ORSAY
N° d'ordre : 5499

UNIVERSITÉ DE PARIS-SUD
U.F.R. SCIENTIFIQUE D'ORSAY

THÈSE

présentée
pour obtenir

Le GRADE de DOCTEUR en SCIENCES
DE L'UNIVERSITÉ PARIS XI ORSAY

Spécialité : INFORMATIQUE

par

CRISTINA BAZGAN

Sujet : **Approximation de problèmes d'optimisation et de fonctions totales de NP**

Soutenue le 27 novembre 1998 devant la Commission d'examen

M. Jean-Pierre Jouannaud	Président
M. Wenceslas Fernandez de la Vega	
M. Jean Fonlupt	Rapporteur
M. Marek Karpinski	Rapporteur
M. Miklos Santha	Directeur de thèse
M. Zsolt Tuza	

À mes parents et Irina,

Remerciements

Je remercie Miklos Santha d'avoir eu l'amabilité de diriger mes recherches ainsi que pour ses conseils.

Je remercie tout particulièrement Jean-Pierre Jouannaud d'avoir accepté d'être le président du jury de cette thèse.

Jean Fonlupt a très gentillement accepté d'être un des rapporteurs. Je le remercie pour sa rigueur scientifique et ses nombreuses suggestions.

Je remercie beaucoup Marek Karpinski pour son intérêt pour mon travail, pour les nombreuses discutions et suggestions tout au long de ma thèse et pour avoir accepté d'être un des rapporteurs.

Un grand merci à Zsolt Tuza d'avoir eu la gentillesse de participer au jury et d'avoir travaillé avec moi malgré l'éloignement géographique.

Je remercie Wenceslas Fernandez de la Vega d'avoir accepté d'être dans mon jury et pour les moments où nous avons cherché ensemble.

Je tiens à remercier Michel de Rougemont de m'avoir donné la chance de venir étudier en France.

Je remercie chaleureusement les autres personnes avec qui j'ai eu le grand plaisir de travailler : Amel Harkat-Benhamdine, Jan van den Hauvel, Claire Kenyon, Hao Li, Antoni Lozano, Mariusz Woźniak.

Je tiens à remercier un de mes professeurs de Roumanie, Cornelius Croitoru, qui a été l'initiateur de mes recherches et qui a toujours été de bon conseil.

Je remercie les personnes qui ont eu l'amabilité et le courage de corriger des parties de ma thèse : Ronan Calippe, David Gross, Sophie Laplante, Frédéric Magniez, Xavier Urbain. Je les remercie pour leur rigueur et leur amitié.

Je remercie particulièrement les personnes qui m'ont aidé à obtenir un financement pour ma thèse : Michel Beaudouin-Lafon, Marie-Claude Gaudel, Dominique Gouyou-Beauchamps, Laurence Puel et les autres personnes non citées. Ce travail a aussi été soutenu par les projets ESPRIT RAND-REC et RAND2.

Je remercie les membres de l'équipe *Algorithmique et Complexité* et les autres amis du L.R.I. pour les moments agréables passés ensemble.

Mulţumesc parinţilor mei pentru dragostea si entuziasmul lor permanent. Mulţumesc Irinei pentru suportul ei continuu. Merci à Ronan pour sa patience et sa gentillesse.

Table des matières

Table des figures

Chapitre 1

Introduction

L'étude de la complexité de l'approximation des problèmes d'optimisation est un domaine de l'informatique théorique bien étudié et en continuel développement.

Un problème d'optimisation est caractérisé par un ensemble d'*instances*, chaque instance ayant un nombre fini de *solutions possibles*. Chaque solution a une *valeur* associée et le but est de trouver pour chaque instance une solution optimum (de valeur maximum ou minimum). Par exemple, une instance du problème TRAVELING SALESMAN[1] consiste en un ensemble de n villes $\{v_1, \ldots, v_n\}$ et pour chaque couple de villes (v_i, v_j) est donnée la distance entre ces deux villes $d(v_i, v_j)$. Une solution possible est un chemin qui part d'une ville, qui passe par toutes les villes exactement une fois et qui retourne dans la ville de départ. Le but est de trouver un tel chemin de longueur minimum.

En général on souhaite trouver l'algorithme le plus « efficace » pour résoudre un problème d'optimisation. La notion d'efficacité concerne toutes les ressources nécessaires pour l'exécution de l'algorithme. Dans cette thèse nous allons juger l'efficacité d'un algorithme qui *résout* un problème d'optimisation en fonction de son temps d'exécution car les demandes en temps sont souvent un facteur déterminant pour qu'un algorithme soit utilisable. Le temps d'exécution d'un algorithme est en général exprimé en fonction de la taille de l'instance du problème qui reflète la quantité de données nécessaires pour décrire l'instance.

Le nombre fini de solutions pour chaque instance d'un problème d'optimisation suggère un algorithme simple pour calculer une solution optimum : on calcule la valeur de chaque solution possible et on considère comme solution celle avec la meilleure valeur. Malheureusement en général le nombre de solutions est très grand, il est bien souvent exponentiel par rapport à la taille de l'entrée. Par exemple, pour une instance du problème TRAVELING SALESMAN avec n villes, il y a $n!$ solutions possibles. Néanmoins, pour certains problèmes il y a des algorithmes qui trouvent une solution optimum *efficacement*, c'est-à-dire en temps

1. Le nom des problèmes utilisés dans cette thèse sera en anglais.

polynomial en la taille de l'instance, même si le nombre de solutions est exponentiel : par exemple, les algorithmes de Belman et Dijkstra pour trouver le plus court chemin dans un graphe ou l'algorithme de Kruskal pour trouver un arbre couvrant de coût minimum dans un graphe. Mais pour plusieurs problèmes d'optimisation intéressants il n'est pas possible de trouver une solution optimum efficacement. Ces problèmes sont appelés des problèmes *intraitables* et la théorie de la NP-complétude due à Cook et Levin fournit une théorie expliquant ce phénomène de la façon suivante :

NP est la classe des problèmes de décision pour lesquels on peut vérifier en temps polynomial la preuve qu'une instance a une solution. Un problème est dans P s'il est soluble en temps polynomial. Une des questions centrales ouvertes de la théorie de la complexité est de savoir si $P \neq NP$. La conjecture $P \neq NP$ est largement acceptée et il y a beaucoup de résultats qui sont montrés sous cette hypothèse. La théorie de la NP-complétude nous permet de montrer qu'un problème d'optimisation spécifique n'est pas soluble en temps polynomial, en supposant que $P \neq NP$. L'idée est de montrer que le problème de décision associé est *NP-difficile*, c'est-à-dire au moins aussi difficile que tout problème de NP. Les problèmes NP-difficiles sont considérés comme étant intraitable.

C'est la raison pour laquelle quand on est intéressé par la résolution d'un problème intraitable, on est motivé par l'exploration de possibilités moins ambitieuses que de *résoudre* le problème en temps polynomial. Une méthode pour attaquer un tel problème d'optimisation est d'étudier le comportement de certaines heuristiques. Une autre méthode est d'essayer de donner des algorithmes polynomiaux qui retournent des solutions proches de l'optimum. Nous nous intéressons dans cette thèse à cette dernière méthode.

Approcher un problème d'optimisation en temps polynomial à un facteur $f(n)$ près veut dire calculer pour chaque instance de taille n du problème en temps polynomial une solution avec une valeur à un facteur multiplicatif $f(n)$ près de la valeur optimum. Le but pour un problème d'optimisation est de trouver le facteur $f(n)$ le plus proche de 1 tel que le problème soit approximable en temps polynomial à un facteur $f(n)$ près. Un problème de minimisation a un schéma d'approximation en temps polynomial s'il existe un algorithme polynomial qui est une approximation à un facteur $(1+\varepsilon)$ près pour ce problème, pour toute constante $\varepsilon > 0$. La définition d'un schéma d'approximation pour un problème de maximisation est symétrique. Évidemment pour un problème d'optimisation qui a un schéma d'approximation on veut trouver le schéma le plus efficace. Si le temps de l'algorithme est le produit entre un polynôme en n et une fonction en $1/\varepsilon$ alors on dit que le problème a un schéma d'approximation en temps polynomial efficace. Et si le temps de l'algorithme est aussi polynomial en $1/\varepsilon$ alors on dit que le problème a un schéma d'approximation en temps complètement polynomial. Pour certains problèmes d'optimisation NP-difficiles on sait comment on peut trouver des solutions à un certain facteur près, mais pour d'autres, il est NP-

difficile de trouver une solution approchée. Il est aussi très intéressant d'étudier et de comprendre quels sont les facteurs d'approximation difficiles à réaliser pour un problème d'optimisation donné.

Ladner [L75] a montré que sous l'hypothèse où $P \neq NP$, il existe des problèmes qui ne sont ni polynomiaux ni NP-complets. La question de l'approximation est aussi intéressante pour ces problèmes pour lesquels aucun algorithme polynomial n'est connu. Pour un problème de décision, pour chaque instance on s'intéresse à l'existence d'une solution. En revanche, pour un problème de recherche associé à un problème de NP, on souhaite *trouver* une solution pour chaque instance si une telle solution existe. Dans cette thèse nous nous sommes particulièrement intéressés aux problèmes de la classe $TFNP$ définie par Meggido et Papadimitriou [MP91] qui contient les problèmes de recherche associés aux problèmes de NP, dont chaque instance du problème a une solution. Des exemples de problèmes de cette classe appelés des problèmes totaux, sont : FACTORING, SECOND HAMILTONIAN CYCLE, PIGEONHOLE SUBSET-SUM, ANOTHER HAMILTONIAN PATH et HAPPYNET.

La classe $TFNP$ est située entre FP et FNP, les classes des problèmes de recherche correspondant à P et NP. Sous l'hypothèse où $P \neq NP \cap co-NP$, il existe un problème de $TFNP$ qui n'est pas soluble en temps polynomial [MP91]. Une propriété intéressante des problèmes totaux est qu'ils ne peuvent pas être FNP-difficiles sauf si $NP = co\text{-}NP$ [MP91]. Pourtant, aucun algorithme polynomial n'est connu pour les exemples précédents.

Dans la présente thèse nous considérons, pour la première fois à notre connaissance, l'approximation de certains problèmes de $TFNP$. Les problèmes étudiés sont parmi les problèmes ci-dessus mentionnés.

Les chapitres 2 et 3 contiennent des notions générales sur l'approximation et la totalité et les chapitres 4, 5 et 6 contiennent les résultats originaux de cette thèse.

Dans le chapitre 2 nous donnons les définitions, les concepts fondamentaux et les principaux résultats concernant l'approximation des problèmes d'optimisation. La dernière section de ce chapitre contient quelques résultats d'approximation sur des instances particulières des problèmes d'optimisation : instances planaires, graphes cubiques, instances denses.

Le chapitre 3 contient certains résultats connus sur les problèmes totaux et classes de problèmes totaux qui ont été définies par Megiddo et Papadimitriou ainsi que quelques exemples.

Le chapitre 4 est basé sur [BST98b], un travail fait en commun avec M. Santha et Zs. Tuza. Dans ce chapitre, nous étudions le problème de trouver deux sous-ensembles disjoints et non vides d'un ensemble de n entiers positifs tels que les sommes des entiers des deux sous-ensembles soient les plus proches possibles. Ce problème a une variante totale intéressante quand la somme des n entiers est au plus 2^n-1. Il est facile de voir, en utilisant le principe des tiroirs, que dans ce cas-là il existe toujours deux sous-ensembles différents avec la même somme, mais il n'y a pas d'algorithme polynomial connu pour trouver deux tels sous-ensembles. En répondant à une question de Woeginger et Yu [WY92] de manière positive, nous avons donné un schéma d'approximation en temps complètement polynomial pour la version du problème où la valeur d'une solution est le rapport entre les sommes d'entiers dans les deux sous-ensembles, obtenant ainsi le meilleur résultat possible pour ce problème. Nous avons voulu montrer que la variante totale précédente est plus facile à approcher que le problème général. Pour cela, nous avons défini une autre version du problème où la valeur d'une solution est la différence entre les sommes des entiers des deux sous-ensembles. Pour cette nouvelle version, le problème général n'est pas approximable à un facteur 2^{n^k} près pour chaque constante k sauf si $P = NP$ et la variante totale est approximable à un facteur $\frac{2^n}{n^{\Omega(\log n)}}$ près.

Le chapitre 5 contient quelques résultats d'approximation que nous avons obtenus pour des problèmes qui font partie d'une autre classe de problèmes totaux. Dans une première partie nous nous sommes intéressés à un problème pour lequel la totalité est montrée par un théorème de Lovász d'après lequel la somme du nombre des chemins Hamiltoniens d'un graphe G et du nombre des chemins Hamiltoniens de son complément $\bar{G}$ est un nombre pair. Nous considérons le problème total suivant : étant donné un chemin Hamiltonien dans un graphe G, trouver un deuxième dans G ou dans $\bar{G}$. Nous donnons pour ce problème une approximation à un terme additif 1 près en temps polynomial. Dans une deuxième partie qui est basée essentiellement sur un travail réalisé en collaboration avec M. Santha et Zs. Tuza [BST98a], nous nous intéressons au problème du plus long cycle dans les graphes cubiques et Hamiltoniens. Smith ([T46]) a montré que chaque graphe cubique et Hamiltonien a au moins deux cycles Hamiltoniens. Nous avons étudié la complexité de l'approximation de ce problème où une solution possible est un deuxième cycle dans le graphe et la valeur de la solution est mesurée par la longueur du cycle. Nous montrons qu'il existe un schéma d'approximation en temps polynomial efficace pour ce dernier problème.

Dans le chapitre 6 qui est basé essentiellement sur [BST98a], nous nous intéressons aussi au problème du plus long cycle dans les graphes cubiques et Hamiltoniens. Trouver un tel cycle est un problème NP-difficile. Karger, Motwani et Ramkumar [KMR93] ont laissé comme problème ouvert la recherche de la complexité de l'approximation du problème du plus long chemin dans un graphe Hamiltonien. Motivés par leur article, nous prouvons un résultat négatif qui montre que le problème du plus long cycle (et plus long chemin) n'est pas approximable à une constante près dans les graphes cubiques et Hamiltoniens à moins que $P = NP$.

Dans le chapitre 7, nous discutons certaines perspectives envisageables.

L'annexe A contient la liste des problèmes d'optimisation utilisés dans ce document.

Chapitre 2

Approximation en temps polynomial

2.1 Problème d'optimisation

Un langage L est un sous-ensemble de chaînes de longueur finie sur un alphabet fini Σ, c'est-à-dire $L \subseteq \Sigma^*$.

Un langage est dans la classe NP s'il est accepté par une machine de Turing non-déterministe qui fonctionne en temps polynomial. On considère que les entrées sont représentées par des chaînes binaires. La longueur d'une chaîne binaire x est notée $|x|$. La taille de l'entrée est la longueur de sa représentation binaire. On utilise aussi une définition (équivalente) d'un langage de NP basée sur des relations binaires [Pap94a, Proposition 9.1, p. 181].

Définition 2.1.1 (Relation polynomiale) *Une relation binaire $R \subseteq \Sigma^* \times \Sigma^*$ est une* relation polynomiale *si pour tout x, y :*

1. *Le problème de « décider si $R(x, y)$ » est soluble en temps polynomial en fonction de $|x| + |y|$.*
2. *Il existe un polynôme p tel que si $R(x, y)$ alors $|y| \leq p(|x|)$.*

Définition 2.1.2 (NP) *Un langage $L \in NP$ si et seulement si il existe une relation polynomiale R telle que $L = \{x :$ il existe un y tel que $R(x, y)\}$.*

Étant donné un langage L, une chaîne $x \in \Sigma^*$ est appelée une *instance positive* si $x \in L$ et *négative* si $x \notin L$.

Ainsi, pour un langage L de NP, chaque instance positive x a au moins un *témoin court* y qui montre que $x \in L$.

Un langage $L \in co\text{-}NP$ si son complément $\bar{L} = \Sigma^* \setminus L$ est dans NP. Pour un langage de $co\text{-}NP$ chaque instance négative x a donc au moins un *témoin court* y qui montre que $x \notin L$.

Un langage $L \in NP \cap co\text{-}NP$ s'il existe deux relations polynomiales R_1, R_2 telles que : si $x \in L$ alors il existe un y tel que $R_1(x, y)$ et si $x \notin L$ alors il existe un z tel que $R_2(x, z)$. Donc pour un langage $L \in NP \cap co\text{-}NP$, chaque instance a soit un témoin court qui montre qu'elle est une instance positive soit un témoin court qui montre qu'elle est une instance négative. Un exemple bien connu d'un langage de $NP \cap co\text{-}NP$ est l'ensemble des entiers premiers.

Étant donnés deux langages $L_1 \subseteq \Sigma_1^*$ et $L_2 \subseteq \Sigma_2^*$, le langage L_1 est *Karp-réductible* à L_2 s'il existe une fonction polynomiale f qui transforme toute $x \in \Sigma_1^*$ en $f(x) \in \Sigma_2^*$ tel que $x \in L_1$ si et seulement si $f(x) \in L_2$.

Un langage L est *NP-difficile* si tout langage de NP est Karp-réductible à L. Un langage L est *NP-complet* si $L \in NP$ et L est NP-difficile.

Pour définir un problème d'optimisation, il est nécessaire de spécifier un ensemble d'instances, un ensemble de solutions possibles pour chaque instance (c'est-à-dire qui satisfont certaines contraintes) et une fonction de coût sur cet ensemble de solutions. On cherche à trouver un optimum (maximum ou minimum) pour cette fonction de coût.

Définition 2.1.3 (Problème d'optimisation) *Un problème d'optimisation est un quadruplet (I, Sol, val, but) où :*

1. *I est l'ensemble des instances ;*
2. *Pour chaque instance $x \in I$, $Sol(x)$ est l'ensemble de solutions possibles ;*
3. *La fonction « val » associe à chaque solution possible une valeur ;*
4. *Il s'agit soit de maximiser (but=max), soit de minimiser (but=min) la fonction de coût « val ».*

Le but d'un problème d'optimisation A sur une instance x est de trouver une *solution optimum*, c'est-à-dire un élément $y^* \in Sol(x)$ tel que :

$$val(x, y^*) = but\{val(x, y) : y \in Sol(x)\}.$$

Dans la suite, la valeur de la fonction val d'une solution optimum d'un problème A sur une instance x est notée $opt_A(x)$.

Définition 2.1.4 (NPO) *Un problème d'optimisation est dans la classe NPO si :*

1. *Il existe un polynôme p tel que pour chaque instance x et pour toute solution $y \in Sol(x)$, $|y| \leq p(|x|)$;*
2. *Pour tout x et tout y avec $|y| \leq p(|x|)$, le problème de « décider si $y \in Sol(x)$ » est soluble en temps polynomial ;*

3. La fonction « val » est calculable en temps polynomial.

L'annexe A contient les problèmes d'optimisation utilisés dans cette thèse.

Trouver le nombre de Ramsey $R(k,k)$ est un exemple de problème d'optimisation qui n'est pas dans NPO. Pour un entier k, le nombre de Ramsey $R(k,k)$ est le plus petit entier n tel que quelque soit une 2-coloration des arêtes du graphe complet K_n avec *rouge* et *bleu*, il existe dans K_n soit un sous-graphe complet K_k colorié en *rouge* seulement soit un sous-graphe complet K_k colorié en *bleu* seulement. On sait que $R(k,k) > 2^{k/2}$ pour $k \geq 3$ et donc ce problème n'est pas un problème de NPO car la taille de la solution (au moins $k/2$) n'est pas polynomiale en la taille de l'instance ($O(\log k)$).

Dans la suite, on va désigner par problème d'optimisation un problème de la classe NPO.

Un problème d'optimisation est *polynomialement borné* s'il existe un polynôme q tel que, pour chaque instance x et chaque solution y de x, $val(x,y) \leq q(|x|)$.

Max Clique est un exemple de problème d'optimisation polynomialement borné car la valeur de la solution optimum est bornée par le nombre de sommets du graphe.

Un exemple de problème d'optimisation qui n'est pas polynomialement borné est Knapsack. Comme la taille d'une instance avec n objets est $O(n \times \log M)$ où M est le plus grand nombre qui apparaît dans l'instance et comme la valeur de la solution optimum peut être plus grande que M, ce problème n'est pas polynomialement borné.

Un algorithme pour un problème est appelé *polynomial* si son temps d'exécution est polynomial en la taille de l'entrée.

La classe PO contient les problèmes de NPO qui peuvent être résolus en temps polynomial.

Un problème d'optimisation est *NP-difficile* si le problème de décision associé est NP-difficile.

2.2 Quatre problèmes à partir d'un problème d'optimisation

La définition d'un problème d'optimisation conduit en général à quatre problèmes :

1. **Le problème de décision :** étant donnés une instance x et un entier k on demande de *décider* s'il existe une solution avec une valeur supérieure à k (dans le cas d'un problème de maximisation) ou inférieure à k (dans le cas d'un problème de minimisation).

2. **Le problème de recherche :** étant donnés une instance x et un entier k on demande, dans le cas d'un problème de maximisation (resp. minimisation) de *trouver* une solution avec une valeur supérieure ou égale (resp. inférieure ou égale) à k si une telle solution existe et de répondre « non » sinon.

3. **Le problème d'évaluation :** étant donnée une instance x, *calculer* la valeur d'une solution optimum.

4. **Le problème constructif :** étant donnée une instance x, *trouver* une solution optimum, c'est-à-dire un élément de $Sol(x)$ avec une valeur maximum ou minimum.

Considérons par exemple le problème MAX CLIQUE. On demande dans le problème de décision s'il existe pour un graphe donné une clique de taille au moins k et dans le problème de recherche de trouver une clique de taille au moins k. On demande dans le problème d'évaluation de calculer la taille maximum d'une clique et dans le problème constructif de trouver une clique de taille maximum.

Les questions naturelles qu'on peut se poser sont les suivantes : quel est le lien entre les complexités de ces quatre problèmes? Est-ce que la réponse à la question précédente dépend du problème d'optimisation?

Il est facile de voir que si on sait résoudre en temps polynomial le problème constructif alors on peut résoudre en temps polynomial le problème d'évaluation et si on sait résoudre en temps polynomial le problème d'évaluation, alors on sait résoudre en temps polynomial le problème de décision. Aussi, si on sait résoudre en temps polynomial le problème constructif, alors on peut résoudre en temps polynomial le problème de recherche et si on sait résoudre en temps polynomial le problème de recherche, alors on sait résoudre en temps polynomial le problème de décision.

Pour un problème d'optimisation, si on peut résoudre le problème de décision alors on peut résoudre le problème d'évaluation en utilisant la méthode de recherche binaire : comme la fonction « val » est calculable en temps polynomial, la valeur maximum d'une solution du problème est bornée par $2^{q(|x|)}$ pour un certain polynôme q et elle peut être trouvée en répondant à au plus $q(|x|)$ instances du problème de décision. Si on peut résoudre le problème de recherche alors on peut résoudre le problème constructif : on trouve d'abord la valeur de la solution optimum en résolvant le problème d'évaluation et on résout ensuite le problème de recherche pour cette valeur.

Intuitivement, il est plus difficile de trouver une solution optimum que la valeur d'une solution optimum et de résoudre le problème de recherche que le problème de décision.

Beigel *et al.* [BBFG91] ont montré que sous l'hypothèse que $EE \neq NEE$ (où EE (resp. NEE) est la classe des langages reconnaissables en temps double-exponentiel (resp. non-déterministe double-exponentiel)), il existe un langage dans NP pour lequel le problème de recherche n'est pas Turing-réductible en temps polynomial au problème de décision. Selman [Sel88a] a montré que le problème de recherche est Turing-réductible en temps polynomial au problème de décision pour les langages disjonctifs auto-réductibles et pour tous les problèmes NP-complets. Hemaspaandra *et al.* [HNOS95] ont construit, sous l'hypothèse acceptée que $NE \neq E$ (où E (resp. NE) est la classe des langages reconnaissables en temps exponentiel (resp. non-déterministe exponentiel)), un langage dans NP pour lequel le problème de recherche est Turing-réductible en temps polynomial au problème de décision mais n'est pas auto-réductible.

Un langage L est *auto-réductible* s'il existe un algorithme polynomial qui, à partir de n'importe quelle instance x, calcule un ensemble d'instances de tailles plus petites que $|x|$ tel que l'appartenance de x à L se réduit à l'appartenance de ces instances à L.

Un langage L est *disjonctif-auto-réductible* s'il existe un algorithme polynomial qui, à partir de n'importe quelle instance x, calcule un ensemble d'instances $x_1, \ldots, x_k$ de tailles plus petites que $|x|$ tel que : $x \in L$ si et seulement si au moins une de ces instances appartient à L.

Étant donnés deux langages L_1 et L_2, on dit que L_1 est *disjonctif-réductible* à L_2 s'il existe une fonction polynomiale qui, à partir de n'importe quelle instance x de L_1 calcule un ensemble fini $\{x_1, \ldots, x_k\}$ d'instances de L_2 tel que $x \in L_1$ si au moins une de ces k instances est dans L_2.

L'exemple classique d'un langage disjonctif auto-réductible est SAT. Le problème SAT demande de décider si une formule CNF est satisfaisable. La réponse à une instance de SAT est « oui » si la formule est satisfaisable ou « non » sinon. Le problème de recherche correspondant à SAT consiste à *trouver* une assignation qui satisfasse la formule si elle est satisfaisable ou à répondre « non » sinon. Ce problème est noté FSAT. On considère une instance ϕ de FSAT qui utilise n variables $x_1, \ldots, x_n$. Soit $\phi[x_1 = valeur]$ la formule ϕ dans laquelle on a remplacé x_1 par $valeur$ où $valeur \in \{vrai, faux\}$. Comme la formule ϕ est satisfaisable si et seulement si au moins une des deux formules $\phi[x_1 = faux]$ ou $\phi[x_1 = vrai]$ est satisfaisable, SAT est disjonctif auto-réductible.

Pour un problème d'optimisation NP-difficile, le problème constructif et le problème d'évaluation sont polynomialement Turing-équivalents :

Théorème 2.2.1 ([ACP94, Théorème 1, p. 6]) *Soit A un problème d'optimisation NP-difficile. Le problème constructif associé à A peut être résolu en temps polynomial par une machine de Turing avec un oracle qui résout le problème d'évaluation.*

Preuve. Supposons que A soit un problème de maximisation. La preuve est similaire pour un problème de minimisation. Nous définissons un autre problème de maximisation B à partir de A. Le problème B est le même que A à l'exception de la valeur associée aux solutions. Soit p le polynôme avec la propriété que pour chaque instance x de A, toute solution possible y de l'instance x a la taille $|y| \leq p(|x|)$. Notons par $n(x, y)$ la position de la solution y pour l'ordre lexicographique. La valeur de la solution y pour l'instance x de B est :

$$val_B(x, y) = 2^{p(|x|)+1} val_A(x, y) + n(x, y).$$

C'est facile de voir que, pour une instance fixée x, la fonction val_B est injective. Cela veut dire qu'il existe une unique solution optimum pour chaque instance du problème B et cette solution est en particulier aussi une solution optimum pour l'instance correspondante du problème A.

Comme le problème de décision associé à A est NP-complet, il existe une Karp-réduction entre les problèmes de décision associés à B et respectivement à A. Utilisant la méthode de recherche binaire, cette réduction et le fait qu'on sait résoudre le problème d'évaluation associé à A, on peut calculer la valeur optimum d'une instance de B. Si on connaît la valeur optimum d'une instance x de B, il est facile de trouver la solution optimum : on divise cette valeur par $2^{p(|x|)+1}$ et le reste de la division est sa position pour l'ordre lexicographique. Ainsi on a trouvé une solution optimum pour l'instance x de A.

□

2.3 Approximation

Un algorithme pour un problème est appelé *pseudo-polynomial* si son temps d'exécution est polynomial dans la taille de l'entrée et dans la taille de la représentation unaire du plus grand nombre qui apparaît dans l'entrée.

Knapsack est un exemple de problème qui a un algorithme pseudo-polynomial. Plus précisément, on peut trouver une solution optimum en temps $O(n^2V)$, où V est la plus grande valeur parmi n objets ([IK75], [Pap94b, p. 305]).

Un problème est *NP-complet au sens fort* s'il reste NP-complet même si le plus grand nombre qui apparaît dans l'instance est polynomial par rapport à la taille de l'entrée. Cela veut dire que la difficulté du problème est due à la structure combinatoire du problème et non pas aux valeurs qui interviennent dans l'instance. Un problème d'optimisation est NP-difficile mais pas au sens fort si la difficulté du problème vient des grands nombres qui interviennent dans l'instance et non du problème.

Il est facile de voir qu'un problème NP-complet au sens fort ne peut pas avoir d'algorithme pseudo-polynomial, à moins que $P = NP$.

Comme la plus part des problèmes d'optimisation sont NP-difficile au sens fort on va dire d'un problème NP-difficile au sens fort qu'il est NP-difficile et on précisera quand un problème est NP-difficile mais pas au sens fort.

Pour les problèmes d'optimisation pour lesquels on ne connaît pas d'algorithme les résolvant en temps polynomial, on essaie de trouver des algorithmes polynomiaux qui donnent des solutions approchées. Pour étudier la qualité d'une solution, on introduit la notion de rapport à l'optimum d'une solution.

Définition 2.3.1 (Rapport à l'optimum) *Étant donnée une instance x d'un problème d'optimisation A, une solution possible y de x a le* rapport à l'optimum

$$R_A(x,y) = \max\left\{\frac{val(x,y)}{opt_A(x)}, \frac{opt_A(x)}{val(x,y)}\right\}.$$

Le rapport à l'optimum est toujours supérieur ou égal à 1 et il est d'autant plus proche de 1 que la valeur de la solution est proche de la valeur optimum.

Définition 2.3.2 (Erreur) *Étant donnée une instance x d'un problème d'optimisation A, une solution possible y de x a* l'erreur $E_A(x,y)$ *si :*

$$\frac{1}{1+E_A(x,y)} \leq \frac{val(x,y)}{opt_A(x)} \leq 1 + E_A(x,y).$$

Il est facile de voir que

$$E_A(x,y) = R_A(x,y) - 1.$$

Un algorithme donne une *approximation à un facteur $f(n)$ près* en temps polynomial pour un problème d'optimisation si pour toute instance x de taille n du problème il calcule en temps polynomial une solution y telle que $R_A(x,y) \leq f(n)$.

Un problème d'optimisation est *approximable à un facteur $f(n)$ près* en temps polynomial s'il existe un algorithme polynomial qui est une approximation à un facteur $f(n)$ près pour le problème. En particulier, un problème d'optimisation est *approximable à un facteur constant près* en temps polynomial s'il existe une constante $c > 1$ et un algorithme polynomial qui donne une approximation à un facteur c près pour le problème.

Un algorithme donne une *approximation à un terme additif $f(n)$ près* en temps polynomial pour un problème d'optimisation A si pour toute instance x de taille n du problème il calcule en temps polynomial une solution y telle que $|m(x,y) - opt_A(x)| \leq f(n)$.

Un algorithme probabiliste donne une *approximation à un facteur $f(n)$ près* en temps polynomial pour un problème d'optimisation si pour toute instance x

de taille n du problème cet algorithme calcule en temps polynomial une solution y telle que

$$\max\left\{\frac{E[val(x,y)]}{opt_A(x)}, \frac{opt_A(x)}{E[val(x,y)]}\right\} \leq f(n)$$

où $E[val(x,y)]$ est la moyenne de la variable aléatoire $val(x,y)$.

Un problème d'optimisation a un *schéma d'approximation en temps polynomial* (abrégé en `ptas` d'après l'appellation anglaise « **p**olynomial-**t**ime **a**pproximation **s**cheme ») si pour toute constante $\varepsilon > 0$, il existe un algorithme polynomial $\mathcal{A}_\varepsilon$ qui soit une approximation à un facteur $(1+\varepsilon)$ près du problème. Le temps de calcul de l'algorithme $\mathcal{A}_\varepsilon$ peut dépendre exponentiellement de $1/\varepsilon$. En général, le temps est de la forme : $n^{O(1/\varepsilon)}$ ou $2^{1/\varepsilon^{O(1)}}n^{O(1)}$. Si le temps de calcul de l'algorithme est de la forme $g(1/\varepsilon)n^{O(1)}$, où g est une fonction quelconque, on dit que le problème a un *schéma d'approximation en temps polynomial efficace* (abrégé en `eptas` de l'anglais « **e**fficient **p**olynomial-**t**ime **a**pproximation **s**cheme ») et si le temps de calcul de l'algorithme est aussi polynomial en $1/\varepsilon$ alors on dit que le problème a un *schéma d'approximation en temps complètement polynomial* (abrégé en `fptas` de l'anglais « **f**ully **p**olynomial-**t**ime **a**pproximation **s**cheme »).

Définition 2.3.3 (APX, PTAS, EPTAS, FPTAS) *APX est la classe des problèmes d'optimisation qui sont approximables à un facteur constant près en temps polynomial. L'ensemble des problèmes qui ont un* `ptas` *(resp.* `eptas`*,* `fptas`*) est noté $PTAS$ (resp. $EPTAS$, $FPTAS$).*

Étant donnée une classe de fonctions F, un problème d'optimisation $A \in F$-APX s'il existe une fonction $f \in F$ telle que A soit approximable à un facteur f près en temps polynomial.

