

Notes de cours:
Notions de complexité algorithmique
Master 2 Informatique Recherche Montpellier
II

Michel Habib,
email: habib@liafa.jussieu.fr

26 octobre 2006

Table des matières

1	Pour l'algorithmique	5
2	Principales classes de complexité	9
2.1	Revenir aux fondamentaux, i.e. les codages binaires pour l'analyse des algorithmes	9
2.1.1	Le problème du tri	9
2.1.2	Le calcul de $n!$	9
2.2	La classe P	10
2.3	La classe NP	10
2.4	La fameuse conjecture	11
2.4.1	Un argument diagonal?	11
2.5	$NP \cap co - NP$	12
2.6	Problèmes difficiles à classer dans NP	14
2.7	Retour sur la fameuse conjecture	15
3	Quelques exemples de réductions polynomiales	17
3.1	Réductions par restriction	17
3.2	Un problème NP-complet typique	20
3.3	SAT et dépendances	20
3.3.1	Application	24
3.3.2	Exercices	24
3.3.3	D'autres cas polynomiaux de SAT	25
3.4	Graphes d'incidence variables-clauses	26
3.4.1	Exercices	29
3.5	Extensions de la théorie	29
4	Recueil d'examens du module complexité	31
4.0.1	Deuxième partie : 15 points	33

4.1	S'il pleut dimanche ... ***	38
4.1.1	Complexité "concrète"	38
4.1.2	SAT Toujours	39
4.1.3	Autour de SAT	39
4.1.4	Un problème de chemins	40
4.1.5	Approximation	40
4.2	Pour le week-end	41
4.2.1	On s'échauffe ...	41
4.2.2	Assez rigolé, au travail maintenant ...	41
4.2.3	Une question d'actualité *	43
4.2.4	Un schéma d'approximation pour le problème du sac à dos	43
4.3	Novembre 2001	45
4.3.1	Complexité concrète	45
4.3.2	Une nouvelle machine de Turing	45
4.3.3	Réductions	45
4.3.4	Encore SAT	46
4.3.5	Des problèmes de rectangles *	46
4.3.6	Comme promis ... ***	46
4.3.7	Commentaire final	47
4.4	Novembre 2002	49
4.4.1	Complexité concrète	49
4.4.2	Quelques Réductions	49
4.4.3	Encore des réductions	50
4.4.4	Rituel ... **	50
4.4.5	Approximation : un problème de rangement	51
4.4.6	Commentaire final	51
4.5	Novembre 2003	52
4.5.1	Compréhension générale des théories	52
4.5.2	Réductions	52
4.6	Novembre 2004	53
4.6.1	Applications des théories de complexité	53
4.6.2	Recouvrement versus partition	54
4.6.3	Renommage	54

Chapitre 1

Pour l'algorithmique

L'algorithmique est la science des calculs et l'un des premiers algorithmes connus est celui du calcul du PGCD de deux nombres proposé par Euclide (300 ans avant J.C.). Ces calculs doivent être bien sûr prouvés, mais aussi analysés (i.e. étudier le comportement de l'algorithme sur les données). On dit alors "étudier la complexité de l'algorithme". Pour mémoire la complexité en moyenne de l'algorithme d'Euclide n'a été établie que dans les années 1970 et cet algorithme reste au centre de développements récents (le codage cryptographique RSA de Rivest, Shamir et Adleman 1977, les calculs de droites discrètes, ...).

La question de l'existence d'un calcul ou algorithme "efficace" pour résoudre un problème donné est une question essentielle et structurante de l'informatique théorique depuis Gödel et Turing (années 1930). Elle est toujours d'actualité pour s'en convaincre il suffit de considérer l'importance économique et sociale de la cryptographie, ou encore le problème de la gestion des grandes masses de données distribuées.

Afin de mieux comprendre pourquoi certains problèmes possèdent un algorithme de résolution polynomial, alors que pour d'autres problèmes personne n'en connaît, la théorie de la complexité a introduit P (resp. NP) la classe des problèmes possédant un algorithme déterministe (resp. non-déterministe) de résolution dont le temps de calcul est borné par un polynôme en la taille de la donnée. Ces notions ont permis de nombreuses avancées aux retombées importantes. Mais malgré tous les efforts, la fameuse conjecture ($P \neq NP$ posée par Edmonds en 1961) est toujours ouverte et figure parmi les 6 questions les plus importantes en mathématiques de notre siècle selon le Clay Mathematics Institute. Cette question est très importante car la classe

de problèmes NP comprend de nombreux problèmes pratiques pour lesquels nous aimerions connaître des algorithmes polynomiaux. De même cette question est très liée à l'existence d'une bonne fonction de codage, existence sur laquelle repose la cryptographie asymétrique actuelle. En outre, il n'est pas certain qu'une réponse négative à cette conjecture (donc si $P=NP$) fournisse des algorithmes efficaces (au sens naïf du terme, i.e. un algorithme capable de résoudre en un temps raisonnable sur nos machines actuelles des problèmes de taille réelle) pour tous les problèmes de P. En effet les résultats récents tels que la résolution polynomiale du test de primalité Agarwal, Saxena et Kayal en 2002, ou encore les algorithmes polynomiaux inspirés de la théorie des mineurs de Seymour et Roberston (Pour certains problèmes il est possibles de montrer qu'il n'existe qu'un nombre fini de configurations exclues et l'on peut donc construire des algorithmes basés sur le test polynomial d'un nombre fini - mais peut-être extraordinairement grand- de configurations exclues) élargissent notre vision des algorithmes polynomiaux.

Dans le cadre de la résolution des problèmes difficiles (appelés NP-difficiles i.e. ayant été classés comme au moins aussi difficiles que le plus difficile problème de NP) et pour lesquels nous ne connaissons pas d'algorithmes polynomiaux de résolution, la question de l'existence d'algorithmes polynomiaux d'approximation se pose naturellement. Ces algorithmes fournissent une solution dont la valeur est bornée par la valeur d'une solution optimale à un facteur multiplicatif constant près.

En matière d'approximation deux résultats récents sont à mentionner, car ils sont les conséquences de développements théoriques importants. Le premier résultat est négatif, il s'agit d'un résultat de non-approximabilité (Arora, Lund, Motwani, Sudan et Szegedy 92) valable pour toute une classe de problèmes NP-difficiles. Ce résultat très profond a été obtenu à l'aide d'une caractérisation de la classe NP en termes d'algorithmes probabilistes (PCP théorème). Le deuxième résultat est positif, il s'agit d'un algorithme polynomial d'approximation pour le problème de la coupe maximale dû à Goemans et Williamson en 1994, introduisant une généralisation de la programmation linéaire.

Même si l'on perçoit des frémissements actuels sur les théories de la complexité et que des résultats très importants ont été obtenus ces dernières années, il n'en demeure pas moins qu'il nous faut quotidiennement proposer des algorithmes ou heuristiques (calculs dont on ne peut pas garantir le résultat) sur des problèmes "complexes". Ainsi par exemple les moteurs de recherche actuels utilisent un grand savoir faire algorithmique (sur l'algorithmique du

texte, des bases de données, mais aussi en parallélisation et distribution des calculs) et les enjeux économiques dans ces domaines ne font que croître.

Hormis les points classiques de l'algorithmique sur lesquels les recherches sont toujours actives et méritent d'être soutenus (tels celui de la conception et analyse d'algorithmes et de structures de données, les recherches sur les fondements du calcul et de la programmation ou encore l'étude systématique de l'approximabilité des problèmes), parmi les domaines qui méritent un effort particulier de développement nous mentionnerons :

- L'algorithmique distribuée imposée par l'émergence des réseaux hétérogènes et de la mobilité (exemples les routages dans les réseaux ad-hoc, la gestion d'un ensemble de robots autonomes, ...).
- La gestion et l'utilisation des grandes masses de données et des grands réseaux. Notion de données dynamiques, graphe du WEB,
- La gestion optimale du trafic dans un réseau à commutation de paquets (modèles discrets/probabilistes/continus ou encore ceux issus de la théorie des jeux).
- Les méthodes et algorithmes probabilistes qui ont permis de nombreux développements ces dernières années sont très prometteuses.
- Les nouveaux modèles de calcul issus des nouvelles technologies ou ceux obtenus dans une interaction avec d'autres disciplines (l'un des modèles les plus intéressants est celui du calcul quantique, mais d'autres modèles sont à étudier tels ceux issus de la biologie ou de la chimie).

En conclusion, la discipline algorithmique entretient des liens historiques et profonds avec les mathématiques, de nombreuses questions théoriques sont économiquement importantes. Par exemple, est-il possible de construire un algorithme efficace pour la décomposition d'un nombre entier composite en ses facteurs premiers ou pour le calcul du logarithme discret ? De tels algorithmes remettraient en cause la plupart des algorithmes de cryptographie actuels (RSA, El Gamal, courbes elliptiques). De même des liens forts existent avec la physique et ils seraient peut-être à soutenir. Ainsi par exemple dans le cadre de la résolution des problèmes difficiles les études sur les effets de seuil ou les métaheuristiques de type recuit simulé sont directement inspirées de la physique statistique. Mais lorsque les données sont de très grande taille, les approches classiques de l'algorithmique ne suffisent plus, il serait judicieux de s'inspirer du savoir faire expérimental de la physique (systèmes dynamiques complexes, petits mondes ...).

Nous avons vu ci-dessus au travers des exemples mentionnés, que l'algorithmique est au centre de l'informatique et des systèmes, elle évolue rapi-

dement face à la demande des applications, ses concepts et outils de base peuvent encore évoluer, ce qui rend cette discipline passionnante.

Chapitre 2

Principales classes de complexité

2.1 Revenir aux fondamentaux, i.e. les codages binaires pour l'analyse des algorithmes

Afin d'éviter les ambiguïtés, il est sage de revenir aux définitions précises de taille de la donnée et de son codage binaire et de considérer comme opération élémentaire une opération sur un bit (**bit-opération**).

2.1.1 Le problème du tri

Nom (*Tri de nombres entiers*)

Données: a_1, \dots, a_n , n entiers positifs

Résultat: σ une permutation de $[1, n]$ vérifiant : $a_{\sigma(1)} \leq a_{\sigma(2)} \dots a_{\sigma(n)}$

Il existe des algorithmes de tri (par exemple tri par fusion), utilisant $O(n \log n)$ comparaisons.

La taille de la donnée est ici $t = \sum_{i=1}^n \lceil \log_2(a_i + 1) \rceil$ bits.

On remarque que $t \geq n$ et qu'une comparaison nécessite au plus

$\text{Maximum}_{1 \leq i \leq n} \{ \lceil \log_2(a_i + 1) \rceil \} \in O(t)$ bit-opérations.

Donc l'algorithme de tri par fusion est en $O(t^2 \log t)$ et donc polynomial en t . \square

2.1.2 Le calcul de $n!$

Prenons l'algorithme classique basé sur n multiplications successives.

La complexité s'obtient par la formule :

$$T(n) = \sum_{i=1}^{i=n} \text{mult}(i, \text{Result}_i)$$

En fait Result_i est exactement de taille $i!$

$$\text{D'où } T(n) = \sum_{i=1}^{i=n} \log(i) \cdot \log(i!)$$

$$T(n) \leq \sum_{i=1}^{i=n} i \cdot \log^2(i).$$

2.2 La classe P

La classe P (resp. FP) est la classe des problèmes de décision (resp. de recherche) qui admettent un algorithme de résolution sur une machine de Turing déterministe dont le nombre de transitions (temps de calcul) est borné par un polynôme en la taille de la donnée.

