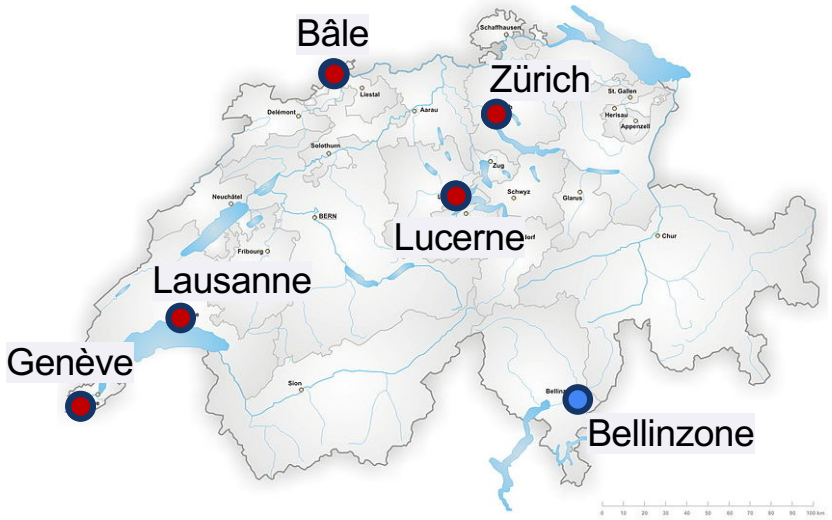


- Algorithme MSP :

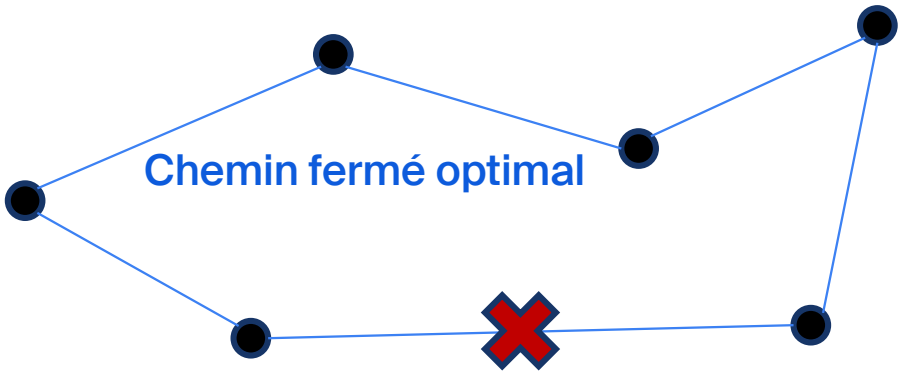
1. Commencer avec T={une ville au hasard} (= racine)
2. Chercher la ville v la plus proche d'une des villes $w \in T$ parmi les restants. Rajouter v et la branche $v - w$ à T.
3. Recommencer en 2. jusqu'à ce que T contienne toutes les villes.

❖ Il existe des algorithmes MSP (Kruskal, Prim) en $O(m \log n)$, pour m arrêtes et n sommets.

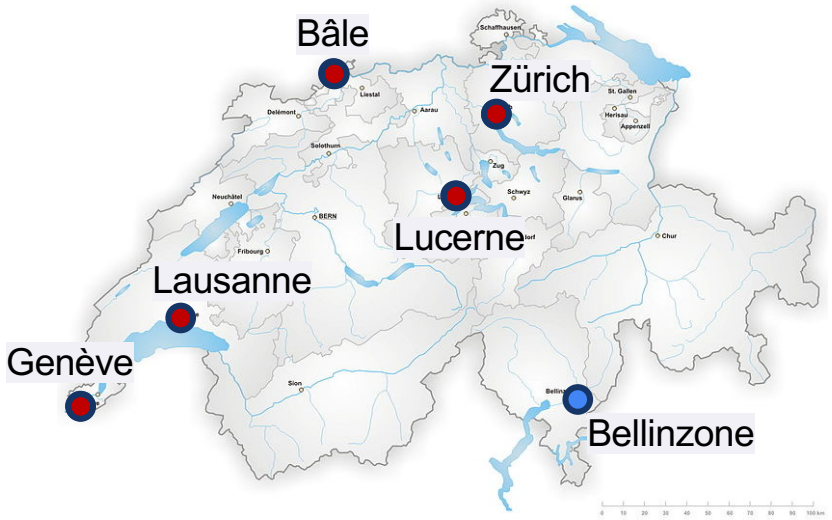
Le problème du voyageur de commerce : Algorithme avec garantie d'approximation et $O(n^k)$