En particulier APX (resp. *log*-APX, *poly*-APX, *exp*-APX) correspond à la classe F-APX où F est l'ensemble $O(1)$ (resp. $O(\log n)$, $O(n^{O(1)})$, $O(2^{n^{O(1)}})$).

APX-PB (resp. *log*-APX-PB) est la classe des problèmes d'optimisation polynomialement bornés qui sont approximables en temps polynomial à un facteur constant près (resp. à un facteur logarithmique près).

On a clairement les inclusions suivantes :

$$PO \subseteq FPTAS \subseteq EPTAS \subseteq PTAS \subseteq APX \subseteq NPO.$$

Dans l'hypothèse où $P \neq NP$, certaines des inclusions précédentes sont strictes : Longest Path est un problème de la classe NPO qui n'est pas approximable à un facteur constant près ([KMR93]). Max 3Sat est approximable à un facteur constant près ([J74]) et n'a pas de `ptas` ([ALMSS92]). Cesati et Trevisan ([CT97]) ont défini un problème artificiel qui admet un `ptas` sans avoir d'`eptas` sous une hypothèse de complexité acceptée (section 2.6.3). Un exemple de problème NP-complet au sens fort et donc qui n'a pas de `fptas` à moins que $P = NP$ (section 2.6.2) mais qui admet un `eptas` ([F96]) est Max Cut sur les instances denses. KNAPSACK est un problème qui a un `fptas` ([IK75]) mais n'est pas soluble en temps polynomial ([K72]).

Entre APX et NPO, les inclusions suivantes sont vérifiées :

$$APX \subseteq log\text{-}APX \subseteq poly\text{-}APX \subseteq exp\text{-}APX \subseteq NPO.$$

MIN SET COVER est un problème approximable à un facteur logarithmique près [J74] et il n'est pas dans APX [LY94]. MAX CLIQUE est évidemment dans $poly\text{-}APX$ car il est polynomialement borné et il n'est pas dans $log\text{-}APX$ (Håstad [H96] a montré que sous une hypothèse acceptée, $co\text{-}RP \neq NP$[1], il n'est pas approximable à un facteur $n^{1-\varepsilon}$ près pour tout $\varepsilon > 0$). MAX ONES SAT est un exemple de problème de NPO pour lequel il est NP-difficile de trouver une solution possible [KSW97].

Pour plusieurs problèmes d'optimisation NP-difficile, il est aussi NP-difficile de trouver une solution approchée. La plupart des résultats de non-approximation montrent que la réalisation d'un certain facteur d'approximation est *NP-difficile.* Ça veut dire que s'il existe un algorithme polynomial qui réalise un tel facteur d'approximation, alors $P = NP$. Certains résultats sont moins forts, ils montrent que l'approximation est *quasi NP-difficile.*

Un problème est quasi NP-difficile si un algorithme polynomial qui peut le résoudre, peut résoudre tout problème de NP en temps quasi-polynomial, c'est-à-dire en temps $n^{p(\log n)}$, où p est un polynôme.

Il est conjecturé pas seulement que NP n'est pas inclus dans P mais aussi qu'un problème NP-difficile n'a pas d'algorithme sous-exponentiel (déterministe ou probabiliste). Une preuve de quasi NP-difficulté est un « certificat crédible » que le problème n'a pas d'algorithme polynomial.

2.4 Réductions et complétude

Comme dans la théorie de la NP-complétude, on peut regarder les problèmes *les plus difficiles* dans les classes précédentes, problèmes qui ne peuvent pas être mieux approchés, à moins que $P = NP$. Être le plus difficile dans une classe veut dire complet sous une certaine réduction.

On introduit dans la suite une réduction entre deux problèmes d'optimisation qui conserve l'approximation, réduction que nous allons utilisé pour montrer que

1. La classe RP contient tous les langages L pour lesquels il existe un algorithme probabiliste polynomial $\mathcal{A}$ tel que pour chaque x :

$$si\ x \in L \text{ alors } P(\mathcal{A}(x) = accept) \geq 3/4,$$

$$si\ x \notin L \text{ alors } P(\mathcal{A}(x) = accept) = 0.$$

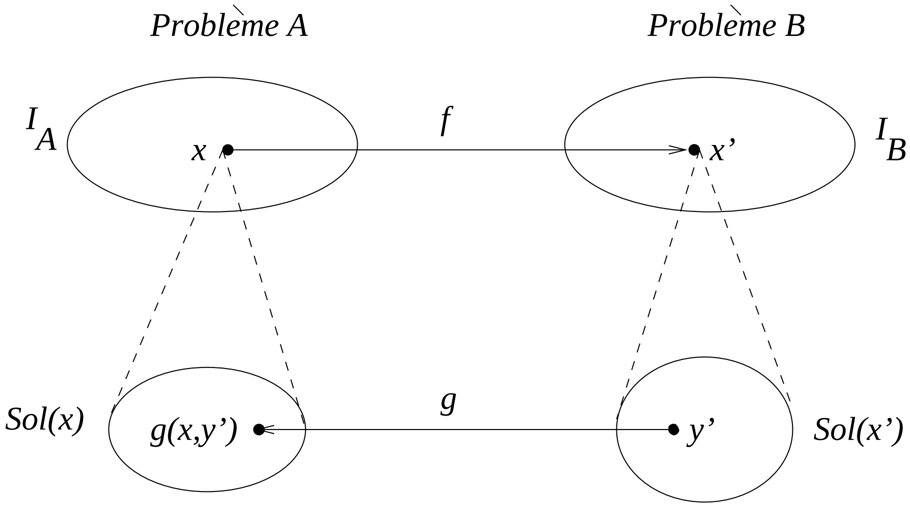


FIG. 2.1 – *La L-réduction*

LONGEST PATH dans les graphes cubiques et Hamiltoniens n'a pas de schéma d'approximation en temps polynomial. Il s'agit de la réduction classique qui est appelée la L-réduction (L pour *linéaire*) et qui a été définie par Papadimitriou et Yannakakis [PY91].

Définition 2.4.1 (L-réduction) *Soient A et B deux problèmes d'optimisation. On dit que A est L-réductible à B et on écrit $A \leq_L B$ s'il existe deux fonctions calculables en temps polynomial f, g et deux constantes $\alpha, \beta > 0$ telles que :*

1. *f transforme une instance x de A en une instance $x' = f(x)$ de B telle que $opt_B(x') \leq \alpha \cdot opt_A(x)$,*

2. *g transforme toute solution y' de x' en une solution $y = g(x, y')$ de x telle que*
$$|val(x,y) - opt_A(x)| \leq \beta \cdot |val(x',y') - opt_B(x')|.$$

La propriété fondamentale de la L-réduction est de préserver le schéma d'approximation.

Lemme 2.4.2 ([PY91]) *Si $B \in PTAS$ et $A \leq_L B$, alors $A \in PTAS$.*

Définition 2.4.3 (Complétude) *Soit $\mathcal{C}$ une classe de problèmes d'optimisation. Un problème d'optimisation A est $\mathcal{C}$-difficile sous la L-réduction si tout problème $B \in \mathcal{C}$ est L-réductible au problème A. Un problème d'optimisation A est $\mathcal{C}$-complet si $A \in \mathcal{C}$ et A est $\mathcal{C}$-difficile.*

La L-réduction a l'inconvénient de ne pas préserver l'appartenance dans APX à moins que $P = NP \cap co\text{-}NP$ ([CKST95]).

Nous présentons ci-dessous la définition de la E-réduction (« error-preserving reduction ») introduite par Khanna, Motwani, Sudan et Vazirani [KMSV94] pour montrer un résultat important de la théorie de l'approximation que nous allons présenter à la fin de la section 2.7. Une propriété importante de la E-réduction est qu'elle peut être appliquée uniformément à tous les niveaux d'approximation. Cela veut dire que si A est E-réductible à B et $B \in \mathcal{C}$ où $\mathcal{C} \in \{PTAS, APX, \mathit{log}\text{-}APX, \mathit{poly}\text{-}APX, \mathit{exp}\text{-}\mathrm{APX}\}$ alors $A \in \mathcal{C}$.

La L-réduction demande que les optimums des deux instances des problèmes soient linéairement liés. C'est une contrainte assez restrictive qui n'apparaît pas dans la E-réduction.

Définition 2.4.4 (E-réduction) *Soient A et B deux problèmes d'optimisation. On dit que A est E-réductible à B et on écrit $A \leq_E B$ s'il existe deux fonctions calculables en temps polynomial f, g, un polynôme p et une constante $\beta > 0$ tels que :*

1. *f transforme une instance x de A en une instance $x' = f(x)$ de B telle que $opt_B(x') \leq p(|x|) \cdot opt_A(x)$,*
2. *g transforme toute solution y' de x' en une solution $y = g(x, y')$ de x telle que*
$$E_A(x, y) \leq \beta \cdot E_A(x', y').$$

La E-réduction demande que l'erreur de la solution pour le premier problème soit linéairement liée à l'erreur de la solution pour le deuxième problème.

Les notions de difficulté et complétude se définissent de la même manière que pour la L-réduction.

Il existe d'autres réductions conservant l'approximation. [Cres97] constitue un guide avec les définitions et certaines corrélations.

2.5 Caractérisation logique

Inspirés de la caractérisation logique donnée par Fagin pour la classe NP, qui établie que toute propriété de NP peut être exprimée dans la logique existentielle de second ordre, Papadimitriou et Yannakakis [PY91] ont défini les classes de problèmes d'optimisation $MAX\ SNP$ et $MAX\ NP$.

Une sous-classe intéressante de NP est la classe *strict* NP ou SNP qui contient les propriétés exprimables ainsi :

$$\exists S \forall y_1 \ldots \forall y_k \phi(x, S, y_1, \ldots, y_k)$$

où x est l'entrée et ϕ est une formule du premier ordre sans quantificateur. En fait, on demande de décider s'il existe une relation S telle que tout k-uplet de variables $(y_1, \ldots, y_k)$ satisfasse ϕ.

On considère dans la suite qu'il est demandé de trouver une relation S telle que ϕ soit satisfaite pour un nombre maximum de k-uplets.

Définition 2.5.1 ($MAX\ SNP$) *Un problème de maximisation A est dans la classe $MAX\ SNP$ s'il existe une séquence des relations $R_1, \ldots, R_m, S$ sur un univers $\mathcal{U}$ et une formule du premier ordre sans quantificateur ϕ telles que la valeur de la solution optimum sur l'entrée x satisfasse :*

$$opt_A(x) = \max_S |\{(y_1, \ldots, y_k) \in \mathcal{U}^k : \phi(x, R_1, \ldots, R_m, S, y_1, \ldots, y_k)\}|.$$

Max Cut est un exemple de problème de $MAX\ SNP$. Pour une instance x donnée, l'univers $\mathcal{U}$ est l'ensemble des sommets du graphe, E est la relation binaire qui représente l'adjacence de deux sommets et S est une relation unaire qui représente une partie de la partition. L'optimum du problème est :

$$opt(x) = \max_S |\{(y_1, y_2) : E(y_1, y_2) \wedge (y_1 < y_2) \wedge ((S(y_1) \wedge \neg S(y_2)) \vee (\neg S(y_1) \wedge S(y_2)))\}|.$$

Un problème A est $MAX\ SNP$-difficile sous la L-réduction si tout problème de $MAX\ SNP$ est L-réductible à A. Un problème est $MAX\ SNP$-complet sous la L-réduction s'il est dans $MAX\ SNP$ et $MAX\ SNP$-difficile sous la L-réduction.

Théorème 2.5.2 ([PY91, Théorème 2, p. 431]) Max 3Sat *est $MAX\ SNP$-complet sous la L-réduction.*

Papadimitriou et Yannakakis [PY91] ont montré que plusieurs autres problèmes sont $MAX\ SNP$-difficiles : Max 2Sat, Max Cut ainsi que quelques problèmes sur les graphes dont le degré maximum est borné par une constante: Max Independent Set-B, Min Vertex Cover-B.

Définition 2.5.3 ($MAX\ NP$) *Un problème de maximisation A est dans la classe $MAX\ NP$ s'il existe une séquence des relations $R_1, \ldots, R_m, S$ sur un univers $\mathcal{U}$ et une formule du premier ordre sans quantificateur ϕ telles que la valeur de la solution optimum sur l'entrée x satisfasse :*

$$opt_A(x) = \max_S |\{(y_1, \ldots, y_k) \in \mathcal{U}^k : \exists z_1 \ldots \exists z_t \phi(x, R_1, \ldots, R_m, S, y_1, \ldots, y_k, z_1, \ldots, z_t)\}|.$$

Max Sat est un problème de $MAX\ NP$. Pour une instance x de Max Sat, l'univers $\mathcal{U}$ est l'ensemble des clauses, $P(y, c)$ veut dire que la variable y apparaît positivement dans la clause c et $N(y, c)$ que la variable y apparaît négativement dans c. Pour une solution de x, T contient les variables vraies. Alors, l'optimum de Max Sat est de la forme:

$$opt(x) = \max_T |\{c : \exists y(P(y,c) \wedge T(y)) \vee (N(y,c) \wedge \neg T(y))\}|.$$

À ce moment là, quand Papadimitriou et Yannakakis ont défini ces deux classes, on ne connaissait pas de problème $MAX\ NP$-complet. Plus tard Khanna *et al.* [KMSV94] ont montré que MAX SAT est $MAX\ NP$-complet sous la E-réduction.

L'étude des problèmes d'optimisation en utilisant la logique a été continuée par Panconesi et Ranjan [PR90] qui ont défini la classe $RMAX(2)$ qui contient MAX CLIQUE comme problème complet sous une certaine réduction qui préserve l'approximation. Ainsi les problèmes de la classe $RMAX(2)$ sont approximable à un facteur polynomial près en temps polynomial.

Définition 2.5.4 ($RMAX(2)$) *Un problème de maximisation A est dans la classe $RMAX(2)$ s'il existe une relation S sur un univers $\mathcal{U}$ et une formule CNF du premier ordre sans quantificateur ϕ qui contient que des occurrences négatives de S et en nombre de au plus 2 par clause telles que la valeur de la solution optimum sur l'entrée x satisfasse :*

$$opt_A(x) = \max_S\{|S| : \forall y_1 \ldots \forall y_t \phi(x, S, y_1, \ldots, y_t)\}.$$

Une approche syntaxique plus systématique a été réalisée par Kolaitis et Thakur [KT90] qui ont défini deux hiérarchies : une pour les problèmes de maximisation et une pour les problèmes de minimisation. Ils ont défini deux classes $MIN\ F^+\Pi_1$ et $MIN\ F^+\Pi_2(1)$ qui contiennent respectivement les problèmes MIN VERTEX COVER et MIN SET COVER.

Définition 2.5.5 ($MIN\ F^+\Pi_1$) *Un problème de minimisation A est dans la classe $MIN\ F^+\Pi_1$ s'il existe une relation S sur un univers $\mathcal{U}$ et une formule du premier ordre sans quantificateur ϕ qui contient que des occurrences positives de S telles que la valeur de la solution optimum sur l'entrée x satisfasse :*

$$opt_A(x) = \min_S\{|S| : \forall y_1 \ldots \forall y_t \phi(x, S, y_1, \ldots, y_t)\}.$$

La classe $MIN\ F^+\Pi_1$ contient que des problèmes approximable, à un facteur constant près [KT90].

Définition 2.5.6 ($MIN\ F^+\Pi_2(1)$) *Un problème de minimisation A est dans la classe $MIN\ F^+\Pi_2(1)$ s'il existe une relation S sur un univers $\mathcal{U}$ et une formule DNF du premier ordre sans quantificateur ϕ qui contient que des occurrences positives de S et S apparaît au plus une fois dans chaque conjonction telles que la valeur de la solution optimum sur l'entrée x satisfasse :*

$$opt_A(x) = \min_S\{|S| : \forall y_1 \ldots \forall y_s \exists z_1 \ldots \exists z_t \phi(x, S, y_1, \ldots, y_s, z_1, \ldots, z_t)\}.$$

MIN SET COVER est $MIN\ F^{+}\Pi_2(1)$-complet sous une certaine réduction qui préserve l'approximation [KT91]. Ainsi la classe $MIN\ F^{+}\Pi_2(1)$ contient que des problèmes approximables à un facteur logarithmique près.

Les variantes avec des poids positifs des problèmes des classes $MAX\ SNP$, $MAX\ NP$, $MIN\ F^{+}\Pi_1$ et $MIN\ F^{+}\Pi_2(1)$ ont les mêmes propriétés d'approximation que les problèmes sans poids. Plus précisément, les problèmes des deux premières classes sont approximables à une constante près [PY91], les problèmes de $MIN\ F^{+}\Pi_1$ sont aussi approximables à une constante près [Z95] et les problèmes de la quatrième sont approximables à un facteur logarithmique près [Z95]. Zimand a étudié les propriétés d'approximation de quelques problèmes de ces classes quand on considère des poids positifs et négatifs. Il a montré que beaucoup de problèmes deviennent plus difficiles. Par exemple, il a montré qu'il existe une constante $c > 0$ telle que MAX 2SAT et MIN VERTEX COVER ne sont pas approximables à un facteur $2^{\log^c n}$ près, où n est la taille de l'instance, à moins que l'on puisse résoudre tout problème de NP en temps quasi-polynomial.

2.6 Schémas d'approximation et complexité paramétrée

2.6.1 FPTAS

Il est évident que pour un problème NP-difficile donné, l'idéal est de trouver un `fptas`. Quels sont les problèmes qui peuvent avoir un `fptas`? Nous allons voir dans la suite que les problèmes NP-difficiles au sens fort ne peuvent pas avoir de `fptas`, les seuls problèmes qui peuvent en avoir sont donc ceux qui ne le sont pas.

Théorème 2.6.1 ([GJ76a, Théorème 6.8]) *Soit A un problème d'optimisation avec des valeurs entières pour lequel il existe un polynôme p tel que pour chaque instance x de A, $opt_A(x) < p(|x|, max(x))$, où $max(x)$ est le plus grand nombre apparaissant dans l'instance x. Alors l'existence d'un* `fptas` *pour A implique l'existence d'un algorithme pseudo-polynomial pour A.*

Preuve. Supposons que A soit un problème de maximisation. La preuve pour un problème de minimisation est similaire. Notons par $\mathcal{A}_\varepsilon$ un schéma d'approximation complètement polynomial pour A qui donne une solution avec le rapport à l'optimum $1 + \varepsilon$. Soit $\mathcal{B}$ l'algorithme suivant : sur une instance x de A il exécute l'algorithme $\mathcal{A}_\varepsilon$ où

$$\varepsilon = \frac{1}{p(|x|, \max(x))}.$$

L'algorithme $\mathcal{B}$ travaille en temps polynomial en $|x|$ et $p(|x|, \max(x))$ et donc en temps pseudo-polynomial. La solution y retournée par l'algorithme a une valeur

$$m(x, y) \geq \frac{opt_A(x)}{1 + \varepsilon}$$

et donc $opt_A(x) - m(x, y) \leq \varepsilon m(x, y) \leq \varepsilon opt_A(x) < 1$. Ainsi y est une solution optimum car le problème A a des solutions avec des valeurs entières.

□

Corollaire 2.6.2 ([GJ76a, Corollaire, p. 141]) *Soit A un problème d'optimisation avec des valeurs entières pour lequel il existe un polynôme p tel que pour chaque instance x de A, $opt_A(x) < p(|x|, max(x))$, où $max(x)$ est le plus grand nombre qui apparaît dans l'instance x. Si A est NP-difficile au sens fort, alors A n'a pas de* `fptas` *à moins que $P = NP$.*

Ce corollaire est souvent utilisé car la plupart des problèmes d'optimisation sont NP-difficiles au sens fort. MAX CLIQUE a l'optimum polynomialement borné et il est un problème NP-difficile au sens fort. Ainsi, il n'a pas de schéma d'approximation en temps complètement polynomial à moins que $P = NP$. TSP(1,2) est un autre exemple d'un problème qui n'a pas de `fptas` à moins que $P = NP$ car il satisfait les hypothèses du corollaire précédent : il a l'optimum borné par $2n$ où n est le nombre des sommets du graphe et en même temps la taille de l'entrée et il est NP-difficile ([K72]).

2.6.2 Complexité paramétrée

La théorie de la complexité à paramètre fixé a été introduite par Downey et Fellows au début des années 90. Ils observent que de nombreux problèmes de calcul ont dans l'entrée un paramètre qui peut contribuer à la complexité du problème de différentes façons.

Le problème MIN VERTEX COVER qui consiste à décider, pour un graphe G d'ordre n, s'il existe un ensemble de sommets, S, de taille k, contenant au moins une des extrémités de chaque arête, est NP-complet. Si on fixe le paramètre k, ce problème devient polynomial parce que l'algorithme qui prend en compte tous les sous-ensembles de taille k est en $O(n^k)$. Mais, on peut faire mieux. Il existe un algorithme qui résout ce problème en temps $O(n^c)$, où c est une constante qui ne dépend pas de k [DF95].

Par contre pour le problème MIN DOMINATING SET, qui consiste à décider pour un graphe G d'ordre n s'il existe un ensemble de sommets S de taille k tel que chaque sommet de G est dans S ou est lié avec un sommet de S, le meilleur algorithme connu est l'algorithme qui essaie exhaustivement tous les ensembles possibles de dimension k, et donc nécessite un temps $O(n^{k+1})$.

Pour étudier les problèmes paramétrés difficiles, Downey et Fellows ont introduit une hiérarchie nommée la hiérarchie W. Nous allons définir ensuite les classes de cette hiérarchie.

Une motivation pour l'étude de la faisabilité paramétrée vient de sa liaison avec l'approximation des problèmes d'optimisation. Cette liaison est expliquée dans la section 2.6.3.

Notion de paramètre et complexité paramétrée

Nous allons définir naturellement les notions de problème paramétré, de problème traitable et de réduction entre deux problèmes paramétrés qui préserve la traitabilité.

Définition 2.6.3 (Problème paramétré) *Un* problème paramétré *est un ensemble $A \subseteq \Sigma^* \times \mathbb{N}$, où Σ est un alphabet fixé. Une instance de A est une paire $\langle x, k\rangle$, où le deuxième argument k est en unaire. On appelle k le paramètre du problème.*

La complexité d'un problème paramétré est spécifiée en fonction des deux composantes d'une instance.

Définition 2.6.4 (Traitabilité paramétrée) *Un problème paramétré A est* traitable *s'il existe une constante c et un algorithme qui décide si $\langle x, k\rangle \in A$ dans un temps $f(k) \cdot |x|^c$, où $f : \mathbb{N} \to \mathbb{N}$ est une fonction récursive.*

On note avec FPT la classe des problèmes paramétrés qui sont traitables.

Définition 2.6.5 (Réduction paramétrée) *Soient A, B deux problèmes paramétrés. A* est réductible à *B s'il existe un algorithme qui transforme une instance $\langle x, k\rangle$ de A en une instance $\langle x', g(k)\rangle$ de B en temps $O(f(k)|x|^c)$, où $f, g : \mathbb{N} \to \mathbb{N}$ sont des fonctions récursives et c une constante indépendante de k et*

$$\langle x, k\rangle \in A \Leftrightarrow \langle x', g(k)\rangle \in B \ .$$

On note une réduction entre deux problèmes paramétrés A et B avec $A \prec_{fp} B$.

Lemme 2.6.6 *Si $A \prec_{fp} B$ et $B \in FPT$, alors $A \in FPT$.*

Circuits booléens et classes des problèmes paramétrés

Un circuit booléen C à n variables est un graphe orienté sans circuit qui admet n entrées booléennes.

Les nœuds de degré entrant 0 s'appellent noeuds d'entrée et ils sont étiquetés avec des éléments de $\{0, 1, x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$, où $x_1, \ldots, x_n$ sont des variables et $\bar{x}_i$ est la négation de la variable x_i, $1 \leq i \leq n$. Les nœuds de degré entrant strictement positif s'appellent des portes. Les nœuds de degré sortant 0 s'appellent des sorties.

Un circuit booléen est de type mixte s'il contient 2 sortes de portes :

des portes petites : **not, and, or** de degré entrant borné

et des portes larges : **AND, OR** de degré entrant non-borné.

Un circuit de décision est un circuit avec une seule sortie.

Un circuit de décision C accepte un vecteur x si la sortie a la valeur 1 sur l'entrée x.

Le poids d'un vecteur booléen x est le nombre de 1 dans le vecteur.

La profondeur d'un circuit C, $d(C)$, est le nombre maximum de portes qui se trouvent sur un chemin de l'entrée à la sortie de C.

Si sur un chemin de l'entrée à la sortie d'un circuit on trouve deux portes consécutives différentes on dit qu'il s'agit de portes alternées.

Le nombre des branchements d'un circuit C, $w(C)$, est le nombre de portes larges alternées qui se trouvent sur un chemin de l'entrée à la sortie de C.

Une famille de circuits est un ensemble $\{C_n, n \in \mathbb{N}\}$, où chaque C_n a une entrée de longueur n.

Une famille de circuits F est de profondeur bornée s'il existe une constante h telle que $\forall C \in F$, $d(C) \leq h$.

Une famille de circuits F est de nombre de branchements borné s'il existe une constante t telle que $\forall C \in F$, $w(C) \leq t$.

Intuitivement, les classes de complexité paramétré que nous allons définir sont basées sur la complexité des circuits qui vérifient une solution.

Définition 2.6.7 *Soit F une famille de circuits de décision. On associe à F* le problème paramétré

$L_F = \{(C, k) : C \in F \textit{ et } C \textit{ accepte un vecteur de poids } k\}$.

Définition 2.6.8 (W[t]) *Un problème paramétré A est dans $W[t]$ s'il existe une constante h et une famille de circuits de décision de profondeur au plus h, de nombre des branchements au plus t et avec un nombre polynomial de portes, $F(h, t)$, tel que $A \prec_{fp} L_{F(h,t)}$.*

On note $WH = \bigcup_{t\geq 1} W[t]$.

Définition 2.6.9 (W[P]) *Un problème paramétré A est dans $W[P]$ s'il existe une famille de circuits de décision F avec un nombre polynomial de portes telle que $A \prec_{fp} L_F$ où L_F est le problème paramétré associé à la famille F.*

Évidemment,

$$FPT \subseteq W[1] \subseteq W[2] \subseteq ... \subseteq WH \subseteq W[P].$$

On a conjecturé que toutes les inclusions de cette hiérarchie sont strictes.

On va donner ensuite une définition formelle d'un problème paramétré associé à un problème d'optimisation. On va voir que la non-traitabilité paramétrée d'un problème implique la non-approximabilité du problème d'optimisation. Donc l'étude de non-traitabilité paramétrée donne un nouvel instrument pour prouver la non-approximabilité des problèmes d'optimisation.

Pour paramétriser un problème d'optimisation, on va associer le paramètre avec la valeur optimum de la fonction de coût du problème.

Définition 2.6.10 *Soit $A = (I, Sol, val, but)$ un problème d'optimisation. Le problème paramétré correspondant $A_=$ est défini comme*

$$A_= = \{\langle x, k\rangle | x \in I \wedge k = opt_A(x)\}$$

où $opt_A(x)$ est la valeur optimum de A sur l'entrée x.

Définition 2.6.11 *Un problème d'optimisation A est* traitable *si le problème paramétré associé est traitable.*

2.6.3 EPTAS

Le résultat de Garey et Johnson (théorème 2.6.1) peut être amélioré ainsi : si un problème d'optimisation à valeurs entières a un `eptas` alors il est traitable du point de vue de la complexité paramétrée[2] ([CT97]). Donc, si on réussit à montrer qu'un problème n'est pas traitable alors on peut déduire qu'il n'a pas d'`eptas`.

Cesati et Trevisan ([CT97]) ont défini un problème artificiel qui admet un `ptas` sans avoir d'`eptas` à moins que $W[P] = FPT$. Donc, sous une hypothèse plus forte que l'hypothèse $P \neq NP$, c'est-à-dire si $W[P] \neq FPT$, alors $EPTAS \subset PTAS$.

2. J'ai démontré ce résultat pour la première fois dans *Schémas d'approximation et complexité paramétrée*, Rapport de DEA de l'Université Paris-Sud, 1995.

2.7 PCP et non-approximation

Nous avons vu que NP est la classe des langages vérifiables en temps polynomial (définition 2.1.2). Cela veut dire que $L \in NP$ si et seulement s'il existe une machine de Turing M, appelée *vérificateur*, qui décide en temps polynomial R où $L = \{x : \text{il existe un } y \text{ tel que } R(x,y)\}$.

Cook a montré qu'on peut réduire la puissance du vérificateur à une machine de Turing déterministe qui travaille en espace logarithmique. La question naturelle qu'on s'est posé est de savoir jusqu'où on peut réduire la puissance du vérificateur tout en caractérisant la classe NP.

Pour cela on a défini un nouveau vérificateur qui est une machine probabiliste avec un ruban d'entrée, un ruban de travail, un ruban contenant une chaîne aléatoire et un ruban qui contient la preuve. Le vérificateur travaille ainsi : il lit l'entrée, la chaîne aléatoire et il écrit sur le ruban de travail certaines adresses de la preuve. Il lit ensuite les bits de la preuve à ces adresses et en fonction de ces bits, de l'entrée et de la chaîne aléatoire, répond « oui » ou « non ».

Un vérificateur est un $(r(n), q(n))$-*vérificateur* si sur chaque entrée de taille n, il utilise au plus $O(r(n))$ bits aléatoires et lit au plus $O(q(n))$ bits de la preuve.

Un vérificateur M *vérifie aléatoirement les preuves d'appartenance* pour le langage L si :

– Pour chaque entrée $x \in L$ il existe une preuve y_x telle que M répond « oui » pour chaque chaîne aléatoire.

– Pour chaque entrée $x \notin L$ et chaque preuve y il existe au moins la moitié des chaînes aléatoires pour lesquelles le vérificateur répond « non ».

Définition 2.7.1 (PCP) *Un langage $L \in PCP(r(n), q(n))$ s'il existe un $(r(n), q(n))$-vérificateur qui vérifie aléatoirement les preuves d'appartenance pour le langage L.*

Observons que $NP = PCP(0, poly(n))$ car $PCP(0, poly(n))$ est la classe des langages pour lesquels la preuve d'appartenance peut être vérifiée en temps polynomial déterministe et NP est la classe des langages qui ont pour chaque instance positive un témoin de taille polynomiale.

Le théorème suivant qui est connu comme **le théorème de PCP** est un résultat très intéressant qui a beaucoup de conséquences dans la théorie de l'approximation.

Théorème 2.7.2 ([ALMSS92]) $NP = PCP(\log n, 1)$

L'une de ses conséquences est le théorème suivant qui est un raffinement du théorème de Cook sur la NP-difficulté de 3SAT.

Théorème 2.7.3 ([ALMSS92]) *Soit L un langage dans NP. Il existe un algorithme polynomial et une constante $0 < \varepsilon < 1$ tels que, étant donnée une entrée*

x, l'algorithme construit une instance F_x de 3SAT *qui satisfait les propriétés suivantes:*

1. *Si $x \in L$ alors F_x est satisfaisable.*
2. *Si $x \notin L$ alors il n'y a pas d'assignation qui satisfait plus qu'une fraction $(1-\varepsilon)$ des clauses.*

Corollaire 2.7.4 MAX 3SAT *n'a pas de schéma d'approximation en temps polynomial, à moins que $P = NP$.*

Comme MAX 3SAT est $MAX\ SNP$-difficile sous la L-réduction on peut conclure que:

Corollaire 2.7.5 *Tout problème $MAX\ SNP$-difficile sous la L-réduction n'a pas de schéma d'approximation en temps polynomial, à moins que $P = NP$.*

Nous allons voir dans la suite d'autres résultats de non-approximation qu'on peut montrer en utilisant le théorème 2.7.2 (pour plus de détails voir le chapitre 10 de [Hoch95]) :

Corollaire 2.7.6

- *Il existe un $\varepsilon > 0$ tel que* MAX CLIQUE *n'est pas approximable à un facteur n^ε près en temps polynomial.*
- *Pour tout $\varepsilon > 0$, l'approximation de* MIN LABEL COVER *à un facteur $2^{\log^{1-\varepsilon} n}$ près en temps polynomial est quasi NP-difficile.*
- *Il existe une constante $c > 1$ telle que l'approximation de* MIN SET COVER *à un facteur $\frac{\log n}{c}$ près en temps polynomial est quasi NP-difficile.*
- *Il existe un $\varepsilon > 0$ tel que* MIN COLORING *n'est pas approximable à un facteur n^ε près en temps polynomial.*

Certains des résultats de non-approximation mentionnés ne sont pas les meilleurs connus. Pour connaître les meilleurs résultats d'approximation et non-approximation pour les problèmes d'optimisation les plus connus on peut consulter le compendium de Crescenzi et Kann ([CK]).

Nous allons définir ensuite une réduction « qui préserve l'intervalle », réduction qui peut être couplée avec un des résultats du corollaire précédent pour déduire d'autres résultats de non-approximation.

Définition 2.7.7 *Soit A et B deux problèmes de maximisation. Une* réduction qui préserve l'intervalle *de A à B avec les paramètres $(c, \rho), (c', \rho')$ est une fonction polynomiale π qui calcule pour chaque instance x de A une instance $x' = \pi(x)$ de B telle que :*

- *si $opt_A(x) \geq c$ alors $opt_B(x') \geq c'$;*

– *si* $opt_A(x) < \frac{c}{\rho}$ *alors* $opt_B(x') < \frac{c'}{\rho'}$.

