La complexité du problème TETRIS

Brussels Summer School of Mathematics

Keno Merckx

ULB

Août 2015

Plan

Approche

Voici une liste de tâches:

Approche

Voici une liste de tâches:

- Corriger des copies d'examens
- Trier des dossiers par ordre alphabétique
- Préparer un sac-à-dos avant un départ en vacance
- Factoriser des nombres
- Jouer à Tetris
- Gagner une partie d'échec contre une intelligence artificielle
- Réaliser une thèse de doctorat

Approche

Voici une liste de tâches:

- Corriger des copies d'examens
- Trier des dossiers par ordre alphabétique
- Préparer un sac-à-dos avant un départ en vacance
- Factoriser des nombres
- Jouer à Tetris
- Gagner une partie d'échec contre une intelligence artificielle
- Réaliser une thèse de doctorat

Pour affirmer qu'une de ces tâches est "plus dure" qu'une autre il nous faut formaliser un certain nombre de notions.

Qu’est-ce qu’un problème difficile ?

Un exposé facile à propos de choses difficiles

Nous allons tenter d’approcher une définition de "difficulté". L’accent sera mis sur l’aspect algorithmique de la résolution d’un problème donné.

$\Rightarrow$ Théorie de la complexité : entre les mathématiques et l’informatique

Qu’est-ce qu’un problème difficile ?

Un exposé facile à propos de choses difficiles

Nous allons tenter d’approcher une définition de "difficulté". L’accent sera mis sur l’aspect algorithmique de la résolution d’un problème donné.

$\Rightarrow$ Théorie de la complexité : entre les mathématiques et l’informatique

Pourquoi faire cela?

Qu’est-ce qu’un problème difficile ?

Un exposé facile à propos de choses difficiles

Nous allons tenter d’approcher une définition de "difficulté". L’accent sera mis sur l’aspect algorithmique de la résolution d’un problème donné.

$\Rightarrow$ Théorie de la complexité : entre les mathématiques et l’informatique

Pourquoi faire cela?

- Nouvelle manière de penser aux problèmes et à leurs solutions

Qu’est-ce qu’un problème difficile ?

Un exposé facile à propos de choses difficiles

Nous allons tenter d’approcher une définition de "difficulté". L’accent sera mis sur l’aspect algorithmique de la résolution d’un problème donné.

$\Rightarrow$ Théorie de la complexité : entre les mathématiques et l’informatique

Pourquoi faire cela?

- Nouvelle manière de penser aux problèmes et à leurs solutions
- La théorie est souvent en avance sur la pratique

Qu’est-ce qu’un problème difficile ?

Un exposé facile à propos de choses difficiles

Nous allons tenter d’approcher une définition de "difficulté". L’accent sera mis sur l’aspect algorithmique de la résolution d’un problème donné.

$\Rightarrow$ Théorie de la complexité : entre les mathématiques et l’informatique

Pourquoi faire cela?

- Nouvelle manière de penser aux problèmes et à leurs solutions
- La théorie est souvent en avance sur la pratique
- L’honneur de l’esprit humain

Qu’est-ce qu’un problème difficile ?

Un exposé facile à propos de choses difficiles

Nous allons tenter d’approcher une définition de "difficulté". L’accent sera mis sur l’aspect algorithmique de la résolution d’un problème donné.

$\Rightarrow$ Théorie de la complexité : entre les mathématiques et l’informatique

Pourquoi faire cela?

- Nouvelle manière de penser aux problèmes et à leurs solutions
- La théorie est souvent en avance sur la pratique
- L’honneur de l’esprit humain
- De nombreuses applications pratiques

Les problèmes dont il est question ici

Quelques définitions et notations standards:

Les problèmes dont il est question ici

Quelques définitions et notations standards:

- soit Σ^* un alphabet, c'est-à-dire un un ensemble d'éléments appelés symboles,

Les problèmes dont il est question ici

Quelques définitions et notations standards:

- soit Σ^* un alphabet, c'est-à-dire un un ensemble d'éléments appelés symboles,
- appelons un mot, la concaténation d'un nombre fini de symboles,

Les problèmes dont il est question ici

Quelques définitions et notations standards:

- soit Σ^* un alphabet, c'est-à-dire un un ensemble d'éléments appelés symboles,
- appelons un mot, la concaténation d'un nombre fini de symboles,
- un langage Σ sera un ensemble de mots.

Les problèmes dont il est question ici

Quelques définitions et notations standards:

- soit Σ^* un alphabet, c'est-à-dire un un ensemble d'éléments appelés symboles,
- appelons un mot, la concaténation d'un nombre fini de symboles,
- un langage Σ sera un ensemble de mots.

Exemple

- $\Sigma^* = \{0, 1, 2, 3, 4, 5, 6, 7, 9\}$,
- $\Sigma = \mathit{PREMIER} = \{x : x \text{ est un nombre premier}\}$.

Les problèmes dont il est question ici

Quelques définitions et notations standards:

- soit Σ^* un alphabet, c'est-à-dire un un ensemble d'éléments appelés symboles,
- appelons un mot, la concaténation d'un nombre fini de symboles,
- un langage Σ sera un ensemble de mots.

Exemple

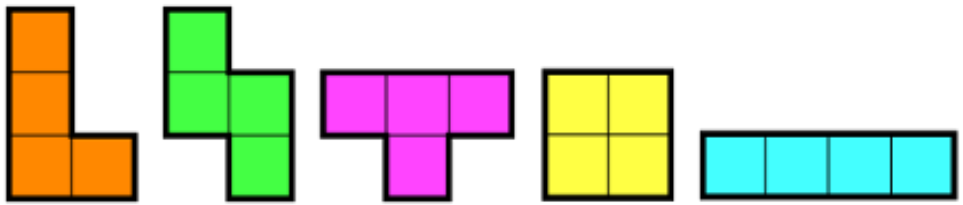
- $\Sigma^* = \{0, 1, 2, 3, 4, 5, 6, 7, 9\}$,
- $\Sigma = \mathit{PREMIER} = \{x : x \text{ est un nombre premier}\}$.

Nous allons observer des problèmes de décision, du type

$$\text{Soit } x, \text{ est-ce que } x \in \Sigma?$$

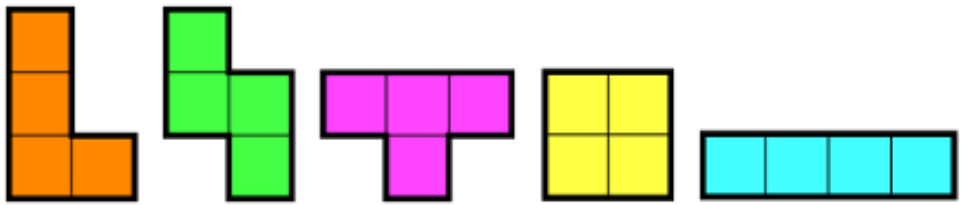
Aperçu rapide des règles du jeux Tetris

Le jeux Tetris est un puzzle inventé par Alexey Pajitnov en 1984 composé de formes appelées tétrominos:



Aperçu rapide des règles du jeux Tetris

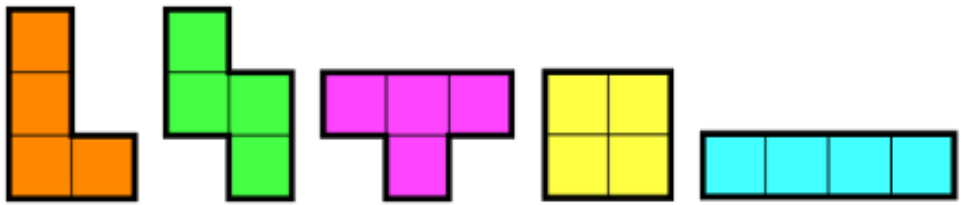
Le jeux Tetris est un puzzle inventé par Alexey Pajitnov en 1984 composé de formes appelées tétrominos:



Des tétrominos tombent un par un de haut en bas sur le plateau de jeu: un rectangle vertical de 10 blocs sur 18 blocs.

Aperçu rapide des règles du jeux Tetris

Le jeux Tetris est un puzzle inventé par Alexey Pajitnov en 1984 composé de formes appelées tétrominos:

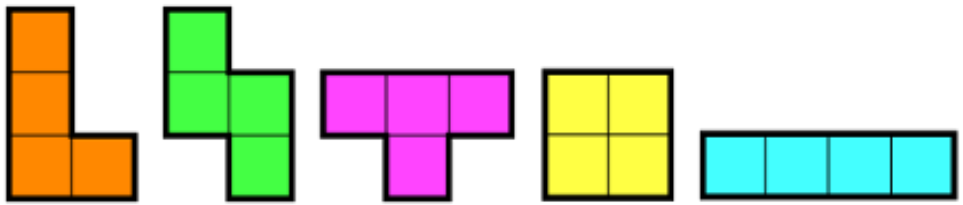


Des tétrominos tombent un par un de haut en bas sur le plateau de jeu: un rectangle vertical de 10 blocs sur 18 blocs.

Objectif: manipuler les tétrominos (latéralement ou par rotation de $\frac{\pi}{2}$) pendant leurs chutes pour créer en bas du plateau le plus de lignes horizontales de 10 blocs sans trou.

Aperçu rapide des règles du jeux Tetris

Le jeux Tetris est un puzzle inventé par Alexey Pajitnov en 1984 composé de formes appelées tétrominos:



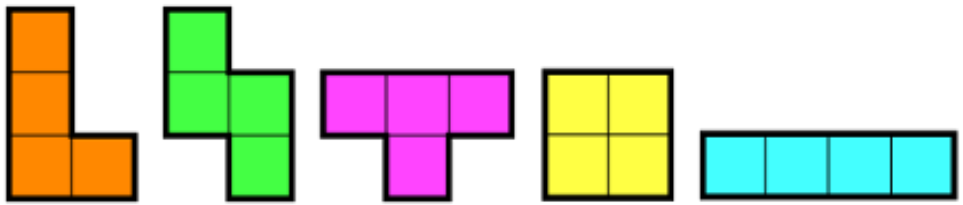
Des tétrominos tombent un par un de haut en bas sur le plateau de jeu: un rectangle vertical de 10 blocs sur 18 blocs.

Objectif: manipuler les tétrominos (latéralement ou par rotation de $\frac{\pi}{2}$) pendant leurs chutes pour créer en bas du plateau le plus de lignes horizontales de 10 blocs sans trou.

Quand une ligne de 10 est crée, elle disparaît du plateau et les morceaux de tétrominos au dessus d'elle tombent par un effet de gravité.

Aperçu rapide des règles du jeux Tetris

Le jeux Tetris est un puzzle inventé par Alexey Pajitnov en 1984 composé de formes appelées tétrominos:



Des tétrominos tombent un par un de haut en bas sur le plateau de jeu: un rectangle vertical de 10 blocs sur 18 blocs.

Objectif: manipuler les tétrominos (latéralement ou par rotation de $\frac{\pi}{2}$) pendant leurs chutes pour créer en bas du plateau le plus de lignes horizontales de 10 blocs sans trou.

Quand une ligne de 10 est crée, elle disparaît du plateau et les morceaux de tétrominos au dessus d'elle tombent par un effet de gravité.

Fin de jeu si une colonne de tétrominos atteint le haut du plateau de jeu, le joueur a perdu.

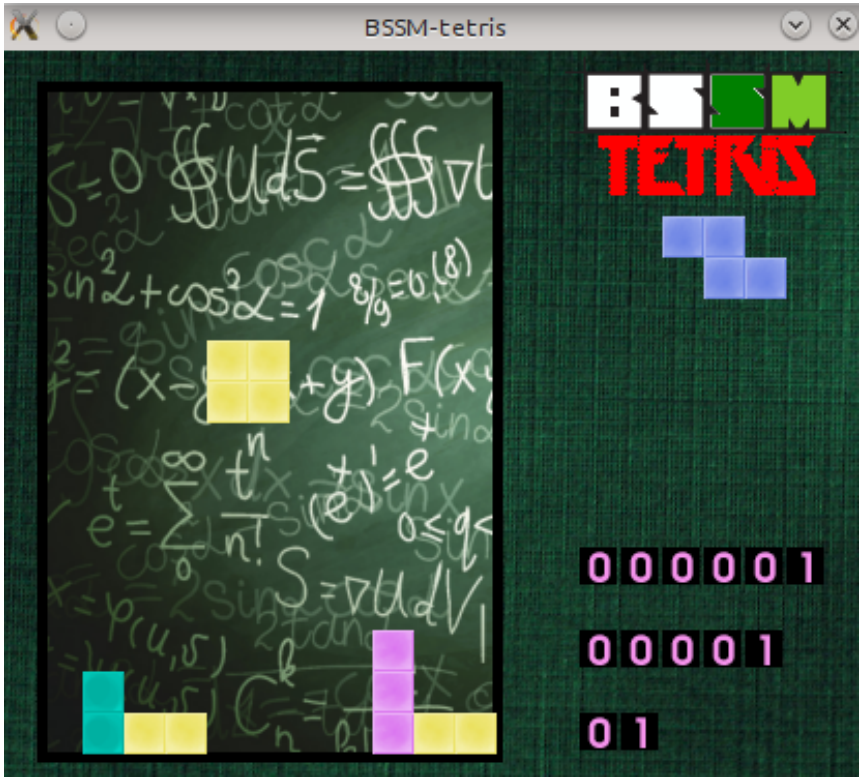
Pause programmation

Un petit programme vaut mieux qu'un long discours:

Pause programmation

Un petit programme vaut mieux qu'un long discours:

- cf. le Tetris officiel de la BSSM 2015.



Tetris est-il un problème?

Le problème TETRIS est la transformation du jeu en problème de décision.

Tetris est-il un problème?

Le problème TETRIS est la transformation du jeu en problème de décision.

Tetris offline Nous voyons la séquence finie de tétrominos qui se présentera à nous. Celle-ci sera notée s.

Tetris est-il un problème?

Le problème TETRIS est la transformation du jeu en problème de décision.

Tetris offline Nous voyons la séquence finie de tétrominos qui se présentera à nous. Celle-ci sera notée s.

Tetris avancé Le plateau est partiellement rempli. Celui-ci est donné sous forme d'une matrice binaire notée p.

Tetris est-il un problème?

Le problème TETRIS est la transformation du jeu en problème de décision.

Tetris offline Nous voyons la séquence finie de tétrominos qui se présentera à nous. Celle-ci sera notée s.

Tetris avancé Le plateau est partiellement rempli. Celui-ci est donné sous forme d'une matrice binaire notée p.

Le langage TETRIS peut donc être défini:

$$TETRIS = \{(p, s) : \text{on peux jouer } s \text{ pour vider entierement } p\}$$

Tetris est-il un problème?

Le problème TETRIS est la transformation du jeu en problème de décision.

Tetris offline Nous voyons la séquence finie de tétrominos qui se présentera à nous. Celle-ci sera notée s.

Tetris avancé Le plateau est partiellement rempli. Celui-ci est donné sous forme d'une matrice binaire notée p.

Le langage TETRIS peut donc être défini:

$$TETRIS = \{(p, s) : \text{on peux jouer } s \text{ pour vider entierement } p\}$$

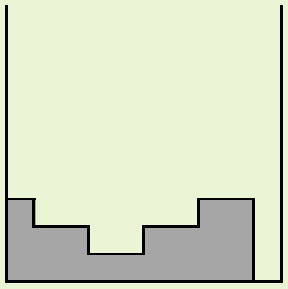
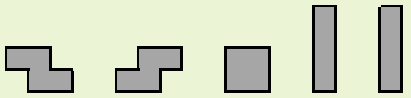
Le problème TETRIS

Etant donné un plateau p et une séquence de tétrominos s, est-il possible de vider de p à l'aide de s? Autrement dit:

$$\text{Soit } (p, s), \text{ est-ce que } (p, s) \in TETRIS\,?$$

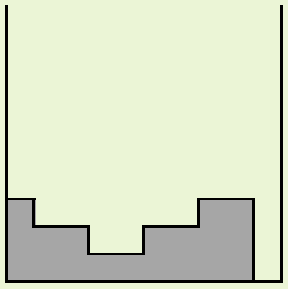
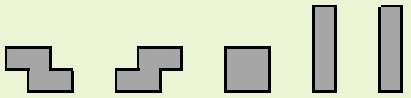
Instances de problèmes de décision

Exemples

- Est-ce que $2147483647 \in$ *PREMIER* ?
- Est-ce que $\Big($  ,  $\Big) \in$ *TETRIS* ?

Instances de problèmes de décision

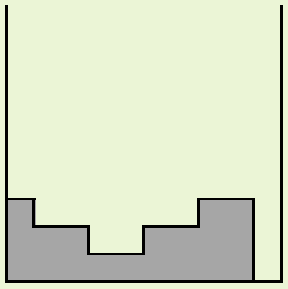
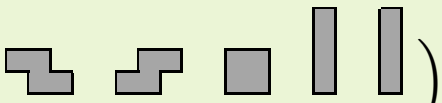
Exemples

- Est-ce que $2147483647 \in$ *PREMIER* ?
- Est-ce que $($  ,  $) \in$ *TETRIS* ?

$\Rightarrow$ A quel point ces questions deviennent "difficiles" si l'on agrandit le nombre ou le plateau donné ?

Instances de problèmes de décision

Exemples

- Est-ce que $2147483647 \in$ *PREMIER* ?
- Est-ce que $\Big($  ,  $\Big) \in$ *TETRIS* ?

$\Rightarrow$ A quel point ces questions deviennent "difficiles" si l'on agrandit le nombre ou le plateau donné ?

Ce qu'on peut envisager

- Trouver un algorithme $\mathcal{A}$ qui décide si $(p, s) \in$ *TETRIS*.
- Mesurer les ressources nécessaires à $\mathcal{A}$.

Mesure des ressources

Comment mesurer l'utilisation des ressources d'un algorithme $\mathcal{A}$?

Mesure des ressources

Comment mesurer l'utilisation des ressources d'un algorithme $\mathcal{A}$?

- Etre pessimiste:
  - Nous considérons la consommation dans le pire des cas en fonction de l'input.

Mesure des ressources

Comment mesurer l'utilisation des ressources d'un algorithme $\mathcal{A}$?

- Etre pessimiste:
  - Nous considérons la consommation dans le pire des cas en fonction de l'input.
- Ressources: le temps et l'espace.
  - Lancer $\mathcal{A}$ sur toutes les instances de taille n à l'aide d'une machine.
  - Etudier (asymptotiquement) la fonction de n par rapport à la ressources consommées par $\mathcal{A}$ sur "la pire" instance de taille n.

