Complexité

Jean-Marc Talbot

Université d'Aix-Marseille

DIU EIL, semaine 5, promo 2019

Encore de la complexité ?

Complexité (en temps) d'un algorithme : nombre d'opérations effectuées par un algorithme pour résoudre un problème (en fonction de la taille de l'entrée)

Vous connaissez

- Quelles opérations ?

Encore de la complexité ?

Complexité (en temps) d'un algorithme : nombre d'opérations effectuées par un algorithme pour résoudre un problème (en fonction de la taille de l'entrée)

Vous connaissez

- Quelles opérations ?
- Abstraction de la machine et des opérations : ordres de grandeur

$$\theta(n) \quad \theta(n \log n) \quad O(n^2) \quad \theta(n^3) \quad O(2^n)$$

Une autre complexité

Complexité (en temps) d'un problème : nombre d'opérations nécessaires pour résoudre un problème en fonction de la taille de l'entrée

- borne supérieure : le problème est soluble en $f(n)$ opérations (il existe un algorithme en $f(n)$ opérations qui résout le problème)
- borne inférieure : tout algorithme résolvant le problème a besoin d'au moins $g(n)$ opérations

n la taille de l'entrée

Borne inférieure de complexité : exemple

Tri par comparaison : trouver une permutation triée du tableau initial en comparant les éléments du tableau 2 à 2.

Borne inférieure de complexité : exemple

Tri par comparaison : trouver une permutation triée du tableau initial en comparant les éléments du tableau 2 à 2.

- Chaque comparaison scinde en 2 l'espace des permutations possibles (celles telles que $T[i] \leq T[j]$ et celles telles que $T[i] > T[j]$)
- Arbre de décision binaire étiqueté par des ensembles de permutation. L'arbre possède $n!$ feuilles (les singletons).
- $\log(n!)$ est une borne inférieure sur la hauteur de l'arbre. Par la formule de Stirling (ou la formule plus simple $\log(n!) \equiv n \log n$), cette branche est au moins de longueur $n \log n$ et comporte autant de comparaisons.

Borne inférieure de complexité : exemple

Tri par comparaison : trouver une permutation triée du tableau initial en comparant les éléments du tableau 2 à 2.

- Chaque comparaison scinde en 2 l'espace des permutations possibles (celles telles que $T[i] \leq T[j]$ et celles telles que $T[i] > T[j]$)
- Arbre de décision binaire étiqueté par des ensembles de permutation. L'arbre possède $n!$ feuilles (les singletons).
- $\log(n!)$ est une borne inférieure sur la hauteur de l'arbre. Par la formule de Stirling (ou la formule plus simple $\log(n!) \equiv n \log n$), cette branche est au moins de longueur $n \log n$ et comporte autant de comparaisons.

!! Le tri par casiers est linéaire dans la taille du tableau

Complexité de problème

Formellement,

Complexité d'un problème : nombre de transitions exécutées pour une machine de Turing pour résoudre le problème (en fonction de la taille de l'entrée)

Complexité de problème

Formellement,

Complexité d'un problème : nombre de transitions exécutées pour une machine de Turing pour résoudre le problème (en fonction de la taille de l'entrée)

Classes de complexité : ensemble de problèmes nécessitant la même quantité de ressources pour être résolus

Très gros grain : polynomial, exponentiel , ...

Encore des problèmes ...

Problème : calculer une information sur des données d'entrée

Instance d'un problème : valeur des données d'entrée

Encore des problèmes ...

Problème : calculer une information sur des données d'entrée
Instance d'un problème : valeur des données d'entrée

Problème d'optimisation

[Min-colorabilité]

entrée : G un graphe non dirigé

sortie : le nombre minimal de couleurs pour colorer G

Problème de décision

[3-colorabilité]

entrée : G un graphe non dirigé

sortie : oui si une coloration de G avec 3 couleurs existe

Encore des problèmes ...

