Master MPRO

Cours CAP

Approximabilité des problèmes d'optimisation

Dimitri Watel

19 janvier 2025

Introduction

Ce polycopié contient l'ensemble des exemples et preuves présentés en cours d'approximabilité de CAP. Chaque preuve est écrite au propre dans un style propre à l'auteur que celui-ci apprécie particulièrement. Il vous sera demandé d'avoir à l'examen au moins la même rigueur que celle présentée ici sans nécessairement avoir le même style.

On s'intéresse ici à résoudre des problèmes d'optimisations, pour lesquels, connaissant une entrée, on recherche à maximiser ou minimiser une fonction objectif. Usuellement, quand on a besoin de résoudre un tel problème, ce problème est NP-Complet. Il peut s'agir de problèmes de tournée, avec le voyageur de commerce, de problème de logistique avec le sac à dos, de réseaux avec le problème de Steiner, ... Si posséder une solution optimale est intéressant, perdre du temps à la calculer est souvent plus pénalisant que ne pas posséder une solution optimale. On peut donc avoir un intérêt à chercher des solutions sous-optimales si ce calcul est rapide.

Une approximation est, dans ce cours, un algorithme heuristique avec *garantie de performance* qui se termine *rapidement*. Ils sont rarement (si ce n'est jamais) exacts mais renvoient toujours une solution *proche* d'une solution optimale. Le caractère proche sera défini formellement ultérieurement mais, basiquement, on entend par proche une solution qui a une valeur objective proche de celle de la solution optimale. Plus la solution est proche, meilleure est l'approximation. On cherche donc des solutions quasi-optimales. Le terme *rapidement* est à prendre au sens complexité du terme, c'est-à-dire qu'on cherchera généralement des algorithmes polynomiaux.

Ce cours n'a, bien entendu, pas pour vocation d'être exhaustif sur le sujet. L'objectif premier de ce document est surtout de vous fournir un support avec lequel réviser. Vous seront présentés au cours des différents chapitres et séances les points suivants. Dans une première partie, nous introduirons le sujet de l'approximabilité des problèmes d'optimisation par une série d'exemples. Le second chapitre formalisera toutes les définitions introduites dans le premier chapitre, présentera les différentes classes d'approximabilité de problèmes d'optimisation et les propriétés que l'on peut tirer de ces classes. Le troisième chapitre s'intéressera à une manière de produire des algorithmes d'approximation à l'aide de la modélisation des problèmes en programmes linéaires. Enfin la dernière partie s'intéressera à l'inapproximabilité, c'est-à-dire comment prouver qu'il n'existe pas d'approximation de qualité pour un problème.

Ce cours est issu, en plus des connaissances personnelles de l'auteur, de deux livres et d'une thèse :

- *Complexity and approximation, Combinatorial optimization problems and their approximability properties* de G. Ausiello, P. Crescenzi, G.Gambosi, V. Khan, A. Marchetti-Spaccamela et M.Protasi, édition Springer-Verlag Berlin Heidelberg, 1999. **ISBN : 3-540-65431-3**
- *Approximation algorithms* de V. V. Vazirani, édition Springer-Verlag Berlin Heidelberg, 2003. **ISBN : 978-3-642-08469-0**
- *Approximation polynomiale de problèmes d'optimisation : aspects structurels et opérationnels* de B. Escoffier, 2005, `http://www.theses.fr/2005PA090065`.

Si le sujet vous intéresse, une version traduite en français de second livre existe et la thèse est également en français. Un autre lien utile est le compendium des problèmes d'optimisations. Il contient des résultats d'approximabilités pour de nombreux problèmes d'optimisation : `https://www.csc.kth.se/tcs/compendium/`.

Chapitre 1

Exemples d'approximations polynomiales

Ce chapitre illustre divers algorithmes d'approximation polynomiaux au travers de 7 problèmes classiques d'optimisation. Ce chapitre est une bonne introduction au chapitre suivant qui introduira les classes d'approximabilité.

1.1 Le problème de l'arbre de Steiner

Le problème de l'arbre de Steiner est le suivant.

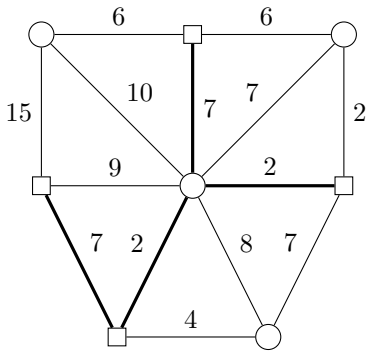
Problème : Problème de l'arbre de Steiner (Undirected Steiner Tree en anglais)

Instance :
- Un graphe non orienté $G = (V, E)$ contenant n nœuds et m arêtes.
- Un ensemble de k nœuds $X \subset V$ appelés *terminaux*.
- Une fonction de poids $\omega : E \to \mathbb{N}$.

Solution réalisable : Un sous-arbre T de G couvrant chaque terminal de X.

Optimisation : Minimiser le poids $\omega(T) = \sum_{e \in T} \omega(e)$.

La figure suivante vous donne un exemple d'instance avec une solution optimale. Les terminaux sont les nœuds carrés et les poids sont indiqués à côté de chaque arête. La solution optimale est indiquée en gras.

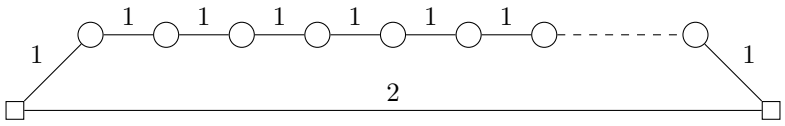


Ce problème est NP-Difficile, il fait parti des 21 problèmes originaux pour lesquels Karp a prouvé la complexité.

On peut remarquer que ce problème possède toutefois 2 problèmes polynomiaux proches : le problème de l'arbre couvrant de poids minimum et le problème du plus court chemin. Le premier apparaît quand $k = n$. Le second quand $k = 2$. Dans ces deux cas le problème devient polynomial [1]. Le problème de l'arbre couvrant peut être facilement résolu avec l'algorithme de Prim ou l'algorithme de Kruskal. Le second peut être résolu avec l'algorithme de Dijkstra. On peut tenter de se servir de ces résultats pour produire un algorithme heuristique pour résoudre le problème de Steiner.

1. Il se trouve que lorsqu'on fixe le paramètre k ou lorsqu'on fixe $n - k$, le problème de Steiner devient polynomial. Mais vous reviendrez sur ces notions dans l'autre module du cours CAP, concernant la complexité paramétrée.

Une première idée serait de chercher un arbre couvrant et de supprimer les arêtes superflues. Mais comme le montre l'exemple suivant, ce n'est pas une bonne idée :



Un arbre couvrant sélectionnerait alors toutes les arêtes de poids 1 et non l'arête de poids 2 qui est pourtant la solution optimale. Il faudrait restreindre l'arbre couvrant pour éviter qu'il ne sélectionne des arêtes qui ne sont trivialement pas dans la solution optimale.

Pour palier à ce défaut, on pourrait relier tous les nœuds avec un plus court chemin. Seul problème, on a beaucoup trop de plus courts chemins. On en a $k * (k-1)/2$. Que faire avec ? Lesquels faut-il garder ? C'est ici que l'arbre couvrant va nous aider puisqu'il va nous aider à sélectionner les *meilleurs plus courts chemins*. Dans les années 70-80, des chercheurs ont réutilisé cette idée pour produire un algorithme d'approximation. Pour l'anecdote, ceci s'est produit au moins 4 fois par 4 équipes indépendantes (internet n'existait pas, difficile de communiquer ses résultats efficacement).

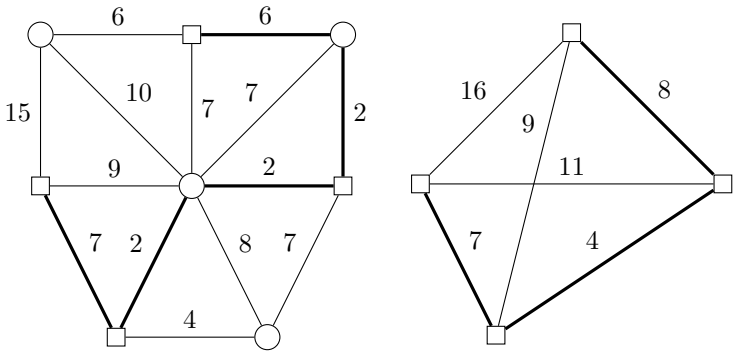
L'algorithme est le suivant :

Algorithme 1 Choukkmane (1978) et de Kou et al. (1981), PLesnìk (1981) et Iwainsky et al. (1986)

1: Créer un graphe H complet contenant chaque nœud de X
2: **Pour** $x, x' \in X$ **Faire**
3: Calculer un plus court chemin $p(x, x')$ dans G de poids $d(x, x')$
4: Pondérer l'arête (x, x') de H avec $d(x, x')$
5: Calculer un arbre couvrant T_H de H de poids minimum
6: Calculer $T = \bigcup_{(x,x') \in T_H} p(x, x')$
7: **Tant que** T possède un cycle **Faire**
8: Supprimer l'arête de ce cycle de poids maximum
9: **Tant que** T possède une feuille qui n'est pas dans X **Faire**
10: Supprimer cette feuille de T
11: **Renvoyer** T

Puisque les calculs de plus court chemin et d'arbre couvrant de poids minimum se font en temps polynomial, cet algorithme est polynomial.

Appliquons cet algorithme sur notre premier exemple.



G et T H et T_H

On observe sur cette figure que l'algorithme a échoué à remarquer qu'il était plus intéressant de passer par le centre pour couvrir le nœud du haut que le chemin de droite. Toutefois le résultat n'est pas mauvais puisqu'il diffère de la solution optimale d'une valeur de 1. On va chercher à montrer que T est toujours au pire deux fois moins bon que la solution optimale.

Les lignes 7 à 10 ne permettent que de nettoyer T pour en faire un arbre sans arête superflue. Chaque opération supprime des arêtes et diminue le poids de T. Au mieux le poids diminue grandement, au pire, il ne change pas. Dans la suite, on s'intéressera surtout à démontrer des propriétés sur le poids du résultat de la ligne 6. Si ces propriétés sont intéressantes, elles le seront encore plus après nettoyage.

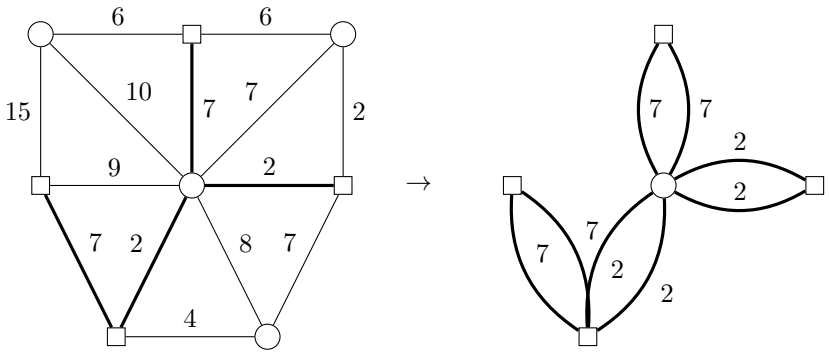
Théorème 1.1.1. *Soit T^* une solution optimale de (G, X, ω) alors $\omega(T) \leq 2\omega(T^*)$.*

On dit alors qu'il s'agit d'une *approximation polynomiale de rapport 2* ou une *2-approximation.*

Avant de prouver ce théorème, on va chercher des résultats intermédiaires. En particulier, on va rechercher une borne inférieure de $\omega(T^*)$. C'est une technique classique en approximabilité : on cherche une valeur $b \leq 2\omega(T^*)$ et on va montrer que $\omega(T) \leq b$. Cette technique de borne reviendra régulièrement mais n'est pas systématique. C'est ici un moyen simple de faire le lien entre T et T^*.

Une chose est sûre, c'est qu'on sait peu de choses de T^* : c'est un arbre, il couvre X et il le fait de la manière la moins couteuse possible. Mais on a une propriété intéressante de T qu'il va falloir exploiter : T est la meilleure manière de couvrir X dans H. Autrement dit, si on transpose T^* dans H alors on aura quelque chose de plus cher que T. C'est ici que notre borne b apparait : $\omega(T)$ est plus petit que le poids b de la transposition de T^* dans H.

Comment transposer T^* dans H ? Après tout, H est constitué des plus courts chemins entre les terminaux. Et rien n'indique que T^* contient ces plus courts chemins (une preuve de ceci est notre exemple). C'est ici que notre facteur 2 va apparaître. Supposons qu'on double toutes les arêtes de T^* et qu'on supprime tous les nœuds qui ne sont pas dans T^*. On obtient alors un graphe eulérien (puisque tous les degrés sont pairs). On calcule un cycle eulérien C^* dans ce graphe.

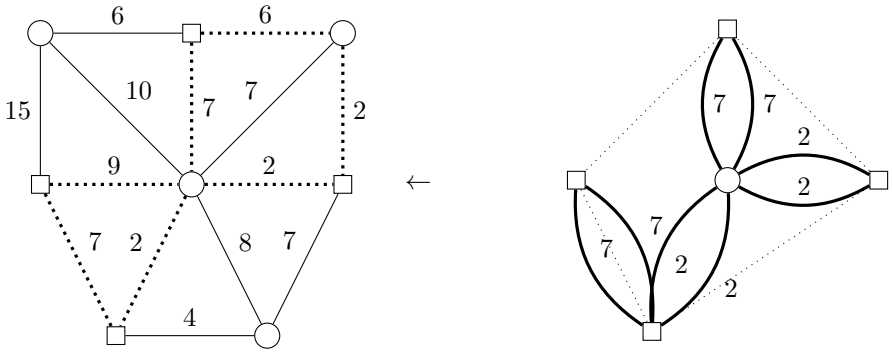


G et T^* C^*

Sans surprise, on vérifie le lemme suivant :

Lemme 1.1.1. $\omega(C^*) = 2\omega(T^*)$

Soient $(x_1, x_2, \ldots, x_k, x_1)$ les terminaux de X numérotés dans l'ordre d'apparition dans C^*. On notera $x_1 = x_{k+1}$ pour simplifier les calculs. On calcule alors dans G le cycle C égal à la concaténation des chemins $p(x_i, x_{i+1})$ pour i de 1 à k dans cet ordre.

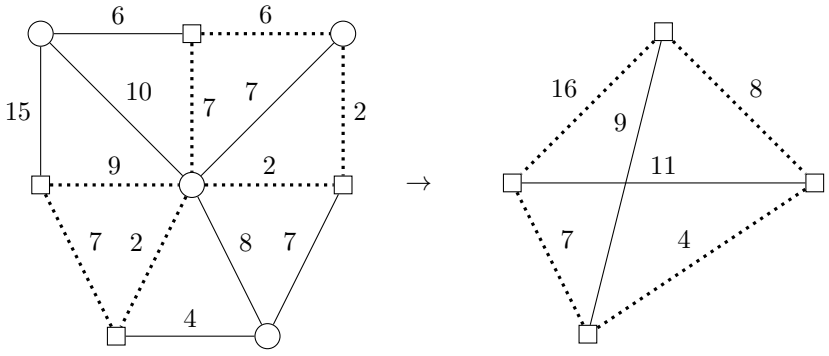


G et C C^* et $(x_1, x_2, \ldots, x_k, x_1)$ en pointillé.

Lemme 1.1.2. $\omega(C) \leq \omega(C^*)$

Démonstration. En effet, C^* est une union de chaînes P_i entre x_i et x_{i+1}, pour tout $i \leq k$ et, puisque C^* est eulérien, toutes ces chaînes sont arêtes-disjointes. Chacune des chaînes P_i est nécessairement de poids plus grand que $d(x_i, x_{i+1})$, par définition du plus court chemin. Donc le poids de C, égal à la somme des poids des chaînes $p(x_i, x_{i+1})$ est plus petit que le poids de C^*. □

On peut maintenant transposer C dans H en un cycle $C_H = (x_1, x_2, \ldots, x_k, x_1)$.



G et C H et C_H

Par définition des poids des arêtes de H, on vérifie le lemme suivant :

Lemme 1.1.3. $\sum_{e \in C_H} d(e) = \omega(C)$

Enfin, on construit un arbre T'_H égal au cycle C_H moins une arête (choisie arbitrairement). Trivialement :

Lemme 1.1.4. $\sum_{e \in T'_H} d(e) \leq \sum_{e \in C_H} d(e)$

Lemme 1.1.5. $\sum_{e \in T_H} d(e) \leq \sum_{e \in T'_H} d(e)$

Démonstration. T'_H est un arbre couvrant tous les nœuds de H. Il est donc de poids plus élevé que l'arbre couvrant de poids minimum T_H. □

Lemme 1.1.6. $\omega(T) \leq \sum_{e \in T_H} d(e)$

Démonstration.

$$\begin{aligned} \omega(T) &= \omega\left(\bigcup_{(x,x') \in T_H} p(x,x')\right) \\ &\leq \sum_{(x,x') \in T_H} \omega(p(x,x')) \\ &= \sum_{(x,x') \in T_H} d(x,x') \end{aligned}$$

□

L'ensemble de ces lemmes mis ensemble prouve le théorème 1.1.1. On peut faire légèrement mieux. On peut prouver le théorème suivant :

Théorème 1.1.2. *Soit T^* une solution optimale de (G, X, ω) alors $\omega(T) \leq 2 \cdot (1 - \frac{1}{k})\omega(T^*)$.*

Démonstration. On peut modifier le résultat du lemme 1.1.4 en $\sum_{e \in T'_H} d(e) \leq (1 - \frac{1}{k}) \cdot \sum_{e \in C_H} d(e)$. En effet, au lieu de produire T'_H en supprimant une arête arbitraire de C_H, ce qui fait gagner le plus consiste à supprimer l'arête de poids maximum.

Et on vérifie :

$$\max_{e \in C_H} d(e) \geq \frac{1}{k} \sum_{e \in C_H} d(e)$$

En effet, dans le cas contraire, pour tout arête $e \in C_H$, on aurait $d(e) \leq \max_{e \in C_H} d(e) < \frac{1}{k} \sum_{e \in C_H} d(e)$. Et donc $\sum_{e \in C_H} d(e) < |C_H| \cdot \frac{1}{k} \sum_{e \in C_H} d(e)$. Et puisque $|C_H| = k$ cette inégalité est fausse.

Donc $\sum_{e \in T'_H} d(e) = \sum_{e \in C_H} d(e) - \max_{e \in C_H} d(e) \leq (1 - \frac{1}{k}) \cdot \sum_{e \in C_H} d(e)$.

□

On a donc ici une approximation qui est meilleure quand k est petit.

En particulier, si $k = 2$, alors l'approximation est exacte. En effet, d'après le théorème précédent, on a $\omega(T) \leq \omega(T^*)$ et par optimalité de T^*, on vérifie l'inégalité opposée. Donc T est une solution optimale.

Par contre attention, ce théorème ne prouve pas par exemple, que si $k = n$ alors l'algorithme n'est pas optimal. En particulier, il se trouve que l'algorithme est optimal quand $k = n$, mais que notre approche ne permet pas de le prouver.

1.2 Le problème du voyageur de commerce

Le problème du voyageur de commerce est le suivant :

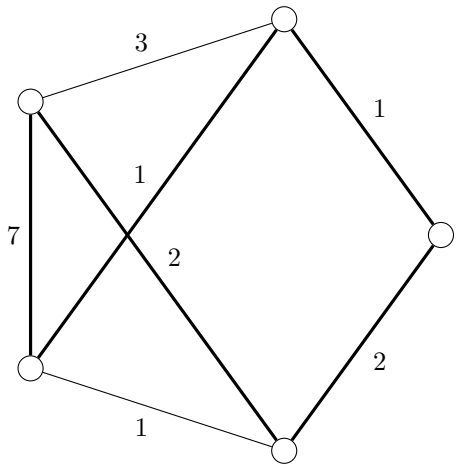
Problème : Problème du voyageur de commerce (ou Traveling Salesman problem en anglais)

Instance :
- Un graphe non orienté $G = (V, E)$ contenant n nœuds et m arêtes.
- Une fonction de poids $\omega : E \to \mathbb{R}^+$.

Solution réalisable : Un cycle hamiltonien C de G.

Optimisation : Minimiser le poids $\omega(C) = \sum_{e \in C} \omega(e)$.

Comme pour le problème de Steiner, commençons par un exemple.



Le voyageur de commerce est aussi un des 21 problèmes démontré NP-Difficiles par Karp. Mais il existe pour nous un problème plus grave : les solutions réalisables sont des cycles hamiltoniens de G. Et savoir si un graphe est hamiltonien est un problème NP-Complet.

Théorème 1.2.1. *Si $P \neq NP$, pour tout $r \in \mathbb{N}$, il n'existe pas d'algorithme d'approximation polynomial qui connaissant une instance (G, ω) du voyageur de commerce renvoie un cycle hamiltonien C de poids inférieur à r fois le poids d'une solution optimale.*

Démonstration. Supposons qu'il existe un tel algorithme $\mathcal{A}$.

Posons nous la question de savoir que renvoie $\mathcal{A}$ quand G n'est pas hamiltonien. $\mathcal{A}$ ne peut pas nous renvoyer un cycle hamiltonien puisque celui-ci n'existe pas. Et il ne peut pas non plus ne rien renvoyer. En effet, sinon il suffirait d'utiliser $\mathcal{A}$ et de regarder s'il renvoie ou non un cycle hamiltonien. Et selon la réponse, on pourrait déduire si G est hamiltonien ou non. Et on aurait donc un algorithme polynomial pour résoudre un problème NP-Complet. On peut donc en déduire que, si $P \neq NP$, l'algorithme $\mathcal{A}$ n'existe pas. □

Il faut bien aussi remarquer que si $P = NP$, la question d'approcher des problèmes d'optimisation devient un peu obsolète en pratique. On pourrait toujours tenter d'approcher des problèmes plus difficiles, des problèmes PSPACE-Difficiles ou EXPTIME-Difficiles, mais généralement l'industrie travaille avec des problèmes NP-Difficiles. Et donc la plupart des problèmes de l'industrie pourraient être résolus en temps polynomial si $P = NP$.

Bref, on peut donc partir de l'hypothèse que $P \neq NP$. Dans ce cas, il n'est pas possible de définir un algorithme d'approximation polynomial pour le voyageur de commerce. Et l'origine du problème est de savoir

s'il existe une solution réalisable ou non. On a une claire différence avec le problème de Steiner. Dans ce dernier, il est facile de savoir s'il existe une solution réalisable (deux terminaux sont dans des composantes connexes différentes si et seulement s'il n'y a pas de solution). Il faudrait donc, pour le voyageur de commerce, se restreindre au cas où on est certain que le graphe est hamiltonien, par exemple, s'il est complet.

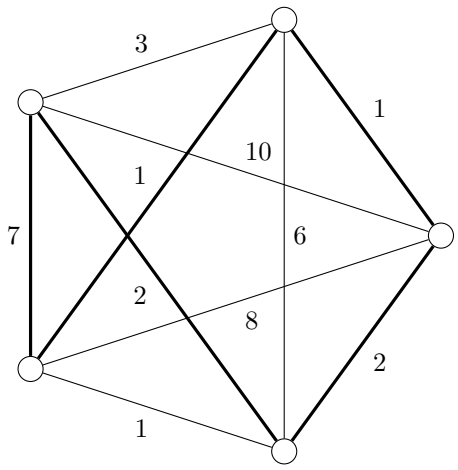
Problème : Problème du voyageur de commerce (graphe complet)

Instance :
- Un graphe complet non orienté $G = (V, E)$ contenant n nœuds et m arêtes.
- Une fonction de poids $\omega : E \to \mathbb{R}^+$.

Solution réalisable : Un cycle hamiltonien C de G.

Optimisation : Minimiser le poids $\omega(C) = \sum_{e \in C} \omega(e)$.

On se retrouve donc avec un pentagramme.



Un premier algorithme

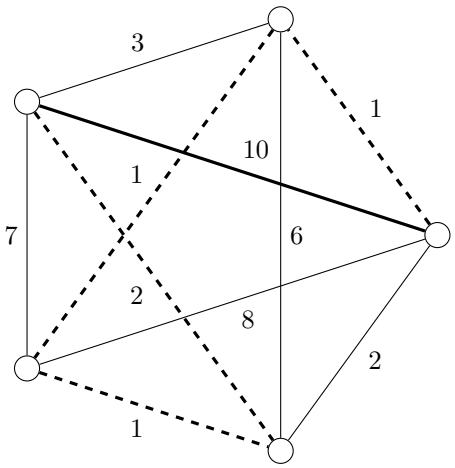
Commençons par un constat simple, issu de la preuve de l'approximation de rapport 2 pour le problème de Steiner. Si j'ai une solution optimale C^* et que je retire une arête, j'ai alors un arbre couvrant tous les sommets de G. On peut donc peut-être déduire d'un arbre couvrant de poids minimum une bonne solution.

Algorithme 2 Christofides (version simplifiée) (1976)

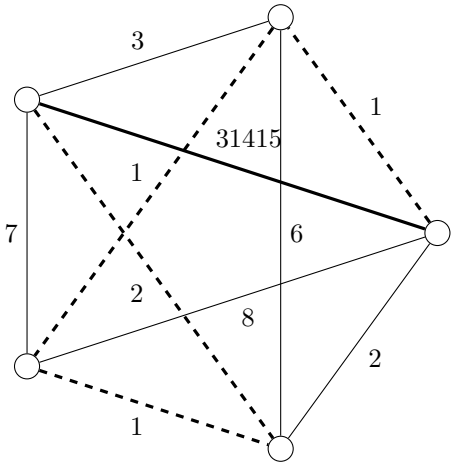
1: $T \leftarrow$ un arbre couvrant de poids minimum de G
2: Ordonner les nœuds de T selon un parcours en profondeur
3: **Renvoyer** le cycle C visitant les nœuds dans cet ordre

Le calcul de l'arbre couvrant se faisant en temps polynomial, cet algorithme est polynomial.

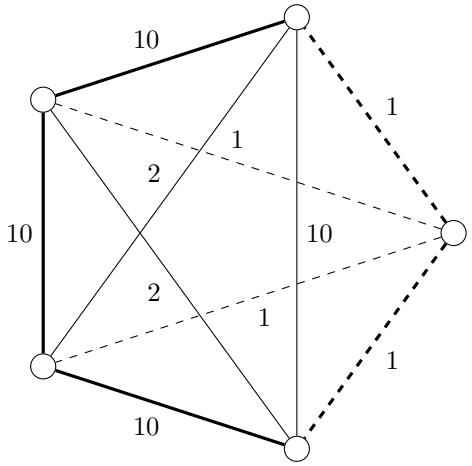
Sur notre exemple, cela donnerait la chose suivante, avec T correspondant aux arêtes en pointillé et C les arêtes en gras (ici T est inclus dans C).



Cela donne une bonne solution, de poids 15, alors que la solution optimale est de poids 12. Changeons un peu les poids :



Clairement l'algorithme a un problème. Autre exemple :



Il est difficile ici de décider quelles arêtes rajouter à l'arbre pour produire un cycle. Avec les arêtes en gras, on a une solution de valeur 32 alors que la solution optimale vaut 16. On verra dans un chapitre ultérieur qu'on ne pourra pas s'en sortir.

Notre soucis c'est qu'on prend une décision basée sur des arêtes de T qui ne mesure pas les arêtes qu'il faudra utiliser pour obtenir C. Il faudrait que les poids vérifient une propriété du type *Si j'ai un arbre couvrant qui relie u vers v de manière peu coûteuse, alors l'arête qui relie u à v n'est pas trop coûteuse.* Dans les deux exemples ci-dessus, cette propriété n'est pas vérifiée. Par exemple, les deux nœuds reliés par l'arête de poids 31415 sont reliés dans T par une chaîne de poids 5. On va donc encore restreindre le problème pour satisfaire cette propriété.

Problème : Problème du voyageur de commerce (version métrique)

Instance :
- Un graphe complet non orienté $G = (V, E)$ contenant n nœuds et m arêtes.
- Une fonction de poids $\omega : E \to \mathbb{R}^+$ qui vérifie l'inégalité triangulaire : pour tous nœuds u, v, w, $\omega(u, v) + \omega(v, w) \geq \omega(u, w)$.

Solution réalisable : Un cycle hamiltonien C de G.

Optimisation : Minimiser le poids $\omega(C) = \sum_{e \in C} \omega(e)$.

On peut alors montrer un rapport d'approximation de 2. Comme pour le problème de Steiner, on peut utiliser une technique de type borne inférieure : le poids d'une solution optimale est plus grande que le poids d'un arbre couvrant de poids minimum, car le cycle hamiltonien couvre tous les nœuds. Il suffit ensuite de montrer que le cycle C a un poids au plus deux fois plus grand que celui de l'arbre couvrant.

Théorème 1.2.2. *Si (G, ω) est une instance du problème de voyageur de commerce métrique, alors l'algorithme de Christofides est une approximation polynomiale de rapport 2.*

Démonstration. La preuve est similaire à celle utilisée pour le problème de Steiner. Soit C^* une solution optimale, on construit un arbre couvrant T^* en supprimant une arête de C^*. Cet arbre couvrant est de poids plus grand que celui de T qui est un arbre couvrant de poids minimum. On a donc $\omega(T) \leq \omega(C^*)$.

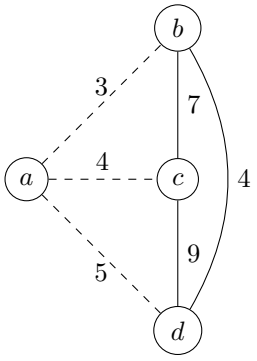
Ordonner les nœuds selon un parcours en profondeur de T revient, comme dans la preuve du problème de Steiner, à dédoubler toutes les arêtes de T et à calculer ensuite un cycle eulérien C' avec le résultat. On vérifie $\omega(C') = 2\omega(T)$ puisque C' passe exactement 2 fois par chaque arête de T. L'ordre des nœuds $(v_1, v_2, \ldots, v_n)$ est l'ordre d'apparition des nœuds dans C' (si un nœud apparaît plusieurs fois il n'est pas dupliqué dans cette liste). Remarque : pour être tout à fait rigoureux, il faudrait montrer que, quelque soit le parcours en profondeur de T, un tel cycle C' visitant les nœuds dans le même ordre existe, et inversement.

Enfin, on renvoie le cycle $(v_1, v_2, \ldots, v_n, v_1)$. Posons $v_{n+1} = v_1$ pour simplifier. Puisque ω vérifie l'inégalité triangulaire, pour tout $i \leq n$, le poids de l'arête (v_i, v_{i+1}) est plus petit que le poids de la chaîne reliant v_i à v_{i+1} dans C'. Donc $\omega(C) \leq \omega(C')$.

Donc $\omega(C) \leq 2\omega(C^*)$. □

Version évoluée de l'algorithme de Christofides

En réalité, l'algorithme de Christofides est plus élaboré et permet d'atteindre un rapport d'approximation de $\frac{3}{2}$. Le rapport 2 apparaît lorsqu'on effectue le parcours en profondeur de T, qui revient à doubler toutes les arêtes pour obtenir le cycle C'. Le fait de doubler est nécessaire pour être précis. En effet, on peut très bien tomber sur le cas de figure ci-dessous :



Notre arbre couvrant est en pointillé, dont on déduit le parcours en profondeur $C' = (a, b, a, c, a, d, a)$ et le cycle que l'on va déduire est donc $C = (a, b, c, d, a)$. Ici, aucun gain lorsqu'on passe de C' à C : le poids du parcours en profondeur est $2 * (3 + 4 + 5) = 24$ et le poids de C est $3 + 7 + 9 + 5 = 24$. C'est dû au fait que le poids de l'arête (b, c) est égal au poids de la chaîne reliant b et c dans l'arbre. De même pour c et d.

Peut-être que l'on s'est donc trompé de parcours en profondeur. Et si on avait choisi le parcours $C' = (a, c, a, b, a, d, a)$ à la place, alors on a un gain, toujours 24 pour C' mais 20 pour C.

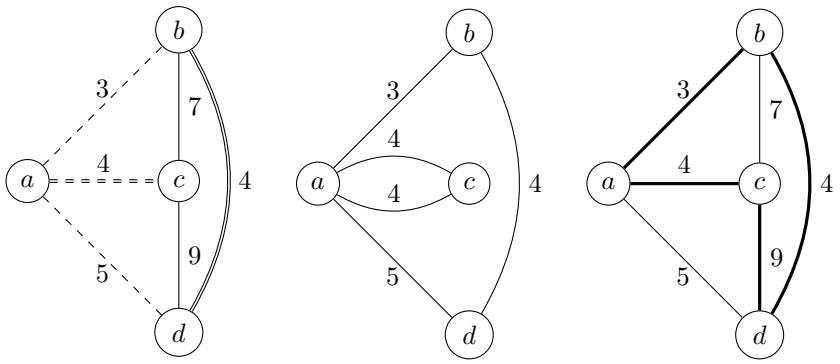
Sans connaissance préalable sur la manière dont est calculé ce parcours, on ne peut que supposer le pire cas. C'est ce que fait la version simplifiée avec son rapport 2. On peut toutefois orienter le choix du parcours en profondeur pour obtenir un meilleur rapport.

Algorithme 3 Christofides (1976)

1: $T \leftarrow$ un arbre couvrant de poids minimum de G
2: $V' \leftarrow$ les nœuds de degré impair de T
3: $M \leftarrow$ un couplage parfait de V' de poids minimum
4: $TM \leftarrow$ le graphe T où toutes les arêtes de M ont été ajoutées (si une arête est dans T et M, elle est doublée)
5: $C' \leftarrow$ un cycle eulérien de TM
6: **Renvoyer** le cycle C visitant les nœuds de G par ordre d'apparition dans C'

Remarque 1. Il existe toujours un couplage parfait de V' car il existe un nombre pair de nœuds de degré impair dans un graphe.