Mesure des ressources

Comment mesurer l'utilisation des ressources d'un algorithme $\mathcal{A}$?

- Etre pessimiste:
  - Nous considérons la consommation dans le pire des cas en fonction de l'input.
- Ressources: le temps et l'espace.
  - Lancer $\mathcal{A}$ sur toutes les instances de taille n à l'aide d'une machine.
  - Etudier (asymptotiquement) la fonction de n par rapport à la ressources consommées par $\mathcal{A}$ sur "la pire" instance de taille n.

Une première définition possible de la "difficulté"

Au vue de cette étude, on peut qualifier un problème de

- "difficile" si cette fonction pour l'algorithme de résolution croît de façon exponentielle,
- "facile" si cette fonction pour l'algorithme de résolution croît de façon polynomiale.

Quelques problèmes triés en fonction de leurs complexités

Facile

Difficile

Quelques problèmes triés en fonction de leurs complexités

Facile

Difficile

Est-ce que le premier nombre de ce vecteur est positif ?

Quelques problèmes triés en fonction de leurs complexités

Facile

Difficile

Est-ce que cette liste de nombre est triée ?

Est-ce que le premier nombre de ce vecteur est positif ?

Quelques problèmes triés en fonction de leurs complexités

Facile

Difficile

Est-ce que je peux colorier cette carte avec trois couleurs ?

Est-ce que cette liste de nombre est triée ?

Est-ce que le premier nombre de ce vecteur est positif ?

Quelques problèmes triés en fonction de leurs complexités

Facile

Difficile

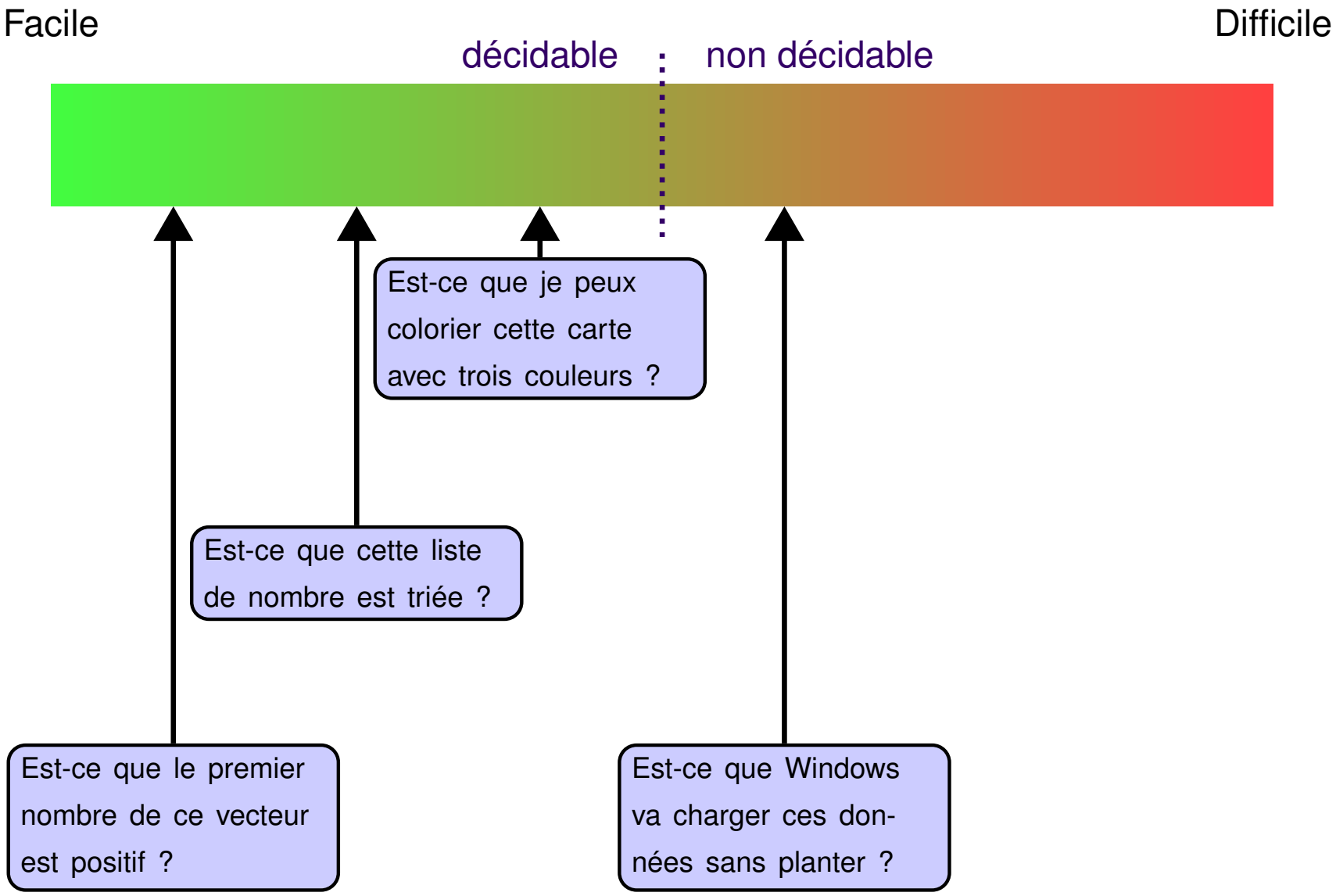
Est-ce que je peux colorier cette carte avec trois couleurs ?

Est-ce que cette liste de nombre est triée ?

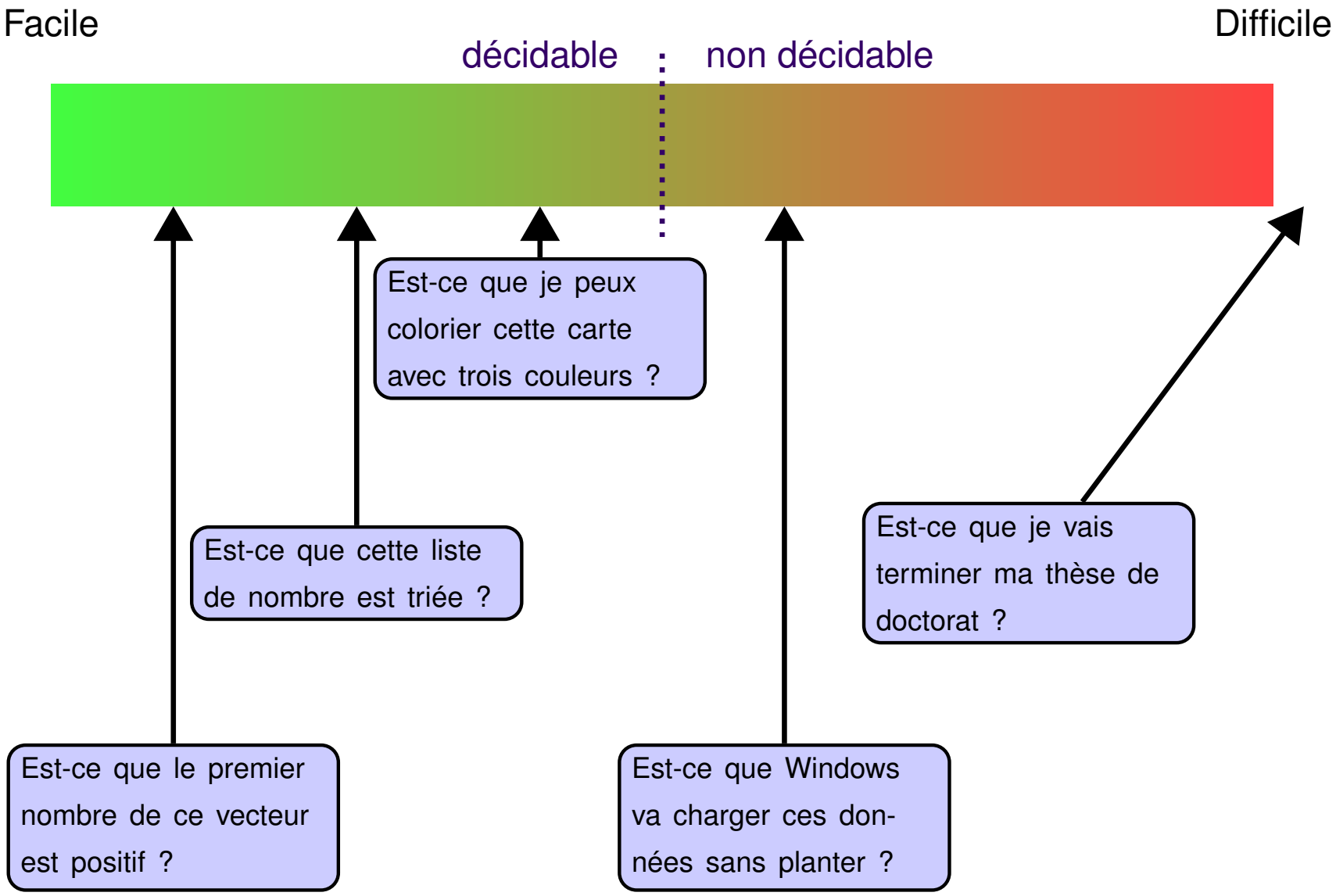
Est-ce que le premier nombre de ce vecteur est positif ?

Est-ce que Windows va charger ces données sans planter ?

Quelques problèmes triés en fonction de leurs complexités



Quelques problèmes triés en fonction de leurs complexités



Classification

Une classe de complexité est un ensemble de langages.

Classification

Une classe de complexité est un ensemble de langages.

- Un langage L est dans une des classes de complexité suivantes s'il existe un algorithme qui...

Classification

Une classe de complexité est un ensemble de langages.

- Un langage L est dans une des classes de complexité suivantes s'il existe un algorithme qui...

$\mathcal{P}$	...	décide toutes les questions $x \in L$ en temps polynomial.

Classification

Une classe de complexité est un ensemble de langages.

- Un langage L est dans une des classes de complexité suivantes s'il existe un algorithme qui...

$\mathcal{P}$	...	décide toutes les questions $x \in L$ en temps polynomial.
$\mathcal{P}$-SPACE	...	décide toutes les questions $x \in L$ à l'aide d'un espace polynomial.

Classification

Une classe de complexité est un ensemble de langages.

- Un langage L est dans une des classes de complexité suivantes s'il existe un algorithme qui...

$\mathcal{P}$	...	décide toutes les questions $x \in L$ en temps polynomial.
$\mathcal{P}$-SPACE	...	décide toutes les questions $x \in L$ à l'aide d'un espace polynomial.
$\mathcal{EXP}$-TIME	...	décide toutes les questions $x \in L$ en temps exponentiel.

Classification

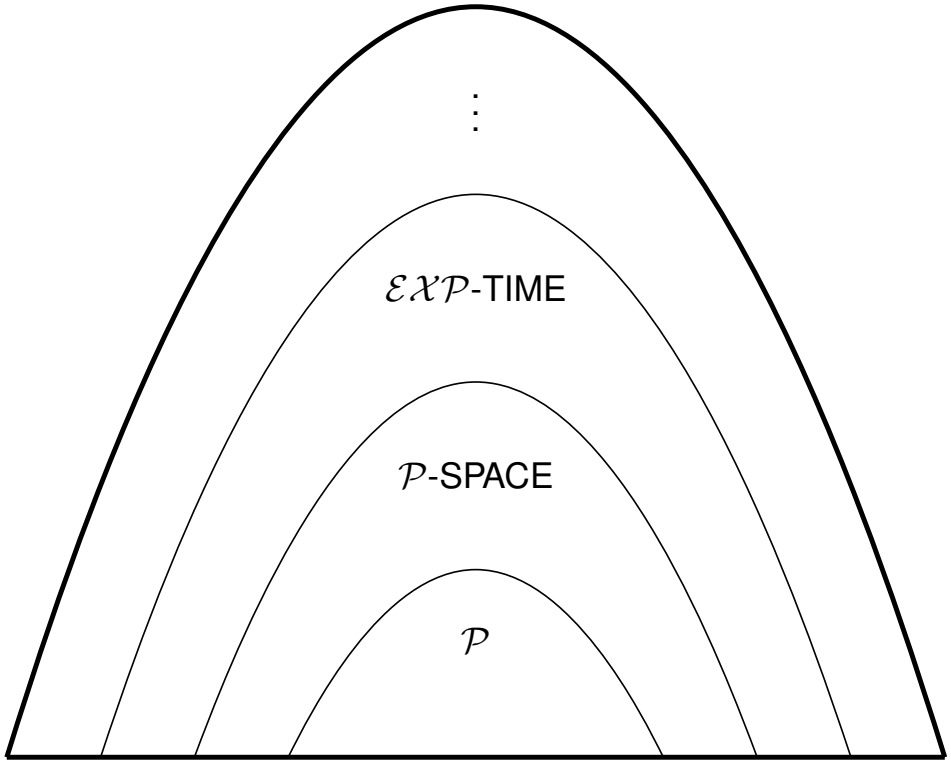
Une classe de complexité est un ensemble de langages.

- Un langage L est dans une des classes de complexité suivantes s'il existe un algorithme qui...

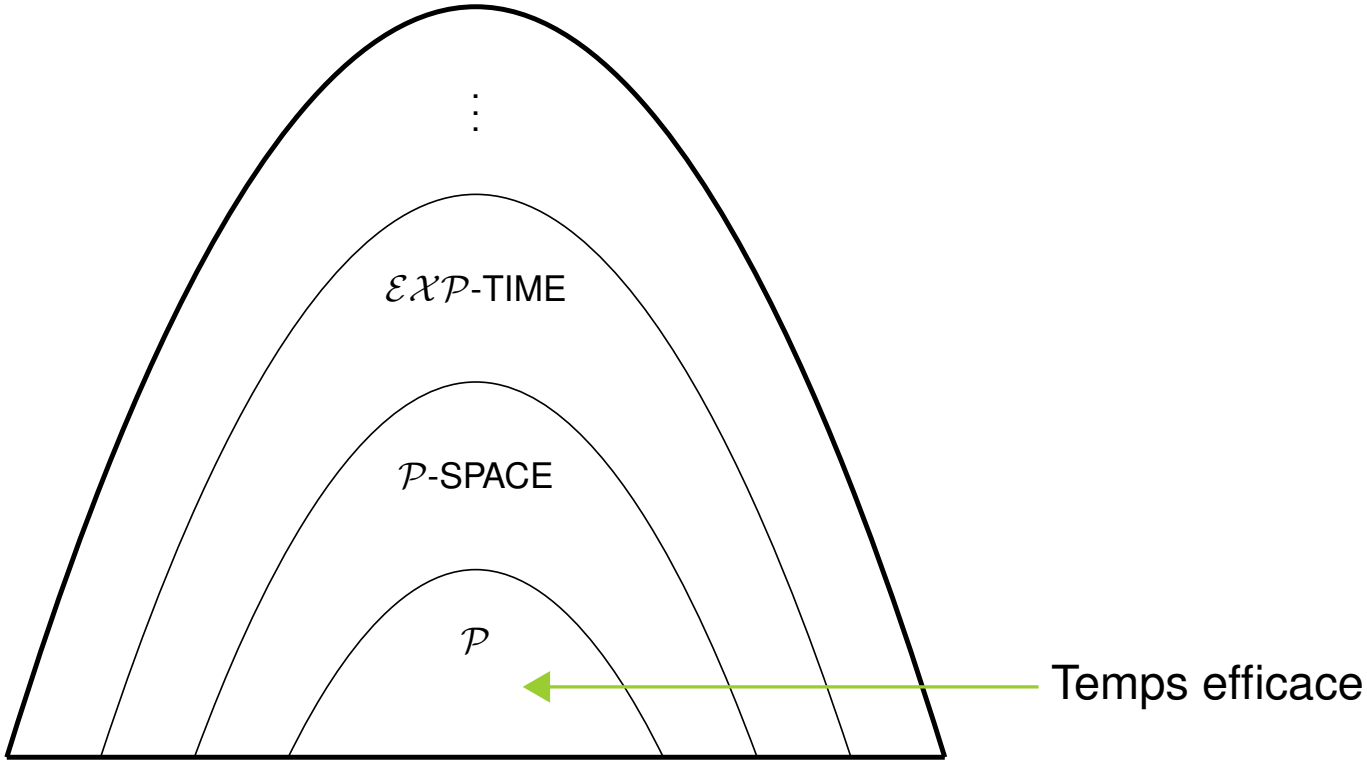
$\mathcal{P}$	...	décide toutes les questions $x \in L$ en temps polynomial.
$\mathcal{P}$-SPACE	...	décide toutes les questions $x \in L$ à l'aide d'un espace polynomial.
$\mathcal{EXP}$-TIME	...	décide toutes les questions $x \in L$ en temps exponentiel.

Il existe une multitude d'autres classes de complexité, beaucoup sont listées sur le site *Complexity Zoo*.