La classe des problèmes polynomiaux contient de nombreux problèmes tels le problème du calcul de la fermeture transitive d'un graphe orienté, le calcul du diamètre d'un graphe ou encore le calcul des plus courts chemins dans un graphe quand ils sont définis.

Une fois que l'on montré qu'un problème était dans P en exhibant un algorithme polynomial, la question naturelle suivante est :

Quel est le meilleur algorithme pour ce résoudre ce problème ?

Cette question n'a pas nécessairement un sens très général, car il faut bien préciser les données et leur codage ainsi que le modèle de machine choisi sous peine d'imprécisions, lorsqu'on veut établir des bornes inférieures de complexité.

Pour clore dans cet esprit, voici deux exercices :

- Ecrire un algorithme linéaire qui calcule le diamètre d'un arbre.
- Ecrire un algorithme linéaire qui teste l'isomorphisme de deux arbres (non orientés).

2.3 La classe NP

La classe NP (resp. FNP) est la classe des problèmes de décision (resp. de recherche) qui admettent un algorithme de résolution sur une machine de Turing non-déterministe dont le nombre de transitions (temps de calcul) est borné par un polynôme en la taille de la donnée.

En utilisant la notion de certificat polynomial on peut dire que :

si P est la classe des problèmes de décision calculables polynomialement, NP est donc celle des problèmes de décision vérifiables polynomialement.

Une classe de complexité se structure à l'aide de relations de préordre (appelées réductions entre problèmes). Ces relations de préordres sont simplement transitives et réflexives. La plus connue d'entre elles, la réduction polynomiale ou Karp-réduction est notée $L_1 \ll_K L_2$ et signifie que le problème L_1 est plus facile que le problème L_2 ou encore que le problème L_2 est au moins aussi difficile que le problème L_1 . Cette réduction n'est définie qu'entre problèmes de décision.

On écrit que $L_1 \ll_K L_2$, s'il existe une transformation polynomiale A qui transforme toute instance I de L_1 en une instance $A(I)$ de L_2 vérifiant la condition suivante :

La réponse est OUI pour I pour L_1 ssi la réponse est OUI pour $A(I)$ pour L_2 .

On la note parfois : \ll_m^p en sous-entendant "polynomial many-to-one reduction", c'est à dire, en faisant référence à la réduction "many-to-one" utilisée dans le cadre de l'étude des fonctions récursives [16].

Une deuxième réduction, celle de Turing :

$$Pb1 \ll_T Pb2 \text{ ssi } PB1 \in P^{Pb2}$$

En clair cela veut dire qu'il existe un algorithme polynomial pour $Pb1$ qui utilise un nombre polynomial d'appels d'un algorithme de résolution de $Pb2$. Parfois on dit que l'on dispose d'un oracle de résolution pour $Pb2$.

En particulier cette relation \ll_T permet de comparer entre eux des problèmes qui ne sont pas dans NP .

2.4 La fameuse conjecture

Depuis 1972, on se pose la question :

$$P \neq NP?$$

La réponse vaut 1 Million de dollars et cette question figure parmi les 6 problèmes les plus importants de mathématiques du XXI ème siècle (Cf. Clay Mathematics Institute).

La question :

$$NP \neq co - NP?$$

n'est pas moins intéressante.

2.4.1 Un argument diagonal ?

En constatant les faits suivants :

- La question sur le $|N| \neq |R|$ a été résolue par Cantor en utilisant un argument diagonal.
- De même la question de l'existence d'une fonction non calculable se résout en utilisant un argument diagonal (par exemple pour sur le problème de l'arrêt d'une machine de Turing).
- La question de l'incomplétude de la logique du 1er ordre a été résolue par Godel en utilisant encore un argument diagonal.

Il est naturel de se demander si un tel argument diagonal ne permettrait pas de conclure pour notre conjecture :

$$P \neq NP?$$

Hélas il semblerait que cela ne soit pas possible.

2.5 $NP \cap co - NP$

Les problèmes de $NP \cap co - NP$, sont appelés bien caractérisés, car ils possèdent un certificat polynomial en cas de réponse OUI **et** en cas de réponse NON.

Bien sûr nous avons : $P \subseteq NP \cap co - NP$.

Cette question de la bonne caractérisation fut à l'origine du questionnement sur la complexité des problèmes par J. Edmonds [8]. Ainsi peut-on expliquer la différence de difficulté algorithmique entre le problème de la recherche d'un parcours Eulérien (passant une fois et une seule par chaque arête) dans un graphe et celui de la recherche d'un parcours Hamiltonien (passant une fois et une seule par chaque sommet).

Dans le cas du parcours Eulérien, le fameux théorème d'Euler propose une caractérisation de l'existence d'un tel parcours, dans le deuxième cas, il n'y a toujours pas de "bonne" caractérisation de l'hamiltonisme connue.

Les exemples classiques :

- Premier. Il est facile de voir que $co\text{-Premier} = \text{Composé} \in NP$ en utilisant la notion de certificat polynomial : les diviseurs d'un nombre qui n'est pas premier permettent de produire un certificat polynomial. Donc $\text{Premier} \in co - NP$. Par la suite Pratt [15] a prouvé l'existence d'un certificat polynomial d'un nombre premier et ainsi $\text{Premier} \in NP \cap co - NP$. Enfin en 2002, un algorithme polynomial a été trouvé [1].
- La programmation linéaire. Pour ce problème le scénario est à peu près identique. La dualité de la programmation linéaire permet de monter

que ce problème est dans $NP \cap co - NP$.

Le célèbre algorithme du simplexe Dantziq des années 60 [5] ne permettait pas d'affirmer la polynomialité du problème, car sa complexité dans le plus mauvais cas est exponentielle.

Khachiyan a produit le premier algorithme polynomial en 1979 [13].

- Théorèmes Min-Max
- Théorèmes du type Kuratowski ou Seymour et Roberston (caractérisation à l'aide d'un ensemble fini de configurations, par exemple de type mineurs de graphes).

En fait les exemples ci-dessus suggèrent que l'existence d'une bonne caractérisation implique la polynomialité du problème, d'où la très intéressante conjecture :

$$P = NP \cap co - NP?$$

Dans ce cadre, il reste deux problèmes sur lesquels de nombreux informaticiens essayent de construire un algorithme polynomial. Parity Game et Transversal Minimal.

De nombreuses personnes conjecturent que Parity Game est polynomial, car il existe des algorithmes qui marchent très bien en pratique pour ce problème (ils ont l'air d'être linéaires et l'on ne sait pas caractériser leurs plus mauvais cas). Parity Game est équivalent au calcul propositionnel dans le $\mu - calcul$.

Intéressons nous au problème du transversal minimal issu des bases de données et considérons une formalisation en termes de graphes.

Etant donné un graphe biarti $G = (X, Y, E)$ un transversal minimal est un ensemble $T \subseteq X$ tel que $V(T) = Y$ (où $V(T)$ représente le voisinage de T) et T est minimal pour l'inclusion avec cette propriété. Notons $Tr(G)$ l'ensemble des transversaux minimaux de G .

Nom (*Transversal minimal*)

Données: $G = (X, Y, E)$ un graphe biparti, F une famille de sous ensembles de X

Résultat: A-t-on $Tr(G) = F$?

Il est facile de vérifier polynomialement $F \subseteq Tr(G)$, c'est l'autre inégalité qui est difficile à vérifier algorithmiquement. Clairement Transversal Minimal $\in co - NP$, car en cas de réponse négative, un transversal manquant est un certificat polynomial.

Ce problème de transversal minimal a de nombreuses formulations équivalentes en logique, bases de données, treillis ... et a été très étudié. Parmi les résultats intéressants, Fredman et Khachiyan ont proposé un algorithme

en $O(n^{\log n})$ [9].

2.6 Problèmes difficiles à classer dans NP

On connaît peu de choses sur le problème suivant, en particulier il est non classé (i.e. on ne sait s'il appartient à P ou NP).

Nom (*Triangulation minimale*)

Données: n points du plan

Résultat: Trouver une triangulation qui minimise la somme des longueurs des triangles

Bien entendu la triangulation de Delaunay ne fournit pas toujours une solution optimale. On peut essayer de résoudre le problème en posant des conditions sur les points.

Le graphe de visibilité d'un polygone simple du plan, est un graphe dont les sommets sont ceux du polygone et l'on met une arête entre deux sommets du graphe lorsque les sommets correspondants du polygone se "voient".

Nom (*Graphe de visibilité*)

Données: $G = (X, E)$ un graphe non orienté

Résultat: Construire le polygone dont il est le graphe de visibilité.

On ne sait même pas si ce problème (la version problème de décision) est dans NP .

Il a été montré que pour dessiner le polygone associé à un graphe de visibilité donné il fallait prévoir une grille de taille au moins exponentielle en la taille du graphe (Lin, Skiera).

Doit-elle être doublement exponentielle ?

Enfin pour clore par un autre exemple sur les graphes, histoire de changer un peu.

On considère un ensemble de n cercles du plan et l'on définit le graphe d'intersection de ces n cercles. Une question naturelle comment reconnaître si un graphe est le graphe d'intersection de cercles.

Même dans le cas où les cercles sont de rayon unité, on ne sait pas si ce problème de décision appartient à NP .

2.7 Retour sur la fameuse conjecture

Et si nous avons $P = NP$?

Voici quelques arguments :

- La plupart des problèmes NP-complets sont dans NDLIN (non déterministe linéaire).
- Les algorithmes à base de configurations exclues, un peu à la manière de théories à base de mineurs, de Seymour et Roberston, sont polynomiaux mais avec des constantes incroyablement grandes. Ceci élargi ce que notre connaissance sur P .
- Les meilleurs bornes de complexité sur NP sont de type $4n$.

Toutes ces raisons font dire à Bill Cook que finalement NP, c'est peut-être l'ensemble des problèmes possédant un algorithme linéaire de résolution !

Chapitre 3

Quelques exemples de réductions polynomiales

Dans ce chapitre, vous trouverez des indications sur les réductions, mais elles ne sont pas toutes complètement rédigées, par contre la transformation polynomiale est décrite, il vous suffit de vérifier...

3.1 Réductions par restriction

On considère le problème classique de la partition d'entiers :

Nom (*Partition*)

Données: a_1, \dots, a_n , n entiers positifs

Résultat: Existe-t-il un sous-ensemble $I \subset [1, n]$ tel que :

$$\sum_{i \in I} a_i = \sum_{j \notin I} a_j ?$$

Il est naturel de considérer la restriction suivante :

Nom (*Partition-paire*)

Données: a_1, \dots, a_n , n entiers positifs, tels que :

$$\sum_{i=1}^n a_i \text{ soit paire}$$

Résultat: Existe-t-il un sous-ensemble $I \subset [1, n]$ tel que :

$$\sum_{i \in I} a_i = \sum_{j \notin I} a_j ?$$

Le résultat suivant est immédiat :

Théorème 1 *Partition – paire* \ll_K *Partition* et
Partition \ll_K *Partition – paire*.

Preuve: La première inégalité est triviale, car Partition-paire est un cas particulier de Partition et il suffit donc d'utiliser comme transformation polynomiale l'identité.

La deuxième l'est un peu moins, mais il suffit de remarquer que l'existence d'une solution implique de la somme des a_i soit paire.

Ainsi pour démontrer la deuxième inégalité on va tout simplement transformer polynomialement une instance de Partition en un instance de Partition-paire de la façon suivante :

On calcule $S = \sum_{i=1}^{i=n} a_i$.

Si S est paire on a affaire à une donnée de Partition-paire et la transformation sera l'identité.