Problème : calculer une information sur des données d'entrée
Instance d'un problème : valeur des données d'entrée

Problème d'optimisation

[Min-colorabilité]

entrée : G un graphe non dirigé

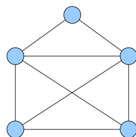
sortie : le nombre minimal de couleurs pour colorer G

Problème de décision

[3-colorabilité]

entrée : G un graphe non dirigé

sortie : oui si une coloration de G avec 3 couleurs existe



Encore des problèmes ...

Problème : calculer une information sur des données d'entrée
Instance d'un problème : valeur des données d'entrée

Problème d'optimisation

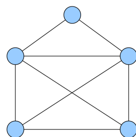
[Min-colorabilité]

entrée : G un graphe non dirigé
sortie : le nombre minimal de couleurs pour colorer G

Problème de décision

[3-colorabilité]

entrée : G un graphe non dirigé
sortie : oui si une coloration de G avec 3 couleurs existe



On se focalise sur les problèmes de décision : la machine de Turing accepte ou rejette l'entrée

La classe de complexité $\mathbf{P}$

$\mathbf{P}$ est la classe des problèmes polynomiaux (en temps)

Définition : $P \in \mathbf{P}$ si P est solvable sur une machine de Turing en temps polynomial dans la taille de l'entrée.

$\mathbf{P}$ contient les problèmes « faciles »

La classe de complexité $\mathbf{P}$

$\mathbf{P}$ est la classe des problèmes polynomiaux (en temps)

Définition : $P \in \mathbf{P}$ si P est solvable sur une machine de Turing en temps polynomial dans la taille de l'entrée.

$\mathbf{P}$ contient les problèmes « faciles »

$\mathbf{P}$ est robuste et indépendant du modèle de calcul.

La classe $\mathbf{P}$: exemples

[Appartenance]

entrée : Un élément e et un ensemble S donné en extension (liste)

sortie : oui si e appartient à S

A vous de jouer !

La classe **P** : exemples

[Appartenance]

entrée : Un élément e et un ensemble S donné en extension (liste)

sortie : oui si e appartient à S

A vous de jouer !

[IdentitéMatrice]

entrée : Une matrice M de taille $n \times n$

sortie : oui si M est l'identité

A vous de jouer !

La classe **P** : exemples (II)

[connexité]

entrée : G un graphe non dirigé

sortie : oui si G est connexe

A vous de jouer !

La classe de complexité **NP**

NP est la classe des problèmes qu'on peut vérifier en temps polynomial

La classe de complexité **NP**

NP est la classe des problèmes qu'on peut vérifier en temps polynomial

- Pour toute solution au problème, il existe un certificat de taille polynomiale pour cette solution.
- Il existe une machine de Turing qui partant d'une bande contenant l'entrée du problème et le certificat (de taille polynomiale) vérifie en temps polynomial que le certificat est bien témoin d'une solution.

La classe de complexité **NP**

NP est la classe des problèmes qu'on peut vérifier en temps polynomial

- Pour toute solution au problème, il existe un certificat de taille polynomiale pour cette solution.
- Il existe une machine de Turing qui partant d'une bande contenant l'entrée du problème et le certificat (de taille polynomiale) vérifie en temps polynomial que le certificat est bien témoin d'une solution.

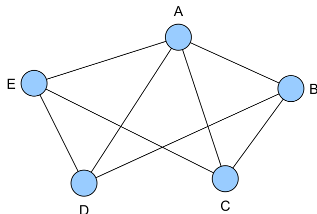
- $\mathbf{P} \subseteq \mathbf{NP}$: il suffit d'ignorer le certificat.