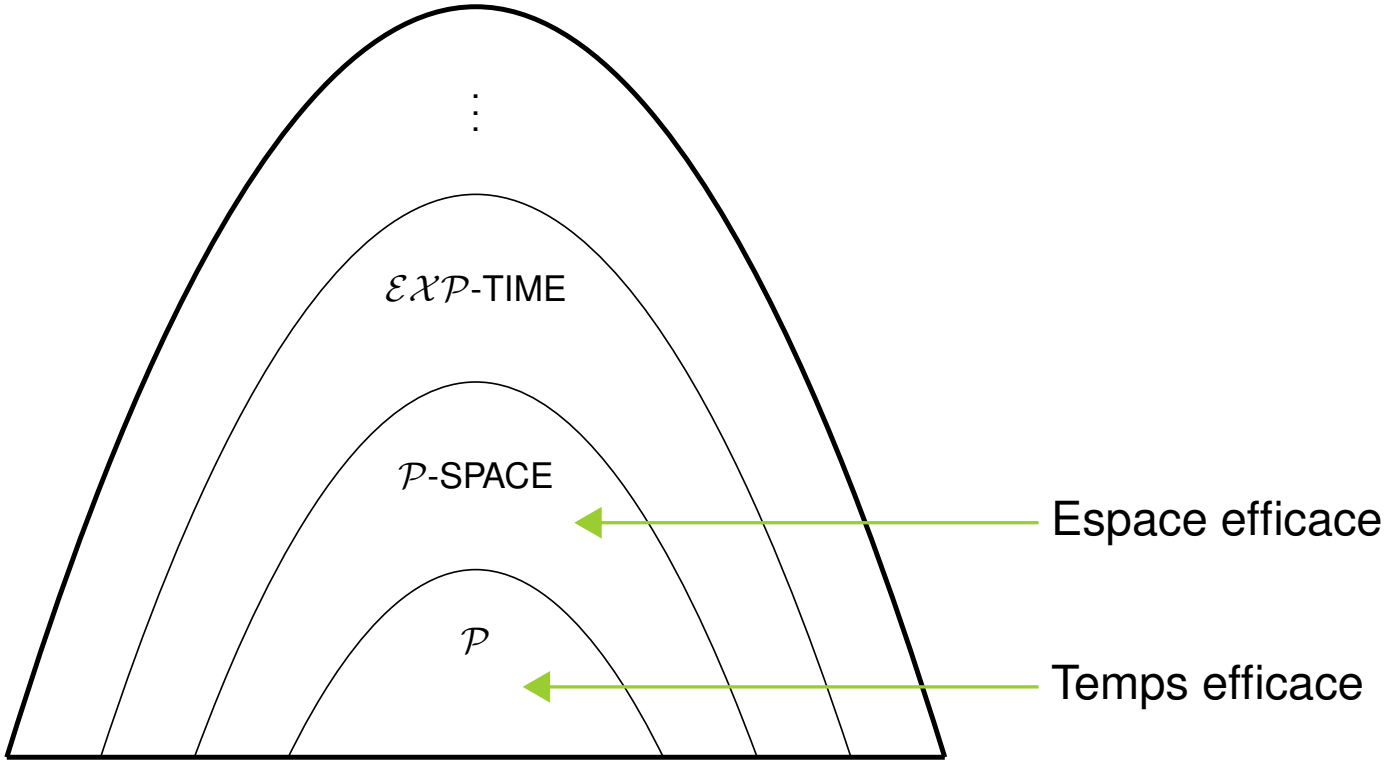
Hiérarchie



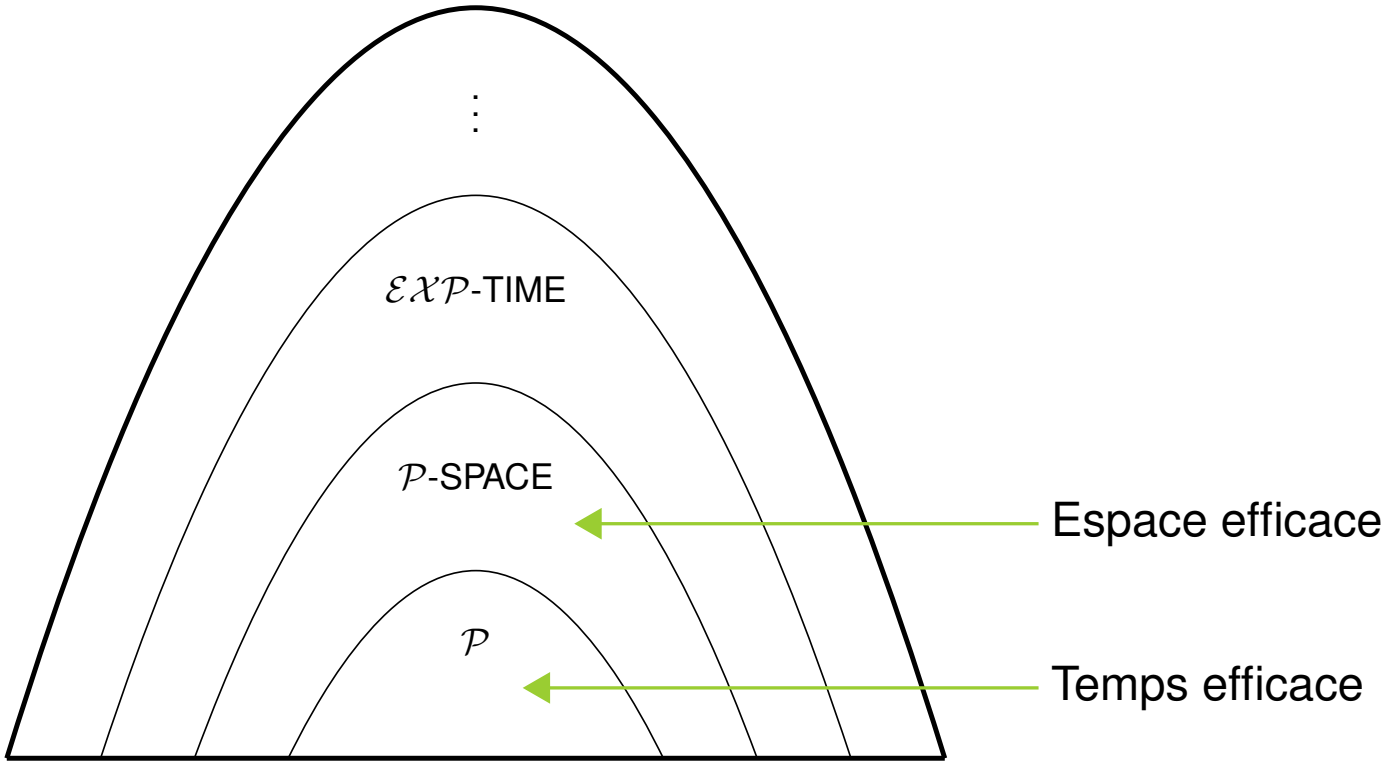
Hiérarchie



Hiérarchie


$\mathcal{EXP}$-TIME
$\mathcal{P}$-SPACE
$\mathcal{P}$
Espace efficace
Temps efficace

Hiérarchie



On sait que

- $\mathcal{P} \subseteq \mathcal{P}$-SPACE $\subseteq \mathcal{EXP}$-TIME
- $\mathcal{P} \neq \mathcal{EXP}$-TIME

Illustration: *TSP*

Le célèbre problème du voyageur de commerce (*TSP*).

Illustration: *TSP*

Le célèbre problème du voyageur de commerce (*TSP*).

Un exemple: *TSP*-Europe

Je veux me rendre dans les capitales des 28 états membres de l'UE. Chaque déplacement en avion entre deux capitales me coûte un prix donné. Je ne veux jamais passer deux fois par la même ville. Existe-t-il un ordre de visite dont le prix est inférieur à 1000 euros ?

Illustration: *TSP*

Le célèbre problème du voyageur de commerce (*TSP*).

Un exemple: *TSP*-Europe

Je veux me rendre dans les capitales des 28 états membres de l'UE. Chaque déplacement en avion entre deux capitales me coûte un prix donné. Je ne veux jamais passer deux fois par la même ville. Existe-t-il un ordre de visite dont le prix est inférieur à 1000 euros ?

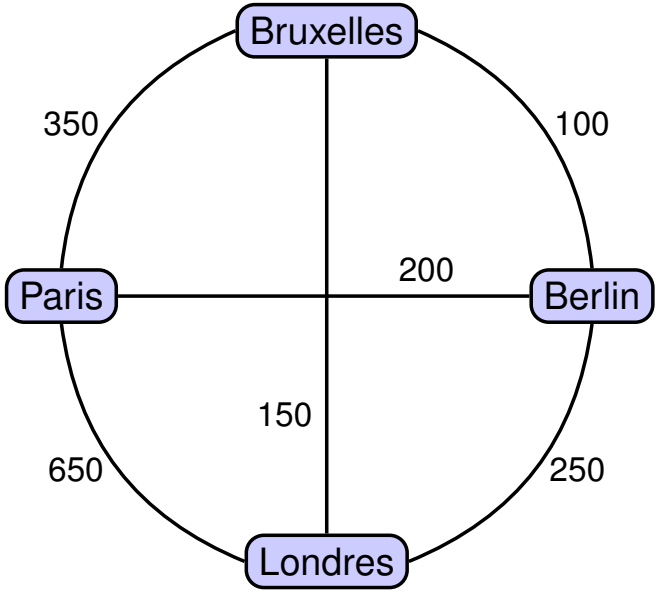
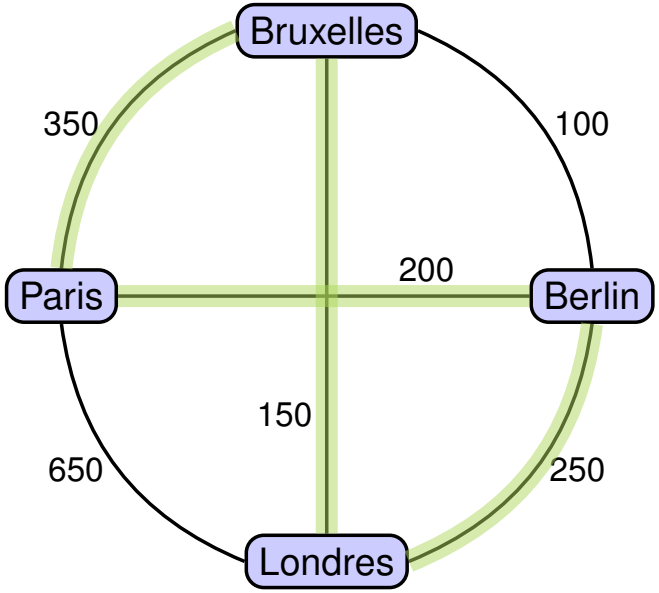


Illustration: *TSP*

Le célèbre problème du voyageur de commerce (*TSP*).

Un exemple: *TSP*-Europe

Je veux me rendre dans les capitales des 28 états membres de l'UE. Chaque déplacement en avion entre deux capitales me coûte un prix donné. Je ne veux jamais passer deux fois par la même ville. Existe-t-il un ordre de visite dont le prix est inférieur à 1000 euros ?



Un problème dans $\mathcal{EXP}$-TIME

Un algorithme pour le *TSP*.

Approche naïve

Un problème dans $\mathcal{EXP}$-TIME

Un algorithme pour le *TSP*.

Approche naïve

1. Ordonner les villes de toutes les façons possibles.

Un problème dans $\mathcal{EXP}$-TIME

Un algorithme pour le *TSP*.

Approche naïve

1. Ordonner les villes de toutes les façons possibles.
2. Pour chaque ordre, calculer le prix total du voyage.

Un problème dans $\mathcal{EXP}$-TIME

Un algorithme pour le *TSP*.

Approche naïve

1. Ordonner les villes de toutes les façons possibles.
2. Pour chaque ordre, calculer le prix total du voyage.
3. Vérifier si ce prix est inférieur à 1000 euros.

Un problème dans $\mathcal{EXP}$-TIME

Un algorithme pour le *TSP*.

Approche naïve

1. Ordonner les villes de toutes les façons possibles.
2. Pour chaque ordre, calculer le prix total du voyage.
3. Vérifier si ce prix est inférieur à 1000 euros.

$\Rightarrow$ Au plus le nombre de villes augmente, au plus l'étape 1 va prendre du temps.

- Pour 63 villes, il existe plus d'ordres que d'atomes dans l'univers observable.

On peut montrer que *TSP* est dans $\mathcal{EXP}$-TIME.

Efficacité des algorithmes pour le *TSP*

Peut-on faire mieux?

Personne n'a encore

Efficacité des algorithmes pour le *TSP*

Peut-on faire mieux?

Personne n'a encore

- trouvé une solution beaucoup plus efficace pour résoudre le *TSP*,

Efficacité des algorithmes pour le *TSP*

Peut-on faire mieux?

Personne n'a encore

- trouvé une solution beaucoup plus efficace pour résoudre le *TSP*,
- prouvé qu'il faille obligatoirement un temps exponentiel pour résoudre le *TSP*.

Efficacité des algorithmes pour le *TSP*

Peut-on faire mieux?

Personne n'a encore

- trouvé une solution beaucoup plus efficace pour résoudre le *TSP*,
- prouvé qu'il faille obligatoirement un temps exponentiel pour résoudre le *TSP*.

$\Rightarrow$ Il est possible que *TSP* soit dans $\mathcal{P}$.

Efficacité des algorithmes pour le *TSP*

Peut-on faire mieux?

Personne n'a encore

- trouvé une solution beaucoup plus efficace pour résoudre le *TSP*,
- prouvé qu'il faille obligatoirement un temps exponentiel pour résoudre le *TSP*.

$\Rightarrow$ Il est possible que *TSP* soit dans $\mathcal{P}$.

Remarque: Pour un ordre particulier des villes, il est "rapide" (polynomial) de vérifier si le prix du voyage est inférieur à 1000 euros.

Certificats

Une autre façon de mesurer la difficulté d'un langage:

- Supposons que j'affirme: $x \in L$, avec un argument supposé vous convaincre. A quel point est-il difficile de vérifier si mon argument est correct?

Certificats

Une autre façon de mesurer la difficulté d'un langage:

- Supposons que j'affirme: $x \in L$, avec un argument supposé vous convaincre. A quel point est-il difficile de vérifier si mon argument est correct?

Pour le problème *TETRIS*, si on nous donne (p, s):

- si l'on nous donne tous les mouvements m à faire dans le jeu,
- il est "facile" de vérifier si les mouvements m vident le plateau p à l'aide des pièces s.

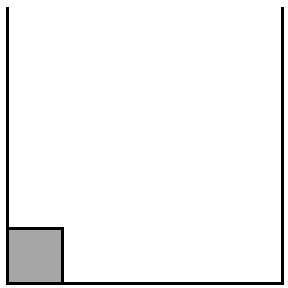
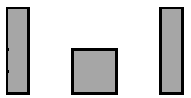
Certificats

Une autre façon de mesurer la difficulté d'un langage:

- Supposons que j'affirme: $x \in L$, avec un argument supposé vous convaincre. A quel point est-il difficile de vérifier si mon argument est correct?

Pour le problème *TETRIS*, si on nous donne (p, s):

- si l'on nous donne tous les mouvements m à faire dans le jeu,
- il est "facile" de vérifier si les mouvements m vident le plateau p à l'aide des pièces s.

J'affirme que $\Big($  ,  $\Big) \in$ *TETRIS*.

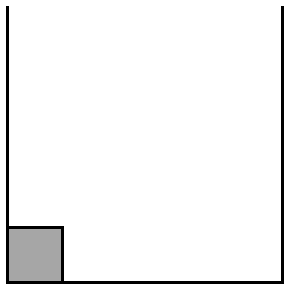
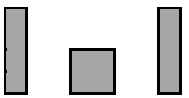
Certificats

Une autre façon de mesurer la difficulté d'un langage:

- Supposons que j'affirme: $x \in L$, avec un argument supposé vous convaincre. A quel point est-il difficile de vérifier si mon argument est correct?

Pour le problème *TETRIS*, si on nous donne (p, s):

- si l'on nous donne tous les mouvements m à faire dans le jeu,
- il est "facile" de vérifier si les mouvements m vident le plateau p à l'aide des pièces s.

J'affirme que $\Big($  ,  $\Big) \in$ *TETRIS*.

Rotation gauche de la pièce 1 placée aux colonnes 5,6,7,8. Placer la pièce 2 aux colonnes 3,4. Rotation gauche de la pièce 3 placée aux colonnes 5,6,7,8.

La classe $\mathcal{NP}$

Nous dirons qu'un langage L est dans $\mathcal{NP}$, si étant donné un certificat C, on peut vérifier en temps polynomial que C est vrai.

La classe $\mathcal{NP}$

Nous dirons qu'un langage L est dans $\mathcal{NP}$, si étant donné un certificat C, on peut vérifier en temps polynomial que C est vrai.

$\Rightarrow$ $\mathcal{NP}$ = "Nondeterministic Polynomial Time"

Non déterministe Etant donné un certificat, on le vérifie.

La classe $\mathcal{NP}$

Nous dirons qu'un langage L est dans $\mathcal{NP}$, si étant donné un certificat C, on peut vérifier en temps polynomial que C est vrai.

$\Rightarrow$ $\mathcal{NP}$ = "Nondeterministic Polynomial Time"

Non déterministe Etant donné un certificat, on le vérifie.

Ce n'est pas le cas pour tous les problèmes

Par exemple, pour le jeu d'échecs, la séquence de mouvements peut être trop longue à vérifier en temps polynomial.

La classe $\mathcal{NP}$

Nous dirons qu'un langage L est dans $\mathcal{NP}$, si étant donné un certificat C, on peut vérifier en temps polynomial que C est vrai.

$\Rightarrow$ $\mathcal{NP}$ = "Nondeterministic Polynomial Time"

Non déterministe Etant donné un certificat, on le vérifie.

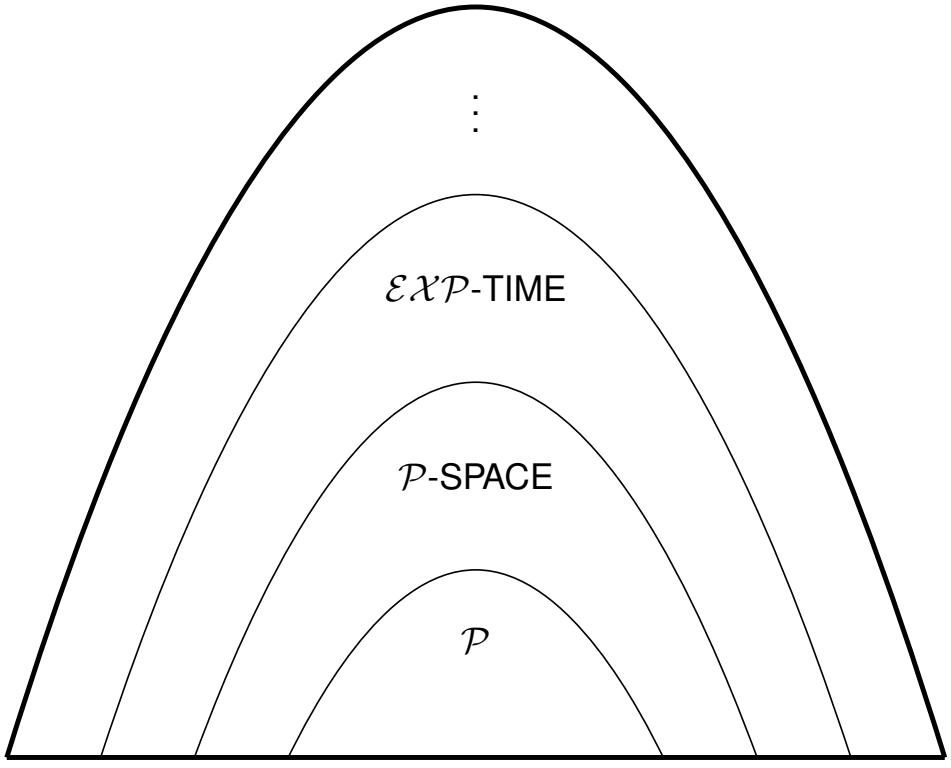
Ce n'est pas le cas pour tous les problèmes

Par exemple, pour le jeu d'échecs, la séquence de mouvements peut être trop longue à vérifier en temps polynomial.