Dans notre exemple, on a, ci-dessous, T en pointillé et M doublé. On peut remarquer qu'on a toujours un choix à faire entre l'étape TM et le cycle C. On peut ici avoir (a, c, b, d, a) ou (a, c, d, b, a). En l'occurrence les deux choix mènent au même poids mais ce n'est pas une généralité, il est possible que l'un de ces deux choix soit meilleur que l'autre. Dans tous les cas, ce qui est important dans cette exemple, c'est qu'on emprunte toujours l'arête (b, d).



Théorème 1.2.3. *Si (G, ω) est une instance du problème de voyageur de commerce métrique, alors l'algorithme de Christofides est une approximation polynomiale de rapport $\frac{3}{2}$.*

Démonstration. Comme pour la version simplifié, on peut directement démontrer que $\omega(C) \leq \omega(C') \leq \omega(T) + \omega(M) \leq \omega(C^*) + \omega(M)$. Pour finir la démonstration, il suffit de prouver que $2\omega(M) \leq \omega(C^*)$ (c'est donc, encore une fois, une technique de type borne inférieure, sauf qu'on a ici 2 bornes inférieures).

Numérotons les nœuds de V' par ordre d'apparition dans $C^* : V' = (v_1, v_2, \ldots, v_{2k})$ et considérons maintenant le cycle $D^* = (v_1, v_2, \ldots, v_{2k}, v_1)$. Du fait de l'inégalité triangulaire, on vérifie $\omega(D^*) \leq \omega(C^*)$. Maintenant supprimons une arête sur deux, on obtient le couplage parfait $M_1^* = \{(v_1, v_2), (v_3, v_4), \ldots, (v_{2k-1}, v_{2k})\}$ ou le couplage parfait $M_2^* = \{(v_2, v_3), (v_4, v_5), \ldots, (v_{2k}, v_1)\}$. Puisque ces deux couplages sont disjoints, $\omega(M_1^*) + \omega(M_2^*) = \omega(D^*)$. Donc $\omega(M_1^*) \leq \frac{1}{2}\omega(D^*)$ ou $\omega(M_2^*) \leq \frac{1}{2}\omega(D^*)$.

De plus M est un couplage parfait de V' de poids minimum, donc $\omega(M) \leq \omega(M_1^*)$ et $\omega(M) \leq \omega(M_2^*)$. Donc $\omega(M) \leq \frac{1}{2}\omega(D^*) \leq \frac{1}{2}\omega(C^*)$. □

Encore une fois, cette preuve ne fonctionne que grâce à l'inégalité triangulaire et on peut démontrer que ce rapport est faux si l'inégalité triangulaire n'est pas vérifiée.

1.3 Le problème de couverture par ensembles

Le problème de couverture par ensembles est le suivant.

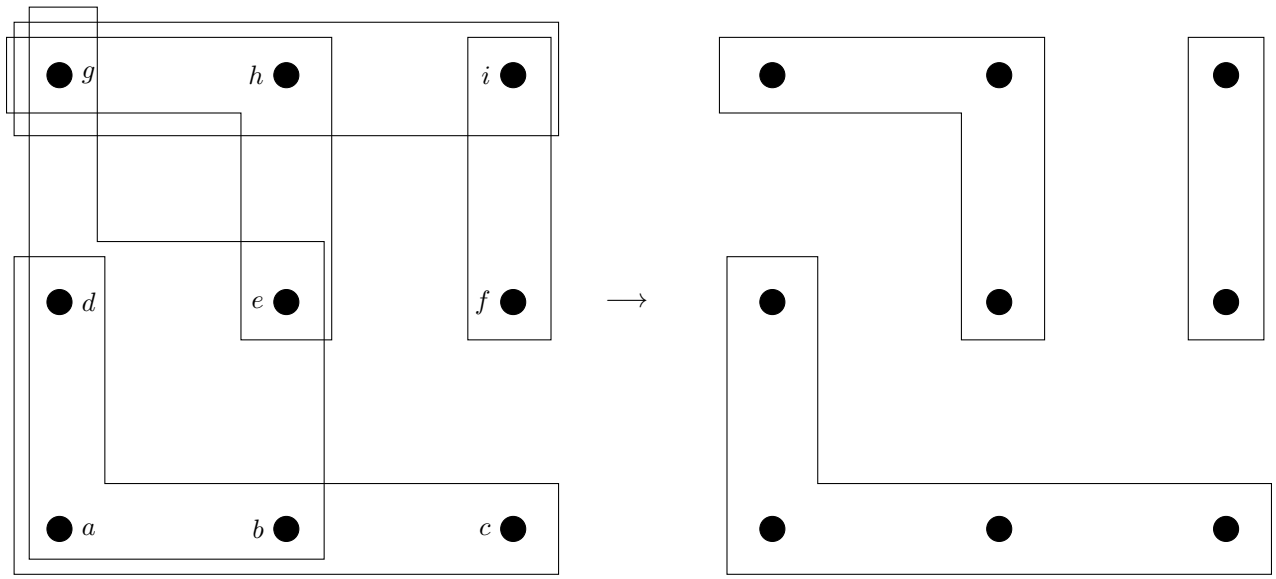
Problème : Problème de couverture par ensembles (Set Cover en anglais)

Instance :
- Un ensemble X contenant k éléments.
- Un ensemble $\mathcal{S} \subset 2^X$ de parties de X
- Une fonction de poids $\omega : S \to \mathbb{N}$

Solution réalisable : Un sous-ensemble C de $\mathcal{S}$ couvrant X, c'est-à-dire tel que, pour tout $x \in X$, il existe $S \in C$ tel que $x \in S$.

Optimisation : Minimiser le poids $\omega(C) = \sum_{S \in C} \omega(S)$.

La figure suivante vous donne un exemple d'instance avec une solution optimale. X est représenté par des ronds et S par les ensembles entourant les ronds. Il y a 5 ensembles : $\{\{a, b, c, d\}, \{a, b, d, e, g\}, \{e, g, h\}, \{f, i\}, \{g, h, i\}\}$. On suppose que tous les ensembles sont de poids 1. La solution optimale est à droite avec 3 ensembles.

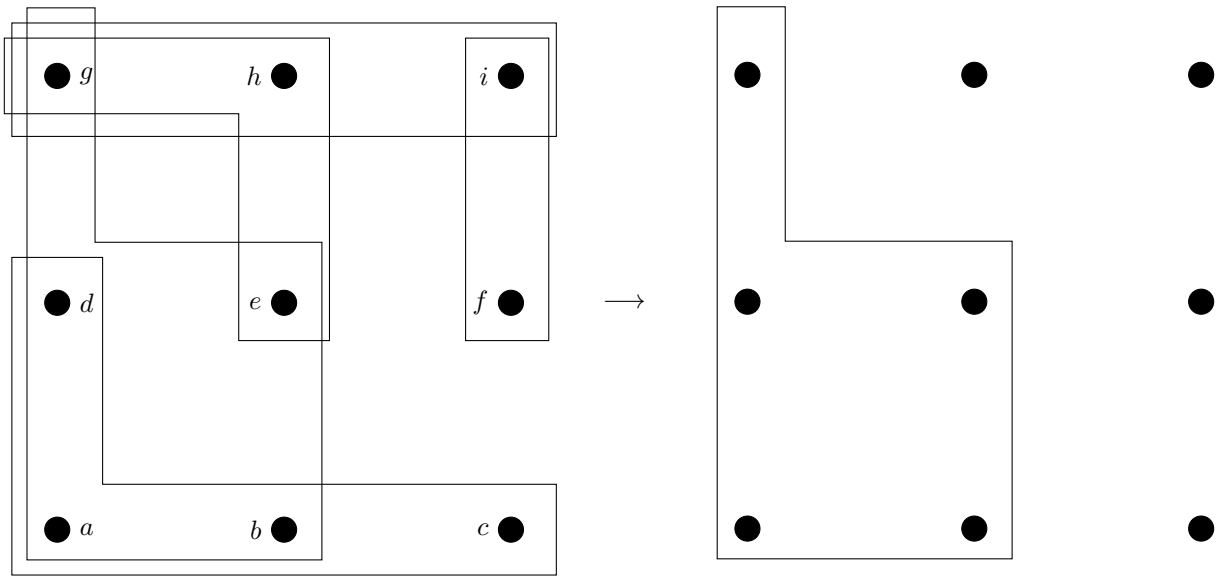


Quand tous les poids sont égaux à 1, il semble naturel de toujours prendre l'ensemble le plus gros en premier. Une fois cet ensemble pris, on le retire lui et tous ses éléments couverts et on recommence jusqu'à ce qu'il n'y ait plus d'élément.

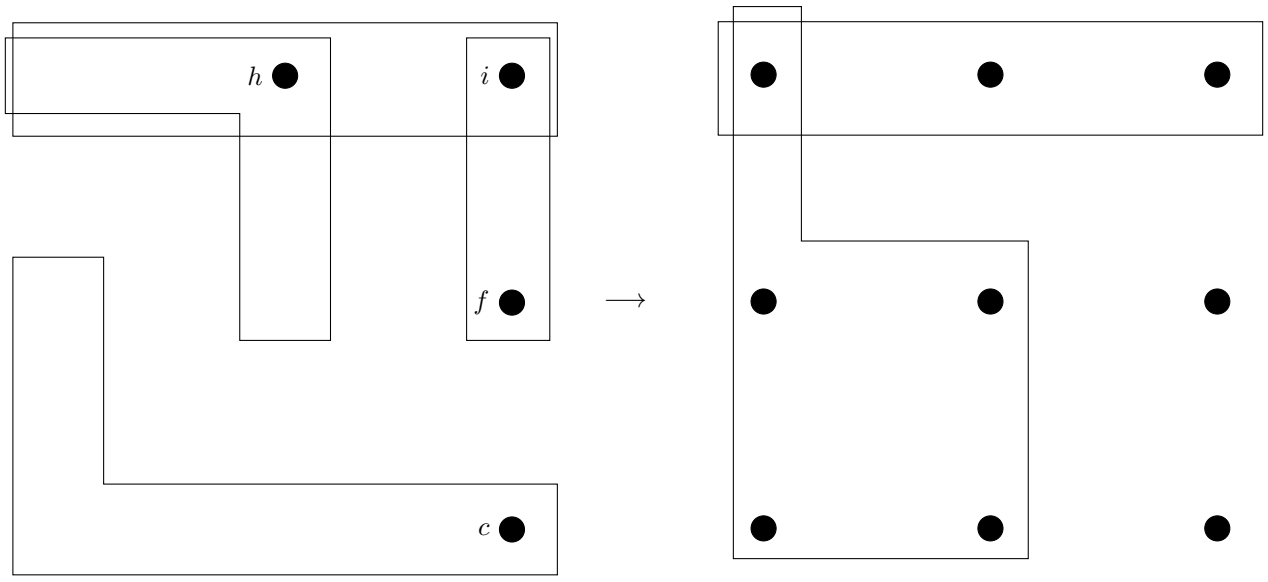
Algorithme 4 Johnson (1974), Lovàsz (1975) et Chvàtal (1979), quand tous les poids sont unitaires

1: $X' \leftarrow X$
2: $C \leftarrow \emptyset$
3: **Tant que** $X' \neq \emptyset$ **Faire**
4: $\quad S_m \leftarrow \arg\max_{S \in \mathcal{S}} |S \cap X'|$
5: $\quad$ Ajouter S_m à C
6: $\quad$ Retirer de X' les éléments de S_m.
7: **Renvoyer** C

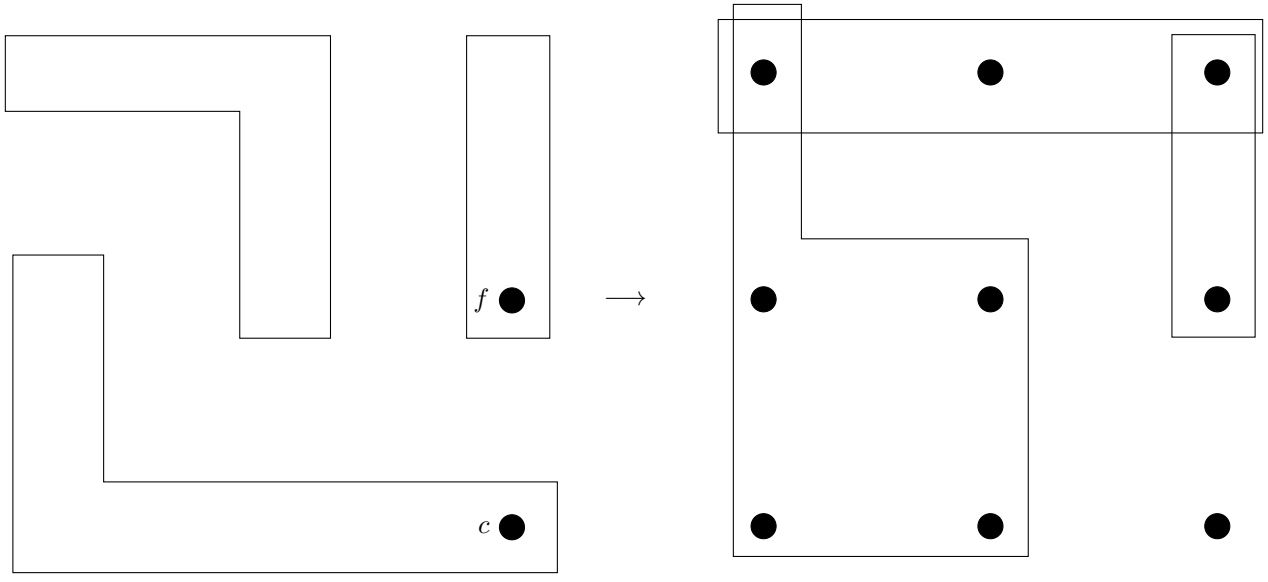
Si on applique cet algorithme sur un exemple, on a alors la solution suivante :



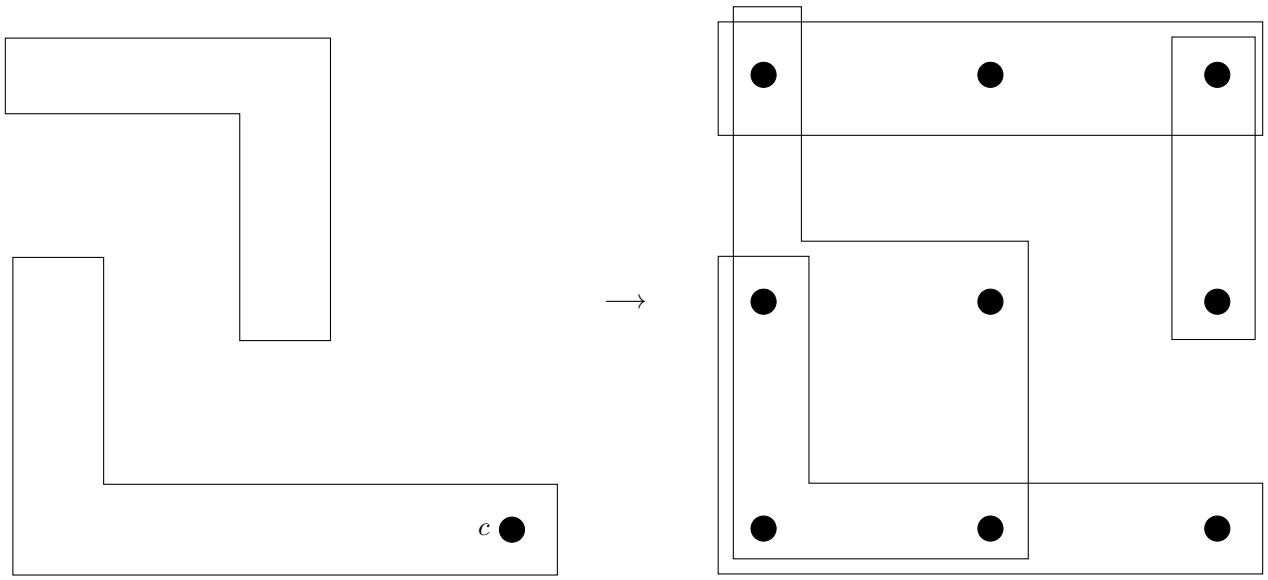
X' et $\mathcal{S}$ S_m



X' et $\mathcal{S}$ S_m



X' et $\mathcal{S}$ S_m



X' et $\mathcal{S}$ $\qquad$ S_m

On tombe ici sur une solution avec 4 ensembles, ce qui n'est pas optimal. Cet algorithme est un algorithme glouton. Il prend des décisions basées sur un objectif local (ici couvrir le plus possible avec un seul ensemble) puis ne remet plus jamais en question cette décision. Si cette stratégie semble prometteuse localement, elle a un défaut. Chaque choix est possiblement une erreur. On effectue le meilleur choix localement oui, mais, comme c'est le cas dans l'exemple, ce choix peut nous mener à devoir prendre plus d'ensembles par la suite. De plus, à chaque choix, on risque d'augmenter l'erreur. Ce qui signifie que l'erreur dépend du nombre de choix qu'on doit faire. Dans le pire cas, on effectue k choix (chaque ensemble couvre 1 nouvel élément), il est donc logique de penser que le rapport d'approximation va dépendre de k.

Pour la preuve suivante, on ne va pas utiliser la technique de la borne inférieure. On va juste essayer de mesurer la qualité de la solution au fur et à mesure des itérations. En effet, l'algorithme étant glouton, on peut chercher des propriétés concernant le premier ensemble choisi. Par exemple, on sait qu'il est plus grand que tous les ensembles de C^*. Puis, récursivement, essayer de trouver des propriétés pour le deuxième ensemble, le troisième, ...

Théorème 1.3.1. *Soit C^* une solution optimale alors $|C| = \omega(C) \leq (\ln(\frac{k}{|C^*|}) + 1) \cdot |C^*|$*

On va montrer quelques lemmes intermédiaires pour faire la preuve. Soit X'_i l'ensemble X' au début de la i^e itération et S_i l'ensemble S_m choisi à cette itération. On va d'abord s'intéresser à la taille de X'_i. Pourquoi ? Parce que si on peut prouver que $|X'_i| = 0$ alors on sait que l'algorithme fait moins de i itérations et donc $|C| < i$.

Lemme 1.3.1. $|X'_i| \leq k \cdot \left(1 - \frac{1}{|C^*|}\right)^{i-1}$

Démonstration. On va le prouver par récurrence sur i. Puisque $X'_1 = X$, alors $|X'_1| = k = k \cdot \left(1 - \frac{1}{|C^*|}\right)^0$. Le lemme est donc prouvé pour $i = 1$.

Pour rendre la suite plus compréhensible, on va effectuer la démonstration pour la première itération avant de poursuivre avec l'hérédité.

On sait que C^* couvre k éléments avec $\omega(C^*) = |C^*|$ ensembles (puisque tous les poids valent 1). Donc il existe un ensemble de taille au moins $\frac{k}{|C^*|}$, sinon la somme des tailles des ensembles de C^* serait strictement inférieure à k. Or $|S_1|$ est l'ensemble de $\mathcal{S}$ qui contient le plus d'éléments de X. Donc $|S_1| \geq \frac{k}{|C^*|}$. Lors de la deuxième itération, X'_2 contient donc au plus $k \cdot \left(1 - \frac{1}{|C^*|}\right)$ éléments. On peut appliquer maintenant le même raisonnement à n'importe quelle instance.

On suppose le lemme vrai pour une valeur i donnée. On veut le montrer pour $i + 1$. À l'itération i, on travaille avec l'instance $\mathcal{I}_i = (X'_i, \mathcal{S} \backslash \{S_1, S_2, \ldots, S_{i-1}\})$ où X'_i contient au plus $k \cdot \left(1 - \frac{1}{|C^*|}\right)^{i-1}$ par hypothèse de récurrence. La couverture $C^* \backslash \{S_1, S_2, \ldots, S_{i-1}\}$ est une solution réalisable. En effet, C^* couvre X et $\{S_1, S_2, \ldots, S_{i-1}\}$ couvre $X \backslash X'_i$. Donc la solution optimale C^*_i de l'instance $\mathcal{I}_i$ est de taille plus petite que celle de $C^* \backslash \{S_1, S_2, \ldots, S_{i-1}\}$. Donc $|C^*_i| \leq |C^* \backslash \{S_1, S_2, \ldots, S_{i-1}\}| \leq |C^*|$.

Comme dans la première itération, il existe dans C_i^* un ensemble de taille au moins $\frac{k\cdot\left(1-\frac{1}{|C^*|}\right)^{i-1}}{|C_i^*|} \geq \frac{k\cdot\left(1-\frac{1}{|C^*|}\right)^{i-1}}{|C^*|}$. Et $|S_i|$ est l'ensemble de $\mathcal{S}\backslash\{S_1, S_2, \ldots, S_{i-1}\}$ qui contient le plus d'éléments de X_i'. Donc $|S_i| \geq \frac{k\cdot\left(1-\frac{1}{|C^*|}\right)^{i-1}}{|C^*|}$. Lors de la $i+1^{\text{e}}$ itération, X_{i+1}' contient maintenant au plus $k\cdot\left(1-\frac{1}{|C^*|}\right)^{i-1} - \frac{k\cdot\left(1-\frac{1}{|C^*|}\right)^{i-1}}{|C^*|} = k\cdot\left(1-\frac{1}{|C^*|}\right)^{i}$ éléments.

Par théorème de récurrence, le lemme est vrai. □

On sait donc maintenant que l'algorithme glouton s'arrêtera au début de l'itération $i+1$ si $k\cdot\left(1-\frac{1}{|C^*|}\right)^{i} < 1$. Pour continuer, on va avoir besoin d'un autre lemme intermédiaire :

Lemme 1.3.2. *Pour tout $x \geq 1$, $f(x) = \left(1-\frac{1}{x}\right)^x < \frac{1}{e}$*

Démonstration.

$$\begin{aligned} f(x) &= e^{\ln\left(1-\frac{1}{x}\right)\cdot x} \\ f'(x) &= \left(\ln\left(1-\frac{1}{x}\right) + \frac{1}{x^2}\frac{1}{1-\frac{1}{x}}\cdot x\right)\cdot f(x) \\ f'(x) &= \left(\ln\left(1-\frac{1}{x}\right) + \frac{1}{x-1}\right)\cdot f(x) \end{aligned}$$

Puisque $x > 1$, cette dérivée est toujours strictement positive, la fonction est donc strictement croissante pour $x > 1$.

De plus, quand $x \to +\infty$, on peut utiliser le développement limité de $\ln(1-\frac{1}{x}) = \frac{-1}{x} + o(\frac{1}{x})$

$$\textit{Donc} \lim_{x\to+\infty} f(x) = \lim_{x\to+\infty} e^{\left(-\frac{1}{x}+o(\frac{1}{x})\right)\cdot x} = \frac{1}{e}$$

Donc f est strictement croissante pour tout $x > 1$ et tend vers $\frac{1}{e}$. Cette limite est donc inatteignable et $f(x) < \frac{1}{e}$ pour $x > 1$. Et, par continuité de f, c'est vrai aussi pour $x = 1$. □

Avec ce lemme, on peut donc montrer par exemple que, après $i = |C^*|\ln(k)$ itérations :

$$\begin{aligned} |X_{i+1}'| &= k\cdot\left(1-\frac{1}{|C^*|}\right)^{|C^*|\ln(k)} \\ k\cdot\left(1-\frac{1}{|C^*|}\right)^{|C^*|\ln(k)} &< k\frac{1}{e}^{\ln(k)} \\ &< 1 \end{aligned}$$

On montre ainsi que si $i = |C^*|\ln(k)$ alors l'algorithme glouton s'arête nécessairement et donc $\omega(C) \leq |C^*|\ln(k)$.

Ce n'est pas suffisant pour montrer le théorème 1.3.1. Mais, par une petite astuce, on peut arriver au résultat désiré :

Preuve du théorème 1.3.1. Si $|X_{i+1}'| \leq |C^*|$, on sait qu'il faut au plus $|C^*|$ ensembles pour couvrir les éléments restants. Et puisqu'on a déjà mis i ensembles dans C lors des i premières itérations, $|C| \leq i + |C^*|$.

Prenons $i = |C^*|\ln(\frac{k}{|C^*|})$ alors, d'après les lemmes 1.3.1 et 1.3.2 :

$$\begin{aligned} |X_{i+1}'| &= k\cdot\left(1-\frac{1}{|C^*|}\right)^{|C^*|\ln(k/|C^*|)} \\ k\cdot\left(1-\frac{1}{|C^*|}\right)^{|C^*|\ln(k/|C^*|)} &< k\frac{1}{e}^{\ln(k/|C^*|)} \\ &< |C^*| \end{aligned}$$

Donc $\omega(C) = |C| \leq |C^*|\ln(\frac{k}{|C^*|}) + |C^*| = (\ln(\frac{k}{\omega(C^*)}) + 1)\omega(C^*)$.

□

On peut facilement adapter cet algorithme au cas général, c'est-à-dire au cas où on a des poids différents de 1.

Algorithme 5 Johnson (1974), Lovàsz (1975) et Chvàtal (1979)

1: $X' \leftarrow X$
2: $C \leftarrow \emptyset$
3: **Tant que** $X' \neq \emptyset$ **Faire**
4: $\quad S_m \leftarrow \arg\min_{S \in \mathcal{S}} \frac{\omega(S)}{|S \cap X'|}$
5: $\quad$ Ajouter S_m à C
6: $\quad$ Retirer de X' les éléments de S_m.
7: **Renvoyer** C

Dans ce cas le rapport reste logarithmique.

Théorème 1.3.2. *Soit C^* une solution optimale alors $\omega(C) \leq (\ln(k)+1) \cdot \omega(C^*)$*

Cette fois, il est plus difficile de s'intéresser à la taille de X'_i, comme on l'a fait dans le cas non pondéré car on ne maîtrise pas la taille de S_i. On va s'intéresser au i^e élément couvert par l'algorithme : soit $x \in X$, on note $i(x)$ l'itération où x a été couvert. On ordonne ensuite $X = (x_1, x_2, \ldots, x_k)$ selon $i(x)$ (les premiers éléments couverts en premier, les derniers en dernier). Par exemple, S_1 contient les éléments $x_1, x_2, \ldots, x_{|S_1|}$.

Définition 1. Soit $j \in [\![1;k]\!]$, $S = S_{i(x_j)}$ et $X' = X'_{i(x_j)}$ alors on note $price_j = \omega(S)/|S \cap X'|$.

Vous remarquerez que $price_j$ est la formule utilisée à la ligne 4 pour sélectionner S_m. À quoi correspond cette valeur ? Si on prend tous les éléments couverts par S, et qu'on additionne leur valeur $price_j$, on tombe sur $\omega(S)$. En effet, il y a $|S \cap X'|$ éléments couverts par S à l'itération $i(x_j)$. Donc $price_j$ est le prix qu'il a fallu payer pour couvrir x_j. On va utiliser $price_j$ comme une borne inférieure.

Lemme 1.3.3. *Pour tout $j \in [\![1;k]\!]$, $price_j \leq \omega(C^*)/(k-j+1)$.*

Démonstration. Plaçons nous à l'itération $i = i(x_j)$.

Puisqu'à l'itération $i(x_j)$ on couvre l'élément x_j, et puis que x_j est couvert avant $x_{j+1}, x_{j+2}, \ldots, x_k$, on sait que X'_i contient tous ces éléments : $|X'_i| \geq k-j+1$. Il suffit donc de montrer que $price_j \leq \omega(C^*)/|X'_i|$ pour trouver le résultat.

On s'intéresse donc à l'instance $(X'_i, \mathcal{S} \backslash \{S_1, S_2, \ldots, S_{i-1}\})$. Dans cette instance $C^*_i = C^* \backslash \{S_1, S_2, \ldots, S_{i-1}\}$ est une solution réalisable de coût inférieur à $\omega(C^*)$.

Or

$$\begin{aligned}\sum_{S^* \in C^*_i} \sum_{x \in S^* \cap X'_i} \frac{\omega(S^*)}{|S^* \cap X'_i|} &= \sum_{S^* \in C^*_i} \omega(S^*) \\ &\leq \omega(C^*)\end{aligned}$$

Il existe donc au moins un ensemble $S^* \in C^*_i$ tel que $\frac{\omega(S^*)}{|S^* \cap X'_i|} \leq \frac{\omega(C^*)}{|X'_i|}$. En effet, dans le cas contraire :

$$\begin{aligned}\sum_{S^* \in C^*_i} \sum_{x \in S^* \cap X'_i} \frac{\omega(S^*)}{|S^* \cap X'_i|} &> \sum_{S^* \in C^*_i} \sum_{x \in S^* \cap X'_i} \frac{\omega(C^*)}{|X'_i|} \\ &> \sum_{S^* \in C^*_i} |S^* \cap X'_i| \frac{\omega(C^*)}{|X'_i|} \\ &> \left(\sum_{S^* \in C^*_i} |S^* \cap X'_i| \right) \frac{\omega(C^*)}{|X'_i|}\end{aligned}$$

Or C^*_i couvre X'_i donc $\sum_{S^* \in C^*_i} |S^* \cap X'_i| \geq |X'_i|$, donc

$$> \omega(C^*)$$

Ce qui est absurde, étant donné qu'on a prouvé le contraire plus haut.

Or $price_j = \omega(S_i)/|S_i \cap X'_i|$ et $S_i = \arg\min_{S \in \mathcal{S} \backslash \{S_1, S_2, \ldots, S_{i-1}\}} \frac{\omega(S)}{|S \cap X'_i|}$ d'après la 4e ligne de l'algorithme. Donc $price_j \leq \min_{S^* \in C^*_i} \frac{\omega(S^*)}{|S^* \cap X'_i|} \leq \frac{\omega(C^*)}{|X'_i|}$, ce qui conclue la preuve de ce lemme. □

On peut maintenant terminer la preuve du théorème :

Preuve du Théorème 1.3.2. D'après le lemme 1.3.3, $price_j \leq \omega(C^*)/(k-j+1)$. De plus, à toute itération i de l'algorithme glouton, si on additionne les valeurs $price_j$ des élements x_j couverts pendant cette itération, on obtient le poids $\omega(S_i)$ de l'ensemble sélectionné à la ligne 4, donc $\sum_{x_j \in S_i} price_j = \omega(S_i)$.

Ainsi :

$$\begin{aligned}\omega(C) &= \sum_{i=1}^{i(x_k)} \omega(S_i) \\ &= \sum_{j=1}^{k} price_j \\ &\leq \sum_{j=1}^{k} \omega(C^*)/(k-j+1) \\ &\leq \left(\sum_{j=1}^{k} \frac{1}{j}\right) \omega(C^*)\end{aligned}$$

Enfin, $H_k = \left(\sum_{j=1}^{k} \frac{1}{j}\right)$ est ce qu'on appelle le k^{e} nombre harmonique.
Ce nombre est inférieur à $\ln(k)+1$ ([2]). □

Pour finir cette partie, on peut montrer que le problème de couverture par ensembles ne peut pas être approché avec un meilleur rapport (sauf si P = NP). Donc, contrairement au problème de l'arbre de Steiner ou au problème de voyageur de commerce (dans le cas métrique), il n'existe pas d'algorithme d'approximation de rapport constant. Cette preuve est difficile et requiert des outils complexes que l'on abordera dans le dernier chapitre. Toujours est-il que ce problème est donc un bon exemple de cas où faire simple donne de bons résultats.

Il est bien entendu possible de faire mieux en pratique mais, en terme de garantie, aucun algorithme ne peut battre l'algorithme glouton en temps polynomial. On verra toutefois dans le troisième chapitre comment fabriquer des algorithmes d'approximations polynomiaux à l'aide de la programmation linéaire, et nous utiliserons la couverture par ensembles comme exemple. En particulier, nous redécouvrirons l'algorithme glouton à l'aide de la programmation linéaire.

1.4 Le problème du sac à dos

Le problème de sac à dos est le suivant.