En fait dans la définition précédente c, ρ et respectivement c', ρ' sont fonctions de $|x|$ et respectivement de $|x'|$ et $\rho(x), \rho'(x') \geq 1$.

La réduction précédente peut être modifiée pour le cas où un ou les deux problèmes sont de minimisation.

En fait dans une réduction « qui préserve l'intervalle » ρ peut être plus petit que ρ' ou inverse.

Supposons qu'on a montré une réduction qui préserve l'intervalle entre A et B avec les paramètres de la définition précédente. Supposons aussi qu'il existe une réduction polynomiale f entre SAT et A telle que pour chaque formule x :

– si $x \in$ SAT alors $opt_A(f(x)) \geq c$;
– si $x \notin$ SAT alors $opt_A(f(x)) < \frac{c}{\rho}$.

En composant les deux réductions précédentes on obtient une réduction $\pi \circ f$ de SAT à B telle que pour chaque formule x :

– si $x \in$ SAT alors $opt_B(\pi \circ f(x)) \geq c'$;
– si $x \notin$ SAT alors $opt_B(\pi \circ f(x)) < \frac{c'}{\rho'}$.

Cette réduction montre que B n'est pas approximable à un facteur ρ' près à moins que $P = NP$.

La réduction qui préserve l'intervalle est une notion moins restrictive que la L-réduction car pour les instances x pour lesquelles $\frac{c}{\rho} \leq opt(x) < c$ la réduction peut se comporter arbitrairement. Une L-réduction entre deux problèmes A et B et un algorithme d'approximation pour B donnent un algorithme d'approximation pour A. Il n'est pas le cas pour une réduction qui préserve l'intervalle. Par contre, une réduction qui préserve l'intervalle est suffisante pour déduire un résultat de non-approximation pour B en sachant que A est non-approximable. Par exemple, il existe une réduction qui préserve l'intervalle pour montrer la non-approximation de MIN SET COVER mais une L-réduction correspondante n'est pas connue.

Définition 2.7.8 ($\overline{\mathcal{C}}^E$) *Pour une classe de problèmes* $\mathcal{C}$*, soit* $\overline{\mathcal{C}}^E$ *l'ensemble des problèmes E-réductibles aux problèmes de* $\mathcal{C}$*.*

Les classes d'approximation comme APX, $PTAS$ ont la propriété qu'elles contiennent des problèmes avec une complexité d'approximation connue. Mais il y avait peu des problèmes complets connus pour ces classes et ceux connus était artificiels. Les classes syntaxiques comme $MAX\ SNP$, $MAX\ NP$, $RMAX(2)$, $MIN\ F^+\Pi_2(1)$ ont des problèmes complets naturels mais leur complexité d'approximation n'était pas connue.

En vue d'obtenir un lien entre ces classes syntaxiques d'un coté et celles avec une approximation connue d'un autre, Khanna, Motwani, Sudan et Vazirani [KMSV94] ont comparé une relaxation d'une classe syntaxique, $\overline{MAX\ SNP}^E$ avec APX. En utilisant le théorème de PCP, ils ont montré que MAX 3SAT est

APX-PB-complet sous la E-réduction. Le résultat de Papadimitriou et Yannakakis [PY91] peut être adapté pour montrer que Max kSat est $MAX\ SNP$-complet sous la E-réduction, pour tout $k \geq 2$ [KMSV94]. Donc,

Théorème 2.7.9 ([KMSV94]) $\overline{MAX\ SNP}^E = APX\text{-}PB$.

Comme $MAX\ SNP \subseteq APX$-PB et la E-réduction préserve l'appartenance en APX, tout problème E-réductible à un problème de $MAX\ SNP$ est approximable à un facteur constant près. Le théorème précédent « dit » en plus que tout problème polynomial borné et approximable à un facteur constant près peut être E-réductible à un problème de $MAX\ SNP$.

Ce théorème a quelques conséquences :

Corollaire 2.7.10 ([KMSV94]) $\overline{MAX\ SNP}^E = \overline{MAX\ NP}^E$.

Papadimitriou et Yannakakis ont laissé comme question ouverte de décider si $MAX\ NP$ a des problèmes complets. La conséquence suivante répond à cette question affirmativement.

Corollaire 2.7.11 ([KMSV94]) Max Sat *est $MAX\ NP$-complet sous la E-réduction.*

Dans le même article, ils ont montré des résultats similaires pour les classes $RMAX(2)$ et $MIN\ F^+\Pi_2(1)$. Plus précisément que $\overline{RMAX(2)}^E = poly\text{-}APX$ et sous une hypothèse de complexité acceptée que $\overline{MIN\ F^+\Pi_2(1)}^E = log\text{-}APX$.

2.8 Techniques fréquemment utilisées

Un moyen pour montrer qu'un problème d'optimisation n'a pas de schéma d'approximation $PTAS$ est la *technique de l'intervalle* que nous allons illustrer sur un exemple de problème de coloration des arêtes d'un graphe. Holyer [Hol81] a montré qu'il est NP-difficile de décider si les arêtes d'un graphe sont coloriables avec 3 couleurs. Nous montrons que ce problème n'est pas approximable à un facteur mieux que 4/3 en temps polynomial, à moins que $P = NP$. Si un tel algorithme existait, alors il colorierait un graphe 3 coloriable avec 3 couleurs, donc on pourrait l'utiliser pour décider si un graphe est 3 coloriable ou non. Pour des problèmes comme Min Vertex Cover, Max Sat et Max Independent Set ce raisonnement ne marche pas car pour ces problèmes on peut décider en temps polynomial si leurs valeurs optimum sont plus grandes ou plus petites qu'une constante k fixée.

Une méthode utilisée en général pour montrer qu'un problème d'optimisation n'est pas dans APX est de montrer que le problème a un *schéma d'auto-amélioration* (c'est-à-dire que si le problème est dans APX alors il est dans $PTAS$) et après de montrer que le problème n'est pas dans $PTAS$. Le produit des graphes a été utilisé pour établir une propriété d'auto-amélioration pour MAX CLIQUE [GJ76b] et aussi pour LONGEST PATH [KMR93]. En utilisant une construction spéciale, Alon, Feige, Wigderson et Zuckerman [AFWZ93] ont montré qu'il existe un $\varepsilon > 0$ tel que MAX CLIQUE n'est pas approximable à un facteur n^{ε} près, à moins que $P = NP$. En fait ce dernier résultat a été montré pour la première fois dans [ALMSS92], mais la preuve est plus compliquée. Arora, Babai, Stern et Sweedyk [ABSS93] ont montré le même résultat pour MAX SATISFIABILITY en utilisant la même construction.

L'arrondissage aléatoire est une autre technique utilisée pour montrer des résultats d'approximation pour des problèmes d'optimisation. L'idée est de formuler le problème comme un programme linéaire entier, de résoudre la relaxation de ce programme et après en utilisant l'aléa de trouver une solution pour le problème. En utilisant cette méthode, Goemans et Williamson [GW93] ont donné un algorithme d'approximation pour MAX SAT à un facteur 4/3 près. Cette idée a été améliorée par Goemans et Williamson [GW95] qui ont donné une approximation à un facteur 1,14 près pour MAX CUT et pour MAX 2SAT : ils modélisent les problèmes comme un programme semi-défini et ils utilisent l'aléa pour passer d'une solution de la relaxation de ce programme à une solution pour le problème-même. Cette méthode a été beaucoup utilisée depuis pour donner des algorithmes avec un meilleur facteur d'approximation pour certains problèmes d'optimisation : Karger, Motwani et Sudan ont amélioré le facteur d'approximation pour le problème de coloration des sommets d'un graphe k-coloriable [KMS94], Frieze et Jerrum ont amélioré le facteur d'approximation pour MAX kCUT (la variante de MAX CUT dans laquelle on veut partitioner l'ensemble des sommets en k parties) et MAX BISECTION (la variante de MAX CUT dans laquelle on demande que les deux parties soit de la même taille) [FJ95].

Une technique pour montrer qu'un problème d'optimisation est dans $FPTAS$ est de montrer d'abord en utilisant la programmation dynamique que le problème a un algorithme pseudo-polynomial et après de réduire l'instance du problème à une instance pour laquelle l'algorithme pseudo-polynomial prend un temps polynomial. Finalement en utilisant une solution optimum de cette nouvelle instance on déduit une solution approchée pour l'instance de départ. Cette technique a été utilisée par Ibarra et Kim [IK75] pour obtenir un `fptas` pour KNAPSACK et aussi par Horowitz et Sahni [HS76] et Sahni [S76] pour obtenir des `fptas` pour des problèmes d'ordonnancement.

2.9 Instances particulières des problèmes d'optimisation

Certains problèmes d'optimisation deviennent plus faciles à approcher lorsque l'on se restreint à certaines instances du problème.

Trevisan [T97] a étudié l'approximation de certaines instances de quelques problèmes d'optimisation. Il s'agit des instances pour lesquelles la valeur optimum est la plus grande possible. Par exemple, il donne des algorithmes d'approximation pour les instances satisfaisables de MAX 3SAT et MAX SAT avec un meilleur facteur d'approximation que les meilleurs algorithmes connus pour MAX 3SAT et MAX SAT.

Alimonti et Kann [AK97] ont étudié la complexité de l'approximation de quelques problèmes sur les graphes cubiques. Ils ont montré que MIN VERTEX COVER, MAX CUT, MAX INDEPENDENT SET, MIN DOMINATING SET restent APX-difficiles sur les graphes cubiques.

Jusqu'à il y a peu de temps seulement quelques schémas d'approximation étaient connus : KNAPSACK [IK75] et BIN-PACKING [KK82]. Baker [B94] a montré que plusieurs problèmes NP-difficiles de la théorie des graphes ont un schéma d'approximation efficace pour les graphes planaires : MIN VERTEX COVER, MAX INDEPENDENT SET, MIN DOMINATING SET. Ces résultats ont été étendus par Khanna et Motwani [KM96]. En fait, incité par la caractérisation de la classe APX-PB comme étant l'ensemble des problèmes E-réductibles aux problèmes de $MAX\ SNP$, on a cherché une caractérisation similaire pour $PTAS$. Dans cette direction, Khanna et Motwani [KM96] ont défini plusieurs classes de problèmes de satisfiabilité des contraintes et ils ont montré que les instances planaires des problèmes de ces classes sont dans $PTAS$. De nombreux schémas d'approximation ont été construits une fois que l'étude de la complexité d'approximation des problèmes sur les instances denses a commencé.

2.9.1 Instances denses

Nous présentons dans la suite des résultats d'approximation obtenus pour des instances denses de quelques problèmes d'optimisation NP-difficiles. Certains résultats sont en liaison étroite avec des problèmes étudiés dans le chapitre 6.

Un graphe G est appelé *ε-dense partout* où ε est une constante, $0 < \varepsilon < 1$, si $\delta(G)$ le degré minimum de G est tel que $\delta(G) \geq \varepsilon n$ et *ε-dense* s'il a au moins εn^2 arêtes. Un graphe G d'ordre n est *dense* s'il a $\Theta(n^2)$ arêtes et il est *dense partout* si le degré minimum de G est $\Omega(n)$. De manière similaire, une formule kSAT est *dense* si elle a $\Theta(n^k)$ clauses.

Arora, Karger et Karpinski [AKK95] ont donné les premiers schémas d'approximation pour des problèmes sur les instances denses. Ils ont montré que les instances denses des problèmes de $MAX\ SNP$ ont un schéma d'approximation. Leur idée de base est de représenter les problèmes comme des programmes entiers de degré k d'une certaine forme et puis d'appliquer des résultats généraux d'approximation pour ces programmes.

Définition 2.9.1 *Un* programme entier de degré d lisse de maximisation *est de la forme :*

$$\max p(x_1, \ldots, x_n)$$

$$x_i \in \{0,1\}, i = 1, \ldots, n$$

où $p(x_1, \ldots, x_n)$ est un polynôme de degré d dans lequel les coefficients des monômes de degré i sont $O(n^{d-i})$.

Un *programme entier de degré d lisse de minimisation* est défini de manière similaire.

Max Cut peut être écrit comme un programme entier de degré 2 lisse de maximisation. Étant donné un graphe $G = (V, E)$, on cherche à diviser les sommets en deux ensembles telles que le nombre des arêtes de G dont les extrémités se trouvent dans des parties différentes soit maximum. Pour écrire Max Cut comme un programme entier on introduit une variable binaire x_i pour chaque sommet i. Le problème consiste à attribuer des valeurs 0 ou 1 aux variables x_i telles que la somme suivante soit maximum :

$$\frac{1}{2} \sum_{\{i,j\} \in E} (x_i(1 - x_j) + x_j(1 - x_i)).$$

À partir d'une assignation de 0 et 1 aux variables x_i on construit une solution pour Max Cut ainsi : les sommets correspondant aux variables avec la même valeur font parti du même ensemble de la partition.

En développant la somme précédente on observe que les coefficients des monômes de degré 2 sont des constantes et les coefficients des monômes de degré 1 sont $O(n)$.

Théorème 2.9.2 *Soit « opt » la valeur optimum d'un programme entier de degré d lisse. Pour chaque $\varepsilon > 0$, il existe un algorithme polynomial qui retourne une solution x telle que :*

$$p(x_1, \ldots, x_n) \geq opt - \varepsilon n^d$$

dans le cas d'un programme de maximisation et

$$p(x_1, \ldots, x_n) \leq opt + \varepsilon n^d$$

dans le cas d'un programme de minimisation.

L'idée de l'algorithme du théorème précédent dans le cas $d = 2$ est de réduire l'instance du programme quadratique à un programme linéaire et après d'utiliser la technique d'arrondissage aléatoire de Raghavan et Thompson [RT87] de transformation d'une solution fractionnelle du programme linéaire en une solution entière. Le temps nécessaire est de $n^{O(1/\varepsilon^2)}$. Pour le cas $d > 2$ on réduit de la même manière les programmes de degré d à des programmes de degré $d-1$.

Nous appliquons dans la suite l'algorithme précédent pour obtenir un schéma pour Max Cut. Étant donné un graphe α-dense, la valeur d'une solution optimum *opt* est au moins $\alpha n^2/2$ car la moyenne de la taille d'une coupe obtenue en considérant un sommet d'un coté ou d'un autre avec une probabilité $1/2$ est $\alpha n^2/2$. En appliquant l'algorithme précédent, en temps $n^{O(1/\varepsilon^2\alpha^2)}$, on trouve une solution de valeur au moins

$$opt - \frac{\varepsilon}{1+\varepsilon}\alpha\frac{n^2}{2} \geq \frac{opt}{1+\varepsilon}.$$

Indépendamment, Fernandez de la Vega [F96] a donné un `eptas` pour Max Cut sur les instances denses.

Arora, Karger et Karpinski [AKK95] ont construit des `ptas` pour certains problèmes qui ne font pas partie de $MAX\ SNP$: par exemple pour Min Bisection sur les instances denses partout. Pour ce dernier problème ils ont utilisé des idées supplémentaires.

Le travail sur les instances denses a été continué par Arora, Frieze et Kaplan [AFK96], Frieze et Kannan [FK96], Fernandez de la Vega et Karpinski [FK97], qui, en utilisant de nouvelles méthodes, ont obtenu des schémas d'approximation plus efficaces pour certains des problèmes précédents et des schémas d'approximation pour d'autres problèmes : Min Linear Arrangement, Max Cut avec des poids sur les arêtes.

TSP(1,2) et Longest Path

Dirac a montré qu'un graphe G avec n sommets et de degré minimum $\delta(G) \geq n/2$ est Hamiltonien. Le cycle Hamiltonien peut être trouvé en temps polynomial. Du côté négatif, Dahlhaus, Hajnal et Karpinski ([DHK93]) ont montré que pour chaque $\alpha < 1/2$ le problème de décider si un graphe G α-dense partout est Hamiltonien est NP-complet.

Du point de vue de l'approximation TSP(1,2) [3] est $MAX\ SNP$-hard ([PY93]) et il est approximable à un facteur 2 près trivialement. Le problème Longest Path n'est pas approximable à un facteur constant près, à moins que $P = NP$ ([KMR93]). Une question intéressante est d'étudier si ces deux problèmes ont des

3. Récemment, Engebretsen a montré que TSP(1,2) n'est pas approximable à $4709/4708 - \varepsilon$ près pour chaque $\varepsilon > 0$ à moins que $P = NP$; *ECCC, Report No. 46, 1998.*

meilleurs approximations pour les instances denses. Il est facile de voir que LONGEST PATH est approximable à un facteur constant près sur les graphes α-denses partout, constante qui dépend de α.

Fernandez de la Vega et Karpinski [FK98] et indépendamment Fotakis et Spirakis [FS98] ont montré que TSP(1,2) restreint aux instances où le graphe formé avec les arêtes de poids 1 est α-dense partout et Hamiltonien est $MAX\ SNP$-hard pour chaque $\alpha < 1/2$. Ainsi, LONGEST PATH est $MAX\ SNP$-hard sur les graphes α-denses partout et Hamiltoniens pour chaque $\alpha < 1/2$.

Des résultats très intéressants ont été démontrés sur les instances denses de quelques problèmes de couverture : MIN SET COVER, MIN STEINER TREE, MIN VERTEX COVER. En particulier, dans [KZ96] il est montré que MIN VERTEX COVER s'approxime mieux en temps polynomial sur les instances denses que sur les instances générales : il est approximable à un facteur $\frac{2}{2-\sqrt{1-\alpha}}$ près dans les graphes α-denses et approximable à un facteur $\frac{2}{1+\alpha}$ près dans les graphes α-denses partout. En plus MIN VERTEX COVER ne peut pas avoir de schéma d'approximation sur les instances denses sous l'hypothèse où $P \neq NP$ [CT96],[KZ96]. Dans [K97] on trouve des résultats d'approximation et de non-approximation concernant les instances denses d'autre problèmes d'optimisation : MIN SET COVER, MIN STEINER TREE, MIN BANDWIDTH.

Chapitre 3

Totalité

Certaines problèmes ont toujours une solution. Pour d'autres problèmes on promet d'avoir toujours une solution et on demande à trouver une telle solution. Dans ce chapitre nous nous concentrons sur ces problèmes pour lesquels une solution est garantie pour chaque instance du problème.

3.1 Problèmes de promesse

Un problème de promesse a la structure suivante :

Entrée : x.
Promesse : $Q(x)$.
Question : $R(x)$?
où Q et R sont deux prédicats.

Un algorithme qui résout le problème précédent répond « oui » ou « non » pour les instances x qui satisfont le prédicat Q si le prédicat R est satisfait ou non et donne une réponse arbitraire si le prédicat Q n'est pas satisfait.

Soit $G = (V, E)$ un graphe non-orienté. Le *nombre chromatique* d'un graphe G est le nombre minimum des couleurs nécessaires pour colorier les sommets de G tel que n'importe quel couple de sommets adjacents ont des couleurs différentes. Le *nombre de clique* d'un graphe G est la taille de la plus grande clique de G. Un graphe G est *parfait* si pour tout sous-graphe induit de G, le nombre chromatique est égal au nombre de clique.

Le problème MAX CLIQUE qui consiste à trouver la taille maximale d'une clique dans un graphe est NP-difficile [K72]. Grötschel, Lovász et Schrijver [GLS81] ont montré que le problème de promesse suivant est soluble en temps polynomial :

Entrée : un graphe G non-orienté.
Promesse : G est un graphe parfait.

Sortie : une clique dans G de taille maximale.

Un problème de promesse n'est pas toujours plus facile à résoudre que le problème même. Nous allons voir dans la suite comment définir à partir d'un langage un problème de promesse aussi difficile à résoudre que l'appartenance au langage.

À n'importe quel ensemble récursif A on peut associer naturellement un problème de promesse PP-A comme suit :

Entrée : x et y.
Promesse : exactement un des x et y appartient à A.
Question : $x \in A$?

Selman ([Sel88b, Théorème 3, p. 93]) a montré que pour n'importe quel problème A qui est disjonctif-réductible à un problème de NP qui est disjonctif-auto-réductible, décider PP-A est aussi difficile que décider A. En particulier, le problème PP-SAT est NP-difficile car SAT est un problème disjonctif-auto-réductible.

Considérons dans la suite deux problèmes pour lesquels on promet qu'il y a au plus une solution. La question qu'on se pose est de savoir si ces problèmes sont vraiment plus faciles que les problèmes de décision correspondants.

Soit AT MOST ONE SAT le problème de promesse défini par :

Entrée : une formule ϕ en forme CNF.
Promesse : ϕ a au plus une assignation satisfaisante.
Question : ϕ est-t-elle satisfaisable?

Étant donné un graphe $G = (V, E)$, un automorphisme de G est une fonction $f : V \to V$ telle que $\{i, j\} \in E$ si et seulement si $\{f(i), f(j)\} \in E$. L'automorphisme trivial est la fonction identité. Nous allons considérer dans la suite le problème bien connu dont la complexité est toujours un problème ouvert, de décider si un graphe a un automorphisme non-trivial et une variante de promesse de ce problème.

GA
Entrée : un graphe G.
Question : G admet-t-il un automorphisme non-trivial?

AT MOST ONE GA
Entrée : un graphe G.
Promesse : G a au plus un automorphisme non-trivial.
Question : G a-t-il un automorphisme non-trivial?

Le problème GA est Turing-réductible au problème At Most One GA ([KST93, Théorème 1.31, p. 41]). Donc, si on pouvait résoudre At Most One GA en temps polynomial, on pourrait résoudre GA en temps polynomial.

La situation est un peu différente pour le problème Sat : une Turing-réduction de Sat au problème At Most One Sat n'est pas connue mais Sat est réductible aléatoirement à At Most One Sat ([VV86] ou [KST93, Section 1.4.1]). La réduction aléatoire précédente transforme une formule booléenne F dans une autre F' telle que si F est non-satisfaite alors F' non plus et si F est satisfaite alors F' a une assignation unique avec une probabilité plus grande que $1/poly(|F|)$. Ainsi, si At Most One Sat $\in P$ (ou même dans RP) alors $NP \subseteq RP$.

Donc, la promesse qu'une instance d'un problème a au plus une solution ne rend pas forcement plus facile le problème de décider s'il y a une solution.

Nous allons considérer dans un chapitre ultérieur la complexité de l'approximation d'un problème constructif qui est un problème de promesse : trouver dans un graphe cubique et *Hamiltonien* un plus long cycle.

3.2 Les classes FP, FNP et $TFNP$

Nous avons vu qu'un problème de décision de NP est caractérisé par une relation polynomiale R (définition 2.1.2). Nous donnons ensuite la définition d'un problème de recherche associé à un problème de décision.

Définition 3.2.1 (Problème de recherche associé à un problème de décision) *Soit le problème de décision suivant : étant donnée une instance x décider s'il existe une solution y telle que $R(x,y)$.* Le problème de recherche associé à ce problème de décision *est le suivant : étant donnée une instance x* trouver *une solution y telle que $R(x,y)$ si une telle solution existe, sinon répondre « non ».*

On dit que z est une *réponse correcte* à une instance x d'un problème de recherche caractérisé par une relation polynomiale R si z est « non » s'il n'existe pas de y qui est en relation R avec x ou $R(x,z)$ sinon.

Une *relation* polynomiale R est *totale* si pour n'importe quel x il existe un y tel que $R(x,y)$.

Un problème caractérisé par une relation totale est appelé *problème total.*

Définition 3.2.2 (FP, FNP, TFNP, F(NP $\cap$ *co*-NP))

FNP (l'abréviation de « Fonctional NP *») est la classe des problèmes de recherche associés aux problèmes de décision de NP.*

$TFNP$ est la sous-classe des problèmes de FNP caractérisés par une relation totale.

FP est la sous-classe des problèmes de FNP solubles en temps polynomial.

$F(NP \cap co\text{-}NP)$ est la classe des problèmes de recherche associés aux problèmes de décision de $NP \cap co\text{-}NP$.

Soit un problème de décision $L \in NP \cap co\text{-}NP$. Alors il existe deux relations polynomiales R_1, R_2 telles que : si $x \in L$ alors il existe un y tel que $R_1(x, y)$ et si $x \notin L$ alors il existe un z tel que $R_2(x, z)$. Le problème de recherche associé à L consiste à trouver pour une instance $x \in L$ un y tel que $R_1(x, y)$ et pour une instance $x \notin L$ un z tel que $R_2(x, z)$.

La classe $TFNP$ a été introduite par Meggido et Papadimitriou [MP91]. Un exemple bien connu de problème de $TFNP$ est de trouver la décomposition d'un entier en nombres premiers, problème pour lequel on ne connaît pas d'algorithme polynomial :

FACTORING
Entrée : un entier N.
Sortie : la décomposition en nombres premiers de N, $N = p_1^{k_1} \dots p_n^{k_n}$ avec une preuve de primalité de $p_1, \dots, p_n$.

Meggido et Papadimitriou ont introduit une réduction entre deux problèmes de recherche correspondant à la réduction Karp entre deux problèmes de décision.

Définition 3.2.3 (F-réduction) *Soient A et B deux problèmes de recherche. On dit que A est F-réductible à B et on écrit $A \leq_F B$ s'il existe deux fonctions f, g calculables en temps polynomial telles que :*

1. *Pour chaque instance x de A, $f(x)$ est une instance de B ;*
2. *Pour chaque réponse correcte y de $f(x)$, $g(x, y)$ est une réponse correcte de x.*

La Karp-réduction de Cook qui montre que SAT est NP-complet est aussi une F-réduction et donc FSAT est FNP-complet. En utilisant l'équivalence polynomiale entre SAT et FSAT on peut déduire que :

Théorème 3.2.4 ([Pap94b]) *FP=FNP si et seulement si P=NP.*

Est-ce que $TFNP$ pourrait coïncider avec FP ou avec FNP? Megiddo et Papadimitriou ont étudié les conséquences d'une réponse affirmative à cette question [MP91]. Ils ont donné d'abord une caractérisation de la classe $TFNP$ en utilisant la classe $F(NP \cap co\text{-}NP)$:

Lemme 3.2.5 ([MP91, p. 318]) *$TFNP$=$F(NP \cap co\text{-}NP)$.*

Preuve. Soit $A \in TFNP$. Comme $TFNP \subseteq FNP$, il existe une relation polynomiale R telle que pour chaque instance x de A il existe un témoin court y tel que $R(x, y)$. Pour montrer que $A \in F(NP \cap co\text{-}NP)$ on prend $R_1 = R$ et $R_2 = \emptyset$.

Considérons maintenant un problème $A \in F(NP \cap co\text{-}NP)$ caractérisé par deux relations polynomiales R_1, R_2. La relation polynomiale totale qui montre que $A \in TFNP$ est $R = R_1 \cup R_2$. □

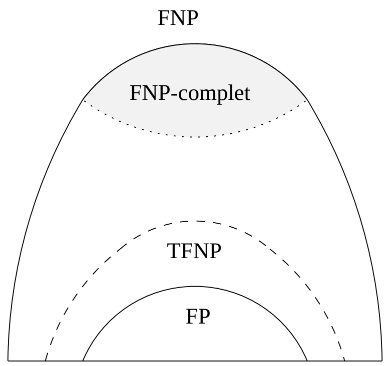


FIG. 3.1 – *Les 3 classes dans l'hypothèse où $NP \neq co\text{-}NP$ et $P \neq NP \cap co\text{-}NP$*

Comme $P \subseteq NP \cap co\text{-}NP$ nous avons que

$$FP \subseteq TFNP$$

et donc

$$FP \subseteq TFNP \subseteq FNP.$$

Est-ce qu'un problème total peut être FNP-difficile sous la F-réduction? Intuitivement, il ne peut pas exister une F-réduction du problème FSAT à un problème total, car il n'existe pas de moyen de distinguer les instances positives et négatives de FSAT en utilisant ce problème total.

En utilisant le lemme précédent il est facile de montrer le théorème suivant.

Théorème 3.2.6 ([MP91]) *Si $NP \neq co\text{-}NP$ et $P \neq NP \cap co\text{-}NP$ il existe un problème de $TFNP$ qui n'est pas dans FP et aucun problème de $TFNP$ est FNP-difficile.*

Preuve. Si tout problème de $TFNP$ est dans FP alors $TFNP = FP$ et donc $P = NP \cap co\text{-}NP$.

Considérons qu'il existe un problème $A \in TFNP$ qui est FNP-difficile. Alors il existe une F-réduction entre FSAT et A caractérisée par deux fonctions polynomiales f et g. Nous allons montrer que FSAT $\in F(co\text{-}NP)$. Soit ϕ une instance non-satisfaisable de FSAT. La fonction f transforme ϕ en une instance $f(\phi)$ de A. Chaque réponse correcte y de $f(\phi)$ est transformée par g en une réponse correcte $g(\phi, y)$ de ϕ. Comme ϕ est non-satisfaisable, cette réponse est « non » et y est un témoin court que ϕ est non-satisfaisable. Ainsi, $NP = co\text{-}NP$. □

La figure 3.1 illustre les inclusions entre les 3 classes FP, FNP, $TFNP$ dans l'hypothèse où $NP \neq co\text{-}NP$ et $P \neq NP \cap co\text{-}NP$.

3.3 Quelques sous-classes de $TFNP$

Selman [Sel88a] a montré que pour tous les problèmes NP-complets, le problème de décision et le problème de recherche associé sont polynomialement équivalents. Pourtant, en toute généralité, il est possible que le problème de décision et le problème de recherche ne soient pas polynomialement équivalents. C'est le cas, par exemple lorsque, l'existence d'une solution est garantie : le problème de décision est alors trivial, néanmoins on ne connaît pas toujours d'algorithme polynomial pour le problème de recherche.

Meggido et Papadimitriou [MP91] ont introduit certaines sous-classes de $TFNP$ qui regroupent les problèmes en fonction de leur preuve d'existence d'une solution. En général, à partir d'une instance d'un problème de ces classes on construit un graphe d'ordre exponentiel par rapport à la taille du problème et une solution pour l'instance est n'importe quel sommet du graphe qui satisfait une certaine propriété. Les sous-classes de $TFNP$ sont définies en fonction du type du graphe obtenu et la propriété qu'une solution satisfait.

Nous définissons d'abord une classe plus générale et après les sous-classes de $TFNP$ qui nous intéressent.

Définition 3.3.1 *La classe* **Graphe** *contient les problèmes de recherche A pour lesquels il existe un polynôme p, une famille de fonctions f et une propriété P tels que :*

- pour toute instance x de A, $f_x : U_x \to 2^{U_x}$ où $U_x = \{0,1\}^{p(|x|)}$ et l'ensemble $f_x(y)$ est reconnaissable en temps polynomial. Notons par $G_A(x)$ le graphe orienté avec l'ensemble de sommets U_x et l'ensemble d'arcs $E_x = \{\{y, z\} : z \in f_x(y), y \in U_x\}$;

- la propriété P est décidable en temps polynomial ;

- pour toute instance x de A, $y \in Sol(x)$ si et seulement si $P(x, y)$ est satisfaite.

3.3.1 La classe PLS

PLS est une abréviation de l'appellation anglaise « **P**olynomial **L**ocal **S**earch ». Cette classe a été introduite et étudiée dans [JPY88], [PSY90], [SY91], [K89].

Définition 3.3.2 (PLS) *Un problème A de la classe PLS est caractérisé par un ensemble d'instances. Pour chaque instance x :*

- est associé un ensemble de solutions possibles $Sol(x)$ qui est reconnaissable en temps polynomial ;

- chaque solution $y \in Sol(x)$ a une valeur associée calculable en temps polynomial ;

- il est possible de trouver en temps polynomial une solution initiale $y_0 \in Sol(x)$;

- à chaque solution $y \in Sol(x)$ est associé un sous-ensemble $V_x(y)$ de solutions possibles appelé le voisinage *de y qui est reconnaissable en temps polynomial ;*

- pour chaque solution $y \in Sol(x)$ on peut tester en temps polynomial si y est un optimum local dans son voisinage et sinon produire une solution du voisinage de y avec une meilleure valeur.

Pour une instance donnée on essaie de trouver un optimum local par rapport au voisinage fixé, c'est-à-dire une solution qui a la meilleure valeur dans son voisinage.

Théorème 3.3.3 *PLS est l'ensemble des problèmes A de* **Graphe** *avec la propriété que pour toute instance x de A :*

- $G_A(x)$ est sans circuit ;

- $P(x, y)$ est satisfaite si et seulement si y est un sommet de $G_A(x)$ avec le degré sortant 0.

Preuve. Soit $A \in PLS$. Comme l'ensemble $Sol(x)$ des solutions de l'instance x est polynomialement reconnaissable, il existe un polynôme p tel que $|y| \leq p(|x|)$. Nous prenons $U_x = \{0,1\}^{p(|x|)}$ et nous définissons

$$f_x(y) = \{z : z \in V_x(y) \text{ et } z \text{ a une meilleur valeur que } y\}.$$

Il est facile de voir que pour chaque instance x, le graphe $G_A(x)$ n'a pas de circuit. Une solution est un optimum local par rapport au voisinage fixé si et seulement si elle correspond à un sommet y de $G_A(x)$ avec le degré sortant 0.