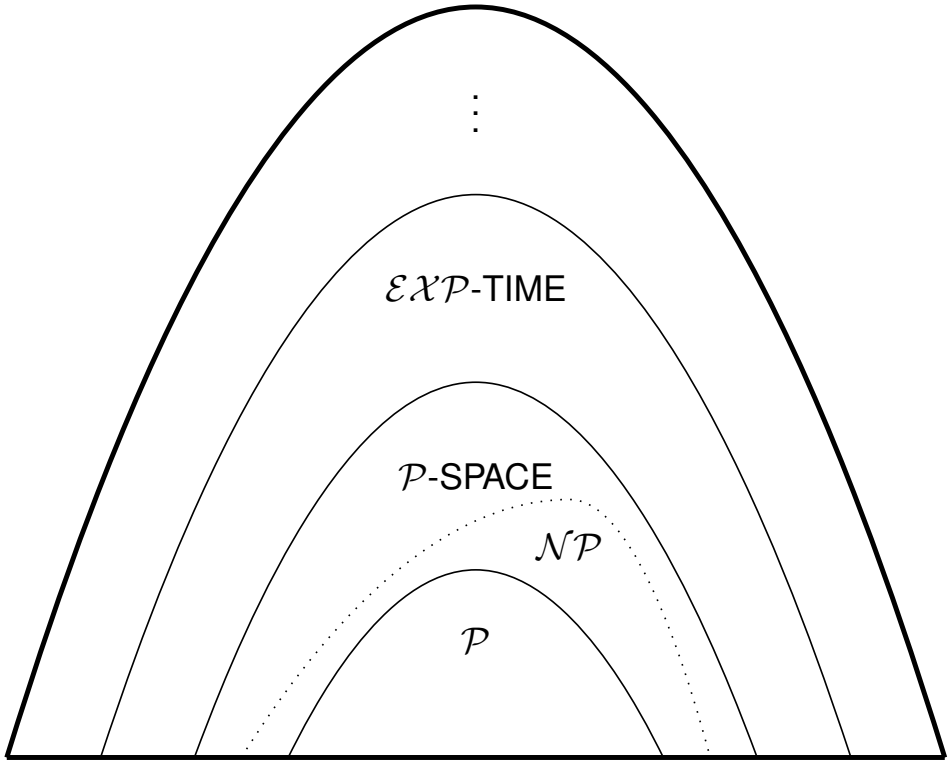
Résumé

- $L \in \mathcal{P}$ peut être rapidement décidé.
- $L \in \mathcal{NP}$ peut être rapidement vérifié.

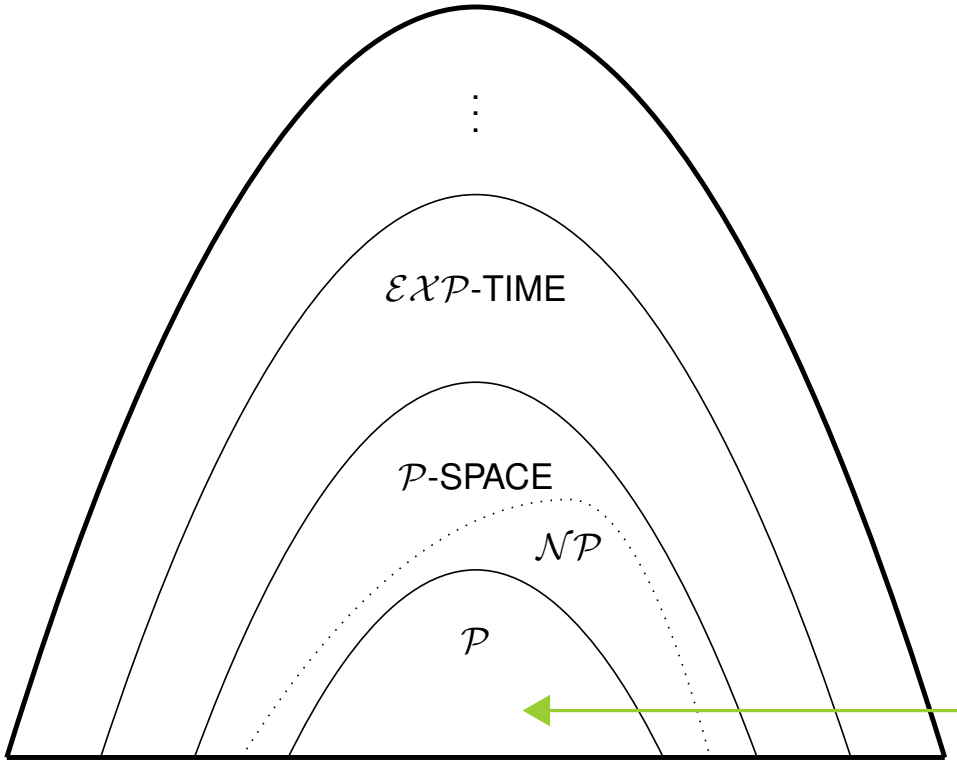
Hiérarchie avec exemples



Hiérarchie avec exemples

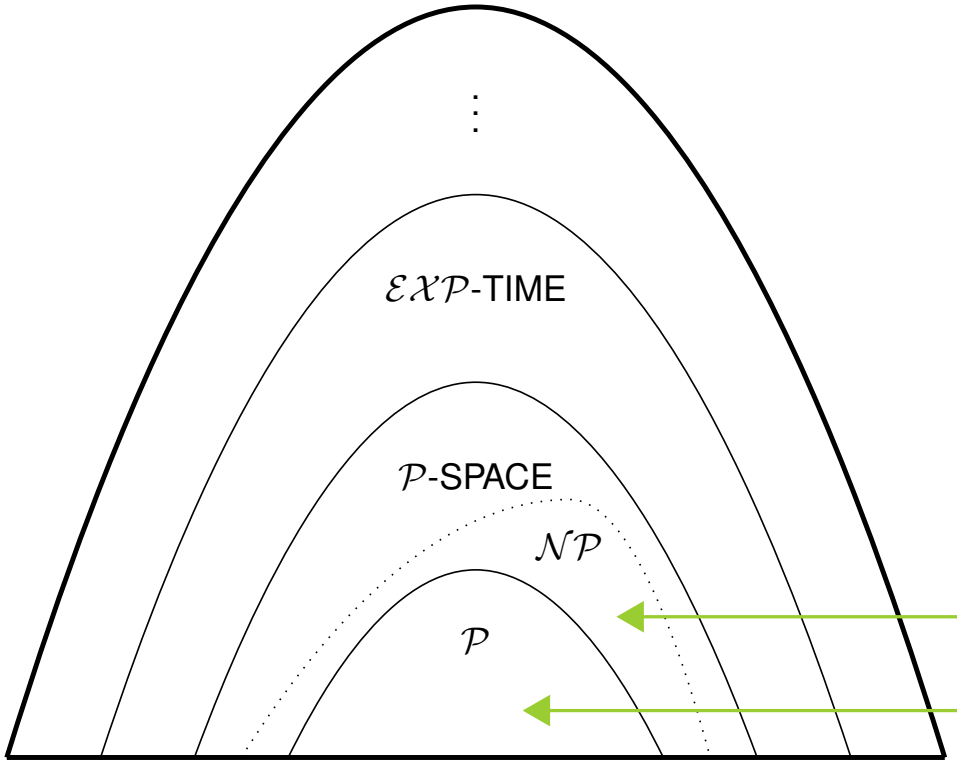


Hiérarchie avec exemples



JANGA, Deviner le nombre

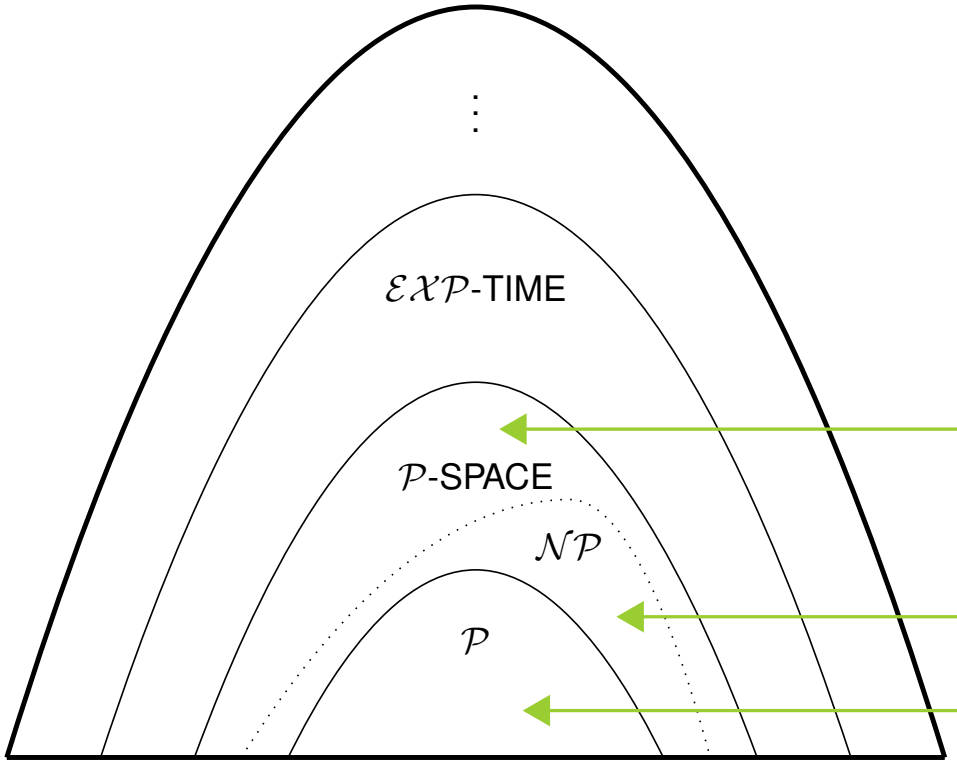
Hiérarchie avec exemples



Mastermind, Démineur, TSP

JANGA, Deviner le nombre

Hiérarchie avec exemples

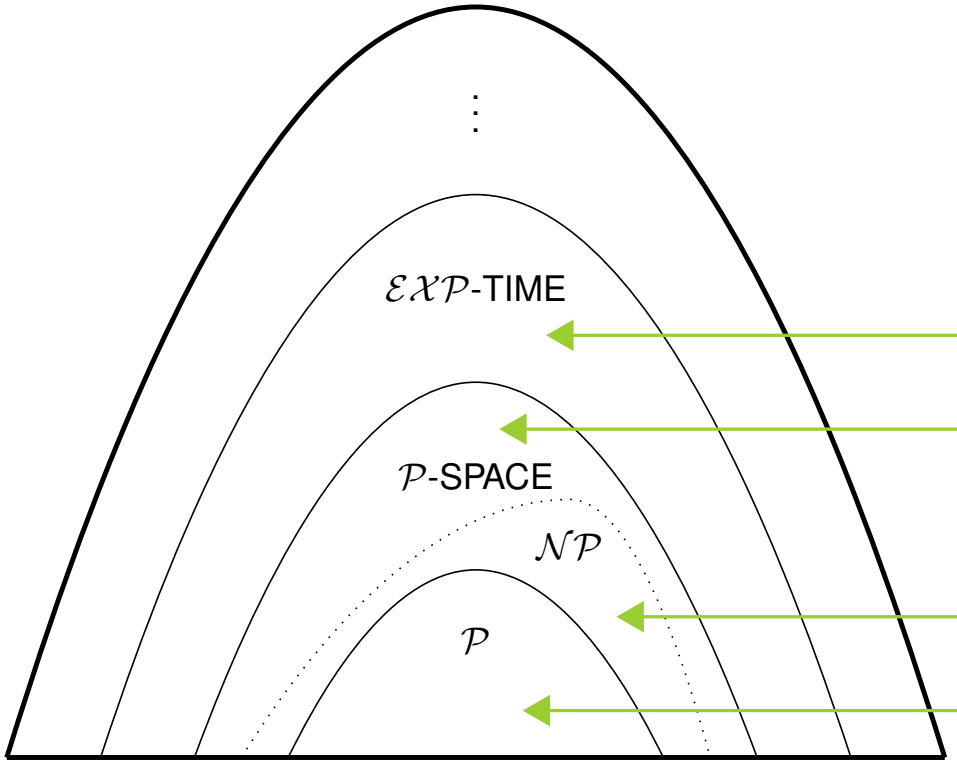


"Go", Sokoban

Mastermind, Démineur, TSP

JANGA, Deviner le nombre

Hiérarchie avec exemples



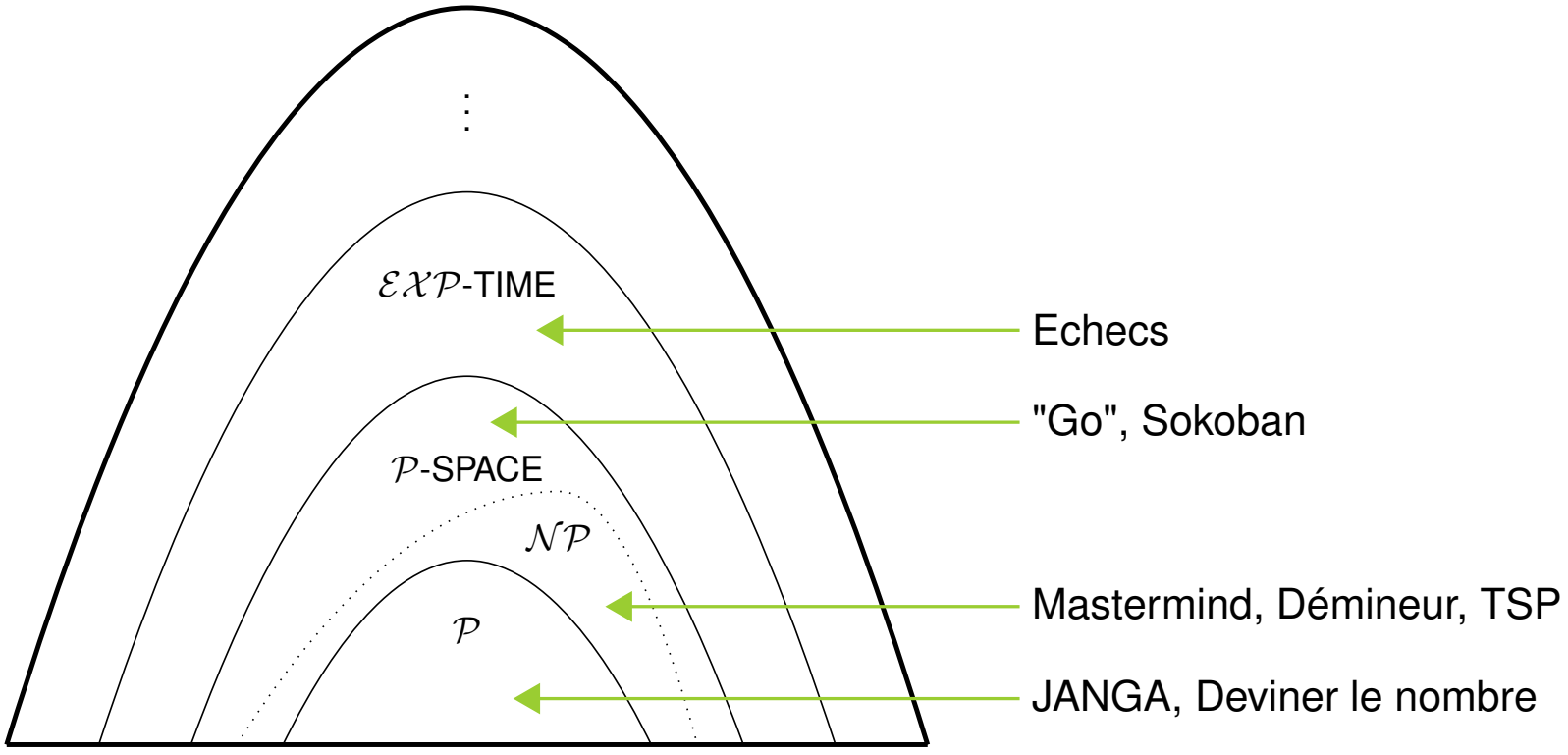
Echecs

"Go", Sokoban

Mastermind, Démineur, TSP

JANGA, Deviner le nombre

Hiérarchie avec exemples



Relations entre les classes

- $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{P}$-SPACE $\subseteq \mathcal{EXP}$-TIME
- Au moins une de ces inclusions est stricte.
  - Conjecture: elles sont toutes strictes.

$\mathcal{P}$ versus $\mathcal{NP}$

En particulier, nous ne savons pas si $\mathcal{P} = \mathcal{NP}$.

$\mathcal{P}$ versus $\mathcal{NP}$

En particulier, nous ne savons pas si $\mathcal{P} = \mathcal{NP}$.

- Un des plus grand problème irrésolu de l'informatique théorique (et des mathématiques).

$\mathcal{P}$ versus $\mathcal{NP}$

En particulier, nous ne savons pas si $\mathcal{P} = \mathcal{NP}$.

- Un des plus grand problème irrésolu de l'informatique théorique (et des mathématiques).
- Est-il plus facile de vérifier qu'une solution est correcte pour un problème que de résoudre le problème?

$\mathcal{P}$ versus $\mathcal{NP}$

En particulier, nous ne savons pas si $\mathcal{P} = \mathcal{NP}$.

- Un des plus grand problème irrésolu de l'informatique théorique (et des mathématiques).
- Est-il plus facile de vérifier qu'une solution est correcte pour un problème que de résoudre le problème?
- L'institut Clay offre un 1.000.000 $ pour une solution à ce problème.

$\mathcal{P}$ versus $\mathcal{NP}$

En particulier, nous ne savons pas si $\mathcal{P} = \mathcal{NP}$.