Problème : Problème de sac à dos (Knapsack en anglais)

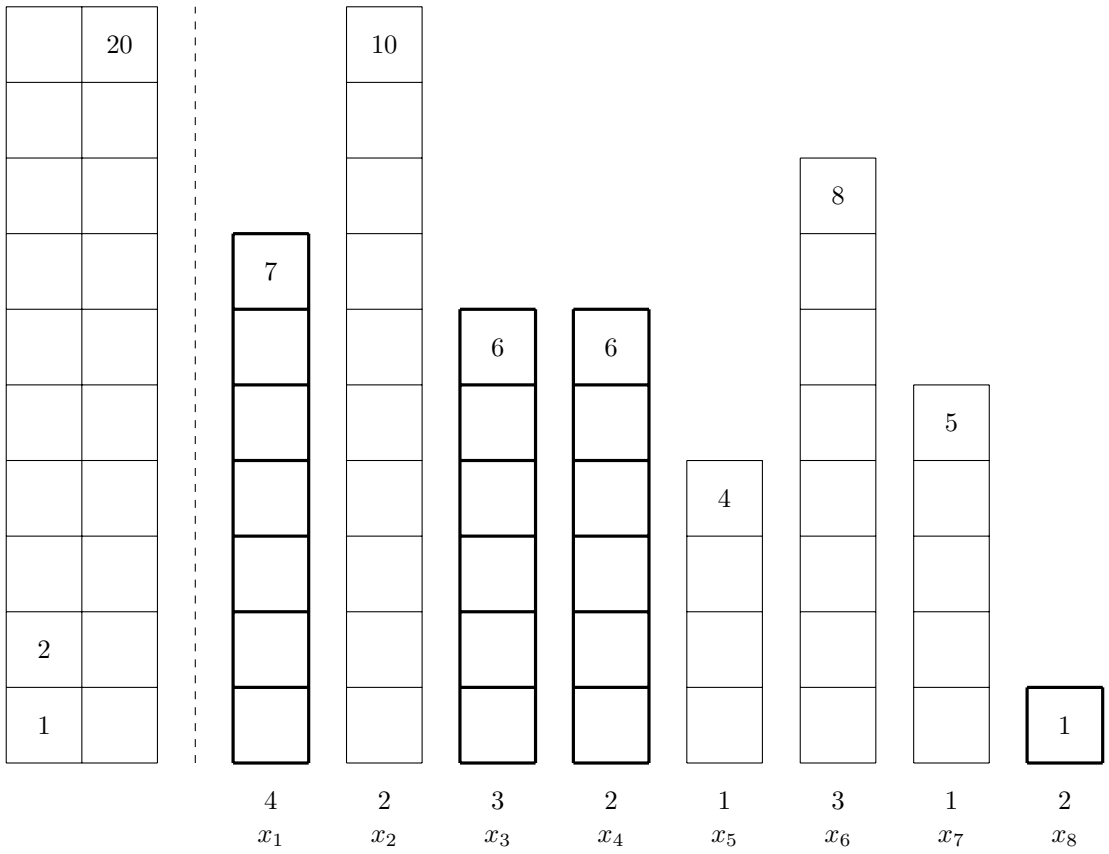
Instance :
- Un entier V, correspondant au volume d'un sac
- Des entiers $v_1, v_2, \ldots, v_n$ correspondant aux volumes de n objets
- Des entiers $u_1, u_2, \ldots, u_n$ correspondant à l'utilité (ou la valeur) des n objets.

Solution réalisable : Un sous-ensemble $I \subset [\![1; n]\!]$ des n objets tel que $\sum_{i \in I} v_i \leq V$

Optimisation : Minimiser l'utilité $U(I) = \sum_{i \in I} u_i$.

Prenons par exemple l'instance suivante où le sac, à gauche, est de volume 20. A droite, les objets, disposés avec leur utilité en dessous et le volume au dessus. Chaque carré est un volume de 1. La solution optimale est en gras avec les objets x_1, x_3, x_4 et x_8, avec un volume de 20 et une utilité de 11.

2. `https://fr.wikipedia.org/wiki/Série_harmonique`



Ce problème est NP-Difficiles (et fait lui aussi partie des 21 problèmes NP-Complets de Karp). Cependant, à la différence des 3 problèmes précédents, le problème de Sac à dos est plus facile à résoudre : il est NP-Difficile au sens faible. C'est-à-dire qu'il existe une réduction qui le prouve NP-Difficile mais qu'il existe tout de même un algorithme *pseudo-polynomial* pour le résoudre. Pour rappel, un algorithme est pseudo-polynomial si la complexité de cet algorithme dépend polynomialement de la valeur de tous ou une partie des entiers qui constitue l'instance. Ici l'instance est constituée de $2n + 1$ entiers.

Il existe 2 algorithmes pseudo-polynomiaux classiques pour résoudre le problème du sac à dos. Les deux sont présentés ici car une différence subtile les séparent et cette différence va beaucoup jouer quant à la technique utilisée pour approcher le problème du sac à dos.

Le premier algorithme pseudo polynomial utilise la programmation dynamique basée sur la formule de récurrence suivante. Soit $f(i, v)$ l'utilité maximum qu'on peut atteindre si on utilise seulement les i premiers objets et un sac de volume v. Savoir calculer f pour tout i et v nous donnerait la réponse à l'aide de $f(n, V)$. La fonction f peut se calculer par récurrence avec la formule

$$f(i,v) = \begin{cases} 0 \text{ si } i = 0 \text{ quelque soit } v \\ f(i-1,v) \text{ si } v < v_i \\ \max f(i-1,v), f(i-1,v-v_i) + u_i \text{ sinon} \end{cases}$$

Cette formule s'exprime simplement en français : s'il n'y a pas d'objet, alors l'utilité maximum est nulle. Sinon, si on ne peut mettre l'objet i dans le sac, on ne le prend pas ; et si on peut le mettre alors on teste les deux possibilités qui consistent à le prendre ou à ne pas le prendre.

Pour calculer cette formule, on utilise la programmation dynamique, par exemple en construisant le tableau suivant. (Ce tableau a été construit à l'aide de la technique de mémoïsation, ce qui explique que certaines cases n'ont pas été calculées.)

i \ v	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4			4	4
2	0	0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4			6	6
3	0	0	0	0	0	0	3	4	4	4	4	4	4	7	7	7	7			7	7
4			0	0			3	4	4		4	4	5		7	7	7			9	9
5							3	4				5	5		7	7				9	9
6															7	7				9	9
7																				9	9
8																					11

On peut y lire par exemple que $f(8, 20) = 11$, comme prévue par la solution optimale, et que ça a été calculé par récurrence comme $f(8, 20) = \max(f(7, 20), f(7, 19) + 2) = \max(9, 9 + 2)$.

La complexité de cet algorithme est de $O(nV)$. Pour rappel, cette complexité n'est pas polynomiale si V est codé en binaire (avec $\log(V)$ bits) puisqu'elle est égale à $O(n2^{\log(V)})$. Par contre elle est polynomiale si V est codé en unaire (avec V bits). Cette différence de codage est cruciale puisque c'est elle qui peut faire perdre beaucoup de temps à l'exécution de l'algorithme et qui explique qu'on catégorise cet algorithme comme *pseudo-polynomial*. Alors certes, si V est égal par exemple à 100, quelque soit l'algorithme utilisé, il faudra créer un tableau de taille $100n$ et le remplir ; et ce remplissage prendra autant de temps que V soit codé en binaire ou en unaire. Donc en soit, dire que la complexité est polynomiale ou exponentielle semble relèver plus d'un aspect théorique que pratique.

Malgré tout, cette différence se ressent quand on change la valeur de V. Supposons que la personne qui crée l'instance et la personne qui la résout ne soit pas la même. Supposons que V soit codé en binaire et que la première personne décide de multiplier V par 2. Alors, elle verra le temps de calcul nécessaire à la génération de l'instance quasiment inchangée. En binaire, multiplier par 2 revient à ajouter un bit 1 devant le nombre : temps constant. Pour la 2e personne par contre, le temps de calcul sera multiplié par 2. Et c'est bien là qu'on voit à quel point notre algorithme est exponentiel. Si on augmente la taille de i bits, alors on multiplie le temps de calcul par 2^i. Une autre manière de voir les choses est que l'algorithme nécessite que V soit écrit en unaire, et c'est la transformation binaire vers unaire qui prend un temps exponentiel. Par contre si V est déjà écrit en unaire, alors cette transformation ne se fait pas et l'algorithme est polynomial.

Bref, que nous apprennent toutes ces considérations ? Encore une fois, si V est fixé égal à 100, le temps de calcul de l'algorithme sera sensiblement le même. Pour répondre à cette question, il faut se poser une dernière question. Pourquoi écrivons-nous les nombre en binaire au lieu de les écrire en unaire ? Parce que cela prend moins de place et qu'il est possible d'effectuer les opérations de bases sur des nombres binaires sans les transformer en unaire. Un disque dur aura donc des nombres écrits en binaire. Et sur un disque dur il est donc possible d'écrire des nombres très grands, exponentiellement plus grand que s'ils étaient écrits en unaire. Il y a donc toutes les chances pour que V ne vaille pas 100 mais plutôt 10^9, auquel cas la transformation binaire vers unaire se fera ressentir grandement [3] ; voire sera impossible à utiliser car il n'y aura pas la place mémoire pour écrire le tableau de taille nV. Donc l'algorithme pseudopolynomial n'est utilisable qu'avec de petites valeurs de V, des valeurs bien plus petites que si la complexité était polynomiale.

Le second algorithme pseudopolynomial est aussi basé sur une formule de récurrence. Soit $g(i, u)$ le volume minimum qu'on peut atteindre si on utilise seulement les i premiers objets et qu'on cherche à atteindre exactement l'utilité u. On suppose que $g(i, u) = +\infty$ si l'utilité u n'est pas atteignable avec les i premiers objets. Savoir calculer g pour tout i et u nous donnerait la réponse à l'aide de $\arg\max\{u \leq \sum_{i=1}^{n} u_i | g(n, u) \neq +\infty\}$. La fonction g peut se calculer par récurrence avec la formule

$$g(i,u) = \begin{cases} +\infty \text{ si } i = 0 \text{ et } u \neq 0 \\ 0 \text{ si } i = 0 \text{ et } u = 0 \\ g(i-1,u) \text{ si } u < u_i \\ \min g(i-1,u), g(i-1,u-u_i) + v_i \text{ sinon} \end{cases}$$

Cette formule s'exprime également en français : s'il n'y a pas d'objet, alors la seule utilité atteignable est 0, et le volume utilisé par 0 objets dans ce cas est 0. Sinon, si l'objet i a une utilité trop élevée par rapport à u, on ne le prend pas ; et si son utilité est plus petite que u alors on teste les deux possibilités qui consistent à le prendre ou à ne pas le prendre. On peut calculer cette formule par programmation dynamique aussi. On obtient le tableau suivant (par soucis de place, les $+\infty$ n'ont pas été affichés).

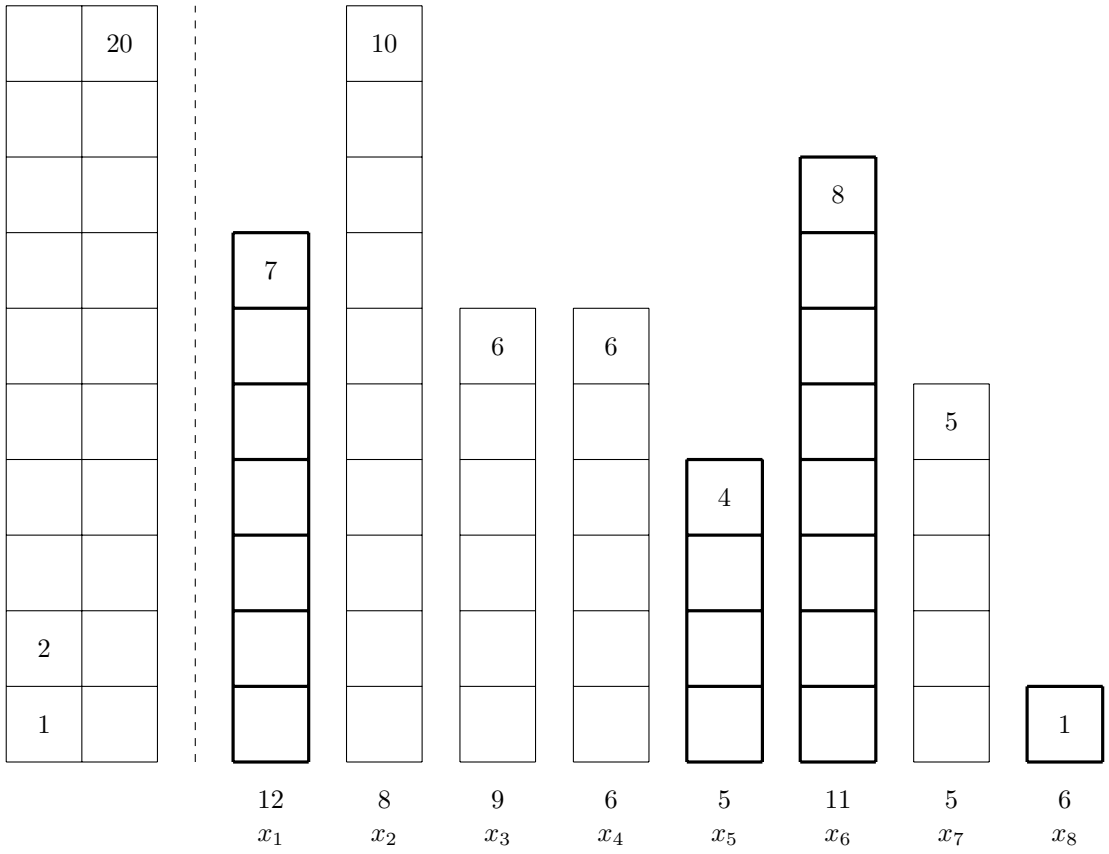
3. Pour information, il faudra toute une vie humaine, si ce n'est plus, pour compter à voix haute de 1 à 10^9, et c'est pourtant ce qui est demandé à notre ordinateur. `https://www.youtube.com/watch?v=oqMYAVV-hsA`

i \ u	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0				7														
1	0		10		7		17												
2	0		10	6	7	16	17	13											
3	0		6	6	7	12	13	13		19									
4	0	4	6	6	7	11	13	13	17	19									
5	0	4	6	6	7	11	13	13	17	19									
6	0	4	6	6	7	11	13	13	17	19									
7	0	4	1	5	7	7	8	12	14	14	18	20							

On observe donc ici qu'il est possible d'atteindre une utilité de 11 avec les 7 objets, à l'aide d'un volume 20, qu'il est possible d'atteindre une utilité de 9 avec les 3 premiers objets, avec un volume 19, ... La complexité de cet algorithme est $O(n \sum_{i=1}^{n} u_i)$. Pour les mêmes raisons que celles expliquées plus haut, cet algorithme ne peut être utilisé pour de grandes valeurs d'utilité.

Une idée, pour résoudre rapidement le problème du sac à dos serait donc de réduire artificiellement V ou $\sum_{i=1}^{n} u_i$ pour que le problème devienne solvable avec un algorithme pseudo-polynomial. On transpose ensuite la solution obtenue dans l'instance originale.

Prenons l'instance suivante où la solution optimale d'utilité 34 est constituée des objets x_1, x_5, x_6 et x_8.



On veut diviser les utilités par 3. Attention, on ne peut pas faire bêtement $\frac{u_i}{3}$. En effet, le problème de sac à dos veut des entiers. Et nos algorithmes pseudo-polynomiaux se basent sur le fait que ces utilités soient entières. Donc, à la place, u_i devient $\lfloor \frac{u_i}{3} \rfloor$. On retombe sur l'instance présentée en début de partie. Le second algorithme pseudo polynomial, basé sur les utilités, trouve et renvoie la solution x_1, x_3, x_4, x_8. C'est bien une solution réalisable (les volumes n'ont pas changé), de valeur 33. Et puisque les utilités ont été réduites le temps de calcul est plus faible.

Peut-on utiliser le premier algorithme pseudo-polynomial, basé sur V et la fonction f, plutôt que le second ? En un mot, non. Il faut réduire V pour pouvoir l'utiliser rapidement. On divise V par une valeur K sous la forme $\lfloor \frac{V}{K} \rfloor$. Il y a alors un risque, en réduisant V, de modifier l'ensemble des solutions réalisables. Un exemple simple, si vous réduisez V à 0 en le divisant par $K \geq V+1$, plus aucune solution n'est réalisable. Une alternative serait de réduire V en $\lfloor \frac{V}{K} \rfloor$ et de réduire les volumes v_i en $\lfloor \frac{v_i}{K} \rfloor$. Mais le problème reste le même : si $K \geq V+1$

et $K \geq v_i + 1$, tous les volumes deviennent nuls et donc toutes les solutions deviennent réalisables. Dans les deux cas, utiliser l'algorithme pseudo-polynomial risque de vous donner une solution qui n'est pas réalisable ou de ne pas trouver de solution réalisable. L'algorithme pseudo polynomial sur les utilités est très adapté à cette technique parce que réduire les utilités sans toucher aux volumes conserve l'ensemble des solutions réalisables.

Voici donc l'algorithme utilisé :

Algorithme 6 Ibarra et Kim (1975)

Entrées: Un réel $\varepsilon > 0$

1: $K \leftarrow \frac{\varepsilon \max\{u_i\}}{n}$
2: **Pour** $i \in [\![1; n]\!]$ **Faire**
3: $\quad u'_i \leftarrow \lfloor \frac{u_i}{K} \rfloor$
4: $\mathcal{I}' \leftarrow$ l'instance avec les mêmes volumes V et v_i, et avec les utilités u'_i
5: **Renvoyer** la solution I' de l'instance $\mathcal{I}'$ obtenue avec l'algorithme pseudopolynomial basé sur les utilités

Cet algorithme prend en entrée un paramètre ε. Ce paramètre est fixé tout au long de l'algorithme, il est connu à l'avance et n'est pas choisi en fonction de l'instance. Donc il ne dépend pas des valeurs des entiers de l'instance. Donc, pour chaque valeur de ε, on a un algorithme différent. On utilisera par exemple l'algorithme pour $\varepsilon = 0.1$ ou $\varepsilon = 0.01$.

La complexité de cet algorithme est $O(n \cdot \sum_{i=1}^{n} u'_i)$. Or $u'_i = \lfloor \frac{u_i \cdot n}{\varepsilon \max(u_i)} \rfloor \leq \lfloor \frac{n}{\varepsilon} \rfloor$, donc la complexité est de l'ordre de $O(n \lfloor \frac{n}{\varepsilon} \rfloor)$. Autrement dit plus ε est petit, plus la complexité en temps augmente. En contrepartie, on va montrer que plus ε est petit et meilleur est le rapport d'approximation. Ce qui est logique car, dans ce cas, K diminue et si K diminue, la différence entre les u_i et u'_i s'estompe. Donc le temps de calcul augmente (pour se rapprocher du temps de résolution exact de l'instance) et la précision augmente (pour se rapprocher aussi de celle d'un algorithme exact).

Théorème 1.4.1. *Soit I^* la solution optimale de l'instance, alors $U(I') \geq (1 - \varepsilon)U(I^*)$.*

Démonstration. Posons $U'(I) = \sum_{i \in I} u'_i$.

On connait une propriété sur I' qui est que I' est la meilleure solution réalisable de $\mathcal{I}'$. Puisque les solutions réalisables de $\mathcal{I}$ et $\mathcal{I}'$ sont les mêmes, I^* est solution réalisable de $\mathcal{I}'$. Donc $U'(I') \geq U'(I^*)$.

De plus, par définition de la partie entière, $\frac{u_i}{K} - 1 \leq u'_i = \lfloor \frac{u_i}{K} \rfloor \leq \frac{u_i}{K}$.

Donc

$$\begin{aligned} U(I') &= \sum_{i \in I'} u_i \\ &\geq \sum_{i \in I'} (K \cdot u'_i) \\ &\geq K U'(I') \\ &\geq K U'(I^*) \\ &\geq K \sum_{i \in I^*} u'_i \\ &\geq K \sum_{i \in I^*} \frac{u_i}{K} - nK \\ &\geq U(I^*) - nK \end{aligned}$$

Enfin $nK = \varepsilon \max(u_i)$. Et $\max(u_i) \leq U(I^*)$ (on suppose qu'il n'y a pas d'objet trop gros pour rentrer dans le sac, ainsi toute solution constistant à placer un objet dans le sac est réalisable). Donc $nK \leq \varepsilon U(I^*)$.

$$U(I') \geq (1 - \varepsilon)U(I^*)$$

□

D'après ce théorème, il suffit de choisir un ε suffisamment petit pour améliorer le rapport d'approximation, contre une augmentation du temps de calcul. On dit que cet algorithme est un schémas d'approximations polynomiales ou PTAS pour polynomial time approximation scheme. Plus précisément, il s'agit ici d'un FPTAS ou Fully polynomial time approximation scheme car la complexité dépend polynomialement de $\frac{1}{\varepsilon}$ et non exponentiellement [4].

4. Contrairement à ce qui est expliqué plus haut, le fait que l'algorithme soit un FPTAS nous permet de choisir un ε non constant, qui dépend de n ; par exemple $\varepsilon = \frac{1}{n^c}$. Dans ce cas, la complexité de l'algorithme n'est plus quadratique mais devient $O(n^{c+2})$ ce qui reste polynomial. Ce n'est pas faisable si la complexité dépend exponentiellement de $\frac{1}{\varepsilon}$.

Quelques détails pour finir : on voit ici une différence entre l'approximation d'un problème de maximisation et de minimisation, dans le sens des inégalités. C'est un détail important pour la suite, afin de ne pas se tromper dans les définitions. Bien sûr on pourra toujours se ramener à un problème de minimisation en multipliant les poids par -1 mais il est plus agréable de travailler avec des nombres positifs ; et les théorèmes décrits dans le cours ne fonctionnent pas si les poids sont négatifs : il existe une réelle différence entre ces deux catégories de problèmes.

Autre détail, que se passe-t-il si on fait tendre ε vers 0 ? Le théorème nous dit qu'on tend vers un algorithme exact. En fait, K tendra lui aussi vers 0. Étrangement, il n'est pas nécessaire d'avoir $K = 0$ pour obtenir un algorithme exact, il suffit d'avoir $K = 1$ c'est-à-dire $\varepsilon = \frac{n}{\max(u_i)}$. Dans ce cas, pourquoi le rapport d'approximation $(1 - \varepsilon)$ du théorème ne nous informe pas de cette possibilité ? C'est parce qu'on se place toujours dans le pire cas. Notre rapport d'approximation est vrai quelle que soit l'instance. Et la pire instance possible, c'est l'instance où les valeurs des utilités tendent vers l'infini. Dans ce cas, ε doit se rapprocher de 0 pour que K tende vers 1.

1.5 Le problème du bin packing

Le problème du bin packing est le suivant.

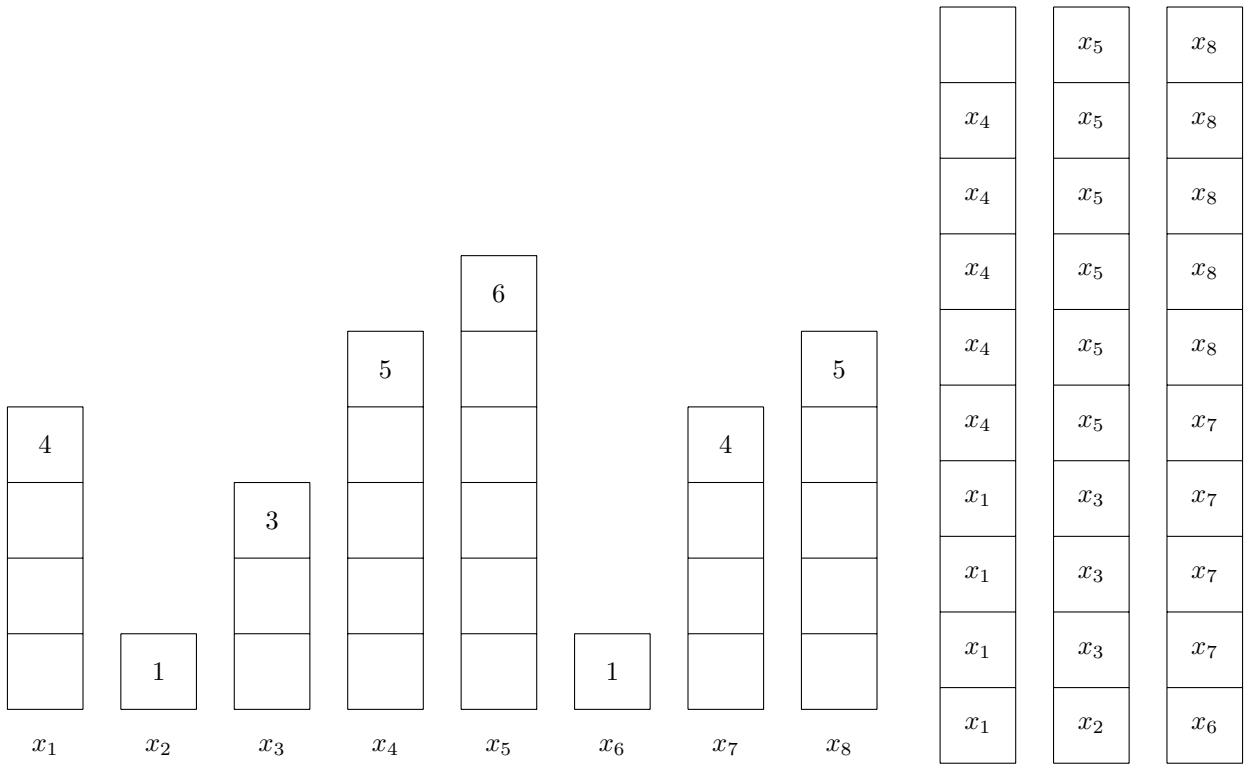
Problème : Problème du bin packing (Pas de traduction en français à ma connaissance)

Instance :
- Un entier V, correspondant au volume d'un sac
- Des entiers $v_1, v_2, \ldots, v_n$ correspondant aux volumes de n objets

Solution réalisable : Un sous-ensemble de sacs de volumes V dans lesquels les n objets sont placés : n entiers $S = (s_1, s_2, \ldots, s_n)$ tels que, pour tout entier i, $\sum_{i|s_i=j} v_i \leq V$

Optimisation : Minimiser le nombre de sacs utilisés, soit $\max(S)$.

Ce problème est NP-Difficile. Malgré sa ressemblance avec le problème du sac à dos, il n'existe pas d'algorithme pseudo-polynomial pour le résoudre, il est NP-Difficile au sens fort. La figure ci-dessous présente un exemple où $V = 10$ avec une solution optimale.



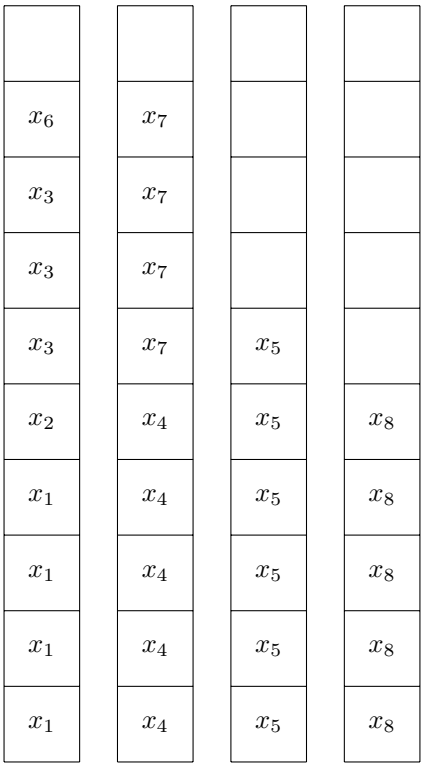
Des approximations basiques

L'algorithme le plus basique qui soit pour résoudre ce problème a un rapport constant.

Algorithme 7 FirstFit

1: $V_1, V_2, \ldots, V_n \leftarrow 0, 0, \ldots, 0$
2: **Pour** j de 1 à n **Faire**
3: **Pour** i de 1 à n **Faire**
4: **Si** $V_i + v_j \leq V$ **Alors**
5: $S_j \leftarrow i$
6: $V_i \leftarrow V_i + v_j$
7: Passer à l'itération suivante de la boucle For extérieure.
8: **Renvoyer** $S_1, S_2, \ldots, S_n$

Cet algorithme consiste à mettre chaque objet dans le premier sac où il peut rentrer. La complexité de cet algorithme est $O(n)$, trivialement polynomiale. Appliqué à notre exemple ci-dessus, il donnerait la solution suivante :



Théorème 1.5.1. *Soit S^* une solution optimale et S la solution renvoyée par FirstFit, alors* $\max(S) \leq 2\max(S^*)$.

Démonstration. On peut réutiliser une borne inférieure ici : le volume des sacs utilisés est supérieur au volume des objets qu'on met dedans. Autrement dit $V\max(S^*) \geq \sum_{i=1}^{n} v_i$.

On va maintenant prouver que $\sum_{i=1}^{n} v_i > \frac{\max(S)-1}{2}V$. Ce qui prouverait que $\frac{\max(S)-1}{2} < \max(S^*)$, donc $\max(S) < 2\max(S^*) + 1$ et donc le résultat recherché car $2\max(S^*) + 1$ est un entier.

Pour cela, il suffit de constater que chaque sac, sauf éventuellement un sac, est au moins à moitié rempli. Plaçons nous dans le cas contraire pour le prouver par l'absurde : sans perte de généralité, supposons que $s_1, s_2, \ldots, s_p = 1$ et $s_{p+1}, s_{p+2}, \ldots, s_q = 2$ et que $\sum_{i=1}^{p} v_i \leq \frac{V}{2}$ et $\sum_{i=p+1}^{q} v_i \leq \frac{V}{2}$. Dans ce cas, $v_{p+1} \leq \frac{V}{2}$. Or, à l'itération $j = p+1$ et $i = 1$ de l'algorithme First fit, on a placé dans le sac 1 tous les objets de 1 à p et aucun autre objet (qui seront placés aux itérations ultérieures), donc $V_i = \sum_{i=1}^{p} v_i \leq \frac{V}{2}$; ainsi à la ligne 4, $V_i + v_j \leq V$ et donc l'objet $p+1$ aurait dû être placé dans le sac 1 lui aussi et non dans le sac 2.

Donc tous les sacs sont à moitié remplis sauf 1, le volume total des objets est donc au moins strictement supérieur à la moitié des volumes de $\max(S) - 1$ sacs, autrement dit $\sum_{i=1}^{n} v_i > \frac{\max(S)-1}{2} V$. D'où le résultat d'approximation. □

Il existe un algorithme encore plus simple qui est lui aussi une approximation de rapport 2. Il s'agit de l'algorithme NextFit, qui consiste à toujours mettre l'objet dans le dernier sac ouvert si cela est possible et sinon ouvrir un nouveau sac. On perd clairement en efficacité au niveau de la valeur de la solution mais on gagne en simplicité. La preuve est un peu différente de celle de FirstFit puisqu'on a plus cette propriété que les sacs sont à moitié remplis.

Algorithme 8 NextFit

1: $c \leftarrow 1$
2: $V_c \leftarrow 0$
3: **Pour** j de 1 à n **Faire**
4: **Si** $V_c + v_j > V$ **Alors**
5: $c \leftarrow c + 1$
6: $V_c \leftarrow 0$
7: $S_j \leftarrow c$
8: $V_c \leftarrow V_c + v_j$
9: **Renvoyer** $S_1, S_2, \ldots, S_n$

Ici c correspond au dernier sac ouvert, si l'objet ne rentre pas, on ouvre un nouveau sac à la ligne 5. Sur l'exemple présenté plus haut, cela donnerait la solution suivante :

			x_8
x_3			x_8
x_3		x_6	x_8
x_3		x_5	x_8
x_2	x_4	x_5	x_8
x_1	x_4	x_5	x_7
x_1	x_4	x_5	x_7
x_1	x_4	x_5	x_7
x_1	x_4	x_5	x_7

Théorème 1.5.2. *Soit S^* une solution optimale et S la solution renvoyée par NextFit, alors* $\max(S) \leq 2\max(S^*)$.

Démonstration. On a toujours $V \max(S^*) \geq \sum_{i=1}^{n} v_i$ et on va encore prouver que $\sum_{i=1}^{n} v_i > \frac{\max(S)-1}{2} V$. Pour cela, il suffit de constater que pour tout $i < \max(S)$, la somme des volumes des objets placés dans le sac i et dans le sac $i+1$ est strictement supérieure à V. En effet, dans cas contraire, le premier objet placé dans le sac $i+1$ aurait été placé dans le sac i.

Soit V_i le volume du sac i. Si $\max(S)$ est pair, alors $\sum_{i=1}^{\max(S)} V_i = \sum_{i=1}^{\max(S)/2} (V_{2i}+V_{2i+1}) > \frac{\max(S)}{2}V > \frac{\max(S)-1}{2}V$. Si $\max(S)$ est impair, alors $\sum_{i=1}^{\max(S)} V_i = \sum_{i=1}^{(\max(S)-1)/2} (V_{2i}+V_{2i+1}) > \frac{\max(S)-1}{2}V$. D'où le résultat d'approximation. □

Ces deux approximations fonctionnent très bien en pratique, de par leur simplicité, et la garantie de performances qu'elles apportent ne font que conforter les utilisateurs dans le choix qu'ils ont fait. On peut ensuite utiliser des heuristiques ou méta-heuristiques pour améliorer la solution trouvée.

Un algorithme (beaucoup) moins basique

On va maintenant s'intéresser à un dernier algorithme, sans aucun doute l'algorithme le plus compliqué de tous, d'un intérêt pratique douteux mais d'un intérêt théorique certain.

Cet algorithme est à considérer en plusieurs parties. On se place dans un cas simple, et on montre qu'il est facile de résoudre le problème dans ce cas. On se place ensuite dans un cas plus difficile et on essaie de se ramener au cas facile, en estimant la perte due à la transformation *cas difficile vers cas facile*. Enfin on se place dans le cas général et on transforme en le cas difficile.

Commençons par remarquer que, à condition d'accepter d'utiliser des rationnels au lieu d'utiliser des entiers pour les volumes, on peut considérer que le volume V est égal à 1. En effet, il suffit alors de transformer v_i en $\frac{v_i}{V}$. On peut noter que cette transformation ne pose pas de problème aux algorithmes FirstFit et NextFit.

Avant de continuer, nous aurons besoin d'un lemme sur le dénombrement. Supposons que vous ayez n boules placées sur une ligne et que vous souhaitiez inscrire un nombre entre 1 et k sur chaque boule. Combien y a-t-il de résultats *triés* possibles ? Autrement dit, vous devez commencer par les boules de gauches, et écrire autant de 1 que vous le souhaitez, puis autant de 2 que vous le souhaitez, et ainsi de suite jusqu'à k. Il est possible de sauter un numéro. On note $f(n,k)$ ce nombre.