Sinon, comme cette instance n'a pas de solution il suffit de lui associer une instance de Partition-paire, n'ayant pas de solution. Par exemple l'instance $a_1 = 1, a_2 = 3$. \square

Les deux problèmes sont polynomialement équivalents on notera \approx . On remarquera en outre que les réductions polynomiales n'utilisent pas nécessairement des transformations bijectives.

Nous allons maintenant introduire une variante du problème de la partition.

Nom (*Somme-exacte*)

Données: a_1, \dots, a_n , n entiers positifs, k un entier

Résultat: Existe-t-il un sous-ensemble $I \subset [1, n]$ tel que :

$$\sum_{i \in I} a_i = k?$$

Théorème 2 *Partition \approx Somme-exacte.*

Preuve: En remarquant que lorsque le problème de la partition admet une solution I , nécessairement :

$$\sum_{i \in I} a_i = \sum_{j \notin I} a_j = \frac{1}{2} \sum_{1 \leq i \leq n} a_i = h.$$

Ainsi Partition est un cas particulier du problème Somme-exacte en prenant $k = h$.

D'où *Partition \ll_K Somme - exacte.*

Pour la deuxième inégalité, nous allons transformer une donnée a_1, \dots, a_n, k de Somme-exacte en une donnée $b_1, \dots, b_n, b_{n+1}, b_{n+2}$ de Partition.

Pour ce faire on pose $S = \sum_{i=1}^{i=n} a_i$, et $b_i = a_i$ pour $1 \leq i \leq n$, $b_{n+1} = S + k$ et $b_{n+2} = 2S - k$.

On remarque : $\sum_{i=1}^{i=n+2} a_i = 4S$.

Donc s'il existe une solution de ce problème de partition, b_{n+1} et b_{n+2} ne peuvent appartenir au même ensemble de cette partition, car $b_{n+1} + b_{n+2} = 3S > 2S$.

S'il existe une solution, soit I l'ensemble des indices des éléments qui sont avec b_{n+2} , nécessairement : $\sum_{i \in I} a_i = k$, et nous obtenons donc une solution du problème initial Somme-exacte.

La réciproque est triviale. \square

Théorème 3 $3DM \ll_K$ Somme-exacte.

Preuve: On utilise le problème 3DM (couplage d'hypergraphes) défini comme suit :

Nom (3DM)

Données: Trois ensembles $X = \{x_1, \dots, x_q\}$, $Y = \{y_1, \dots, y_q\}$, $Z = \{z_1, \dots, z_q\}$, un hypergraphe $H = (X.Y.Z, E)$, la famille de parties E est donc constituée de triplets.

Résultat: H admet-il un couplage parfait, (i.e. un ensemble $M \cup E$, tel que $|M| = q$ et M est constitué de parties deux à deux disjointes) ?

A une donnée de 3DM on associe une donnée de Somme-exacte en associant à chaque $m = (x_i, y_j, z_h) \in E$ un entier $f(m)$ dont le développement binaire se définit comme suit :

On considère des entiers définis sur $3q$ bits, les q premiers correspondent aux x_i , les q suivants aux y_j et les q derniers aux z_h . $f(m)$ sera l'entier ayant exactement 3 bits non nuls correspondants aux coordonnées x_i , y_j et z_h .

Autrement dit $f(m) = 2^i + 2^{q+j} + 2^{2q+h}$.

Afin d'éviter les problèmes des retenues dans les sommes, on considérera pour chaque coordonnée un paquet de $p = \log_2(|E|)$ bits.

Donc $f(m) = 2^{p(i-1)+1} + 2^{pq+p(j-1)+1} + 2^{2pq+p(h-1)+1}$.

Pour terminer il suffit de prendre pour k le nombre dont le développement binaire admet un 1 pour chaque coordonnée :

$$k = \sum_{i=0}^{i=3q-1} 2^{ip+1}$$

Il suffit alors de vérifier qu'une solution à Somme-exacte correspond à un couplage parfait. \square

Nous allons maintenant utiliser la NP-complétude du problème Partition, pour en déduire celle du problème du Sac-à-dos.

Théorème 4 $Partition \ll_K$ Sac-à-dos

Preuve: Présentons tout d'abord le problème du sac-à-dos :

Nom (*Sac-à-dos*)

Données: n objets numérotés de 1 à n , munis d'un poids $p_i \in R^+$ et d'une utilité $u_i \in R^+$, P_{max} , U deux réels positifs

Résultat: Existe-t-il un sous-ensemble $I \subset [1, n]$ tel que :

$$\sum_{i \in I} p_i \leq P_{max} \text{ et}$$

$$\sum_{i \in I} u_i \geq U ?$$

Etant donné une instance du problème Partition, nous allons lui associer une donnée de Sac-à-dos en prenant n objets numérotés de 1 à n tels que :

$$p_i = a_i = u_i \quad \forall i, 1 \leq i \leq n. \text{ Il suffit alors de prendre}$$

$$P_{max} = U = 1/2 \sum_{i \in [1, n]} a_i.$$

□

En conclusion, si l'on admet que 3DM est NP-complet, cf. Garey-Johnson [11] p.50, nous en déduisons que Partition et Somme-Exacte sont aussi NP-complets.

3.2 Un problème NP-complet typique

Le problème de l'arrêt d'une machine de Turing non déterministe est en fait le prototype du problème NP-complet.

Théorème 5 *CycleHamiltonien* \ll_K *Arret(NDTM)*

Preuve:

On construit une machine de Turing non déterministe, comme suit :

Choisir une arête du graphe de manière non déterministe.

Après $n-1$ choix successifs d'arêtes, l'algorithme termine si on a trouvé un cycle hamiltonien, sinon on part dans une boucle infinie.

Donc si on sait résoudre le problème de l'arrêt d'une NDTM, on sait résoudre le problème du cycle Hamiltonien. □

Cette transformation est générique et fonctionne pour tous les problèmes de NP.

3.3 SAT et dépendances

SAT étant le problème NP-complet emblématique, il a été étudié sous tous les angles, et c'est l'un des problèmes pour lequel la frontière P/NP-Complet a été le mieux étudiée. En effet si l'on ajoute des contraintes sur

les données d'un problème NP-complet il est parfois possible de proposer un algorithme polynomial sur le sous-problème ainsi défini.

On introduit la restriction naturelle de SAT :

Nom (k -SAT)

Données: un ensemble de variables booléennes x_1, \dots, x_n , une collection de clauses $C = \{C_1, \dots, C_m\}$ où chaque clause possède **au plus** k variables

Résultat: C est-elle satisfiable ?

1-SAT est trivialement polynomial et de plus :

Théorème 6 $2 - SAT \in P$.

Preuve: À une instance de 2-SAT on associe un graphe orienté G dont les sommets correspondent aux variables. À chaque variable booléenne x_i on associe deux sommets x_i et \bar{x}_i , et à chaque clause $(\bar{u} \vee v)$ ou $(v \vee \bar{u})$, on associe les deux arcs (u, v) et (\bar{v}, \bar{u}) .

Il suffit alors de remarquer que l'ensemble de clauses est insatisfiable ss'il existe une variable et sa complémentaire qui appartiennent à une même composante fortement connexe de G .

Montrons cette équivalence :

Supposons les clauses satisfiables et qu'il existe une variable x qui appartient à la même composante fortement connexe que \bar{x} .

Si dans une solution on affecte la valeur vrai à x , en considérant le chemin de x à \bar{x} cela implique que l'on doit aussi affecter la valeur vrai à \bar{x} ce qui est absurde.

Dans le cas où l'on affecte la valeur faux à x , on considère la chemin de \bar{x} à x pour obtenir une contradiction.

Réciproquement, s'il n'existe pas de composante fortement connexe de G qui contienne une variable et sa variable complémentaire alors, tant qu'il existe une variable non affectée, on choisit x telle qu'il n'existe pas de chemin de x à \bar{x} , et l'on affecte vrai à la variable x ainsi qu'à tous ses descendants dans G . Clairement ce procédé fournit une solution de 2-SAT. \square

La preuve ci-dessus a été tirée de [2], et l'on peut en déduire un très bel algorithme linéaire. Certains auteurs présentent ce cas particulier de SAT comme celui des clauses binaires.

Théorème 7 $SAT \ll_K 3 - SAT$

Preuve: On remplace chaque clause ayant $k > 3$ littéraux par un ensemble de clauses à exactement 3 variables comme suit :

$$C = (y_1 \vee y_2 \dots \vee y_k) \text{ est remplacée par } \\ (y_1 \vee y_2 \vee z_1) \wedge (\bar{z}_1 \vee y_3 \vee z_2) \dots (\bar{z}_{k-3} \vee y_{k-1} \vee y_k) \\ \text{en introduisant } z_1, \dots, z_{k-3} \text{ soit } k-3 \text{ nouvelles variables booléennes. } \square$$

Théorème 8 *Tovey [18] 3-SAT reste NP-complet même si chaque variable apparaît au plus trois fois et chaque littéral au plus deux fois.*

Preuve: S'il existe une variable x intervenant $k \geq 3$ fois, on va remplacer ses occurrences par x_1, x_1, \dots, x_k , soit k nouvelles variables booléennes et on ajoute les clauses suivantes :

$$(\bar{x}_1 \vee x_2)(\bar{x}_2 \vee x_3) \dots (\bar{x}_k \vee x_1)$$

Cet ensemble de clauses assure l'égalité (dans toute instance satisfaisant l'ensemble de clauses) des variables $x_1, x_1 \dots x_k$. \square

Considérons maintenant la variante suivante de SAT :

Nom $((r, s)\text{-SAT})$

Données: un ensemble de variables booléennes x_1, \dots, x_n , une collection de clauses $C = \{C_1, \dots, C_m\}$ où chaque clause possède **exactement** r variables et chaque variables apparaît dans au plus s clauses

Résultat: C est-elle satisfiable ?

Théorème 9 *Tovey [18] $\forall r (r, r)\text{-SAT}$, tout instance est satisfiable.*

Autrement dit le problème admet toujours au moins une solution et donc ce sous-problème est trivialement dans P. Et la contradiction n'est qu'apparante avec le théorème précédent. En effet le théorème précédent ne permet pas d'affirmer que $(3, 3)\text{-SAT}$ est NP-complet.

La différence tient essentiellement sur le nombre de variables par clause. En effet avec la définition que nous avons choisie pour 3-SAT , une clause admet au plus 3 variables, et curieusement pour les instances de 3-SAT du théorème précédent, il peut y avoir des clauses à 2 variables et ce sont elles qui sont difficiles !

Preuve: Considérons le graphe biparti d'incidence variables clauses. Dans ce graphe les sommets associés aux clauses sont de degré 3 et donc de degré maximum. Un théorème classique sur les couplages dans les graphes bipartis affirme l'existence d'un couplage saturant les sommets de degré maximum.

Ainsi il existe un couplant saturant les clauses, ce couplage permet d'associer à chaque clause une variable unique qui va permettre de trouver un affectation valide. \square

Pas-tous-égaux ou NAE-SAT est la variante de 3-SAT où l'on cherche une solution où chaque clause $(x \vee y \vee z)$ interdit les deux cas :

$x=y=z= \text{Faux}$ et $x=y=z= \text{VRAI}$.

Théorème 10 *Schaeffer [17] NAE-SAT est NP-complet.*

Preuve: À chaque clause $C = (x \vee y \vee z)$ on associe les deux clauses suivantes :

$C' = (x \vee y \vee u)$ et $C'' = (\bar{u} \vee z \vee \text{Faux})$

où u est une nouvelle variable.

Il est facile de voir que C est satisfiable ssi C' et C'' sont NAE-satisfiables.