Soit A un problème de la classe **Graphe** avec la propriété que $G_A(x)$ pour chaque instance x est un graphe orienté sans circuit et $P(x, y)$ est satisfait si y est un sommet de degré sortant 0 dans $G_A(x)$. Pour chaque instance x nous définissons $Sol(x) = \{0,1\}^{p(|x|)}$ et considérons comme solution initiale y_0 un élément quelconque de $Sol(x)$. Le voisinage d'une solution y est l'ensemble $f_x(y)$ et nous considérons que les voisins de y ont une meilleure valeur que y.

Ainsi, pour une solution y, $P(x, y)$ est satisfaite si et seulement si y est un optimum local par rapport au voisinage fixé. □

Le problème HAPPYNET de Papadimitriou [Pap94a] est un exemple de problème de PLS :

HAPPYNET

Entrée : un graphe $G = (V, E)$ où l'ensemble des sommets $V = \{1, \ldots, n\}$ et une fonction w qui associe des poids entiers relatifs aux arêtes de G.

Sortie : une assignation $x \in \{-1, 1\}^n$, où x_i est la valeur assignée au sommet i de G, telle que pour chaque sommet i l'inégalité suivante ait lieu :

$$x_i \cdot \sum_{\{i,j\} \in E} x_j w(\{i, j\}) \geq 0.$$

Un sommet i pour lequel

$$x_i \cdot \sum_{\{i,j\}\in E} x_j w(\{i,j\}) \geq 0$$

est appelé sommet « *heureux* » *en* x.

Théorème 3.3.4 (Hopfield [Hop82],Papadimitriou [Pap94a, p. 230])
HAPPYNET $\in PLS$.

Preuve. En fait, ils ont montré que trouver une solution pour HAPPYNET est équivalent à trouver un *optimum local* (par rapport au voisinage qui contient pour une assignation y tous les vecteurs qui se trouvent à la distance de Hamming 1 de y) pour le problème suivant :

MAX SUM
Entrée : un graphe $G = (V, E)$ où l'ensemble des sommets $V = \{1, \ldots, n\}$ et une fonction w qui associe des poids entiers relatifs aux arêtes de G.
Sortie : une assignation $x \in \{-1, 1\}^n$, où x_i est la valeur assignée au sommet i de G, telle que la somme suivante soit maximum :

$$S(x) = \sum_{\{i,j\}\in E} x_i x_j w(\{i,j\}).$$

Nous présentons dans la suite cette équivalence.

Supposons que x ne soit pas une solution pour l'instance G de HAPPYNET, c'est-à-dire qu'il existe un sommet i qui n'est pas heureux en x. Donc,

$$x_i \cdot \sum_{\{i,j\}\in E} x_j w(\{i,j\}) = -\delta \text{ où } \delta > 0.$$

Soit x' l'assignation définie à partir de x ainsi : $x'_j = x_j$ si $j \neq i$ et $x'_i = \bar{x}_i$ où $\bar{x}_i$ est la valeur opposée de x_i. Plus précisément, $\bar{x}_i = 1$ si $x_i = -1$ et $\bar{x}_i = -1$ si $x_i = 1$. C'est facile de voir que $S(x') = S(x) + 2\delta > S(x)$. Donc, x n'est pas un optimum local par rapport à son voisinage.

Supposons que x ne soit pas un optimum local pour l'instance G de MAX SUM, c'est-à-dire qu'il existe une assignation x' dans le voisinage de x telle que $S(x') > S(x)$. Comme x' est dans le voisinage de x il existe un sommet i tel que $x'_j = x_j$ si $j \neq i$ et $x'_i = \bar{x}_i$. Comme avant, il est facile de montrer que i n'est pas heureux en x et donc x n'est pas une solution pour l'instance G de HAPPYNET.

Donc x est une solution pour l'instance G de HAPPYNET si et seulement si x est un optimum local pour S, c'est-à-dire que la fonction S prend la valeur la plus grande pour x parmi les solutions du voisinage de x.

□

HAPPYNET peut être résolu ainsi : on part d'une assignation arbitraire et on construit une autre en changeant la valeur d'un sommet qui n'est pas heureux. On répète cette procédure jusqu'à ce que tous les sommets deviennent heureux. Dans le pire des cas cet algorithme exécute la procédure $O(W)$ fois, où

$$W = \sum_{\{i,j\} \in E} |w(\{i,j\})|.$$

Définition 3.3.5 (**PLS-réduction**) *Étant donnés deux problèmes A et B, A est PLS-*réductible *à B s'il existe deux fonctions calculables en temps polynomial f et g telles que :*
- chaque instance x de A est transformée en une instance $f(x)$ de B ;
- la fonction g transforme un optimum local de l'instance $f(x)$ de B en un optimum local de l'instance x de A.

HAPPYNET est un problème PLS-complet sous la PLS-réduction [PSY90] et ainsi au moins aussi difficile que tout problème de PLS. Le problème de programmation linéaire LINEAR PROGRAMMING avec le voisinage de simplex est dans PLS car pour le simplex un optimum local coïncide avec un optimum global. Donc, un algorithme polynomial pour un problème PLS-complet donnerait en particulier une autre preuve que LINEAR PROGRAMMING est dans P. Plusieurs autres problèmes ont été montrés PLS-complets [K89], [JPY88], [PSY90].

Évidemment la variante de HAPPYNET où tous les poids sont -1 est soluble en temps polynomial et elle a été montrée P-complète sous la *logspace*-réduction [PSY90].

Nous présentons ensuite un algorithme simple d'approximation pour HAPPYNET.

Théorème 3.3.6 *Il existe un algorithme probabiliste qui donne en temps polynomial une approximation à un facteur 2 près pour* HAPPYNET.

Preuve. L'algorithme probabiliste est le suivant : on considère $x_i = 1$ avec la probabilité $1/2$ et $x_i = -1$ avec la probabilité $1/2$.

Nous montrons ensuite que l'algorithme précédent retourne une solution telle qu'en moyenne la moitié des sommets sont heureux.

Supposons i fixé. Nous montrons que la somme

$$S_i(x) = \sum_{\{i,j\} \in E} x_j w(\{i,j\})$$

prend des valeurs positives avec la probabilité $1/2$. Pour une assignation donnée x, notons par $\bar{x}_i$ la valeur opposée de x_i. Notons par $x^{(i)}$ l'assignation suivante :

$x_j^{(i)} = x_j$ si $\{i,j\} \notin E$ et $x_j^{(i)} = \bar{x}_j$ si $\{i,j\} \in E$. En fait $x^{(i)}$ est l'assignation x dans laquelle les voisins de i ont changé de valeur. C'est facile de voir que $S_i(x^{(i)}) = -S_i(x)$ et que les deux assignations x et $x^{(i)}$ apparaissent avec la même probabilité dans l'algorithme précédent.

Notons par Y_i la variable aléatoire qui prend la valeur 1 si i est heureux et 0 sinon. Posons :

$$Y = \sum_{i=1}^{n} Y_i.$$

La probabilité que i soit heureux est $P(Y_i = 1) = 1/2$ et la moyenne de Y_i est $E[Y_i] = 1/2$. Ainsi, la moyenne du nombre des sommets heureux est $E[Y] = n/2$. □

Remarque. On peut montrer que pour chaque $\varepsilon > 0$,

$$P(Y \geq \frac{n}{2+\varepsilon}) > \frac{\varepsilon}{2(1+\varepsilon)}.$$

Soit $\varepsilon > 0$ fixé. Notons $P(Y \geq \frac{n}{2+\varepsilon}) = p$.

$$\frac{n}{2} = E[Y] \leq \frac{n}{2+\varepsilon}(1-p) + np \Rightarrow p > \frac{\varepsilon}{2(1+\varepsilon)}.$$

On peut répéter l'algorithme précédent jusqu'à ce qu'il retourne une assignation qui rende plus de la moitié des sommets heureux. On obtient ainsi un algorithme probabiliste Las Vegas car il donne toujours une réponse correcte.

3.3.2 La classe PPP

PPP est l'abréviation de « **P**olynomial **P**igeonhole **P**rinciple ».

Papadimitriou [Pap94b] a donné une définition de la classe PPP en utilisant une machine de Turing :

Soit M une machine de Turing déterministe à deux rubans : sur le premier ruban où on peut seulement lire se trouve l'entrée, le deuxième est un ruban de travail où on peut écrire et lire. Supposons que M travaille en temps borné par un polynôme $p(|x|)$ par rapport à la taille de l'entrée $|x|$. Soit $S(x)$ l'ensemble des configurations possibles du ruban de travail quand la machine M a x comme entrée. Notons par $M(x,y)$ la configuration du ruban de travail quand la machine M s'arrête à partir de l'entrée x et ayant y au départ sur le ruban de travail.

Définition 3.3.7 (PPP) *Un problème $A \subseteq \{0,1\}^* \times (\{0,1\}^* \cup \{0,1\}^* \times \{0,1\}^*)$ est dans PPP s'il existe une machine de Turing M décrite précédemment telle que :*

$$A = \{(x,y) : y \in S(x), M(x,y) = 0^{p(|x|)}\} \cup$$
$$\{(x,(y,y')) : y,y' \in S(x), y \neq y', M(x,y) = M(x,y')\}.$$

Le lemme utilisé pour montrer l'existence d'une solution pour chaque instance d'un problème de cette classe est *le principe des tiroirs*.

Théorème 3.3.8 *PPP est l'ensemble des problèmes A de* **Graphe** *avec la propriété que pour toute instance x de A :*

- $G_A(x)$ est un graphe biparti (L_x, R_x), $L_x = \{0,1\}^{p(|x|)-1}1$, $R_x = \{0,1\}^{p(|x|)-1}0$, les arcs sont orientés de L_x à R_x et chaque sommet de L_x a pour degré 1 ;

- $P(x,y)$ est satisfaite s'il existe un arc dans $G_A(x)$ de y à $0^{p(|x|)}$ et $P(x,y_1,y_2)$ est satisfaite s'il existe un sommet $z \in R_x$ tel que les arcs qui partent de y_1 et y_2 arrivent à z.

Preuve. Soit $A \in PPP$. Alors il existe une machine de Turing M décrite cidessus et un polynôme q qui est une borne supérieure de la taille d'une configuration possible du ruban de travail de M. Prenons $p(n) = q(n) + 1$, plus précisément pour chaque instance x de A, $U_x = \{0,1\}^{q(|x|)+1}$, où $L_x = \{0,1\}^{q(|x|)}1$ et $R_x = \{0,1\}^{q(|x|)}0$. Pour chaque $y \in L_x$, $y = y'1$, considérons $f_x(y) = \{M(x,y')0\}$. Il est facile de voir que pour chaque instance x de A, $G_A(x)$ a les propriétés souhaités et y ou (y_1, y_2) est une solution de x si et seulement si $P(x, y1)$ ou $P(x, y_1 1, y_2 1)$ est satisfaite.

Soit A un problème de **Graphe** qui satisfait les propriétés du théorème. La machine de Turing M qui montre l'appartenance de A à PPP est définie ainsi : pour toute instance x, $M(x,y) = z$ si et seulement si $f_x(y1) = \{z0\}$. □

Dans la suite nous allons voir quelques exemples de problèmes de PPP :

PIGEONHOLE CIRCUIT

Entrée : un circuit booléen C avec n entrées, n sorties et avec des portes *and*, *or* (de degré intérieur 2) et *not* (de degré intérieur 1).

Sortie : soit une entrée $x \in \{0,1\}^n$ telle que $C(x) = 0^n$, soit deux entrées différentes $x, x' \in \{0,1\}^n$ telles que $C(x) = C(x')$.

PIGEONHOLE SUBSET-SUM

Entrée : un ensemble $\{a_1, \ldots, a_n\}$ d'entiers positifs tels que

$$\sum_{i=1}^{n} a_i < 2^n - 1.$$

Sortie : deux sous-ensembles disjoints et non-vides $S_1, S_2 \subseteq \{1, \ldots, n\}$ tels que

$$\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i \ .$$

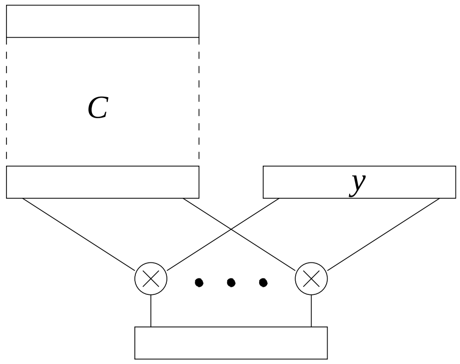


FIG. 3.2 – *Le circuit* C'

PIGEONHOLE 3CNF
Entrée : n formules $3CNF$ sur n variables.
Sortie : soit une assignation qui satisfait toutes les formules ou bien deux assignations distinctes qui satisfont les mêmes formules.

Nous justifions dans la suite le fait que ces exemples sont dans PPP.

La machine de Turing qui montre l'appartenance de PIGEONHOLE SUBSET-SUM à la classe PPP contient sur le ruban d'entrée le codage en binaire des n nombres entiers et sur le ruban de travail le vecteur caractéristique d'un sous-ensemble de ces n nombres. La machine de Turing M calcule en temps polynomial la somme des nombres de ce sous-ensemble et écrit le codage en binaire de cette somme sur le ruban de travail. Comme cette somme ne dépasse pas $2^n - 1$ elle peut être écrite sur n bits. Deux configurations différentes qui représentent deux sous-ensembles non-vides et différents de ces n nombres sont une solution du problème.

Pour montrer que PIGEONHOLE CIRCUIT est dans PPP on considère une machine de Turing qui contient sur le ruban d'entrée le codage en binaire du circuit booléen et sur le ruban de travail une chaîne de n bits qui représente une entrée possible pour le circuit booléen. La machine de Turing calcule la sortie du circuit sur cette entrée, sortie qui est écrite aussi sur le ruban de travail. Évidemment deux configurations différentes représentent deux entrées différentes qui donnent la même sortie.

L'appartenance de PIGEONHOLE 3CNF à PPP se montre de manière similaire à l'appartenance de PIGEONHOLE CIRCUIT à PPP.

Papadimitriou a observé [Pap94b] que cette classe ne contient probablement pas que des problèmes solubles en temps polynomial.

Définition 3.3.9 *Une* permutation à sens unique *est une fonction calculable en temps polynomial* $\pi : \Sigma^n \to \Sigma^n$ *telle que :*
- π *est injective ;*
- π^{-1}*, l'inverse de* π*, n'est pas polynomial calculable.*

Théorème 3.3.10 ([Pap94b, Proposition 3, p.529]) *Si* $PPP = FP$*, alors il n'existe pas de permutation à sens unique.*

Preuve. Supposons que $PPP = FP$. En utilisant le fait que PIGEONHOLE CIRCUIT $\in FP$, nous montrons qu'on peut trouver l'inverse de toute fonction polynomiale et injective $\pi : \Sigma^n \to \Sigma^n$. Étant donné y un élément de l'image de π, comment peut-on trouver un x dans le domaine de π tel que $\pi(x) = y$? Comme π est polynomialement calculable, il existe un circuit C avec $|y|$ entrées et $|y|$ sorties qui la calcule. Nous appliquons l'algorithme polynomial de PIGEONHOLE CIRCUIT sur le circuit C' suivant (voir la figure 3.2) : l'entrée est l'entrée du circuit C de taille $|y|$ et la sortie est le vecteur qui est le or exclusif entre la sortie de C et le vecteur y. Comme π est injective, l'algorithme ne peut pas retourner deux entrées qui donnent la même sortie et donc il va retourner un x tel que $C'(x) = 0^{|y|}$. Cela veut dire que $C(x) = y$ et donc $\pi(x) = y$. □

Nous allons voir dans la suite la conséquence de l'inexistence d'une permutation à sens unique.

Définition 3.3.11 *Une* machine de Turing *non-déterministe est appelée* non-ambiguë *si pour chaque entrée il existe au plus un chemin qui accepte. La classe* UP *est la classe des langages acceptés par des machines de Turing non-ambiguë qui travaillent en temps polynomial.*

Évidemment on a l'inclusion :

$$P \subseteq UP \subseteq NP.$$

Théorème 3.3.12 ([Pap94a, Théorème 12.1, p. 283]) $UP = P$ *si et seulement s'il n'existe pas de permutation à sens unique.*

Papadimitriou a remarqué [Pap94b] que PIGEONHOLE CIRCUIT est PPP-complet sous la F-réduction et que le problème devient polynomial si on considère en entrée des circuits monotones. Donc PIGEONHOLE CIRCUIT est un des plus difficiles problèmes de la classe PPP et si $P \neq UP$ alors PIGEONHOLE CIRCUIT n'est pas soluble en temps polynomial.

3.3.3 La classe PPA

PPA vient de l'anglais « **P**olynomial **P**arity **A**rgument ».

Soit M une machine de Turing déterministe telle que pour chaque instance x et chaîne $y \in \{0,1\}^{p(|x|)}$, elle retourne un sous-ensemble $M(x,y)$ de $\{0,1\}^{p(|x|)}$ reconnaissable en temps polynomial. En plus $M(x, 0^{p(|x|)}) = \{1^{p(|x|)}\}$ et $0^{p(|x|)} \in M(x, 1^{p(|x|)})$.

À partir de M, on définit un graphe $G_M(x)$ ainsi : les sommets du graphe sont les chaînes $\{0,1\}^{p(|x|)}$ et deux sommets y_1 et y_2 sont liés dans $G_M(x)$ si $y_2 \in M(x, y_1)$ et $y_1 \in M(x, y_2)$.

Définition 3.3.13 (PPA) *Un problème A est dans PPA s'il existe une machine de Turing M comme avant telle que pour chaque instance x de A trouver une solution pour x consiste à trouver un sommet de degré impair dans $G_M(x)$ différent de $0^{p(|x|)}$.*

Un *graphe orienté* $G = (V, E)$ est *symétrique* si et seulement si pour tout arc $\{x, y\} \in E$, l'arc $\{y, x\} \in E$. Le degré d'un sommet d'un graphe orienté est le nombre d'arcs incidents.

Théorème 3.3.14 *PPA est l'ensemble des problèmes A de* **Graphe** *avec la propriété que pour toute instance x de A :*

- $G_A(x)$ est un graphe symétrique ;

- soit y_0 un sommet de $G_A(x)$ avec le degré $2(2k+1)$ où $k \geq 0$. $P(x,y)$ est satisfaite si y est un sommet de $G_A(x)$ différent de y_0 de degré $2(2t+1)$ pour un certain $t \geq 0$.

Preuve. Soit $A \in PPA$ et soit M une machine de Turing qui montre l'appartenance de A à cette classe. Nous prenons pour chaque instance x, $U_x = \{0,1\}^{p(|x|)}$ et $y_0 = 0^{p(|x|)}$. Nous définissons pour chaque $y \in \{0,1\}^{p(|x|)}$,

$$f_x(y) = \{z \in M(x,y) : y \in M(x,z)\}.$$

Ainsi y_0 est un sommet de $G_A(x)$ de degré 2 et $G_A(x)$ est un graphe orienté symétrique. Un sommet a le degré impair dans $G_M(x)$ si et seulement si il a le degré $2(2t+1)$ dans $G_A(x)$ pour un certain $t \geq 0$.

Soit A un problème de **Graphe** tel que à chaque instance x est associé un graphe $G_A(x)$ qui satisfait les conditions du théorème. Cela veut dire que $G_A(x)$ a le sommet y_0 de degré $2(2k+1)$ pour un certain $k \geq 0$ et n'importe quel autre sommet du graphe de degré $2(2t+1)$ pour un certain $t \geq 0$ est une solution de l'instance x. Nous définissons une machine de Turing M qui montre que $A \in PPA$:

$$M(x, 0^{p(|x|)+2}) = \{1^{p(|x|)+2}\}$$
$$M(x, 1^{p(|x|)+2}) = \{0^{p(|x|)+2}, y_0 01\}$$

$$M(x, y_0 01) = \{1^{p(|x|)+2}, y01 : y \in f_x(y_0)\}$$

$$M(x, y01) = \{z01 : z \in f_x(y)\}$$

$$M(x, y) = \emptyset, \text{ pour les autres } y.$$

Un sommet de degré impair dans $G_M(x)$ coïncide avec un sommet de degré $2(2t + 1)$ pour un certain $t \geq 0$ dans $G_A(x)$. □

Nous allons voir dans la suite quelques exemples de problèmes de cette classe. Nous commençons par donner quelques définitions sur les graphes.

Soit G un graphe non-orienté. Un *chemin* de longueur k dans G est une séquence de sommets distincts $v_0, v_1, \ldots, v_k$ tels que pour $0 \leq i \leq k-1$, il existe une arête entre v_i et v_{i+1}. Un chemin de longueur au moins trois pour lequel le premier et le dernier sommet coïncident est appelé *cycle*. Un chemin *couvre* un sous-graphe H s'il contient tous les sommets de H. Un *chemin* ou un *cycle* est *Hamiltonien* s'il couvre G. Un *graphe* est appelé *Hamiltonien* s'il a un cycle Hamiltonien. Un *graphe* est appelé *cubique* si le degré de tous ses sommets est trois.

Notons par $\bar{G}$ le complémentaire du graphe G et par $h(G)$ le nombre des chemins Hamiltoniens de G. Lovász a montré le résultat suivant ([Lov79, Exercice 5.19, p. 37]) : dans un graphe G le nombre $h(G) + h(\bar{G})$ est un nombre pair.

Donc le problème suivant est un problème total :

Another Hamiltonian Path
Entrée : un graphe G et un chemin Hamiltonien P dans G.
Sortie : soit un autre chemin Hamiltonien de G différent de P, soit un chemin Hamiltonien dans le complémentaire de G.

Papadimitriou a montré en plus ([Pap94b, Théorème 10, p. 516]) que ce problème est dans PPA. Il a construit à partir du graphe G un graphe biparti de taille exponentielle par rapport au nombre des sommets de G dont les sommets de degré impair coïncident avec les chemins Hamiltoniens de G et $\bar{G}$.

Il n'existe pas d'algorithme polynomial connu pour résoudre le problème précédent.

Smith a montré qu'un graphe avec des sommets de degré impair a un nombre pair de cycles Hamiltoniens qui contiennent une certaine arête [T46]. Nous décrivons dans la suite la preuve pour les graphes cubiques.

Théorème 3.3.15 ([T46]) *Dans un graphe cubique il existe un nombre pair de cycles Hamiltoniens utilisant une arête arbitraire donnée.*

Preuve. Soit $G = (V, E)$ un graphe cubique et Hamiltonien. Notons par n le nombre des sommets de G et considérons que $E = \{e_1, \ldots, e_m\}$ où $m = 3n/2$. Un cycle Hamiltonien de G est de longueur paire et donc on peut colorier ses arêtes alternativement avec deux couleurs 1 et 2, et on peut utiliser une troisième couleur, par exemple la couleur 3 pour toutes les autres arêtes. Notons par T_i, $1 \leq i \leq 3$ les ensembles d'arêtes coloriées avec la couleur i d'une 3 coloration des arêtes de G.

Une coloration des arêtes avec les 3 couleurs 1, 2 et 3 détermine trois vecteurs $\mu_{12}, \mu_{23}, \mu_{13}$ de taille m ainsi définis : $\mu_{ij} = (\mu_{ij}^1, \ldots, \mu_{ij}^m)$ où $\mu_{ij}^k = 1$ si l'arête e_k est de couleur i ou j et $\mu_{ij}^k = 0$ sinon.

Soit M l'ensemble des vecteurs μ de taille m qui définissent une famille de cycles pairs disjoints deux à deux et utilisant tous les sommets.

Pour chaque coloration avec les ensembles T_1, T_2, T_3, les vecteurs $\mu_{12}, \mu_{23}, \mu_{13} \in M$ et

$$\mu_{12} + \mu_{23} + \mu_{13} \equiv 0 (\text{mod } 2).$$

Un vecteur $\mu \in M$ qui consiste en une famille de $k(\mu)$ cycles pairs disjoints correspond à $2^{k(\mu)-1}$ colorations distincts car on peut fixer la coloration d'un des $k(\mu)$ cycles et considerer toutes les combinaisons des colorations pour les autres cycles.

Donc, en sommant l'identité précedente par rapport à toutes les 3 colorations on obtient :

$$\sum_{\mu \in M} 2^{k(\mu)-1} \mu \equiv 0 (\text{mod } 2).$$

On peut conclure que le nombre de cycles Hamiltoniens utilisant une arête arbitraire donnée est un nombre pair. □

Le problème suivant est donc total : étant donné un graphe avec des sommets de degré impair et un cycle Hamiltonien C, trouver un autre cycle Hamiltonien différent de C. La preuve de Smith n'est pas constructive. Thomason [Tho78] a donné un algorithme simple et élégant pour trouver un deuxième cycle Hamiltonien étant donné le premier.

Utilisant l'idée de Thomason, Papadimitriou a montré ([Pap94b, p. 501]) que le problème précédent est dans PPA. Pour cela, à partir du graphe G, il construit un graphe $H(G)$ dont les sommets de degré impair coïncident avec les cycles Hamiltoniens de G qui contiennent l'arête fixée. Dans la figure 3.3 est illustré le graphe $H(G)$ correspondant au graphe cubique G et l'arête $\{x, y\}$. Dans le cas d'un graphe cubique, le graphe H sera un chemin. On fixe une extrémité, par exemple x. Pour décrire plus facilement la construction de H, on considère que les chemins Hamiltoniens sont orientés de x à l'autre extrémité. Chaque sommet de H correspond à un chemin Hamiltonien de G à partir de x. À chaque étape on considère le voisin $v(z)$ de l'autre extrémité du chemin Hamiltonien (soit z) qui n'était pas liée à z sur le précédent chemin Hamiltonien. À partir du chemin

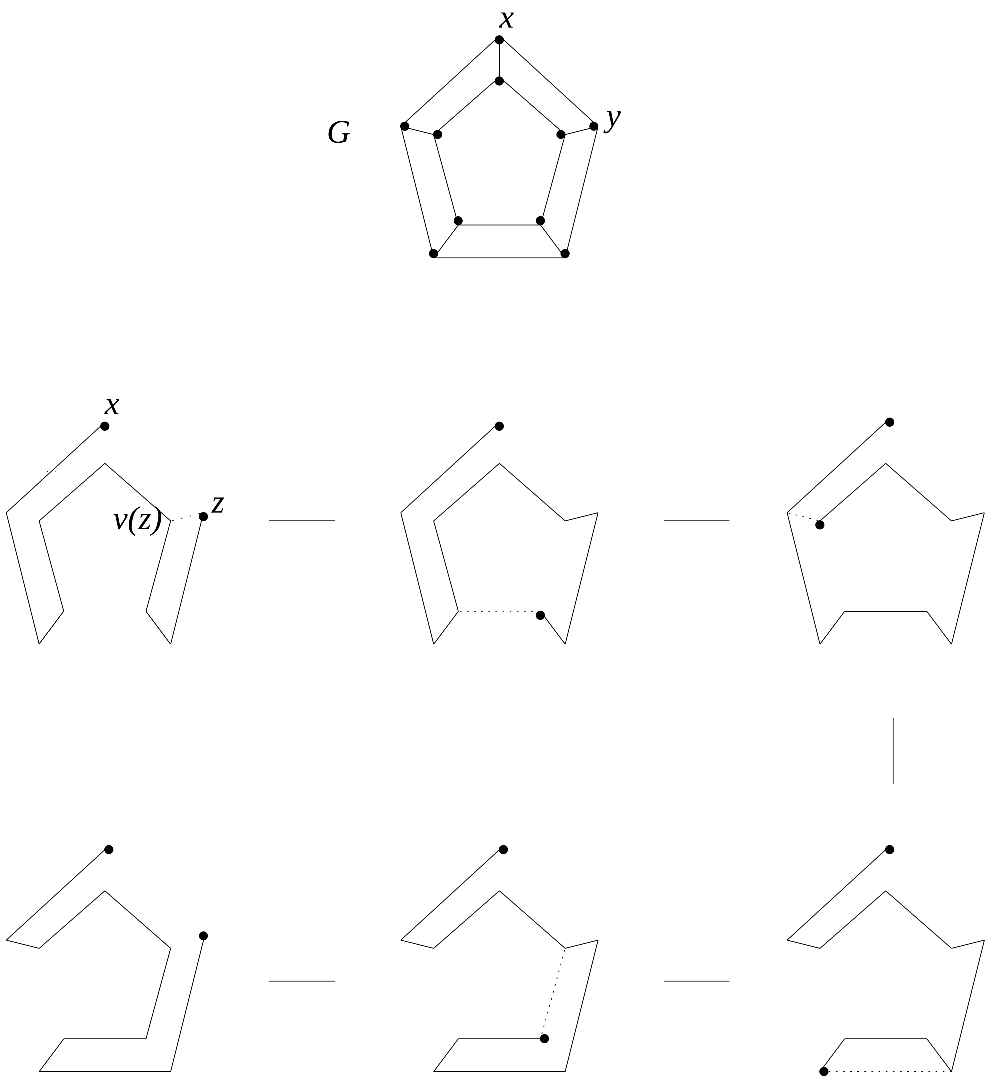


FIG. 3.3 – *Le graphe $H(G)$ pour le graphe cubique G et l'arête $\{x, y\}$.*

Hamiltonien courant on construit un autre en suivant les arêtes du chemin à partir de x jusqu'au sommet $v(z)$, après l'arête $\{v(z), z\}$ et en continuant avec les arêtes du chemin à partir de z en ordre inverse. On s'arrête quand le nouveau chemin Hamiltonien construit a comme extrémités x et y.

Krawczyk [Kraw95] a *trouvé* une famille de graphes cubiques sur laquelle l'algorithme de Thomason est exponentiel. Cameron [Cam98] a donné *la preuve* de ce résultat. Actuellement, aucun algorithme polynomial n'est connu pour résoudre le problème précédent même pour le cas des graphes cubiques.

SECOND HAMILTONIAN CYCLE
Entrée : un graphe cubique G et un cycle Hamiltonien C de G.
Sortie : un autre cycle Hamiltonien différent de C.

Thomason a montré [Tho78] que chaque graphe 4-régulier a un nombre pair de décompositions des arêtes en deux cycles Hamiltoniens. Le problème total qu'on peut considérer est :

SECOND HAMILTONIAN DECOMPOSITION
Entrée : un graphe 4-régulier et une décomposition des arêtes en deux cycles Hamiltoniens.
Sortie : une autre de ces décompositions.

Pour ce problème on ne connaît pas non plus d'algorithme polynomial. Papadimitriou a montré de plus ([Pap94b, Théorème 11, p. 517]) que ce problème est dans PPA.

Chapitre 4

Subset-Sum Equality

Dans ce chapitre nous présentons nos résultats concernant le problème Subset-Sum Equality définit par : étant donnés n entiers positifs, décider s'il existe deux sous-ensembles disjoints et non vides avec la même somme. Ce problème introduit par Woeginger et Yu [WY92] a un lien avec le problème Subset-Sum, une variante connue de Knapsack.

Knapsack a été montré NP-complet, mais pas au sens fort, en 1972 par Karp [K72]. Il reste NP-complet même si le poids de chaque objet est égal à sa valeur. Ce dernier problème est appelé Subset-Sum. L'entrée d'une instance de Subset-Sum est constituée d'un ensemble de n entiers positifs $a_1, \ldots, a_n$ ainsi que d'un autre entier positif b. Le problème consiste à décider s'il existe un ensemble d'entiers (un sous-ensemble de $\{a_1, \ldots, a_n\}$) dont la somme est égale à b. Dans la version d'optimisation, le but est de trouver un ensemble d'entiers dont la somme est la plus grande possible tout en ne dépassant pas b. Sahni ([S75]) a donné un schéma d'approximation en temps polynomial pour le problème d'optimisation associé à Knapsack qui, par conséquent, s'applique aussi à Subset-Sum. Pour le même problème, Ibarra et Kim ([IK75]) ont donné un schéma d'approximation en temps complètement polynomial. Le plus efficace des schémas d'approximation en temps complètement polynomial connus pour le problème Subset-Sum est dû à Kellerer *et al.* [KMPS97]. Le temps de calcul de leur algorithme est en $O(\min\{n/\varepsilon, n + (1/\varepsilon)^2 \log(1/\varepsilon)\})$ et l'espace requis est en $O(n + 1/\varepsilon)$ où n est le nombre d'entiers et ε l'erreur.

Woeginger et Yu ([WY92]) ont défini un problème d'optimisation lié à Subset-Sum Equality que nous appelons Subset-Sum Ratio : on demande de trouver deux sous-ensembles disjoints tels que le rapport entre les sommes est le plus proche possible de 1. Woeginger et Yu ont dans le même article prouvé la NP-complétude de Subset-Sum Equality et donné un algorithme polynomial lequel est une approximation à un facteur 1,324 près pour Subset-Sum Ratio. Savoir si ce problème a un schéma d'approximation en temps polynomial a été laissé comme question ouverte [WY92]. Comme leur preuve de NP-complétude ne montre pas que ce problème est NP-complet au sens fort (ce qui n'est pas le cas comme nous allons le voir dans une section ultérieure), ce problème peut même avoir un schéma d'approximation en temps complètement polynomial.

Dans ce chapitre nous répondons à la question de Woeginger et Yu affirmativement, en montrant une assertion plus forte : que SUBSET-SUM RATIO a un schéma d'approximation en temps complètement polynomial, donnant ainsi le meilleur résultat possible d'approximation pour ce problème.