Lemme 1.5.1. *$f(n,k) = \binom{n+k-1}{k-1}$ séquences de boules possibles.*

Démonstration. Prouvons le par récurrence sur le nombre de boules n. Si $n = 1$, alors vous pouvez écrire tout nombre entre 1 et k sur la boule, donc $k = \binom{1+k-1}{k-1}$ possibilités.

Si le résultat est vrai pour n boules, est-il vrai pour $n+1$ boules ? Si on dispose de $n+1$ boules, alors on a k possibilités pour la boule 1. Par contre, si on écrit i sur la boule 1, on ne peut plus écrire $1, 2, \ldots, i-1$ sur les boules 2 à $n+1$. Le possibilités où il est écrit i sur la boule 1 est donc égal au nombre de possibilités d'écrire les entiers de i à k sur les boules 2 à $n+1$, c'est à dire $f(n, k-(i-1))$. On a donc

$$f(n+1,k) = \sum_{i=1}^{k} f(n, k-(i-1))$$
$$f(n+1,k) = \sum_{i=1}^{k} f(n,i)$$

Par hypothèse de récurrence,

$$= \sum_{i=1}^{k} \binom{n+i-1}{i-1}$$
$$= \binom{n}{0} + \binom{n+1}{1} + \binom{n+2}{2} + \cdots + \binom{n+k-1}{k-1}$$

Or, $\binom{n}{0} = \binom{n+1}{0} = 1$

$$= \binom{n+1}{0} + \binom{n+1}{1} + \binom{n+2}{2} + \cdots + \binom{n+k-1}{k-1}$$

Or, $\binom{a}{b} + \binom{a}{b+1} = \binom{a+1}{b+1}$

$$
\begin{aligned}
&= \binom{n+2}{1} + \binom{n+2}{2} + \cdots + \binom{n+k-1}{k-1} \\
&= \binom{n+3}{2} + \binom{n+3}{3} + \cdots + \binom{n+k-1}{k-1} \\
&= \dots \\
&= \binom{n+k-1}{k-2} + \binom{n+k-1}{k-1} \\
&= \binom{n+k}{k-1}
\end{aligned}
$$

Par théorème de récurrence, le lemme est prouvé. [5] □

Lemme 1.5.2. *Soit K et ε deux constantes. On suppose qu'on se restreint aux instances où pour tout $i < n$, $v_i \geq \varepsilon$ et où il y a au plus K volumes d'objets différents, autrement dit $|\{v_1, v_2, \dots, v_n\}| \leq K$. Dans ce cas le problème est polynomial.*

Démonstration. Dans un sac, on ne peut pas mettre plus de $M = \lfloor \frac{1}{\varepsilon} \rfloor$ objets, du fait que $V = 1$ et de la borne sur les volumes des objets. Il y a au plus K volumes différents. On peut donc, théoriquement, si on disposait d'un nombre infini d'objets de chaque sorte, fabriquer $R = (K+1)^M$ sacs différents possibles : pour chaque i de 1 à M, on choisit un volume parmi K à mettre en i^{e} position du sac, ou on choisit de ne pas mettre d'objet en position i. Ce nombre est une constante, il ne dépend pas de n ni des valeurs des volumes. Soit $\mathcal{R}$ l'ensemble des sacs possibles que l'on peut former avec nos n objets. On a donc $|\mathcal{R}| \leq R$.

Une solution réalisable est une suite d'au plus n sacs. On veut donc choisir entre 1 et n fois de suite un sac parmi les (au plus R) sacs de $\mathcal{R}$. On peut voir les choses autrement. Supposons que les sacs de $\mathcal{R}$ soient numérotés de 1 à $|\mathcal{R}| \leq R$. Alors, pour former une solution réalisable, il faut placer i sacs, avec i entre 1 et n, et inscrire sur chacun des sacs le numéro d'un sac de $\mathcal{R}$. Le nombre de solutions réalisables, c'est le nombre de manière *triées* d'inscrire ces chiffres sur les sacs, autrement dit $f(i, |\mathcal{R}|)$.

D'après le lemme 1.5.1, on peut former au plus $\sum_{i=1}^{n} \binom{i+R-1}{R-1} \leq \binom{n+R}{R}$ séquences de sacs possibles avec nos n objets. Clairement toutes les séquences ne sont pas valides (il peut exister des séquences qui n'utilisent pas tous les objets par exemple).

Il suffit donc d'énumérer ces séquences et de vérifier, pour chaque séquence, si cette séquence est une solution réalisable et de garder la meilleure. Le temps de calcul est polynomial car $\binom{n+R}{R} = O((n+R)^R)$ qui est polynomial puisque R est une constante. □

Petite remarque sur la preuve précédente : en toute logique, il existe bien $O(\binom{n}{M})$ types de sacs possibles, ce qui n'est pas une constante. Si on peut borner ce nombre par la constante R, c'est parce que certains objets sont indistinguables, du fait qu'ils ont la même taille. Ainsi si vous avez 6 objets, 3 de volume 18 et 3 de volume 13 et que vos sacs peuvent contenir au plus 3 objets, alors vous pouvez former les sacs suivants : (18, 18, 18), (18, 18, 13), (18, 13, 13), (13, 13, 13), (18, 18), (18, 13), (13, 13), (18), (13) et le sac vide. On a donc 10 sacs possibles, inférieur à $(K+1)^M = 3^3$. Si on avait considéré les objets comme distinguables, et donc x_1, x_2, x_3, x_4, x_5 et x_6 de volumes respectifs 18, 18, 18, 13, 13 et 13, alors on aurait eu beaucoup plus de sacs : $(x_1, x_2, x_3), (x_1, x_2, x_4), (x_1, x_2, x_5), (x_1, x_2, x_6), \dots$. En l'occurence, on a énuméré plusieurs fois le sac (18, 18, 13), avec les sacs $(x_1, x_2, x_4), (x_1, x_2, x_5)$ et (x_1, x_2, x_6). Donc, dans la preuve précédente, lorsqu'on énumère toutes les solutions possibles, il faut bien garder en tête qu'il ne faut pas énumérer deux solutions possibles qui sont indistinguables sous peine de faire exploser le temps de calcul. Cependant, même avec cette remarque, R reste un nombre très grand.

Seconde remarque, au lieu de $R = (K+1)^M$, on aurait pu montrer que R est de l'ordre de $\binom{M+K}{K}$ en utilisant le lemme 1.5.1. Mais ça n'aurait pas eu d'impact sur le résultat.

Maintenant qu'on dispose d'un cas simple, passons à des instances plus complexes : on supprime la contrainte liée au nombre K, les volumes sont potentiellement tous différents. On va donc se ramener au cas où on a K volumes différents.

5. NDLA : Je suis intimement convaincu qu'il est possible de faire cette preuve sans récurrence, en disant que, pour numéroter les boules, il suffit de sélectionner $k-1$ quelque chose parmi $n+k-1$ quelque chose ; et je pense que ce $k-1$ quelque chose sont les $k-1$ séparations entre les k groupes de boules de même numéro, mais je n'arrive pas à mettre la main sur ce $n+k-1$ quelque chose.

Algorithme 9 Vega et Lueker (partie 1)

Entrées: Une instance de Bin Packing où chaque objet a un volume $v_i \geq \varepsilon$.

1: Trier les objets par volume croissant.
2: Former $\lceil \frac{1}{\varepsilon^2} \rceil$ groupes d'objets consécutifs. Tous les groupes contiennent $\lfloor n\varepsilon^2 \rfloor$ objets sauf éventuellement le premier qui en contient moins.
3: Remplacer le volume de chaque objet par le volume de l'objet le plus gros de son groupe
4: Trouver une solution optimale
5: Transposer cette solution dans l'instance originale

Cet algorithme est polynomial puisqu'il se ramène, en temps polynomial au cas du lemme 1.5.2 avec $K = \lceil \frac{1}{\varepsilon^2} \rceil$. Autre information importante : la dernière ligne est faisable puisqu'on ne fait que diminuer les volumes des objets avec lesquels on a travaillé à la ligne 4. Il n'y a donc pas de risque que la solution optimale de la ligne 4 ne soit pas réalisable dans l'instance originale. Il nous reste à prouver le rapport d'approximation de cet algorithme.

Lemme 1.5.3. *Soit S^* une solution optimale et S la solution renvoyée par l'algorithme 9, alors* $\max(S) \leq (1+\varepsilon)\max(S^*)$.

Démonstration. On va utiliser ici deux bornes inférieures de $\max(S^*)$. La première vient de la borne inférieure des volumes : pour tout i, $v_i \geq \varepsilon$, donc on ne peut mettre plus de $\frac{1}{\varepsilon}$ objets dans un sac, donc il faut au moins $n\varepsilon$ sacs pour ranger tous les objets. Donc $n\varepsilon \leq \max(S^*)$.

La seconde borne est un peu plus subtile, et se base sur les groupes d'objets construits dans l'algorithme. Pour une lecture plus simple, on note $p = \lceil \frac{1}{\varepsilon^2} \rceil$ et $R_1, R_2, \ldots, R_p$ les groupes formés à la ligne 2. Sans perte de généralité, on suppose que les objets sont numérotés par volume croissant.

Pour tout $j \in [\![1;n]\!]$ et $i \in [\![1;p]\!]$ tel que $j \in R_i$, on note $v_j^\downarrow = \min\{v_j, j \in R_i\}$ et $v_j^\uparrow = \max\{v_j, j \in R_i\}$. On considère 3 instances et 3 solutions différentes :

— L'instance $\mathcal{I}^\uparrow$ où on remplace chaque volume v_j par $v_j^\uparrow$ et $S_1^\uparrow, S_2^\uparrow, \ldots, S_n^\uparrow$ la solution optimale associée, calculée à la ligne 4 ;
— l'instance $\mathcal{I}^\downarrow$ où on remplace chaque volume v_j par $v_j^\downarrow$ et $S_1^\downarrow, S_2^\downarrow, \ldots, S_n^\downarrow$ une solution optimale associée ;
— et l'instance $\mathcal{I}$ original et $S_1^*, S_2^*, \ldots, S_n^*$ une solution optimale associée.

Pour tout $j \in [\![1;n]\!]$, on a $v_j^\downarrow \leq v_j$. Donc une solution réalisable de $\mathcal{I}$ l'est aussi pour $\mathcal{I}^\downarrow$. Donc, par optimalité de $S^\downarrow$, on vérifie $\max(S^\downarrow) \leq \max(S^*)$. On peut noter que, comme expliqué au dessus de ce lemme, le même argument suffit à justifier que l'algorithme renvoie une solution réalisable.

On va montrer qu'il existe une solution réalisable de $\mathcal{I}^\uparrow$ dont la valeur est $(1+\varepsilon)$ approchée par rapport à S^*. On peut noter que pour tout $i \in [\![1;p-1]\!]$, $\max\{v_j, j \in R_i\} \leq \min\{v_j, j \in R_{i+1}\}$ car les objets sont triés avant d'être groupés. On construit une solution de $\mathcal{I}^\uparrow$ avec l'algorithme suivant : remplacer dans les sacs de la solution $S^\downarrow$ les objets du groupe R_{i+1} par ceux du groupe R_i et mettre chaque objet du groupe R_p dans un sac tout seul. Par construction, on a $|R_1| \leq |R_2| = |R_3| = \cdots = |R_p|$, donc quelques objets de R_2 ne seront pas remplacés, ils sont juste supprimés.

1: **Pour** $i \in [\![1;p-1]\!]$ **Faire**
2: **Pour** $j \in R_i$ **Faire**
3: $S_j \leftarrow S^\downarrow_{j+\lfloor n\varepsilon^2 \rfloor}$
4: **Pour** $j \in R_p$ **Faire**
5: $S_j \leftarrow \max(S) + 1$

Cette solution est réalisable dans $\mathcal{I}^\uparrow$ puisque les sacs contenant les objets de R_p ne contiennent qu'un objet et puisque les autres respectaient la contrainte de volume dans l'instance $\mathcal{I}^\downarrow$ et qu'on a remplacé chaque objet par un objet de volume plus petit ($\min\{v_j, j \in R_{i+1}\}$ par $\max\{v_j, j \in R_i\}$).

$$\begin{aligned}\max(S) &= \max(S^\downarrow) + |R_p| \\ &\leq \max(S^\downarrow) + \lfloor n\varepsilon^2 \rfloor\end{aligned}$$

Or $\max(S^\downarrow) \leq \max(S^*)$

$$\begin{aligned}&\leq \max(S^*) + \lfloor n\varepsilon^2 \rfloor \\ &\leq \max(S^*) + n\varepsilon^2\end{aligned}$$

Or $n\varepsilon \leq \max(S^*)$

$$\leq (1+\varepsilon)\cdot\max(S^*)$$

□

Enfin, passons au cas général où ni la contrainte sur ε ni la contrainte sur K n'existe.

Algorithme 10 Vega et Lueker (1975)

Entrées: Un réel $\varepsilon > 0$

1: Supprimer tous les objets de taille strictement inférieure à ε
2: Trouver une solution S' avec l'algorithme 9
3: Réinsérer les éléments supprimés dans cette solution avec l'algorithme FirstFit

Encore une fois la polynomialité de cet algorithme et la faisabilité des solutions renvoyées ne pose pas de problème.

Théorème 1.5.3. *Soit S^* une solution optimale et S la solution renvoyée par cet algorithme, alors* $\max(S) \leq (1+2\varepsilon)\max(S^*) + 1$.

Démonstration. Supposons dans un premier temps que $\varepsilon \geq \frac{1}{2}$. Alors, à la ligne 2, lorsqu'on applique l'algorithme 9, on a que des objets de volume supérieur à $\frac{1}{2}$, donc tous les objets vont dans des sacs différents et toute solution réalisable est optimale. Cette solution est complétée par une solution de l'algorithme FirstFit à la ligne 3 qui remonte le rapport d'approximation à 2 puisque, comme dans la preuve du théorème 1.5.1 tous les sacs seront remplis au moins à moitié, sauf éventuellement le dernier. Puique $2 \leq 1 + 2\varepsilon$, le rapport annoncé est bien respecté.

On suppose maintenant que $\varepsilon < \frac{1}{2}$. Soit $\mathcal{I}$ l'instance originale et $\mathcal{I}'$ l'instance créée à la ligne 1. D'après le lemme 1.5.3, la solution S' de l'instance $\mathcal{I}'$ créée à la ligne 2 est une $(1+\varepsilon)$-approximation. Ainsi, si $S^{*\prime}$ est une solution optimale de $\mathcal{I}'$ alors $\max(S') \leq (1+\varepsilon)\max(S^{*\prime})$. De plus, $\max(S^{*\prime}) \leq \max(S^*)$ puisque la solution S^* privée des éléments de taille supérieur à ε est une solution de $\mathcal{I}'$.

Tous les éléments restant sont placés avec l'algorithme First Fit. Si aucun sac n'est ouvert, alors $\max(S) = \max(S') \leq (1+\varepsilon)\max(S^{*\prime}) \leq (1+\varepsilon)\max(S^*) \leq (1+2\varepsilon)\max(S^*) + 1$. Sinon, de manière similaire à la preuve du Théorème 1.5.1, on montre que tous les sacs sauf 1 sont remplis avec un volume strictement plus haut que $1-\varepsilon$ (on rappelle que le volume V a été ramené à 1). Donc, $\sum_{i=1}^{n} v_i > (\max(S) - 1)(1-\varepsilon)$.

Ainsi $\max(S) < \frac{\max(S^*)}{1-\varepsilon} + 1$. Or, $\varepsilon < \frac{1}{2}$, donc $\frac{1}{1-\varepsilon} \leq 1 + 2\varepsilon$.

Petite remarque, on ne peut pas déduire de $\max(S) < \frac{\max(S^*)}{1-\varepsilon} + 1$ que $\max(S) \leq \frac{\max(S^*)}{1-\varepsilon}$ car $\frac{\max(S^*)}{1-\varepsilon} + 1$ n'est pas nécessairement un entier. □

Une chose intéressante à voir avec cet algorithme ; c'est le facteur *+ 1* à la fin du rapport d'approximation. Ce facteur ajoute un terme constant dans le décalage entre la solution approchée et la solution optimale. Il devient toutefois négligeable quand la valeur optimale augmente. Donc plus l'instance a une solution optimale avec beaucoup de sacs et plus l'approximation se rapproche d'une approximation de rapport $(1 + 2\varepsilon)$.

Une seconde remarque est *que se passe-t-il si ε est trop grand ?*. Dans ce cas, à la ligne 1, on risque de supprimer tous les objets, on se retrouve alors avec un algorithme FirstFit. Il est intéressant de voir que le rapport d'approximation montre que FirstFit est meilleur quand tous les objets sont petits. Par exemple, si tous les objets sont de volume inférieur à 0.1, alors avec $\varepsilon = 0.11$, l'algorithme n'applique qu'un FirstFit, mais conserve un rapport asymptotique d'approximation de 1.22, soit mieux que le rapport 2 prouvé plus haut. On peut noter aussi que, si $\varepsilon = \frac{1}{2}$ alors le rapport devient égal à 2, mais que si $\varepsilon > \frac{1}{2}$, on passe au dessus de 2. C'est étrange puisque si $\varepsilon = 1$ par exemple, on supprime tous les objets à la ligne 1 et on les place tous avec First Fit donc on a un rapport d'approximation de 2, mais le rapport $(1 + 2\varepsilon)$ nous dit qu'on a une approximation de l'ordre de 3. Ce cas est en fait bien géré par la preuve (mais n'est pas annoncé dans le théorème) : au delà de $\varepsilon > \frac{1}{2}$, le rapport stagne à 2.

1.6 Problèmes de coloration

Coloration d'un graphe planaire

Problème : Problème de la coloration d'un graphe planaire (Planar coloring)

Instance :
— Un graphe planaire non orienté $G = (V, E)$

Solution réalisable : Une coloration du graphe, c'est-à-dire des entiers $C = \{c_v \in \mathbb{N} | v \in V\}$ tels que pour toute arête (u, v) de E, $c_u \neq c_v$.

Optimisation : Minimiser le nombre de couleurs utilisées, c'est à dire $\max(C)$.

Le problème de la coloration d'un graphe planaire est NP-Difficile. En particulier, savoir s'il existe une coloration d'un graphe planaire avec 3 couleurs est un problème NP-Complet.

Cependant, il existe pour ce problème une approximation un peu différente des autres, basée sur le théorème suivant :

Théorème 1.6.1 ([6]). *Tout graphe planaire peut être colorié avec 4 couleurs.*

1: Si le graphe est déconnecté, **Renvoyer** 1.
2: Si le graphe est biparti, **Renvoyer** 2.
3: **Renvoyer** 4.

Théorème 1.6.2. *Soit C^* une solution optimale et M la valeur renvoyée par cet algorithme. Alors $M \leq \max(C^*) + 1$.*

Démonstration. Un graphe est déconnecté si et seulement si on peut le colorier avec 1 couleur. Un graphe est biparti si et seulement si on peut le colorier avec 2 couleurs. Sinon, le graphe est 4 coloriable. Donc $\max(C^*) = 3$ ou 4. Donc $M = 4 \leq \max(C^*) + 1$. □

Cet algorithme quoi qu'un peu trivial (une fois le théorème des 4 couleurs démontré, ce qui n'est pas rien), montre qu'il existe un catégorie d'approximations qui n'avait pas été envisagée jusqu'à présent : des approximations dont le rapport n'est pas multiplicatif, mais additif. En réalité, on peut rapprocher ces approximations de l'approximation de Vega et Lueker du problème de Bin Packing et son fameur facteur *+1* dont on a déjà parlé.

On parle alors d'un *algorithme d'approximation de rapport absolu 1.*

Coloration des arêtes

On va démontrer un résultat très similaire pour le problème de coloration des arêtes.

Problème : Problème de la coloration des arêtes (Minimum Edge coloring)

Instance :
— Un graphe non orienté $G = (V, E)$

Solution réalisable : Une coloration des arêtes du graphe, c'est-à-dire des entiers $C = \{c_e \in \mathbb{N} | e \in E\}$ tels que pour tout couples d'arêtes e et f ayant une extrémité commune, $c_e \neq c_f$.

Optimisation : Minimiser le nombre de couleurs utilisées, c'est à dire $\max(C)$.

Lemme 1.6.1. *Toute coloration des arêtes d'un graphe possède au moins Δ couleurs où Δ est le degré maximum du graphe.*

Démonstration. Soit v un nœud de degré Δ. Alors toutes ses arêtes incidentes doivent avoir une couleur distincte dans une coloration réalisable. Donc toute coloration possède au moinds Δ couleurs. □

6. https://fr.wikipedia.org/wiki/Théorème_des_quatre_couleurs

Théorème 1.6.3 ([7]). *Dans toute graphe de degré maximum Δ, on peut trouver une coloration des arêtes avec $\Delta + 1$ couleurs en temps polynomial.*

Comme précédemment, on peut démontrer le théorème suivant à l'aide des résultats précédents.

Théorème 1.6.4. *Il existe un algorithme d'approximation de rapport absolu 1 pour le problème de coloration des arêtes.*

1.7 Conclusion

Dans ce chapitre, nous avons introduit l'approximabilité polynomiale au travers d'exemples. Nous avons vu qu'il existait de nombreux types d'approximabilité : des problèmes approximables avec un rapport constant, des problèmes non approximables, des problèmes approximables avec un rapport aussi proche de 1 qu'on le souhaite, des problèmes approximables avec un rapport non constant, des problèmes approximables avec un rapport différentiel absolu et des problèmes approximables dont le rapport s'améliore asymptotiquement quand la solution optimale tend vers l'infini. Il faut bien sûr garder en tête que l'espace des possibles est infini et qu'il existe d'autres types d'approximations qui pourront ou non être abordés dans ce cours, par exemple les approximations probabilistes, les approximations en temps exponentielles, les approximations paramétrées, ...

À partir de toutes ces catégories, nous allons, dans le chapitre suivant, définir des classes de problèmes d'optimisation, comme il existe des classes de problèmes de décision.

7. https://fr.wikipedia.org/wiki/Théorème_de_Vizing

Chapitre 2

Quelques classes d'approximabilité

Dans ce chapitre, nous allons formaliser les différentes approximations que nous avons introduites dans le chapitre précédent et classer les problèmes dans différents ensembles selon la possibilité de les approcher efficacement ou non.

2.1 Les problèmes d'optimisation, PO et NPO

Avant de formaliser les classes d'approximabilité, il faut formaliser la notion de problème d'optimisation. Un tel problème est, à l'image d'un problème de décision, défini par ses instances et la réponse que l'on attend d'un ordinateur lorsqu'on lui donne une telle instance. Si pour un problème de décision, la réponse est oui ou non, pour un problème d'optimisation, la réponse est plus complexe puisqu'il s'agit de trouver une solution optimale et sa valeur.

Un problème d'optimisation Π se défini à l'aide de 4 éléments $\mathcal{I}, \mathcal{S}, \mathcal{M}$ et $\mathcal{O}$ pour *Instances, Solutions réalisables, Mesures, Objectif*. Les *instances* sont, à priori, constituées d'objets mathématiques représentables dans la mémoire d'un ordinateur (ou plus généralement d'une machine de Turing) : graphes, entiers, listes, matrices, ... Les *solutions réalisables* diffèrent pour chaque instance, elles dépendent de l'entrée. Il peut y en avoir un nombre conséquent. À priori, chaque solution est également représentable dans un ordinateur. La *mesure* détermine pour chaque solution la valeur de celle-ci et cette valeur est, à priori, facile à calculer. Et l'*objectif* détermine si on cherche la solution qui minimise ou qui maximise la mesure.

Plus formellement :

— $\mathcal{I}$ est un ensemble d'instances.

— $\mathcal{S}$ est une fonction qui, à chaque instance $x \in \mathcal{I}$, associe un ensemble de solutions réalisables de x.

— $\mathcal{M}$ est une fonction qui, à chaque instance $x \in \mathcal{I}$ et à chaque solution réalisable $y \in \mathcal{S}(x)$, associe un entier strictement positif [1]

— $\mathcal{O}$ est soit min, soit max.

Exemple 1. Pour le problème de Voyageur de commerce, $\mathcal{I}$ est l'ensemble des graphes G non orientés dont les arêtes sont pondérées positivement par une fonction ω. Pour $(G, \omega) \in \mathcal{I}$, $\mathcal{S}(G, \omega)$ est l'ensemble des chaînes hamiltoniennes de (G, ω). Pour $(G, \omega) \in \mathcal{I}$ et $C \in \mathcal{S}(G, \omega)$, $\mathcal{M}((G, \omega), C) = \sum_{e \in C} \omega(e)$. Enfin $\mathcal{O} = \min$.

Exemple 2. Pour le problème de sac à dos, $\mathcal{I}$ est l'ensemble des listes de $2n + 1$ entiers positifs $x = (V, v_1, v_2, \ldots, v_n, u_1, u_2, \ldots, u_n)$. Pour $x \in \mathcal{I}$, $\mathcal{S}(x)$ est l'ensemble des sous-ensembles I de $[\![1; n]\!]$ tels que $\sum_{i \in I} v_i \leq V$. Pour $x \in \mathcal{I}$ et $I \in \mathcal{S}(x)$, $\mathcal{M}(x, I) = \sum_{i \in I} u_i$. Enfin $\mathcal{O} = \max$.

Définition 2 (Valeur et solutions optimales)**.** Soit $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O}\}$ un problème d'optimisation, on définit $\mathcal{M}^*(x) = \min_{y \in \mathcal{S}(x)} \mathcal{M}(x, y)$ si $\mathcal{O} = \min$ et $\mathcal{M}^*(x) = \max_{y \in \mathcal{S}(x)} \mathcal{M}(x, y)$ sinon. On note $\mathcal{S}^*(x) = \{y \in \mathcal{S}(x) | \mathcal{M}(x, y) = \mathcal{M}^*(x)\}$.

Définition 3. Soit $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O}\}$ un problème d'optimisation, on définit 3 problèmes à résoudre :

— *Problème de décision* Π_D, soit $x \in \mathcal{I}$ et $K \in \mathbb{N}$, si $\mathcal{O} = \min$, est-ce que $\mathcal{M}^*(x) \leq K$?, si $\mathcal{O} = \max$, est-ce que $\mathcal{M}^*(x) \geq K$?

— *Problème d'évaluation* Π_E, soit $x \in \mathcal{I}$, trouver $\mathcal{M}^*(x)$.

— *Problème de construction* Π_C, soit $x \in \mathcal{I}$, trouver $\mathcal{M}^*(x)$ et une solution $y^* \in \mathcal{S}^*(x)$.

Si on ne s'intéresse qu'à la complexité algorithmique, on utilise généralement Π_D pour montrer que Π est NP-Difficile. En approximabilité, on s'intéressera plus aux problèmes d'évaluation et de construction.

1. On le verra plus tard, mais autoriser des mesures nulles peut poser des problèmes.

Théorème 2.1.1. $\Pi_D \leq_T \Pi_E \leq_T \Pi_C$

Remarque 2. La notation $\Pi_1 \leq_T \Pi_2$ est ici la *réduction polynomiale de Turing* qui signifie "Si vous avez un algorithme polynomial $\mathcal{A}$ pour résoudre le problème Π_2, il existe une algorithme polynomial qui résout le problème Π_1 (en utilisant un nombre polynomial de fois $\mathcal{A}$).

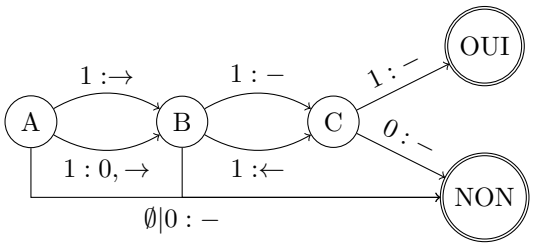
Démonstration. Si vous savez résoudre le problème de construction, le problème d'évaluation est résolu par définition. Si vous savez résoudre le problème d'évaluation, le problème de décision peut être résolu en comparant $\mathcal{M}^*(x)$ avec K. □

Les premières classes que l'on va exhiber dans ce chapitre sont les pendants de P et NP du point de vue de l'optimisation, appelés honorablement PO et NPO. On pourrait directement définir PO comme l'ensemble des problèmes d'optimisation dont on peut résoudre le problème de construction en temps polynomial, mais il y a une petite subtilité nécessaire à la suite qui nous oblige à définir NPO en premier.

On pourrait définir un problème de NPO comme un problème dont le problème de décision est dans NP, mais ce serait oublier que le problème de construction et d'évaluation sont plus difficiles que le problème de décision et que celui-ci ne peut donc refléter à lui seul la complexité de son problème d'optimisation. On doit donc définir NPO comme l'ensemble des problèmes dont on peut résoudre le problème de construction avec une machine de Turing non déterministe en temps polynomial. Que signifie résoudre dans ce contexte ?

Une machine de Turing non déterministe possède, comme toute machine de Turing, un objectif : rejoindre un état terminal. Mais puisqu'elle est non déterministe, il existe possiblement plusieurs manières de rejoindre un état terminal. Une bonne manière de résoudre un problème de décision avec une machine de Turing consiste à créer 2 états terminaux, l'état acceptant et l'état refusant, et selon l'état sur lequel s'arrête la machine, elle accepte ou refuse [2]. Donc la machine répond OUI ou NON. Et comme expliqué plus haut, puisqu'elle est non déterministe, il peut exister plusieurs exécutions qui mènent à OUI et plusieurs qui mènent à NON. Par exemple dans la machine ci-dessous, si la machine a en entrée un mot dont l'un de deux premiers caractères est 0 ou blanc ($\emptyset$), alors elle finit dans l'état NON. Si elle voit un 1 puis un autre 1, alors il y a 2 choix possibles pour aller de A vers B ou de B vers C. Voici les exécutions possibles :

— La tête va à droite, puis ne bouge plus. On finit dans l'état OUI.
— La tête va à droite, puis à gauche. On finit dans l'état OUI.
— La tête écrit un 0 puis va à droite, puis ne bouge plus. On finit dans l'état OUI.
— La tête écrit un 0 puis va à droite, puis à gauche. On finit dans l'état NON.

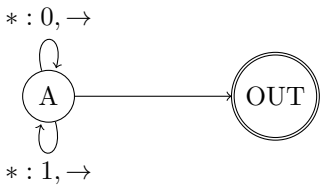


La machine résout un problème de décision Π si,
— quand elle reçoit en entrée une instance positive de Π, il existe une exécution qui mène à l'état OUI
— quand elle reçoit en entrée une instance négative de Π, toute une exécution mène à l'état NON

Supposons ci-dessus que le problème que cherche à résoudre la machine soit : est-ce que le mot en entrée commence par "11" ? Alors elle résout bien le problème au sens de la définition ci-dessus. Essayons de transposer cette définition au problème de construction. On obtiendrait "la machine résout un problème d'optimisation Π si, pour toute instance $x \in \mathcal{I}$, il existe une exécution qui mène à un état terminal et qui écrit sur la bande une solution $y \in \mathcal{S}^*(x)$ et sa valeur $\mathcal{M}^*(x)$". Cette définition peut sembler convaincante, on imagine fort bien une machine de Turing qui résout le problème de l'arbre de Steiner sous la forme : "Ecrire un sous-arbre du graphe dans la bande, calculer son poids et rejoindre un état terminal si l'arbre couvre tous les terminaux de l'instance".

Sauf que cette définition n'est pas bonne, en effet, outre le fait qu'on écarte la possibilité qu'il n'y ait pas de solution [3] la machine suivante, où $*$ signifie "quel que soit le caractère", résout tous les problèmes d'optimisation selon la définition ci-dessus.

2. Une autre manière consiste à ne pas mettre d'état refusant, en considérant que si la machine ne s'arrête pas alors elle refuse.
3. Ce problème n'en est pas vraiment un en soit. On peut rajouter un état refusant spécifiquement pour ce cas.



Pour tout mot, il existe effectivement, pour cette machine, une exécution qui écrit ce mot sur la bande, y compris si le mot est la solution optimale recherchée. On remarque donc que le non déterminisme n'est pas fait pour manipuler des solutions plus complexes qu'un nombre binaire. Voilà pourquoi la définition de NPO est un peu plus subtile et ne fait pas intervenir, à première vue, le non-déterminisme.

Définition 4 (Classe NPO)**.** Soit $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O}\}$ un problème d'optimisation, alors $\Pi \in NPO$ si

1. les instances de $\mathcal{I}$ sont reconnaissables en temps polynomial
2. pour toute instance $x \in \mathcal{I}$, toute solution de $\mathcal{S}(x)$ est de taille polynomiale vis-à-vis de $|x|$
3. pour toute instance x de $\mathcal{I}$ les solutions réalisables de $\mathcal{S}(x)$ sont reconnaissables en temps polynomial vis-à-vis de $|x|$
4. $\mathcal{M}$ est calculable en temps polynomial vis-à-vis de $|x|$.

Pour ceux que ça intéresse, une définition plus formelle serait

1. il existe un encodage des instances de $\mathcal{I}$ en nombre binaire et un algorithme capable de dire, pour tout nombre binaire x, si x est l'encodage d'une instance de $\mathcal{I}$ en temps polynomial vis-à-vis de $|x|$
2. il existe un polynôme q tel que, pour toute instance $x \in \mathcal{I}$, il existe un encodage des solutions de $\mathcal{S}(x)$ en nombre binaire de taille au plus $q(|x|)$ et un algorithme capable de dire, pour tout nombre binaire y de taille au plus $q(|x|)$, si y est l'encodage d'une solution de $\mathcal{S}(x)$ en temps polynomial vis-à-vis de $|x|$
3. $\mathcal{M}$ est calculable en temps polynomial vis-à-vis de $|x|$.