\square

Une-parmi-trois-SAT est la variante de 3-SAT où l'on cherche une solution qui satisfasse exactement une variable par clause. Nous allons maintenant montrer que ce problème est NP-complet.

Théorème 11 *Schaeffer [17] Une-parmi-trois-SAT est NP-complet.*

Preuve: À chaque clause $C = (x \vee y \vee z)$ on associe les trois clauses suivantes :

$C_1 = (\bar{x} \vee u \vee v)$, $C_2 = (v \vee y \vee w)$ et $C_3 = (w \vee w' \vee \bar{z})$

où u , v , w et w' sont 4 nouvelles variables.

Il est facile de voir que C est satisfiable ssi C_1, C_2 et C_3 sont une-parmi-trois-satisfiables.

\square

Nom (*Max-2-SAT*)

Données: Un ensemble de variables booléennes x_1, \dots, x_n , une collection de clauses $C = \{C_1, \dots, C_m\}$ où chaque clause possède **au plus 2** variables, k un entier

Résultat: Existe-t-il une interprétation satisfaisant au moins k clauses ?

Théorème 12 *Max2-SAT est NP-complet.*

Preuve:

À chaque clause $C = (x \vee y \vee z)$ on associe les 10 clauses suivantes :

$(x)(y)(z)(w)$

$(\bar{x} \vee \bar{y})(\bar{y} \vee \bar{z})(\bar{x} \vee \bar{z})$

$$(x \vee \overline{w})(y \vee \overline{w})(z \vee \overline{w})$$

Où w est une nouvelle variable.

On peut vérifier qu'une interprétation satisfaisant la clause C peut s'étendre en une interprétation satisfaisant au moins 7 des 10 clauses.

Lorsque $x = y = z = \text{Faux}$ au plus 6 clauses parmi les 10 seront satisfaites.

Pour la réduction il suffit de prendre $k = 7m$.

□

3.3.1 Application

Une question soulevée par C. Bessière à partir d'un problème de satisfaction de contraintes.

On considère le problème de coloration de graphe suivant :

Nom (*Coloration avec contraintes*)

Données: un graphe $G = (X, E)$ non orienté, k couleurs et des contraintes précisant les deux couleurs autorisées (parmi les k) pour chaque sommet

Résultat: G admet-il une coloration respectant ces contraintes ?

Bien que la coloration soit un problème NP-complet, nous allons ici nous ramener au problème 2-SAT.

Pour ce faire on définit des variables booléennes x_i^j avec $1 \leq i \leq n$ et $1 \leq j \leq k$.

$x_i^j = \text{VRAI}$ ssi la couleur j est autorisée pour le sommet i .

Avec ces variables la contrainte sur les deux couleurs autorisées pour chaque sommet se traduit avec deux clauses à deux variables : la première clause spécifie qu'au moins une des deux couleurs est prise et l'autre que les deux ne peuvent être prises en même temps.

Il suffit après d'ajouter pour chaque arête du graphe une clause interdisant que les deux sommets extrémités possèdent la même couleur.

Ainsi la construction suivante nous ramène à 2-SAT et le problème initial est donc polynomial.

3.3.2 Exercices

1. Montrer que (3,4)-SAT est NP-complet.
2. Montrer que le problème suivant est NP-complet.

Nom (*Partie héréditaire maximale*)

Données: Un graphe orienté $G = (X, U)$ sans circuit, une valuation $\omega : X \rightarrow Z$, et $k \in Z$

Résultat: Existe-t-il une partie héréditaire $S \subseteq X$ telle que $\omega(S) \geq k$?

On rappelle qu'une partie héréditaire vérifie la propriété suivante :

$(x \in S)$ et (il existe un chemin de x à y dans G) impliquent que $y \in S$.

3.3.3 D'autres cas polynomiaux de SAT

Les clauses de Horn.

Nom (*Horn-SAT*)

Données: un ensemble de variables booléennes x_1, \dots, x_n , une collection de clauses $C = \{C_1, \dots, C_m\}$ où chaque clause possède **au plus** un littéral positif

Résultat: C est-elle satisfiable ?

Théorème 13 *Horn-SAT est polynomial.*

Preuve:

1- On commence par éliminer toutes les clauses à un littéral en simplifiant tant que c'est possible.

2- Ensuite on affecte à Vrai tous les littéraux positifs figurant dans les clauses restantes. Cela peut créer éventuellement de nouvelles clauses à un littéral, on applique donc une deuxième fois la simplification décrite en 1).

3- Il ne reste donc plus que des clauses dont tous les littéraux sont négatifs.

Il existe des algorithmes linéaires pour résoudre Horn-SAT.

Théorème 14 *Toute instance de k -SAT dont les clauses ont exactement k variables et ayant strictement moins de 2^k clauses est satisfiable.*

Preuve: Il suffit de remarquer qu'une clause rend impossible exactement une des affectations 2^k possibles. Pour ceux qui ne sont pas satisfaits par cet argument grossier, on peut faire un raisonnement probabiliste simple. \square

Théorème 15 *Toute instance de k -SAT dont les clauses ont exactement k variables et dans laquelle une variable apparaît dans au plus $\frac{2^k-2}{k}$ clauses est satisfiable.*

Preuve: On peut retrouver ce théorème à partir d'un résultat d'Erdos et Lovász, mais aussi par un raisonnement probabiliste en utilisant le lemme local de Lovász, cf. l'article de Molloy dans [12]. \square

Tovey [18] avait conjecturé la même propriété pour 2^{k-1} , mais Dubois [7] a proposé un contre-exemple.

Une question naturelle : le théorème précédent est-il le meilleur possible ? Nous connaissons d'autres classes polynomiales de SAT. Horn-SAT définies positives, instances de SAT pour lesquelles chaque clause admet exactement un littéral positif.

La question du renommage

3.4 Graphes d'incidence variables-clauses

Une clique est un sous-graphe complet et le problème Clique maximale se définit comme suit :

Nom (*Clique maximale*)

Données: $G=(X,E)$ un graphe non-orienté, k un entier

Résultat: Existe-t-il une clique de cardinal $\geq k$ dans G ?

Théorème 16 $3\text{-SAT} \ll_K \text{Clique maximale}$

Preuve: Etant donné une instance I des 3-SAT on construit un graphe $G(I)$ non orienté sur les clauses, comme suit :

À chaque clause $C_j = (u_1^i, u_2^i, u_3^i)$, $1 \leq j \leq m$, on associe un stable à trois sommets, notés u_1^i, u_2^i, u_3^i et l'on construit l'arête $u_l^i u_k^j$ ssi $u_l^i \neq \overline{u_k^j}$ pour $l, k \in \{1, 2, 3\}$.

La donnée I est satisfiable ssi le graphe $G(I)$ admet une clique de taille m . \square

Nom (*Arcs interdits*)

Données: un graphe orienté sans circuit $G=(X, U)$, un ensemble de couples (a_i, b_i) , $1 \leq i \leq k$, deux sommets $s, p \in X$

Résultat: Existe-t-il un chemin de s à p dans G , pour lequel aucun (a_i, b_i) ne soit une corde ?

Ce problème intervient dans les problèmes de raisonnement liés à l'héritage non-monotone, les cordes représentant les exceptions à l'héritage. Il m'avait été posé par Roland Ducournau en ces termes d'arcs d'exception, et le théorème de NP-complétude ci-dessous interdit tout algorithme vraiment efficace !

Théorème 17 [10] $3\text{-SAT} \ll_K \text{Arcs interdits}$

Preuve: Soit I une donnée de 3-SAT, on va lui associer un graphe sans circuit $G(I)$, en fait une orientation d'un sous graphe du graphe utilisé dans la transformation $3\text{-SAT} \ll_K \text{Clique maximale}$.

On choisit un ordre quelconque sur les clauses, par exemple $C_1 \leq C_2 \leq \dots \leq C_m$.

À chaque clause $C_j = (u_1^i, u_2^i, u_3^i)$, $1 \leq j \leq m$ on associe un stable à trois sommets, notés u_1^i, u_2^i, u_3^i . On crée deux nouveaux sommets s et p respectivement l'unique source et l'unique puits de G . On ajoute tous les arcs de s aux sommets de C_1 (resp. des sommets de C_m à p), ainsi que tous les arcs des sommets de C_i à ceux de C_{i+1} pour $i=1$ à $m-1$ (biparti complet entre deux clauses consécutives).

Il suffit alors de prendre comme arcs interdits les couples : u_l^i, u_k^j ssi $u_l^i = \overline{u_k^j}$ avec $i \leq j$ et $l, k \in \{1, 2, 3\}$.

La donnée I est satisfiable ssi le graphe $G(I)$ admet un chemin de s à p sans corde interdite.

En effet tout chemin de s à p traverse chaque clause C_i en un sommet unique u_j^i . On peut donc associer à ce chemin une interprétation des variables satisfaisant toutes les clauses, à la condition qu'une même variable n'intervienne pas sous forme directe et complémentée sur ce chemin (ce qui correspond aux cordes interdites).

Réciproquement si I est satisfiable, il existe une affectation des variables satisfaisant toutes les clauses (i.e. il existe au moins une variable à VRAI par clause. Il suffit alors de considérer un chemin de s à p , obtenu en prenant pour chaque un sommet dont la valeur de vérité est VRAI dans cette affectation. Ce chemin est trivialement sans corde interdite.

□

On remarquera que dans cette transformation à une affectation satisfaisant I , il peut correspondre plusieurs chemins de s à p .

De cette transformation on peut déduire un algorithme de résolution de 3-SAT en termes de chemins. Il suffit d'énumérer tous les chemins de s à p . Hélas il peut y en avoir un nombre exponentiel!

Nom : Sous-ordre total maximal

Données : $G = (X, U)$ un graphe sans circuit, un entier k

Question : Existe-t-il un chemin $[x_1, \dots, x_k]$ de G dont tous les arcs de transitivité appartiennent à G ?

Ce problème m'a été posé par C. de La Higuera et provenait d'un problème d'apprentissage.

Théorème 18 $3\text{-SAT} \ll_K$ *Sous-ordre total maximal*

Preuve: On adapte facilement la transformation précédente. À chaque clause on associe comme ci-dessus un ensemble de 3 sommets non adjacents entre eux, et entre deux clauses C_i et C_{i+1} , on intercale un sommet intermédiaire z_i successeur de tous les sommets de C_i et prédécesseur de tous les sommets de C_{i+1} . Enfin on ajoute tous les arcs de transitivité exceptés les arcs du type :

$$u_l^i, u_k^j \text{ tels que } u_l^i = \overline{u_k^j} \text{ avec } i \leq j \text{ et } l, k \in \{1, 2, 3\}.$$

Il est alors clair que l'ensemble des clauses est satisfiable ss'il existe un sous-ordre total ayant tous ses arcs de transitivité. \square

Un arc de retour d'un chemin $[s = x_0, x_1, \dots, x_k = p]$ est tout simplement un arc $x_j x_i$ avec $0 \leq i \leq j \leq k$. Le problème suivant intervient dans l'analyse des graphes de programmes à des fins de compilation efficace.

Nom (*Arcs de retour*)

Données: un graphe orienté $G=(X, U)$, deux sommets $s, p \in X$

Résultat: Existe-t-il un chemin de s à p dans G , sans arc de retour ?

