La classe NP

ACT

Sophie Tison
Université Lille
Master1 Informatique

Menu du jour

- Rappels
- Paradigmes d'algorithme
- Complexité des problèmes
- Algorithmique Avancée

Rappels du dernier cours

- La complexité d'un problème est la complexité minimale dans le pire des cas d'un algorithme qui le résout.
- On définit des classes de problèmes selon l'ordre de grandeur de leur complexité, en essayant d'avoir des classes indépendantes du modèle de calcul. Pour simplifier, on se limite aux problèmes de décision ou propriétés.

Rappels du dernier cours (suite et fin)

- À une propriété abstraite définie sur un ensemble quelconque, on peut associer une propriété concrète définie sur des mots -les représentations des instances-. Décider une propriété se ramène donc à tester l'appartenance d'un mot à un langage, celui des représentations des instances positives de la propriété (les données qui vérifient la propriété).
- La classe P (ou PTIME) est la classe des problèmes P est la classe des problèmes de décision dits praticables, i.e. pour lesquels il existe un algorithme de résolution polynomial en temps.

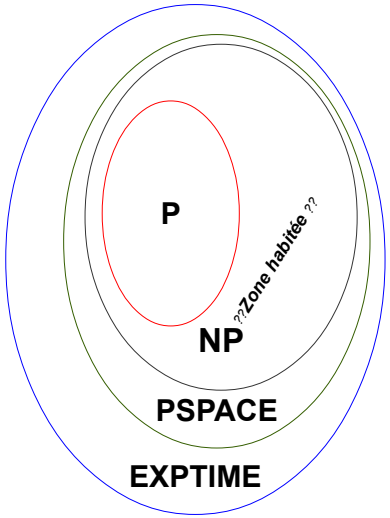
Quelques autres exemples de classes

- PSPACE: la classe des problèmes de décision pour lesquels il existe un algorithme de résolution polynomial en espace.

  PTIME est inclus dans PSPACE: un algorithme polynomial en temps "consomme au plus un espace polynomial".
- EXPTIME: la classe des problèmes de décision pour lesquels il existe un algorithme de résolution exponentiel en temps.
  PSPACE est inclus dans EXPTIME. Pourquoi?

La classe NP

1. Elle contient beaucoup de propriétés associées à des problèmes courants: problèmes de tournées, d'emploi du temps, placement de tâches, affectation de fréquences, rotation d'équipages, découpage...
2. La classe NP contient P.
3. Pour beaucoup de propriétés NP, on n'a pas trouvé d'algorithme polynomial, mais pour aucune d'entre elles, on n'a prouvé qu'elle ne pouvait pas être décidée en temps polynomial.
4. On suppose que P $\neq$ NP mais personne jusque maintenant n'a su prouver -ni infirmer- cette conjecture émise en 1971 et déclarée un des problèmes du millénaire par l'Institut Clay qui propose 1 million de dollars pour sa résolution.



Exemple 1: Le 3-coloriage de graphes

Donnée: G = (S, A) un graphe

Sortie: oui, si le graphe peut être coloré en 3 couleurs i.e. on peut trouver:

une application col de S dans $\{1, 2, 3\}$ telle que

$col(s) \neq col(s')$ si s, s' sont reliés par un arc ($(s, s') \in A$ ou $(s', s) \in A$)

Exemple 2: Le circuit Hamiltonien

Donnée: $G = (S, A)$ un graphe, n un entier

Sortie: oui, si le graphe contient un circuit hamiltonien i.e. on peut trouver:

une application sommet de $\{1, 2, ..., n\}$ dans S telle que:

elle soit injective (on ne passe pas deux fois par le même sommet)

$(\text{sommet}(i), \text{sommet}(i+1)) \in A$ pour $1 <= i < n$

$(\text{sommet}(n), \text{sommet}(1)) \in A$

Exemple 3: Atteindre la cible

Donnée: $x_1, ..., x_n$, n entiers
c un entier (cible)

Sortie: oui, si on peut obtenir c comme somme d'un sous-ensemble des x_i

i.e. on peut trouver

$J \subset \{1, ..., n\}$ tel que $c = \sum_{i \in J} x_i$

Exemple 4: SAT, la satisfiabilité d'une expression booléenne

Donnée: n, un entier, et Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = \text{Vrai}$

Exemples:
si $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$

Φ est satisfiable: $v(x_1) = \text{Vrai}, v(x_3) = \text{Faux}, ...$

si $\Phi = (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (\neg x_1 \vee x_3)$

Φ n'est pas satisfiable (Comment le prouver?)

Quel est le lien?

Pour chacune des propriétés précédentes, on cherche si il existe une solution qui vérifie une certaine contrainte.