Le problème défini par Woeginger et Yu a une instance spéciale intéressante. Considérons le cas où la somme de n nombres est plus petite que $2^n - 1$. Nous avons déjà mentionné ce problème dans la section 3.3.2 où nous avons vu qu'en utilisant le principe des tiroirs, il existe toujours deux sous-ensembles disjoints et non vides dont les sommes sont égales. Néanmoins, il n'existe pas à ce jour d'algorithme polynomial pour trouver effectivement de tels sous-ensembles. Nous appelons ce dernier problème PIGEONHOLE SUBSET-SUM.

Bien que la solubilité en temps polynomial de PIGEONHOLE SUBSET-SUM reste une question encore ouverte, nous allons montrer que, dans un certain sens, ce problème s'approxime beaucoup mieux en temps polynomial que SUBSET-SUM EQUALITY. Dans ce but, nous définissons en rapport avec ce dernier un nouveau problème d'optimisation que nous appelons SUBSET-SUM DIFFERENCE. La valeur d'une solution est ici la différence entre les sommes de deux sous-ensembles plus 1. Le même problème, avec la contrainte supplémentaire que la somme des nombres est plus petite que $2^n - 1$, est appelé PIGEONHOLE SUBSET-SUM DIFFERENCE.

L'existence d'un schéma d'approximation en temps complètement polynomial pour SUBSET-SUM RATIO implique que, pour chaque constante k, il existe un algorithme polynomial qui donne une approximation à un facteur $2^n/n^k$ près pour PIGEONHOLE SUBSET-SUM DIFFERENCE. Nous allons montrer un résultat plus fort, en donnant un algorithme polynomial qui est une approximation à un facteur $2^n/n^{\Omega(\log n)}$ près pour ce problème. Ce résultat sera suivi d'un théorème plus général : nous montrons que SUBSET-SUM DIFFERENCE est approximable à un facteur $K/n^{\Omega(\log n)}$ près en temps polynomial où K est le plus grand entier de l'instance. D'autre part, nous présentons un résultat négatif pour SUBSET-SUM DIFFERENCE, prouvant qu'il n'est pas approximable à un facteur 2^{n^k} près en temps polynomial pour chaque constante k, à moins que $P = NP$.

4.1 Problèmes

Nous donnons maintenant les définitions formelles des problèmes que nous allons étudier.

Subset-Sum Equality
Entrée : un ensemble $\{a_1, \ldots, a_n\}$ d'entiers positifs.
Question : existe-t-il deux sous-ensembles disjoints et non vides $S_1, S_2 \subseteq \{1, \ldots, n\}$ tels que

$$\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i \ ?$$

Pigeonhole Subset-Sum
Entrée : un ensemble $\{a_1, \ldots, a_n\}$ d'entiers positifs tels que

$$\sum_{i=1}^{n} a_i < 2^n - 1.$$

Sortie : deux sous-ensembles disjoints et non vides $S_1, S_2 \subseteq \{1, \ldots, n\}$ tels que

$$\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i.$$

Subset-Sum Ratio
Entrée : un ensemble $\{a_1, \ldots, a_n\}$ d'entiers positifs.
Sortie : deux sous-ensembles disjoints et non vides $S_1, S_2 \subseteq \{1, \ldots, n\}$ avec

$$\sum_{i \in S_1} a_i \geq \sum_{i \in S_2} a_i$$

tels que le rapport

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i}$$

soit minimum.

Subset-Sum Difference
Entrée : un ensemble $\{a_1, \ldots, a_n\}$ d'entiers positifs.
Sortie : deux sous-ensembles disjoints et non vides $S_1, S_2 \subseteq \{1, \ldots, n\}$, avec

$$\sum_{i \in S_1} a_i \geq \sum_{i \in S_2} a_i$$

tels que la différence

$$\sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i + 1$$

soit minimum.

Remarque. On ajoute 1 à la différence dans le problème précédent pour éviter que la valeur optimum soit 0, le rapport à l'optimum ne pouvant alors être défini.

Pigeonhole Subset-Sum Difference
Entrée : un ensemble $\{a_1, \dots, a_n\}$ d'entiers positifs tels que

$$\sum_{i=1}^{n} a_i < 2^n - 1.$$

Sortie : deux sous-ensembles disjoints et non vides $S_1, S_2 \subseteq \{1, \dots, n\}$, avec

$$\sum_{i \in S_1} a_i \geq \sum_{i \in S_2} a_i$$

tels que la différence

$$\sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i + 1$$

soit minimum.

4.2 Subset-Sum Ratio $\in$ **FPTAS**

Nous allons d'abord rappeler l'algorithme polynomial de Woeginger et Yu qui donne une approximation à un facteur 1,324 près pour le problème Subset-Sum Ratio puis nous donnerons un algorithme pseudo-polynomial pour ce problème qui sera utilisé dans la suite pour construire un schéma d'approximation en temps complètement polynomial.

4.2.1 L'algorithme de Woeginger et Yu

L'algorithme d'approximation pour Subset-Sum Ratio de Woeginger et Yu est simple. Soit $c \approx 1,324$ la solution réelle positive de l'équation $c^3 = c + 1$. Supposons que les entiers sont donnés par ordre croissant : $a_1 < \cdots < a_n$. On distingue entre trois cas :

Cas 1 :

S'il existe deux entiers consécutifs a_j, a_{j+1} tels que $a_{j+1} \leq ca_j$ alors on prend comme solution $S_1 = \{j+1\}$ et $S_2 = \{j\}$ qui est évidemment une approximation à un facteur c près de l'optimum ;

Cas 2 :

Sinon si pour tout $i : 1 \leq i \leq n-1$,

$$a_{i+1} > \sum_{k=1}^{i} a_k$$

on retourne une solution optimum en considérant le rapport minimum réalisé par les $n-1$ solutions suivantes : $S_1^i = \{i+1\}$ et $S_2^i = \{1, \dots, i\}$, pour $1 \leq i \leq n-1$;

Cas 3 :

Sinon, il existe $i : 1 \leq i \leq n-1$ tel que

$$a_{i+1} \leq \sum_{k=1}^{i} a_k.$$

Déterminons l'indice m tel que

$$\sum_{k=m+1}^{i} a_k < a_{i+1} \leq \sum_{k=m}^{i} a_k.$$

Nous montrons, en tenant compte du fait que c est la solution réelle positive de l'équation $c^3 = c + 1$, qu'au moins une des deux solutions suivantes a une valeur qui vaut au plus c: $S_1 = \{i+1\}$ et $S_2 = \{m+1, \ldots, i\}$ ou $S_1' = \{m, \ldots, i\}$ et $S_2' = \{i+1\}$.

Supposons le contraire et notons

$$s_1 = \sum_{k=m+1}^{i} a_k,$$

$$s_2 = a_{i+1}$$

et

$$s_3 = \sum_{k=m}^{i} a_k.$$

Ainsi, $s_2 > cs_1$ et $s_3 > cs_2$ et donc $s_3 > c^2 s_1$ qui est équivalent à :

$$\sum_{k=m}^{i} a_k > c^2 (\sum_{k=m+1}^{i} a_k).$$

Donc

$$a_m > (c^2 - 1)(\sum_{k=m+1}^{i} a_k) > (c^2 - 1) a_{m+1} > (c^2 - 1) c a_m.$$

On obtient $c^3 < c + 1$ qui contredit le choix de c.

En regardant la preuve de NP-complétude de SUBSET-SUM EQUALITY, la première idée a été de montrer que SUBSET-SUM EQUALITY n'est pas NP-complet au sens fort. Nous avons pour cela donné un algorithme pseudo-polynomial qui résout SUBSET-SUM RATIO.

4.2.2 Un algorithme pseudo-polynomial

Supposons que les n nombres sont en ordre croissant, $0 < a_1 < \cdots < a_n$ et posons $B = \sum_{i=1}^{n} a_i$. Nous allons donner un algorithme qui trouve une solution optimum en temps $O(nB^2)$.

L'algorithme pseudo-polynomial consiste en deux parties : la première partie qui constitue la partie principale est exécutée pour toutes les instances et la deuxième partie est exécutée que pour les instances dont l'optimum est plus grand que 1.

La partie principale de l'algorithme est une procédure de programmation dynamique. Nous allons remplir (peut-être partiellement) deux tables : $t[0..n, 0..B]$ avec des valeurs dans $\{0, 1\}$ et $c[0..n, 0..B]$ avec des entrées *sous-ensembles* de $\{1, \ldots, n\}$. Les parties non-vides des tables vont satisfaire les propriétés suivantes pour $(i, j) \neq (0, 0)$:

1. $t[i, j] = 1$ si et seulement s'il existe un ensemble $S \subseteq \{1, \ldots, n\}$ avec $\sum_{k \in S} a_k = j$ et contenant comme plus grand entier i;

2. $c[i, j] = S$ où $S \subseteq \{1, \ldots, n\}$ est le sous-ensemble unique satisfaisant les conditions précédentes, si un tel S existe et $S = \emptyset$ sinon.

Nous arrêtons la procédure si, pour un certain j, deux entiers $i_1 \neq i_2$ sont trouvés tels que $t[i_1, j] = t[i_2, j] = 1$. En fait, la procédure est arrêtée quand le premier (le plus petit) j qui convient est trouvé. Les tables sont remplies complètement dans le cas contraire.

La procédure est comme suit :

```
t[0,0] := 1, c[0,0] := ∅;
for i = 1 to n do t[i,0] := 0, c[i,0] := ∅;
for j = 1 to B do t[0,j] := 0, c[0,j] := ∅;
for j = 1 to B do
   for i = 1 to n do
      if (j ≥ a_i and ∃ k ∈ {0,...,i−1} with t[k, j − a_i] = 1) then
         t[i,j] := 1, c[i,j] := c[k, j − a_i] ∪ {i};
      else t[i,j] := 0, c[i,j] := ∅;
    if (∃ i_1 ≠ i_2 with t[i_1,j] = t[i_2,j] = 1) then STOP.
```

Si la procédure s'arrête avant de remplir les tables alors on retourne comme solution le couple des ensembles : $(c[i_1, j], c[i_2, j])$.

Sinon, donc si les tables t et c sont remplies complètement, nous continuons l'algorithme :

Nous disons d'un entier $j > 0$ qu'il est un *candidat* s'il est la somme de certains entiers de l'entrée ; c'est-à-dire si nous avons $t[i, j] = 1$ pour un certain i.

Pour chaque candidat j, soit $i(j)$ l'unique entier avec cette propriété. De plus, pour chaque candidat j, soit k_j le plus grand des candidats plus petits que j tel que $c[i(j), j] \cap c[i(k_j), k_j] = \emptyset$, si un tel k_j existe. La solution est alors le couple $(c[i(j), j], c[i(k_j), k_j])$ pour lequel j/k_j est minimisé.

Théorème 4.2.1 *L'algorithme précédent retourne une solution optimum en* $O(nB^2)$.

Preuve. D'abord nous montrons que après exécution de la procédure de programmation dynamique ci-dessous les parties non vides des tables satisfont les propriétés 1 et 2 mentionnées précédemment. Nous faisons une preuve par induction sur j.

Pour $j = 0$ et pour tout $i \in \{1, \ldots, n\}$ comme $t[i, j] = 0$ et $c[i, j] = \emptyset$, les propriétés 1 et 2 sont évidemment satisfaites car toutes les entiers sont positifs.

Pour $j = 1$ il est facile de voir que les deux propriétés sont satisfaites pour tout $i \in \{0, 1, \ldots, n\}$:

– si le plus petit entier $a_1 = 1$ alors $t[1, 1] = 1$ et $c[1, 1] = \{1\}$ et pour tout $i \geq 2$, $t[i, 1] = 0$ et $c[i, 1] = \emptyset$.

– si $a_1 > 1$ alors pour tout $i \geq 1$, $t[i, 1] = 0$ et $c[i, 1] = \emptyset$.

Supposons que les deux propriétés sont satisfaites pour tout j, $0 \leq j \leq \ell - 1$ et tout i, $0 \leq i \leq n$. Si $t[i, \ell] = 1$ pour un certain i alors la condition du premier « if » de la procédure est satisfaite : $\ell \geq a_i$ et il existe un $k < i$ tel que $t[k, \ell - a_i] = 1$. Par induction la propriété 1 est satisfaite pour $\ell - a_i$ et donc il existe un ensemble S' qui contient comme plus grand entier k et $\sum_{m \in S'} a_m = \ell - a_i$. En plus $c[i, \ell] = c[k, j - a_i] \cup \{i\}$ et par induction $S' = c[k, j - a_i]$. L'ensemble $S = S' \cup \{i\}$ satisfait les conditions de la propriété 1 car il contient comme plus grand entier i et $\sum_{m \in S} a_m = \ell$.

S'il existe un ensemble $S \subseteq \{1, \ldots, n\}$ avec $\sum_{m \in S} a_m = \ell$ et qui contient comme plus grand entier i considérons l'ensemble $S' = S \backslash \{i\}$. Notons par k le plus grand entier contenu de S'. Par induction $t[k, j - a_i] = 1$ et $c[k, j - a_i] = S'$. Ainsi, après l'exécution de la procédure nous avons $t[i, \ell] = 1$ et $c[i, \ell] = c[k, j - a_i] \cup \{i\} = S$.

Nous montrons ensuite que l'algorithme précédent retourne une solution optimum.

Si la valeur de l'optimum de l'instance du problème est 1 alors notons par j le plus petit entier tel qu'il existe deux sous-ensemble différents S_1 et S_2 avec

$$\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i = j.$$

Notons par i_1 et i_2 les plus grands entiers qui se trouvent dans S_1 et S_2 respectivement. Utilisant la propriété 1 nous avons que $t[i_1, j] = t[i_2, j] = 1$ et donc dans ce cas la procédure s'arrête et retourne comme solution le couple de sous-ensembles : (S_1, S_2). La minimalité de la somme assure que ces sous-ensembles sont, en fait, disjoints.

Si la valeur de l'optimum est plus grande que 1 alors les tables t et c seront remplies complètement. Comme pour chaque candidat j on cherche le plus grand candidat plus petit que lui, les deux correspondant à la somme de deux ensembles disjoint, l'algorithme retourne deux sous-ensembles disjoints dont le rapport entre les sommes est minimum.

Il est facile de voir que dans le pire des cas il nous faut un temps de l'ordre de $O(nB^2)$ pour trouver une solution optimum : pour décider la valeur d'une case de deux tables il nous faut un temps de $O(n)$ et donc pour remplir les deux tables il nous faut $O(n^2B)$. Si l'optimum du problème n'est pas 1, alors pour trouver pour un candidat j le plus grand candidat plus petit que lui k_j tel que $c[i(j), j] \cap c[i(k_j), k_j] = \emptyset$, on peut avoir besoin d'un temps de l'ordre de $O(nj)$ et donc pour trouver une solution optimum de $O(nB^2)$. $\square$

Nous appliquons l'algorithme pseudo-polynomial précédent sur un exemple.

Exemple : Considérons comme instance les entiers $a_1 = 2, a_2 = 5, a_3 = 6, a_4 = 10, a_5 = 14$. La valeur optimum de cette instance est 1. Nous remplissons les tables t et c jusqu'à trouver une solution optimum. La solution trouvée est $S_1 = \{3, 4\}$ et $S_2 = \{1, 5\}$, les deux ensembles ayant la somme 16.

t	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**...**	**37**
0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
1	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
2	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0			
3	0	0	0	0	0	0	**1**	0	**1**	0	0	**1**	0	**1**	0	0	0			
4	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	0	**1**	**1**			
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**			

c	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**...**	**37**
0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	...	$\emptyset$
1	$\emptyset$	$\emptyset$	{1}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$			
2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{2}	$\emptyset$	{1,2}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$			
3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{3}	$\emptyset$	{1,3}	$\emptyset$	$\emptyset$	{2,3}	$\emptyset$	{1,2,3}	$\emptyset$	$\emptyset$	$\emptyset$			
4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{4}	$\emptyset$	{1,4}	$\emptyset$	$\emptyset$	{2,4}	{3,4}			
5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{5}	$\emptyset$	{1,5}			

4.2.3 Un schéma d'approximation en temps complètement polynomial

Comme pour les algorithmes précédents, nous commençons par trier les entiers de l'instance x par ordre croissant. Après ce prétraitement nous avons donc

$$a_1 < a_2 < \cdots < a_n.$$

Pour $m = 2, \ldots, n$, notons x_m l'instance de SUBSET-SUMS RATIO qui contient les m premiers entiers $a_1, \ldots, a_m$.

Au niveau supérieur, l'algorithme exécute sa procédure principale sur les entrées x_m, pour $m = 2, \ldots, n$, et retourne comme résultat la meilleure des solutions obtenues par la procédure pour ces instances.

Pour n'importe quel ε dans l'intervalle $0 < \varepsilon < 1$, notons

$$k(m) = \frac{\varepsilon^2 \cdot a_m}{2m}.$$

Soit $n_0 \leq n$ le plus grand entier tel que $k(n_0) \leq 1$.

Nous allons décrire la procédure sur l'instance x_m.

Si $m \leq n_0$, alors nous appliquons l'algorithme pseudo-polynomial de la section précédente à x_m. Comme $a_{n_0} \leq 2n/\varepsilon^2$, l'algorithme va prendre un temps polynomial en n et $1/\varepsilon$.

Si $n_0 < m \leq n$, alors nous transformons l'instance x_m en une autre qui contient seulement des entiers de taille polynomiale. Soient $a'_i = \lfloor a_i/k(m) \rfloor$ pour $i = 1, \ldots, m$. Nous pouvons observer que le plus grand entier de cette nouvelle instance $a'_m = \lfloor 2m/\epsilon^2 \rfloor$ est effectivement de taille polynomiale. De ces m entiers nous n'allons garder que ceux qui sont « grands ». Plus précisément, notons x'_m l'instance de SUBSET-SUMS RATIO qui ne contient que les entiers a'_i tels que $a'_i \geq m/\varepsilon$. Supposons que x'_m contient t entiers : $a'_{m-t+1}, \ldots, a'_m$. En fait l'instance x'_m contient au moins un entier car $\varepsilon \leq 1$ et donc $a'_m \geq m/\varepsilon$. Nous allons distinguer deux cas en fonction du nombre d'entiers que x'_m contient (la valeur de t).

Cas 1 : $t = 1$.

Soit j le plus petit entier positif ou nul tel que

$$a_{j+1} + \ldots + a_{m-1} < a_m.$$

Si $j = 0$, alors nous considérons la solution $S_1 = \{m\}$ et $S_2 = \{1, \ldots, m-1\}$. Sinon, la solution est $S_1 = \{j, j+1, \ldots, m-1\}$ et $S_2 = \{m\}$.

Cas 2 : $t > 1$.

Nous allons résoudre (exactement) x'_m, en utilisant l'algorithme pseudo-polynomial qui va prendre un temps polynomial sur cette instance. Nous distinguons deux cas en fonction de la valeur de l'optimum de x'_m.

Cas 2a : $opt(x'_m) = 1$.
La procédure retourne la solution qui réalise cet optimum pour x'_m.

Cas 2b : $opt(x'_m) > 1$.
Dans ce cas nous engendrons une collection de paires de sous-ensembles comme suit : considérons 3^{t-1} paires $(P(\bar{v}, m), Q(\bar{v}, m))$,

$$P(\bar{v}, m), Q(\bar{v}, m) \subseteq \{m - t + 1, \ldots, m\},$$

d'ensembles disjoints paramétrés par les vecteurs

$$\bar{v} = (v_1, \ldots, v_{t-1}) \in \{0, 1, 2\}^{t-1}.$$

Les ensembles sont définis en fonction de la règle : pour $1 \leq i \leq t - 1$,
- si $v_i = 1$ alors $m - t + i \in P(\bar{v}, m)$ et $m - t + i \notin Q(\bar{v}, m)$,
- si $v_i = 2$ alors $m - t + i \notin P(\bar{v}, m)$ et $m - t + i \in Q(\bar{v}, m)$,
- si $v_i = 0$ alors $m - t + i \notin P(\bar{v}, m)$ et $m - t + i \notin Q(\bar{v}, m)$.
Nous introduisons de plus m dans $P(\bar{v}, m)$.

Définissons $R_1(\bar{v}, m) = P(\bar{v}, m)$ si

$$\sum_{i \in P(\bar{v}, m)} a_i > \sum_{i \in Q(\bar{v}, m)} a_i$$

et $R_1(\bar{v}, m) = Q(\bar{v}, m)$ sinon. Soit $R_2(\bar{v}, m)$ l'autre ensemble.

La paire $(S_1(\bar{v}, m), S_2(\bar{v}, m))$ est définie comme suit :
Soit j le plus petit entier positif ou nul tel que

$$\sum_{i \in R_2(\bar{v}, m)} a_i + \sum_{i=j+1}^{m-t} a_i < \sum_{i \in R_1(\bar{v}, m)} a_i.$$

Si $j = 0$, alors $S_1(\bar{v}, m) = R_1(\bar{v}, m)$ et $S_2(\bar{v}, m) = R_2(\bar{v}, m) \cup \{1, \ldots, m - t\}$. Sinon :

- si $m \in R_1(\bar{v}, m)$, alors $S_1(\bar{v}, m) = R_2(\bar{v}, m) \cup \{j, \ldots, m - t\}$ et $S_2(\bar{v}, m) = R_1(\bar{v}, m)$;
- si $m \in R_2(\bar{v}, m)$, alors $S_1(\bar{v}, m) = R_1(\bar{v}, m)$ et $S_2(\bar{v}, m) = R_2(\bar{v}, m) \cup \{j + 1, \ldots, m - t\}$.

Finalement, nous choisissons un vecteur $\bar{v} \in \{0,1,2\}^{t-1}$ pour lequel le rapport

$$\frac{\sum_{i \in S_1(\bar{v},m)} a_i}{\sum_{i \in S_2(\bar{v},m)} a_i}$$

est minimum. La solution donnée par la procédure est alors $S_1 = S_1(\bar{v},m)$ et $S_2 = S_2(\bar{v},m)$.

Théorème 4.2.2 *L'algorithme précédent donne une approximation à un facteur $(1+\varepsilon)$ près en temps polynomial en n et $1/\varepsilon$.*

Preuve. L'algorithme travaille en temps polynomial chaque fois que le nombre 3^{t-1} de vecteurs dans le cas 2b est polynomial en n et $1/\varepsilon$. Comme $opt(x'_m) > 1$ dans ce cas, les 2^t sous-ensembles de l'ensemble $\{a'_{m-t+1}, \ldots, a'_m\}$ ont tous des sommes distinctes. Comme

$$a'_m \leq \frac{2m}{\varepsilon^2},$$

nous avons

$$\sum_{i=m-t+1}^{m} a'_i < \frac{2m^2}{\varepsilon^2}.$$

Par conséquent

$$2^t \leq \frac{2m^2}{\varepsilon^2}$$

et donc $t \leq 2\log(m/\varepsilon) + 1$.

Nous allons maintenant prouver que l'algorithme donne effectivement une approximation à un facteur $(1+\varepsilon)$ près. D'abord nous montrons que pour chaque instance x_m on trouve soit une solution optimum, soit une solution optimum parmi les solutions de x_m qui contiennent a_m, soit deux ensembles S_1 et S_2 tels que

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i} \leq 1 + \varepsilon.$$

Si $m \leq n_0$, alors l'algorithme pseudo-polynomial donne une solution optimum. Supposons donc $m > n_0$.

Dans le cas 1 :

- Si $j = 0$, alors la solution donnée est une solution optimum de x_m parmi les solutions de x_m qui contiennent a_m;

- Si $j > 0$, alors

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i} \leq 1 + \frac{a_j}{a_m} < 1 + \varepsilon.$$

Dans le cas 2a, nous avons

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i} \leq \frac{\sum_{i \in S_1} k(m) \cdot (1 + a'_i)}{\sum_{i \in S_2} k(m) \cdot a'_i} = 1 + \frac{|S_1|}{\sum_{i \in S_2} a'_i} \leq 1 + \frac{t}{m/\varepsilon} < 1 + \varepsilon\,.$$

Dans le cas 2b, soit $S_1 = S_1(\bar{v}, m)$ et $S_2 = S_2(\bar{v}, m)$ pour un certain $\bar{v} \in \{0,1,2\}^{t-1}$.

- Si $j = 0$, alors le couple S_1, S_2 est une solution optimum parmi les solutions de x_m qui contiennent a_m ;

- Sinon, nous avons

$$\sum_{i \in R_2(\bar{v},m)} a_i + \sum_{i=j+1}^{m-t} a_i < \sum_{i \in R_1(\bar{v},m)} a_i \leq \sum_{i \in R_2(\bar{v},m)} a_i + \sum_{i=j}^{m-t} a_i.$$

  Par conséquent,

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i} \leq 1 + \frac{a_j}{\sum_{i \in S_2} a_i} \leq 1 + \frac{a_j}{a_m} < 1 + \varepsilon.$$

Soit m un entier tel que a_m soit le plus grand entier qui apparaît dans une solution optimum de l'instance x. Si $m \leq n_0$, la solution retournée est une solution optimum pour x. Si $m > n_0$ et si on retourne une solution optimum parmi les solutions de x_m qui contiennent a_m alors elle sera une solution optimum pour x. Si $m > n_0$ et on retourne deux ensembles S_1 et S_2 tels que

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i} \leq 1 + \varepsilon,$$

alors cette solution sera évidemment une approximation à un facteur $1 + \varepsilon$ près pour l'instance x. $\square$

Remarque. Notre algorithme retourne comme solution soit deux ensembles S_1, S_2 tels que :

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i} \leq 1 + \varepsilon$$

soit une solution optimum.

4.3 SUBSET-SUM DIFFERENCE

Comme SUBSET-SUM RATIO a un schéma d'approximation en temps complètement polynomial, du point de vue de l'approximation, nous ne pouvons pas distinguer SUBSET-SUM EQUALITY de PIGEONHOLE SUBSET-SUM quand la valeur de la solution est le rapport entre les sommes des deux sous-ensembles. La situation change complètement quand on considère un problème plus difficile, où la valeur de la solution est la différence entre les deux sommes. Dans cette section nous montrons que PIGEONHOLE SUBSET-SUM DIFFERENCE est approximable à un facteur $2^n/n^{\Omega(\log n)}$ près en temps polynomial et que pour chaque constante k, SUBSET-SUM DIFFERENCE n'est pas approximable à un facteur 2^{n^k} près en temps polynomial, à moins que $P = NP$.

Le schéma d'approximation en temps complètement polynomial pour SUBSET-SUM RATIO donne un algorithme polynomial qui est une approximation à un facteur $2^n/n^k$ près pour PIGEONHOLE SUBSET-SUM DIFFERENCE quand on prend $\varepsilon = 1/n^k$. Mais, en fait, nous pouvons faire mieux.

Théorème 4.3.1 SUBSET-SUM DIFFERENCE *est approximable à un facteur* $K/n^{\Omega(\log n)}$ *près en temps polynomial où* K *est le plus grand entier de l'instance.*

Preuve. Nous allons décrire un algorithme polynomial qui trouve une solution de valeur au plus $K/n^{\Omega(\log n)}$. Comme la valeur optimum de chaque instance est au moins 1 par définition, le théorème sera prouvé.

L'idée est de construire à partir d'une instance de SUBSET-SUM DIFFERENCE avec n entiers dont le plus grand est K, une instance du même problème ayant au moins $n/3$ entiers dont le plus grand est au plus $3K/n$. Chaque entier de la nouvelle instance représente une solution pour la première instance. On itère ensuite cette construction $\lfloor \log_3 n \rfloor$ fois. Le plus petit entier de la dernière instance représente une solution de l'instance de départ et nous allons montrer qu'il est plus petit que $K/n^{\Omega(\log n)}$.

Soit $a_1 < a_2 < \cdots < a_n$ une instance de SUBSET-SUMS DIFFERENCE et $a_0 = 0$. Considérons la séquence d'entiers :

$$0 = a_0 < a_1 < a_2 < \cdots < a_n = K$$

et les différences entre deux entiers consécutifs :

$$a_1 - a_0,\ a_2 - a_1,\ a_3 - a_2, \ldots,\ a_n - a_{n-1}.$$

Il est facile de voir qu'au plus $n/3$ différences peuvent être aussi grandes que $3K/n$. Donc, au moins $2n/3$ différences sont plus petites que $3K/n$. Parmi ces dernières nous en choisissons une sur deux (dans l'ordre d'occurrence). Notons

$n^{(1)}$ le nombre des différences choisies. Avec ces entiers nous créons une autre instance du problème : on trie ces entiers par ordre croissant, obtenant ainsi la séquence

$$a_1^{(1)} < a_2^{(1)} < \cdots < a_{n^{(1)}}^{(1)}$$

à laquelle nous ajoutons $a_0^{(1)} = 0$. Notons $K^{(1)} = a_{n^{(1)}}^{(1)}$. Donc, $K^{(1)} < 3K/n$ et $n^{(1)} \geq n/3$.

Nous répétons ce type de « sélection de différences » $t = \lfloor \log_3 n \rfloor$ fois, en créant les séquences

$$0 = a_0^{(i)} < a_1^{(i)} < a_2^{(i)} < \cdots < a_{n^{(i)}}^{(i)} = K^{(i)}$$

pour $i = 2, \ldots, t$, avec $K^{(i)} < 3K^{(i-1)}/n^{(i-1)}$ et $n^{(i)} \geq n^{(i-1)}/3$.

Dans la dernière séquence il reste encore $n^{(t)} \geq n/3^t \geq 1$ entiers. Sélectionons le plus petit entier $a_1^{(t)}$.

Observons que chaque entier de chaque séquence représente un sous-ensemble de $\{a_1, \ldots, a_n\}$, certains des éléments de ce sous-ensemble apparaissent avec « $+$ » et les autres avec « $-$ ». Les éléments avec le même signe spécifient un sous-ensemble. Donc, nous avons un ensemble « $+$ » qui contient les éléments avec le signe « $+$ » et un ensemble « $-$ » qui contient les éléments avec le signe « $-$ ». La différence entre les sommes d'entiers de l'ensemble « $+$ » et de l'ensemble « $-$ » est au plus la valeur modifiée de K.

Nous considérons comme solution les deux sous-ensembles représentés par $a_1^{(t)}$. Nous allons montrer que $K^{(t)} \leq K/n^{\Omega(\log n)}$. Nous avons

$$K^{(1)} < \frac{3K}{n}$$

et

$$K^{(i)} < \frac{3K^{(i-1)}}{n^{(i-1)}}$$

pour $i = 2, \ldots, t$. En prenant le produit de ces inégalités, nous obtenons

$$K^{(t)} < \frac{3^{t(t+1)/2} \cdot K}{n^t} = \frac{K}{n^{\Omega(\log n)}}.$$

Comme la valeur de la solution considérée est au plus $K^{(t)}$, le théorème est prouvé. □

Exemple : Considérons l'instance x suivante : $a_1 = 23, a_2 = 47, a_3 = 192, a_4 = 211, a_5 = 250, a_6 = 280, a_7 = 295, a_8 = 320, a_9 = 390, a_{10} = 475$. On applique l'algorithme du théorème précédent sur l'instance x.

D'abord on ajoute $a_0 = 0$ à l'instance :

x :

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
0	23	47	192	211	250	280	295	320	390	475

On calcule les différences entre les entiers consécutifs :

$a_1 - a_0$	$a_2 - a_1$	$a_3 - a_2$	$a_4 - a_3$	$a_5 - a_4$	$a_6 - a_5$	$a_7 - a_6$	$a_8 - a_7$	$a_9 - a_8$	$a_{10} - a_9$
23	24	145	19	39	30	15	25	70	85

De ces différences on enlève celles qui sont plus grandes que $3K/n = 3 \times 475/10 = 142,5$. On obtient :

$a_1 - a_0$	$a_2 - a_1$	$a_4 - a_3$	$a_5 - a_4$	$a_6 - a_5$	$a_7 - a_6$	$a_8 - a_7$	$a_9 - a_8$	$a_{10} - a_9$
23	24	19	39	30	15	25	70	85

Parmi les différences qui restent on en considère une sur deux :

$a_1 - a_0$	$a_4 - a_3$	$a_6 - a_5$	$a_8 - a_7$	$a_{10} - a_9$
23	19	30	25	85

On obtient une autre instance du problème en triant les entiers par ordre croissant :

$x^{(1)}$:

$a_4 - a_3$	$a_1 - a_0$	$a_8 - a_7$	$a_6 - a_5$	$a_{10} - a_9$
19	23	25	30	85

Après ajout de $a_0^{(1)} = 0$ à la nouvelle instance on calcule les différences entre les entiers consécutifs :

$a_4 - a_3 - a_0^{(1)}$	$a_1 - a_0 - a_4 + a_3$	$a_8 - a_7 - a_1 + a_0$	$a_6 - a_5 - a_8 + a_7$	$a_{10} - a_9 - a_6 + a_5$
19	4	2	5	55

On enlève de ces différences celles plus grandes que $3 \times 85/5 = 51$, on considère une sur deux et après on trie les entiers par ordre croissant :

$x^{(2)}$:

$a_8 - a_7 - a_1 + a_0$	$a_4 - a_3 - a_0^{(1)}$
2	19

On sélectionne le plus petit entier de cette nouvelle instance. Cet entier est 2 et il correspond à la différence $a_8 - a_7 - a_1 + a_0$. Cela veut dire que la solution qu'on obtient est $S_1 = \{8\}$ et $S_2 = \{1, 7\}$.