Théorème 19 $3\text{-SAT} \ll_K$ *Arcs de retour*

Preuve: Même transformation que pour arcs interdits, on considère le graphe $G(I)$ et au lieu d'introduire des cordes interdites, on considère des arcs de retour :

$$u_k^j u_l^i \text{ ssi } u_l^i = \overline{u_k^j} \text{ avec } i \leq j \text{ et } l, k \in \{1, 2, 3\}. \square$$

Théorème 20 $3\text{-SAT} \ll_K$ *Coloration*

Preuve: On considère une donnée I de 3-SAT utilisant $C_j, 1 \leq j \leq p$ clauses et $x_1, \dots, x_n, n \geq 4$ variables. On lui associe un graphe $G(I)=(X, E)$ de la façon suivante :

$$X = \{v_i, 1 \leq i \leq n\} \cup \{x_i, 1 \leq i \leq n\} \cup \{\overline{x_i}, 1 \leq i \leq n\} \cup \{C_j, 1 \leq j \leq p\}$$

$$E = \{v_i v_j, i \neq j\} \cup \{x_i \overline{x_i}, 1 \leq i \leq n\} \cup \{v_i x_j, v_i \overline{x_j}, i \neq j\} \cup \{u_i C_j, u_i \notin C_j \text{ où } u_i \text{ est un littéral (soit } x_i \text{ ou } \overline{x_i})\}$$

On remarque que $G(\{v_1, \dots, v_n\})$ est un sous graphe complet de $G(I)$, et donc il faut exactement n couleurs pour colorer $G(\{v_1, \dots, v_n\})$, et au moins autant pour $G(I)$.

On peut montrer que $G(I)$ est colorable avec $n+1$ couleurs ssi I est satisfiable. \square

3.4.1 Exercices

1. Montrer à l'aide d'une réduction directe que : Coloration \ll_K 3-SAT.
2. Montrer que Clique maximale \ll_K Max-2-SAT.
3. Le problème de l'indépendant (stable) de taille maximale reste NP-complet sur les graphes sans triangles.

Piste : on associe au graphe G , un nouveau graphe G' en subdivisant deux fois chaque arête de G et l'on a :

$$\alpha(G) = \alpha(G') + |E(G)| \text{ et } \chi(G) = 3.$$

(Argument dû à Poljak 74).

3.5 Extensions de la théorie

Bien faire la différence entre k donné et k fixé, lorsque k est un paramètre que l'on a introduit pour transformer un problème d'optimisation en problème de décision.

#-complet se lit sharp-complet en anglais.

Complexité paramétrée [6].

Chapitre 4

Recueil d'examens du module complexité

Novembre 95

Complexité concrète

On suppose disposer d'un programme A et d'une machine dont l'horloge est accessible par programme. Vous savez que le nombre d'opérations élémentaires exécutées par le programme A sur une donnée de taille n vérifie :

$$T_A(n) = an^2 + bn + c$$

- 1 Comment mesurer pratiquement les constantes a, b et c ?
- 2 Que représentent ces constantes ? En particulier c et b .
- 3 Mêmes questions lorsque $T_A(n) \leq an^2 + bn + c$
- 4* Comment mesurer pratiquement l'influence du swap sur une machine pour un programme donné ? (Le temps d'exécution peut dépendre de l'encombrement de la mémoire au cours de l'exécution du programme).
Comment programmer les algorithmes afin que leur temps de calcul mesurés dépendent le moins possible des algorithmes de swap ?

Réductions

On considère le problème suivant :

Nom : Sous-ordre total max

Données : $G = (X, U)$ un graphe sans circuit, un entier k

Question : Existe-t-il un chemin $[x_1, \dots, x_k]$ de G dont tous les arcs de transitivité appartiennent à G ?

1 Montrer que ce problème est bien dans NP.

2* Montrer que ce problème est NP-complet.

Piste : on peut s'inspirer de la preuve sur arcs interdits, et partir de 3-SAT.

3* Où se trouve la barrière Polynomial/NP-complet sur ce problème de graphes ?

4* En particulier, si tous les degrés entrant et sortant sont ≤ 2 . Est-ce polynomial (fournir un algo) ou NP-complet (fournir une réduction) ?

5* On considère maintenant le problème :

Nom : Chemin sans arc de retour

Données : $G = (X, U)$ un graphe orienté, a, b deux sommets

Question : Existe-t-il un chemin reliant a et b dans G sans arc de retour ?

Ce problème est-il NP-complet ou polynomial ?

6* Où se trouve la barrière Polynomial/NP-complet sur ce problème de graphes ?

Novembre 96

Première partie : 4 points

Votre patron vous demande d'améliorer les performances d'un algorithme qu'il a écrit très rapidement sans trop prendre garde à la complexité. Après plusieurs semaines d'un travail harassant, la lecture exhaustive des livres d'algorithmique, vous réussissez laborieusement à programmer un algorithme optimal.

Hélas les tests de performance sont formels, l'algorithme du patron bien que non optimal, est beaucoup plus rapide sur les jeux de données utilisés.

Comment expliquer un tel phénomène ?

4.0.1 Deuxième partie : 15 points

On considère le problème 2-SAT, i.e. la version du problème SAT dans laquelle les clauses admettent au plus deux variables. On associe à chaque donnée ϕ de 2-SAT, un graphe $G(\phi)$ dont les sommets sont les variables (sous forme directe et complémentée). Plus précisément à chaque variable booléenne x_i correspond deux sommets x_i et $\neg x_i$.

Deux sommets x et y de G sont reliés par un arc ssi la clause $(\neg x \vee y) \in \phi$ or $(y \vee \neg x) \in \phi$.

1. Montrer que $2 - SAT \in NP$.
2. Construire le graphe associé à la formule suivante : $(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_4)$.
3. Montrer que si l'arc $(x, y) \in G$, alors $(\neg y, \neg x) \in G$. Que représentent ces arcs ?
4. Montrer que ϕ n'est pas satisfiable si et seulement s'il existe une variable x et deux chemins de x à $\neg x$ et de $\neg x$ à x dans $G(\phi)$.
5. En déduire que $2 - SAT \in P$.
6. Peut-on poursuivre le raisonnement avec 3-SAT ? Idem pour SAT.
7. Proposer une méthode exacte, à base de chemins dans un graphe (à définir) pour résoudre 3-SAT.

Novembre 97

Analyse de texte : 5 points

Vous trouverez en annexe les premières pages d'un texte sur l'efficacité de la programmation par objet.

une photocopie d'un texte litigieux était jointe!

1. Quelles mesures de complexité faut-il inventer pour répondre aux besoins de cet article. Que veulent mesurer les auteurs?
2. Quel sens donner à la définition d'un programme optimal?
3. Commentaire général sur ces 3 pages.

Réductions entre problèmes polynomiaux : 5 points

Définition 1 *On dit qu'un problème $A \leq_{f(n)} B$, s'il existe un algorithme qui permet de résoudre le problème A en $O(f(n))$ où n représente la taille de la donnée du problème A . Cet algorithme peut utiliser une procédure de résolution du problème B sur des données de taille $O(n)$. Chaque utilisation sera comptée en $O(1)$.*

1. Montrer que cette relation est un préordre.
2. Quel est le rapport avec la réduction définie dans le polycopié chapitre 1, page 11 (du genre l'une implique l'autre)?
3. Y-a-t-il un rapport avec les réductions de Turing et de Karp?
4. Donner des exemples d'application.

Des réductions, des vraies : 15 points

Démarrons par le problème de l'isomorphisme. On prendra comme problème standard, le problème de décision de l'existence d'un isomorphisme entre deux graphes non orientés.

Définition 2 *Un problème est dit **isomorphisme-complet** s'il est équivalent au problème standard. Équivalent au sens de la Turing réduction.*

1. Montrer que l'isomorphisme de graphes bipartis est isomorphisme complet.
2. Montrer que le problème de la recherche d'un isomorphisme entre deux graphes est isomorphisme complet.
3. Montrer que le problème de la recherche de la partition en classes de sommets automorphes d'un graphe est isomorphisme complet. Deux sommets x et y appartiennent à la même classe si et seulement s'il existe un automorphisme du graphe qui échange x et y .
4. Les questions précédentes contredisent-elles le rapport évoqué en cours entre problèmes de décision et problèmes de recherche ?
5. Montrer que le problème de la recherche d'un isomorphisme entre deux graphes réguliers est isomorphisme complet.

Définition 3 On appelle *centre* d'un graphe $G = (X, E)$, un sommet $x \in X$ dont l'excentricité est minimale. L'excentricité $ecc(x)$ d'un sommet est égale à $Max_{y \in X} \{d(x, y)\}$.

1. Montrer que l'isomorphisme de deux arbres est polynomial.
2. * Montrer que le problème de l'isomorphisme reste isomorphisme complet sur la classe des graphes ayant un seul centre.

Définition 4 Un automorphisme θ d'un graphe $G = (X, E)$ est dit **sans point fixe**, s'il n'existe pas de sommet $x \in X$ tel que $\theta(x) = x$.

** Montrer que le problème de l'existence d'un tel automorphisme pour un graphe donné est NP-complet.

Piste : on peut partir de 3-SAT.

** **Une petite dernière** : 2-SAT est polynomial ainsi que Horn-SAT (restriction de SAT aux clauses de Horn), qu'en est-il de l'union des deux ? C'est à dire les ensembles de clauses qui sont soit de Horn soit à deux variables.

Polynomial ou NP-complet ?

Décembre 98

Complexité concrète

- 1-facile** : Évaluer sur une machine RAM le coût d'un algorithme de calcul de factorielle(n)= $n!$.
Peut-on dire que ce calcul est polynomial ?
- 2** : On considère maintenant une machine RAM dont les mots ont une longueur fixée (k bits). Évaluer sur ce nouveau modèle de machine l'algorithme présenté à la question précédente.
- 3** : Qu'en conclure sur ces deux modèles de machines ?

Réductions

Nous avons vu en cours une réduction concernant Max 2-SAT. Bien que 2-SAT soit polynomial, Max 2-SAT est NP-difficile ! Les questions qui suivent devraient permettre de mieux comprendre ce résultat.

1-facile : **Nom** (*Max(2, 2)-SAT*)

Données: un ensemble de variables booléennes x_1, \dots, x_n , deux collections de clauses $C = \{C_1, \dots, C_m\}$ et $S = \{S_1, \dots, S_k\}$ où chaque clause possède au plus 2 littéraux

Recherche (*On cherche une interprétation que satisfasse les clauses de C et maximisant la satisfaction des clauses de S*)

Montrer que ce problème est NP-difficile ?

2-facile : Considérons les deux restrictions suivantes de Max(2, 2)-SAT :

Nom (*Max(2, 1)-SAT*)

Données: un ensemble de variables booléennes x_1, \dots, x_n , deux collections de clauses $C = \{C_1, \dots, C_m\}$ et $S = \{S_1, \dots, S_k\}$ où chaque clause possède de C au plus 2 littéraux, mais dans lequel chaque clause de S possède au plus un littéral.

Recherche (*On cherche une interprétation que satisfasse les clauses de C et maximisant la satisfaction des clauses de S*)

Nom (*Max(2, 1+)-SAT*)

Données: un ensemble de variables booléennes x_1, \dots, x_n , deux collections de clauses $C = \{C_1, \dots, C_m\}$ et $S = \{S_1, \dots, S_k\}$ où chaque clause

possède de C au plus 2 littéraux, mais dans lequel chaque clause de S est un un littéral positif.

Recherche (*On cherche une interprétation que satisfasse les clauses de C et maximisant la satisfaction des clauses de S*)

Montrer que Max(2, 1)-SAT et Max(2, 1+)-SAT sont polynomialement équivalents. (Piste : Il suffit d'exhiber les deux Turing-réductions, notées $<_T$).

3-* Montrer que Max(2, 1+)-SAT est NP-difficile ?