.les "candidats-solutions" sont des objets pas trop "gros": un coloriage, une valuation..

.vérifier si une solution vérifie la contrainte est "facile"

C'est la spécificité des NP ...

Remarque: cela ne nous donne pas pour autant d'algo efficace. Enumérer les candidats solutions nous amène en général à une solution exponentielle!

Exemple 1: Le 3-coloriage de graphes

Donnée: $G = (S, A)$ un graphe

Sortie: oui, si le graphe peut être coloré en 3 couleurs i.e. on peut trouver:

une application col de S dans $\{1, 2, 3\}$ telle que

$col(s) \neq col(s')$ si s, s' sont reliés par un arc ($(s, s') \in A$ ou $(s', s) \in A$)

Exemple 2: Le circuit Hamiltonien

Donnée: $G = (S, A)$ un graphe, n un entier

Sortie: oui, si le graphe contient un circuit hamiltonien i.e. on peut trouver:

une application sommet de $\{1, 2, ..., n\}$ dans S telle que:

elle soit injective (on ne passe pas deux fois par le même sommet)

$(sommet(i), sommet(i + 1)) \in A$ pour $1 <= i < n$

$(sommet(n), sommet(1)) \in A$

Exemple 3: Atteindre la cible

Donnée: $x_1, ..., x_n$, n entiers
c un entier (cible)

Sortie: oui, si on peut obtenir c comme somme d'un sous-ensemble des x_i

i.e. on peut trouver

$J \subset \{1, ..., n\}$ tel que $c = \sum_{i \in J} x_i$

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: n, un entier, et Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Exemples:
si $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$

Φ est satisfiable: $v(x_1) = Vrai, v(x_3) = Faux, ...$

si $\Phi = (x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (\neg x_1 \vee x_3)$

Φ n'est pas satisfiable (Comment le prouver?)

Enumérer?

Par exemple, il y a plus de 3-coloriages d'un graphe de 200 sommets que d'atomes dans l'univers.

Notre Univers (sa partie visible) contiendrait 10^{80} atomes.
Le nombre de 3-coloriages d'un graphe de 200 sommets est 3^{200}.
$3^{200} = 3^{5*40} = 243^{40}, 10^{80} = 10^{2*40} = 100^{40}$
$3^{200} > 10^{80}$

La définition de NP via les certificats

NP

L est dit NP si il existe un polynôme Q, et un algorithme polynomial à deux entrées et à valeurs booléennes tels que:

$$L = \{u / \exists c, A(c, u) = \text{Vrai}, |c| \leq Q(|u|)\}$$

algorithme de vérification
certificat

L'intuition?

On peut par exemple voir c comme une preuve, A comme un algorithme qui vérifie la preuve; vous pouvez être capable de vérifier facilement la preuve courte qu'un gentil génie, (un prof, par exemple:-)) vous donne mais cela n'implique pas forcément qu'elle soit facile à trouver...

Une propriété NP sera donc une propriété pour laquelle les instances positives ont une preuve "courte" et "facile" à vérifier.

Comment montrer qu'une propriété est NP?

Pour montrer qu'une propriété est NP, il faut:

- définir la notion de certificat et montrer que la taille d'un certificat est bornée polynomialement par la taille du problème
- définir l'algo de Vérification et montrer qu'il est polynomial (et correct...)

Etre 3-coloriable: les certificats

Un certificat pour G est juste un coloriage des noeuds.
On peut par exemple le représenter par un tableau de couleurs indexé par les sommets.
On a donc :

taille du certificat <= taille du graphe

(on suppose que la taille d'un graphe est au moins le nombre de sommets plus le nombre d'arcs)
La taille d'un certificat est alors bien linéaire, donc polynomialement bornée, par rapport à celle de la donnée.

Etre 3-coloriable: l'algo de vérification

Un certificat est valide Ssi aucun arc ne relie deux noeuds de même couleur:

```
boolean A(col, G){
 Pour chaque arc (s,d) de G
       si col(s)=col(d) retourner Faux;
 retourner Vrai;
}
```

Etre 3-coloriable: l'algo de vérification

Un certificat est valide Ssi aucun arc ne relie deux noeuds de même couleur:

```
boolean A(col, G){
 Pour chaque arc (s,d) de G
       si col(s)=col(d) retourner Faux;
 retourner Vrai;
}
```

La complexité de l'algorithme est de l'ordre de card(A) donc bien polynomiale.

'Etre 3-coloriable' est donc bien une propriété NP.