La classe **NP** : exemples (I)

[HAMILTONIEN]

entrée : G un graphe non dirigé

sortie : oui si G contient un cycle hamiltonien (un cycle passant par tous les sommets une et une seule fois)

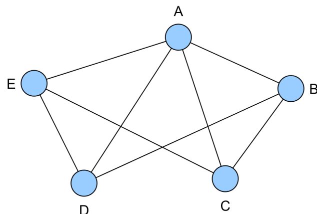


La classe **NP** : exemples (I)

[HAMILTONIEN]

entrée : G un graphe non dirigé

sortie : oui si G contient un cycle hamiltonien (un cycle passant par tous les sommets une et une seule fois)



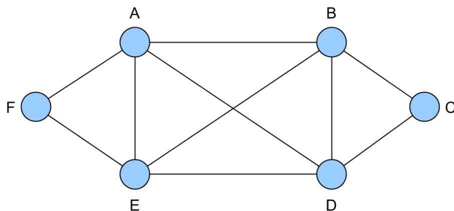
Certificat : A-E-D-B-C

La classe **NP** : exemples (II)

[EULERIEN]

entrée : G un graphe non dirigé

sortie : oui si G contient un cycle eulérien (un cycle passant par toutes les arêtes une et une seule fois)

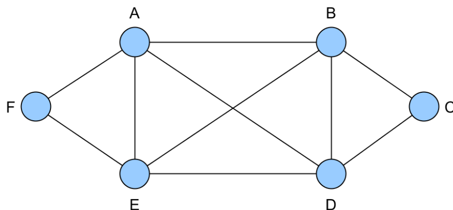


La classe **NP** : exemples (II)

[EULERIEN]

entrée : G un graphe non dirigé

sortie : oui si G contient un cycle eulérien (un cycle passant par toutes les arêtes une et une seule fois)



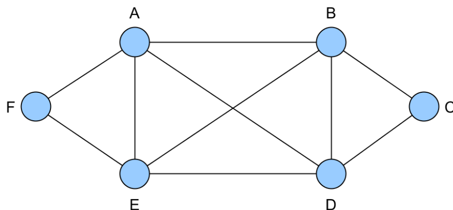
Certificat : A-D-C-B-D-E-A-B-E-F

La classe **NP** : exemples (II)

[EULERIEN]

entrée : G un graphe non dirigé

sortie : oui si G contient un cycle eulérien (un cycle passant par toutes les arêtes une et une seule fois)



Certificat : A-D-C-B-D-E-A-B-E-F

[EULERIEN] est également dans **P**

La classe **NP** : exercice

[3-colorabilité]

entrée : G un graphe non dirigé

sortie : oui si une coloration de G avec 3 couleurs existe

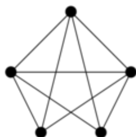
A vous de jouer !

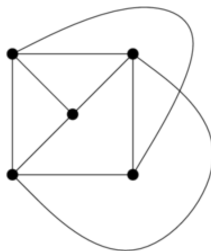
La classe **NP** : exercice (II)

[GI]

entrée : $G = (V, E)$ et $G' = (V', E')$ deux graphes non dirigés

sortie : oui si G et G' sont isomorphes, c'est-à-dire s'il existe une bijection $\sigma\colon V \to V'$ tel que pour tout v de V, si les voisins de v sont $\{v_1, \ldots, v_k\}$ alors les voisins de $\sigma(v)$ sont $\{\sigma(v_1), \ldots, \sigma(v_k)\}$.





A vous de jouer !

La classe **NP** : exemples (III)

Formules logiques : φ

- littéraux : proposition x, négation de proposition $\bar{x}$
- conjonction : $\varphi \wedge \varphi'$ (φ et φ')
- disjonction : $\varphi \vee \varphi'$ (φ ou φ')

$$\overline{mauvais_temps} \vee chapeau \vee parapluie$$

La classe **NP** : exemples (III)