4 : Soit le problème d'optimisation de graphe :

Nom (*Indépendant Maximum*)

Données: Un graphe $G = (X, E)$

Recherche (*On cherche un ensemble $I \subset X$ de taille maximum tel que $\forall x, y \in I, xy \notin E$*)

Montrer que :

Max(2, 1+)-SAT $<_T$ Indépendant Maximum.

5-* : Montrer que Indépendant Max $<_T$ Max(2,1+)-SAT.

6-* : Mêmes questions (4,5) avec la coloration minimale d'un graphe.

7-* Que peut-on dire de la frontière P/NP-difficile sur Max(2,1+)-SAT ? En particulier, peut-on ajouter encore beaucoup de contraintes et rester NP-difficile ?

Variations autour de SAT

1-facile : Montrer que Décision-SAT $<_T$ Recherche-SAT $<_T$ Décision-SAT

2-* : Quelle propriété avez vous utilisée dans la preuve de la question précédente ? Est-ce transposable au problème du voyageur de commerce ?
 Décision-voyageur $<_T$ Recherche-voyageur $<_T$ Décision-voyageur

3 : On considère le problème suivant :

Nom (*SAT-critique* :)

Données: un ensemble de variables booléennes x_1, \dots, x_n , une collection de clauses $C = \{C_1, \dots, C_m\}$

Recherche (*On cherche à vérifier que cette instance n'est pas satisfiable mais qu'elle le devient dès qu'on enlève une clause*)

Montrer que SAT-critique est NP-difficile.

4 Est-ce que SAT-critique est dans NP ou co-NP ?

5-* Que dire du problème SAT-unique, qui consiste à se poser la question si une instance de SAT admet une unique solution ? Idem pour le problème qui consiste à déterminer la parité du nombre des solutions de SAT.

Parallélisme

1 : À quelles conditions peut-on espérer obtenir un algorithme polynomial sur une machine parallèle pour un problème NP-complet ?

(Piste : On peut comparer le temps mis par un algorithme séquentiel avec celui mis par un algorithme parallèle sur une machine ayant p processeurs).

2 : Quelles conséquences pratiques en tirer ?

4.1 S'il pleut dimanche . . . ***

On considère un graphe $G = (X, E)$ et une numérotation π de ces sommets.

On définit $Succ_{\pi}(x)$ (resp. $Pred_{\pi}(x)$) le nombre de voisins de x dans G qui sont après (resp. avant) x dans π .

Quelle est la complexité du calcul de la numérotation π qui minimise : $\sum_{x \in X} |Succ_{\pi}(x) - Pred_{\pi}(x)|$.

Novembre 1999

4.1.1 Complexité "concrète"

On considère le problème du routage dans un réseau de routeurs (par exemple de type Internet). Les routeurs sont les sommets d'un graphe ayant n sommets et l'on supposera l'existence d'une numérotation globale des routeurs (i.e. à chaque routeur on associe un identifiant $i \in [1, n]$).

De plus chaque routeur i possède et connaît ses canaux d'entrée-sortie numérotés de 1 à $degré(i)$.

Le problème du routage consiste pour un routeur donné, lorsqu'il reçoit un paquet, à calculer le numéro du canal de sortie en fonction de l'adresse

(identifiant) du destinataire du paquet (cette adresse est en général disponible dans l'entête du paquet).

Deux techniques principales sont utilisées :

- un simple accès à une table de routage précalculée qui contient des informations du type : (destination, numéro du canal de sortie).
- Le calcul du plus court chemin du routeur jusqu'à la destination permettant d'en déduire la canal de sortie (dans ce cas, le réseau est stocké en mémoire locale du routeur).

1. Évaluez précisément ces deux techniques en précisant la taille de la donnée, le temps de calcul et l'espace mémoire utilisé.
2. Déterminez l'appartenance de ces méthodes à FP, FNP (F pour fonctionnel, car nous ne sommes pas dans le cadre des problèmes de décision, l'algorithme du routage produit un nombre).
3. ** Peut-on envisager des méthodes plus efficaces en temps, en mémoire ? On peut s'inspirer de la solution proposée par IP dans Internet.

4.1.2 SAT Toujours

On considère la restriction de SAT dans laquelle chaque clause admet au plus un littéral positif.

1. Montrer que cette sous-classe est polynomiale.
2. Il n'est pas si facile de reconnaître les instances de cette sous classe de SAT. En effet il est possible qu'une instance ait au plus un littéral positif par clause, lorsqu'on a procédé à un renommage des variables. Donner un algorithme de renommage.

4.1.3 Autour de SAT

1. Le problème de la vérification qu'une instance de SAT admet **exactement une solution** est-il dans NP ?
2. Quelle est la complexité de la vérification qu'une instance de 2-SAT admet **exactement une solution** ?
3. *

On considère maintenant la résolution d'équations du type :

$$Q_1x_1, \dots, Q_kx_k C_1 \wedge \dots \wedge C_m = 1$$

où Q_i est un quantificateur existentiel (\exists) ou universel (\forall).

3-SAT quantifié est P-espace complet.

Montrer que 2-SAT quantifié est polynomial.

4. On considère maintenant une sous-classe polynomiale de SAT, montrer que si l'on borne le nombre de quantificateurs universels, la résolution quantifiée sur cette sous classe reste polynomiale.
5. ** Que se passe-t-il dans le cas général, i.e. la résolution d'équations quantifiées sur une sous classe polynomiale de SAT ?

4.1.4 Un problème de chemins

On considère le problème suivant :

Nom : Chemins disjoints

Données un graphe orienté $G = (X, U)$, des couples de sommets (s_i, t_i) , $1 \leq i \leq k$.

Question Existe-t-il k chemins sommet-disjoints reliant s_i à t_i , pour $1 \leq i \leq k$.

1. Montrer que ce problème est dans NP.
2. Montrer qu'il est NP-complet.
On pourra pour ce faire partir de 3-SAT.
3. Même résultat pour les graphes non-orientés.
4. * Même résultat avec $k=2$.

4.1.5 Approximation

Notation : On s'intéresse dans cet exercice au coloriage des arêtes d'un graphe. Un coloriage des arêtes d'un graphe est valide si 2 arêtes incidentes sont coloriées dans 2 couleurs différentes. Le problème du coloriage des arêtes d'un graphe consiste à trouver un coloriage valide des arêtes utilisant un nombre minimum de couleurs.

Dans la suite, on appelle $\Delta(G)$, le maximum des degrés des sommets du graphe G .

On sait que :

- le nombre minimal de couleur nécessaires pour colorier les arêtes d'un graphe est $\Delta(G)$ ou $\Delta(G) + 1$,
- savoir si on peut colorier les arêtes d'un graphe en $\Delta(G)$ couleurs est un problème NP-complet,
- le résultat précédent est vrai même si on se limite à la classe des graphes tels que $\Delta(G) = 3$,
- on peut colorier les arêtes d'un graphe en $\Delta(G) + 1$ couleurs en temps polynomial.

Question : Sous les hypothèses $P \neq NP$, montrez qu'il n'existe pas d'algorithme ρ -approchant (s'exécutant en temps polynomial) pour le problème du coloriage des arêtes d'un graphe si $\rho \leq \frac{4}{3}$, mais qu'il existe un algorithme $\frac{4}{3}$ -approchant pour ce problème.

4.2 Pour le week-end

Donner une généralisation du théorème 6 du polycop (sur (r,r)-SAT).

Novembre 2000

4.2.1 On s'échauffe ...

1. Donner une définition en termes simples (compréhensible par le télé-spectateur moyen) de la complexité d'un algorithme, d'un problème.
2. On considère un graphe non orienté G dont le degré maximum est Δ . Ecrire un algorithme polynomial qui détermine si G contient (ou non) une clique de cardinal $\Delta + 1$. (Rappel une clique est un sous graphe complet).

Peut-on généraliser la méthode pour déterminer si G contient une clique de cardinal Δ ?

4.2.2 Assez rigolé, au travail maintenant ...

On considère l'algorithme de parcours de graphe suivant, appelé Maximal Cardinality Search (MCS) :

Parcours MCS**Données:** un graphe non orienté G , un sommet x **Résultat:** un marquage de la composante connexe de x , i.e.

l'ensemble des sommets FERMES à la fin du parcours

 $OUVERTS \leftarrow \{x\}$ $FERMES \leftarrow \emptyset$ $\forall y, Etiquette(y) \leftarrow 0$ **tant que** $OUVERTS \neq \emptyset$ **faire**

$z \leftarrow \{y \in OUVERTS \mid Etiquette(y) \text{ maximum}\}$
$Ajout(z, FERMES)$
Explorer(z)
$Virer(z, OUVERTS)$

Explorer(z)**pour** Tous les voisins y de z **faire**

si $y \in FERMES$ alors
└ Ne rien faire
si $y \in OUVERTS$ alors
sinon
└ $Etiquette(y) \leftarrow Etiquette(y) + 1$
$Ajout(y, OUVERTS)$
$Etiquette(y) \leftarrow Etiquette(y) + 1$

1. **Nom** (*MCS*)**Données:** Un graphe G non orienté, deux sommets a et b , un entier k **Résultat:** Existe-t-il une exécution de l'algorithme précédent (MCS) démarrant au sommet a et explorant b , après avoir exploré k sommets entre a et b ?Montrer que MCS est dans NP et que $MCS \ll_K 3-SAT$ (on s'inspirera de la réduction 3-SAT et clique maximale).

2. *

Nom (*Extrémité d'un MCS*)**Données:** Un graphe G non orienté, deux sommets a et b **Résultat:** Existe-t-il une exécution de l'algorithme précédent (MCS) démarrant au sommet a se terminant en b

Ce problème est-il NP-complet ?

3. *

Pour quels autres types de parcours de graphes peut-on avoir des résultats de ce type (parcours en profondeur, ...)?

4.2.3 Une question d'actualité *

Existe-t-il des systèmes d'élection présidentielle pour lesquels la détermination du vainqueur soit un problème NP-complet?

4.2.4 Un schéma d'approximation pour le problème du sac à dos

Rappel de l'énoncé du problème du sac à dos :

$$\begin{aligned} \text{Max } & \sum_{i=1}^n p_i x_i \\ & \sum_{i=1}^n a_i x_i \leq B \\ & x_i \in \{0, 1\} \end{aligned}$$

Quelques notations : on appelle I , l'ensemble des entiers de 1 à n . On dit que S est une solution si $S \subset I$ et $\sum_{i \in S} a_i \leq B$. On note S^* une solution optimale et P^* le poids du sac à dos correspondant (c'est-à-dire $\sum_{i \in S^*} p_i$).

On peut obtenir une solution en utilisant la méthode gloutonne suivante : on trie les objets par ordre décroissant du rapport $\frac{p_i}{a_i}$ (plus un objet est dense plus on a intérêt à le mettre dans le sac à dos pour maximiser son poids), puis on remplit le sac à dos en prenant les objets dans l'ordre précédemment défini (si un objet ne rentre pas il est rejeté et on passe à l'objet suivant).

1) Calculer la solution obtenue par l'heuristique gloutonne sur l'instance suivante :

	1	2	3	4	5	6	7	8	9
p_i	19	17	9	14	10	7	12	3	4
a_i	14	13	7	11	8	6	11	3	5

($S^* = \{3, 4, 5\}$ et donc $P^* = 33$).

2) montrer que cette heuristique donne une solution S dont le poids du sac à dos correspondant P peut être à un facteur aussi grand que l'on veut de P^* (c'est-à-dire $\frac{P^*}{P}$ est non borné). Indication : pour faire la preuve il suffit

de donner une famille infinie d'instances sur laquelle ce rapport tend vers l'infini.