Le circuit Hamiltonien: les certificats

Donnée: G = (S, A) un graphe, n un entier, n <= card(S)

Certificat: une suite de n sommets, soit par exemple un tableau de n sommets.
Donc la taille d'un certificat est au plus n (ou $n * \log(\text{card}(S))$, si on prend en compte la taille du codage d'un sommet)

Vérification: if faut vérifier que c'est bien un circuit : tous les sommets sont différents, un sommet et son suivant sont bien reliés par un arc.

Le circuit Hamiltonien: la vérification

```
//cert: tableau de n sommets
A(cert,G){
boolean dejapasse=new passe[nbsommets];
//pour vérifier on ne passe pas 2 fois par le meme
pour i de 1 à n
  si dejapasse[cert[i]] retourner Faux;
   //on passe deux fois par  ce sommet
  si (cert[i],cert[i+1]) n'est pas un arc de G
   retourner Faux;
  dejapasse[cert(i)]=true;
fin pour;
  si (dejapasse[cert[n]]
   ou (cert[n],cert[1]) n'est pas un arc)
  alors retourner Faux;
  sinon retourner Vrai;
```

Le circuit Hamiltonien

La complexité de l'algorithme est de l'ordre de n donc bien polynomiale.

Donc, la propriété de l'existence d'un circuit hamiltonien est bien NP.

Combien de certificats possibles? n^n (ou n! si on impose que ce soit une permutation dans le type du certificat).

Exemple 3: Atteindre la cible

Donnée: $x_1, ..., x_n$, n entiers
c un entier (cible)

Sortie: oui, si on peut obtenir c comme somme d'un sous-ensemble des x_i i.e. on peut trouver $J \subset \{1, ..., n\}$ tel que $c = \sum_{i \in J} x_i$
Certificat: $J \subset \{1, ..., n\}$
On peut le représenter par un tableau de n booléens: la taille d'un certificat est inférieure à la taille du problème.

À vérifier: $c = \sum_{i \in J} x_i$:

Combien de certificats possibles? 2^n

Exemple 3: Atteindre la cible

À vérifier: $c = \sum_{i \in J} x_i$:

```
boolean A(cert, x1,...,xn,s){
   int s==0;
   Pour i de 1 à n
        si  cert(i) alors s=s+x_i
   fin pour;
   retourner (s==c);
 }
```

Exemple 3: Atteindre la cible

```
boolean A(cert, x1,...,xn,s){
   int s==0;
   Pour i de 1 à n
        si  cert(i) alors s=s+x_i
   fin pour;
   retourner (s==c);
 }
```

Algo en O(n)

La propriété est bien NP!

Exemple 4: SAT: satisfiabilité d'une expression Booléenne

Donnée: Φ une expression booléenne avec n variables booléennes, $x_1, ..., x_n$

Sortie: oui, si Φ est satisfiable, i.e. il existe une valuation v telle que $v(\Phi) = Vrai$

Certificat: v, par exemple représentée par un tableau de n booléens, donc de taille polynomiale.

Algo de vérification: évaluer $v(\Phi)$, ce qui est bien polynomial.

Sat est bien une propriété NP.

Combien de valuations possibles? 2^n

La définition via le non-déterminisme

NP=Non-Déterministe Polynomial

Définition Alternative: une propriété Pr est NP si il existe un algorithme non déterministe polynomial qui décide Pr.

Remarque: NP=Non-Déterministe Polynomial et non pas Non Polynomial!

Algorithmes non déterministes

Un algorithme non-déterministe peut être vu comme un algorithme avec des instructions de type "choix(i,1..n)": on choisit aléatoirement un entier dans l'intervalle [1..n]. On peut se restreindre à n = 2.

On peut prendre comme modèle de calcul non-déterministe, les Machines de Turing non déterministes

Algos non déterministes polynomiaux

Complexité d'un algorithme non-déterministe: Un algorithme non-déterministe A est dit polynomial si il existe un polynôme Q tel que pour toute entrée u, tous les calculs de A sur u ont une longueur d'exécution bornée par Q(|u|).

Algos non déterministes à valeurs booléennes

Soit A un algorithme non déterministe à valeurs booléennes et dont tous les calculs s'arrêtent;
il décide la propriété Pr suivante: "u vérifie Pr Ssi il existe un calcul de A sur u qui retourne Vrai."
Remarque: penser à un automate non déterministe: un mot est accepté si et seulement si il existe au moins un chemin acceptant.

Algos non déterministes à valeurs booléennes: exemple

```
Cherchercible(x1,....xn, c){
s=0;
Pour  i in 1..n
 Choisir(onleprend?,1..2);
  Si (onleprend?==1) s=s+xi;
 finPour;
 retourner (s==c);}
```

L'algorithme est bien un algorithme non-déterministe polynomial pour "Atteindre la cible".