Formules logiques : φ

- littéraux : proposition x, négation de proposition $\bar{x}$
- conjonction : $\varphi \wedge \varphi'$ (φ et φ')
- disjonction : $\varphi \vee \varphi'$ (φ ou φ')

$$\overline{mauvais_temps} \vee chapeau \vee parapluie$$

Valuations : assigne `vrai` ou `faux` aux propositions

Formule satisfiable s'il existe une valuation qui la rend vraie

La classe **NP** : exemples (III)

Formules logiques : φ

- littéraux : proposition x, négation de proposition $\bar{x}$
- conjonction : $\varphi \wedge \varphi'$ (φ et φ')
- disjonction : $\varphi \vee \varphi'$ (φ ou φ')

$$\overline{mauvais_temps} \vee chapeau \vee parapluie$$

Valuations : assigne `vrai` ou `faux` aux propositions

Formule satisfiable s'il existe une valuation qui la rend vraie

$$\{mauvais_temps \mapsto \texttt{true}, chapeau \mapsto \texttt{true}, parapluie \mapsto \texttt{false}\}$$

La classe **NP** : exemples - suite

Une disjonction de littéraux = une clause

$$(\bar{x_1} \vee x_3 \vee x_4)$$

[SAT]

entrée :	Une formule booléenne donnée comme une conjonction de clauses
sortie :	oui si la formule est satisfiable.

$$(\bar{x_1} \vee x_3 \vee x_4) \wedge (x_1 \vee \bar{x_2}) \wedge (\bar{x_3} \vee \bar{x_4}) \wedge (x_2)$$

La classe **NP** : exemples - suite

Une disjonction de littéraux = une clause

$$(\bar{x_1} \vee x_3 \vee x_4)$$

[SAT]

entrée :	Une formule booléenne donnée comme une conjonction de clauses
sortie :	oui si la formule est satisfiable.

$$(\bar{x_1} \vee x_3 \vee x_4) \wedge (x_1 \vee \bar{x_2}) \wedge (\bar{x_3} \vee \bar{x_4}) \wedge (x_2)$$

Certificat : $(x_1 \mapsto \texttt{vrai}, x_2 \mapsto \texttt{vrai}, x_3 \mapsto \texttt{vrai}, x_4 \mapsto \texttt{faux})$

La classe **NP** : Algorithme naïf

Pour résoudre un problème dans **NP**,

- On génère un à un chaque certificat (en temps polynomial)
- On teste si le certificat est témoin d'une solution du problème

La classe **NP** : Algorithme naïf

Pour résoudre un problème dans **NP**,

- On génère un à un chaque certificat (en temps polynomial)
- On teste si le certificat est témoin d'une solution du problème

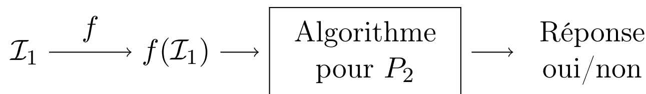
Le certificat possède une taille polynomiale dans la taille du problème. Donc, il existe un nombre exponentiel dans la taille du problème de certificats.

Réduction polynomiale

Définition : Soit P_1, P_2 deux problèmes.

P_1 se réduit polynomialement à P_2 s'il existe une fonction polynomiale f

- transformant une instance $\mathcal{I}_1$ de P_1 en une instance $f(\mathcal{I}_1)$ de P_2 ...
- telle que $\mathcal{I}_1$ est une instance positive de P_1 si et seulement si $f(\mathcal{I}_1)$ est une instance positive de P_2.



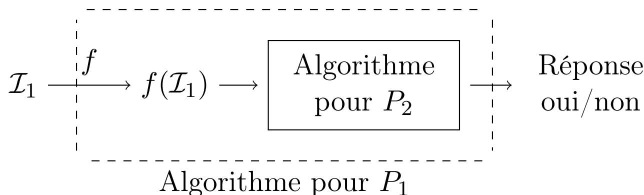
Si P_1 se réduit polynomialement à P_2 alors P_1 est plus "facile" que P_2.