- Un des plus grand problème irrésolu de l'informatique théorique (et des mathématiques).
- Est-il plus facile de vérifier qu'une solution est correcte pour un problème que de résoudre le problème?
- L'institut Clay offre un 1.000.000 $ pour une solution à ce problème.

Applications multiples

- cryptographie,
- problèmes d'ordonnancements,
- confections d'horaires,
- bioinformatique,
- etc.

$\mathcal{NP}$ complétude

Premier résultat important sur la question $\mathcal{P}$ versus $\mathcal{NP}$:

$\Rightarrow$ Théorème de Cook-Levin (1970).

$\mathcal{NP}$ complétude

Premier résultat important sur la question $\mathcal{P}$ versus $\mathcal{NP}$:

$\Rightarrow$ Théorème de Cook-Levin (1970).

Un problème $\mathcal{NP}$-complet:

- est dans $\mathcal{NP}$,
- est aussi difficile que tout autre problème dans $\mathcal{NP}$.

$\mathcal{NP}$ complétude

Premier résultat important sur la question $\mathcal{P}$ versus $\mathcal{NP}$:

$\Rightarrow$ Théorème de Cook-Levin (1970).

Un problème $\mathcal{NP}$-complet:

- est dans $\mathcal{NP}$,
- est aussi difficile que tout autre problème dans $\mathcal{NP}$.

Implications

$\mathcal{NP}$ complétude

Premier résultat important sur la question $\mathcal{P}$ versus $\mathcal{NP}$:

$\Rightarrow$ Théorème de Cook-Levin (1970).

Un problème $\mathcal{NP}$-complet:

- est dans $\mathcal{NP}$,
- est aussi difficile que tout autre problème dans $\mathcal{NP}$.

Implications

- Si nous pouvons décider d'un seul problème $\mathcal{NP}$-complet en temps polynomial, alors nous pouvons décider de tous les problèmes de $\mathcal{NP}$ en temps polynomial.

$\mathcal{NP}$ complétude

Premier résultat important sur la question $\mathcal{P}$ versus $\mathcal{NP}$:

$\Rightarrow$ Théorème de Cook-Levin (1970).

Un problème $\mathcal{NP}$-complet:

- est dans $\mathcal{NP}$,
- est aussi difficile que tout autre problème dans $\mathcal{NP}$.

Implications

- Si nous pouvons décider d'un seul problème $\mathcal{NP}$-complet en temps polynomial, alors nous pouvons décider de tous les problèmes de $\mathcal{NP}$ en temps polynomial.
- S'il existe un problème qui est dans $\mathcal{NP}$ mais pas dans $\mathcal{P}$. Alors l'ensemble des problèmes $\mathcal{NP}$-complets sont extérieurs à $\mathcal{P}$.

Technique 1: être touché par la grâce

Théorème

SAT est $\mathcal{NP}$-complet.

— Cook et Levin, 1970

Technique 1: être touché par la grâce

Théorème

SAT est $\mathcal{NP}$-complet.

— Cook et Levin, 1970

Nous dirons qu'une formule logique ϕ donnée sous forme normale conjonctive est satisfaisable s'il existe une assignation des variables de ϕ telle que ϕ est vrai.

Technique 1: être touché par la grâce

Théorème

SAT est $\mathcal{NP}$-complet.

— Cook et Levin, 1970

Nous dirons qu'une formule logique ϕ donnée sous forme normale conjonctive est satisfaisable s'il existe une assignation des variables de ϕ telle que ϕ est vrai.

$$SAT = \{\phi : \phi \text{ est satisfaisable}\}$$

Technique 1: être touché par la grâce

Théorème

SAT est $\mathcal{NP}$-complet.

— Cook et Levin, 1970

Nous dirons qu'une formule logique ϕ donnée sous forme normale conjonctive est satisfaisable s'il existe une assignation des variables de ϕ telle que ϕ est vrai.

$$SAT = \{\phi : \phi \text{ est satisfaisable}\}$$

Exemple de SAT

La formule

$$\phi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$$

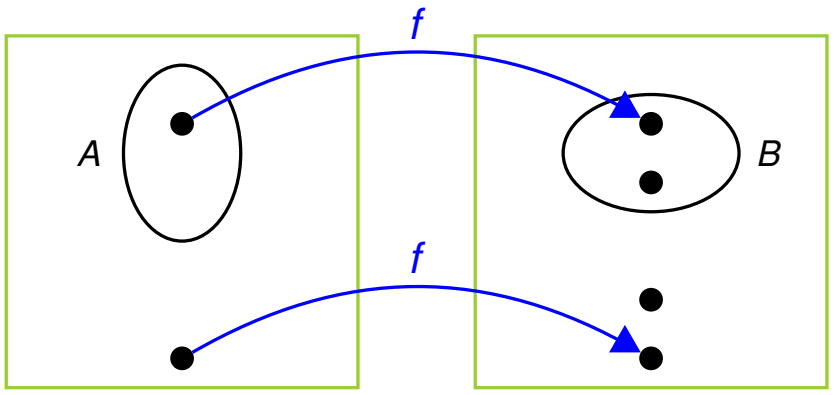
appartient à *SAT*, car si x_1 est vrai et x_2 est faux alors ϕ sera satisfaite.

Technique 2: réduction

Une réduction est une correspondance d'un problème à un autre.

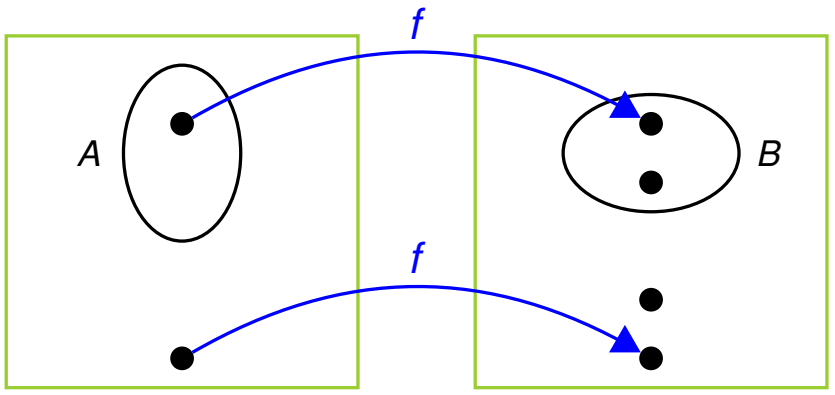
Technique 2: réduction

Une réduction est une correspondance d'un problème à un autre.



Technique 2: réduction

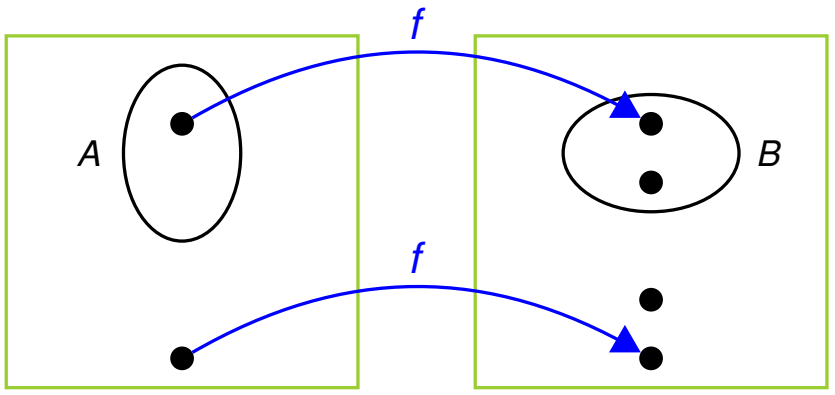
Une réduction est une correspondance d'un problème à un autre.



- $x \in A$ si et seulement si $f(x) \in B$
- Nous cherchons des f efficaces

Technique 2: réduction

Une réduction est une correspondance d'un problème à un autre.

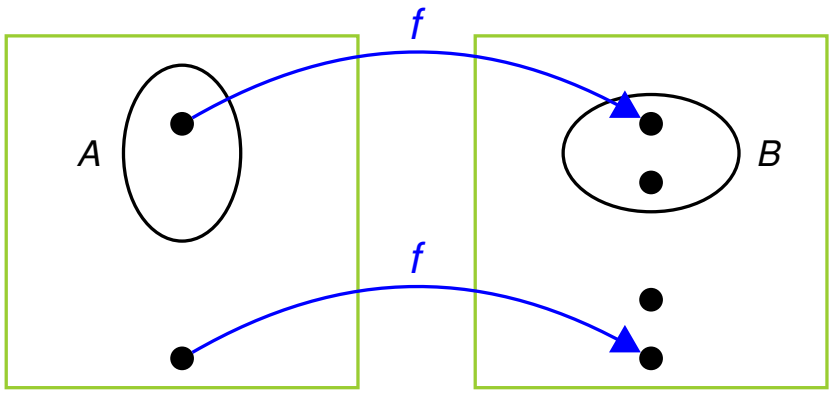


- $x \in A$ si et seulement si $f(x) \in B$
- Nous cherchons des f efficaces

Ce qui veut dire

Technique 2: réduction

Une réduction est une correspondance d'un problème à un autre.



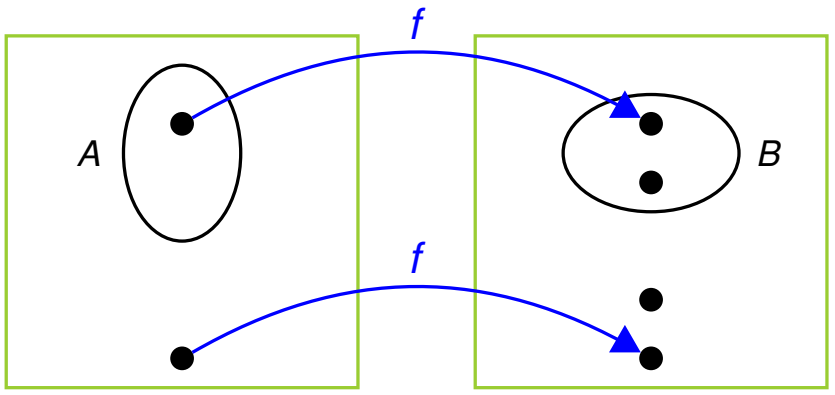
- $x \in A$ si et seulement si $f(x) \in B$
- Nous cherchons des f efficaces

Ce qui veut dire

- S'il existe un algorithme rapide pour B, il en existe un pour A.

Technique 2: réduction

Une réduction est une correspondance d'un problème à un autre.



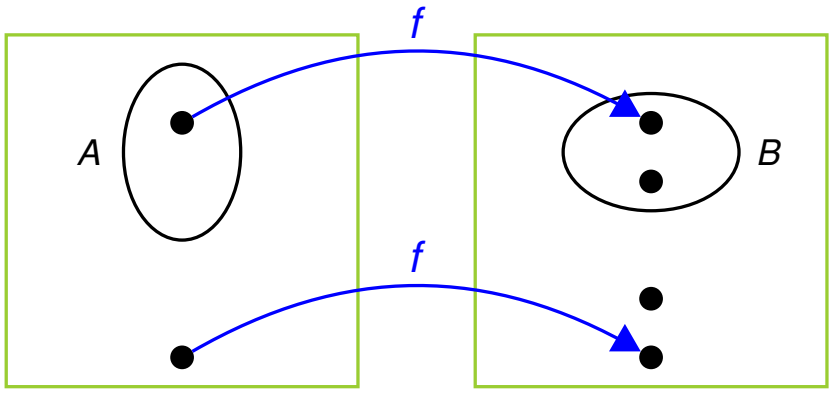
- $x \in A$ si et seulement si $f(x) \in B$
- Nous cherchons des f efficaces

Ce qui veut dire

- S'il existe un algorithme rapide pour B, il en existe un pour A.
- S'il n'existe pas d'algorithme rapide pour A, il n'en existe pas pour B

Technique 2: réduction

Une réduction est une correspondance d'un problème à un autre.



- $x \in A$ si et seulement si $f(x) \in B$
- Nous cherchons des f efficaces

Ce qui veut dire

- S'il existe un algorithme rapide pour B, il en existe un pour A.
- S'il n'existe pas d'algorithme rapide pour A, il n'en existe pas pour B

$\Rightarrow$ B est au moins aussi difficile que A.

Problèmes $\mathcal{NP}$-complets et réductions

Théorème

Si

- A est $\mathcal{NP}$-complet,
- B est $\mathcal{NP}$,
- il existe une réduction efficace A vers B.

Alors B est $\mathcal{NP}$-complet.

— Karp, 1972

Problèmes $\mathcal{NP}$-complets et réductions

Théorème

Si

- A est $\mathcal{NP}$-complet,
- B est $\mathcal{NP}$,
- il existe une réduction efficace A vers B.

Alors B est $\mathcal{NP}$-complet.

— Karp, 1972

Pour appliquer cela en pratique, il faut disposer d'un problème qu'on sait $\mathcal{NP}$-complet. Heureusement, nous avons le théorème de Cook-Levin.

Revenons au Tetris

Vous êtes venu pour parler du Tetris et pour l'instant vous êtes assez déçus...

Revenons au Tetris

Vous êtes venu pour parler du Tetris et pour l'instant vous êtes assez déçus...

Théorème

TETRIS est $\mathcal{NP}$-complet.

— Demaine, Hohenberger et Liben-Nowell, 2008

Revenons au Tetris

Vous êtes venu pour parler du Tetris et pour l'instant vous êtes assez déçus...

Théorème

TETRIS est $\mathcal{NP}$-complet.

— Demaine, Hohenberger et Liben-Nowell, 2008

Pour nous en convaincre, il nous faut réaliser les étapes suivantes:

Revenons au Tetris

Vous êtes venu pour parler du Tetris et pour l'instant vous êtes assez déçus...

Théorème

TETRIS est $\mathcal{NP}$-complet.

— Demaine, Hohenberger et Liben-Nowell, 2008

Pour nous en convaincre, il nous faut réaliser les étapes suivantes:

- définir une correspondance f entre les instances d'un problème $\mathcal{NP}$-complet connu L et *TETRIS*,

Revenons au Tetris

Vous êtes venu pour parler du Tetris et pour l'instant vous êtes assez déçus...

Théorème

TETRIS est $\mathcal{NP}$-complet.

— Demaine, Hohenberger et Liben-Nowell, 2008

Pour nous en convaincre, il nous faut réaliser les étapes suivantes:

- définir une correspondance f entre les instances d'un problème $\mathcal{NP}$-complet connu L et *TETRIS*,
- montrer que si $x \in L$ alors $f(x) \in$ *TETRIS*,

Revenons au Tetris

Vous êtes venu pour parler du Tetris et pour l'instant vous êtes assez déçus...

Théorème

TETRIS est $\mathcal{NP}$-complet.

— Demaine, Hohenberger et Liben-Nowell, 2008

Pour nous en convaincre, il nous faut réaliser les étapes suivantes:

- définir une correspondance f entre les instances d'un problème $\mathcal{NP}$-complet connu L et *TETRIS*,
- montrer que si $x \in L$ alors $f(x) \in$ *TETRIS*,
- montrer que si $x \notin L$ alors $f(x) \notin$ *TETRIS*.

3-*PARTITION*

Nous dirons qu'un multiensemble S de nombre est 3-partitionable si S peut être partitionné en sous-multiensembles de taille 3, tel que la somme des éléments dans chaque sous-multiensemble soit la même.

3-*PARTITION*

Nous dirons qu'un multiensemble S de nombre est 3-partitionable si S peut être partitionné en sous-multiensembles de taille 3, tel que la somme des éléments dans chaque sous-multiensemble soit la même.

$$3\text{-}PARTITION = \{S : S \text{ est 3-partitionable}\}$$

3-*PARTITION*

Nous dirons qu'un multiensemble S de nombre est 3-partitionable si S peut être partitionné en sous-multiensembles de taille 3, tel que la somme des éléments dans chaque sous-multiensemble soit la même.

$$3\text{-}PARTITION = \{S : S \text{ est 3-partitionable}\}$$

Théorème

3-*PARTITION* est $\mathcal{NP}$-complet.

— Garey et Johnson, 1975

3-*PARTITION*

Nous dirons qu'un multiensemble S de nombre est 3-partitionable si S peut être partitionné en sous-multiensembles de taille 3, tel que la somme des éléments dans chaque sous-multiensemble soit la même.

$$3\text{-}PARTITION = \{S : S \text{ est 3-partitionable}\}$$

Théorème

3-*PARTITION* est $\mathcal{NP}$-complet.

— Garey et Johnson, 1975

Exemple de 3-PART

Le multiensemble $S = \{26, 29, 33, 33, 33, 34, 35, 36, 41\}$ appartient à 3-*PARTITION*, car la partition suivante existe:

$$\{26, 33, 41\}, \{29, 35, 36\} \text{ et } \{33, 33, 34\}.$$

3-*PARTITION*

Nous dirons qu'un multiensemble S de nombre est 3-partitionable si S peut être partitionné en sous-multiensembles de taille 3, tel que la somme des éléments dans chaque sous-multiensemble soit la même.

$$3\text{-}PARTITION = \{S : S \text{ est 3-partitionable}\}$$

Théorème

3-*PARTITION* est $\mathcal{NP}$-complet.

— Garey et Johnson, 1975

Exemple de 3-PART

Le multiensemble $S = \{26, 29, 33, 33, 33, 34, 35, 36, 41\}$ appartient à 3-*PARTITION*, car la partition suivante existe:

$$\{26, 33, 41\}, \{29, 35, 36\} \text{ et } \{33, 33, 34\}.$$

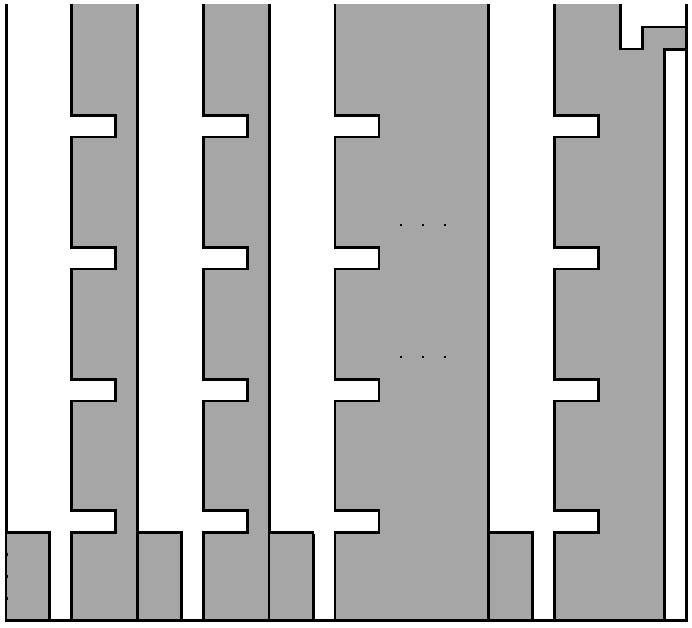
Observons maintenant un aperçu d'une réduction efficace de 3-*PARTITION* à *TETRIS*.