La définition via le non-déterminisme versus La définition via les certificats

Les deux définitions sont équivalentes:

- Un certificat correspond à une suite de choix dans l'exécution de l'algorithme non déterministe. À partir d'un algorithme non-déterministe polynomial pour vérifier P, on peut définir la notion de certificat qui correspond à une suite de choix. L'algorithme de vérification consiste à vérifier que l'exécution de l'algorithme non déterministe correspondant à la suite de choix donnée par le certificat retourne Vrai.
- À partir d'une notion de certificat et d'algorithme de vérification, on construit un algorithme non-déterministe qui consiste à d'abord générer aléatoirement un certificat -la partie non déterministe- et ensuite le vérifier en utilisant l'algorithme -déterministe- de vérification.

NP et les autres

- NP par rapport à P?

  Bien sûr, P est inclus dans NP: toute propriété P est une propriété NP; l'algorithme de vérification est l'algorithme de décision et n'a pas besoin de certificat: on peut prendre pour certificat le mot vide.

- On peut aussi montrer que NP est inclus dans EXPTIME et même dans PSPACE: pensez à l'algorithme qui énumère et teste tous les certificats possibles.

La conjecture NP $\neq$ P?

On conjecture -en général- que P $\neq$ NP, mais personne n'a su le prouver!

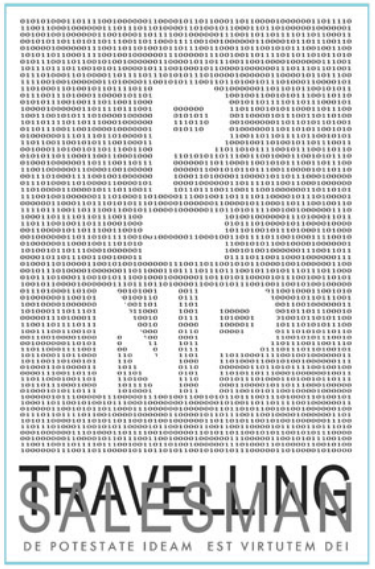
- Aucune propriété NP n'a été prouvée à ce jour ne pas être P.
- D'un autre côté, pour beaucoup de propriétés NP, aucun algorithme polynomial n'a été trouvé (ou tout du moins prouvé exister) malgré les efforts de milliers de personnes!
- On verra qu'il existe des propriétés NP telles que si on trouvait un algorithme polynomial pour l'une d'entre elles, il y aurait un algorithme polynomial pour n'importe quelle propriété NP.

La conjecture NP $\neq$ P?

La conjecture a été émise par Stephen Cook en 1971 et le "Clay Mathematics Institute" offre 1 million de dollars à celui qui trouve (et prouve) la réponse à la question!

CLAY
MATHEMATICS
INSTITUTE

La conjecture NP $\neq$ P? au cinéma



NP et la négation...

La classe NP est close par union, concaténation, étoile, intersection (voir TD).

Par contre, on ne sait pas si NP est close par complémentaire:

Disposer de la notion de certificat et d'algorithme de vérification pour une propriété Q, n'implique à priori pas que non Q soit NP;
Exemple: comment vérifier qu'il n'existe pas de 3−coloriage?

NP et co-NP

Une propriété Q telle que non Q soit NP est dite co-NP.

- Bien sûr, co-NP contient P.
- Bien sûr, co-NP est contenue dans ExpTime (même PSpace).
- On conjecture aussi que NP $\neq$ co $-$ NP mais, là encore ce n'est qu'une conjecture!
- Si NP = P, on aurait NP=co-NP.
- On pourrait avoir NP = co $-$ NP sans que NP soit égal à P!

Le bilan

- Une propriété NP est une propriété pour la quelle trouver une "solution" (une preuve ..) est peut-être difficile, mais vérifier une solution (une preuve...) est facile.
- P: easy to find
- NP: easy to check
- Montrer qu'une propriété est NP n'est en général qu'une étape... On cherche en général ensuite à montrer qu'elle est aussi NP-dure: voir le prochain cours.
- Attention: Montrer qu'une propriété est NP n'est pas montrer qu'elle n'est pas P!!!

La présentation par l'institut Clay

Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair from taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe!

La présentation par l'institut Clay

Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971.

La semaine prochaine

- DS sur la première partie du cours (jusqu'aux algorithmes gloutons(inclus))
- DS ici
- Documents autorisés
- DS d'1h : arrivez un peu avant 9h.
- Apportez du brouillon et de quoi écrire!
- Si vous avez un aménagement d'examen, prévenez-moi rapidement.