Réduction polynomiale

Définition : Soit P_1, P_2 deux problèmes.

P_1 se réduit polynomialement à P_2 s'il existe une fonction polynomiale f

- transformant une instance $\mathcal{I}_1$ de P_1 en une instance $f(\mathcal{I}_1)$ de P_2 ...
- telle que $\mathcal{I}_1$ est une instance positive de P_1 si et seulement si $f(\mathcal{I}_1)$ est une instance positive de P_2.



Si P_1 se réduit polynomialement à P_2 alors P_1 est plus "facile" que P_2.

Réduction polynomiale : exemple

[3-coloriabilité] se réduit polynomialement à **[3-SAT]**

Construire (en temps polynomial) à partir d'un graphe non orienté G une formule φ définie comme une conjonction de clauses de 3 littéraux tels que G est 3-coloriable si et seulement si φ est satisfiable.

Réduction polynomiale : exemple

[3-coloriabilité] se réduit polynomialement à **[3-SAT]**

Construire (en temps polynomial) à partir d'un graphe non orienté G une formule φ définie comme une conjonction de clauses de 3 littéraux tels que G est 3-coloriable si et seulement si φ est satisfiable.

La formule doit exprimer la propriété d'être une coloration

- Couleurs : $\{1, 2, 3\}$
- Propositions : $\{v_i \mid v \in V,\ i \in \{1, 2, 3\}\}$

- chaque sommet possède une couleur et une seule

$$\bigwedge_{v \in V} \left[(v_1 \vee v_2 \vee v_3) \ \wedge \ \bigwedge_{i \neq j} (\bar{v_i} \vee \bar{v_j}) \right]$$

- Les extrémités des arêtes n'ont pas la même couleur

$$\bigwedge_{(u,v) \in E, i \in \{1,2,3\}} (\bar{u_i} \vee \bar{v_i})$$

Réduction polynomiale : exemple (II)

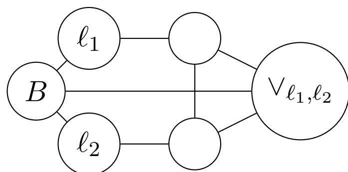
[3-SAT] se réduit polynomialement à **[3-coloriabilité]**

Construire (en temps polynomial) à partir d'une formule φ définie comme une conjonction de clauses de 3 littéraux un graphe non orienté G tels que φ est satisfiable si et seulement si G est 3-coloriable.

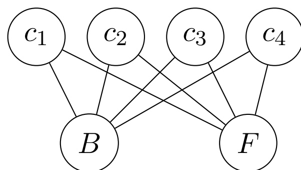
- le graphe contient notamment 3 sommets V, F et B ainsi qu'un sommet pour chaque proposition de la formule et sa négation (v_i et $\bar{v}_i$).
- Les 3 sommets V, F et B sont reliés en triangle : ils devront avoir des couleurs différentes
- la couleur associée à V sera la couleur marquant la valeur booléenne `vrai` tandis que la couleur associée à F sera la couleur marquant la valeur booléenne `faux`.
- Les 3 sommets v_i, $\bar{v}_i$ et B sont reliés en triangle (pour tout i) : ils devront avoir des couleurs différentes

Réduction polynomiale : exemple (II) - suite

Codage (du fragment) de la clause $\ell_1 \vee \ell_2$:



- $\vee_{\ell_1,\ell_2}$ a soit la couleur de V, soit celle de F.
- $\vee_{\ell_1,\ell_2}$ a la même couleur que V si et seulement si ℓ_1 ou ℓ_2 possède cette même couleur.
- Pour $(\ell_1 \vee \ell_2) \vee \ell_3$, on étend le gadget à partir de $\vee_{\ell_1,\ell_2}$ et de ℓ_3.



pour les conjonctions

Réduction polynomiale : exercices

[SAT] se réduit polynomialement à **[3-SAT]**

A vous de jouer !