Corollaire 4.3.2 PIGEONHOLE SUBSET-SUM DIFFERENCE *est approximable à un facteur $2^n/n^{\Omega(\log n)}$ près en temps polynomial.*

Nous montrons enfin un résultat simple de non-approximabilité pour Subset-Sum Difference qui est en opposition avec l'approximabilité de Pigeonhole Subset-Sum Difference.

Théorème 4.3.3 *Si $P \neq NP$, alors pour chaque constante k,* Subset-Sum Difference *n'est pas approximable à un facteur 2^{n^k} près en temps polynomial.*

Preuve. Nous allons montrer que si Subset-Sum Difference était approximable à un facteur 2^{n^k} près en temps polynomial, alors nous pourrions résoudre Subset-Sum Equality en temps polynomial.

Étant donnée une instance $x = \{a_1, a_2, \ldots, a_n\}$ de Subset-Sum Equality, nous créons (en temps polynomial) une instance $x' = \{b_1, b_2, \ldots, b_n\}$ de Subset-Sum Difference où $b_i = 2^{n^k} \times a_i$. La taille de x' est polynomiale par rapport à la taille de x et, clairement, x est une instance positive si et seulement si la valeur d'une solution optimum pour x' est 1. Soit q cette valeur optimum et soit s la valeur de la solution de x' donnée par l'algorithme qui est une approximation à un facteur 2^{n^k} près.

Nous affirmons que $q = 1$ si et seulement si $s = 1$. La condition suffisante est triviale car $q \leq s$. Pour montrer la condition nécessaire, supposons que $s > 1$. Nous avons alors

$$s \leq 2^{n^k} \times q$$

car la solution a été donnée par un algorithme qui est une approximation à un facteur 2^{n^k} près. Comme chaque entier dans x' est un multiple de 2^{n^k}, la valeur d'une solution pour x' est soit 1, soit plus grande que 2^{n^k}. Par conséquent, nous avons aussi que

$$s > 2^{n^k}$$

et donc $q > 1$.

Ainsi, x est une instance positive de Subset-Sum Equality si et seulement si la valeur de la solution retournée par l'algorithme qui est une approximation à un facteur 2^{n^k} près pour Subset-Sum Difference est 1. $\square$

Remarques. Soit d la taille d'une instance x de Subset-Sum Difference. Si $d = \Omega(2^n)$ alors x est soluble en temps polynomial.

Nous pouvons montrer un résultat plus fort que le théorème 4.3.3 : si $P \neq NP$, pour chaque constante k, Subset-Sum Difference n'est pas approximable à un facteur 2^{d^k} près en temps polynomial.

Chapitre 5

Another Longest Path (Cycle)

Longest Path et Longest Cycle sont deux problèmes bien connus dans la théorie des graphes. Nous considérons dans ce chapitre l'approximation de deux problèmes totaux liés à ces derniers qui font partie de la classe PPA et donc qui ne sont pas NP-difficiles à moins que $NP = co\text{-}NP$ et pour lesquels aucun algorithme polynomial n'est connu.

Lovász a montré [Lov79] que la somme du nombre des chemins Hamiltoniens d'un graphe G et du nombre des chemins Hamiltoniens dans son complément $\bar{G}$ est un nombre pair. Nous appelons Another Hamiltonian Path le problème qui consiste à trouver un autre chemin Hamiltonien dans un graphe G ou dans son complément $\bar{G}$ quand on connaît déjà un chemin Hamiltonien de G. On a vu dans la section 3.3.3 que ce problème est un problème total qui fait partie de la classe PPA. Nous montrons qu'on peut obtenir un chemin de longueur $n-2$ en temps polynomial dans un graphe G avec n sommets ou dans son complément $\bar{G}$ quand un chemin Hamiltonien de G est déjà connu.

Papadimitriou [Pap94a] a montré que le problème suivant est NP-difficile : étant donnés un graphe G et un cycle Hamiltonien C dans G, décider si G a un autre cycle Hamiltonien. On a vu dans la section 3.3.3 que si le graphe G est cubique et si il a un cycle Hamiltonien alors il y a au moins deux cycles Hamiltoniens. Nous avons appelé Second Hamiltonian Cycle le problème de trouver un autre cycle Hamiltonien dans un graphe cubique quand un cycle Hamiltonien est donné. Ce problème est une instance de la classe PPA [Pap94b, p. 501]. Nous donnons un schéma d'approximation en temps polynomial efficace pour le problème d'optimisation associé.

5.1 ANOTHER LONGEST PATH

Nous commençons par donner la définition du problème utilisé.

ANOTHER LONGEST PATH
Entrée : un graphe G et un chemin Hamiltonien P de G.
Sortie : un chemin de G ou de $\bar{G}$ différent de P de longueur maximum.

Dans la suite nous donnons un algorithme polynomial simple qui est une approximation à un terme additif 1 près pour ANOTHER LONGEST PATH. Ce résultat a comme consequence l'existence d'un schéma d'approximation en temps polynomial efficace pour ce problème.

Théorème 5.1.1 *Soit $G = (V, E)$ un graphe d'ordre n et le chemin Hamiltonien $P = v_1 \cdots v_n$. On peut trouver en temps polynomial un chemin de longueur $n - 2$ dans G ou dans $\bar{G}$ différent de P.*

Preuve. Si $v_1v_n \in E(G)$ alors $v_1v_nv_{n-1} \cdots v_3v_2$ est un chemin Hamiltonien de G différent de P.

Si $v_1v_n \notin E(G)$ nous distinguons deux cas :

- S'il existe une corde de longueur 2, v_iv_{i+2}, alors le chemin $P' = v_1 \cdots v_iv_{i+2} \cdots v_n$ est un chemin de G de longueur $n - 2$.
- S'il n'existe pas de corde de longueur 2 alors nous considérons deux cas en fonction du nombre de sommets :

  Si $n = 2k + 1$ alors le chemin suivant est un chemin Hamiltonien de $\bar{G}$: $v_1v_3v_5 \cdots v_{2k+1}v_2v_4 \cdots v_{2k}$.

  Si $n = 2k$ alors on prend le chemin : $v_2v_4 \cdots v_{2k}v_1v_3v_5 \cdots v_{2k-1}$ qui est un chemin Hamiltonien de $\bar{G}$.

□

Corollaire 5.1.2 ANOTHER LONGEST PATH *a un schéma d'approximation en temps polynomial efficace.*

5.2 ANOTHER LONGEST CYCLE dans les graphes cubiques et Hamiltoniens

Nous définissons d'abord la variante d'optimisation de SECOND HAMILTONIAN CYCLE.

ANOTHER LONGEST CYCLE
Entrée : un graphe cubique et Hamiltonien G et un cycle Hamiltonien C.
Sortie : un cycle différent de C de longueur maximum.

Nous montrons que ANOTHER LONGEST CYCLE admet un schéma d'approximation en temps polynomial efficace. Puis nous proposons un schéma d'approximation en temps linéaire pour ANOTHER LONGEST CYCLE.

Nous allons utiliser les notations et la terminologie suivantes.

Définitions Pour un ensemble d'arêtes $F \subseteq E(G)$, le graphe $G(F) = (W, F)$ est le graphe avec l'ensemble d'arêtes F et l'ensemble de sommets W qui coïncident avec les sommets de G qui sont incidents à au moins une arête de F.

Nous supposons dans la suite que les sommets $v_1, v_2, \ldots, v_n$ se suivent l'un après l'autre dans cet ordre sur le cycle Hamiltonien donné C de G.

La *longueur* d'une corde $e = v_i v_j \in E(G) \setminus E(C)$ $(i < j)$ est définie comme $||e|| := \min\{j-i, n+i-j\}$. Nous dénotons par P_e le plus court chemin de C avec les extrémités v_i et v_j si $||e|| < n/2$, et dénotons $P_e := v_i v_{i+1} \cdots v_j$ si $||e|| = n/2$.

On dit que deux cordes e, e'

–*se croisent* si $P_e \cap P_{e'} \neq \emptyset$, $P_e \not\subset P_{e'}$ et $P_{e'} \not\subset P_e$;

–*sont incomparables* si $P_e \cap P_{e'} = \emptyset$;

–*sont parallèles* si elles ne se croisent pas, c'est-à-dire, soit elles sont incomparables, soit $P_e \subset P_{e'}$ ou $P_{e'} \subset P_e$.

Si $P_e \subset P_{e'}$, nous disons que e est *plus petite* que e'. La corde e est *minimale* s'il n'existe pas de corde plus petite que e.

5.2.1 Un schéma d'approximation en temps polynomial efficace

L'existence d'un schéma d'approximation en temps polynomial efficace pour ANOTHER LONGEST CYCLE est une conséquence du théorème suivant.

Théorème 5.2.1 *Soit $G = (V, E)$ un graphe cubique d'ordre n avec le cycle Hamiltonien $C = v_1 v_2 \cdots v_n$. Un cycle $C' \neq C$ de longueur au moins $n - 4\sqrt{n}$ peut être trouvé en temps $O(n^{3/2} \log n)$.*

Corollaire 5.2.2 ANOTHER LONGEST CYCLE *a un schéma d'approximation en temps polynomial efficace.*

Preuve du Théorème 5.2.1. Soit $k := \lceil \sqrt{n} \rceil$.

Premièrement, on vérifie en $n/2$ étapes si C a une corde de longueur au plus k. Si une telle corde e existe, alors $(E(C) \cup \{e\}) \setminus E(P_e)$ est un cycle de longueur cherchée .

Supposons dans la suite que toutes les cordes sont plus longues que k. Nous considérons alors k cordes consécutives, qui commencent en $v_1, \ldots, v_k$. Dénotons par w_i l'autre extrémité de la corde e_i incidente avec v_i et le chemin $P' := v_{k+1} v_{k+2} \cdots v_n$. L'ordre de ces k sommets $w_1, \ldots, w_k$ sur P' peut être déterminé en au plus $O(k \log k) = O(n^{1/2} \log n)$ étapes en utilisant l'algorithme de tri par tas. Puis la paire la plus proche peut être sélectionnée en k étapes. Dénotons par i_1 et i_2 les indices de cette paire. Ainsi, w_{i_1} et w_{i_2} sont écartés sur le chemin P' d'une distance plus petite que

$$\frac{n-k}{k-1} < k.$$

Si e_{i_1} et e_{i_2} se croisent (supposons que $i_1 < i_2$), alors

$$v_{i_2} v_{i_2+1} \cdots w_{i_1-1} w_{i_1} v_{i_1} v_{i_1-1} \cdots w_{i_2+1} w_{i_2}$$

est un cycle de longueur au moins $n - 2k + 2 > n - 2\sqrt{n}$.

Sinon, si e_{i_1} et e_{i_2} sont parallèles, on les retient comme une configuration de départ.

Pour simplifier la notation, dénotons $e := e_{i_1}$, $e' := e_{i_2}$, et supposons que $e = v_\ell v_j$, $e' = v_{\ell'} v_{j'}$. Il est possible que e et e' soient incomparables (c'est-à-dire, aucun d'entre eux n'est plus petit que l'autre), mais nous pouvons supposer, sans perte de généralité (en renumérotant les sommets si nécessaire) que $P_e = v_\ell v_{\ell+1} \cdots v_{j-1} v_j$ et que $P_{e'} \not\subset P_e$. Nous considérons alors les k cordes $e'_1, \ldots, e'_k$, avec une extrémité dans les sommets $v_{\ell+1}, \ldots, v_{\ell+k}$; et nous sélectionnons parmi ces cordes deux cordes f et f' avec la propriété que les autres extrémités de ces cordes sont à une distance plus petite que k.

- Si f et f' se croisent, alors un cycle de longueur au moins $n - 2\sqrt{n}$ est facile à trouver comme précédemment.

  Par conséquent nous pouvons supposer que f et f' ne se croisent pas.

- Si les deux cordes f et f' sont plus petites que e, alors nous allons renommer $e := f$ et $e' := f'$ et nous recommençons l'étape précédente. Notons que cette situation ne peut apparaître plus de $O(n)$ fois, puis que $||e||$ décroît à chaque étape.

- Supposons que f' ou f croisent e (peut être les deux) mais qu'ils ne croisent pas e'. Dans cette situation, e et la corde qui croise e forment un cycle de longueur au moins $n - 2\sqrt{n}$.

– De manière similaire, si f est plus petite que e mais f' croise e et e', alors f' avec e ou e' est un bon choix pour construire un cycle de longueur cherchée.

– Finalement, supposons que f et f' croisent e et e'. On enlève les deux paires d'arcs courts (de longueurs $< k$) joignant les cordes parallèles (c'est-à-dire, on enlève les sous-chemins de C qui joignent P_e à $P_{e'}$, et aussi ceux entre P_f et $P_{f'}$) pour créer quatre chemins de longueur totale au moins $n - 4k$. Nous obtenons après un cycle plus long que $n - 4\sqrt{n}$ en ajoutant les quatre arêtes e, e', f, f'.

□

5.2.2 Un schéma d'approximation en temps linéaire

Pour designer une approximation en temps linéaire, nous avons besoin du lemme suivant qui est une conséquence de théorème de Dilworth [Dil50].

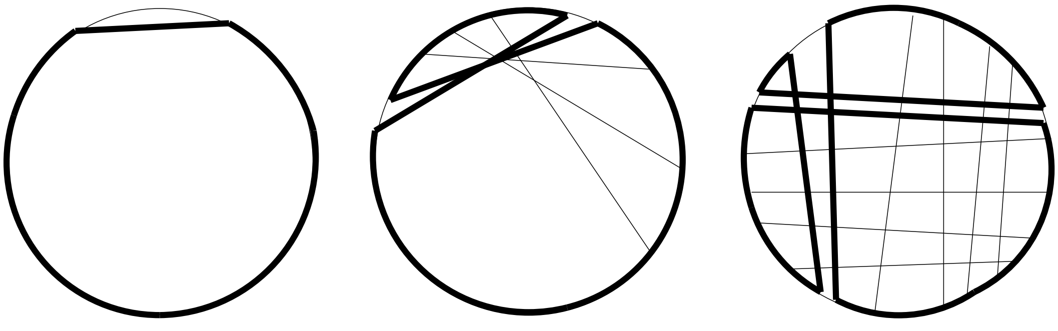
Lemme 5.2.3 *Soient e une corde de cycle Hamiltonien C et Q un ensemble de $3k^2 - 3k + 1$ cordes qui croisent e. Alors Q contient soit k cordes qui se croisent mutuellement soit $3k + 1$ cordes parallèles.*

Preuve. Soit $e = v_i v_j$, $i < j$. Pour n'importe quel couple de cordes $e', e'' \in Q$, nous écrivons $e' <_e e''$ si et seulement si sur les deux chemins P_e et $E(C) \setminus E(P_e)$, v_i est plus proche des extrémités de e' que de celles de e''. Clairement, '$<_e$' définit un ordre partiel et deux cordes de Q sont incomparables sous $<_e$ quand précisément elles se croisent.

En enlevant tous les éléments minimaux de l'ensemble partiel ordonné $(Q, <_e)$, la longueur de chaque chaîne maximale décroît de un exactement. Par conséquent, il est facile de voir par induction que soit $(Q, <_e)$ contient une chaîne Q' de longueur $3k + 1$, soit il peut être décomposé en au plus $3k$ antichaînes. Dans le premier cas, Q' est un ensemble suffisamment grand de cordes mutuellement parallèles. Dans le dernier cas, une antichaîne Q'' de la décomposition a la cardinalité $|Q''| \geq \lceil |Q|/(3k) \rceil = k$. Cette antichaîne Q'' fournit k cordes qui se croisent mutuellement. □

Notons que, pour n'importe quel nombre naturel fixé k, il est demandé seulement un nombre constant d'étapes (constante qui ne dépend que de k) pour trouver le sous-ensemble Q' ou Q'' de cordes de n'importe quel Q avec les hypothèses précédentes.

Théorème 5.2.4 *Pour chaque $\varepsilon > 0$, un cycle $C' \neq C$ de longueur au moins $(1 - \varepsilon)n$ peut être trouvé en temps $2^{O(1/\varepsilon^2)} \times n$.*



(1) (2) (3)

FIG. 5.1 – *Trois manières de trouver un cycle long*

Preuve. Étant donné $\varepsilon > 0$, définissons $k = \lceil 1/\varepsilon \rceil$. Nous supposons $n > 3k^3$, sans perte de généralité. (Pour $16k^2 \leq n < 3k^3$ nous pouvons trouver une approximation à un facteur $(1+\varepsilon)$ près en utilisant le théorème 5.2.1 et pour $n < 16k^2$ un deuxième cycle *Hamiltonien* $C' \neq C$ peut être trouvé en temps $2^{O(1/\varepsilon^2)}$ en faisant une recherche exhaustive ou comme dans le théorème de Smith [Pap94b, Tho78].)

Notre but est de prouver qu'au moins une des trois situations suivantes apparaît dans n'importe quel graphe cubique $G = (V, E)$ d'ordre n, qui a pour cycle Hamiltonien $v_1 v_2 \cdots v_n$.

1. Il existe une corde de longueur au plus $3k^2$ dans G.
2. Il existe k cordes qui se croisent mutuellement dans G.
3. Il existe deux ensembles S_1 et S_2 de cordes tels que
   (a) $|S_1| = |S_2| = k + 1$;
   (b) S_i est constitué de cordes mutuellement parallèles, pour $i = 1, 2$;
   (c) chaque $e_1 \in S_1$ croise chaque $e_2 \in S_2$.

Nous allons d'abord montrer, comment n'importe laquelle de ces trois possibilités implique la validité du théorème (voir la figure 5.1 pour illustration.)

Cas 1. Si e est une corde telle que $||e|| \leq 3k^2$, alors $(E(C) \setminus E(P_e)) \cup \{e\}$ est un cycle plus long que $n - 3k^2 > (1 - \varepsilon)n$.

Cas 2. Si G contient k cordes qui se croisent, alors il existe une séquence $1 \leq i_1 < i_2 < \cdots < i_{2k}$ telle que $e_\ell := v_{i_\ell} v_{i_{\ell+k}} \in E(G)$ est une corde pour tout $1 \leq \ell \leq k$. Définissons $d_\ell := (i_{\ell+1} - i_\ell) + (i_{\ell+k+1} - i_{\ell+k})$ pour $1 \leq \ell < k$ et $d_k := (i_{k+1} - i_k) + (n + i_1 - i_{2k})$. Nous avons $d_1 + d_2 + \ldots + d_k = n$, donc nous pouvons supposer (après renumérotation des sommets si nécessaire) que $d_1 \leq n/k \leq \varepsilon n$. En enlevant les sommets v_j de C, pour tout $i_1 < j < i_2$ et tout $i_{k+1} < j < i_{k+2}$, les deux sous-chemins qui restent avec e_1 et e_2 forment un cycle suffisamment long.

Cas 3. Avec un choix adéquat du sommet de départ du cycle Hamiltonien, les deux sous-ensembles S_1, S_2 donnent maintenant une séquence:

$$1 \leq i_1 < i_2 < \cdots < i_{k+1} < j_1 < j_2 < \cdots < j_{k+1} < i_{k+2} < i_{k+3} < \cdots$$
$$\cdots < i_{2k+2} < j_{k+2} < j_{k+3} < \cdots < j_{2k+2} \leq n\,,$$

correspondant aux cordes $e_\ell := v_{i_\ell} v_{i_{2k+3-\ell}} \in S_1$ et $f_\ell := v_{j_\ell} v_{j_{2k+3-\ell}} \in S_2$. Nous considérons, pour $1 \leq \ell \leq k$, les sommes

$$d_\ell := (i_{\ell+1} - i_\ell) + (j_{\ell+1} - j_\ell) + (i_{2k+3-\ell} - i_{2k+2-\ell}) + (j_{2k+3-\ell} - j_{2k+2-\ell}).$$

Maintenant $d_1 + \ldots + d_k \leq n - 4$, par conséquent au moins un de ces d_ℓ, disons d_1 est plus petit que $n/k \leq \varepsilon n$. Alors

$$v_{i_1} v_{i_{2k+2}} v_{i_{2k+2}+1} \cdots v_{j_{2k+1}-1} v_{j_{2k+1}} v_{j_2} v_{j_2+1} \cdots v_{i_{2k+1}-1} v_{i_{2k+1}} v_{i_2} v_{i_2+1} \cdots$$
$$\cdots v_{j_1-1} v_{j_1} v_{j_{2k+2}} v_{j_{2k+2}+1} \cdots v_{i_1-1}$$

est un cycle de longueur cherchée.

Il reste à montrer qu'un des cas précédents apparaît nécessairement, et que la configuration correspondante peut être trouvée en temps linéaire. Si une de ces $n/2$ cordes a une longueur d'au plus $3k^2$, alors elle peut être trouvée en $O(n)$ étapes et la preuve est finie. Donc, supposons dans la suite que chaque corde est plus longue. Dans ce cas, nous pouvons sélectionner une corde e de longueur minimum, en $O(n)$ étapes. Disons, $P_e = v_\ell v_{\ell+1} \cdots v_j$.

Comme e est minimale, chacune des $3k^2 - 3k + 1$ cordes incidentes aux sommets $v_{\ell+1}, v_{\ell+2}, \ldots, v_{\ell+3k^2-3k+1}$ croise e. De plus, il suit du lemme 5.2.3 que nous pouvons trouver parmi elles k cordes qui se croisent mutuellement (Cas 2), ou $2k+1$ cordes parallèles qui croisent e (en fait plus, mais $2k+1$ suffisent pour le moment). Parmi ces $2k+1$ cordes qui croisent e au moins $k+1$ forment une « chaîne », c'est-à-dire que nous pouvons choisir un $(k+1)$-uplet $S := (e_0, e_1, \ldots, e_k)$ de cordes, comme une configuration de départ, telle que $P_{e_0} \subset P_{e_1} \subset \cdots \subset P_{e_k}$.

En un nombre d'étapes qui ne dépend que de k, donc constant sous la condition où k est fixé, il se produit une des deux situations suivantes:
– soit la paire S_1, S_2 est obtenue,
– soit S est modifié de sorte que $||e_k||$ décroît.

La décroissance de $||e_k||$ ne peut pas apparaître plus de $n/2$ fois, par conséquent S_1, S_2 ne peut apparaître après, au plus que $O(n)$ étapes.

Soit X n'importe quel ensemble de $3k^2 - 3k + 1$ sommets intérieurs consécutifs de P_{e_0}, et Q l'ensemble des cordes incidentes avec X. En fait, $|Q| = |X|$ en excluant les cordes courtes (Cas 1). Premièrement nous vérifions en $3k^2 - 3k + 1$ étapes si toute corde $f \in Q$ croise e_0.

– Si ce n'est pas le cas, par exemple si $f_0 \in Q$ est plus petite que e_0, alors nous redéfinissons $e_i := e_{i-1}$ pour $i = k, k-1, \ldots, 1$ dans cet ordre inverse.

Soit après $e_0 := f_0$. Donc, $||e_k||$ a diminué et S reste toujours une chaîne de cordes parallèles.

– Si c'est le cas, alors nous appliquons le lemme 5.2.3 pour trouver soit k membres de Q qui se croisent mutuellement (Cas 2), soit $3k+1$ cordes parallèles $f_0, f_1, \ldots, f_{3k}$ dans Q. Supposons que les extrémités de f_i se suivent toutes dans l'ordre croissant de leurs indices sur P_{e_0}. Nous formons trois groupes avec ces $3k+1$ cordes dans lesquels f_k et f_{2k} apparaissent deux fois:

$$\begin{aligned} S' &:= \{f_i \mid 0 \leq i \leq k\} \\ S'' &:= \{f_i \mid k \leq i \leq 2k\} \\ S''' &:= \{f_i \mid 2k \leq i \leq 3k\}. \end{aligned}$$

Si une certaine corde $f \in S''$ est parallèle à e_k, alors P_f contient entièrement $P_{f'}$ pour toute $f' \in S'$ ou pour toute $f' \in S'''$. Ainsi, en redéfinissant $S := S'$ ou $S := S'''$, la valeur de $||e_k||$ décroît. De plus, si toute $f \in S''$ croise e_k, alors $S_1 := S$ et $S_2 := S''$ satisfont les propriétés demandées (Cas 3).

□

Chapitre 6

Longest Cycle dans les graphes cubiques et Hamiltoniens

Les problèmes Longest Path et Longest Cycle ont été montrés NP-complets en 1972 par Karp [K72]. L'approximation des problèmes d'optimisation associés reste d'actualité bien que des efforts considérables y aient été consacrés.

Dans ce chapitre nous montrons que le problème du plus long chemin n'est pas approximable à une constante près en temps polynomial sur les graphes cubiques et Hamiltoniens, à moins que $P = NP$.

Monien [Mon84] a donné un algorithme pour trouver un chemin de longueur k en temps $O(k!nm)$ où n et m sont respectivement le nombre de sommets et le nombre d'arêtes du graphe. Karger, Motwani et Ramkumar [KMR93] ont donné un algorithme polynomial qui trouve un chemin de longueur $\Omega(\log n)$ dans n'importe quel graphe qui est *1-tough*. Un *graphe* est 1-tough si pour n'importe quel ensemble de sommets non-vide S, le nombre des composantes connexes qui reste en enlevant S du graphe est au plus $|S|$. Un résultat similaire a aussi été obtenu par Fürer et Raghavachari [FR94]. En fait, Fürer et Raghavachari donnent une approximation à un terme additif 1 près en temps polynomial pour le problème qui consiste à trouver dans un graphe un arbre couvrant dont le degré maximum est minimal. Comme un graphe 1-tough contient au moins un arbre couvrant de degré maximum au plus 3 [W78], leur algorithme trouve un arbre de profondeur $\Omega(\log n)$. Un graphe Hamiltonien est 1-tough et donc ces algorithmes peuvent être utilisés en particulier pour trouver de tels chemins dans les graphes qui contiennent un cycle Hamiltonien. Alon, Yuster et Zwick [AYZ94] ont généralisé ce résultat en donnant un algorithme polynomial qui, pour toute constante $c > 0$, trouve un chemin de longueur $c \log n$ dans n'importe quel graphe qui contient un tel chemin. Trouver un chemin de longueur $\omega(\log n)$ en temps polynomial est un problème ouvert même pour les graphes Hamiltoniens.

Du coté négatif, Karger, Motwani et Ramkumar [KMR93] ont prouvé que, à moins que $P{=}NP$, Longest Path n'est pas approximable à un facteur constant près en temps polynomial. Leur preuve est constituée de deux parties. Premièrement, ils ont montré que LONGEST PATH n'a pas de schéma d'approximation en temps polynomial, à moins que $P{=}NP$. Ils ont montré ceci même quand les instances sont restreintes aux graphes Hamiltoniens. Après, ils ont donné un schéma d'auto-amélioration pour le problème, en montrant qu'un algorithme polynomial qui donne une approximation à un facteur constant près pour une *certaine* constante peut être transformé en un algorithme polynomial qui donne une approximation à un facteur constant près pour *n'importe quelle* constante. Ces résultats restent valables aussi dans le cas où le degré maximum du graphe est borné par une constante supérieure à quatre. Mais leur schéma d'auto-amélioration ne conserve pas la propriété que le graphe est Hamiltonien, et ils se demandent si LONGEST PATH reste non approximable à un facteur constant près en temps polynomial pour les graphes Hamiltoniens, sous l'hypothèse où $P \neq NP$.

Dans ce chapitre nous allons montrer un résultat négatif plus fort que la question précédente ouverte jusque là. Nous allons donner un schéma d'auto-amélioration pour LONGEST PATH qui préserve la propriété que le graphe est Hamiltonien quand les graphes sont en plus cubiques. Le fait que LONGEST PATH reste NP-complet même pour les graphes cubiques a été montré par Garey, Johnson et Tarjan [GJT76]. En plus nous montrons que ce problème n'a pas de schéma d'approximation polynomial dans les graphes cubiques et Hamiltoniens, à moins que $P{=}NP$. Ces deux résultats impliquent que LONGEST PATH n'est pas approximable à un facteur constant près en temps polynomial pour aucune constante dans les graphes cubiques et Hamiltoniens, à moins que $P{=}NP$. Un résultat similaire suit immédiatement pour LONGEST CYCLE. Karger, Motwani et Ramkumar [KMR93] ont montré que pour chaque $\varepsilon > 0$, LONGEST PATH n'est pas approximable à un facteur $2^{O(\log^{1-\varepsilon} n)}$ près, à moins que $NP \subseteq DTIME(2^{O(\log^{1/\varepsilon} n)})$. Nous allons montrer le même résultat pour CH LONGEST PATH.

Galbiati, Morzenti et Maffioli [GMM97] ont utilisé le résultat « LONGEST PATH n'est pas approximable en temps polynomial à un facteur constant près dans les graphes Hamiltoniens, à moins que $P = NP$ » pour montrer que les deux problèmes suivants ne sont pas approximables en temps polynomial à un facteur constant près sous l'hypothèse que $P \neq NP$. Le premier est de trouver dans un graphe un arbre couvrant avec une racine fixée qui maximise la somme des distances entre les sommets et la racine. Le deuxième est de trouver dans un graphe un arbre couvrant qui maximise la somme des distances des sommets dans l'arbre.

6.1 Définitions

Étant donnés un graphe G et deux sommets s et t dans G, un *s-t chemin* est un chemin tel que son premier sommet est s et son dernier sommet est t.

Un *graphe* est appelé *cubique avec sommets distingués* s et t si tous ses sommets sont de degré trois sauf s et t qui sont de degré deux, et il existe une arête entre s et t.

Nous définissons formellement ci-dessous les problèmes utilisés dans ce chapitre.

CH Longest Path (Cycle)
Entrée : un graphe cubique et Hamiltonien G.
Sortie : un chemin (cycle) de longueur maximum.

Comme CH Longest Path est facilement L-reductible à CH Longest Cycle, nous allons montrer notre résultat de non-approximation pour CH Longest Path. En fait, pour des raisons techniques il est plus facile de montrer ce résultat pour la variante suivante du problème.

CH Longest s-t Path
Entrée : un graphe cubique et Hamiltonien G avec sommets distingués s et t.
Sortie : un s-t chemin de longueur maximum.

Ce qui est particulier dans ces instances du problème de plus long chemin est que la valeur d'une solution optimum est connue d'avance. Bien que ces instances restent difficiles à approcher, Max 3Sat ne peut pas être L-réductible à eux, comme nous allons le montrer dans la partie qui suit.

Par conséquent, pour montrer tout de même qu'ils n'ont pas de schéma d'approximation en temps polynomial, nous allons réduire à eux un cas spécial de Max 3Sat où la valeur d'une solution optimum est aussi connue. Définissons formellement ces problèmes.

Satisfiable Max 3Sat
Entrée : une formule F avec des variables $x_1, \ldots, x_n$ et des clauses $C_1, \ldots, C_m$ contenant au plus 3 littéraux, où F est satisfaisable.
Sortie : une assignation de valeurs de vérité aux variables qui satisfait un nombre maximum de clauses.

Satisfiable Max 3Sat$(4, \bar{4})$ est la restriction de Satisfiable Max 3Sat aux formules dans lesquelles chaque variable et sa négation apparaissent au plus quatre fois dans F.

6.2 Une réduction

« Trouver le plus long chemin » et « trouver le plus long chemin entre deux sommets fixés » sont deux problèmes avec la même difficulté d'approximation (Yannakakis [Yan96]). Nous décrivons dans la suite la réduction exacte dont nous avons besoin.

Lemme 6.2.1 *Si* CH LONGEST PATH *est approximable à un facteur constant près en temps polynomial alors* CH LONGEST s-t PATH *est aussi approximable à un facteur constant près en temps polynomial.*

Preuve. Supposons que $\mathcal{A}$ est un algorithme polynomial qui donne une approximation à un facteur c près pour CH LONGEST PATH pour une certaine constante $c > 1$. Nous allons construire un algorithme polynomial $\mathcal{B}$ qui donnera une approximation à un facteur $2c$ près pour CH LONGEST s-t PATH.

Soit $G = (V, E)$ une instance de CH LONGEST s-t PATH avec deux sommets distingués s et t. Nous définissons le graphe H qui contient n copies $G_1, \ldots, G_n$ du graphe G, où les sommets distingués de G_i sont s_i et t_i. En plus H contient les arêtes $\{t_i, s_{i+1}\}$ pour $i = 1, \ldots, n-1$ et l'arête $\{t_n, s_1\}$. Évidemment H est cubique et Hamiltonien.

L'algorithme $\mathcal{B}$ exécute $\mathcal{A}$ sur H qui trouve un chemin P de longueur $p \geq (n^2-1)/c$. Ce chemin commence dans une copie de G et après visite une série de copies qui sont toutes distinctes sauf peut être la dernière qui pourrait être la même que la première. Supposons que P visite k copies. À l'intérieur de chaque copie sauf peut être dans la première et la dernière, le chemin induit par P est un s-t chemin dans G. Soit ℓ la longueur du plus long chemin parmi ces s-t chemins. Nous avons

$$p \leq (k-2)\ell + 2(n-1) + k \leq n(\ell+3) - 2,$$

qui implique que $\ell > n/c - 3$. Donc, si $n \geq 6c$, un chemin de longueur ℓ est une approximation à un facteur $2c$ près pour un s-t chemin le plus long dans G. Si $n < 6c$, nous pouvons trouver un s-t chemin Hamiltonien en temps constant en faisant une recherche exhaustive. □

6.3 CH LONGEST s-t PATH n'a pas de schéma d'approximation en temps polynomial

La base de notre résultat de non-approximation est le théorème 2.7.3.