Dans la suite, sauf si cela est explicitement précisé, nous admettrons que les problèmes étudiés sont dans NPO. Sans être totalement absente, la notion d'encodage nous intéressera peu dans la suite, ces considérations sont souvent plus utiles lorsqu'on parle de complexité algorithmique. On se contentera de savoir qu'un entier, un rationnel, un graphe, un ensemble, une liste, ou tout autre objet mathématique/informatique classique est reconnaissable en temps polynomial. Remarques importantes : ce n'est pas le cas des nombres réels. Outre l'encodage, reste la taille des solutions réalisables qui doit être polynomiale en la taille des instances et le temps de calcul de la mesure, mais encore une fois, en ce qui concerne les problèmes étudiés dans ce cours, nous n'aurons pas de soucis à nous faire.

Voici cependant, à titre d'exemple deux preuves montrant respectivement l'appartenance et la non appartenance à NPO :

Exemple 3. Le problème de couverture par ensembles est dans NPO. On peut le définir avec le quadruplet suivant : $\mathcal{I}$ est constitué d'un entier $k \in \mathbb{N}$ et d'un ensemble S de parties de $[\![1; k]\!]$. Les éléments $(x_1, x_2, \ldots, x_k)$ de l'ensemble X sont identifiés par leur indice $(1, 2, \ldots, k)$. Pour $(k, S) \in \mathcal{I}$, $\mathcal{S}(k, S)$ est l'ensemble des sous-ensembles C de S tels que C couvre tous les entiers de 1 à k. Pour $(k, S) \in \mathcal{I}$ et $C \in \mathcal{S}$, $\mathcal{M}((k, S), C) = |C|$. Enfin $\mathcal{O} = \min$.

Pour montrer que ce problème est dans NPO, il faut définir premièrement un encodage est entrées et des solutions. L'entier k sera codé avec un nombre unaire, plus exactement k 1 suivis de un 0. On code ensuite l'entier $|S|$ déterminant la taille de S de la même façon, avec $|S|$ 1 et un 0. On code ensuite S avec une suite de $|S| \cdot k$ bits. Pour $i \leq |S|$ et $j \leq k$, le $ik + j^{\text{e}}$ bit vaut 1 si le i^{e} ensemble de S contient j et 0 sinon. Par exemple pour l'instance

$$(4, \{\{1, 3\}, \{2, 4\}, \{2, 3\}\})$$

on aurait le nombre binaire suivant

$$11110111010100101010110$$

ou, de manière plus lisible :

$$\begin{gathered} 11110\ (k = 4) \\ 1110\ (|S| = 3) \\ 1010\ (S_1 = \{1, 3\}) \\ 0101\ (S_2 = \{2, 4\}) \\ 0110\ (S_3 = \{2, 3\}) \end{gathered}$$

Il existe un algorithme qui, connaissant un nombre binaire, indique si ce nombre correspond à une instance ou non. Cet algorithme commence par lire les premiers bits jusqu'au premier 0 pour définir k, puis les suivants jusqu'au 2e 0 pour définir $|S|$. Si l'algorithme tombe sur un blanc avant un 0 (par exemple pour les nombres 111 ou 110111), il répond NON. Il vérifie enfin que la taille du reste est bien égale à $k|S|$ et répond OUI ou NON en conséquence. Cet algorithme lit chaque bit du nombre une seule fois et effectue un calcul de multiplication, il est donc polynomial en la taille du nombre binaire donné en entrée.

On encodera une solution réalisable C avec un nombre contenant $|S|$ bits, le i^{e} bit vaut 1 si et seulement si le i^{e} ensemble de S est dans C. Par exemple

110

pour la solution $C = \{\{1,3\},\{2,4\}\}$. Ce nombre est de taille inférieure à $|x| \geq k \cdot |S| \geq |S| = |C|$, donc on a bien un polynôme q tel que $|C| \leq q(|x|)$ (le polynôme est l'identité). On peut vérifier qu'il s'agit d'une solution réalisable de (k, S) en vérifiant si le nombre binaire contient bien $|S|$ bits en $O(|S|)$ et, pour tout entier j entre 1 et k qu'il existe dans C un ensemble qui couvre j (en regardant pour chaque i^{e} bit de C qui vaut 1 si le bit $k + 1 + |S| + 1 + i \cdot k + j$ de x vaut 1). Cet algorithme est polynomial, il ne lit les deux nombres binaires que k fois.

Enfin la mesure consiste à calculer le nombre de bits de C qui sont à 1 ce qui se fait également en temps polynomial.

Le problème de couverture par ensemble est donc bien dans NPO.

Exemple 4. Voici un exemple de problème qui n'est pas dans NPO. Le problème de l'arbre de Steiner euclidien consiste, connaissant des points du plan, à relier ces points par des segments dont la longueur totale est minimum. Afin d'éviter des problèmes avec les nombres réels, on va supposer que les coordonnées des points de l'instance et des segments des solutions sont rationnels.

On peut donc coder les points de l'instance avec une suite de nombres rationnels, chaque nombre rationnel étant codé avec une suite de 2 entiers codés en binaire. On peut donc les reconnaître en temps polynomial. Pareil pour les solutions, chaque segment est constitué de 2 points, et chaque point est constitué de 2 rationnels.

Le problème c'est que, pour que le problème soit dans NPO, la taille des solutions réalisables doit être bornée polynomialement par rapport à la taille de l'instance. Or le nombre de solutions réalisables possible est infini. Il existe une infinité d'ensembles de segments possibles reliant les points de l'instance. Il n'est donc pas possible que la taille de ces solutions soient bornées. En effet, si la taille était bornée, leur nombre serait fini (certes possiblement très grand, mais fini). Donc ce problème n'est pas dans NPO.

Le non-déterminisme n'apparaît pas dans la définition de NPO, et on voit mal la relation qui existe entre NP et NPO. Il est pourtant bien présent si on considère le résultat suivant.

Théorème 2.1.2. *Si* Π *est dans NPO alors* Π_D *est dans NP.*

Démonstration. En effet, la machine suivante résout Π_D en temps polynomial. On se place, sans perte de généralité, dans le cas où $\mathcal{O} = \min$.

1. la machine prend en entrée deux nombres binaires x et K
2. vérifier que le nombre binaire x est une instance de Π sinon répondre NON
3. générer de manière non déterministe un nombre binaire y de taille au plus $q(|x|)$
4. vérifier que y est une solution réalisable de x sinon répondre NON
5. calculer $\mathcal{M}(x, y)$
6. vérifier que $\mathcal{M}(x, y) \leq K$ et répondre OUI ou NON en conséquence.

S'il existe une solution réalisable de coût inférieur à K alors cette machine la trouve en temps polynomial (puisque Π est dans NPO, les 5 premières étapes se font en temps polynomial) et répond OUI. Sinon, aucune exécution de cet algorithme ne peut mener à OUI et la machine répond NON en temps polynomial. □

Les problèmes de NPO ont quelques propriétés sympathiques.

Lemme 2.1.1. *Si* $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O}\}$ *est dans NPO, alors, il existe un polynôme* p *tel que, pour toute instance* $x \in \mathcal{I}$ $\mathcal{M}^*(x) \leq 2^{p(|x|)}$.

Démonstration. On sait que $\mathcal{M}$ est calculable en temps polynomial vis-à-vis de $|x|$ puisque Π est dans NPO. Donc, il existe un polynôme p tel que, pour tout instance x et solution réalisable y de x, $\mathcal{M}(x, y)$ se calcule en temps $p(|x|)$. Puisque $\mathcal{M}$ renvoie un entier positif, on sait que ce nombre ne peut être plus grand que $2^{p(|x|)}$ sinon il faudrait plus de temps que $p(|x|)$ pour l'écrire. □

Théorème 2.1.3. *Si* Π *est dans NPO, alors* $\Pi_E \leq_T \Pi_D$.

Démonstration. Supposons qu'on dispose d'un algorithme polynomial $\mathcal{A}$ pour résoudre Π_D. Pour trouver $\mathcal{M}^*(x)$, on peut donc utiliser une dichotomie en vérifiant, à l'aide de l'algorithme $\mathcal{A}$, pour plusieurs valeur de K si $\mathcal{M}^*(x)$ est plus petit ou plus grand que K.

D'après le lemme 2.1.1, il existe un polynôme p tel que $\mathcal{M}^*(x) \leq 2^{p(|x|)}$, il faudra au plus $O(p(|x|))$ utilisations de l'algorithme $\mathcal{A}$ pour trouver $\mathcal{M}^*(x)$. Il existe donc un algorithme qui résout Π_E en temps polynomial en utilisant $\mathcal{A}$ un nombre polynomial de fois. □

Théorème 2.1.4. *Si* Π *est dans NPO et* Π_D *est NP-Complet, alors* $\Pi_C \leq_T \Pi_D$.

Nous ne démontrons pas ce théorème, non pas car la preuve est un peu complexe (on a pas peur de la complexité ici) mais parce qu'elle est complexe et superflue. Nous n'utiliserons pas ce théorème dans le cours et l'énoncer ne ferait que retarder le moment d'introduire les classes d'approximabilité. Il est toutefois intéressant de savoir que le problème de construction est parfois équivalent au problème de décision ; ce qui est très contre-intuitif.

Terminons cette section par la définition de PO.

Définition 5 (Classe PO)**.** Un problème Π de NPO est dans la classe PO s'il existe un algorithme polynomial pour résoudre son problème de construction.

Par définition, on a la propriété suivante.

Théorème 2.1.5. *PO* $\subset$ *NPO*

Théorème 2.1.6. *Si* Π *est dans PO alors* Π_D *est dans P.*

Il est important de remarquer qu'on ne s'intéresse pas dans ce cours aux problèmes qui sortent de NPO. Il existe sans aucun doute des problèmes polynomiaux qui ne sont pas dans NPO. Et donc la contraposée du théorème ci-dessus est fausse. NPO n'est qu'une limite arbitraire des problèmes d'optimisation. Elle est utile, permet de définir de nombreux résultats mais n'est pas parfaite, notamment sur ce point des problèmes polynomiaux.

Enfin, pour bien clarifier la différence entre un problème d'optimisation NP-Difficile et un problème de décision NP-Complet, remarquons qu'on ne peut pas dire qu'un problème d'optimisation est NP-Complet, car ce n'est pas un problème de décision et que NP ne contient que des problèmes de décision. Cependant, on peut parler de difficulté dans le sens où résoudre le problème de construction permet de résoudre un problème de décision NP-Complet.

Définition 6. Un problème d'optimisation Π_1 est NP-Difficile si, pour tout problème Π_2 de NP, $\Pi_2 \leq_T \Pi_{1C}$.

Théorème 2.1.7. *Si* Π_D *est NP-Difficile alors* Π *est NP-Difficile.*

2.2 Approximations polynomiales

Soit une fonction $r : \mathbb{N} \to \mathbb{R}$. On définit un algorithme d'approximation de rapport r, ou r-approximation comme un algorithme qui calcule une solution dont le coût est inférieur/supérieur (selon l'objectif) à r fois le coût d'une solution optimale.

Définition 7 (Approximation pour un problème de minimisation)**.** Soit un problème d'optimisation $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \min\}$, un algorithme d'approximation de rapport r pour Π est un algorithme qui, connaissant une instance $x \in \mathcal{I}$, renvoie une solution $y \in \mathcal{S}(x)$ telle que $\mathcal{M}(x, y) \leq r(|x|) \cdot \mathcal{M}^*(x)$.

Définition 8 (Approximation pour un problème de maximisation)**.** Soit un problème d'optimisation $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \max\}$, un algorithme d'approximation de rapport r pour Π est un algorithme qui, connaissant une instance $x \in \mathcal{I}$, renvoie une solution $y \in \mathcal{S}(x)$ telle que $\mathcal{M}(x, y) \geq r(|x|) \cdot \mathcal{M}^*(x)$.

On parle ici de rapport d'approximation relatif. On supposera dans la suite que $0 < r(|x|) \leq 1$ si Π est un problème de maximisation et $r(|x|) \geq 1$ sinon.

On rappelle qu'on ne s'intéresse qu'aux problèmes d'optimisation dont la mesure est non nulle. Une mesure nulle impliquerait nécessairement que toute approximation soit exacte sur les instances dont la solution optimale est de valeur nulle. De même, on évitera un rapport $r(x) = 0$, notamment pour les problèmes de maximisation.

Remarque 3. Dans la littérature, vous trouverez peut-être une autre formulation pour les problèmes de maximisation sous la forme $\mathcal{M}^*(x) \leq r(|x|) \cdot \mathcal{M}(x, y)$ avec $r(|x|) \geq 1$. Cela revient bien entendu à notre définition après avoir fait passé le rapport d'approximation de l'autre côté, ce qui n'est possible que s'il est non nul.

Remarque 4. Les définitions ne mentionnent pas ce qu'il se passe si l'instance n'a pas de solution réalisable. Dans ce cas, on peut considérer que l'algorithme ne renvoie rien ou renvoie une erreur. Cela signifie en particulier que s'il est NP-Difficile de savoir si une instance possède une solution, comme c'est le cas par exemple pour le problème de voyageur de commerce, il sera impossible de construire un algorithme d'approximation polynomial pour ce problème, sauf si P = NP.

Si r est une constante, on parlera, par abus de langage, d'une r-approximation (on devrait parler d'une $(x \to r(|x|))$-approximation pour s'en tenir à la définition).

Définition 9. Une r-approximation est polynomiale si elle se calcule en temps polynomial.

On peut maintenant définir nos premières classes d'approximabilité.

La classe APX

Définition 10 (Classe APX)**.** Un problème Π de NPO est dans la classe APX si et seulement s'il existe une constante r et une r-approximation polynomiale pour Π.

Par exemple, comme on l'a vu dans le premier chapitre, il existe une 2-approximation polynomiale pour le problème de l'arbre de Steiner et une $\frac{3}{2}$ approximation polynomiale pour le problème de voyageur de commerce métrique. Ces deux problèmes sont donc dans APX. Par définition, APX est inclu dans NPO. On peut prouver que ces classes ne sont pas identiques.

Théorème 2.2.1. *Si P $\neq$ NP alors APX $\subsetneq$ NPO.*

Démonstration. D'après le théorème 1.2.1, si P $\neq$ NP, alors, pour tout $r \in \mathbb{N}$, il n'existe pas de r-approximation pour le problème de voyageur de commerce (non métrique). Donc le voyageur de commerce n'est pas dans APX. Cependant ce problème est dans NPO. □

La classe PTAS

Définition 11 (Classe PTAS)**.** Un problème Π de NPO de minimisation (respectivement de maximisation) est dans la classe PTAS si et seulement si, pour tout $\varepsilon > 0$, il existe une approximation $\mathcal{A}_\varepsilon$ polynomiale de rapport $(1+\varepsilon)$ (respectivement $(1-\varepsilon)$) pour Π.

Par exemple, comme on l'a vu dans le premier chapitre, le problème du sac à dos est dans la classe PTAS. On pourra également dire indifféremment que le problème possède un PTAS ou un schémas d'approximation en temps polynomial. Le temps de calcul de l'algorithme $\mathcal{A}_\varepsilon$ est polynomial vis-à-vis de la taille de l'instance $|x|$ mais rien n'est dit concernant ε. On peut donc avoir une complexité de l'ordre de $O(|x|^{\lceil \frac{1}{\varepsilon} \rceil})$ ou $O(|x|^2 \cdot (\lceil \frac{1}{\varepsilon} \rceil)!)$, qui augmentent exponentiellement avec ε. Ce qui est important ici c'est que, pour un ε fixé, ces deux complexités sont polynomiales en n.

Comme pour APX et NPO, on peut montrer que PTAS est strictement inclu dans APX. Pour cela, on va montrer le résultat intermédiaire suivant sur le problème Bin Packing.

Théorème 2.2.2. *Sauf si P = NP, alors pour tout $\varepsilon > 0$, il n'existe pas de $(\frac{3}{2} - \varepsilon)$-approximation polynomiale pour le problème de Bin Packing.*

Démonstration. On considère le problème de Partition d'entiers : soit $X = (x_1, x_2, \ldots, x_n)$, existe-t-il un sous-ensemble $I \subset [\![1; n]\!]$ tel que $\sum_{i \in I} x_i = \sum_{i \notin I} x_i$. Ce problème est NP-Complet (et fait partie des 21 problèmes de Karp). Soit une instance X du problème de Partition, posons $B = \sum_{i=1}^{n} x_i/2$. Alors, une solution I existe si et seulement si $\sum_{i \in I} x_i = \sum_{i \notin I} x_i = B$.

Soit $\varepsilon > 0$. Supposons qu'il existe une $(\frac{3}{2} - \varepsilon)$-approximation polynomiale pour le problème de Bin Packing. Créons maintenant une instance $\mathcal{J}$ de Bin Packing avec n volumes $v_i = x_i$ et un volume V des sacs égal à B. S'il existe une solution I du problème de partition alors la solution optimale de $\mathcal{J}$ a pour valeur 2. En effet, dans ce cas, on place tous les objets de volume $v_i = x_i, i \in I$ dans le sac 1 et les autres dans le sac 2. Puisque $\sum_{i \in I} x_i = \sum_{i \notin I} x_i = B$ alors les sacs sont remplis avec un volume $V = B$, soit remplis à ras-bord, on a donc bien une solution réalisable de valeur 2. Si on applique $\mathcal{A}$ sur cette instance, l'algorithme nous renvoie donc une solution de valeur au plus $(\frac{3}{2} - \varepsilon) \cdot 2 \leq 3 - 2\varepsilon < 3$. Donc l'algorithme nous renvoie une solution de valeur strictement inférieure à 3 sacs, donc 2 sacs.

Inversement supposons qu'il n'existe pas de solution pour le problème de partition, alors la solution optimale de $\mathcal{J}$ vaut au moins 3. En effet, s'il existe une solution de valeur 2 pour l'instance $\mathcal{J}$, alors, posons I l'ensemble des indices des objets du sac 1. Alors $\sum_{i \in I} x_i = \sum_{i \notin I} x_i = B$. En effet, autrement, $\sum_{i=1}^{n} x_i = \sum_{i \in I} x_i + \sum_{i \notin I} x_i < 2V$. Or

$2V = 2B = \sum_{i=1}^{n} x_i$. On a donc une contradiction. La solution optimale de $\mathcal{J}$ est donc bien au moins 3. Si on applique l'algorithme $\mathcal{A}$ sur cette instance, il nous en renvoie une solution réalisable, et cette solution a pour valeur au moins 3 (plus que la solution optimale).

Donc, en appliquant $\mathcal{A}$ sur $\mathcal{J}$, on peut répondre en temps polynomial au problème de partition : s'il nous renvoie une solution de valeur 2, il existe une partition, sinon il n'en existe pas. Si P $\neq$ NP, un tel algorithme ne peut donc exister. □

Corollaire 2.2.1. *Si P $\neq$ NP, Bin Packing n'est pas dans la classe PTAS.*

Démonstration. D'après le théorème précédent, pour tout $\varepsilon > 0$, il n'existe pas de $(\frac{3}{2} - \varepsilon)$ approximation polynomiale pour le problème Bin Packing si P $\neq$ NP. En particulier, il n'existe pas de 1.1-approximation polynomiale pour Bin Packing. Or s'il existait un PTAS pour ce problème, il existerait, pour tout $\varepsilon' > 0$ une $(1 + \varepsilon)$-approximation pour ce problème, et en particulier une 1.1-approximation polynomiale ce qui est une contradiction. □

Théorème 2.2.3. *PTAS $\subset$ APX et, si P $\neq$ NP, alors PTAS $\subsetneq$ APX.*

Démonstration. Supposons que $\Pi \in$ PTAS. Sans perte de généralité, on suppose que Π est un problème de minimisation. Alors posons $\varepsilon = 1$, il existe par définition de la classe PTAS une $(1 + \varepsilon = 2)$-approximation polynomiale pour Π, donc $\Pi \in$ APX.

Enfin, on a vu a premier chapitre qu'il existe une 2-approximation pour le problème Bin Packing avec l'algorithme First Fit, il est donc dans APX. Et d'après le corollaire précédent, Bin Packing n'est pas dans PTAS sauf si P = NP. □

La classe FPTAS

Définition 12 (Classe FPTAS)**.** Un problème Π de NPO de minimisation (respectivement de maximisation) est dans la classe FPTAS si et seulement si, pour tout rationnel $\varepsilon > 0$, il existe une approximation $\mathcal{A}_\varepsilon$ polynomiale de rapport $(1 + \varepsilon)$ (respectivement $(1 - \varepsilon)$) pour Π et si l'algorithme qui, connaissant une instance x de Π et un rationnel $\varepsilon > 0$, renvoie $\mathcal{A}_\varepsilon(x)$ a une complexité polynomiale en $|x|$ et $\frac{1}{\varepsilon}$.

La différence avec PTAS est ici la dépendance de la complexité en temps avec ε. Remarquons qu'on ne peut pas, sans abus de langage, dire que $\mathcal{A}_\varepsilon$ est polynomial en $|x|$ et $\frac{1}{\varepsilon}$, car, du point de vue de cet algorithme, ε est une constante et non une entrée. Sa complexité ne dépend donc que de $|x|$.

Comme on l'a vu au chapitre 1, la complexité de l'algorithme de Ibarra et Kim pour le problème du sac à dos est bien polynomiale en $\frac{1}{\varepsilon}$, donc ce problème appartient bien à la classe FPTAS. On pourra également dire indifféremment que le problème possède un FPTAS ou un schémas d'approximation en temps complètement polynomial.

On vérifie bien entendu que les problèmes de la classe FPTAS possèdent un PTAS. On montrera que FPTAS est strictement inclu dans PTAS à la fin du chapitre.

Théorème 2.2.4. *Si P $\neq$ NP, alors FPTAS $\subsetneq$ PTAS.*

Démonstration. À la fin du chapitre. □

La classe EPTAS

EPTAS est une classe à mi-chemin entre PTAS et FPTAS.

Définition 13 (Classe EPTAS)**.** Un problème Π de NPO de minimisation (respectivement de maximisation) est dans la classe EPTAS si et seulement si, pour tout rationnel $\varepsilon > 0$, il existe une approximation $\mathcal{A}_\varepsilon$ polynomiale de rapport $(1+\varepsilon)$ (respectivement $(1-\varepsilon)$) pour Π et s'il existe une constante $c \in \mathbb{N}$ et une fonction calculable f tels que l'algorithme qui, connaissant une instance x de Π et un rationnel $\varepsilon > 0$, renvoie $\mathcal{A}_\varepsilon(x)$ a une complexité polynomiale en $O(|x|^c \cdot f(\varepsilon))$.

Ici on autorise les complexités exponentielles en ε, contrairement à la classe FPTAS, mais la part exponentielle n'affecte pas la partie contenant n dans la complexité : cette dernière est toujours le même polynome en n. Par exemple les complexités suivantes permettent de prouver qu'un problème a un EPTAS : $O(n^2 2^{\frac{1}{\varepsilon}})$, $O(n^{19\lceil\frac{1}{\varepsilon}\rceil!})$. Par contre une complexité de l'ordre de $n^{\frac{1}{\varepsilon}}$ n'est pas autorisée.

On vérifie bien entendu que les problèmes de la classe FPTAS possèdent un EPTAS qui eux-même possèdent un PTAS.

La classe r-APX

Soit une fonction $r : \mathbb{N} \to \mathbb{R}$.

Définition 14 (Classe r-APX)**.** Un problème Π de NPO est dans la classe r-APX si et seulement s'il existe une r-approximation polynomiale pour Π.

On sait par exemple d'après le chapitre 1 que le problème de couverture par ensemble appartient à la classe ln-APX.

Théorème 2.2.5. *Sauf si $P \neq NP$, alors pour toute fonction r, r-APX $\subsetneq$ NPO.*

Démonstration. La preuve du théorème 1.2.1 démontrant que TSP n'est pas approximable avec un rapport constant n'utilise pas le fait que r est constant et peut très bien s'adapter à n'importe quelle fonction. □

Rapport absolu : la classe AAPX

Soit une fonction $r : \mathbb{N} \to \mathbb{R}$. On définit un algorithme d'approximation de rapport absolu r comme un algorithme qui calcule une solution dont le coût est inférieur/supérieur (selon l'objectif) à r plus/moins le coût d'une solution optimale.

Définition 15 (Approximation absolue pour un problème de minimisation)**.** Soit un problème d'optimisation $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \min\}$, un algorithme d'approximation de rapport absolu r pour Π est un algorithme qui, connaissant une instance $x \in \mathcal{I}$, renvoie une solution $y \in \mathcal{S}(x)$ telle que $\mathcal{M}(x, y) \leq \mathcal{M}^*(x) + r(|x|)$.

Définition 16 (Approximation pour un problème de maximisation)**.** Soit un problème d'optimisation $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \max\}$, un algorithme d'approximation de rapport absolu r pour Π est un algorithme qui, connaissant une instance $x \in \mathcal{I}$, renvoie une solution $y \in \mathcal{S}(x)$ telle que $\mathcal{M}(x, y) \geq \mathcal{M}^*(x) - r(|x|)$.

Définition 17 (Classe AAPX)**.** Un problème Π de NPO est dans la classe AAPX si et seulement s'il existe une constante r et un algorithme d'approximation polynomial de rapport absolu r pour Π.

On sait par exemple d'après le chapitre 1 que le problème de coloration des arêtes ou le problème de coloration des graphes planaires est dans la classe AAPX.

Remarque 5. Attention, la notion de *rapport différentiel* existe dans la littérature. Bien que le rapport absolu soit une différence, car le rapport borne la différence $|\mathcal{M}(x, y) - \mathcal{M}^*(x)|$, le rapport différentiel n'est pas le même. Ce dernier consiste à minimiser $\frac{\mathcal{M}(x,y)-w(x)}{\mathcal{M}^*(x)-w(x)}$ où $w(x)$ est le poids de la pire solution de l'instance x. La classe relative à ce rapport est appelée DAPX.

On peut de la même manière définir les classes APTAS, AFPTAS, r-AAPX.

Il est difficile pour un problème d'appartenir à la classe AAPX, notamment dans les problèmes ayant des poids, car il est souvent possible d'écarter la solution optimale et la première solution réalisable non optimale en multipliant les poids. On peut par exemple le montrer sur le problème du sac à dos.

Théorème 2.2.6. *Sauf si $P = NP$, le problème du sac à dos n'est pas dans AAPX.*

Démonstration. Supposons qu'il existe une constante r et un algorithme d'approximation polynomial $\mathcal{A}$ de rapport absolu r pour le problème du sac à dos. Soit une instance $x = (V, v_1, v_2, \ldots, v_n, u_1, u_2, \ldots, u_n)$ du problème du sac à dos. Soit I^* une solution optimale de x alors $\mathcal{A}(x)$ renvoie une solution réalisable I telle que $U(I) \geq U(I^*) - r$.

Multiplions maintenant les poids par $r + 1$. On a donc une instance $x' = (V, v_1, v_2, \ldots, v_n, (r + 1)u_1, (r + 1)u_2, \ldots, (r + 1)u_n)$. Les solutions réalisables ne changent pas et puisqu'on a multiplié tous les poids par la même valeur, I^* est toujours une solution optimale de r' et la valeur de cette solution est $U'(I^*) = (r+1)U(I^*)$. Si on applique $\mathcal{A}$ sur l'instance x', il renvoie une solution réalisable I telle que $U'(I) \geq U'(I^*) - r$.

Or I serait solution réalisable de x et de valeur $U(I) = \frac{U'(I)}{r+1} \geq U(I^*) - \frac{r}{r+1}$. Puisque $U(I)$ et $U(I^*)$ sont entiers, alors $U(I) = U(I^*)$. Donc, à l'aide de $\mathcal{A}$, il existe un algorithme polynomial qui résout le problème de sac à dos, ce qui est exclus, sauf si P = NP.

□

En supposant que P ≠ NP, en remarquant que le problème de coloration des graphes planaires n'admets pas de PTAS, ce théorème nous permet de montrer assez simplement que $AAPX \not\subset PTAS$ et que $PTAS \not\subset AAPX$.

Approximation asymptotique : la classe $\mathbf{APX}_\infty$

Soit une fonction $r : \mathbb{N} \to \mathbb{R}$. On définit un algorithme d'approximation de rapport asymptotique r comme un algorithme qui calcule une solution dont le coût est inférieur/supérieur (selon l'objectif) à r fois le coût d'une solution optimale, plus une constante. Cette notion mélange donc rapport relatif et rapport absolu. Comme pour les approximations classiques, on supposera que $0 < r(|x|) \leq 1$ si Π est un problème de maximisation et $r(|x|) \geq 1$ sinon.

Définition 18 (Approximation asymptotique pour un problème de minimisation)**.** Soit un problème d'optimisation $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \min\}$, un algorithme d'approximation asymptotique de rapport r pour Π est un algorithme tel qu'il existe une constante $k \in \mathbb{R}^+$ et qui, connaissant une instance $x \in \mathcal{I}$, renvoie une solution $y \in \mathcal{S}(x)$ telle que $\mathcal{M}(x, y) \leq r(|x|) \cdot \mathcal{M}^*(x) + k$.

Définition 19 (Approximation asymptotique pour un problème de maximisation)**.** Soit un problème d'optimisation $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \max\}$, un algorithme d'approximation asymptotique de rapport r pour Π est un algorithme tel qu'il existe une constante $k \in \mathbb{R}^+$ et qui, connaissant une instance $x \in \mathcal{I}$, renvoie une solution $y \in \mathcal{S}(x)$ telle que $\mathcal{M}(x, y) \geq r(|x|) \cdot \mathcal{M}^*(x) - k$.

Définition 20 (Classe APX_∞)**.** Un problème Π de NPO est dans la classe APX_∞ si et seulement s'il existe une constante $r > 0$ et un algorithme d'approximation asymptotique polynomial de rapport r pour Π.

Définition 21 (Classe PTAS_∞)**.** Un problème Π de NPO de minimisation (respectivement de maximisation) est dans la classe PTAS_∞ si et seulement s'il existe une constante k telle que, pour toute $\varepsilon > 0$, il existe un algorithme $\mathcal{A}_\varepsilon$ qui, connaissant une instance x de Π renvoie une solution réalisable y de x telle que $\mathcal{M}(x, y) \leq (1 + \varepsilon)\mathcal{M}^*(x) + k$ (respectivement $\mathcal{M}(x, y) \geq (1 - \varepsilon)\mathcal{M}^*(x) - k$).

On peut de la même manière définir les classes FPTAS_∞ et r-APX_∞.

On sait par exemple d'après le chapitre 1 que le problème de bin packing est dans la classe PTAS_∞, à l'aide de l'algorithme de Vega et Lueker.

Théorème 2.2.7. *$APX_\infty = APX$*

Démonstration. Soit un problème de minimisation de la classe APX, alors il existe une constante r telle que le problème est r-approximable, autrement dit, il existe un algorithme polynomial qui, pour toute instance x du problème, renvoie une solution de coût inférieur à $r \cdot \mathcal{M}^*(x) = r \cdot \mathcal{M}^*(x) + 0$, donc le problème est dans APX_∞. L'inclusion est également vérifiée pour les problèmes de maximisation.

Soit un problème de minimisation de la classe APX_∞, alors il existe une constante r et une constante k telles qu'il existe un algorithme polynomial qui, pour toute instance x du problème, renvoie une solution de coût inférieur à $r \cdot \mathcal{M}^*(x) + k \leq (r + k) \cdot \mathcal{M}^*(x)$ (ce qui n'est vrai que parce qu'on a imposé que la mesure d'un problème d'optimisation est toujours non nulle), donc le problème est dans APX.

Si on considère un problème de maximisation, alors la même preuve ne fonctionne pas, on aurait $r \cdot \mathcal{M}^*(x) - k \geq (r-k) \cdot \mathcal{M}^*(x)$ avec le risque que $r - k < 0$. Cependant, puisqu'on a supposé que les rapport d'approximations de problèmes de maximisation étaient non nul :

$$\begin{aligned} r \cdot \mathcal{M}^*(x) - k &\leq \mathcal{M}(x, y) \\ \mathcal{M}^*(x) &\leq \frac{1}{r}\mathcal{M}(x, y) + \frac{k}{r} \\ \mathcal{M}^*(x) &\leq \frac{1}{r} \cdot (1 + k)\mathcal{M}(x, y) \\ \frac{r}{1 + k}\mathcal{M}^*(x) &\leq \mathcal{M}(x, y) \end{aligned}$$

□

On va maintenant prouver quelques propriétés sur la classe PTAS_∞. En particulier, on va montrer que le problème de l'arbre de Steiner n'admets pas de PTAS asymptotique. Mais pour cela, on va avoir besoin d'un résultat non prouvé dans ce chapitre, on en reparlera rapidement dans le dernier chapitre.

Théorème 2.2.8. *Sauf si P = NP, le problème de l'arbre de Steiner n'est pas dans la classe PTAS.*

Démonstration. Dans un chapitre ultérieur... □

De manière similaire à la preuve qu'il n'existe pas d'approximation de rapport absolu constant pour le problème du sac à dos, on va montrer qu'il n'existe pas de PTAS asymptotique pour le problème de l'arbre de Steiner.

Théorème 2.2.9. *Sauf si P = NP, le problème de l'arbre de Steiner n'est pas dans la classe $PTAS_\infty$.*

Démonstration. Supposons qu'il existe une constante k telle que, pour toute $\varepsilon > 0$, il existe un algorithme $\mathcal{A}_\varepsilon$ qui, connaissant une instance (G, ω) renvoie une solution réalisable T de (G, ω) telle que $\omega(T) \leq (1+\varepsilon)\omega(T^*)+k$ où T^* est une solution optimale.