La réduction

Etant donnés les entiers suivants $a_1, a_2, \ldots a_{3s}$ et T la somme commune aux sous-multiensembles, nous construisons une instance de *TETRIS*.

La réduction

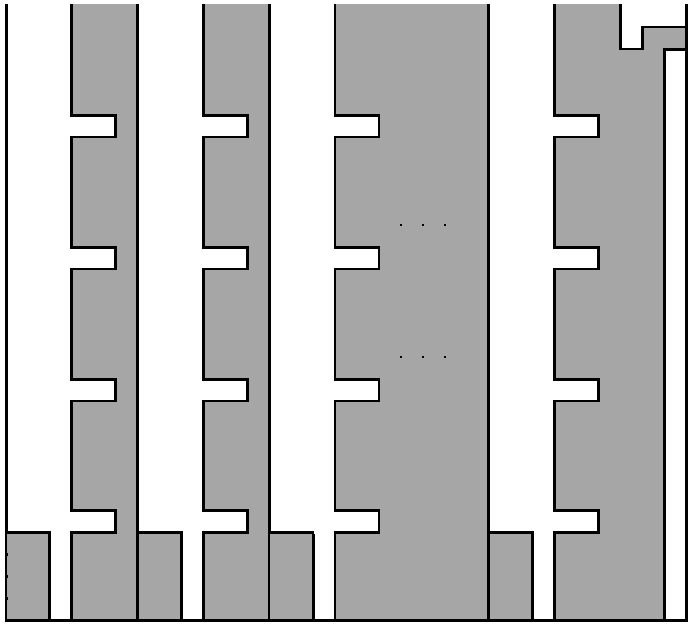
Etant donnés les entiers suivants $a_1, a_2, \ldots a_{3s}$ et T la somme commune aux sous-multiensembles, nous construisons une instance de *TETRIS*.



- Longueur tableau: $6s + 3$
- Hauteur tableau: $6T + 22$

La réduction

Etant donnés les entiers suivants $a_1, a_2, \ldots a_{3s}$ et T la somme commune aux sous-multiensembles, nous construisons une instance de *TETRIS*.

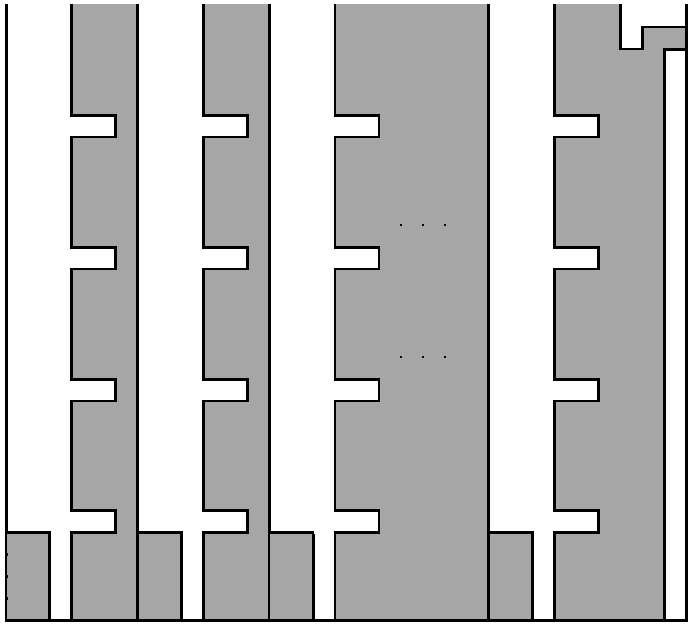


Pour tout $1 \leq i \leq 3s$: a_i devient

- Longueur tableau: $6s + 3$
- Hauteur tableau: $6T + 22$

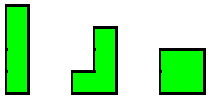
La réduction

Etant donnés les entiers suivants $a_1, a_2, \ldots a_{3s}$ et T la somme commune aux sous-multiensembles, nous construisons une instance de *TETRIS*.



Pour tout $1 \leq i \leq 3s$: a_i devient

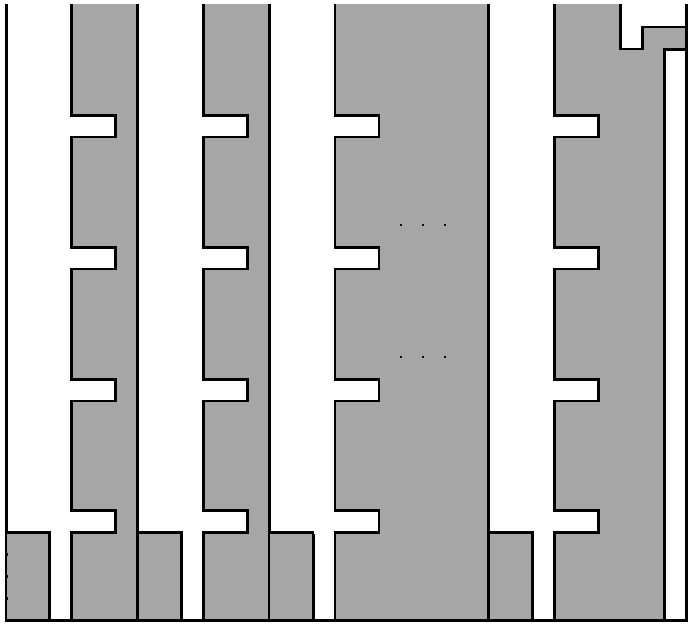
Initialisation:



- Longueur tableau: $6s + 3$
- Hauteur tableau: $6T + 22$

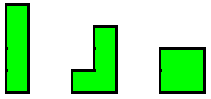
La réduction

Etant donnés les entiers suivants $a_1, a_2, \ldots a_{3s}$ et T la somme commune aux sous-multiensembles, nous construisons une instance de *TETRIS*.



Pour tout $1 \leq i \leq 3s$: a_i devient

Initialisation:

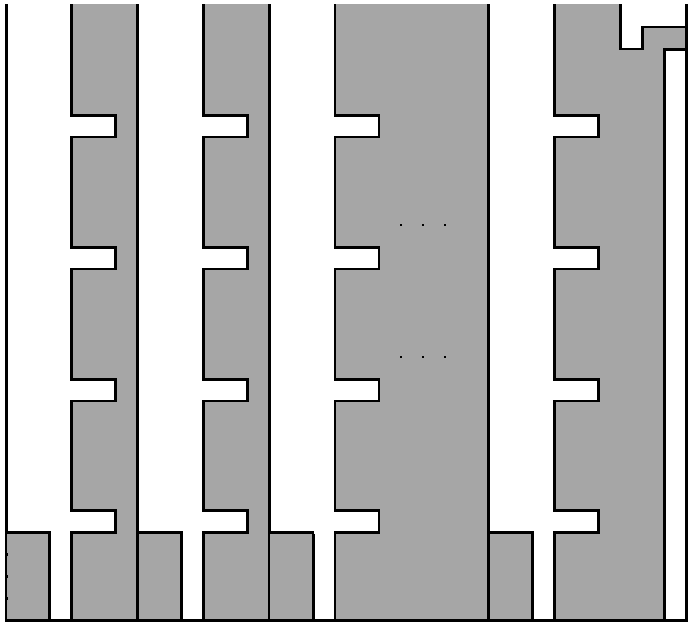


Remplissage: (a_i fois)

- Longueur tableau: $6s + 3$
- Hauteur tableau: $6T + 22$

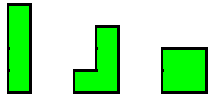
La réduction

Etant donnés les entiers suivants $a_1, a_2, \ldots a_{3s}$ et T la somme commune aux sous-multiensembles, nous construisons une instance de *TETRIS*.

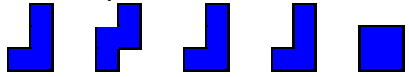


Pour tout $1 \leq i \leq 3s$: a_i devient

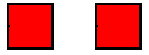
Initialisation:



Remplissage: (a_i fois)



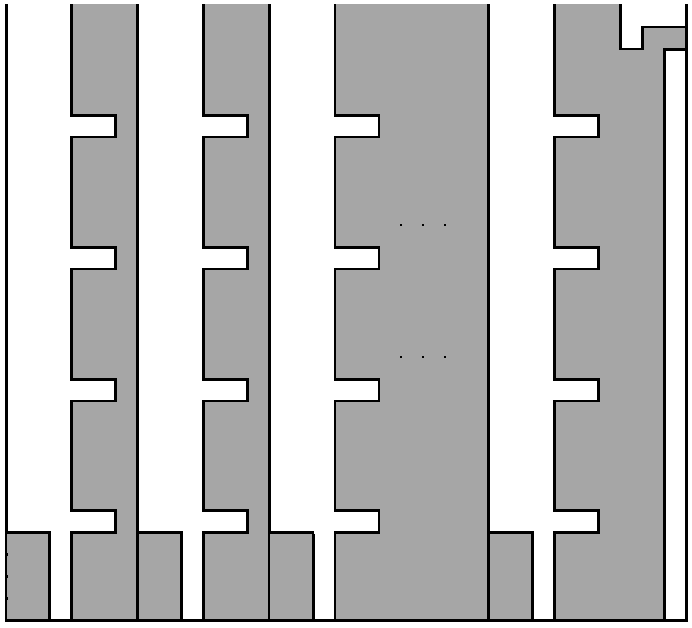
Clôture:



- Longueur tableau: $6s + 3$
- Hauteur tableau: $6T + 22$

La réduction

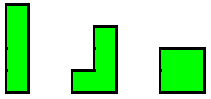
Etant donnés les entiers suivants $a_1, a_2, \ldots a_{3s}$ et T la somme commune aux sous-multiensembles, nous construisons une instance de *TETRIS*.



- Longueur tableau: $6s + 3$
- Hauteur tableau: $6T + 22$

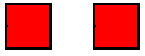
Pour tout $1 \leq i \leq 3s$: a_i devient

Initialisation:



Remplissage: (a_i fois)

Clôture:



Nous terminons avec

 $\times s$  $\times(\frac{3T}{2} + 5)$

Etape 1: OK

- ✓ définir une correspondance f entre une instance d'un problème $\mathcal{NP}$-complet connu L et *TETRIS*,
  - montrer que si $x \in L$ alors $f(x) \in$ *TETRIS*,
  - montrer que si $x \notin L$ alors $f(x) \notin$ *TETRIS*.

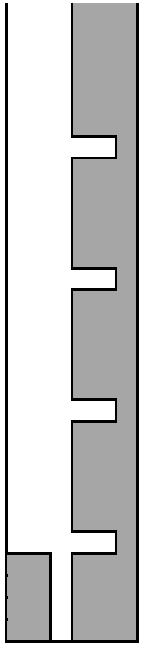
Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

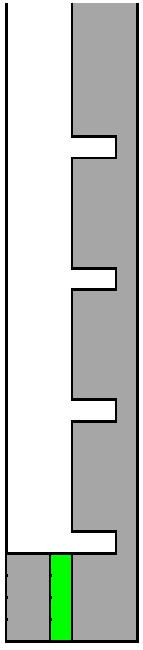
$\Rightarrow$ Empiler les tetrominos en fonction de la partition



Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

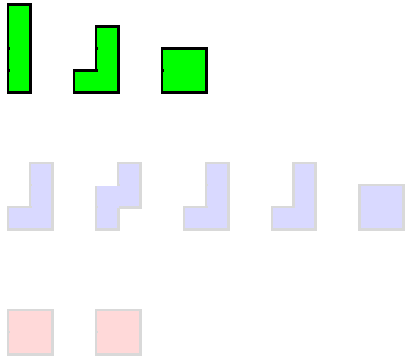
$\Rightarrow$ Empiler les tetrominos en fonction de la partition



Initialisation:

Remplissage:

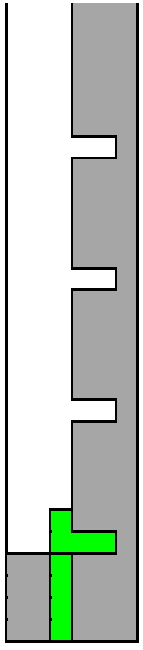
Clôture:



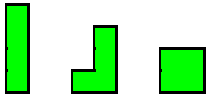
Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition



Initialisation:



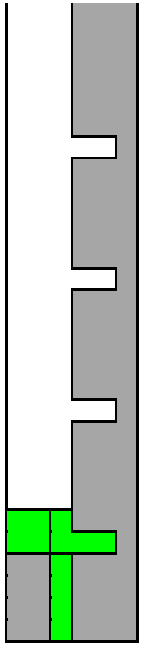
Remplissage:

Clôture:

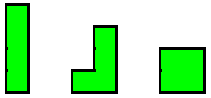
Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition



Initialisation:



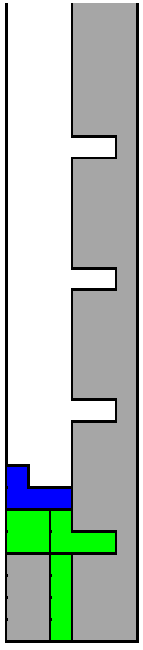
Remplissage:

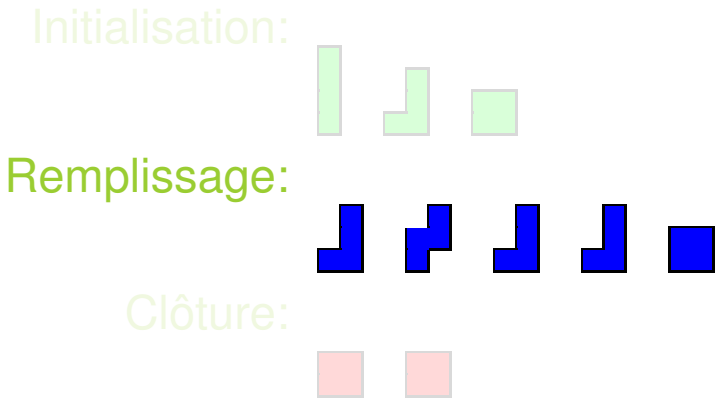
Clôture:

Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition

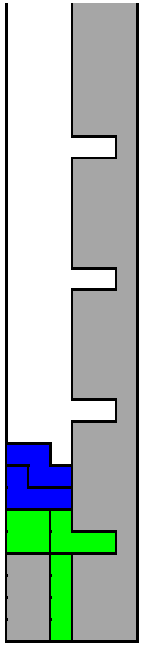


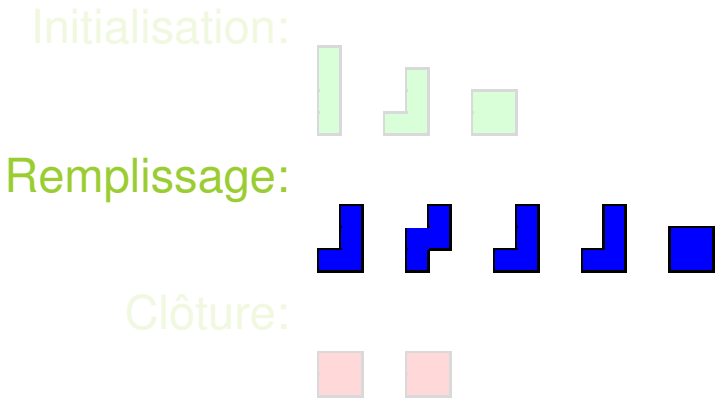


Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition

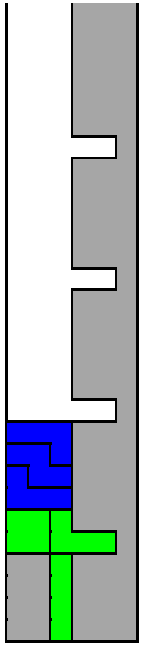


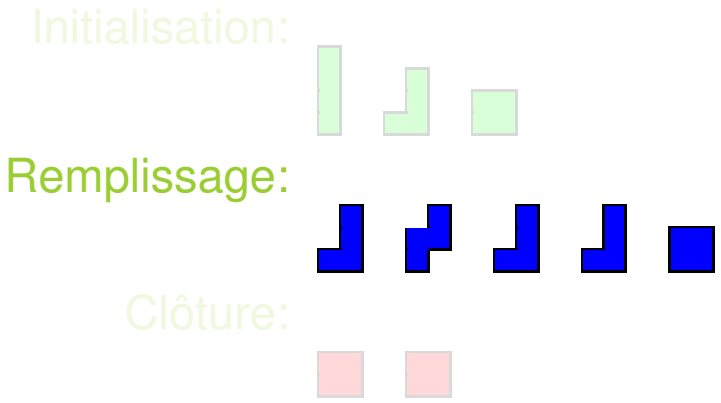


Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition

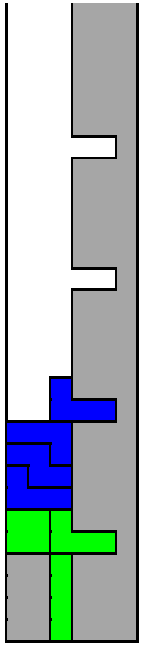


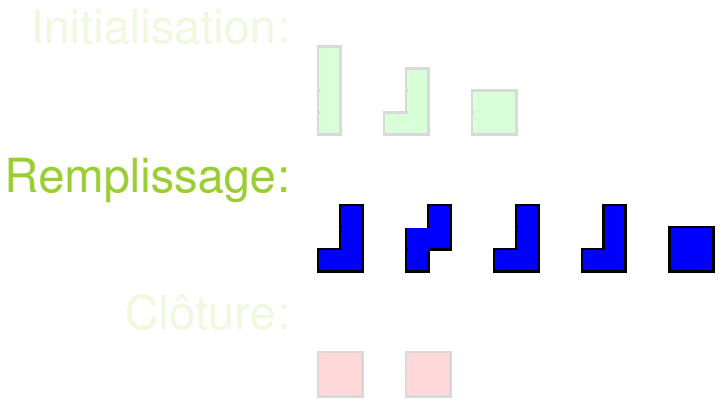


Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition

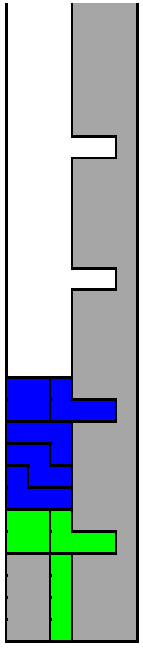


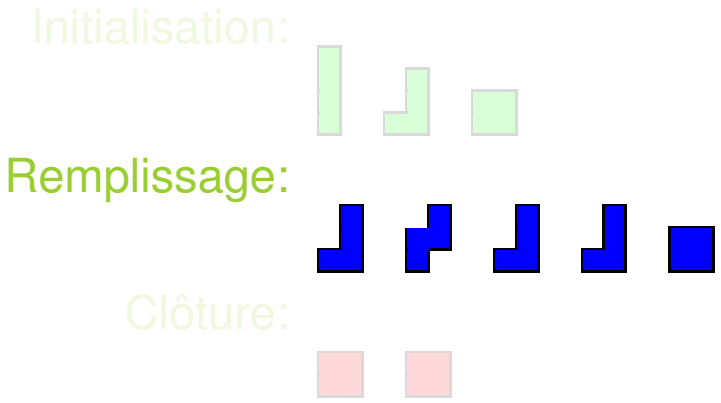


Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition

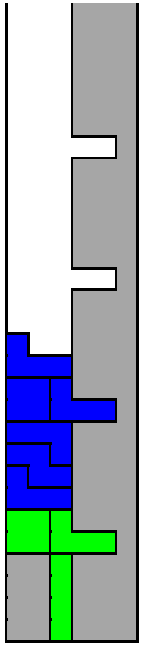




Oui veux dire oui

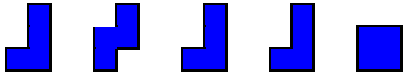
- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition



Initialisation:

Remplissage:

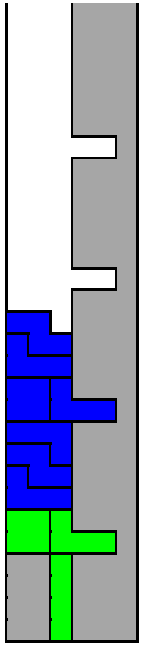


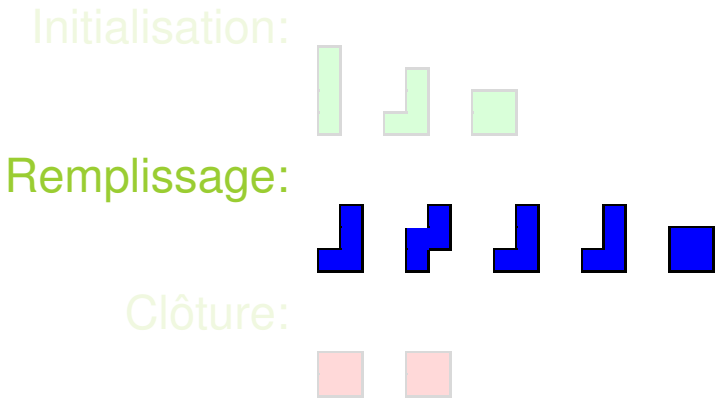
Clôture:

Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition

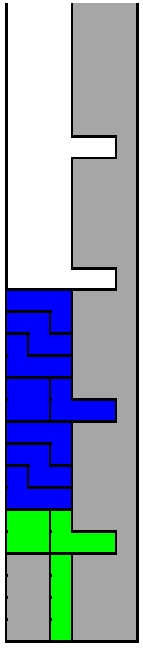




Oui veux dire oui

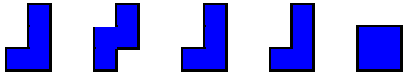
- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition



Initialisation:

Remplissage:

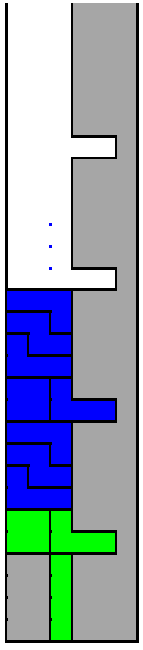


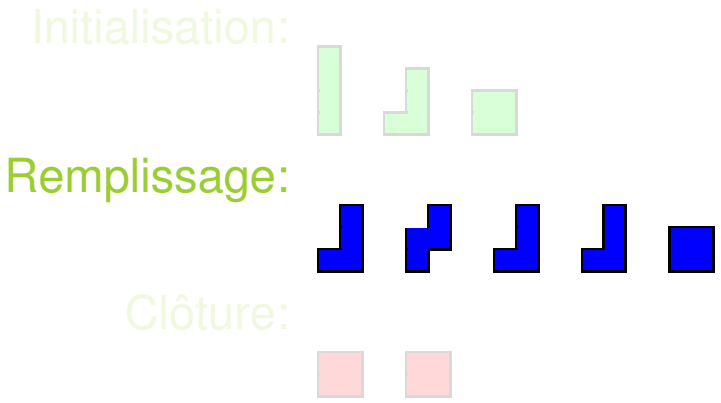
Clôture:

Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition

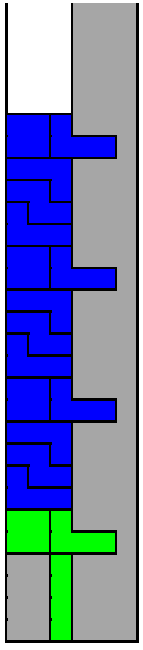


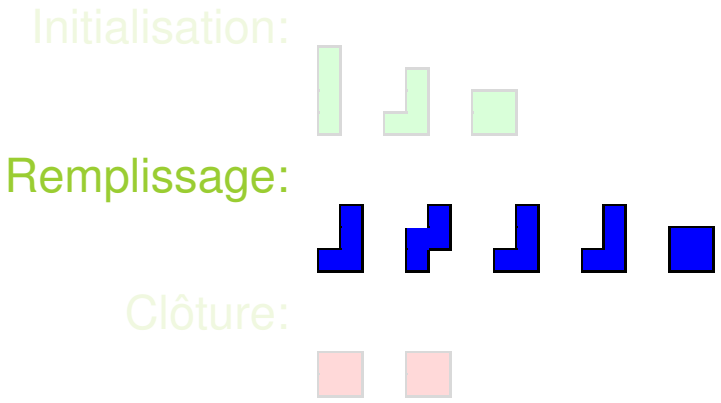


Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

$\Rightarrow$ Empiler les tetrominos en fonction de la partition

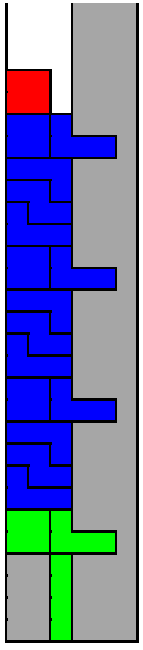




Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

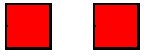
$\Rightarrow$ Empiler les tetrominos en fonction de la partition



Initialisation:

Remplissage:

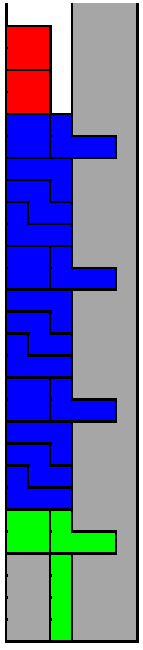
Clôture:



Oui veux dire oui

- Observons que si $x \in$ 3-*PARTITION* alors $f(x) \in$ *TETRIS*.

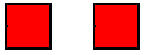
$\Rightarrow$ Empiler les tetrominos en fonction de la partition



Initialisation:

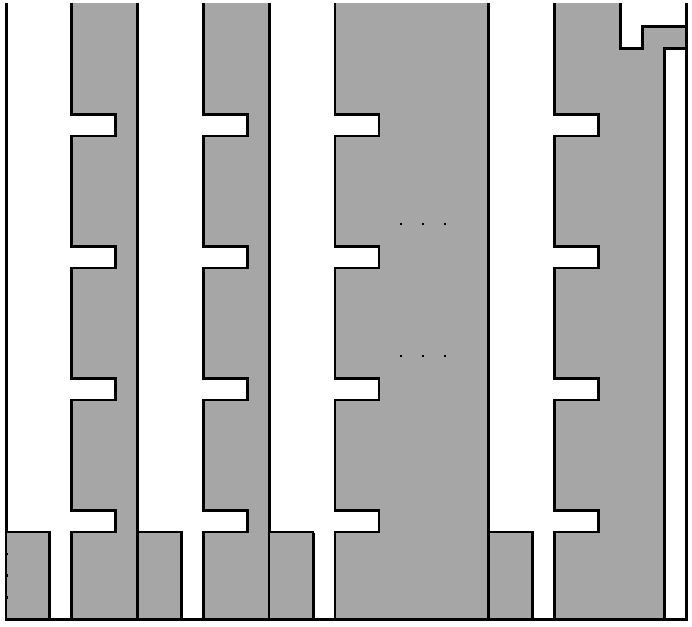
Remplissage:

Clôture:



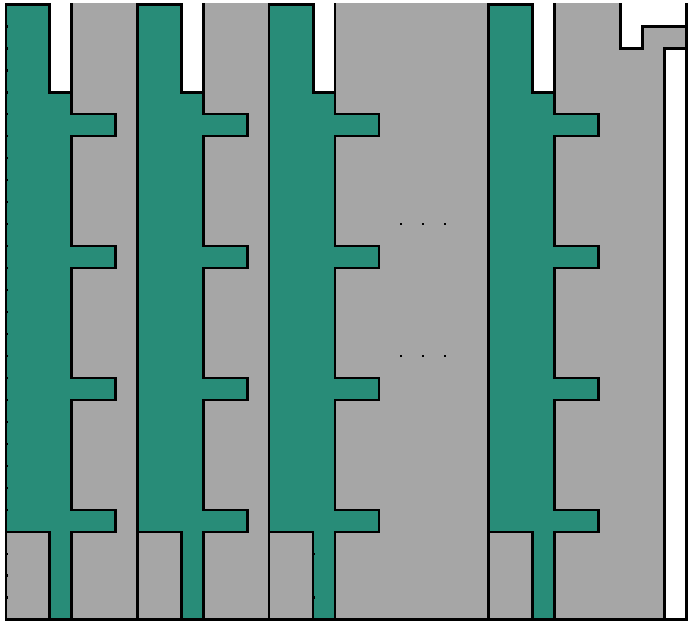
Et donc, oui veut dire oui

Chacun des sous-multiensembles à la même somme, donc les "pilles" dans le tableau ont la même hauteur.



Et donc, oui veut dire oui

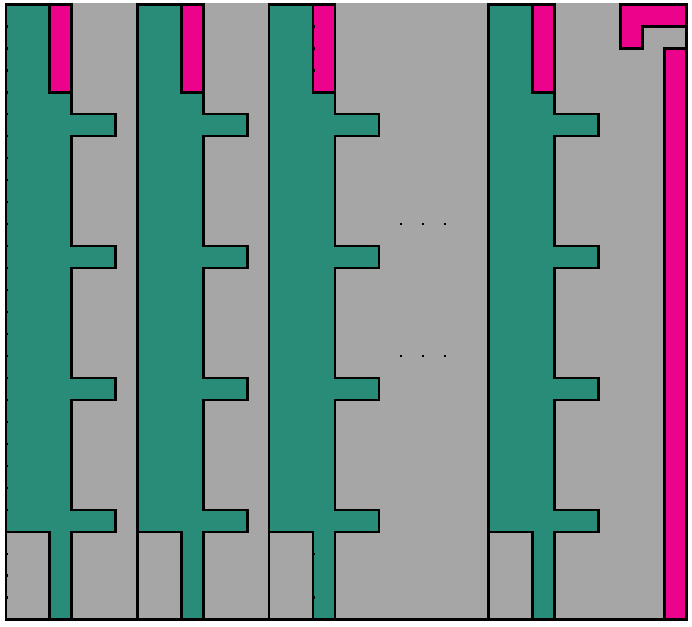
Chacun des sous-multiensembles à la même somme, donc les "pilles" dans le tableau ont la même hauteur.



- Chaque "trou" est remplis jusqu'en haut du plateau.

Et donc, oui veut dire oui

Chacun des sous-multiensembles à la même somme, donc les "pilles" dans le tableau ont la même hauteur.



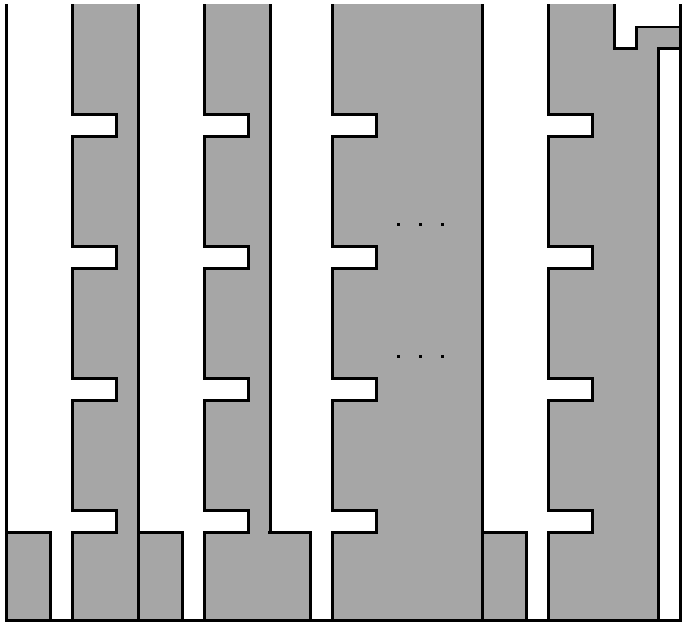
- Chaque "trou" est remplis jusqu'en haut du plateau.
- Les pièces restantes vont exactement vider le tableau.

Etape 2: OK

- ✓ définir une correspondance f entre une instance d'un problème $\mathcal{NP}$-complet connu L et *TETRIS*,
- ✓ montrer que si $x \in L$ alors $f(x) \in$ *TETRIS*,
- montrer que si $x \notin L$ alors $f(x) \notin$ *TETRIS*.

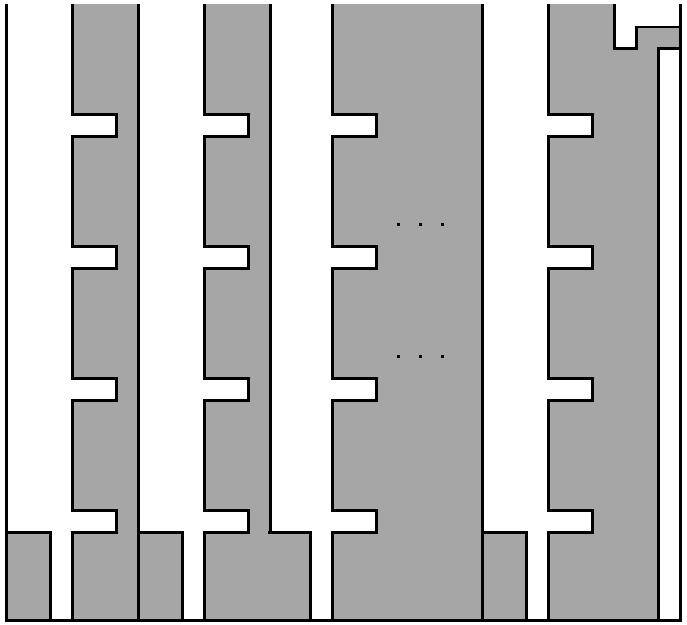
Non veux dire non

- Observons que si $x \notin$ 3-*PARTITION* alors $f(x) \notin$ *TETRIS*.



Non veux dire non

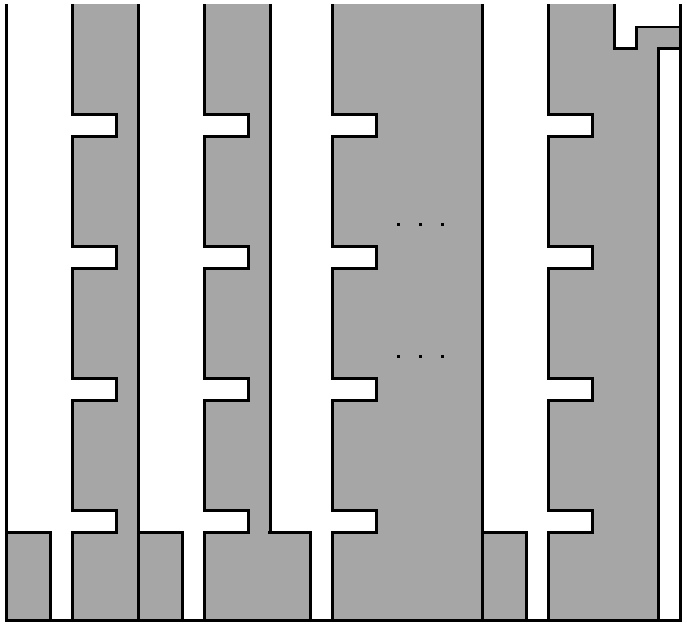
- Observons que si $x \notin$ 3-*PARTITION* alors $f(x) \notin$ *TETRIS*.



- Aucune ligne ne peut être effacée avant l'arrivé de la pièce en "L" en fin de séquence.