Soit J une solution du problème, on appelle $glouton(J)$ la solution obtenue en mettant les objets indicés par les entiers de J dans le sac à dos et en ajoutant de façon gloutonne les objets indicés par les entiers de $I - J$ (en les prenant comme précédemment dans l'ordre décroissant du rapport $\frac{p_i}{a_i}$). La solution gloutonne est donc égale à $glouton(\emptyset)$.

3) Calculer sur l'instance précédente $glouton(\{7\})$ et $glouton(\{2, 9\})$.

Soit k , un entier quelconque. On utilise la technique suivante : pour chaque solution J , telle que J soit de cardinalité inférieure à k , calculer $glouton(J)$, on garde la meilleure solution parmi celles ainsi trouvées.

4) calculez l'ordre de grandeur de la complexité de cette heuristique en fonction de n et de k .

5) (question difficile) Montrer qu'en appliquant l'heuristique pour $k = 1$, on obtient une solution S telle que le poids du sac à dos correspondant est supérieur à $\frac{P^*}{2}$. Indication : soient i_1 l'indice de l'objet le plus lourd de S^* (c'est-à-dire $\max_{i \in S^*}(p_i)$), $J = \{i_1\}$ et P le poids du sac à dos obtenu avec la solution $glouton(J)$. Montrer les résultats intermédiaires suivants (si vous n'arrivez pas à les montrer, admettez les pour continuer la preuve) : soit i_m l'indice de l'objet le plus dense de S^* n'appartenant pas à $glouton(J)$, alors $P^* \leq P + p_{i_m}$. En déduire $P^* \leq 2 * P$ puis le résultat souhaité.

Si on utilise l'heuristique avec k sur une instance quelconque alors la solution optimale divisée par la solution obtenue est inférieure à $1 + \frac{1}{k}$ (on l'admettra mais en question subsidiaire vous pouvez le montrer en vous inspirant de la preuve précédente).

6) En déduire un schéma d'approximation polynomial. Ce schéma est-il totalement polynomial ?

4.3 Novembre 2001

4.3.1 Complexité concrète

- 1** : Peut-on vraiment dire qu'un algorithme en n^{100} est meilleur qu'un algorithme en $1,0000001^n$?
- 2** : Quelles sont à votre avis les retombées pratiques de la question $P \neq NP$?
- 3** : Les classes de complexité probabilistes vous paraissent-elles plus raisonnables ?

4.3.2 Une nouvelle machine de Turing

On appelle Machine de Turing Non-déterministe Normande (MTNN), une machine de Turing à trois états finaux : OUI , NON et P'TETBENQU'OUI (PBQ pour les intimes).

Nous dirons qu'un calcul est accepté par une telle machine pour une donnée I , si tous les calculs possibles se terminent dans les états OUI et PBQ et qu'il existe au moins un calcul se terminant en OUI.

Similairement un calcul est refusé, si tous les calculs donnent NON et PBQ et au moins un se termine sur l'état NON.

Montrer qu'un Langage L est reconnu par une MTNN ssi $L \in NP \cap co-NP$.

4.3.3 Réductions

1 : **Nom** (*Plus long cycle élémentaire*)

Données (*Un graphe $G = (X, E)$ non orienté, k un entier*)

Question (*G admet-il un cycle élémentaire de taille supérieure ou égale à k ?*)

Montrer que ce problème est dans NP ?

Rappel : le mot élémentaire pour un cycle, une chaîne, un chemin ou un circuit veut dire passer au plus une fois par chaque sommet.

2 Montrer que ce problème de décision est NP-complet.

3 Que devient la complexité du problème si l'on remplace cycle par chaîne ?

4 : On considère maintenant le problème du plus long chemin élémentaire valué.

Nom (*Plus long chemin*)

Données (*Un graphe $G = (X, U)$ un graphe orienté, $\mu : U \rightarrow Z$ une valuation des arcs, $k \in Z$*)

Question (*Existe-t-il un chemin élémentaire dans G dont la valuation soit supérieure ou égale à k ?*)

Montrer que ce problème est NP-complet ?

Piste : Pour toutes ces questions on peut utiliser la NP-complétude du problème du voyageur de commerce.

4.3.4 Encore SAT

1. Montrer que $3 - SAT \leq_K Pas - tous - egaux - SAT$.

Pour ce faire on transformera toute clause à trois éléments en deux clauses en ajoutant une nouvelle variable.

2. Montrer que $3 - SAT \leq_K Un - sur - trois - SAT$.

Dans cette variante de SAT, un seul littéral doit être vrai par clause. Dans ce cas on remplacera toute clause par trois clauses en ajoutant 4 nouvelles variables.

4.3.5 Des problèmes de rectangles *

Montrer que les deux problèmes suivants sont NP-difficiles, on peut s'inspirer du problème partition. Ces problèmes sont issus du parallélisme et des algorithmes parallèles de multiplication de matrices.

Nom (*Périmètre*)

Données (*Etant donné p nombres réels s_1, \dots, s_p , t.q. : $\sum_{i=1}^p s_i = 1$*)

Recherche (*Trouver une partition du carré unité en p rectangles de surface s_i et de côtés v_i et h_i telle que $\sum_{i=1}^p (h_i + v_i)$ soit minimale.*)

Nom (*Périmètre Maximum*)

Données (*Etant donné p nombres réels s_1, \dots, s_p , t.q. : $\sum_{i=1}^p s_i = 1$*)

Recherche (*Trouver une partition du carré unité en p rectangles de surface s_i et de côtés v_i et h_i telle que $\text{Max}_{1 \leq i \leq p} (h_i + v_i)$ soit minimale.*)

4.3.6 Comme promis ... ***

On considère le problème suivant :

Nom (*Sac-à-dos sur graphe*)

Données (*Un graphe orienté sans circuit $G = (X, U)$ et une valuation π à valeur dans Z de ces sommets, k un entier relatif.*)

Question (*Existe-t-il un sous ensemble héréditaire $S \subseteq X$ tel que $\sum_{x \in S} \pi(x) \geq k$?*)

Nous avons vu en cours une transformation valable lorsque $G(S)$ est connexe. Montrer que ce problème est NP-complet.

4.3.7 Commentaire final

*Lorsque le résultat d'une question est clairement énoncé, on peut l'admettre si nécessaire dans les questions suivantes. Comme convenu, j'accepterai des réponses pertinentes aux questions marquées d'une ou plusieurs * jusqu'au mercredi 21 novembre 2001 à 14h GMT.*

Un algorithme approché optimal pour le problème du k -centre

Le problème du k -centre.

Données : Un entier k , un ensemble de villes $V = \{v_1, v_2, \dots, v_n\}$ et une matrice D où d_{ij} représente la distance entre la ville i et la ville j (on supposera aussi que D respecte l'inégalité triangulaire c'est-à-dire que $\forall i, j, k$, on a $d_{ij} \leq d_{ik} + d_{kj}$). Pour simplifier, on supposera que D est symétrique.

But : Trouver un ensemble S , $S \subseteq V$, tel que $|S| = k$ qui minimise $\max_{j \in V}(d_{Sj}) = \max_{j \in V}(\min_{i \in S}(d_{ij}))$.

A chaque ensemble S de k villes, on associe la fonction f dans N : $f(S) = \max_{j \in V} d_{Sj}$. Donc le but est de trouver S^* qui minimise f , on notera $d^* = f(S^*)$.

a *) Montrer que $\forall \epsilon > 0$, il n'existe pas d'algorithme polynomial $(2 - \epsilon)$ -approché pour ce problème (sous les hypothèses que $P \neq NP$). Indication : le problème de l'existence d'une couverture de taille k dans un graphe $G=(V,E)$ est un problème NP-complet (rappel : une couverture C est un sous-ensemble de V qui intersecte toutes les arêtes de G : $\forall e \in E, e \cap C \neq \emptyset$).

Dans la suite, on se propose de montrer qu'il existe un algorithme polynomial 2-approché pour le problème du k -centre. On appellera H_d le graphe construit de la façon suivante $H_d = (V, E_d)$ où $E_d = \{\{v_i, v_j\} \mid d_{ij} \leq d\}$. Les questions b, c, et d sont indépendantes.

b) Que peut-on dire s'il existe un dominant de taille k dans H_d ? (un dominant est un sous-ensemble D de sommets tel que $\forall y \in V, \exists x \in D$ tels que $\{x, y\} \in E_d$).

c *) Montrer que si H_{2d} a un stable de taille strictement supérieure à k alors H_d ne possède pas de dominant de taille k .

d) Montrer qu'un stable maximal d'un graphe est un dominant. Rappel : un stable S est dit maximal s'il n'est pas strictement inclus dans un autre stable, en particulier on peut trouver en temps polynomial un stable maximal contrairement à un stable de cardinalité maximum (si $P \neq NP$).

e **) En déduire un algorithme polynomial 2-approché pour le problème du k -centre. On peut évidemment admettre les résultats précédents pour effectuer cette question.

4.4 Novembre 2002

Les parties sont indépendantes. Le barème dépendra de ce que vous aurez fait. Nous avons indiqué une estimation de la difficulté devant chaque question. aucune marque = facile, c'est presque du cours, * = faut avoir une idée, ** c'est difficile.

4.4.1 Complexité concrète

- 1-facile** : Montrer qu'un programme qui fait appel un nombre constant d'appels à des sous-programmes dont le temps d'exécution est polynomial est lui-même polynomial.
- 2** : Que peut-on dire d'un programme qui fait appel un nombre polynomial d'appels à des sous-programmes dont le temps d'exécution est polynomial ?
- 3** : Que peut-on dire sur les relations entre les deux conjectures suivantes :
- $P \neq NP$
 - $NP \neq co - NP$
- Enfinement que pensez vous : $P = NP$ ou $P \neq NP$? Justifier votre intuition.

4.4.2 Quelques Réductions

1 : Nom (*Chemin passant par K*)

Données (*Un graphe $G = (X, U)$ orienté dont les arcs sont valués par $\omega : U \rightarrow Z$, K un entier et deux sommets s et p)*

Question (*G admet-il un chemin de s à p passant par K ?*)

Un chemin $\mu = [s = x_0, x_1, x_2, \dots, x_h = p]$ de G passe par K , s'il existe $i \in [1, h]$ tel que : $\sum_{j=0}^{j=i} \omega(x_j x_{j+1}) = K$

Montrer que ce problème est dans NP ?

- 2** : Montrer que ce problème de décision est NP-complet. Piste : on peut construire une réduction à partir de Somme Exacte.

3 : Que devient le problème si l'on impose que la donnée soit un graphe sans circuit ?

4 Etudions une variante du problème précédent :

Nom (*Chemin évitant K*)

Données (*Un graphe $G = (X, U)$ orienté dont les arcs sont valués par $\omega : U \rightarrow Z$, K un entier et deux sommets s et p)*

Question (*G admet-il un chemin de s à p évitant K ?*)

Un chemin $\mu = [s = x_0, x_1, x_2, \dots, x_h = p]$ de G évite K , si $\forall i \in [1, h]$

$\sum_{j=0}^{j=i} \omega(x_j x_{j+1}) \neq K$

Ce problème est-il dans NP ou co-NP ?

5 : A-t-on Chemin passant par $K = \text{co}(\text{Chemin évitant } K)$?

6-** : Est-il polynomial, NP-complet, co-NP-complet, NP-difficile ?