Réduction polynomiale : exercices (II)

[BinPacking]

entrée :
- n un nombre d'objets
- $p_1, \dots, p_n$ le poids (entier) de chacun des n objets
- c, la capacité d'un sac (entier)
- k le nombre de sacs.

sortie : oui si une mise en sachet existe, une application $m\colon [1\dots n] \to [1\dots k]$ telle que pour tout $j \in [1\dots k]$

$$\sum_{i \text{ t.q. } m(i)=j} p_i \leq c$$

[SubsetSum]

entrée :
- n un nombre d'entiers
- $x_1, \dots, x_n$ des entiers
- s un entier cible.

sortie : oui s'il existe $J \subseteq [1\dots n]$ tel que $\sum_{j\in J} x_j = s$

[SubsetSum] se réduit polynomialement à **[BinPacking]**.

NP-difficulté

Définition : un problème Q est NP-difficile si pour tout problème P dans NP, il existe une réduction polynomiale de P à Q.

Remarque : si Q se réduit polynomialement à R et que Q est NP-difficile, alors R est NP-difficile.

Existe-t-il un problème NP-difficile ?

[SAT] est **NP**-difficile

Théorème de Cook

Soit P un problème dans **NP** et une instance $\mathcal{I}$ de ce problème alors il existe une formule φ définie sous la forme d'une conjonction de clauses constructible en temps polynomial dans la taille de $\mathcal{I}$ telle que

$\mathcal{I}$ est une instance positive de P ssi φ est satisfiable

[SAT] est **NP**-difficile

Théorème de Cook

Soit P un problème dans **NP** et une instance $\mathcal{I}$ de ce problème alors il existe une formule φ définie sous la forme d'une conjonction de clauses constructible en temps polynomial dans la taille de $\mathcal{I}$ telle que

$\mathcal{I}$ est une instance positive de P ssi φ est satisfiable

Idée de la preuve :

- si $P \in$ **NP** alors il existe une machine de Turing et pour chaque instance positive, un certificat (de taille polynomiale) et vérifiable par cette machine en temps polynomial.
- il y a une partie de la formule pour chaque étape du calcul de la machine de Turing (nombre polynomial d'étapes)
- une partie de la formule vérifie la configuration à chaque étape (la configuration est de taille polynomiale)
- une partie de la formule vérifie l'enchainement des configurations

[SAT] est **NP**-difficile

Théorème de Cook

Soit P un problème dans **NP** et une instance $\mathcal{I}$ de ce problème alors il existe une formule φ définie sous la forme d'une conjonction de clauses constructible en temps polynomial dans la taille de $\mathcal{I}$ telle que

$\mathcal{I}$ est une instance positive de P ssi φ est satisfiable

Corollaire

[3-SAT] est **NP**-difficile

NP-complétude

Définition : un problème P est **NP**-complet si

- P est **NP**-difficile
- P est (dans) **NP**

NP-complétude

Définition : un problème P est **NP**-complet si

- P est **NP**-difficile
- P est (dans) **NP**

- **[SAT]** est **NP**-complet
- **[3-colorabilité]** est **NP**-complet
- **[HAMILTONIEN]** est **NP**-complet
- ...

NP-complétude : exercice

Les chevaliers de la table ronde :

On a n chevaliers à installer autour d'une table ronde à n places mais certains chevaliers sont ennemis et ne peuvent donc être assis côte-à-côte. On veut savoir si on peut les installer autour de la table sans que deux ennemis se retrouvent côte-à-côte (l'entrée du problème est les n chevaliers et toutes les paires d'ennemis).

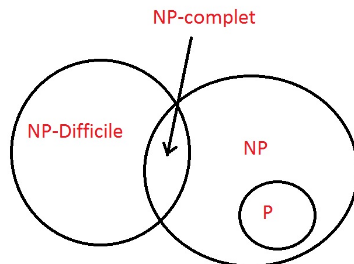
- Montrer que le problème est dans **NP**
- Montrer que le problème est **NP**-difficile (on pourra utiliser le fait que l'existence d'un cycle hamiltonien dans un graphe non orienté est **NP**-difficile).