On va construire un algorithme $\mathcal{B}_{\varepsilon'}$ qui trouve une solution de poids meilleur que $(1+\varepsilon')\omega(T^*)$. L'existence de $\mathcal{B}_{\varepsilon'}$ montrerait qu'il existe une PTAS pour le problème de l'arbre de Steiner, ce qui est faux, d'après le théorème 2.2.8. Donc on prouverait ainsi que $\mathcal{A}_\varepsilon$ ne peut exister sauf si P = NP et donc on prouverait le théorème.

Soit $\varepsilon' > 0$, posons $p = \frac{2k}{\varepsilon'}$. Il est important de voir ici que ces deux valeurs sont des constantes indépendantes de l'instance. Alors l'algorithme $\mathcal{B}_{\varepsilon'}$ est le suivant.

Algorithme 11 $\mathcal{B}_{\varepsilon'}$

1: Créer une nouvelle instance x^p où tous les poids sont multipliés par p, mais où le graphe ne change pas.
2: Renvoyer $\mathcal{A}_{\varepsilon'/2}(x^p)$

Posons $x = (G, \omega)$ et $x^p = (G, \omega^p)$ la nouvelle instance créée par $\mathcal{B}_{\varepsilon'}$. Puisque le graphe ne change pas, toutes les solutions réalisables de x sont solutions réalisables de x^p et inversement. De plus, puisque les poids sont tous multipliés par la même valeur, T^* est toujours une solution optimale de x^p et la valeur de cette solution optimale est $\omega^*(T^*) = p \cdot \omega(T^*)$.

En utilisant l'algorithme $\mathcal{A}_{\varepsilon'/2}$ sur x^p, on obtient donc une solution T^p telle que $\omega^p(T^p) \leq (1+\frac{\varepsilon'}{2})\cdot p\cdot\omega(T^*)+k$. Dans l'instance originale, T^p est aussi une solution réalisable et son poids est $\omega(T^p) = \frac{\omega^p(T^p)}{p}$.

Ainsi $\omega(T^p) \leq (1+\frac{\varepsilon'}{2})\omega(T^*) + \frac{k}{p} = (1+\frac{\varepsilon'}{2})\omega(T^*) + \frac{\varepsilon'}{2} \leq (1+\frac{\varepsilon'}{2}+\frac{\varepsilon'}{2})\omega(T^*)$.

□

Théorème 2.2.10. *$PTAS \subset PTAS_\infty \subset APX$*
Sauf si P = NP, alors $PTAS \subsetneq PTAS_\infty \subsetneq APX$.

Démonstration. L'inclusion se prouve comme dans le théorème 2.2.7.

Pour les inégalités, on sait déjà que Bin Packing n'admets pas de PTAS, d'après le corollaire 2.2.1 mais admets un PTAS asymptotique, donc PTAS $\subsetneq$ PTAS_∞. Pour la seconde inégalité, on peut s'intéresser au problème de l'arbre de Steiner pour qui il existe une 2 approximation polynomiale. D'après le théorème 2.2.9, il n'existe pas de PTAS asymptotique pour ce problème. □

Des théorèmes similaires sont donnés en exemples dans l'exercice 20 à propos de la classe FPTAS_∞.

2.3 NP-Complétude au sens fort et FPTAS

On a vu dans le chapitre 1, que le problème de Sac à dos admettait un FPTAS, et que, pour construire ce FPTAS, on utilisait le fait que ce problème admet un algorithme pseudo-polynomial. Il serait intéressant de généraliser ce résultat. Malheureusement, il n'existe pas aujourd'hui de résultat permettant de prouver que pseudo-polynomialité implique FPTAS. Cependant, l'inverse peut être (partiellement) prouvé.

Et ce résultat permettra de prouver que FPTAS $\subsetneq$ PTAS.

Redonnons tout d'abord quelques définitions de FPTAS.

Définition 22. Soit un problème d'optimisation Π dont chaque instance x peut être coupée en 2 parties : x_0 est un ensemble d'objets non numériques (graphe, tableau, ensemble, ...) et un ensemble d'entiers $x_1, x_2, \ldots, x_k$. On définit la 3 tailles de x :

- $|x|$ est la taille de son encodage, où tous les entiers sont codés en binaire : $|x_0| + \log(x_1) + \log(x_2) + \cdots + \log(x_n)$.
- $l(x)$ est la taille de x indépendante de la valeur des entiers : $|x_0| + k$
- $\max(x)$ est la taille relative aux entiers : $\max\limits_{i\in[\![1;k]\!]}(x_i)$

Exemple 5. Dans le problème du sac à dos, une instance x s'écrit $(V, v_1, v_2, \ldots, v_n, u_1, u_2, \ldots, u_n)$. Il n'y a donc pas d'objet non numérique : on dispose de $2n+1$ entiers. Ainsi $|x| = \log(V) + \log(v_1) + \log(v_2) + \cdots + \log(v_n) + \log(u_1) + \log(u_2) + \cdots + \log(u_n)$, $l(x) = 2n+1$ et $\max(x) = \max(V, v_1, v_2, \ldots, v_n, u_1, u_2, \ldots, u_n)$.

Exemple 6. Dans le problème de l'arbre de Steiner, une instance x s'écrit $(G = (V, E), \omega)$. Posons $n = |V|$ et $m = |E|$. Il y a un objet non numérique, le graphe G de taille $n+m$, et les poids décrit par m entiers. Alors $|x| = n + m + \sum\limits_{e\in E} \log(\omega(e))$, $l(x) = n + 2m$ et $\max(x) = \max\limits_{e\in E} \omega(e)$.

Il est important de noter qu'un algorithme polynomial vis-à-vis de $|x|$ ou $l(x)$ est polynomial, mais un algorithme polynomial vis-à-vis de $\max(x)$ ne l'est pas.

Définition 23. Un algorithme *pseudo-polynomial* pour résoudre un problème Π s'il existe un algorithme qui résout toute instance x de Π en temps polynomial vis-à-vis de $l(x)$ et de $\max(x)$.

On peut remarquer aussi qu'un algorithme est polynomial vis-à-vis de $l(x)$ et de $\max(x)$ si et seulement s'il est polynomial vis-à-vis de $|x|$ et de $\max(x)$.

Définition 24. Un problème est *NP-Difficile au sens fort* s'il reste NP-Difficile même quand on se restreint aux instances où $\max(x)$ est polynomialement borné vis-à-vis de $l(x)$. Inversement, un problème est *NP-Difficile au sens faible* s'il est NP-Difficile mais qu'il existe un algorithme pseudopolynomial pour le résoudre.

Exemple 7. Le sac à dos est NP-Difficile au sens faible. Le problème de l'arbre de Steiner est NP-Difficile au sens fort puisqu'il reste NP-Difficile même si on se restreint aux instances où tous les poids valent 1 (et donc on a $\max(x) = 1 \leq n + m$).

Théorème 2.3.1. *Si P $\neq$ NP, les problèmes NP-Difficiles au sens fort ne sont pas NP-Difficiles au sens faible et inversement.*

Démonstration. Un problème Π NP-Difficile au sens faible ne peut pas être NP-Difficile au sens fort. En effet, il existe un algorithme pseudopolynomial pour le résoudre, donc un algorithme polynomial en $l(x)$ et $\max(x)$.

Ainsi, il existe un polynôme p et un algorithme $\mathcal{A}$ qui résout Π en temps $O(p(l(x), \max(x)))$. Si on se restreint aux instances où $\max(x)$ est polynomialement borné vis-à-vis de $l(x)$ alors il existe un polynôme q tel que $\max(x) \leq q(l(x))$. Si on applique $\mathcal{A}$ sur ces instances, on a donc un algorithme qui résout ces instances de Π en temps $O(p(l(x), q(l(x))))$. Cet algorithme est donc polynomial vis-à-vis de $l(x)$. Donc si P $\neq$ NP, Π n'est pas NP-Difficile quand on se restreint à ces instances. □

Théorème 2.3.2. *Soit $\Pi = \{\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O}\}$ un problème de la classe FPTAS. On suppose qu'il existe un polynôme p tel que, pour tout instance $x \in \mathcal{I}$, alors $\mathcal{M}^*(x) \leq p(l(x), \max(x))$, alors Π admets un algorithme pseudopolynomial.*

Démonstration. Supposons, sans perte de généralité que Π soit un problème de minimisation. Alors il existe un algorithme $\mathcal{A}$ qui connaissant une instance $x \in \mathcal{I}$ et un rationnel $\varepsilon > 0$, renvoie une solution réalisable de $y \in \mathcal{S}(x)$ en temps polynomial en $|x|$ et $\frac{1}{\varepsilon}$ et telle que $\mathcal{M}(x, y) \leq (1 + \varepsilon)\mathcal{M}^*(x)$.

Posons $\varepsilon(x) = \frac{1}{p(l(x), \max(x))+1}$ et $\mathcal{B}$ l'algorithme qui renvoie $\mathcal{A}(x, \varepsilon(x))$. Cet algorithme renvoie une solution en temps polynomial vis-à-vis de $|x|$ et $p(l(x), \max(x))+1$, donc en temps polynomial vis-à-vis de $l(x)$ et $\max(x)$.

De plus, la solution y renvoyée vérifie $\mathcal{M}(x, y) \leq (1 + \frac{1}{p(l(x), \max(x))+1})\mathcal{M}^*(x) \leq \mathcal{M}^*(x) + \frac{p(l(x), \max(x))}{p(l(x), \max(x))+1}$ par hypothèse sur la borne de $\mathcal{M}^*(x)$. Puisque $\mathcal{M}(x, y)$ et $\mathcal{M}^*(x)$ sont entiers, $\mathcal{M}(x, y) \leq \mathcal{M}^*(x)$.

Donc y est une solution optimale de x, et donc $\mathcal{B}$ résout Π en temps pseudopolynomial. □

Corollaire 2.3.1. *Si Π est NP-Difficile au sens fort et s'il existe un polynôme p tel que, pour tout instance $x \in \mathcal{I}$, alors $\mathcal{M}^*(x) \leq p(l(x), \max(x))$, alors si P $\neq$ NP, Π n'appartient pas à la classe FPTAS.*

Démonstration. D'après le théorème 2.3.2, un tel problème serait NP-Difficile au sens faible. Or, d'après le théorème 2.3.1, si P $\neq$ NP, un problème ne peut être NP-Difficile au sens faible et au sens fort. □

Il ne nous reste plus qu'à montrer que FPTAS $\subsetneq$ PTAS. Pour cela, nous aurons besoin d'un résultat intermédiaire que nous admettrons. Sa démonstration est donnée dans le premier livre indiqué en introduction et donnée en exercice de ce poly.

On considère le problème Maximum Planar Independent Set de trouver un stable de taille maximum dans un graphe planaire.

Théorème 2.3.3. *Le problème Maximum Planar Independent Set est dans la classe PTAS.*

Démonstration. Voir l'exercice 7 □

Théorème 2.3.4. *Le problème Maximum Planar Independent Set est NP-Difficile au sens fort.*

Concernant la NP-Difficulté, nous n'en donnons pas la preuve, cependant, pour la partie "au sens fort", il suffit de remarquer que ce problème ne contient pas d'entier. S'il est NP-Difficile, il l'est donc nécessairement au sens fort.

Corollaire 2.3.2. *Si P $\neq$ NP, alors FPTAS $\subsetneq$ PTAS*

Démonstration. La mesure d'une instance du problème Maximum Planar Independent Set varie entre 1 et la taille du graphe de l'instance. Donc cette solution est toujours de taille plus petite que la taille de l'instance. D'après le théorème 2.3.1, il n'existe pas de FPTAS pour ce problème sauf si P = NP. Ce problème appartenant à la classe PTAS, on en déduit le corollaire. □

On peut noter que les mêmes résultats ne sont pas valables pour FPTAS_∞, comme l'atteste l'exercice 20.

2.4 Conclusion

Toutes les classes que nous avons introduites dans ce chapitre, NPO, APX, PTAS, FPTAS, r-APX, AAPX, APX_∞, PTAS_∞, ..., permettent de classer les problèmes les uns par rapport aux autres. Une certaine hiérarchie existe entre ces classes. Nous introduirons dans un chapitre ultérieur la notion de réduction qui permettra, comme pour les problèmes de décision de parler de complétude.

Avant de continuer, rappelons encore une fois que ces classes ne constituent pas l'ensemble des classes existantes et ne résument pas à elles seule l'ensemble des manières d'étudier et de classer des problèmes d'optimisation. Toutes les démonstrations ont été faites en se basant sur une définition la classe NPO. Cette classe n'inclue pas, par exemple, des problèmes dont les mesures sont rationnelles et, bien qu'il soit souvent possible de modifier ces problèmes pour qu'ils collent à la définition de NPO, cela peut se faire au prix d'un temps que l'on n'est pas prêt à payer. Une autre définition de NPO, aurait pu amener à une légère ou drastique modification des classes de complexité et de leurs relations. Ceci étant dit, la définition de rapport d'approximation relatif ou absolu ne fait pas l'hypothèse que le problème étudié est dans NPO. Il est donc tout à fait possible de parler d'une 2-approximation pour un problème dont les poids sont rationnels, ou pour un problème possédant un nombre infini de solutions. On ne pourra juste pas dire qu'il est dans la classe APX.

Pour finir, un petit coup d'oeil au site `Complexity Zoo` vous permettra de vous faire une idée de l'étendue des classes qui existent pour les problèmes d'informatique.

Dans le chapitre suivant, nous allons quitter le monde des classes d'approximabilités pour revenir à ce qui pourrait sembler plus concret. L'objet de l'étude sera une technique particulière pour construire des approximations polynomiales.

Chapitre 3

Approximation polynomiale et programmation linéaire

Dans ce chapitre, nous allons introduire trois techniques de construction d'algorithmes d'approximation polynomiaux à partir de modèles de programmation linéaire.

3.1 Quelques rappels utiles

On supposera dans la suite que les définitions de programme linéaire, de variable, de contrainte, de fonction objectif, de solution optimales ou réalisables d'un programme linéaire sont connues.

Programmes linéaires et dualité

On s'intéressera dans cette partie à des programmes quelconques. On notera dans la suite n le nombre de variables et m le nombre de contraintes.

Programme linéaire (PP1) : Programme générique

Minimiser	$c \cdot {}^t y$		
s.c.	$A \cdot {}^t y$	$\geq$	b
	y	$\in$	$(\mathbb{R}^+)^n$

Le programme ci-dessus possède un programme appelé dual où, entre autres, les nombres de variables et de contraintes sont inversés.

Programme linéaire (DP2) : Programme générique dual

Maximiser	$b \cdot {}^t z$		
s.c.	${}^t A \cdot {}^t z$	$\leq$	c
	z	$\in$	$(\mathbb{R}^+)^m$

Remarque 6. On écrit, par habitude, que les variables sont réelles et continues. Cependant, en informatique, elles sont bien entendues rationnelles. Puisque le reste du programme est décrit avec des vecteurs b et c et une matrice A à coefficients rationnels, la solution optimale est elle aussi rationnelle, il n'est donc pas très grave de travailler ici avec des réels.

Les théorèmes suivants seront utilisés au cours du chapitre.

Théorème 3.1.1 (Méthode de l'ellipsoïde - `https://fr.wikipedia.org/wiki/Méthode_de_l'ellipsoïde`)**.** *Il existe un algorithme polynomial capable de résoudre un programme linéaire.*

(On rappelle au cas où que l'algorithme du simplexe n'est (à l'heure où est écrit ce cours) pas polynomial.)

Théorème 3.1.2 (Théorème de dualité faible - `https://fr.wikipedia.org/wiki/Optimisation_linéaire#Propriétés_de_dualité`)**.** *Si y est réalisable pour le programme (PP1) et z est réalisable pour (DP2) alors $b \cdot {}^t z \leq c \cdot {}^t y$.*

Théorème 3.1.3 (Théorème de dualité forte - `https://fr.wikipedia.org/wiki/Optimisation_linéaire#Propriétés_de_dualité`)**.** *Les programmes (PP1) et (DP2) ont la même valeur de solution optimale : si y° est optimal pour (PP1) et z° est optimal pour (DP2) alors $b \cdot {}^t z^\circ = c \cdot {}^t y^\circ$.*

Remarque 7. Le symbole $\circ$ est utilisé ici pour les solutions optimales, on réservera le symbole $*$ pour les solutions optimales entières.

Corollaire 3.1.1 (Théorème des écarts complémentaires)**.** *Si y est réalisable pour le programme (PP1) et z est réalisable pour (DP2), alors y et z sont optimales si et seulement si*

- *Pour tout $i \leq n$, $y_i = 0$ ou $\sum_{j=1}^{m} a_{ji} z_j = c_i$*
- *Pour tout $j \leq m$, $z_i = 0$ ou $\sum_{i=1}^{n} a_{ji} y_i = b_j$*

Programmes relâchés

Relâcher (les contraintes d'intégrité d') un programme consiste à remplacer l'ensemble des variables entières par des variables continues dont l'ensemble de définition contient celui des variables entières. $x \in \mathbb{N}$ devient $x \in \mathbb{R}$, $x \in [\![a, b]\!]$ devient $x \in [a, b]$.

Par exemple, si on considère le programme suivant, sa version relâchée est (PP1).

Programme linéaire (PP3) : Programme générique en nombre entier

Minimiser	$c \cdot {}^t y$		
s.c.	$A \cdot {}^t y$	$\geq$	b
	y	$\in$	$(\mathbb{N})^n$

Théorème 3.1.4. *Si y° est optimal pour (PP1) et y^* est optimal pour (PP3) alors $c \cdot {}^t y^\circ \leq c \cdot {}^t y^*$.*

Démonstration. L'ensemble des solutions réalisables de (PP1) inclu l'ensemble des solutions de (PP3). □

On rappelle que les inégalités s'inversent en cas de problème de maximisation.

Pourquoi s'y intéresser ?

Problème : Problème de Programmation Linéaire en Nombre Entier (interger Linear Programming problem en anglais)

Instance :

- Un programme linéaire dont les variables sont entières et bornées

Solution réalisable : Une affectation des variables à une valeur entière.

Optimisation : Minimiser la fonction objective.

Ce problème est un problème d'optimisation de NPO notamment parce que les valeurs des variables sont bornées. Le cas non borné n'est pas dans NPO, même si on peut montrer malgré tout que le problème de décision associé est dans NP.

En plus d'être dans NPO, il est NP-Difficile et permet donc à ce titre de résoudre tous les problèmes de décision NP-Difficile. C'est donc un premier bon point pour la programmation linéaire. Mais il est alors au même niveau que n'importe quel autre problème NP-Difficile. En plus de çà, la programmation linéaire apporte un cadre précis permettant résoudre de nombreux problèmes à l'aide de solveurs. Mais cette seule propriété n'est pas suffisante pour qu'on s'intéresse à la programmation linéaire dans ce cours. En effet, le problème 3-SAT offre un cadre tout à fait similaire qui permet de résoudre les problèmes avec des SAT Solvers, cependant, un tel modèle n'est pas exploitable pour produire des approximations polynomiales.

Tout l'avantage des programmes linéaires réside dans les théorèmes précédents, qui offrent naturellement des bornes avec lesquelles on peut travailler pour construire des algorithmes d'approximations. En utilisant la technique de la borne inférieure vue dans le premier chapitre, le rapport d'approximation se déduira uniquement

de l'écart entre les solutions optimales et optimales relâchées, et donc de la qualité des modèles de programmation linéaire utilisés pour résoudre le problème.

On considérera dans toute la suite un problème de **minimisation** $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O})$. On notera, comme dans le chapitre précédent, $\mathcal{M}^*(x)$ le poids d'une solution optimale y^* d'une instance $x \in \mathcal{I}$ et $\mathcal{M}^\circ(x)$ le poids d'une solution optimale y° d'un programme linéaire modélisant Π.

3.2 Techniques d'arrondi

La technique d'arrondi est sans doute la technique la plus simple à décrire pour produire un algorithme d'approximation avec un programme linéaire, mais également la technique la plus complexe pour utiliser un rapport d'approximation de bonne qualité.

Elle consiste à calculer y° et à en déduire une solution y réalisable en arrondissant les coefficients de y°. Le rapport d'approximation se déduit de l'écart entre $\mathcal{M}^\circ$ et le poids de la pire solution y que l'on peut renvoyer par cette méthode.

Exemple : problème de couverture par ensemble

Prenons pour exemple le problème de couverture par ensembles. Le programme suivant est une manière classique de modéliser ce problème, connaissant une instance $x = (X, \mathcal{S}, \omega)$.

Programme linéaire (SCP4) : Programme pour résoudre la couverture par ensembles

Minimiser	$\sum_{S \in \mathcal{S}} \omega(S) \cdot y_S$			
s.c.	$\sum_{S : e \in S} y_S$	$\geq$	1	pour $e \in X$
	y_S	$\in$	$\{0, 1\}$	pour $S \in \mathcal{S}$

Une version relâchée serait

Programme linéaire (RSCP5) : Version relâchée de (SCP4)

Minimiser	$\sum_{S \in \mathcal{S}} \omega(S) \cdot y_S$			
s.c.	$\sum_{S : e \in S} y_S$	$\geq$	1	pour $e \in X$
	y_S	$\in$	$[0, 1]$	pour $S \in \mathcal{S}$

On notera, par abus de langage, $\omega(y) = \sum_{S \in \mathcal{S}} \omega(S) \cdot y_S$ et on identifiera une solution réalisable C de x avec une solution réalisable y de (SCP4).

Une difficulté d'un algorithme d'arrondi est de ne pas arrondir n'importe comment, au risque de produire une solution non réalisable. L'algorithme suivant renvoie toujours une solution réalisable :

Algorithme 12 Algorithme d'arrondi pour le problème de couverture par ensemble.

```
1: f ← le nombre maximum d'ensembles dans lequel un élément e ∈ X se trouve
2: Calculer y°, optimal de (RSCP5)
3: C ← ∅
4: Pour S ∈ S Faire
5:     Si y°_S > 1/f Alors
6:         Ajouter S à C (ou arrondir y°_S à 1)
7: Renvoyer C
```

Cet algorithme est polynomial puisque le calcul de y° est polynomial et puisque la boucle effectue au plus $|\mathcal{S}|$ itérations, chacune en temps constant.

Lemme 3.2.1. *C est une solution réalisable.*

Démonstration. Soit $e \in X$, montrons qu'il existe $S \in C$ tel que $e \in S$. On sait que $\sum_{S : e \in S} y^\circ_S \geq 1$ puisque y° est une solution réalisable du programme (RSCP5).

Si aucun des ensembles contenant e ne se trouve dans C alors $y_S < \frac{1}{f}$ pour tout $S \ni e$; autrement l'ensemble aurait été ajouté à la ligne 6. Or, par définition de f, e est contenu dans au plus f ensembles, donc $\sum_{S:e\in S} y_S^\circ < f\frac{1}{f} = 1$, ce qui est une contradiction. □

Lemme 3.2.2. *Cet algorithme est une f-approximation polynomial.*

Démonstration. Chaque variable y_S°, lorsqu'elle est arrondie en y_S est, dans le pire cas, multipliée par f. Sinon elle est mise à 0. Donc $\omega(C) = \omega(y) \leq f \cdot \omega(y^\circ)$. Enfin, puisque y° est une solution optimale du programme relâché, par le théorème 3.1.4, $\omega(y^\circ) \leq \omega(C^*)$

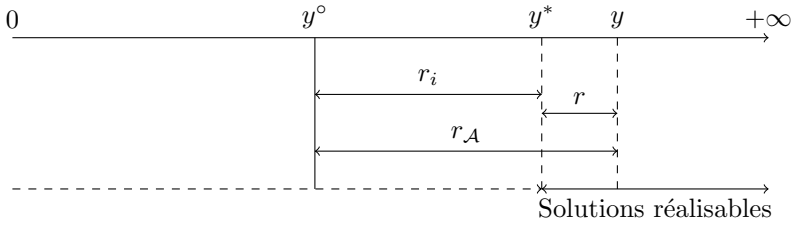
Donc $\omega(C) \leq f \cdot \omega(C^*)$.

D'après le lemme précédent, cet algorithme renvoie toujours une solution réalisable. De plus il est polynomial. On a donc bien un algorithme d'approximation polynomial de rapport f. □

Rapport intégral, rapport d'arrondi et rapport d'approximation

Un algorithme d'arrondi consiste donc à produire, en temps polynomial, une solution réalisable entière y du programme linéaire à partir de la solution optimale relâchée y°.

On a placé sur la figure suivante les trois solutions y°, y^* et y.



Définition 25. On peut voir 3 rapports sur ce dessin.

— Le *rapport intégral* ou *saut intégral* (ou *integral gap* en anglais) est le rapport $r_i = \sup_x \frac{\mathcal{M}^*(x)}{\mathcal{M}^\circ(x)}$.

— Soit un algorithme d'arrondi $\mathcal{A}$ polynomial, le *rapport d'arrondi* est le rapport $r_{\mathcal{A}} = \sup_x \frac{\mathcal{M}(x,\mathcal{A}(x))}{\mathcal{M}^\circ(x)}$.

— Le *rapport d'approximation* est, comme précédemment, le rapport $r = \sup_x \frac{\mathcal{M}(x,\mathcal{A}(x))}{\mathcal{M}^*(x)}$.

Si on regarde bien le dessin, ce qui nous intéresse tout particulièrement est la propriété $r \leq r_{\mathcal{A}}$, qui permet, une fois qu'on a mis la main sur $r_{\mathcal{A}}$, d'en déduire que $\mathcal{A}$ est, au pire, une $r_{\mathcal{A}}$-approximation polynomiale. En soit le rapport r_i n'est pas très intéressant, en effet, r_i et r ne sont pas liés.

Il est possible que $\mathcal{A}$ renvoie toujours une solution très proche de l'optimal, cependant, il n'est pas possible, avec la seule analyse découlant de $r \leq r_{\mathcal{A}}$, de dire mieux que "le rapport d'approximation est $r_{\mathcal{A}}$". Notre algorithme est peut-être très bon, mais on ne peut pas le savoir en utilisant la technique de la borne inférieure (plus exactement, pas avec la borne du théorème 3.1.4). Et on peut clairement voir sur le dessin que $r_i \leq r_{\mathcal{A}}$, donc si on sait que r_i est élevé, alors toute analyse (simple) d'un algorithme d'arrondi résultera en un rapport d'approximation $r_{\mathcal{A}}$ élevé. C'est une des raisons pour lesquelles de nombreuses études s'intéressent au saut intégral, et en particulier, à trouver une borne inférieure de ce saut.

Il faut enfin noter qu'en pratique, on a souvent $r_i \leq r$. Mais aucun résultat ne permet de le prouver dans tous les cas.

Exemple : arrondi probabiliste

On termine cette partie avec un algorithme pour le problème de satisfiabilité maximum qui utilise une manière probabiliste d'arrondir les variables continues.

Problème : Problème de satisfiabilité de poids maximum (maximum Weighted Satisfiability en anglais)

Instance :
- m clauses disjonctives $C_1, C_2, \ldots, C_m$ et n variables $x_1, x_2, \ldots, x_n$.
  $C_j = (l_1^j \vee l_2^j \vee \cdots \vee l_{|C_j|}^j)$, où l_i^j est un litéral x_k ou $\bar{x_k}$.
- Une fonction de poids sur les clauses $\omega : \{C_1, C_2, \ldots, C_m\} \to \mathbb{N}$.

Solution réalisable : Une affectation des variables à vrai ou faux, c'est-à-dire $a_1, a_2, \ldots, a_n \in \{\top, \bot\}$

Optimisation : Maximiser la somme des poids des clauses satisfaites. Si on note $a(C_j)$ la clause C_j où chaque variable x_i est remplacée par a_i, alors C_j est satisfaite si $a(C_j) = \top$. On veut donc maximiser $\sum_{\substack{j=1 \\ a(C_j)=\top}}^{m} \omega(C_j)$.

Détail important, il ne s'agit pas d'un problème de type $3 - SAT$ mais SAT où le nombre de littéraux par clauses est aussi élevé qu'on le souhaite. On notera $x_i \in C_j$ si le littéral x_i apparaît dans la clause C_j. De même avec le littéral $\bar{x_i}$.

On peut résoudre ce problème avec le programme linéaire suivant

Programme linéaire (SATP6) : Programme pour résoudre la satisfiabilité de poids maximum

$$
\begin{array}{lrcll}
\textbf{Maximiser} & \sum_{i=1}^{m} \omega(z_j) \cdot z_j & & & \\
\textbf{s.c.} & \sum_{\substack{i=1 \\ x_i \in C_j}}^{n} y_i + \sum_{\substack{i=1 \\ \bar{x_i} \in C_j}}^{n} (1 - y_i) & \geq & z_j & \text{pour } j \in [\![1; m]\!] \\
& y_i & \in & \{0, 1\} & \text{pour } i \in [\![1; n]\!] \\
& z_j & \in & \{0, 1\} & \text{pour } j \in [\![1; m]\!]
\end{array}
$$

Dans cette version, on souhaite mettre de nombreuses variables z_j à 1, pour indiquer que des clauses sont satisfaites. Mais pour satisfaire une clause, il faut au moins un littéral vrai dans la clause. C'est l'objectif de la première contrainte, qui impose, si la clause est décrétée satisfaite, qu'une variable telle que $x_i \in C_j$ soit vraie ou qu'une variable telle que $\bar{x_i} \in C_j$ soit fausse.

Une version relâchée serait

Programme linéaire (RSATP7) : Version relâchée de (SATP6)

$$
\begin{array}{lrcll}
\textbf{Maximiser} & \sum_{i=1}^{m} \omega(z_j) \cdot z_j & & & \\
\textbf{s.c.} & \sum_{\substack{i=1 \\ x_i \in C_j}}^{n} y_i + \sum_{\substack{i=1 \\ \bar{x_i} \in C_j}}^{n} (1 - y_i) & \geq & z_j & \text{pour } j \in [\![1; m]\!] \\
& y_i & \in & [0, 1] & \text{pour } i \in [\![1; n]\!] \\
& z_j & \in & [0, 1] & \text{pour } j \in [\![1; m]\!]
\end{array}
$$

On utilise ensuite simplement l'algorithme suivant :

Algorithme 13 Algorithme d'arrondi pour le problème de satisfiabilité de poids maximum.

1: Calculer y°, z°, optimal de (RSATP7)
2: **Pour** $i \in [\![1; n]\!]$ **Faire**
3: Mettre x_i à vrai (*i.e. arrondir y_i à 1*) avec une probabilité y_i°
4: **Renvoyer** $x_1, x_2, \ldots, x_n$

Cet algorithme est clairement polynomial et renvoie une solution réalisable. En effet, toute affectation des variables, quelle qu'elle soit, est réalisable. Il nous reste donc à prouver son rapport d'approximation. Ce rapport est moins évident ici puisque l'algorithme est probabiliste, il peut renvoyer une bonne comme une mauvaise solution. On va donc, dans un premier temps du moins, s'intéresser à l'espérance du poids de la solution

renvoyée. On va montrer que cette espérance est supérieure à $(1 - \frac{1}{\mathrm{e}})$ multiplié par le poids de la solution optimale. On verra ensuite comment utiliser ce résultat pour produire une $(1 - \frac{1}{\mathrm{e}})$-approximation polynomiale.

Lemme 3.2.3. *La probabilité p_j que la clause C_j soit vraie est* $1 - \prod_{\substack{i=1 \\ x_i \in C_j}} (1 - y_i^\circ) \cdot \prod_{\substack{i=1 \\ \bar{x_i} \in C_j}} (y_i^\circ)$.

Démonstration. C_j est fausse si toutes ses littéraux sont faux. Autrement dit, on a mis $x_i \in C_j$ à faux, avec un probabilité $(1 - y_i^\circ)$ et $\bar{x_i} \in C_j$ à vrai avec un probabilité y_i°. Enfin, les tirages aléatoires sont indépendants, donc la probabilité que C_j soit fausse est le produit des probabilités de tous ces évènements. □

On note $E(\Omega)$ l'espérance du poids $\Omega = \sum_{\substack{j=1 \\ a(C_j)=\top}}^{m} \omega(C_j)$ de la solution, égale à

$$E(\Omega) = \sum_{j=1}^{m} \omega(C_j) \cdot p_j$$

Lemme 3.2.4. *Pour toute clause C_j, $z_j^\circ \cdot \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) \leq p_j$*

Avant de démontrer ce résultat, remarquons qu'il permettra de relier $E(\Omega)$ avec le poids d'une solution optimale par un rapport $\frac{1}{\left(1-\left(1-\frac{1}{k}\right)^k\right)}$ où k est la taille de la plus grande clause.

Démonstration. Sans perte de généralité, on va supposer tous les littéraux de la clause positifs. Si ce n'est pas vrai, on peut remplacer $\bar{x_i}$ par x_i dans toutes les clauses pour les variables x_i apparaissant négativement dans la clause (et si une variable apparaît 2 fois avec ses deux littéraux, alors la clause est vraie quoi qu'il arrive, on peut donc la remplacer par une clause vide). Pour simplifier encore, on renumérote les variables de l'instance de sorte à ce que $C_j = (x_1 \vee x_2 \vee \cdots \vee x_k)$.

On veut donc prouver, d'après le lemme 3.2.3, que $z_j^\circ \cdot \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \leq p_j = 1 - \prod_{i=1}^{k} (1 - y_i^\circ)$.