4.4.3 Encore des réductions

1 : **Nom** (*Partition d'ensembles*)

Données (*Une famille C de sous-ensembles d'un ensemble X*)

Question (*Existe-t-il une partition de X en deux sous-ensembles X_1 et X_2 telle qu'aucun ensemble de C ne soit inclus dans X_1 ou X_2 ?*)

Montrer que ce problème est NP-complet.

2 : **Nom** (*Fonction monotone*)

Données (*Un ensemble fini X , une famille C de triplets ordonnés (a, b, c) d'éléments de X*)

Question (*Existe-t-il une fonction $f : X \rightarrow \{1, 2, \dots, |X|\}$ de X telle que pour tout (a, b, c) on ait soit $f(a) < f(b) < f(c)$ soit $f(c) < f(b) < f(a)$?*)

Montrer que ce problème est NP-complet. On peut utiliser la question précédente.

4.4.4 Rituel ... **

On considère le problème suivant :

Nom (*Sac-à-dos ordonné*)

Données (*Un graphe orienté sans circuit $G = (X, U)$ et une valuation π à valeur dans Z de ces sommets, k un entier relatif.)*

Question (*Existe-t-il un sous ensemble héréditaire $S \subseteq X$ tel que $\sum_{x \in S} \pi(x) \geq k$?*)

Nous avons vu en cours une transformation valable lorsque $G(S)$ est connexe. Montrer que ce problème est NP-complet.

4.4.5 Approximation : un problème de rangement

Soient n objets à ranger dans un nombre minimum de boîtes (une boîte est utilisée si on y range au moins un objet). Chaque objet i , $1 \leq i \leq n$, est caractérisé par sa hauteur h_i et on supposera que pour tout i , h_i est un réel compris entre 0 et 1. Les boîtes ont toutes une hauteur de 1, c'est-à-dire que si B_j est l'ensemble des objets mis dans la boîte j on a $\sum_{i \in B_j} h_i \leq 1$.

On utilise la stratégie suivante : on prend les objets dans un ordre arbitraire. Soient $B_1, B_2 \dots B_k$ les boîtes déjà utilisées lorsque l'on range le i^{eme} objet. On essaie de mettre l'objet i dans la boîte B_1 et en cas d'échec (boîte trop remplie) dans la boîte $B_2 \dots$ jusqu'à la boîte k . Si nécessaire (échec dans les k boîtes), on prend une $(k+1)^{\text{eme}}$ boîte pour mettre l'objet.

a) Montrer que cet algorithme est 2-approché (indication : montrer que toutes les boîtes sauf éventuellement une sont remplies plus qu'à moitié).

b) Donner un exemple où l'algorithme donne une solution à $\frac{5}{3}$ de l'optimum.

c) Montrer que $\forall \epsilon > 0$, il n'existe pas d'algorithme $(\frac{3}{2} - \epsilon)$ -approché qui s'exécute en temps polynomial par rapport à n .

d) Montrer que si $\forall i, h_i > \frac{1}{3}$ alors le problème devient facile (il existe un algorithme polynomial qui donne toujours la meilleure solution).

e) On reprend maintenant l'algorithme initial mais en prenant les objets en ordre décroissant de hauteur. Quel résultat obtient-on avec l'exemple b) ? Proposer un autre exemple tel qu'en prenant les objets en ordre décroissant de hauteur, on arrive à $\frac{3}{2}$ de l'optimum.

** Question subsidiaire : montrer que l'algorithme est ρ -approché avec $\rho < 2$.

4.4.6 Commentaire final

Lorsque le résultat d'une question est clairement énoncé, on peut l'admettre si nécessaire dans les questions suivantes. Comme convenu, j'accep-

terai des réponses pertinentes aux questions marquées d'une ou plusieurs * jusqu'au vendredi 15 novembre 2002 à 18h GMT.

4.5 Novembre 2003

4.5.1 Compréhension générale des théories

1. Que peut-on dire d'une théorie pour laquelle un algorithme dont la complexité est $\theta(n^{1000})$ est "meilleur" qu'un algorithme en $\theta(2^{\alpha(n)})$ où α représente l'inverse de la fonction d'Ackermann (i.e. une fonction qui croît extrêmement lentement ainsi pour $n = 1$ milliard, $\alpha(n)$ vaut moins de 10) ?
2. Peut-on dire que tout programme (resp. tout programme d'optimisation) tourne en $O(1)$?
3. Que pensez-vous de la conjecture $NP \cap co-NP = P$?
4. Démontrez $NP^P = NP$ (égalité affirmée sur un transparent lors de la présentation de la hiérarchie polynomiale)
5. Expliquer la contradiction apparente entre :
 $NP^P = NP$
 et l'exemple d'un programme à la complexité exponentielle utilisant un nombre polynomial d'appels d'une fonction polynomiale.

4.5.2 Réductions

1. Montrer que le problème de l'isomorphisme de graphes non-orientés est polynomialement équivalent au problème de l'isomorphisme de graphes bipartis ainsi qu'à celui de l'isomorphisme de graphes orientés sans circuits.
2. **Réductions parcimonieuses :**
 Rappelons qu'une réduction parcimonieuse est une réduction polynomiale qui préserve le nombre de solutions.
 Exhiber 3 réductions parcimonieuses vues en cours. Peut-on écrire une réduction parcimonieuse en SAT et 3-SAT ?
3. **Variations sur un problème :**
Données (Un ensemble fini X et un ensemble de triplets $M = \{(a_i, b_i, c_i) \mid a_i, b_i, c_i \in X\}$)

Question (*Existe-t-il un ordre σ sur X , dans lequel c_i est toujours le dernier du triplet et ceci pour $\forall i$?*)

Montrer que ce problème est dans NP. Est-il polynomial ou NP-complet ?

4. **Données** (*Un ensemble fini X et un ensemble de triplets $M = \{(a_i, b_i, c_i) \text{ t.q. } a_i, b_i, c_i \in X\}$*)

Question (*Existe-t-il un ordre σ sur X , dans lequel c_i n'est jamais le dernier du triplet et ceci pour $\forall i$?*)

Mêmes questions que précédemment.

5. **Données** (*Un ensemble fini X et un ensemble de triplets $M = \{(a_i, b_i, c_i) \text{ t.q. } a_i, b_i, c_i \in X\}$*)

6. **Données** (*Un ensemble fini X et un ensemble de triplets $M = \{(a_i, b_i, c_i) \text{ t.q. } a_i, b_i, c_i \in X\}$*)

Question (*Existe-t-il un ordre σ sur X , dans lequel c_i n'est jamais le dernier du triplet et ceci pour $\forall i$?*)

Mêmes questions que précédemment.

Question (*Existe-t-il un ordre σ sur X , dans lequel c_i n'est jamais le premier ni le dernier du triplet et ceci pour $\forall i$?*)

Mêmes questions que précédemment.

7. **

Nom (*Chemin évitant K*)

Données (*Un graphe $G = (X, U)$ orienté dont les arcs sont valués par $\omega : U \rightarrow Z$, K un entier et deux sommets s et p*)

Question (*G admet-il un chemin de s à p évitant K ?*)

Un chemin $\mu = [s = x_0, x_1, x_2, \dots, x_h = p]$ de G évite K , si $\forall i \in [1, h]$

$\sum_{j=0}^{j=i} \omega(x_j x_{j+1}) \neq K$.

Est-ce polynomial ou NP-complet ?

4.6 Novembre 2004

4.6.1 Applications des théories de complexité

Il a été récemment démontré (2002) que le problème de la reconnaissance de la primalité d'un nombre est polynomial.

Que pensez-vous du problème de la factorisation, i.e. le problème de recherche qui calcule les facteurs premiers d'un nombre entier ? Ce problème est

très important pour l'algorithme de cryptage RSA, pensez-vous qu'il existe un algorithme polynomial ? justifiez votre réponse.

Dans le même genre d'idée, que pensez-vous de l'existence d'une fonction de codage sûre (une fonction à sens unique, i.e. polynomiale dont l'inverse est NP-difficile) ?

4.6.2 Recouvrement versus partition

1. On considère le problème

Nom : Partition d'ensembles

Données : X un ensemble fini, S_1, \dots, S_n des sous ensembles de X tels que $\cup S_i = X$

Question : Existe-t-il $I \subseteq [1, n]$ tel que $\{S_i | i \in I\}$ soit une partition de X ?

En gros à partir d'une famille recouvrante peut-on extraire une partition ?

Montrer que ce problème est dans NP.

2. Monter que le problème est NP-complet. Piste : on peut s'inspirer du problème 3DM.

4.6.3 Renommage

1. On appelle clause de Horn une clause dont au plus un littéral est positif. Montrer que Horn-SAT est polynomial. Piste : on peut commencer par considérer les clauses à une variable ...
2. On appelle renommage de variables un ensemble de changements de littéraux du type x en \bar{x} . Est-il possible de reconnaître si une formule appartient à la classe Horn-SAT à un renommage des variables près ?
3. * On appelle clause de Horn définie positive une clause dont exactement un littéral est positif. Reprendre les deux questions précédentes sur cette nouvelle classe de formules.
4. * On considère maintenant une instance de 3-SAT, ayant m clauses. Montrer qu'il existe toujours une solution satisfaisant au moins $\lceil m/2 \rceil$ clauses.

Ecrire un algorithme dont la complexité dépend de k , qui répond à la question suivante :

une instance de 3-SAT ayant m clauses, possède une solution satisfaisant au moins $\lceil m/2 \rceil + k$ clauses.

5. *

On considère un ensemble fini X et un ensemble de triplets (u, v, w) d'éléments de X . La question de savoir s'il existe un ordre total sur X tel que chaque triplet vérifie $u < v < w$ ou $w < v < u$ est un problème NP-complet.

Le problème reste-t-il NP-complet, si pour chaque $x \in X$, il existe au plus un triplet centré en x ?

||

Bibliographie

- [1] M. Agrawal, N. Kayal, and N. Saxena. Primes is in p. 2002.
- [2] B. Aspvall, M. F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 1979.
- [3] G. Ausiello, P. Crezenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation : combinatorial optimization problems and their approximability properties*. Springer-Verlag, 1999.
- [4] D.P. Bovet and P. Crescenzi. *Introduction to the theory of complexity*. Prentice hall, 1994.
- [5] G.B. Dantzig. *Linear Programming and extensions*. Princeton University Press, 1962.
- [6] R.G. Downey and M.R. Fellows. *Parametrized Complexity*. Monographs in Computer Science. Springer Verlag, 1999.
- [7] O. Dubois. On the r,s-sat satisfiability problem and a conjecture of tovey. *Discrete Applied Mathematics*, 1990.
- [8] J. Edmonds. Paths, trees and flowers. *Canad. J. Math.*, 17 :449–467, 1965.
- [9] M. Fredman and L.G. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. of Algorithms*, 21 :618–628, 1996.
- [10] H.N. Gabow, S. N. Maheshwari, and L.J. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, 1976.
- [11] M.R. Garey and D.J. Johnson. *Computers and intractability : a guide to the theory of NP-completeness*. Freeman, 1979.

- [12] M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed. *Probabilistic methods for algorithmic discrete mathematics*. Springer-verlag, 1999.
- [13] L.G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Dokl.*, 20 :191–194, 1979.
- [14] C.H. Papadimitriou. *Computational complexity*. Addison-Welsey, 1994.
- [15] V.R. Pratt. Every prime have a succinct certificate. *SIAM J. on Computing*, (4) :214–220, 1975.
- [16] H. Rogers. *Theory of recursive functions and effective computability*. McGraw-Hill, 1967.
- [17] T.J. Schaefer. The complexity of satisfiability problems. In *ACM Symp. Theory of Computing*, 1978.
- [18] C.A. Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 1984.