Non veux dire non

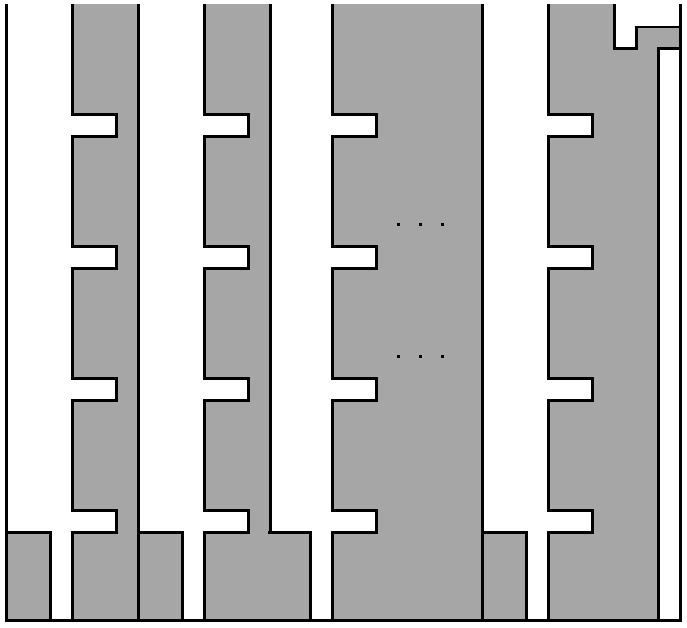
- Observons que si $x \notin$ 3-*PARTITION* alors $f(x) \notin$ *TETRIS*.



- Aucune ligne ne peut être effacée avant l'arrivé de la pièce en "L" en fin de séquence.
- Les trous doivent complètement être remplis sinon c'est l'échec.

Non veux dire non

- Observons que si $x \notin$ 3-*PARTITION* alors $f(x) \notin$ *TETRIS*.



- Aucune ligne ne peut être effacée avant l'arrivé de la pièce en "L" en fin de séquence.
- Les trous doivent complètement être remplis sinon c'est l'échec.
- Les alcôves peuvent très vite faire perdre le jeux.

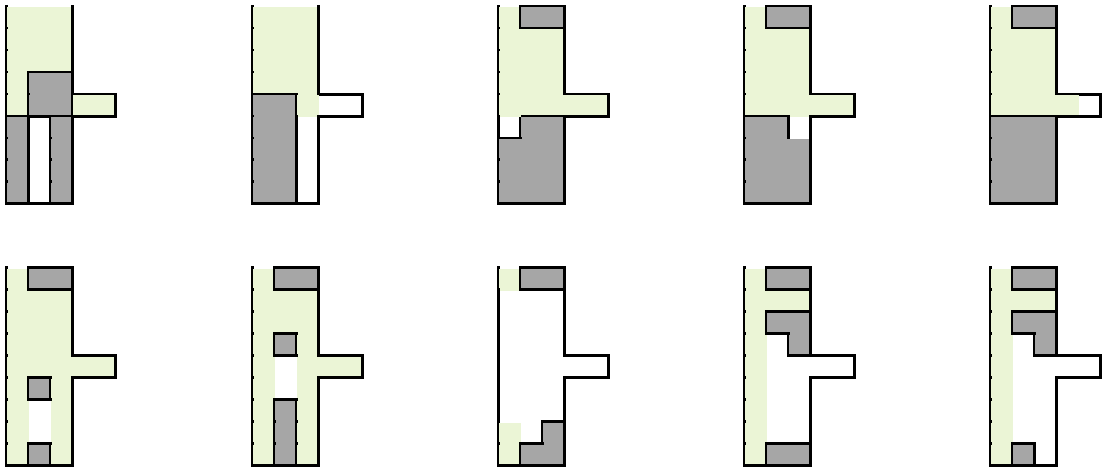
Et donc, non veut dire non

Si les a_i ne forment pas un élément de 3-*PARTITION*, alors le tableau ne sera jamais vidé.

Et donc, non veut dire non

Si les a_i ne forment pas un élément de 3-*PARTITION*, alors le tableau ne sera jamais vidé.

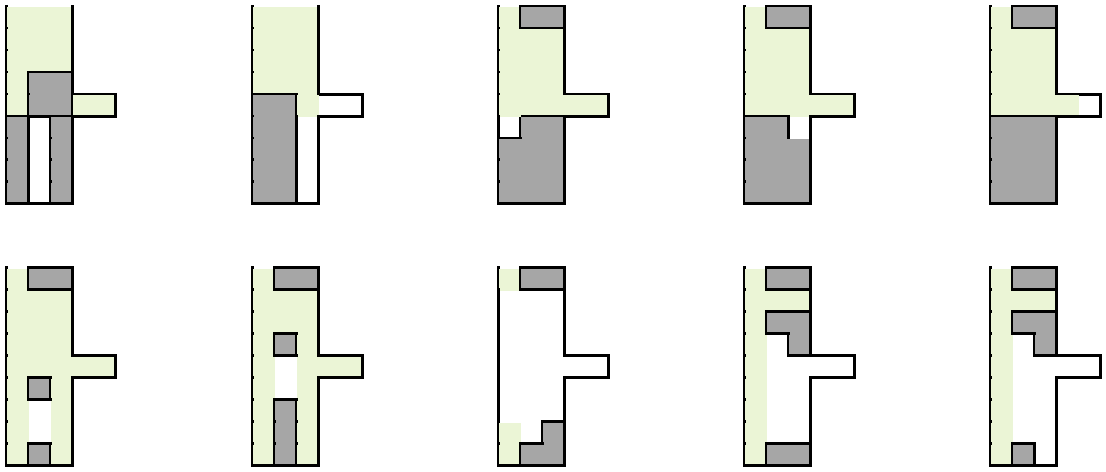
Etape 1 Tout mouvement différent du remplissage précédent conduit à une des configurations suivantes:



Et donc, non veut dire non

Si les a_i ne forment pas un élément de 3-*PARTITION*, alors le tableau ne sera jamais vidé.

Etape 1 Tout mouvement différent du remplissage précédent conduit à une des configurations suivantes:



Etape 2 Ces configurations conduisent inévitablement à l'échec.

Etape 3: OK

- ✓ définir une correspondance f entre une instance d'un problème $\mathcal{NP}$-complet connu L et *TETRIS*,
- ✓ montrer que si $x \in L$ alors $f(x) \in$ *TETRIS*,
- ✓ montrer que si $x \notin L$ alors $f(x) \notin$ *TETRIS*.

Etape 3: OK

- ✓ définir une correspondance f entre une instance d'un problème $\mathcal{NP}$-complet connu L et *TETRIS*,
- ✓ montrer que si $x \in L$ alors $f(x) \in$ *TETRIS*,
- ✓ montrer que si $x \notin L$ alors $f(x) \notin$ *TETRIS*.

$\Rightarrow$ *TETRIS* est $\mathcal{NP}$-complet.

Autres questions

Voici d'autres questions liées au problème *TETRIS*

Autres questions

Voici d'autres questions liées au problème *TETRIS*

- Peut-on vider au moins k lignes ?

Autres questions

Voici d'autres questions liées au problème *TETRIS*

- Peut-on vider au moins k lignes ?
- Peut-on placer au moins n pièces avant de perdre ?

Autres questions

Voici d'autres questions liées au problème *TETRIS*

- Peut-on vider au moins k lignes ?
- Peut-on placer au moins n pièces avant de perdre ?
- Peut-on placer toutes nos pièces sans remplir une case à la ligne h?

Autres questions

Voici d'autres questions liées au problème *TETRIS*

- Peut-on vider au moins k lignes ?
- Peut-on placer au moins n pièces avant de perdre ?
- Peut-on placer toutes nos pièces sans remplir une case à la ligne h?
- Peut-on réaliser au moins t lignes complètes ?

Autres questions

Voici d'autres questions liées au problème *TETRIS*

- Peut-on vider au moins k lignes ?
- Peut-on placer au moins n pièces avant de perdre ?
- Peut-on placer toutes nos pièces sans remplir une case à la ligne h?
- Peut-on réaliser au moins t lignes complètes ?

"Malheureusement"

Tous ces problèmes sont $\mathcal{NP}$-complets.

Un résultat sur le vrai jeu

Théorème

Une série suffisamment longue de pièces "S" et "Z" cause une défaite.

— Brzustowski, 1992

Un résultat sur le vrai jeu

Théorème

Une série suffisamment longue de pièces "S" et "Z" cause une défaite.

— Brzustowski, 1992

Ceci implique qu'un joueur de Tetris (même très doué) finira par perdre avec probabilité 1.

Un résultat sur le vrai jeu

Théorème

Une série suffisamment longue de pièces "S" et "Z" cause une défaite.

— Brzustowski, 1992

Ceci implique qu'un joueur de Tetris (même très doué) finira par perdre avec probabilité 1.

- Si on part du principe que les pièces données au joueur sont "bien" aléatoires.

Résultats sur d'autres jeux vidéo classiques

Avec ce système de réduction, il existe des résultats de complexité sur un bon nombre de jeux classiques:

Résultats sur d'autres jeux vidéo classiques

Avec ce système de réduction, il existe des résultats de complexité sur un bon nombre de jeux classiques:

Par exemple

- *Super Mario Bros* est $\mathcal{NP}$-difficile.
- *Donkey Kong Country* est $\mathcal{NP}$-difficile.
- *Super Metroid* est $\mathcal{NP}$-difficile.
- *Pokemon (Red/Blue)* est $\mathcal{NP}$-difficile.
- *Lemmings* est $\mathcal{NP}$-difficile.
- *Pac-man* est $\mathcal{NP}$-difficile.
- *Starcraft* est $\mathcal{NP}$-difficile.
- *Legend of Zelda: Ocarina of Time* est $\mathcal{P}$-space-difficile.
- *Doom* est $\mathcal{P}$-space-difficile.

Conclusion

- Beaucoup de problèmes intéressants sont $\mathcal{NP}$-complets.

Conclusion

- Beaucoup de problèmes intéressants sont $\mathcal{NP}$-complets.
  - Est-ce que $\mathcal{P} = \mathcal{NP}$?

Conclusion

- Beaucoup de problèmes intéressants sont $\mathcal{NP}$-complets.
  - Est-ce que $\mathcal{P} = \mathcal{NP}$?
  - En tous cas, pour l'instant nous avons plus facile à vérifier qu'une solution est correcte que d'en construire une à partir de rien.

Conclusion

- Beaucoup de problèmes intéressants sont $\mathcal{NP}$-complets.
  - Est-ce que $\mathcal{P} = \mathcal{NP}$?
  - En tous cas, pour l'instant nous avons plus facile à vérifier qu'une solution est correcte que d'en construire une à partir de rien.
- Un lien intéressant entre le jeux, les problèmes $\mathcal{NP}$-difficiles, et le raisonnement humain.

Conclusion

- Beaucoup de problèmes intéressants sont $\mathcal{NP}$-complets.
  - Est-ce que $\mathcal{P} = \mathcal{NP}$?
  - En tous cas, pour l'instant nous avons plus facile à vérifier qu'une solution est correcte que d'en construire une à partir de rien.
- Un lien intéressant entre le jeux, les problèmes $\mathcal{NP}$-difficiles, et le raisonnement humain.
  - Utile pour les créateurs de jeux, mais aussi les joueurs.

Conclusion

- Beaucoup de problèmes intéressants sont $\mathcal{NP}$-complets.
  - Est-ce que $\mathcal{P} = \mathcal{NP}$?
  - En tous cas, pour l'instant nous avons plus facile à vérifier qu'une solution est correcte que d'en construire une à partir de rien.
- Un lien intéressant entre le jeux, les problèmes $\mathcal{NP}$-difficiles, et le raisonnement humain.
  - Utile pour les créateurs de jeux, mais aussi les joueurs.
- Un moyen facile de gagner un million de dollars cette semaine:

Conclusion

- Beaucoup de problèmes intéressants sont $\mathcal{NP}$-complets.
  - Est-ce que $\mathcal{P} = \mathcal{NP}$?
  - En tous cas, pour l'instant nous avons plus facile à vérifier qu'une solution est correcte que d'en construire une à partir de rien.
- Un lien intéressant entre le jeux, les problèmes $\mathcal{NP}$-difficiles, et le raisonnement humain.
  - Utile pour les créateurs de jeux, mais aussi les joueurs.
- Un moyen facile de gagner un million de dollars cette semaine:
  - Montrer que *TETRIS* $\in \mathcal{P}$.

Il faut rendre à César...

Il faut rendre à César...

Erik D. Demaine, Susan Hohenberger et David Liben-Nowell.
Tetris is Hard, Even to Approximate.
COCOON'03, 351–363, 2003.

Il faut rendre à César...

Erik D. Demaine, Susan Hohenberger et David Liben-Nowell.
Tetris is Hard, Even to Approximate.
COCOON'03, 351–363, 2003.

Michael Sipser.
Introduction to the Theory of Computation.
International Thomson Publishing, 1996.

Il faut rendre à César...


Erik D. Demaine, Susan Hohenberger et David Liben-Nowell.
Tetris is Hard, Even to Approximate.
COCOON'03, 351–363, 2003.


Michael Sipser.
Introduction to the Theory of Computation.
International Thomson Publishing, 1996.


Greg Aloupis, Erik D. Demaine, Alan Guo et Giovanni Viglietta.
Classic Nintendo Games Are (Computationally) Hard.
Fun with Algorithms, Springer, 2014.

Il faut rendre à César...


Erik D. Demaine, Susan Hohenberger et David Liben-Nowell.
Tetris is Hard, Even to Approximate.
COCOON'03, 351–363, 2003.


Michael Sipser.
Introduction to the Theory of Computation.
International Thomson Publishing, 1996.


Greg Aloupis, Erik D. Demaine, Alan Guo et Giovanni Viglietta.
Classic Nintendo Games Are (Computationally) Hard.
Fun with Algorithms, Springer, 2014.


Giovanni Viglietta.
Gaming Is a Hard Job, but Someone Has to Do It!
Theory of Computing Systems, 54, 4, Springer, 2015.

Il faut rendre à César...


Erik D. Demaine, Susan Hohenberger et David Liben-Nowell.
Tetris is Hard, Even to Approximate.
COCOON'03, 351–363, 2003.

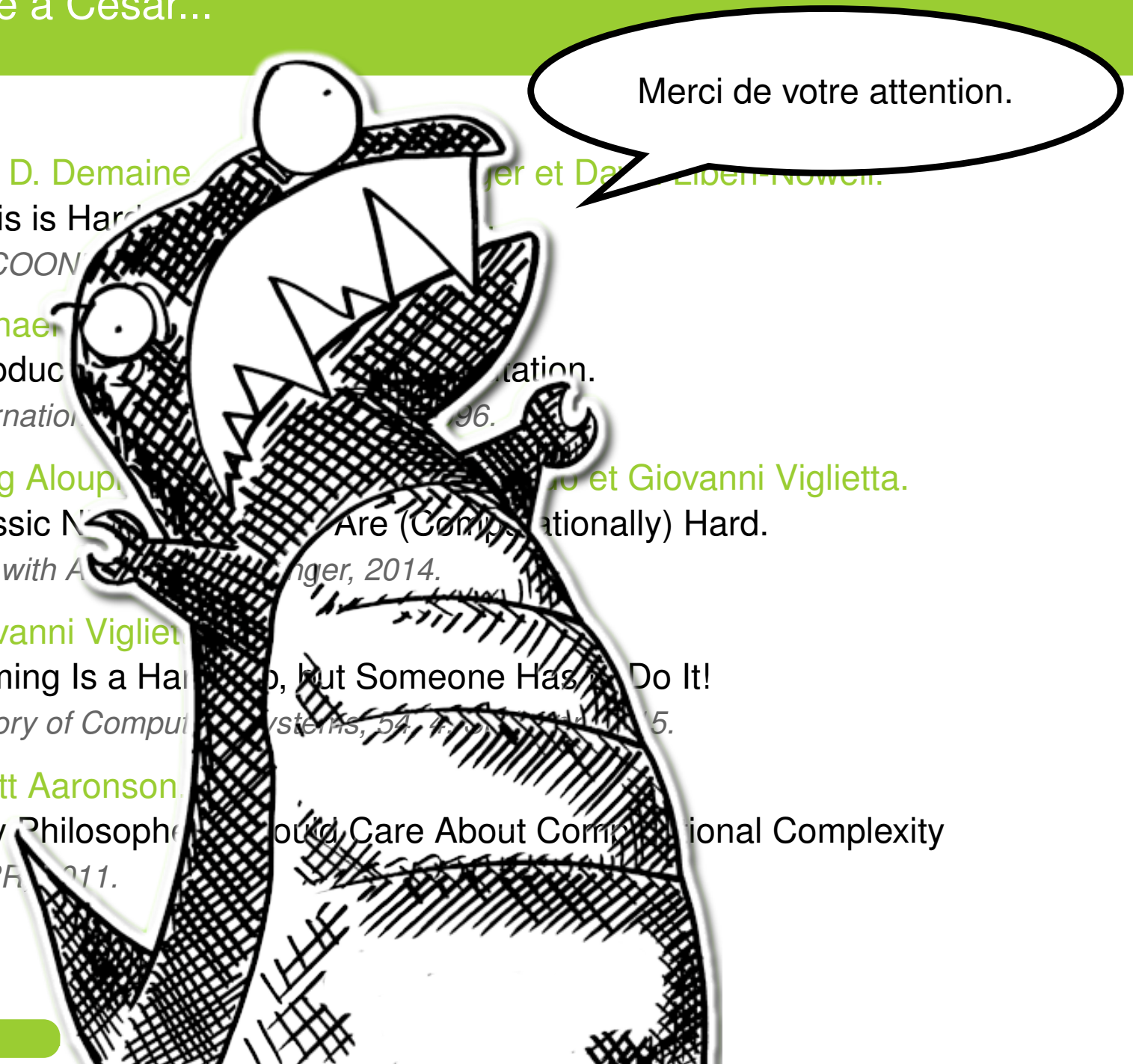

Michael Sipser.
Introduction to the Theory of Computation.
International Thomson Publishing, 1996.


Greg Aloupis, Erik D. Demaine, Alan Guo et Giovanni Viglietta.
Classic Nintendo Games Are (Computationally) Hard.
Fun with Algorithms, Springer, 2014.


Giovanni Viglietta.
Gaming Is a Hard Job, but Someone Has to Do It!
Theory of Computing Systems, 54, 4, Springer, 2015.


Scott Aaronson.
Why Philosophers Should Care About Computational Complexity
CoRR, 2011.

Il faut rendre à César...



Erik D. Demaine
Tetris is Har
COCOON

Michael
Introduc tation.
Internatio 96.

Greg Aloup o et Giovanni Viglietta.
Classic N Are (Com tionally) Hard.
Fun with A nger, 2014.

Giovanni Viglie
Gaming Is a Ha , but Someone Has to Do It!
Theory of Compu ystems, 54, 4 5.

Scott Aaronson
Why Philosoph ould Care About Com ional Complexity
CoRR 11.