On utilise pour ça l'inégalité arithmétique et géométrique : la moyenne arithmétique de nombres positifs est plus grande que leur moyenne géométrique. En employant ce résultat sur les nombres $(1-y_1^\circ, 1-y_2^\circ, \ldots, 1-y_k^\circ)$, on obtient le résultat suivant.

$$\begin{aligned} 1 - \prod_{i=1}^{k} (1 - y_i^\circ) &\geq 1 - \left(\sum_{i=1}^{k} (1 - y_i^\circ)/k\right)^k \\ &\geq 1 - \left(1 - \sum_{i=1}^{k} y_i^\circ/k\right)^k \end{aligned}$$

Or y°, z° est une solution réalisable de (SATP7), donc $\sum_{i=1}^{k} y_i^\circ \geq z_j^\circ$

$$\geq 1 - \left(1 - \frac{z_j^\circ}{k}\right)^k$$

Or la fonction $z_j^\circ \to 1 - \left(1 - \frac{z_j^\circ}{k}\right)^k$ est concave sur $[0, 1]$. En 0 elle vaut 0 et en 1 elle vaut $1 - \left(1 - \frac{1}{k}\right)^k$. Elle est au dessus du segment qui relie ces deux points, c'est-à-dire le segment d'équation $\left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot z_j^\circ$. □

Théorème 3.2.1. *Soit Ω^* le poids d'une solution optimale, $E(\Omega) \geq (1 - \frac{1}{\mathrm{e}})\Omega^*$.*

Démonstration. D'après le lemme 3.2.4,

$$\begin{aligned} E(\Omega) &= \sum_{j=1}^{m} \omega(C_j) \cdot p_j \\ &\geq \sum_{j=1}^{m} \omega(C_j) \cdot z_j^\circ \cdot \left(1 - \left(1 - \frac{1}{|C_j|}\right)^{|C_j|}\right) \end{aligned}$$

Soit k la taille de la plus grande clause

$$\geq \left(\sum_{j=1}^{m} \omega(C_j) \cdot z_j^\circ\right) \cdot \left(1 - \left(1 - \frac{1}{k}\right)^k\right)$$
$$\geq \omega^* \cdot \left(1 - \left(1 - \frac{1}{k}\right)^k\right)$$

On avait enfin prouvé dans le lemme 1.3.2 que $\left(1 - \frac{1}{k}\right)^k$ était une fonction croissante qui tend vers $\frac{1}{\mathrm{e}}$, d'où le résultat du lemme. □

Il est possible d'améliorer cet algorithme pour obtenir un rapport $\frac{4}{3}$, en le combinant avec une autre approximation. Mais cette preuve sortirait du cadre du chapitre.

Déterminisation d'un algorithme probabiliste

Avant de passer à la section suivante, montrons qu'il est possible de déterminiser l'algorithme précédent, c'est-à-dire d'en faire un algorithme non probabiliste qui obtient au moins les même performances. Cette technique est souvent utilisable quand on utilise des probabilités pour arrondir les variables d'un programme linéaire.

On voit qu'on peut calculer l'espérance $E(\Omega)$ en temps polynomial, sans connaître le poids de quelque solution que ce soit. On a juste à calculer les probabilité y_i° en résolvant le programme (RSATP7), à calculer les probabilités p_j et à calculer $\sum_{j=1}^{m} \omega(C_j) \cdot p_j$.

Posons $E(\Omega|x_i = \top)$ le poids moyen d'une solution sachant qu'on force l'affectation $a_i = \top$. On peut facilement calculer la formule obtenue en remplaçant $x_i = \top$: les clauses contenant x_i se simplifient en $\top$, celles contenant $\bar{x_i}$ se simplifient en retirant ce littéral. Dans ce dernier cas, si la clause devient vide, elle est remplacée par $\bot$. Les autres clauses ne changent pas. Avec les probabilités y_i°, on peut encore calculer l'espérance $E(\Omega|x_i = \top)$ sans calcul complexe, en temps polynomial, en utilisant le lemme 3.2.3 sur nos nouvelles clauses. De même, on peut calculer $E(\Omega|x_i = \bot)$ en temps polynomial. L'astuce tient dans le fait que, nécessairement, l'une de ces deux valeurs est supérieure à $E(\Omega)$ car $E(\Omega) = E(\Omega|x_i = \top)y_i^\circ + E(\Omega|x_i = \bot)(1 - y_i^\circ)$. On a donc intérêt à choisir pour x_i la valeur associée à la meilleure espérance. On recommence jusqu'à avoir choisi une valeur pour toutes les variables, auquel cas l'espérance devient une valeur certaine.

On va donc appliquer l'algorithme suivant.

Calculer y°, optimal de (RSATP7)
$\varphi \leftarrow (C_1, C_2, ;C_m)$
Pour $i \in [\![1; n]\!]$ **Faire**
  $E_\top \leftarrow$ l'espérance du poids moyen d'une solution où $a_i = \top$
  $E_\bot \leftarrow$ l'espérance du poids moyen d'une solution où $a_i = \bot$
  Si $E_\top > E_\bot$ **Alors**
    $a_i \leftarrow \top$
    $\varphi \leftarrow$ les clauses de φ où on a remplacé x_i par $\top$
  Sinon
    $a_i \leftarrow \bot$
    $\varphi \leftarrow$ les clauses de φ où on a remplacé x_i par $\bot$
Renvoyer $a_1, a_2, \ldots, a_n$

Exemple 8. Par exemple, supposons pour l'exemple que toutes les variables aient une probabilité $y_i^\circ = \frac{1}{2}$

$\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \bar{x_2}) \wedge (\bar{x_1} \vee x_2) \wedge (\bar{x_1} \vee \bar{x_2})$, où les poids des clauses sont respectivement 1, 4, 5, 6 alors on a $E(\Omega) = \frac{3}{4} \cdot 1 + \frac{3}{4} \cdot 4 + \frac{3}{4} \cdot 5 + \frac{3}{4} \cdot 6 = 12$.

Si on remplace x_1 par $\top$, on obtient $\top \wedge \top \wedge x_2 \wedge \bar{x_2}$ d'espérance $E_\top = 1 + 4 + \frac{1}{2} \cdot 5 + \frac{1}{2} \cdot 6 = 10.5$. Et si x_1 devient $\bot$, alors on obtient $x_2 \wedge \bar{x_2} \wedge \top \wedge \top$ d'espérance $E_\bot = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 4 + 5 + 6 = 13.5$. On choisit donc $x_1 = \bot$ et on applique l'algorithme sur $\varphi \leftarrow x_2 \wedge \bar{x_2} \wedge \top \wedge \top$.

Si on remplace x_2 par $\top$, on obtient $\top \wedge \bot \wedge \top \wedge \top$ d'espérance $E_\top = 12$ et, si on le remplace par $\bot$, $\bot \wedge \top \wedge \top \wedge \top$ d'espérance $E_\bot = 15$. On choisit donc $x_2 = \bot$.

Théorème 3.2.2. *Cet algorithme est une* $(1 - \frac{1}{\mathrm{e}})$*-approximation polynomiale.*

Démonstration. Cet algorithme est polynomial puisqu'il résout un programme linéaire puis n fois de suite, il calcule deux espérances, une comparaison et une formule en $O(m \cdot n)$.

On voit que l'espérance de la formule augmente à chaque itération, par construction. La dernière formule n'a pas de variable, donc sa valeur est nécessairement égale à son espérance. Donc sa valeur est supérieure à l'espérance de la formule d'origine. L'algorithme donne donc en temps polynomial non probabiliste une affectation des variables dont la valeur associée est supérieure à $E(\Omega)$.

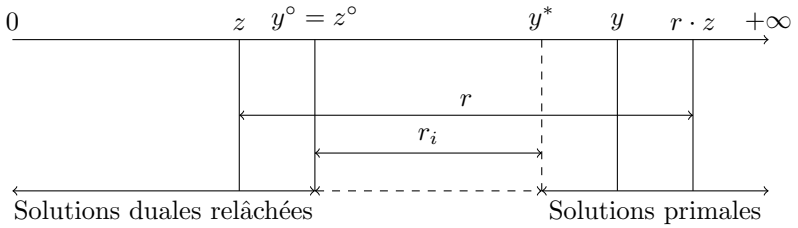
Le rapport se déduit donc du théorème 3.2.1. □

Remarque 8. On pourrait, à chaque itération, recalculer les probabilités y_i° en recalculant la solution optimale du programme (RSATP7), sachant que certaines variables ont été fixées à vrai ou faux. Pour s'assurer qu'on conserve le rapport d'approximation, il faudrait alors montrer que cette technique augmente l'espérance d'une itération à l'autre.

3.3 Dual fitting

La technique de dual fitting est plus complexe à mettre en œuvre que l'arrondi car, comme son nom l'indique, elle fait intervenir le dual. Cependant, conceptuellement, elle reste très simple à comprendre.

Elle consiste à calculer une solution réalisable y de l'instance x et une solution réalisable du dual z telle que $c \cdot {}^t y \leq r \cdot b \cdot {}^t z$. Le rapport d'approximation se déduit de l'écart entre le poids de z et celui de y, puisque y^* a un poids compris entre les deux.



On peut voir, comme pour l'arrondi, que cette technique dévoilera nécessairement un rapport d'approximation r plus grand que le saut intégral r_i. Encore une fois, c'est l'analyse qui est en cause et non l'algorithme car le rapport d'approximation réel, rapport entre le poids de y et de y^*, peut être bien meilleur que r.

Exemple : problème de couverture par ensemble

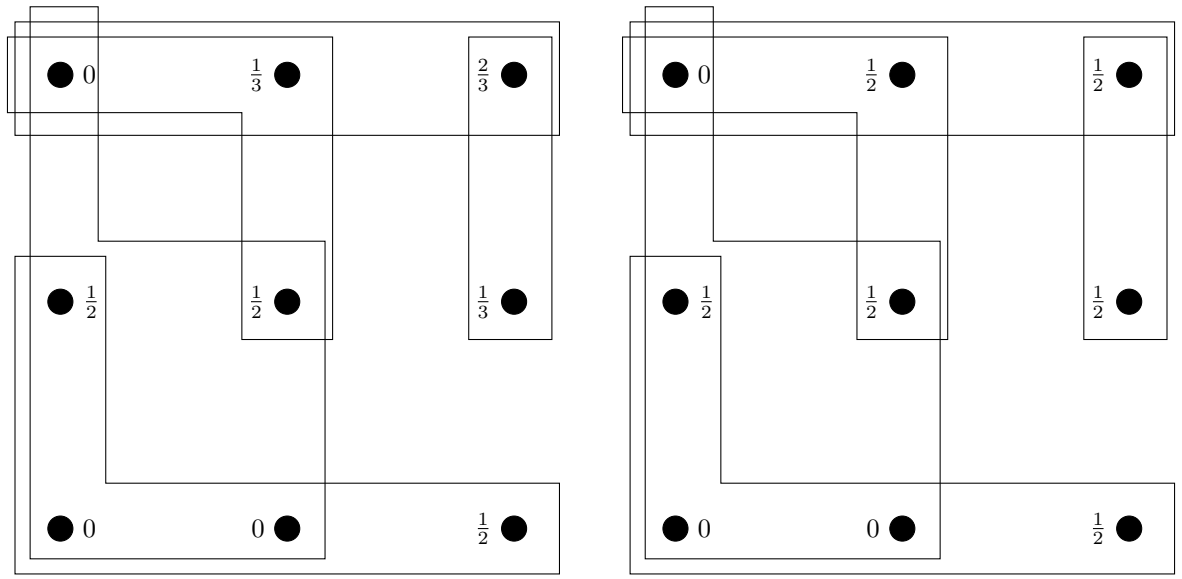
On reprend le programme linéaire (SCP4) et sa version relâchée (RSCP5). Le dual de ce programme relâché est le programme suivant :

Programme linéaire (DSCP8) : Dual relâché de (SCP4)

Minimiser	$\sum_{e \in X} z_e$			
s.c.	$\sum_{e \in S} z_e$	$\leq$	$\omega(S)$	pour $S \in \mathcal{S}$
	z_e	$\geq$	0	pour $e \in X$

Ce dual, comme la plupart des duals de problèmes de couvertures, est un problème de *packing*. On veut charger les éléments de X avec un poids z_e de sorte que le poids des éléments d'un ensemble S ne dépasse pas $\omega(S)$.

Sur l'exemple du premier chapitre, où tous les poids sont unitaires, on aurait par exemple les deux solutions ci dessous, celle de gauche est réalisable, celle de droite est optimale.



On va démontrer que l'algorithme 5 glouton peut être réétudié sous l'angle du dual fitting, en montrant le même résultat, c'est-à-dire qu'il s'agit d'une $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} \leq \ln(k) + 1$ approximation, avec $k = |X|$. On rappelle que l'algorithme glouton sélectionne à chaque itération i l'ensemble S_i dont le rapport $\frac{\omega(S)}{|S \cap X'_i|}$ est minimum, où X'_i est l'ensemble des éléments non couverts au début de l'itération i, autrement dit un mélange entre l'ensemble le plus gros et le moins cher.

On avait introduit la variable $price(e)$. On en redonne la définition ici. Soit $e \in X$, on note $i(e)$ l'itération où e a été couvert. Si on défini $S = S_{i(e)}$ et $X' = X'_{i(e)}$ alors on note $price(e) = \omega(S)/|S \cap X'|$.

Posons $z_e = \frac{price(e)}{H_k}$. On pose C la couverture renvoyée par l'algorithme glouton et y la solution du programme (SCP4) associée. On va montrer que la solution duale z et la solution y vérifient les conditions de l'approche dual fitting avec le rapport H_k.

La première propriété est démontrée par le lemme suivant.

Lemme 3.3.1. $\sum_{S \in \mathcal{S}} \omega(S) \cdot y_S \leq \sum_{e \in X} z_e \cdot H_k$

Démonstration. C'est un point qu'on avait déjà abordé dans le premier chapitre. On avait montré qu'additionner les valeurs des différents $price(e)$ donnait le poids $\omega(C)$ car $price$ a été conçu de sorte à répartir le poids de S parmi les éléments qu'il couvre. C'est ce qu'on va redémontrer ici :

$\sum_{S \in \mathcal{S}} \omega(S) \cdot y_S$ est le poids $\omega(C)$ et $\sum_{e \in X} z_e \cdot H_k = \sum_{e \in X} price_e$.

Soit un ensemble $S \in C$ et i l'itération à laquelle on a choisi S (donc $S = S_i$). Soient $E_i = \{e \in X | i(e) = i\}$, autrement dits les éléments couverts grâce à S, ceux qui n'étaient pas couvert au début de l'itération i mais couverts à la fin de celle-ci, donc ceux qui sont dans X'_i au début de l'itération et qui sont aussi dans S : $E_i = S \cap X'_i$. Pour tout $e \in E_i$, $price(e) = \frac{\omega(S)}{|E_i|}$. Donc $\sum_{e \in E_i} price(e) = \omega(S)$.

Tous les ensembles E_i forment une partition de X : $\sum_{e \in X} price(e) = \sum_{i=1}^{|C|} \sum_{e \in E_i} price(e) = \sum_{i=1}^{|C|} \omega(S_i) = \omega(C)$. On a donc bien l'égalité souhaitée. □

On peut remarquer que $z \cdot H_k$ n'est pas une solution du dual. En effet, S peut contenir des éléments qui n'ont pas été couverts grâce à S mais par un autre ensemble de C choisi plus tôt dans l'algorithme. Donc on peut avoir $\sum_{e \in S} price(e) > \omega(S)$ (il faut donc encore une fois bien faire la différence entre le fait qu'un élément appartienne à S et soit couvert grâce à S). Donc pour obtenir une solution réalisable et donc la seconde propriété du dual fitting, il nous faut réduire les valeurs $price(e)$, comme on l'a fait avec z en divisant par H_k.

Lemme 3.3.2. *z est une solution réalisable du dual.*

Démonstration. Il suffit de montrer que chaque contrainte $\sum_{e \in S} z_e \leq \omega(S)$ est vérifiée pour tout ensemble $S \in \mathcal{S}$. (Détail important, on ne s'intéresse pas qu'aux ensembles de C.)

On note $s = |S|$. On va renommer $e_1, e_2, \ldots, e_s$ les éléments de S, rangés dans l'ordre dans lequel ils sont couverts par l'algorithme glouton. Attention, e_1 a pu être couvert plusieurs itérations avant e_2, et ce, même si $S \in C$.

Soit $j \in [\![1;s]\!]$ et $i = i(e_j)$ l'itération où e_j a été couvert. Puisque l'algorithme a choisi de mettre S_i dans C plutôt que S, il préfère S_i à S (ou alors $S_i = S$) et donc, $price(e_j) = \frac{\omega(S_i)}{|X'_i \cap S_i|} \leq \frac{\omega(S)}{|X'_i \cap S|}$. Puisque e_j n'est pas couvert au début de l'itération i et que e_j est couvert avant $e_{j+1}, e_{j+2}, \ldots, e_s$, alors tous ces éléments sont dans $X'_i \cap S$. Donc, pour tout $j \in [\![1;s]\!]$, $\frac{\omega(S)}{|X'_i \cap S|} \leq \frac{\omega(S)}{s-j+1}$.

Donc $\sum_{j=1}^{s} z_{e_j} = \sum_{j=1}^{s} \frac{price(e_j)}{H_k} \leq \sum_{j=1}^{s} \frac{\omega(S)}{(s-j+1)\cdot H_k} \leq \frac{\omega(S) H_s}{H_k} \leq \omega(S)$.

La solution duale est donc réalisable. □

Vous remarquerez que la preuve ressemble à celle du théorème 1.3.3 mais est plus simple car elle ne fait pas intervenir la solution optimale.

Théorème 3.3.1. *L'algorithme glouton est une H_k-approximation polynomiale.*

Démonstration. L'algorithme est polynomial, comme vu au premier chapitre.

Soient C et C^* respectivement la solution renvoyée par l'algorithme glouton et une solution optimale. Puisque z est une solution réalisable du dual alors $\sum_{e \in X} z_e \leq \omega(C^*) = \sum_{S \in \mathcal{S}} y^*_S$. Et puisque $\omega(C) = \sum_{S \in \mathcal{S}} \omega(S) \cdot y_S \leq \sum_{e \in X} z_e \cdot H_k$, alors $\omega(C) \leq \omega(C^*) H_k$. □

3.4 Schémas prima-dual

La technique du schémas prima-dual est, comme celle du dual fitting, plus complexe à mettre en œuvre que l'arrondi et fait également intervenir le dual.

Elle se base sur les conditions d'écarts complémentaires. Si ces conditions donnent des conditions nécessaires et suffisantes d'optimalité, il est logique de penser qu'en relâchant ces conditions, on pourrait donner des conditions suffisantes d'approximation.

Théorème 3.4.1 (Théorème des écarts complémentaires)**.** *Soient $\alpha \geq 1$ et $\beta \geq 1$. Soit y et z des solutions réalisables primales et duales. Ces deux solutions vérifient les conditions relâchées si*

- *Pour tout $i \leq n$, $y_i = 0$ ou $\frac{c_i}{\alpha} \leq \sum_{j=1}^{m} a_{ji} z_j \leq c_i$*
- *Pour tout $j \leq m$, $z_j = 0$ ou $b_j \leq \sum_{i=1}^{n} a_{ji} y_i \leq b_j \cdot \beta$*

alors dans ce cas

$$c \cdot {}^t y \leq \alpha \cdot \beta \cdot b \cdot {}^t z$$

Remarque 9. Ces conditions sont bien des conditions relâchées dans le sens où si $\alpha = 1$ alors les conditions d'écart complémentaires primales sont vérifiées et si $\beta = 1$ alors les conditions duales sont vérifiées.

Démonstration.

$$\begin{aligned} c \cdot {}^t y &= \sum_{i=1}^{n} c_i \cdot y_i \\ &= \sum_{\substack{i=1 \\ y_i \neq 0}}^{n} c_i \cdot y_i \end{aligned}$$

D'après les conditions relâchées, si $y_i \neq 0$ alors $\frac{c_i}{\alpha} \leq \sum_{j=1}^{m} a_{ji} z_j$

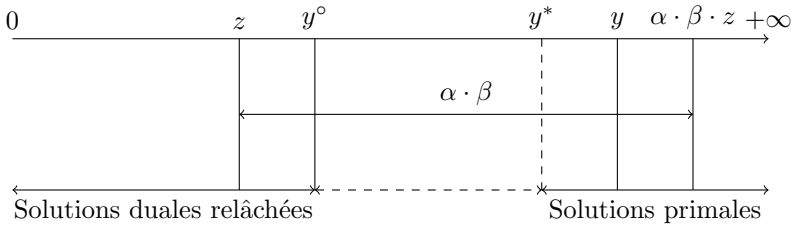
$$\begin{aligned} &\leq \sum_{\substack{i=1 \\ y_i \neq 0}}^{n} \left(\alpha \cdot \sum_{j=1}^{m} a_{ji} z_j \right) \cdot y_i \\ &\leq \alpha \cdot \sum_{i=1}^{n} \left(\sum_{j=1}^{m} a_{ji} z_j \right) \cdot y_i \\ &\leq \alpha \cdot \sum_{j=1}^{m} \left(\sum_{i=1}^{n} a_{ji} y_i \right) \cdot z_j \\ &\leq \alpha \cdot \sum_{\substack{j=1 \\ z_j \neq 0}}^{m} \left(\sum_{i=1}^{n} a_{ji} y_i \right) \cdot z_j \end{aligned}$$

D'après les conditions relâchées, si $z_j \neq 0$ alors $\sum_{i=1}^{n} a_{ji} y_i \leq b_j \cdot \beta$

$$\leq \alpha \cdot \sum_{\substack{j=1 \\ z_j \neq 0}}^{m} \beta \cdot b_j \cdot z_j$$
$$\leq \alpha \cdot \beta \cdot b \cdot {}^t z$$

□

Si on est capable de produire des solutions duales et primales vérifiant les conditions d'écart complémentaires, on a donc une solution $\alpha \cdot \beta$ approchée, comme le montre le dessin ci-dessous. On peut remarquer qu'il s'agit d'une version particulière de la technique de dual fitting.



Remarque 10. On peut noter que la démonstration du théorème 3.4.1 n'utilise pas le fait que les solutions y et z soient réalisables. La faisabilité des solutions est utilisée uniquement pour montrer que y est une solution $\alpha\beta$ approchée (dans le cas contraire, y et z ne seraient par exemple pas du bon côté de y° sur le dessin ci-dessus).

On peut voir, comme pour l'arrondi et le dual fitting, que cette technique dévoilera nécessairement un rapport d'approximation $\alpha\beta$ plus grand que le saut intégral r_i. Encore une fois, c'est l'analyse qui est en cause et non l'algorithme car le rapport d'approximation réel, rapport entre le poids de y et de y^*, peut être bien meilleur que $\alpha\beta$.

Exemple : problème de couverture par ensemble

On réutilise pour cet exemple les programmes (SCP4) et (DSCP8). Pour produire une solution z duale et une solution y réalisable de l'instance x, nous allons utiliser l'algorithme suivant :

Algorithme 14 Schémas prima-dual pour Set Cover

1: $C \leftarrow \emptyset$
2: $z \leftarrow 0$
3: **Tant que** Un élément e de X n'est pas couvert **Faire**
4: Augmenter z_e jusqu'à ce qu'il existe $S \in \mathcal{S}$ tel que $\sum_{e \in S} z_e = \omega(S)$.
5: Ajouter à C tous les ensembles S tels que $\sum_{e \in S} z_e = \omega(S)$
6: **Renvoyer** C

Comme précédemment, on pose y la solution primale associée à C.

Lemme 3.4.1. *z et y vérifient les conditions d'écarts relâchées avec $\alpha = 1$ et $\beta = f$ où f est le nombre maximum d'ensembles contenant un même élément.*

Démonstration. Soit $S \in \mathcal{S}$ tel que $y_S \neq 0$. Dans ce cas, $y_S = 1$. En effet, y_S vaut soit 0, soit 1, selon que $S \notin C$ ou $S \in C$. Donc, S a été ajouté à C à la ligne 5, donc la contrainte associée à S est saturée. On a donc $\sum_{e \in S} z_e = \omega(S)$ et donc $\frac{\omega(S)}{1} \leq \sum_{e \in S} z_e \leq \omega(S)$. De plus cela montre que z est une solution réalisable du dual.

y est une solution réalisable du primal, en effet, à chaque itération, on considère un élément e non couvert et on ajoute au moins un ensemble S contenant e à C. La seconde contrainte des conditions relâchées est toujours vérifiée. En effet, pour tout $e \in X$, $\sum_{\substack{S \in \mathcal{S} \\ e \in S}} y_S$ ne peut être plus grand que f puisque e est contenu dans au plus f ensembles (par définition de f). Donc $1 \leq \sum_{\substack{S \in \mathcal{S} \\ e \in S}} y_S \leq f$. □

Remarque 11. Lorsque l'on prouve les conditions d'écarts relâchées, il ne faut pas oublier de vérifier que y et z sont réalisables pour le primal et le dual.

Théorème 3.4.2. *Cet algorithme est une f-approximation polynomiale.*

Démonstration. L'algorithme est polynomial. Il effectue au plus $|X|$ itérations de la boucle while et chaque itération nécessite de trouver le premier ensemble saturé par l'augmentation de z_e, c'est donc un calcul de minimum sur au plus $f \leq |\mathcal{S}|$ ensembles.

Soit C^* une solution optimale de l'instance. D'après le lemme précédent, les deux solutions calculées vérifient les conditions d'écarts complémentaires relâchées avec $\alpha = 1$ et $\beta = f$. D'après le théorème 3.4.1, $\omega(C) = \sum_{S \in \mathcal{S}} \omega(S) \cdot y_S \leq \alpha\beta \sum_{e \in X} z_e \leq f \cdot \omega(C^*)$. □

3.5 Conclusion

Nous avons vu, dans ce chapitre, trois techniques pour produire des algorithmes d'approximations polynomiaux à l'aide de la programmation linéaire, l'arrondi, le dual fitting et le schéma prima-dual. Chaque technique a ses avantages et ses inconvénients.

Une chose à ne pas oublier quand on utilise ces techniques est que, sans analyse plus approfondie que l'analyse de base fournie dans ce cours, le rapport d'approximation de ces algorithmes ne peut être plus petit que le saut intégral. Ce saut intégral dépend fortement du modèle de programmation linéaire employé. Il est donc toujours bon de se renseigner, avant de se lancer dans une telle technique, sur la valeur connue de ce saut (ou d'une borne inférieure). Si elle est de l'ordre du plus petit rapport d'approximation qu'on connaisse pour le problème, il est inutile de chercher à utiliser les techniques de ce chapitre, sauf si l'objectif est de reproduire un résultat connu en utilisant la programmation linéaire.

Un dernier détail a pu vous échapper au cours de ce chapitre. La programmation linéaire est résolue par les solveurs mécaniquement, numériquement, sans tenir compte de ce que signifie le modèle, les variables ou les contraintes vis-à-vis du problème à résoudre. Un solveur se contrefiche de savoir que la contrainte signifie qu'il faut mettre une variable à vrai ou qu'il faut couvrir un élément. Cependant, pour la conception d'un algorithme d'approximation, nous n'avons quasiment jamais utilisé le modèle seul. Un arrondi trop simpliste peut mener à une solution non réalisable ou inintéressante et il n'y a aucune chance, sans tenir compte des spécificités du problèmes, de pouvoir produire une solution duale réalisable et intéressante pour la technique de dual fitting. Il est donc important, pour mener à bien une des 3 techniques de ce chapitre de bien connaître le modèle et sa relation avec le problème étudié. Que signifie telle variable ? Comment est-ce que je comprend le fait de relâcher la contrainte d'intégrité ? Que veut-dire mettre telle variable à 0.5 au lieu de 1 ? Si cette contrainte est saturée, qu'est-ce que ça change ? Toutes ces questions doivent être posées pour pouvoir produire un algorithme d'approximation de qualité.

Chapitre 4

Inapproximabilité

Nous terminons ce chapitre par des méthodes pour prouver qu'un problème ne peut être approché avec un rapport donné. On a déjà aperçu dans les chapitres précédents des réductions qui permettaient de prouver ce genre de résultats, par exemple en montrant que le voyageur de commerce ne peut être approché avec un rapport constant. Ce chapitre formalisera ce type de preuves via la définitions de réductions spécifiques à l'approximation polynomiale.

4.1 Rapport serré d'un algorithme d'approximation

Une première question à se poser quand on parle d'inapproximabilité est de savoir, quand on affirme qu'un algorithme est une r-approximation polynomiale, si ce resultat est précis. Est-ce que le rapport r n'est pas surévalué par rapport à la réalité. Par exemple, on a prouvé au premier chapitre que l'algorithme de Christofides est une $\frac{3}{2}$-approximation polynomiale. Comment prouver qu'il ne s'agit pas en fait d'une $\frac{4}{3}$-approximation polynomiale ? Comment prouver que, lors de notre analyse de l'algorithme, on a pas effectué un raisonnement qui aurait pu être plus fin et améliorer le résultat ? C'est le principe du rapport serré. Définir ce rapport serré est toutefois quelque chose d'assez complexe.

Intuitivement, un rapport r sera dit serré pour un algorithme $\mathcal{A}$ si $\mathcal{A}$ est une r-approximation et si on est capable de définir une instance pour laquelle l'erreur est r. Mais cette définition n'est pas suffisante, car il peut exister des algorithmes où le rapport n'est jamais atteint mais où l'erreur peut se rapprocher arbitrairement de r. C'est par exemple le cas avec le problème de Steiner. On a vu au chapitre 1 une 2-approximation. En fait il s'agissait d'un rapport de valeur $2 \cdot (1 - \frac{1}{k})$, donc il ne semble exister aucune instance pour laquelle l'erreur est exactement 2. Pourtant, on verra qu'il existe des instances pour lesquelles l'erreur est exactement $2 \cdot (1 - \frac{1}{k})$, et cette erreur se rapproche donc arbitrairement de 2.

On pourrait donc définir un rapport serré comme le fait que $\mathcal{A}$ est une r-approximation mais il existe une suite infinie d'instances dont l'erreur se rapproche arbitrairement de r. Mais cette définition pose un tout petit problème. Si je connais une instance x pour laquelle l'erreur est r, alors je peux considérer la suite infinie des instances toutes égales à x. L'erreur est toujours égale à r, donc d'une certaine manière se rapproche arbitrairement de r. Imaginons maintenant qu'on ait une 5-approximation et qu'on trouve une instance de taille 1 où ce que renvoie l'algorithme a un poids égal $5 \cdot \mathcal{M}^*(x)$. Imaginons également que notre rapport d'approximation soit en fait 5 si l'instance est de taille 1 et 2 sinon. Dans l'approche du pire cas, on peut admettre qu'on a bien une 5-approximation serrée, car il existe une instance où l'algorithme se trompe d'un rapport 5. Mais, en même temps, il serait très simple de construire une 2-approximation : résoudre de manière exacte toute instance de taille 1 (on peut toujours résoudre de manière exacte et en temps constant une instance dont la taille est constante, il suffit d'énumérer toutes les solutions réalisables, qui sont en nombre constant) et utiliser la 2-approximation sur les autres instances. En fait ici, il semble déraisonnable de dire que l'algorithme est une 5-approximation serrée. Ce qui nous intéresse c'est les cas à l'infini [1].

On emploiera donc la définition ci-dessous :

Définition 26. Soit un problème $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O})$ de minimisation et $\mathcal{A}$ une r-approximation polynomiale résolvant Π. Le rapport r est dit serré s'il existe une suite d'instances $(x_n)_{n\in\mathbb{N}} \in \mathcal{I}^{\mathbb{N}}$ telles que $|x_n|$ croit strictement avec n et

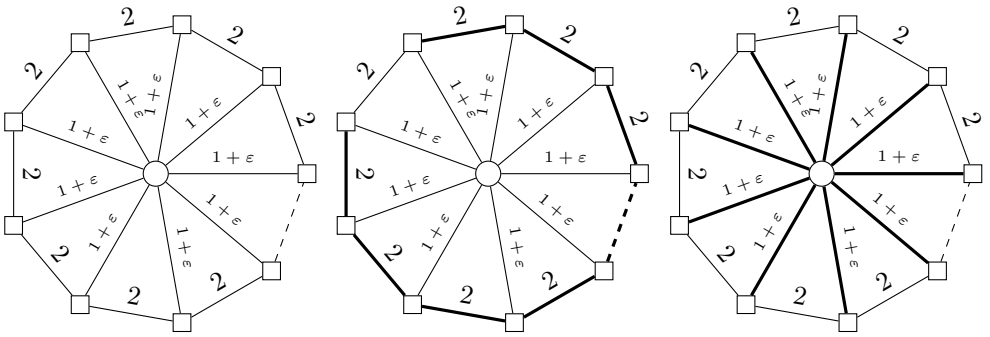
$$\lim_{n\to+\infty} \frac{\mathcal{M}(x_n, \mathcal{A}(x_n))}{r(x_n)\mathcal{M}^*(x_n)} = 1$$

1. NDLA : Pour être honnête, je ne connais aucune définition précise d'un rapport serré. Mais celle qui est donnée dans ce cours me semble la plus logique à mettre en place pour décrire correctement un rapport d'approximation et secondement qui correspond à la définition implicitement employée par toutes les sources que j'ai lues. La petite discussion sur ce lien fait mention de la difficulté de définir un rapport serré : `https://cs.stackexchange.com/questions/6140/providing-tight-example-in-approximation-algorithm-analysis`

Reprenons notre exemple du problème de Steiner.

Théorème 4.1.1. *L'algorithme de Choukkmane, de Kou et al., de PLesnìk et de Iwainsky et al. a un rapport serré égal à* 2.

Démonstration. L'algorithme a un rapport 2 d'après le théorème 1.1.1. Il suffit de considérer les instances suivantes, où les terminaux sont les nœuds carrés et où $\varepsilon > 0$.



Dans cette instance, tous les plus courts chemins entre les terminaux voisins valent 2, et entre les autres $2+2\varepsilon$. Donc l'algorithme va renvoyer le cycle extérieur privé d'une arête y (figure centrale), de poids $\mathcal{M}(x,y) = 2k(1-\frac{1}{k})$. La solution optimale (figure de droite) vaut à la place $\mathcal{M}^*(x) = k \cdot (1+\varepsilon)$.