La technique standard pour montrer qu'un problème d'optimisation n'a pas de schéma d'approximation en temps polynomial consiste à montrer un résultat plus fort, qu'il est APX-difficile sous la L-réduction. Mais ici on ne peut pas

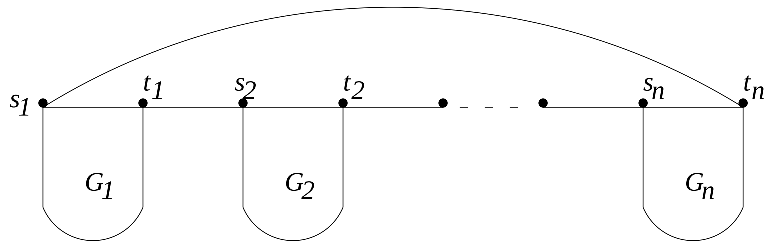


FIG. 6.1 – *Le graphe H*

procéder de cette manière parce que si $NP \neq co\text{-}NP$ alors ce résultat plus fort n'est pas vrai pour des problèmes pour lesquels la valeur d'une solution optimum est connue. Cet argument est analogue à celui utilisé dans le résultat de Megiddo et Papadimitriou [MP91] qui ont montré qu'une fonction FNP-complet ne peut pas être totale, à moins que $NP{=}co\text{-}NP$.

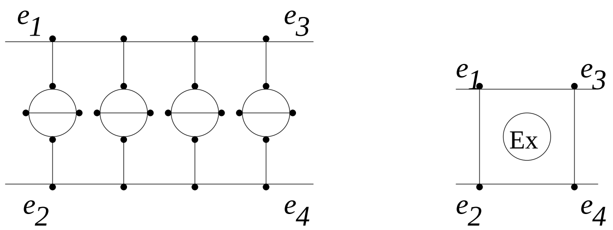
Théorème 6.3.1 *Si $NP \neq co\text{-}NP$ alors un problème d'optimisation pour lequel la valeur d'une solution optimum est connue ne peut pas être APX-difficile sous la L-réduction.*

Preuve. Supposons le contraire. Soit un problème d'optimisation APX-difficile A pour lequel la valeur d'une solution optimum est connue pour chaque instance, et soit L n'importe quel langage de NP. Nous allons montrer que $\bar{L}$, le complément de L, est aussi dans NP. Pour une entrée x, soient F_x l'instance et ε la constante données par le théorème 2.7.3. Puisque MAX 3SAT est approximable à un facteur constant près en temps polynomial, il existe par hypothèse une L-réduction de MAX 3SAT à A. Supposons que cette réduction transforme F_x en une instance x' de A. L'algorithme qui montre que $\bar{L} \in NP$ est le suivant. Sur l'entrée x, on devine une solution optimum y' de x', cela est possible parce que la valeur d'une telle solution est connue. Via la L-réduction on transforme y' en une assignation pour F_x. D'après la définition de la L-réduction, celle-ci doit être une assignation optimum. On accepte x si l'assignation ne satisfait pas plus d'une fraction $(1-\varepsilon)$ des clauses. Le témoin court y' montre que $x \notin L$. □

En utilisant encore le théorème 2.7.3 nous pouvons aussi prouver que SATISFIABLE MAX 3SAT n'a pas de schéma d'approximation en temps polynomial.

Lemme 6.3.2 SATISFIABLE MAX 3SAT *n'a pas de schéma d'approximation en temps polynomial, à moins que $P{=}NP$.*

Preuve. Supposons que SATISFIABLE MAX 3SAT ait un schéma d'approximation en temps polynomial dénoté $\mathcal{A}$. Soit L un langage NP-complet. Nous allons montrer que L peut être reconnu en temps polynomial. Pour une entrée x de L, soit l'instance F_x et la constante $\varepsilon > 0$ données par le théorème 2.7.3, et



(a) (b)

FIG. 6.2 – *La construction ex-or et sa représentation*

soit m le nombre des clauses de F_x. Nous choisissons $\delta < \varepsilon$ et nous appliquons l'algorithme $\mathcal{A}$ à (F_x, δ). Si $x \in L$ alors $\mathcal{A}$ trouve en temps polynomial une assignation satisfaisant au moins $(1-\delta)m > (1-\varepsilon)m$ clauses. Si $x \notin L$ alors $\mathcal{A}$ ne peut pas trouver une assignation satisfaisant plus que $(1-\varepsilon)m$ clauses. Donc, nous pouvons décider en temps polynomial si x est dans L ou non. □

En utilisant maintenant la L-réduction de [PY91] de MAX 3SAT au MAX 3SAT$(4,\bar{4})$, et en observant que les instances satisfaisables sont transformées en instances satisfaisables, nous obtenons le corollaire suivant.

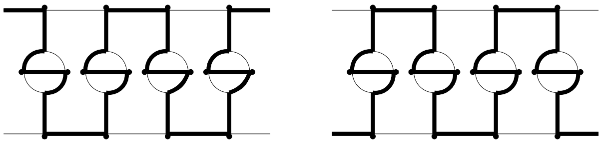
Corollaire 6.3.3 SATISFIABLE MAX 3SAT$(4,\bar{4})$ *n'a pas de schéma d'approximation en temps polynomial, à moins que P=NP.*

Nous allons maintenant montrer le résultat principal de cette section.

Théorème 6.3.4 CH LONGEST s-t PATH *n'a pas de schéma d'approximation en temps polynomial, à moins que P=NP.*

Preuve. Nous allons construire une L-réduction de SATISFIABLE MAX 3SAT$(4,\bar{4})$ à CH LONGEST s-t PATH. La structure de notre construction suit celle de la réduction polynomiale donnée par Papadimitriou et Steiglitz [PS82] de 3SAT au problème de cycle Hamiltonien. Dans [PY93], Papadimitriou et Yannakakis ont donné une L-réduction de MAX 3SAT$(4,\bar{4})$ au TSP(1,2), le problème du voyageur de commerce avec les distances un et deux en exploitant la connexion étroite entre ce dernier problème et le problème de cycle Hamiltonien. Bien que nous donnions une L-réduction qui soit plus contraignante qu'une réduction polynomiale, nous pouvons éviter les complications de la construction de Papadimitriou et Yannakakis car ici nous sommes concernés seulement par des instances satisfaisables de MAX 3SAT$(4,\bar{4})$. Cependant, d'autres difficultés apparaissent car le graphe que nous construisons doit être cubique et Hamiltonien.

En particulier, de manière similaire à [PS82] et [PY93], nous utilisons dans notre L-réduction certaines constructions pour les variables et pour les clauses. La construction correspondant à une variable vient de [PS82]. Elle est plus simple



(a) (b)

FIG. 6.3 – *Deux manières de couvrir la construction ex-or par un chemin entre deux sommets à l'extérieur de la construction*

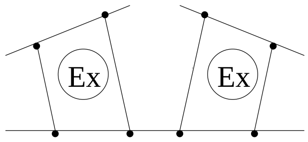


FIG. 6.4 – *Deux constructions ex-or connectées en série*

que celle utilisée dans [PY93]), mais pour la construction correspondant à une clause nous utilisons des caractéristiques supplémentaires.

Un ingrédient de base pour les deux est la modification de la construction ex-or de [PS82] qui est montrée dans la figure 6.2, où seulement les arêtes e_1, e_2, e_3, e_4 sont liées avec le reste du graphe. La seule différence par rapport à la construction ex-or originale est qu'ici tous les sommets sont de degré trois. La construction ex-or a la propriété que n'importe quel chemin qui couvre la construction et qui commence et s'arrête à l'extérieur de celle-ci, utilise soit l'ensemble des arêtes $\{e_1, e_3\}$, soit l'ensemble des arêtes $\{e_2, e_4\}$ comme liaison avec le reste du graphe (comme dans les figures 6.3(a) et 6.3(b)). Une autre propriété de la construction ex-or est qu'il est impossible d'avoir deux chemins disjoints qui commencent et s'arrêtent à l'extérieur de la construction et qui contiennent chacun au moins un sommet de la construction, et qui ensemble couvrent la construction. Les constructions ex-or peuvent être connectées en série comme dans la figure 6.4.

Soit F une instance de SATISFIABLE MAX 3SAT$(4,\bar{4})$ avec n variables et m clauses. Pour chaque variable nous allons considérer une construction correspondant à la variable et pour chaque clause une construction correspondant à la clause.

Pour $1 \leq i \leq n$, soit p_i le nombre d'occurrences positives de x_i dans F et soit r_i le nombre d'occurrences négatives. Pour chaque i, la construction correspondant à la variable x_i est la suivante : pour deux sommets spécifiques u_i et v_i, il existe deux chemins entre u_i et v_i. À un de ces chemins sont attachées p_i constructions ex-or connectées en série comme dans la figure 6.4, et nous disons qu'elles *représentent* x_i. À l'autre chemin sont attachées r_i constructions ex-or en série qui représentent $\bar{x}_i$. Si $p_i = 0$ ou $r_i = 0$ alors le chemin correspondant est constitué d'une arête seule. La figure 6.5 montre la construction correspondant à une variable pour une variable avec $p_i = 1$ et $r_i = 2$.

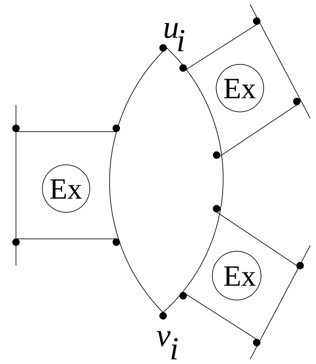


FIG. 6.5 – *La construction correspondant à une variable avec une occurrence positive et deux négatives*

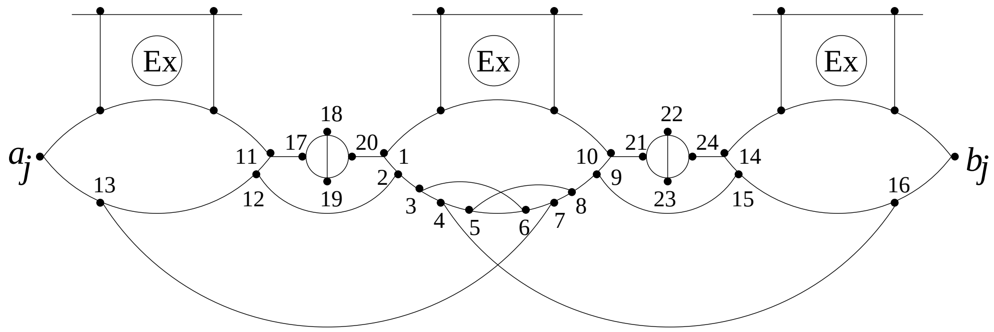


FIG. 6.6 – *La construction correspondant à la clause C_j*

La construction correspondant à la clause C_j est montrée dans la figure 6.6, où les trois constructions ex-or représentent les trois littéraux qui apparaissent dans cette clause. Si C_j contient le littéral x_i, alors la construction correspondant à la clause C_j et la construction correspondant à la variable x_i vont partager une construction ex-or qui représente x_i dans cette dernière construction correspondant à la variable. Si C_j contient $\bar{x}_i$, alors les mêmes constructions partagent aussi une construction ex-or qui maintenant représente $\bar{x}_i$ dans la construction correspondant à la variable. La propriété spécifique satisfaite par les constructions correspondant aux clauses est établie par le lemme suivant.

Lemme 6.3.5 *Pour tout sous-ensemble $S \neq \emptyset$ de trois constructions ex-or dans la construction correspondant à la clause C_j, il existe un chemin de a_j à b_j qui contient exactement les sommets de la construction correspondant à la clause qui ne sont pas dans S. D'autre part, il n'existe pas de chemin de a_j à b_j qui contient tous les sommets de la construction correspondant à la clause C_j.*

Preuve. Nous donnons pour tous les sous-ensembles non vides S le chemin Hamiltonien correspondant.

Si S contient toutes les trois constructions ex-or alors le chemin est: a_j, 13, 12, 11, 17, 18, 19, 20, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 21, 22, 23, 24, 14, 15, 16, b_j.

Si S contient la deuxième et la troisième construction ex-or alors le chemin est: a_j, Ex, 11, 17, 18, 19, 20, 1, 2, 12, 13, 7, 6, 3, 4, 5, 8, 9, 10, 21, 22, 23, 24, 14, 15, 16, b_j.

Pour le sous-ensemble qui contient la première et la deuxième construction ex-or le chemin est symétrique. Si S contient la première et la troisième construction ex-or alors le chemin est: a_j, 13, 7, 6, 3, 2, 12, 11, 17, 18, 19, 20, 1, Ex, 10, 21, 22, 23, 24, 14, 15, 9, 8 , 5, 4, 16, b_j.

Si S contient la troisième construction ex-or alors le chemin est: a_j, Ex, 11, 17, 18, 19, 20, 1, Ex, 10, 21, 22, 23, 24, 14, 15, 9, 8, 7, 13, 12, 2, 3, 6, 5, 4, 16, b_j. Pour le sous-ensemble qui contient la première construction ex-or le chemin est symétrique. Si S contient la deuxième construction ex-or alors le chemin est: a_j, Ex, 11, 17, 18, 19, 20, 1, 2, 12, 13, 7, 8, 5, 6, 3, 4, 16, 15, 9, 10, 21, 22, 23, 24, 14, Ex, b_j.

Finalement il n'existe pas de chemin de a_j à b_j qui couvre les trois constructions ex-or et qui contient les sommets $17, \ldots, 24$ et qui peut contenir un autre sommet de la construction correspondant à la clause. Un chemin entre a_j et b_j qui contient le plus possible de sommets de la construction correspondant à la clause C_j est le chemin a_j, Ex, 11, 17, 18, 19, 20, 1, Ex, 10, 9, 8, 7, 13, 12, 2, 3, 6, 5, 4, 16, 15, 14, Ex, b_j. Ce chemin contient tous les sommets de la construction correspondant à la clause C_j sauf 4 sommets 21, 22, 23, 24. □

Le graphe G (voir la figure 6.7) contient toutes les constructions correspondant aux variables et clauses et deux sommets supplémentaires s et t. En plus des arêtes des constructions, il existe une arête entre s et u_1, entre v_i et u_{i+1} pour $1 \leq i \leq n-1$, entre v_n et a_1, entre b_j et a_{j+1} pour $1 \leq j \leq m-1$, entre b_m et t, et entre s et t.

Nous justifions ci-dessous que G est une instance de CH LONGEST s-t PATH.

S'il existe une assignation satisfaisante T pour la formule F alors le chemin suivant est Hamiltonien : il commence en s, il couvre les constructions ex-or qui représentes un littéral satisfait par T dans les constructions correspondant aux variables, il traverse les constructions correspondant aux clauses selon le lemme 6.3.5 où S représente chaque fois l'ensemble de littéraux satisfaits et il s'arrête en t.

G est aussi cubique sauf les sommets s et t qui ont pour degré deux.

Nous allons montrer dans la suite que la réduction est effectivement une L-réduction.

Soit N le nombre des sommets de G, alors la longueur d'un plus long s-t chemin est $N-1$. Le nombre de clauses m dans F est aussi la valeur d'une assignation optimum pour l'instance F de SATISFIABLE MAX 3SAT$(4, \bar{4})$. Clairement

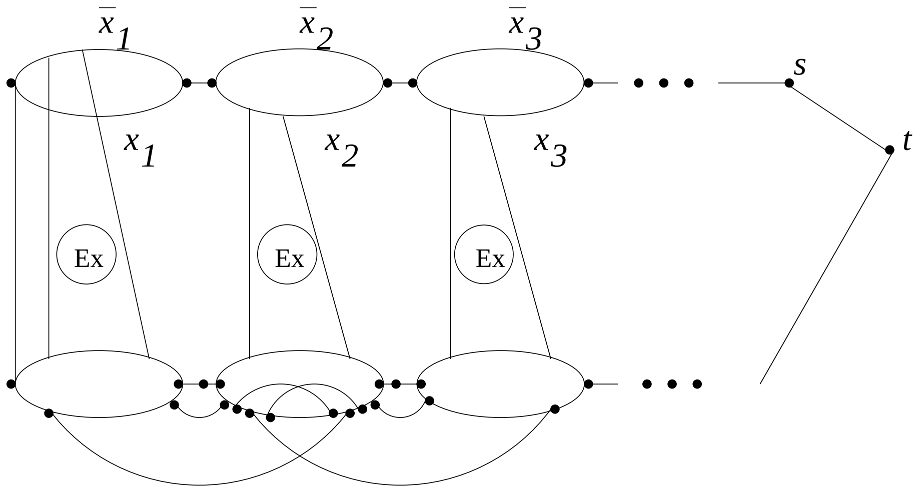


FIG. 6.7 – *Le graphe G quand la première clause de F est $\bar{x}_1 \wedge x_2 \wedge x_3$*

$m = \Theta(n)$ parce que chaque littéral apparaît seulement un nombre constant de fois dans la formule. Comme les constructions correspondant aux variables et aux clauses ont un nombre constant de sommets, $N = \Theta(m+n) = \Theta(n)$, ceci montre que la première condition de la L-réduction est satisfaite.

Pour montrer la deuxième condition considérons un s-t chemin P arbitraire dans G. Un sommet est dit *absent* lorsqu'il n'est pas sur ce chemin.

Nous construisons maintenant à partir de P une assignation partielle T_P pour la formule F. T_P va donner une valeur à toutes les variables pour lesquelles la construction correspondant à la variable est traversée correctement par P pour x_i ou $\bar{x}_i$. Nous disons qu'une construction ex-or est *accessible* par P *à partir de* la construction correspondant à la variable, si P utilise au moins une des deux arêtes de la construction ex-or parmi $\{e_1, e_2, e_3, e_4\}$ qui se trouve sur un des deux chemins entre u et v associés à la construction correspondant à la variable. Nous disons que P *traverse correctement* la construction correspondant à la variable x_i *pour* x_i s'il couvre toutes les constructions ex-or représentant x_i, ces constructions ex-or sont accessibles par P à partir de la construction correspondant à la variable, et aucune construction représentant $\bar{x}_i$ n'est accessible à partir de la construction correspondant à la variable. Dans ce cas T_P attribue la valeur **vraie** à x_i. De même, lorsque P traverse correctement la construction correspondant à la variable x_i pour $\bar{x}_i$, T_P attribue la valeur **fausse** à x_i.

Lemme 6.3.6 *Si le chemin P a k sommets absents, alors l'assignation partielle T_P satisfait au moins $m - 8k$ clauses.*

Preuve. Supposons qu'une clause C_j soit non satisfaite par T_P. Alors soit ses trois littéraux sont faux en T_P, soit au moins un de ses littéraux n'a pas reçu de valeur de vérité.

Dans le premier cas, d'après la définition de T_P, la construction correspondant à la variable de chaque littéral a été traversée correctement pour la négation de ce littéral. Par conséquent, les seuls sommets où P peut entrer et quitter la construction correspondant à la clause C_j sont a_j et b_j, et il doit exister un sommet absent dans cette construction grâce au lemme 6.3.5. Dans ce cas, nous associons à ce sommet absent la clause C_j.

Dans le deuxième cas il doit exister un sommet absent dans la construction correspondant à la variable sans valeur de vérité. À ce sommet absent nous associons la clause C_j.

Comme chaque variable et sa négation apparaissent ensemble au plus 8 fois dans F, à chaque sommet absent nous avons associé au plus 8 clauses non satisfaites. Comme il y a au plus k sommets absents, le lemme est prouvé. □

Pour finir la preuve du théorème 6.3.4, nous montrons maintenant que la deuxième condition de la définition de la L-réduction est aussi satisfaite. Comme F est satisfaisable, sa valeur optimum est m, et comme G a un cycle Hamiltonien, sa valeur optimum est $N-1$. Considérons qu'un s-t chemin P de longueur $N-1-\ell$ est donné. Alors il existe ℓ sommets absents dans le graphe. Soit T une assignation qui étend T_P. Selon le lemme 6.3.6, T satisfait au moins $m - 8\ell$ clauses de F. Donc la deuxième condition est satisfaite avec $\beta = 8$. □

Remarque. Une démonstration très similaire montre que si $P{\neq}NP$, alors le problème de voyageur de commerce avec poids 1 et 2, restreint aux instances pour lesquelles le graphe formé des arêtes de poids un est cubique et Hamiltonien, n'a pas de schéma d'approximation en temps polynomial.

La construction de la L-réduction est la même, le graphe G obtenu étant la restriction du graphe complet sur N sommets aux arêtes de poids un. Donc les arêtes qui ne sont pas dans G sont des arêtes de poids deux.

Un tour entre deux sommets fixés est un chemin qui commence à un des deux sommets, passe par tous les sommets du graphe exactement une fois et s'arrête à l'autre sommet fixé. Nous allons définir à partir d'un tour P, entre s et t, qui contient k sommets mauvais, une assignation partielle T_P qui satisfait au moins $m - 8k$ clauses. Un sommet est *mauvais* s'il est adjacent dans P à au moins une arête de poids deux. S'il est adjacent à deux arêtes de poids deux, on compte ce sommet comme deux sommets mauvais. Dans T_P la variable x_i prend la valeur **vrai** (et resp. **faux**) s'il n'existe pas de sommet mauvais dans les constructions ex-or correspondant à x_i et $\bar{x}_i$, et si P traverse correctement la construction correspondant à la variable x_i pour x_i (et resp. pour $\bar{x}_i$). Sinon, la variable x_i ne prend pas de valeur.

Pour montrer que l'assignation T_P satisfait au moins $m - 8k$ clauses on suit la preuve du lemme 6.3.6. Si une clause C_j n'est pas satisfaite parce que les trois littéraux sont faux, les trois constructions ex-or correspondantes sont accessibles par P à partir de la construction correspondant à la clause. Dans ce cas il existe

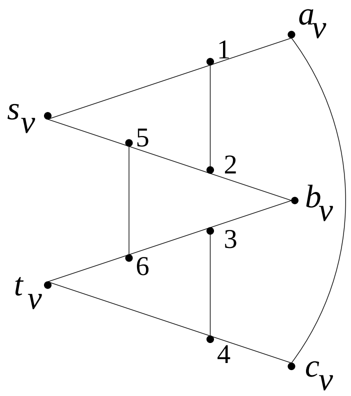


FIG. 6.8 – *Le connecteur C_v*

au moins un sommet mauvais parmi les sommets de la construction correspondant à C_j. Si une clause C_j n'est pas satisfaite parce que au moins un littéral n'a pas de valeur de vérité en T_P, alors il existe un sommet mauvais dans la construction correspondant à la variable sans valeur de vérité.

Comme dans le lemme précédent on peut montrer, en associant les clauses fausses aux sommets mauvais, qu'au moins $m - 8k$ clauses sont satisfaites par T_P. Puisque le poids du tour P est $N - 1 + k/2$, la deuxième contrainte de la L-réduction est satisfaite avec $\beta = 16$.

6.4 CH LONGEST s-t PATH n'est pas dans APX

Dans cette partie nous allons donner un schéma d'auto-amélioration pour CH LONGEST s-t PATH qui implique qu'un algorithme polynomial qui donne une approximation à un facteur constant près pour une certaine constante peut être transformé en un algorithme polynomial qui donne une approximation à un facteur constant près pour n'importe quelle constante.

Étant donnée une instance $G = (V, E)$ de CH LONGEST s-t PATH avec les sommets distingués s et t, nous définissons le graphe *carré* G^2 de G qui sera une instance du même problème. L'idée de base est de remplacer dans G chaque sommet v par une copie G_v de G et par un *connecteur* C_v. La copie du connecteur pour v est montré dans la figure 6.8. Ce graphe C_v va connecter G_v avec le reste de G^2 au moyen des sommets a_v, b_v, c_v que nous appelons sommets *extérieurs*.

Une propriété importante du connecteur est établie dans le lemme suivant.

Lemme 6.4.1 *Pour chaque ensemble $\{x, y\} \subseteq \{a_v, b_v, c_v\}$ il existe deux chemins P_x et P_y tels que :*
- P_x commence en x et P_y commence en y ;
- P_x et P_y sont disjoints ;

- *leur réunion contient tous les sommets du connecteur ;*
- *P_x et P_y s'arrêtent en s_v et t_v (ou t_v et s_v).*

Preuve. Nous décrivons ici les chemins P_x et P_y pour les trois sous-ensembles. Si $\{x, y\} = \{a_v, b_v\}$ nous prenons $P_x = a_v, c_v, 4, t_v$ et $P_y = b_v, 3, 6, 5, 2, 1, s_v$. Les chemins sont symétriques pour l'ensemble $\{b_v, c_v\}$. Si $\{x, y\} = \{a_v, c_v\}$ nous pouvons prendre $P_x = a_v, 1, s_v$ et $P_y = c_v, 4, 3, b_v, 2, 5, 6, t_v$. □

Le graphe G^2 contient une copie G_v de G et une copie C_v du connecteur pour chaque sommet v sauf s et t. G a aussi deux sommets distingués s' et t'. Pour chaque v, nous identifions les sommets distingués de G_v avec les sommets s_v et t_v de C_v, et nous effaçons l'arête $\{s_v, t_v\}$. Nous dénotons le graphe obtenu par H_v, et nous l'appelons le *composant* correspondant à v.

Les composants sont connectés dans G^2 par les arêtes suivantes appelées arêtes *extérieures*. D'abord pour chaque sommet v différent de s et t nous étiquetons les trois arêtes de G incidentes à v de manière bijective par $\{a_v, b_v, c_v\}$. Pour chaque arête $\{v, w\} \in E$, nous allons poser une arête dans G^2 entre les sommets extérieurs de C_v et C_w qui coïncident avec les deux étiquettes de cette arête.

Soit x (resp. y) le voisin de s (resp. t) dans G différent de t (resp. s). Nous ajoutons une arête entre s' et le sommet extérieur de C_x qui coïncide avec l'étiquette de l'arête $\{x, s\}$ et une arête entre t' et le sommet extérieur de C_y qui coïncide avec l'étiquette de l'arête $\{y, t\}$. Finalement nous ajoutons une arête entre s' et t' (voir la figure 6.9).

Remarquons que s'il existe un s-t chemin Hamiltonien dans G, alors il existe un s'-t' chemin Hamiltonien dans G^2. Un s'-t' chemin Hamiltonien dans G^2 utilise donc les mêmes arêtes que le s-t chemin Hamiltonien dans G dans chaque copie de G dans G^2 ; de plus ces chemins sont connectés d'après le lemme 6.4.1.

Nous montrons ensuite un lemme qui sera utile pour démontrer le résultat principal de cette section.

Lemme 6.4.2 *À partir de chaque s'-t' chemin dans G^2 de longueur ℓ, on peut trouver en temps polynomial un s-t chemin dans G de longueur $\sqrt{\ell} - 10$.*

Preuve. Soit P un chemin de longueur ℓ dans G^2 entre s' et t'. Pour chaque sommet v, P peut entrer (et quitter) le graphe H_v via les sommets extérieurs du connecteur. Comme il y a trois sommets extérieurs, cette situation se produit au plus une fois. Par conséquent, P induit un chemin unique P_v dans chaque H_v. Aussi, ce chemin est décomposé en deux sous-chemins P_v^1 et P_v^2 dans C_v, et un sous-chemin P_v' dans G_v, et ce parce que G_v et le connecteur C_v sont connectés seulement via les sommets s_v et t_v. P_v' est clairement un s-t chemin dans G. La somme des longueurs de P_v^1 et P_v^2 est au plus 9.

Supposons que P a k arêtes extérieures. Ces arêtes induisent naturellement un s-t chemin de longueur k dans G. Aussi, P visite $k-1$ composants. Soit h la

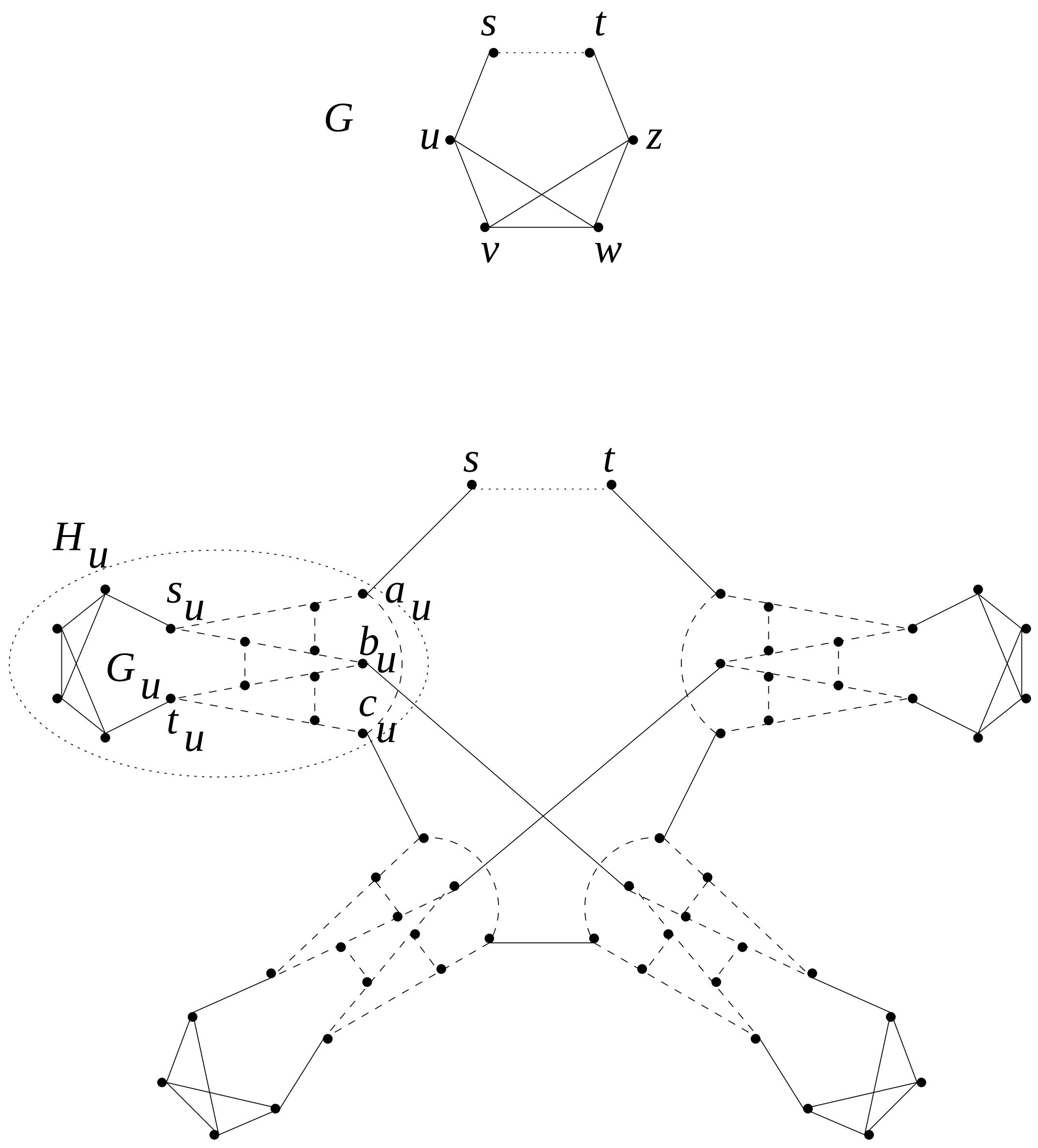


FIG. 6.9 – *Le graphe* G^2

longueur maximum de P'_v parmi tous les composants visités. Nous allons montrer que k ou h est au moins $\sqrt{\ell} - 10$. Supposons le contraire. Alors la longueur de P est au plus $k + (k-1)(h+9) < \ell$, d'où une contradiction. □

En utilisant des arguments standards, ce schéma d'auto-amélioration couplé avec le théorème 6.3.4 donnent le théorème suivant.

Théorème 6.4.3 CH LONGEST s-t PATH *n'est pas approximable à un facteur constant près en temps polynomial, à moins que P=NP.*

Preuve. Supposons qu'il existe un algorithme polynomial $\mathcal{A}$ qui donne une approximation à un facteur c près pour une certaine constante $c > 1$. Nous allons prouver que pour chaque constante $\varepsilon > 0$, nous pouvons construire un algorithme polynomial qui donne une approximation à un facteur $(1+\varepsilon)$ près, ce qui contredit le théorème 6.3.4.

Soit G un graphe cubique et Hamiltonien avec les sommets distingués s et t, et soit n le nombre des sommets de G. Si $n < 10/(c^{-1/2} - c^{-2/3})$ alors nous pouvons trouver une approximation à un facteur de $(1+\varepsilon)$ près en faisant une recherche exhaustive en temps constant. Sinon, soit $k = \lceil \log_{2/3} \log_c (1+\varepsilon) \rceil$.