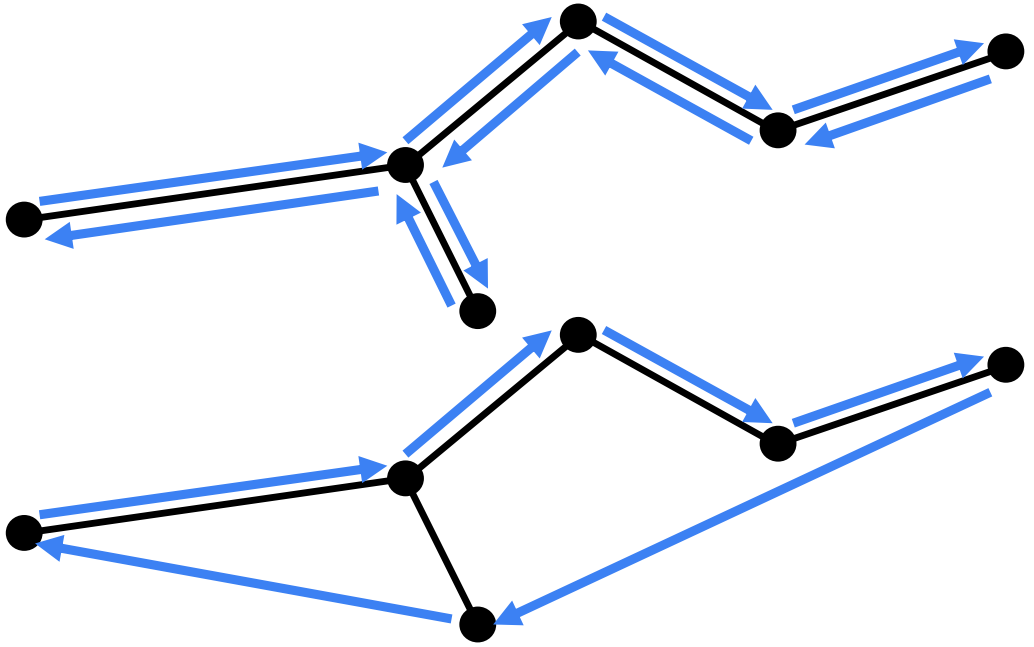
- **Remarque** : Si L_T désigne la somme des longueurs des branches de l'arbre couvrant minimal, et L_{min} désigne la longueur du chemin fermé optimal qui passe une fois par chaque ville, alors $L_{min} \geq L_T$.



Le problème du voyageur de commerce : Algorithme avec garantie d'approximation et $O(n^k)$

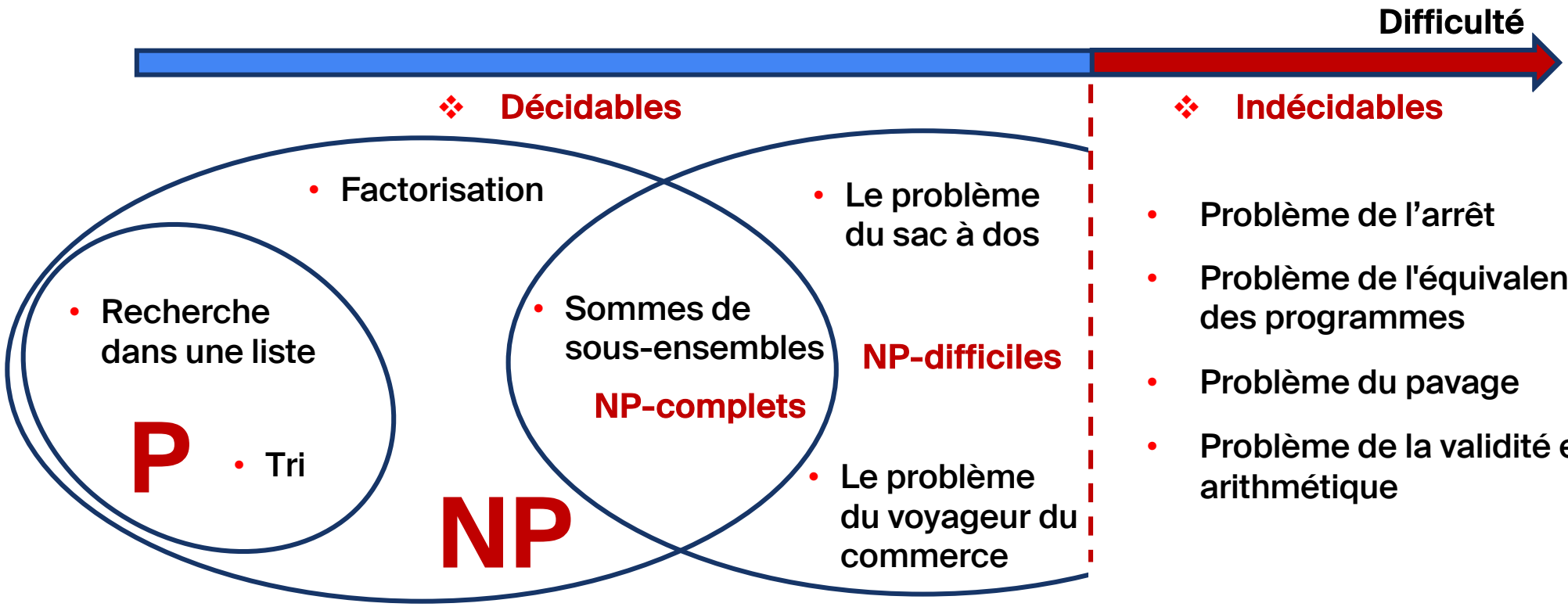


- La longueur du chemin bleu vaut $2L_T \leq 2L_{min}$.



- Ce nouveau chemin n'est pas plus long que le précédent (donc $\leq 2L_{min}$).

Classification des problèmes



Aujourd’hui

- Dénombrement
- P : Résoudre en temps polynomial
- NP : Vérifier en temps polynomial
- Problèmes d’optimisation discrète

Résumé Cours 5 – ICC-T

- **Dénombrement** : Il n'y a **pas assez d'algorithmes** pour calculer toutes les **fonctions booléennes**, car certains ensembles infinis sont plus vastes que d'autres.
- **Classe P** : Problèmes **résolus** efficacement en temps polynomial.
- **Classe NP** : Problèmes où une solution peut être **vérifiée** efficacement en temps polynomial.
- **P vs NP** : Si $P = NP$, tous les problèmes dont la solution se **vérifie** facilement pourraient aussi être **résolus** efficacement.
  - La question reste ouverte, avec un prix de 1 million de dollars à la clé.
- Même pour des **problèmes d'optimisation** très difficiles, il existe des méthodes efficaces qui garantissent une performance satisfaisante, même si elles ne fournissent qu'une **approximation** de la solution optimale.

rafael.pires@epfl.ch

Merci