A vous de jouer ! (On en a gros !)

A one-million dollars problem

Clay Mathematics Institute identifie les 7 problèmes à résoudre durant le 21eme siècle. Parmi eux,

Est-ce que $\mathbf{P} = \mathbf{NP}$ ou $\mathbf{P} \neq \mathbf{NP}$?



L'isomorphisme de graphe [**GI**]

- n'a pas d'algorithme polynomial connu
- n'a pas été prouvé **NP**-complet

László Babai donne en 2015 un algorithme quasi-polynomial en $O(2^{(\log n)^c})$.

Résoudre les problèmes NP-complets ?

- La **NP**-complétude du problème [**SAT**] fait qu'il suffit de résoudre ce problème efficacement, puis d'utiliser une réduction d'un problème **NP**-complet vers lui...

Résoudre les problèmes NP-complets ?

- La **NP**-complétude du problème [**SAT**] fait qu'il suffit de résoudre ce problème efficacement, puis d'utiliser une réduction d'un problème **NP**-complet vers lui...
- Concentration des efforts vers la recherche de bonnes heuristiques pour résoudre le plus rapidement possible le problème [**SAT**]

Résoudre les problèmes NP-complets ?

- La **NP**-complétude du problème [**SAT**] fait qu'il suffit de résoudre ce problème efficacement, puis d'utiliser une réduction d'un problème **NP**-complet vers lui...
- Concentration des efforts vers la recherche de bonnes heuristiques pour résoudre le plus rapidement possible le problème [**SAT**]
- Algorithme de Davis-Putnam-Logemann-Loveland (1960-1962) : propagation unitaire, élimination de littéraux purs, puis heuristique de branchement dans l'exploration des choix possibles des autres variables propositionnelles...

Résoudre les problèmes NP-complets ?

- La **NP**-complétude du problème [**SAT**] fait qu'il suffit de résoudre ce problème efficacement, puis d'utiliser une réduction d'un problème **NP**-complet vers lui...
- Concentration des efforts vers la recherche de bonnes heuristiques pour résoudre le plus rapidement possible le problème [**SAT**]
- Algorithme de Davis-Putnam-Logemann-Loveland (1960-1962) : propagation unitaire, élimination de littéraux purs, puis heuristique de branchement dans l'exploration des choix possibles des autres variables propositionnelles...
- Source de nombreux solveurs [**SAT**] : MiniSAT, Chaff, GRASP, WalkSAT...

Résoudre les problèmes NP-complets ?

- La **NP**-complétude du problème [**SAT**] fait qu'il suffit de résoudre ce problème efficacement, puis d'utiliser une réduction d'un problème **NP**-complet vers lui...
- Concentration des efforts vers la recherche de bonnes heuristiques pour résoudre le plus rapidement possible le problème [**SAT**]
- Algorithme de Davis-Putnam-Logemann-Loveland (1960-1962) : propagation unitaire, élimination de littéraux purs, puis heuristique de branchement dans l'exploration des choix possibles des autres variables propositionnelles...
- Source de nombreux solveurs [**SAT**] : MiniSAT, Chaff, GRASP, WalkSAT...
- Compétition depuis 2002 qui a permis d'énormes améliorations dans l'efficacité de ces outils...

Résoudre les problèmes NP-complets ?