Pour $i = 1, \ldots, k$ nous construisons G^{2^i}, où par définition G^{2^i} est le graphe carré de $G^{2^{i-1}}$. Ensuite, nous appliquons l'algorithme $\mathcal{A}$ sur G^{2^k} pour trouver une approximation à un facteur c près, P_k, d'un s-t chemin Hamiltonien dans G^{2^k}, puis nous itérons l'algorithme du lemme 6.4.2 pour trouver une approximation P_i d'un s-t chemin Hamiltonien dans G^{2^i}, pour $i = k-1, \ldots, 0$. En utilisant le lemme 6.4.2, un calcul simple montre que si le nombre de sommets de $G^{2^{k-1}}$ est plus grand que $10/(c^{-1/2} - c^{-2/3})$ alors P_{k-1} est une approximation à un facteur $c^{2/3}$ près d'un s-t chemin Hamiltonien dans $G^{2^{k-1}}$. Ainsi, P_i est une approximation à un facteur $c^{(2/3)^{k-i}}$ près. Un autre calcul simple montre que grâce au choix de k, P_0 est une approximation à un facteur $(1+\varepsilon)$ près d'un chemin optimum dans G. Pour ε constant, la taille de G^{2^k} et le temps de calcul de l'algorithme sont polynomiaux en n. □

Nos principaux résultats négatifs découlent immédiatement du lemme 6.2.1 et du théorème 6.4.3.

Théorème 6.4.4 CH LONGEST PATH *et* CH LONGEST CYCLE *ne sont pas dans APX, à moins que P=NP.*

6.5 CH LONGEST s-t PATH n'est pas approximable à un facteur $2^{O(\log^{1-\varepsilon} n)}$ près

Ce résultat de non-approximation plus fort nécessite une hypothèse plus forte.

Théorème 6.5.1 *Pour chaque $\varepsilon > 0$,* CH LONGEST PATH *et* CH LONGEST

CYCLE *ne sont pas approximables en temps polynomial à un facteur* $2^{O(\log^{1-\varepsilon} n)}$ *près, à moins que* $NP \subseteq DTIME(2^{O(\log^{1/\varepsilon} n)})$.

Pour montrer ce théorème, on montre d'abord un résultat analogue pour CH LONGEST s-t PATH, puis une variante plus forte du lemme 6.2.1 : si CH LONGEST PATH est approximable en temps polynomial à un facteur $2^{O(\log^{1-\varepsilon} n)}$ près alors CH LONGEST s-t PATH est approximable en temps polynomial à un facteur $2^{O(\log^{1-\varepsilon} n)}$ près.

Théorème 6.5.2 *Pour chaque* $\varepsilon > 0$, CH LONGEST s-t PATH *n'est pas approximable en temps polynomial à un facteur* $2^{O(\log^{1-\varepsilon} n)}$ *près, à moins que* $NP \subseteq DTIME(2^{O(\log^{1/\varepsilon} n)})$.

Preuve. Supposons que $\mathcal{A}$ soit un algorithme polynomial qui soit une approximation à un facteur $2^{O(\log^{1-\varepsilon} n)}$ près pour CH LONGEST s-t PATH, pour un $\varepsilon > 0$ fixé. Soit G une instance de CH LONGEST s-t PATH avec n sommets. Pour $i = 1, \ldots, p$, où p est un entier qui sera fixé dans la suite, nous construisons G^{2^i}, où par définition G^{2^i} est le graphe carré de $G^{2^{i-1}}$. Dénotons par N_p le nombre de sommets du graphe G^{2^p}.

Soit p le plus grand entier tel que

$$n^{2^p} \leq N_p \leq 2^{\log^{1/\varepsilon} n}.$$

Si n est plus grand qu'une certaine constante alors

$$n^{2^p} \leq N_p < n^{2^{p+1}}.$$

Donc,

$$n^{2^p} \leq N_p \leq 2^{\log^{1/\varepsilon} n} \leq n^{2^{p+2}}.$$

Ensuite, nous appliquons l'algorithme $\mathcal{A}$ sur G^{2^p} pour trouver une $2^{O(\log^{1-\varepsilon} n)}$-approximation P_p d'un s-t chemin Hamiltonien de G^{2^p}. Puis nous itérons l'algorithme du lemme 6.4.2 pour trouver une approximation P_i d'un s-t chemin Hamiltonien dans G^{2^i}, pour $i = p-1, \ldots, 0$.

L'algorithme $\mathcal{A}$ trouve un chemin de longueur au moins $N_p/f(N_p)$ dans G^{2^p}, où $f(N_p) = 2^{O(\log^{1-\varepsilon} N_p)}$. En utilisant le lemme 6.4.2 on peut obtenir un chemin de longueur au moins $n/f(N_p)^{1/2^p} - 10p$ dans G. Soit $g(n) = f(N_p)^{1/2^p}$. Nous allons montrer que $g(n) = O(1)$.

$$g(n) = f(N_p)^{1/2^p} = 2^{O(\frac{\log^{1-\varepsilon} N_p}{2^p})} \leq 2^{O(\frac{\log^{1/\varepsilon} n}{\log n} \cdot \frac{4 \log n}{\log^{1/\varepsilon} n})} = O(1)$$

Comme $p = O(\log \log n)$ on obtient une approximation à un facteur constant près dans G en temps $2^{O(\log^{1/\varepsilon} n)}$. Nous avons montré dans le théorème 6.4.3 que « trouver une approximation à un facteur constant près pour CH LONGEST s-t PATH » est un problème NP-difficile. Cela implique que $NP \subseteq DTIME(2^{O(\log^{1/\varepsilon} n)})$.

□

Remarque. Considérons la réduction du lemme 6.2.1. Il est facile de voir que si $n < 6 \cdot 2^{O(\log^{1-\varepsilon} n)}$, alors on peut trouver un s-t chemin Hamiltonien dans G en temps constant (qui ne dépend que de ε), et si $n \geq 6 \cdot 2^{O(\log^{1-\varepsilon} n)}$ alors on obtient un chemin dans G qui est une approximation à un facteur $2^{O(\log^{1-\varepsilon} n)}$ près d'un s-t chemin Hamiltonien de G.

Chapitre 7

Conclusions

Dans cette thèse nous avons étudié, pour la première fois à notre connaissance, l'approximation de certains problèmes totaux pour lesquels aucun algorithme polynomial n'est connu. La classe des problèmes totaux, $TFNP$, ainsi que ses sous-classes auxquelles nous nous sommes intéressés ont été définies par Meggido et Papadimitriou. Une propriété intéressante de la classe $TFNP$ est qu'elle se situe « entre » FP et FNP, les classes des problèmes de recherche associés aux problèmes de P et NP. Plus précisément, sous certaines hypothèses acceptées en complexité, il existe des problèmes totaux qui ne sont pas dans FP et aucun problème total ne peut être FNP-difficile sous la F-réduction.

Les résultats que nous avons obtenus pour les problèmes totaux sont des résultats d'approximation. Pour montrer qu'un problème n'est pas soluble en temps polynomial il est suffit de montrer qu'il est difficile dans une classe de problèmes différente de FP sous certaines hypothèses acceptées en complexité. Un tel résultat est difficile à établir pour les problèmes totaux sachant qu'ils ne sont pas FNP-complets. Parmi les trois classes de problèmes totaux considérées, PLS, PPP, PPA, la classe PPP est la seule classe montrée non incluse dans FP sous l'hypothèse $P \neq UP$. Un résultat similaire pour les deux autres classes serait très intéressant.

Le seul problème PPP-complet connu est PIGEONHOLE CIRCUIT. PIGEONHOLE SUBSET-SUM est un problème plus naturel de la même classe. Il serait intéressant de montrer que PIGEONHOLE SUBSET-SUM est PPP-complet ou de définir une classe non-triviale dans laquelle il serait complet. Papadimitriou [Pap94b] a conjecturé que le problème total qui consiste à trouver dans un graphe avec les degrés impairs un autre cycle Hamiltonien est PPA-complet. Aucun résultat de complétude pour PPA n'est connu. Il serait intéressant d'établir si un des deux cas suivants est vrai : SECOND HAMILTONIAN CYCLE est dans FP ou PPA-complet.

Une perspective intéressante serait d'identifier d'autres problèmes naturels qui se trouvent dans les classes déjà définies et de définir d'autres classes de problèmes totaux pour lesquels la totalité est montrée en utilisant d'autres principes combinatoires.

Annexe A

Problèmes d'optimisation

Nous décrivons ici les problèmes d'optimisation que nous utilisons dans ce document.

MAX SAT
Entrée : une formule booléenne ϕ dans la forme CNF (conjonction de clauses où chaque clause est une disjonction de littéraux) sur un ensemble de variables. Une assignation de valeurs de vérité aux variables *satisfait* une formule booléenne ϕ si en remplaçant les variables par leurs valeurs, la formule ϕ s'évalue à vrai. Si une telle assignation existe, on dit que ϕ est satisfaisable.
Sortie : une assignation qui satisfait un nombre maximum de clauses de ϕ.

MAX kSAT, $k \geq 2$
Entrée : une formule booléenne ϕ dans la forme $kCNF$ (chaque clause contient au plus k littéraux).
Sortie : une assignation qui satisfait un nombre maximum de clauses de ϕ.

SATISFIABLE MAX 3SAT
Entrée : une formule ϕ dans la forme $3CNF$, satisfaisable.
Sortie : une assignation qui satisfait un nombre maximum de clauses de ϕ.

MAX 3SAT$(4, \bar{4})$
Entrée : une formule booléenne ϕ dans la forme $3CNF$ dans laquelle chaque variable et sa négation apparaissent au plus quatre fois.
Sortie : une assignation qui satisfait un nombre maximum de clauses de ϕ.

LINEAR PROGRAMMING
Entrée : deux entiers m et n, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$ et A une matrice $m \times n$ avec les éléments $a_{ij} \in \mathbb{Z}$.
Sortie : un vecteur $x \in \mathbb{R}^n$ tel que $Ax = b$ et tel que cx a une valeur maximum.

Max Cut
Entrée : un graphe G.
Une *coupe* est une partition des sommets en deux parties. La taille d'une coupe est le nombre des arêtes avec les extrémités dans des parties différentes.
Sortie : une coupe de G de taille maximum.

Max kCut
Entrée : un graphe G.
Une *k-coupe* est une partition des sommets en k parties. La taille d'une k-coupe est le nombre des arêtes avec les extrémités dans des parties différentes.
Sortie : une k-coupe de G de taille maximum.

Max Independent Set
Entrée : un graphe G.
Un ensemble de sommets I est un *ensemble stable* de G s'il n'y a pas d'arête entre des sommets de I.
Sortie : un ensemble stable de G de taille maximum.

Max Clique
Entrée : un graphe G.
Une *clique* dans un graphe est un ensemble de sommets liés deux à deux par une arête.
Sortie : une clique dans G de taille maximum.

Max Bisection
Entrée : un graphe G.
Une *bisection* est une partition des sommets en deux parties de taille égale. La taille d'une bisection est le nombre des arêtes avec les extrémités dans les deux parties différentes.
Sortie : une bisection de G de taille maximum.

Knapsack
Entrée : n objets avec les poids $w_1, \ldots, w_n$, les valeurs $v_1, \ldots, v_n$ et un entier k représentant la capacité du sac-à-dos.
Sortie : un sous-ensemble de ces n objets avec un poids qui ne dépasse pas la capacité k du sac-à-dos et avec une valeur maximum.

Subset-Sum
Entrée : un ensemble $\{a_1, \ldots, a_n\}$ d'entiers positifs et un autre entier b.
Sortie : un sous-ensemble $S \subseteq \{1, \ldots, n\}$ avec

$$\sum_{i \in S} a_i \leq b$$

tel que la somme

$$\sum_{i \in S} a_i$$

soit maximum.

Longest Path
Entrée : un graphe G.
Un *chemin* de longueur k dans G est une séquence de sommets distincts $v_0, v_1, \ldots, v_k$ tels que pour $0 \leq i \leq k-1$, il existe une arête entre v_i et v_{i+1}.
Sortie : un chemin de longueur maximum.

Longest Cycle
Entrée : un graphe G.
Un *cycle* est un chemin de longueur au moins trois pour lequel le premier et le dernier sommet coïncident.
Sortie : un cycle de longueur maximum.

CH Longest Path (Cycle)
Entrée : un graphe cubique et Hamiltonien G.
Sortie : un chemin (cycle) de longueur maximum.

CH Longest s-t Path
Entrée : un graphe cubique et Hamiltonien G avec sommets distingués s et t.
Un *graphe* est appelé *cubique avec sommets distingués* s et t si tous ses sommets sont de degré trois sauf s et t qui sont de degré deux, et il existe une arête entre s et t.
Un *s-t chemin* est un chemin tel que son premier sommet est s et son dernier sommet est t.
Sortie : un s-t chemin de G de longueur maximum.

Another Longest Path
Entrée : un graphe G et un chemin Hamiltonien P de G.
Sortie : un chemin de G ou de $\bar{G}$ différent de P de longueur maximum.

Another Longest Cycle
Entrée : un graphe cubique et Hamiltonien G et un cycle Hamiltonien C.
Sortie : un cycle différent de C de longueur maximum.

Max Satisfiability
Entrée : un système de m équations en n variables sur $\mathbb{Q}$.
Un système d'équations est *satisfaisable* s'il a une solution.
Sortie : un ensemble maximum d'équations qui forme un système satisfaisable.

Max Ones Sat
Entrée : un ensemble de m formules en forme CNF sur n variables booléennes $x_1, \ldots, x_n$. À chaque variable booléenne est associé un poids positif.
Sortie : une assignation qui satisfait toutes les contraintes et maximise la somme des poids des variables assignées à **vrai**.

Travelling Salesperson Problem (abrégé TSP)
Entrée : un graphe complet sur n sommets et des poids sur les arêtes du graphe qui correspondent à la distance entre les 2 sommets.
Sortie : un cycle Hamiltonien de poids minimum (le poids du cycle étant la somme des poids des arêtes).

TSP(1,2)
Entrée : un graphe complet sur n sommets et les poids 1 ou 2 sur les arêtes du graphe.
Sortie : un cycle Hamiltonien de poids minimum.

Subset-Sum Ratio
Entrée : un ensemble $\{a_1, \ldots, a_n\}$ d'entiers positifs.
Sortie : deux sous-ensembles disjoints et non vides $S_1, S_2 \subseteq \{1, \ldots, n\}$ avec

$$\sum_{i \in S_1} a_i \geq \sum_{i \in S_2} a_i$$

tels que le rapport

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i}$$

soit minimum.

Subset-Sum Difference
Entrée : un ensemble $\{a_1, \ldots, a_n\}$ d'entiers positifs.
Sortie : deux sous-ensembles disjoints et non vides $S_1, S_2 \subseteq \{1, \ldots, n\}$, avec

$$\sum_{i \in S_1} a_i \geq \sum_{i \in S_2} a_i$$

tels que la différence

$$\sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i + 1$$

soit minimum.

Min Vertex Cover
Entrée : un graphe G.
Un ensemble de sommets C est une *couverture* de G si toute arête de G a au moins une extrémité dans C.
Sortie : une couverture de G de taille minimum.

Min Dominating Set
Entrée : un graphe G.
Un *ensemble dominant* de G est un ensemble des sommets S tel que chaque sommet de G est soit dans S, soit lié avec un sommet du S.
Sortie : un ensemble dominant de G de taille minimum.

Min Set Cover
Entrée : une collection $\mathcal{C}$ de sous-ensembles d'un ensemble fini S.
Une *couverture* pour S est un sous-ensemble $\mathcal{C}'$ de $\mathcal{C}$ tel que chaque élément de S appartient à au moins un élément de $\mathcal{C}'$.
Sortie : une couverture pour S de taille minimum.

Min Bisection
Entrée : un graphe G.
Une *bisection* est une partition des sommets en deux parties de taille égale. La taille d'une bisection est le nombre des arêtes avec les extrémités dans ces deux parties différentes.
Sortie : une bisection de G de taille minimum.

Min Coloring
Entrée : un graphe G.
Une *coloration* est une partition des sommets en un certain nombre de parties telle que deux sommets d'une même partie ne sont pas adjacents dans G. Les sommets de la même partie seront coloriés avec la même couleur.
Sortie : une coloration de G avec un nombre minimum de couleurs.

Bin Packing
Entrée : n objets avec les poids $w_1, \ldots, w_n \in \mathbb{N}$ et un entier b représentant la capacité de chaque boîte.
Sortie : une partition de ces n objets dans un nombre minimum d'ensembles telle que la somme des poids des objets d'un ensemble ne dépasse pas la capacité b.

Min Steiner Tree
Entrée : un graphe complet G, une distance sur les arêtes du graphe et un sous-ensemble de sommets S.
Un *Steiner tree* de G est un sous-arbre de G qui contient tous les sommets de S.
Sortie : un « Steiner tree » avec la somme des poids des arêtes minimum.

Min Bandwidth
Entrée : un graphe $G = (V, E)$.
Sortie : une fonction bijective $f : V \to \{1, \ldots, |V|\}$ telle que

$$\max_{\{u,v\} \in E} |f(u) - f(v)|$$

soit minimum.

Min Linear Arrangement
Entrée : un graphe $G = (V, E)$.
Sortie : une fonction bijective $f : V \to \{1, \ldots, |V|\}$ telle que la somme

$$\sum_{\{u,v\} \in E} |f(u) - f(v)|$$

soit minimum.

Min Label Cover

Entrée : un graphe régulier biparti $G = (V_1, V_2, E)$ (c'est-à-dire qu'il existe deux constantes d_1 et d_2 telles que chaque sommet de V_1 a pour degré d_1 et chaque sommet de V_2 a pour degré d_2), un entier N (qui définit l'ensemble des étiquettes : les entiers de l'ensemble $\{1, 2, \dots, N\}$) et pour chaque arête $e \in E$ une fonction partielle $\pi_e : \{1, 2, \dots, N\} \to \{1, 2, \dots, N\}$.

Un *étiquetage* associe à chaque sommet de G un ensemble non vide d'étiquettes. Un étiquetage couvre une arête $e = \{u, v\}$ où $u \in V_1$ et $v \in V_2$ si pour toute étiquette a_2 assignée à v il existe une étiquette a_1 assignée à u telle que $\pi_e(a_1) = a_2$.

Le *coût* d'un étiquetage est :

$$\sum_{v \in V_1} \text{le nombre d'étiquettes assignées à } v.$$

Sortie : un étiquetage qui couvre toutes les arêtes, utilisant éventuellement plus qu'une étiquette par sommet si nécessaire, tel que le rapport entre le coût d'étiquetage et $|V_1|$ soit minimum.

Bibliographie

[AK97] P. Alimonti, V. Kann, *Hardness of Approximating Problems on Cubic Graphs*, Proceedings of the 3rd Italian Conference on Algorithms and Complexity (1997), LNCS 1203, 288–298.

[AFWZ93] N. Alon, U. Feige, A. Wigderson, D. Zuckerman, *Derandomized graph products*, Manuscript, 1993.

[AYZ94] N. Alon, R. Yuster, U. Zwick, *Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs*, Proceedings of the 26th ACM Symposium on the Theory of Computing (1994), 326–335.

[ABSS93] S. Arora, L. Babai, J. Stern, Z. Sweedyk, *The hardness of approximate optima in lattices, codes and linear equations*, Proceedings of the 34th IEEE Symposium on Foundations of Computer Science (1993), 724–733.

[AFK96] S. Arora, A. Frieze, H. Kaplan, *A New Rounding Procedure for the Assignment Problem with Applications to Dense Graph Arrangements*, Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (1996), 21–30.

[AKK95] S. Arora, D. Karger, M. Karpinski, *Polynomial Time Approximation Schemes for Dense Instances of NP-hard Problems* , Proceedings of 27th ACM Symposium on the Theory of Computing (1995), 284–293.

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, *Proof verification and hardness of approximation problems*, Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science (1992), 14–23.

[AS92] S. Arora, S. Safra, *Probabilistic checking of proofs: A new characterisation of NP*, Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science (1992), 2–13.

[ACP94] G. Ausiello, P. Crescenzi, M. Protasi, *Approximate Solution of NP Optimization Problems*, Theoretical Computer Science 150 (1995), 1-55.

[ADP80] G. Ausiello, A. D'Atri, M. Protasi *Structure preserving reductions among convex optimization problems*, Journal of Computer and System Sciences 21 (1980), 136–153.

[B94] B. S. Baker, *Approximation Algorithms for NP-complete problems in planar graphs*, Journal of the ACM 41(1994), 153–180.

[BST98a] C. Bazgan, M. Santha, Zs. Tuza, *On the approximation of finding a(nother) Hamiltonian cycle in cubic Hamiltonian graphs*, Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (1998), LNCS 1373, 276–286, à paraître au Journal of Algorithms.

[BST98b] C. Bazgan, M. Santha, Zs. Tuza, *Efficient approximation algorithms for the* SUBSET-SUMS EQUALITY *problem* , Proceedings of the 25th International Colloquium on Automata, Languages, and Programming (1998), LNCS 1443, 387–396.

[BBFG91] R. Beigel, M. Bellare, J. Feigenbaum, S. Goldwasser, *Languages that are easier than their proofs*, Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science (1991), 19–28.

[BC76] A. Bondy, V. Chvátal, *A method in graph theory*, Discrete Mathematics 15 (1976), 111–135.

[Cam98] K. Cameron, *Krawczyk's graphs show Thomason's algorithm for finding a second Hamilton cycle through a given edge in a cubic graph is exponential* , Fifth Czech-Slovak International Symposium on Combinatorics, Graph Teory, Algorithms and Applications, Prague, July 1998.

[CT97] M. Cesati, L. Trevisan, *On the Efficiency of Polynomial Time Approximation Schemes*, Information Processing Letters, 64(4) (1997), 165–171.

[Chv72] V. Chvátal, *On Hamiltonian ideals*, Journal of Combinatorial Theory (1972), 163–168.

[CT96] A. Clementi, L. Trevisan, *Improved Nonapproximability Result for Vertex Cover with Density Constraints*, Proceedings of the 2nd International Conference on Computing and Combinatorics (1996), LNCS 1090, 333–342.

[Cres97] P. Crescenzi, *A Short Guide to Approximation Preserving Reductions*, Proceedings of the 12th Annual Structure in Complexity Theory Conference (1997) , 262–273.

[CK] P. Crescenzi, V. Kann,*A compendium of NP optimization problems*, `http://www.nada.kth.se/theory/problemlist.html`.

[CKST95] P. Crescenzi, V. Kann, R. Silvestri, L. Trevisan, *Structure in approximation classes*, Proceedings of the 1st International Conference on Computing and Combinatorics (1995), LNCS 959, 539–548.

[CP91] P. Crescenzi, A. Panconesi, *Completeness in approximation classes*, Information and Computation 93(1991), 241–262.

[DHK93] E. Dahlhaus, P. Hajnal, M. Karpinski, *On the Parallel Complexity of Hamiltonian Cycle and Matching Problem on Dense Graphs*, Journal of Algorithms 15(1993), 367–384.

[Dil50] R. P. Dilworth, *A decomposition theorem for partially ordered sets*, Annals of Mathematics (2) 51 (1950), 161–166.

[DF95] R. Downey, M. Fellows, "Parameterized computational feasibility", Feasible Mathematics II(Ed. P. Clote and J. Remmel), Birkhauser, Boston(1995), pages 219-244.

[F96] W. Fernandez de la Vega, *Max-Cut has a Randomized Approximations Scheme in Dense Graphs*, Random Structures and Algorithms, 8(3) (1996), 187–198.

[FK97] W. Fernandez de la Vega, M. Karpinski, *Polynomial Time Approximability of Dense Weighted Instances of* MAX CUT , Research report No. 85171-CS,University of Bonn (1997).

[FK98] W. Fernandez de la Vega, M. Karpinski, *On Approximation Hardness of Dense TSP and Other Path Problems*, ECCC, report no.24(1998).

[FS98] D. A. Fotakis, P. G. Spirakis, *Graph properties that faciliate travalling.*, ECCC, report no.31(1998).

[FJ95] A. Frieze, M. Jerrum, *Improved Approximation Algorithms for* MAX kCUT *and* MAX BISECTION , Proceedings of the 4th International Conference on Integer Programming and Combinatorial Optimization (1995), 1–13.

[FK96] A. Frieze, R. Kannan, *The Regularity Lemma and Approximation Schemes for Dense Problems* , Proceedings of the 37rd IEEE Symposium on Foundations of Computer Science (1996), 12–20.

[FR94] M. Fürer, B. Raghavachari, *Approximating the Minimum-Degree Steiner Tree to within One of Optimal*, Journal of Algorithms 17 (1994), 409–423.

[GMM97] G. Galbiati, A. Morzenti, F. Maffioli, *On the Approximability of some Maximum Spanning Tree Problems*, Theoretical Computer Science 181(1) (1997), 107–118.

[GJ76a] M. R. Garey, D. S. Johnson, *Strong NP-completeness results: motivation, examples, and implications*, Journal of the ACM 25 (1978), 499–508.

[GJ76b] M. R. Garey, D. S. Johnson, *The complexity of near-optimal graph coloring*, Journal of the ACM 23 (1976), 43–49.

[GJT76] M. R. Garey, D. S. Johnson, R. E. Tarjan, *The planar Hamiltonian circuit problem is NP-complete*, SIAM Journal on Computing 5:4 (1976), 704–714.

[GW93] M. X. Goemans, D. P. Williamson, *New 3/4-approximation algorithms for* MAX SAT, SIAM Journal on Discrete Mathematics 7 (1994), 656–666.

[GW95] M. X. Goemans, D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, Journal of ACM 42 (1995), 1115–1145.

[GKP95] M. Grigni, E. Koutsoupias, C. Papadimitriou, *An Approximation Scheme for Planar Graph* TSP, Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (1995),640–645.

[GLS81] M. Grötschel, L. Lovász, A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica 1(1981), 168–198.

[H96] J. Håstad, *Clique is hard to approximate within* $n^{1-\varepsilon}$, Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (1996),627–636.

[HNOS95] E. Hemaspaandra, A. V. Naik, M. Ogihara, A. Selman, *P-Selective Sets and Reducing Search to Decision vs Self-Reducibility*, Journal of Computer and System Sciences, 53(2) (1996), 194–209.

[Hoch95] D. Hochbaum, *Approximation Algorithms for NP-hard Problems*, PWS Publishing Company, 1995.

[Hol81] I. Holyer, *The NP-completeness of edge-coloring*, SIAM Journal on Computing 10(1981), 718–720.

[Hop82] J. J. Hopfield, *Neural Networks and Physical Systems with Emergent Collective Computational Capabilities*, Proceedings of the National Academy of Sciences of the U.S.A. 79(1982), 2554–2558.

[HS76] E. Horowitz, S. Sahni, *Exact and approximate algorithms for scheduling nonidentical processors*, Journal of the ACM, 23 (1976), 317–327.

[IK75] O. H. Ibarra, C. E. Kim, *Fast approximation algorithms for the Knapsack and Sum of Subset problems*, Journal of the ACM, 22:4 (1975), 463–468.

[J74] D. S. Johnson, *Approximation algorithms for combinatorial problems*, Journal of Computer and System Sciences, 9 (1974), 256–278.

[JPY88] D. S. Johnson, C. Papadimitriou, M. Yannakakis, *How easy is local search?*, Journal of Computer and System Sciences, 37 (1988), 79–100.

[KMR93] D. Karger, R. Motwani, G. Ramkumar, *On Approximating the Longest Path in a Graph*, Proceedings of the 3rd Workshop on Algorithms and Data Structures, LNCS 709 (1993), 421–432. Also appeared in Algoritmica 18 (1997), 82–98.

[KMS94] D. Karger, R. Motwani, M. Sudan, *Approximate Graph Coloring by Semidefinite Programming*, Proceedings of the 35 th IEEE Symposium on Foundations of Computer Science (1994), 2–13.

[KK82] N. Karmaker, R. M. Karp, *An efficient approximation scheme for the one-dimensional bin-packing problem*, Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (1982), 312–320.

[K72] R. M. Karp, *Reducibility among combinatorial problems*, in: Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, eds.) (1972), 85–103.

[K97] M. Karpinski, *Polynomial Time Approximation Schemes for Some Dense Instances of NP-Hard Optimization Problems*, ECCC Technical Report TR 97-024, 1997.

[KZ96] M. Karpinski, A. Zelikovsky, *Approximating Dense Cases of Covering Problems*, ECCC Technical Report TR 97-004, 1997; the preliminary version appeared as Technical Report TR 96-059, International Computer Science Institute, Berkeley, 1996.

[KMPS97] H. Kellerer, R. Mansini, U. Pferschy, M. G. Speranza, *An efficient fully polynomial approximation scheme for the Subset-Sum problem*, manuscript, 1997.

[KM96] S. Khanna, R. Motwani, *Towards a Syntactic Characterization of PTAS*, Proceedings of 28th ACM Symposium on the Theory of Computing (1996), 329–337.

[KMSV94] S. Khanna, R. Motwani, M. Sudan, U. Vazirani, *On Syntactic versus Computational Views of Approximability*, Proceedings of the 35th IEEE Symposium on Foundations of Computer Science (1994), 819–830.

[KSW97] S. Khanna, M. Sudan, D. Williamson, *A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction*, Proceedings of the 29th Annual ACM Symposium on Theory of Computing (1997), 11–20.

[KST93] J. Köbler, U. Schöning, J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäser, 1993.

[KT90] P. G. Kolaitis, M. N. Thakur, *Logical definability of NP optimization problems*, Technical Report CRL 90-48, University of California, Santa Cruz, 1990.

[KT91] P. G. Kolaitis, M. N. Thakur, *Approximation properties of NP minimization classes*, Proceedings of the 6th Annual Structure in Complexity Theory Conference, 1991, 353–366.

[Kraw95] A. Krawczyk, *A note on the complexity of finding a second Hamiltonian cycle in cubic graphs*, preprint, December 1995.

[K89] M. Krentel, *Structure in Locally Optimal Solutions*, Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (1989), 216–221.

[L75] R. E. Ladner, *On the structure of polynomial time reducibility*, Journal of the ACM 22(1975), 155–171.

[LR88] T. Leighton, S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (1988), 422–431.

[Lov79] L. Lovász, *Combinatorial Problems an Exercises*, North-Holland Publishing Company, Amsterdam, 1979.

[LY94] C. Lund, M. Yannakakis, *On the hardness of approximating minimization problems*, Journal of the ACM 41(1994), 960–981.

[MP91] N. Megiddo, C. Papadimitriou, *On total functions, existence theorems and computational complexity*, Theoretical Computer Science, 81 (1991), 317–324.

[Mon84] B. Monien, *How to find long paths efficiently*, Annals of Discrete Mathematics, 25 (1984), 239–254.

[PR90] A. Panconesi, D. Ranjan, *Quantifiers and approximation*, Proceedings of 22th ACM Symposium on the Theory of Computing (1990), 446–456; appeared also in Theoretical Computer Science 107(1993), 145–163.

[Pap94a] C. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Company, 1994.

[Pap94b] C. Papadimitriou, *On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence*, Journal of Computer and System Sciences 48 (1994), 498–532.

[PSY90] C. Papadimitriou, A. Schäffer, M. Yannakakis, *On the Complexity of Local Search*, Proceedings of 22nd ACM Symposium on the Theory of Computing (1990), 438–445.

[PS82] C. Papadimitriou, K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[PY91] C. Papadimitriou, M. Yannakakis, *Optimization, Approximation and Complexity Classes*, Journal of Computer and System Sciences 43 (1991), 425–440.

[PY93] C. Papadimitriou, M. Yannakakis, *The traveling salesman problem with distances one and two*, Mathematics of Operations Research 18(1) (1993), 1–11.

[PM81] A. Paz, S. Moran, *The deterministic polynomial optimization problems and their approximation*, Theoretical Computer Science 15 (1981), 251–277.

[RT87] P. Raghavan, C. Thompson *Randomized Rounding: a technique for provably good algorithms and algorithmic proofs*, Combinatorica 7(1987), 365–374.

[S76] S. Sahni, *Algorithms for scheduling independent tasks*, Journal of the ACM, 23 (1976), 116–127.

[S75] S. Sahni, *Approximate Algorithms for the 0/1 Knapsack Problem*, Journal of the ACM, 22 (1975), 115–124.

[SY91] A. Schäffer, M. Yannakakis, *Simple Local Search Problems that are hard to solve*, SIAM Journal on Computing (1991), 56–87.

[Sel88a] A. Selman, *Natural self-reducible sets*, SIAM Journal on Computing, 17(1988), 989–996.

[Sel88b] A. Selman, *Promise Problems Complete for Complexity Classes*, Information and Computation, 78(1988), 87–98.

[Tho78] A. Thomason, *Hamilton cycles and uniquely edge colourable graphs*, Annals of Discrete Mathematics 3 (1978), 259–268.

[T97] L. Trevisan, *Approximating Satisfiable Satisfiability Problems*, Proceedings of the 5th European Symposium on Algorithms (1997), LNCS 1284, 472–485.

[T46] W. T. Tutte, *On Hamiltonian circuits*, The Journal of the London Mathematical Society 21 (1946), 98–101.

[VV86] L. G. Valiant, V. V. Vazirani, *NP is as easy as detecting unique solutions*, Theoretical Computer Science 47 (1986), 85–93.

[Yan96] M. Yannakakis, personal communication, 1996.

[W78] S. Win, *On the Connection Between the Existence of k-Trees and the Toughness of a Graph*, Graphs and Combinatorics 5 (1989), 201–205.

[WY92] G. J. Woeginger, Z. Yu, *On the equal-subset-sum problem*, Information Processing Letters 42 (1992), 299–302.

[Z95] M. Zimand, *Weighted NP Optimization Problems: Logical Definability and Approximation Properties*, Proceedings of the 10th Annual Structure in Complexity Theory Conference, 1995, 12–28.

Index