- La **NP**-complétude du problème [**SAT**] fait qu'il suffit de résoudre ce problème efficacement, puis d'utiliser une réduction d'un problème **NP**-complet vers lui...
- Concentration des efforts vers la recherche de bonnes heuristiques pour résoudre le plus rapidement possible le problème [**SAT**]
- Algorithme de Davis-Putnam-Logemann-Loveland (1960-1962) : propagation unitaire, élimination de littéraux purs, puis heuristique de branchement dans l'exploration des choix possibles des autres variables propositionnelles...
- Source de nombreux solveurs [**SAT**] : MiniSAT, Chaff, GRASP, WalkSAT...
- Compétition depuis 2002 qui a permis d'énormes améliorations dans l'efficacité de ces outils...
- Mais il y a toujours des instances de [**SAT**] qui posent problème aux solveurs !

Au-delà de **NP**

PSPACE est la classe des problèmes polynomiaux en espace

Définition : $P \in$ **PSPACE** si P est solvable sur une machine de Turing utilisant un espace additionnel de la bande polynomial dans la taille de l'entrée.

Exemple :

[QBF]

entrée : Une formule donnée sous la forme d'une conjonction de clauses où chaque variable est quantifiée

sortie : oui si la formule est satisfiable

$$\exists x_1 \exists x_2 \forall x_3 \exists x_4 \ (\bar{x_1} \vee x_3 \vee x_4) \wedge (x_1 \vee \bar{x_2}) \wedge (\bar{x_3} \vee \bar{x_4}) \wedge (x_2)$$

La classe **PSPACE**

[k-INTERSECTION AFD]

entrée : k automates finis (déterministes)

sortie : oui s'il existe un mot reconnu par les k automates.

La classe **PSPACE**

[k-INTERSECTION AFD]

entrée : k automates finis (déterministes)

sortie : oui s'il existe un mot reconnu par les k automates.

- **P** $\subseteq$ **PSPACE** : en temps polynomial, on ne peut pas parcourir plus qu'un nombre polynomial de cases du ruban

La classe **PSPACE**

[k-INTERSECTION AFD]

entrée : k automates finis (déterministes)

sortie : oui s'il existe un mot reconnu par les k automates.

- **P** $\subseteq$ **PSPACE** : en temps polynomial, on ne peut pas parcourir plus qu'un nombre polynomial de cases du ruban
- **NP** $\subseteq$ **PSPACE** : on génère un par un tous les certificats possibles ; pour chacun d'eux, on le vérifie en temps polynomial.

La classe **PSPACE**

[k-INTERSECTION AFD]

entrée : k automates finis (déterministes)

sortie : oui s'il existe un mot reconnu par les k automates.

- **P** $\subseteq$ **PSPACE** : en temps polynomial, on ne peut pas parcourir plus qu'un nombre polynomial de cases du ruban
- **NP** $\subseteq$ **PSPACE** : on génère un par un tous les certificats possibles ; pour chacun d'eux, on le vérifie en temps polynomial.
- On ne sait pas si **P** = **PSPACE** ou **P** $\neq$ **PSPACE**

Au-delà de **NP** (II)

EXP est la classe des problèmes exponentiels (en temps)

Définition : $P \in \mathbf{EXP}$ si P est solvable sur une machine de Turing en temps exponentiel dans la taille de l'entrée.

Au-delà de **NP** (II)

EXP est la classe des problèmes exponentiels (en temps)

Définition : $P \in \mathbf{EXP}$ si P est solvable sur une machine de Turing en temps exponentiel dans la taille de l'entrée.

- **PSPACE** $\subseteq$ **EXP** : en temps exponentiel, on peut parcourir toutes les configurations d'une machine polynomialement bornée en espace

Au-delà de **NP** (II)

EXP est la classe des problèmes exponentiels (en temps)

Définition : $P \in \mathbf{EXP}$ si P est solvable sur une machine de Turing en temps exponentiel dans la taille de l'entrée.

- **PSPACE** $\subseteq$ **EXP** : en temps exponentiel, on peut parcourir toutes les configurations d'une machine polynomialement bornée en espace
- On ne sait pas non plus si **EXP** $=$ **PSPACE** ou **EXP** $\neq$ **PSPACE**...