On a donc $\frac{\mathcal{M}(x,y)}{\mathcal{M}^*(x)} = \frac{2(1-\frac{1}{k})}{1+\varepsilon}$. Prenons donc la suite d'instances $(x_n)_{n\in\mathbb{N}}$ où, dans x_n, on a $k = n$ et $\varepsilon = \frac{1}{n}$. On voit que la taille de x_n est $n+1$ et est strictement croissante avec n.

$$\frac{\mathcal{M}(x_n,y)}{r(x_n)\cdot\mathcal{M}^*(x_n)} = \frac{2k\cdot(1-\frac{1}{k})}{2k\cdot(1+\varepsilon)} = \frac{1-\frac{1}{n}}{1+\frac{1}{n}} \to_{n\to+\infty} 1$$

. □

Étrangement, la même série d'instances démontre également un rapport serré de $2(1-\frac{1}{k})$. En effet

$$\lim_{n\to+\infty} \frac{\mathcal{M}(x_n,\mathcal{A}(x_n))}{2(1-\frac{1}{k})\mathcal{M}^*(x_n)} = 1$$

On a étrangement 2 rapports serrés, alors qu'intuitivement, on aurait pu supposer le rapport serré unique. Comment le même algorithme pourrait-il avoir 2 rapports d'approximation ? C'est tout le problème quand les rapports sont des fonctions et non des constantes, la notion asymptotique peut mener à plusieurs résultats. On recherche un rapport r tel que $\mathcal{M}(x_n,\mathcal{A}(x_n))$ soit équivalent à l'infini à $r(x_n)\mathcal{M}^*(x_n)$. Et il peut existe plusieurs fonctions qui vérifient cette propriété.

Remarque 12. On a pris une petite liberté avec les poids ici. Nos poids ne sont pas entiers. Ce n'est, normalement, pas autorisé pour les problèmes de NPO. Mais il suffirait de multiplier tous les poids par n pour obtenir des poids entiers avec le même résultat.

On donne ci-après un second exemple avec l'algorithme Next Fit pour le problème de Bin Packing.

Théorème 4.1.2. *L'algorithme Next Fit a un rapport serré égal à* 2.

Démonstration. L'algorithme a un rapport 2 d'après le théorème 1.5.2.

Posons x_n l'instance contenant $2n$ éléments avec $V = n$, $v_{2i} = n$ et $v_{2i+1} = 1$ pour tout $i \leq n$. On observe que la taille de x_n croit bien avec n.

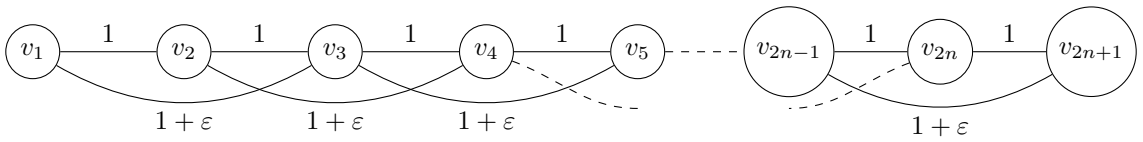
L'algorithme Next Fit va renvoyer une solution y en $2n$ sacs, un par objet, puisque deux objets consécutifs ne peuvent être mis dans le même sac. Par contre la solution optimale consiste à mettre tous les objets v_{2i+1} dans le même sac et les n autres dans n sacs.

$$\frac{\mathcal{M}(x_n,y)}{r(x_n)\cdot\mathcal{M}^*(x_n)} = \frac{2n}{2(n+1)} \to_{n\to+\infty} 1$$

. □

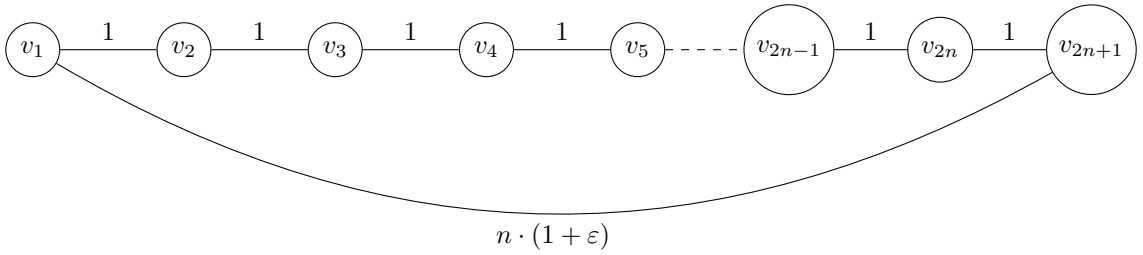
Théorème 4.1.3. *L'algorithme de Christofides a un rapport serré égal à* $\frac{3}{2}$.

Démonstration. Posons x_n l'instance suivante, où le graphe est complet mais où toutes les arêtes n'ont pas été dessinées. Ces arêtes ont un poids égal au plus court chemin dans le graphe, respectant ainsi le principe de l'inégalité triangulaire.

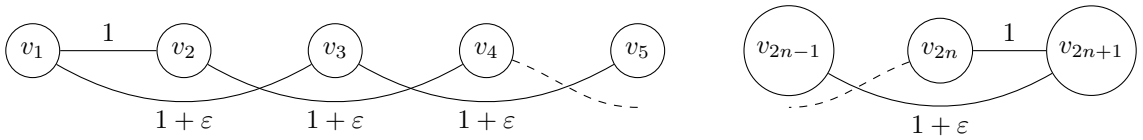


La taille de x_n croit bien strictement avec n.

L'algorithme de Christofides va calculer un arbre couvrant de poids minimum, consistant à prendre la chaîne $(v_1, v_2, \ldots, v_{2n+1})$. Cette chaîne a 2 nœuds de degré impair, v_1 et v_{2n+1}, reliés par une arête de poids égal au poids d'un plus court chemin reliant ces deux nœuds. Ce plus court chemin consiste à sauter les nœuds 2 par 2 avec les arêtes de poids $1+\varepsilon$, égal à $n \cdot (1+\varepsilon)$. Le couplage de poids minimum reliant ces deux nœuds est donc cette arête. Le cycle eulérien y renvoyé par l'algorithme est donc un cycle de poids $2n + n \cdot (1+\varepsilon)$.



La solution optimale consiste à relier les nœuds d'indice impair puis les nœuds d'indice pair :
$(v_1, v_3, v_5, \ldots, v_{2n+1}, v_{2n}, v_{2n-2}, v_{2n-4}, \ldots, v_2, v_1)$. On a donc n arêtes de poids $1+\varepsilon$ pour relier les nœuds d'incide impair, $n-1$ arêtes de poids $1+\varepsilon$ pour relier les nœuds d'indice pair et 2 arêtes de poids 1 pour relier ces deux chaînes. Soit un poids total de $(2n-1) \cdot (1+\varepsilon) + 2$.



Posons enfin $\varepsilon = \frac{1}{n}$.

$$\frac{\mathcal{M}(x_n, y)}{r(x_n) \cdot \mathcal{M}^*(x_n)} = \frac{2n + n \cdot (1+\varepsilon)}{\frac{3}{2}((2n-1) \cdot (1+\varepsilon) + 2)} = \frac{2n + n \cdot (1+\frac{1}{n})}{\frac{3}{2}((2n-1) \cdot (1+\frac{1}{n}) + 2)} \rightarrow_{n \rightarrow +\infty} 1$$

.

□

Il est possible que l'on n'arrive pas à démontrer qu'un rapport est serré, mais on démontre une forme relâchée, à une constante près [2].

Définition 27. Soit un problème $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O})$ de minimisation et $\mathcal{A}$ une r-approximation polynomiale résolvant Π. Le rapport r est dit serré, à une constante près, s'il existe une constante L et une suite d'instances $(x_n)_{n \in \mathbb{N}} \in \mathcal{I}^{\mathbb{N}}$ telles que $|x_n|$ croit strictement avec n et

$$\lim_{n \rightarrow +\infty} \frac{\mathcal{M}(x_n, \mathcal{A}(x_n))}{r(x_n)\mathcal{M}^*(x_n)} = L$$

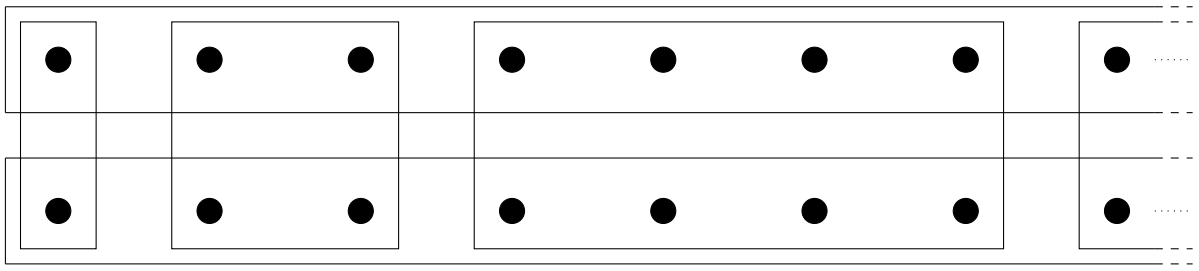
On a $L < 1$ si Π est un problème de minimisation et $L > 1$ sinon.

On a vu pour le problème de couverture par ensemble que, lorsque tous les poids sont égaux à 1, l'approximation gloutonne a un rapport $\ln(|X|/\omega(C^*)) + 1$ où $\omega(C^*)$ est le poids d'une solution optimale.

Théorème 4.1.4. *Dans les instances où tous les poids sont égaux à 1, l'algorithme de Johnson, Lovàsz et Chvàtal a un rapport serré (à une constante près) égal à* $\ln(|X|/\omega(C^*)) + 1$

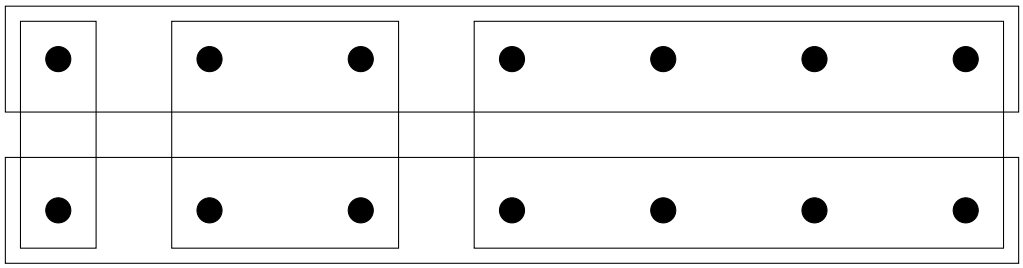
Démonstration. Le rapport est prouvé par le théorème 1.3.1. Considérons les instances suivantes :

2. Comme pour le rapport serré, il n'existe pas vraiment de définition claire de rapport serré à une constante près, mais ce sont des notions implicites que l'on retrouve dans la littérature.



On a donc 2 lignes contenant $1 + 2 + 4 + 8 + \cdots + 2^{p-1}$ éléments dans X, pour un entier $p > 0$ constant. Donc en tout, $2 * (2^p - 1)$ éléments. Il y a $p + 2$ ensembles dans S, un ensemble contenant tous les éléments du haut, nommé A, un autre contenant tous les éléments du bas, nommé B, puis p ensembles $C_1, C_2, \ldots, C_p$, qui contiennent respectivement les 2 éléments de gauche, puis les 4 suivants, puis les 8 suivants, puis ..., puis les 2^p éléments à droite. On note x_p cette instance.

Par exemple, si $p = 3$, on a cette instance là :



La solution optimale consiste à prendre A et B, donc 2 ensembles, $\omega(C^*) = 2$.

Pourtant l'algorithme glouton va choisir $C_1, C_2, \ldots, C_p$. En effet, au début d'algorithme, A contient $2^p - 1$ éléments, B aussi, mais C_p en contient 2^p. Donc on ajoute C_p à la couverture et on retire tous ses éléments de X. Alors A et B contiennent $2^{p-1} - 1$ éléments mais C_{p-1} en contient 2^{p-1}. Donc on ajoute C_{p-1} à la couverture et on retire tous ses éléments de X. Et ainsi de suite, jusqu'à ce qu'il ne reste que A, B et C_1 auquel car l'algorithme rajoutera C_1 à la couverture.

On a donc p ensembles dans la solution contre 2 dans une solution optimale. On rappelle que $\ln(2^p) = p\ln(2)$.

$$\lim_{p\to+\infty} \frac{\mathcal{M}(x_p, \mathcal{A}(x_p))}{r(x_p)\mathcal{M}^*(x_p)} = \lim_{p\to+\infty} \frac{p}{\left(\ln(\frac{2\cdot(2^p-1)}{2}) + 1\right)\cdot 2} = \lim_{p\to+\infty} \frac{p}{\left(p\ln(2) + \ln(1 - \frac{1}{2^p}) + 1\right)\cdot 2} = \frac{1}{2\ln(2)}$$

□

On a donc bien la limite recherchée à une constante près. Cela signifie que, si on ne tient compte de ces instances, on a bien un rapport logarithmique, mais il est possible que l'algorithme soit en fait une $\frac{1}{2\ln(2)}(\ln(k/\omega(C^*)) + 1)$ approximation polynomiale. Soit notre analyse, dans la preuve du théorème 1.3.1 n'est pas assez poussée, soit on n'a pas choisi les bonnes instances pour prouver que le rapport est serré [3].

4.2 Réductions préservant les rapports d'approximations

La réduction de Karp permet de dire qu'un problème de décision est *plus facile à résoudre* qu'un autre, dans le sens où disposer d'un algorithme pour résoudre le second en temps polynomial permettrait de disposer d'un algorithme pour résoudre le premier en temps polynomial. On va s'intéresser ici à une généralisation de ce concept : définir des réductions qui permettent de prouver qu'un problème est *plus approximable* qu'un autre, dans le sens où si on dispose d'une approximation polynomiale pour le second, alors on peut construire une approximation polynomiale pour le premier. Il existe de très nombreuses réductions, on va en exhiber quelques unes.

3. Il semblerait que la première hypothèse soit la bonne, d'après `https://www.sciencedirect.com/science/article/pii/S0196677497908877`, cet algorithme aurait un rapport serré de $\ln(|X|) - \ln\ln(|X|) + \Theta(1)$

Réduction L

La réduction L a plusieurs avantages : sa définition ne change pas que Π et Π' soient des problèmes de minimisation ou de maximisation et elle préserve l'appartenance aux classes APX et PTAS.

Définition 28. Soient $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O})$ et $\Pi' = (\mathcal{I}', \mathcal{S}', \mathcal{M}', \mathcal{O}')$ deux problèmes de NPO.

Une réduction L de Π vers Π', notée $\Pi \leq_L \Pi'$, est une double transformation polynomiale, soit deux fonctions f, g calculables en temps polynomial, et deux réels α, β :

— $f : \mathcal{I} \to \mathcal{I}'$, avec $\mathcal{M}'^*(f(x)) \leq \alpha \mathcal{M}^*(x)$

— $g : \mathcal{S}'(f(x)) \to \mathcal{S}(x)$, avec $|\mathcal{M}(x, g(y')) - \mathcal{M}^*(x)| \leq \beta \cdot |\mathcal{M}'(f(x), y') - \mathcal{M}'^*(f(x))|$

Un point important qui a été omis dans la définition précédente est la notion d'instance sans solution réalisable. Dans ce cas, la notation $\mathcal{M}^*$ n'a pas de sens et la fonction g non plus puisqu'elle ne peut renvoyer de solution, enfin, l'approximabilité n'a pas de sens non plus. On supposera donc que pour ces cas là, $f(x)$ renvoie une instance de Π' sans solution réalisable. On supposera qu'il est possible de savoir si une instance de Π ou de Π' possède des solutions réalisables est un problème polynomial. Dans le cas contraire, il est souvent assez simple de montrer qu'un tel problème n'a pas d'approximation polynomial quel que soit le rapport d'approximation recherché. Ce point sera rediscuté ultérieurement dans ce chapitre.

Théorème 4.2.1. *Soient $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O})$ et $\Pi' = (\mathcal{I}', \mathcal{S}', \mathcal{M}', \mathcal{O}')$ deux problèmes de NPO tels que $\Pi \leq_L \Pi'$, avec la réduction L constituée de (f, g, α, β).*

Si $\mathcal{O} = \min$, $\mathcal{O}' = \min$, et si Π' est ρ-approximable $(\rho > 1)$ alors Π est $1 + \alpha\beta(\rho - 1)$-approximable.
Si $\mathcal{O} = \max$, $\mathcal{O}' = \max$, et si Π' est ρ-approximable $(\rho < 1)$ alors Π est $1 - \alpha\beta(1 - \rho)$-approximable.
Si $\mathcal{O} = \min$, $\mathcal{O}' = \max$, et si Π' est ρ-approximable $(\rho < 1)$ alors Π est $1 + \alpha\beta(1 - \rho)$-approximable.
Si $\mathcal{O} = \max$, $\mathcal{O}' = \min$, et si Π' est ρ-approximable $(\rho > 1)$ alors Π est $1 - \alpha\beta(\rho - 1)$-approximable.

On peut remarquer que les deux premiers rapports et les deux derniers sont les mêmes, mais du fait du signe de $\rho - 1$, ils ont été réécrits pour être mis sous une forme (1− nombre positif) ou (1+ nombre positif).

Démonstration. Supposons qu'il existe une ρ-approximation polynomiale $\mathcal{A}'$ pour Π' alors, quelque soit le cas, on construit un algorithme heuristique pour Π avec $\mathcal{B} = g \circ \mathcal{A} \circ f$.

Soit $x \in \mathcal{I}$, $x' = f(x) \in \mathcal{I}'$, $y' = \mathcal{A}'(x') \in \mathcal{S}'(x')$ et $y = g(y') = \mathcal{B}(x) \in \mathcal{S}(x)$. On vérifie donc les hypothèses suivantes :

$$\begin{aligned} \mathcal{M}'^*(x') &\leq \alpha \mathcal{M}^*(x) \\ \mathcal{M}'(x', y') &\leq \rho \mathcal{M}'^*(x') \text{ si } \mathcal{O}' = \min \\ \mathcal{M}'(x', y') &\geq \rho \mathcal{M}'^*(x') \text{ si } \mathcal{O}' = \max \\ |\mathcal{M}(x, y) - \mathcal{M}^*(x)| &\leq \beta \cdot |\mathcal{M}'(x', y') - \mathcal{M}'^*(x')| \end{aligned}$$

Si Π et Π' sont tous les deux des problèmes de minimisation, alors $\rho \geq 1$

$$\begin{aligned} \mathcal{M}(x, y) - \mathcal{M}^*(x) &\leq \beta \cdot (\mathcal{M}'(x', y') - \mathcal{M}'^*(x')) \\ \mathcal{M}(x, y) - \mathcal{M}^*(x) &\leq \beta \cdot (\rho - 1)\mathcal{M}'^*(x') \\ \mathcal{M}(x, y) &\leq \beta \cdot (\rho - 1) \cdot \alpha \mathcal{M}^*(x) + \mathcal{M}^*(x) \end{aligned}$$

Si Π et Π' sont tous les deux des problèmes de maximisation, alors $\rho \leq 1$

$$\begin{aligned} \mathcal{M}(x, y) - \mathcal{M}^*(x) &\geq \beta \cdot (\mathcal{M}'(x', y') - \mathcal{M}'^*(x')) \\ \mathcal{M}(x, y) - \mathcal{M}^*(x) &\geq \beta \cdot (\rho - 1)\mathcal{M}'^*(x') \\ \mathcal{M}(x, y) &\geq \beta \cdot (\rho - 1) \cdot \alpha \mathcal{M}^*(x) + \mathcal{M}^*(x) \end{aligned}$$

Si Π est un problème de minimisation et Π' est un problème de maximisation, alors $\rho \leq 1$

$$\begin{aligned} \mathcal{M}(x, y) - \mathcal{M}^*(x) &\leq \beta \cdot (\mathcal{M}'^*(x') - \mathcal{M}'(x', y')) \\ \mathcal{M}(x, y) - \mathcal{M}^*(x) &\leq \beta \cdot (1 - \rho) \cdot \mathcal{M}'^*(x') \\ \mathcal{M}(x, y) &\leq \beta \cdot (1 - \rho)\alpha \cdot \mathcal{M}^*(x) + \mathcal{M}^*(x) \end{aligned}$$

Si Π est un problème de maximisation et Π' est un problème de minimisation, alors $\rho \geq 1$

$$\begin{aligned}\mathcal{M}(x,y) - \mathcal{M}^*(x) &\geq \beta \cdot (\mathcal{M'}^*(x') - \mathcal{M}'(x',y')) \\ \mathcal{M}(x,y) - \mathcal{M}^*(x) &\geq \beta \cdot (1-\rho) \cdot \mathcal{M'}^*(x') \\ \mathcal{M}(x,y) &\geq \beta \cdot (1-\rho)\alpha \cdot \mathcal{M}^*(x) + \mathcal{M}^*(x)\end{aligned}$$

□

Remarque 13. Attention, on ne suppose pas que ρ est constant, il peut dépendre de x', donc, il faut, par exemple, lire la formule $1 + \alpha\beta(\rho - 1)$ comme $1 + \alpha\beta(\rho(f(x)) - 1)$. De même, la définition impose que α et β soient constants mais on peut très bien imaginer la même démonstration avec des fonctions.

Théorème 4.2.2. *Si $\Pi \in$ NPO et $\Pi' \in$ APX et $\Pi \leq_L \Pi'$ alors $\Pi \in$ APX*

Démonstration. Si ρ est constant alors $1+\alpha\beta(\rho-1)$ et $1+\alpha\beta(1-\rho)$ le sont également. Donc le théorème 4.2.1 le prouve que $\Pi \in$ APX. □

Théorème 4.2.3. *Si $\Pi \in$ NPO et $\Pi' \in$ PTAS et $\Pi \leq_L \Pi'$, alors $\Pi \in$ PTAS*

Démonstration. Soit $\varepsilon > 0$. Posons $\varepsilon' = \frac{\varepsilon}{\alpha\beta}$.

Supposons que $\mathcal{O} = \mathcal{O}' = \min$. Puisque $\Pi' \in$ PTAS, alors il existe une approximation de rapport $(1+\varepsilon')$ pour Π'. D'après le Théorème 4.2.1, il existe une approximation de rapport $1 + \alpha\beta\varepsilon' = 1 + \varepsilon$ pour Π.

Supposons que $\mathcal{O} = \mathcal{O}' = \max$. Puisque $\Pi' \in$ PTAS, alors il existe une approximation de rapport $(1-\varepsilon')$ pour Π'. D'après le Théorème 4.2.1, il existe une approximation de rapport $1 - \alpha\beta\varepsilon' = 1 - \varepsilon$ pour Π.

Supposons que $\mathcal{O} = \min$ et $\mathcal{O}' = \max$. Puisque $\Pi' \in$ PTAS, alors il existe une approximation de rapport $(1-\varepsilon')$ pour Π'. D'après le Théorème 4.2.1, il existe une approximation de rapport $1 + \alpha\beta\varepsilon' = 1 + \varepsilon$ pour Π.

Supposons que $\mathcal{O} = \max$ et $\mathcal{O}' = \max$. Puisque $\Pi' \in$ PTAS, alors il existe une approximation de rapport $(1-\varepsilon')$ pour Π'. D'après le Théorème 4.2.1, il existe une approximation de rapport $1 - \alpha\beta\varepsilon' = 1 - \varepsilon$ pour Π. □

Prenons pour exemple les problèmes Max-Stable-B et Max-2SAT.

Problème : Problème Max-Stable-B

Instance :
— Un graphe non orienté $G = (V, E)$ tel que le degré maximum des nœuds est B.

Solution réalisable : Un stable S de G

Optimisation : Maximiser $|S|$.

Problème : Problème Max-2SAT

Instance :
— m clauses disjonctives $C_1, C_2, \ldots, C_m$ et n variables $x_1, x_2, \ldots, x_n$.
$C_j = (l_1^j \vee l_2^j)$, où l_i^j est un litéral x_k ou $\bar{x_k}$.

Solution réalisable : Une affectation des variables à vrai ou faux, c'est-à-dire $a_1, a_2, \ldots, a_n \in \{\top, \bot\}$

Optimisation : Maximiser le nombre de clauses satisfaites. Si on note $a(C_j)$ la clause C_j où chaque variable x_i est remplacée par a_i, alors C_j est satisfaite si $a(C_j) = \top$. On veut donc maximiser $|a| = |\{a(C_j) = \top, j \in [\![1, ; m]\!]\}$.

Théorème 4.2.4. *Max-StableB $\leq_L$ Max-2SAT*

Démonstration. Soit une instance G de Max-StableB. On crée une instance $f(G)$ de Max-2SAT avec $|V|$ variables et $|V| + |E|$ clauses. Pour chaque nœud $v \in V$ de G, on crée une variable x_v et une clause $C_v = (x_v \vee x_v)$. Pour chaque arête $e = (u, v) \in E$, on crée une clause $C_e = (\bar{x_u} \vee \bar{x_v})$.

Soit une solution réalisable $(a_v, v \in V) \cup (a_e, e \in E)$ de $f(G)$. Pour toute arête $e = (u, v) \in E$, si C_e n'est pas vérifiée, alors $a_u = a_v = \top$. On remplace alors a_u par $\bot$. Au pire une clause devient fausse : C_v et au moins

une clause devient vraie : C_e. Donc le nombre de clauses vraies augmente. Notons a' cette nouvelle solution réalisable de $f(G)$. Soit $g(a) = S = \{v | a'_v = \top\}$, alors S est un stable. En effet, toute clause $C_e, e \in E$ est vérifiée, donc au plus un nœud par arête a une variable associée qui est vraie, donc qui est dans le stable. On peut remarquer que $|a| \leq |a'| = |E| + |S| = |E| + |g(a)|$.

Notons ici que f et g se calculent en temps polynomial.

Posons S^* un stable maximum et a^* une affectation optimale des variables. Alors $|a^*| \leq |a^{*\prime}| = |E|+|g(a^*)| \leq |E|+|S^*|$. Remarquons maintenant qu'on peut, avec S^* construire une affectation a satisfaisant $|E|+|S^*|$ clauses en mettant à vrai toute variable x_v telle que $v \in S^*$ et fausse les autres. Donc $|E| + |S^*| \leq |a| \leq |a^*|$. Donc $|a^*| = |E| + |S^*|$.

G a un degré maximum égal à B, donc $|E| \leq \frac{|V|B}{2}$ et $|S^*| \geq \frac{|V|}{B+1}$. Donc $|E| \leq \frac{(B+1)B|S^*|}{2}$.
Donc $|a^*| = |E| + |S^*| \leq (1 + \frac{(B+1)B}{2})|S^*|$. On a donc $|a^*| \leq \alpha|S^*|$ avec $\alpha = (1 + \frac{(B+1)B}{2})$.

Enfin, soit a une solution réalisable et $S = g(a)$. Alors $|S^*| - |S| = (|E| + |S^*|) - (|E| + |S|) \leq |a^*| - |a|$. Donc $||S^*| - |S|| \leq \beta||a^*| - |a||$ avec $\beta = 1$.

On a donc bien Max-StableB $\leq_L$ Max-2SAT. □

Cette réduction prouve que le problème Max-2SAT est au moins aussi difficile à approcher que le problème Max-StableB. On sait par exemple qu'il existe $\varepsilon > 0$ tel que, pour tout $B \geq 3$, Max-StableB ne peut être approché avec un rapport inférieur à $B^{-\varepsilon}$ [4]. Ainsi, on peut donc déduire, grâce à la réduction L, que, pour tout $B \geq 3$, Max-2SAT ne peut être approché avec un rapport meilleur que $1 - \frac{1-B^{-\varepsilon}}{1+\frac{B\cdot(B+1)}{2}}$. Plus concrètement, on sait aussi que si $B = 5$, on ne peut approcher Max-StableB avec un rapport meilleur que $\frac{1}{1.0030} < 0.998$. Donc Max-2SAT ne peut être approché ave un rapport meilleur que $1 - \frac{1-0.998}{1+5\cdot 6/2} < 0.9999$. Ces rapports, mêmes très proches de 1, nous prouvent entre autres qu'il n'existe pas de PTAS pour Max-2SAT.

Réduction isofacteur

La réduction isofacteur est un autre type de réduction qui, comme la réduction L, préserve l'appartenance à la classe APX et partiellement à la classe PTAS. Le principe est similaire.

Définition 29. Soient $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O})$ et $\Pi' = (\mathcal{I}', \mathcal{S}', \mathcal{M}', \mathcal{O}')$ deux problèmes de NPO.

Une réduction isofacteur de Π vers Π', notée $\Pi \leq_I \Pi'$, est une double transformation polynomiale $f : \mathcal{I} \to \mathcal{I}'$ et $g : \mathcal{S}'(f(x)) \to \mathcal{S}(x)$ et deux réels α, β tels que :

$$\text{Si } \mathcal{O} = \min, \mathcal{O}' = \min, \text{ alors } \mathcal{M}'^*(f(x)) \leq \alpha\mathcal{M}^*(x) \text{ et } \mathcal{M}(x, g(y')) \leq \beta \cdot \mathcal{M}'(f(x), y')$$
$$\text{Si } \mathcal{O} = \max, \mathcal{O}' = \max, \text{ alors } \mathcal{M}'^*(f(x)) \geq \alpha\mathcal{M}^*(x) \text{ et } \mathcal{M}(x, g(y')) \geq \beta \cdot \mathcal{M}'(f(x), y')$$
$$\text{Si } \mathcal{O} = \min, \mathcal{O}' = \max, \text{ alors } \mathcal{M}'^*(f(x)) \geq \frac{1}{\alpha\mathcal{M}^*(x)} \text{ et } \mathcal{M}(x, g(y')) \leq \beta \cdot \frac{1}{\mathcal{M}'(f(x), y')}$$
$$\text{Si } \mathcal{O} = \max, \mathcal{O}' = \min, \text{ alors } \mathcal{M}'^*(f(x)) \leq \frac{1}{\alpha\mathcal{M}^*(x)} \text{ et } \mathcal{M}(x, g(y')) \geq \beta \cdot \frac{1}{\mathcal{M}'(f(x), y')}$$

Théorème 4.2.5. *Si Π' est ρ-approximable alors*
- *si $\mathcal{O} = \mathcal{O}'$, alors Π est $\alpha\beta\rho$-approximable.*
- *si $\mathcal{O} \neq \mathcal{O}'$, alors Π est $\frac{\alpha\beta}{\rho}$-approximable.*

Démonstration. Supposons qu'il existe une ρ-approximation $\mathcal{A}$ polynomiale pour Π'.

Supposons qu'il existe une ρ-approximation polynomiale $\mathcal{A}'$ pour Π' alors, quelque soit le cas, on construit un algorithme heuristique pour Π avec $\mathcal{B} = g \circ \mathcal{A} \circ f$.

Soit $x \in \mathcal{I}$, $x' = f(x) \in \mathcal{I}'$, $y' = \mathcal{A}'(x) \in \mathcal{S}'(x')$ et $y = g(y') = \mathcal{B}(x) \in \mathcal{S}(x)$.

Si Π et Π' sont tous les deux des problèmes de minimisation, alors

$$\mathcal{M}(x, y) \leq \beta\mathcal{M}'(x', y')$$
$$\mathcal{M}(x, y) \leq \rho\beta\mathcal{M}'^*(x')$$
$$\mathcal{M}(x, y) \leq \rho\beta\alpha\mathcal{M}^*(x)$$

4. https://www.csc.kth.se/~viggo/wwwcompendium/node34.html

Si Π et Π' sont tous les deux des problèmes de maximisation, alors

$$\mathcal{M}(x,y) \geq \beta \mathcal{M}'(x',y')$$
$$\mathcal{M}(x,y) \geq \rho\beta \mathcal{M}'^{*}(x')$$
$$\mathcal{M}(x,y) \geq \rho\beta\alpha \mathcal{M}^{*}(x)$$

Si Π est un problème de minimisation et Π' est un problème de maximisation, alors

$$\mathcal{M}(x,y) \leq \beta \frac{1}{\mathcal{M}'(x',y')}$$
$$\mathcal{M}(x,y) \leq \beta \frac{1}{\rho\mathcal{M}'^{*}(x')}$$
$$\mathcal{M}(x,y) \leq \frac{\beta\alpha}{\rho} \mathcal{M}^{*}(x)$$

Si Π est un problème de maximisation et Π' est un problème de minimisation, alors

$$\mathcal{M}(x,y) \geq \beta \frac{1}{\mathcal{M}'(x',y')}$$
$$\mathcal{M}(x,y) \geq \beta \frac{1}{\rho\mathcal{M}'^{*}(x')}$$
$$\mathcal{M}(x,y) \geq \frac{\beta\alpha}{\rho} \mathcal{M}^{*}(x)$$

□

Remarque 14. Attention, comme pour la réduction L, on ne suppose pas que ρ est constant, il peut dépendre de x', donc, il faut, par exemple, lire la formule $\alpha\beta\rho$ comme $\alpha\beta\rho(f(x))$. De même, la définition impose que α et β soient constants mais on peut très bien imaginer la même démonstration avec des fonctions.

Théorème 4.2.6. *Si* $\Pi \in$ *NPO et* $\Pi' \in$ *APX et* $\Pi \leq_I \Pi'$ *alors* $\Pi \in$ *APX*

Démonstration. Si ρ est une constante alors $\alpha\beta\rho$ et $\frac{\alpha\beta}{\rho}$ sont constantes aussi. □

Théorème 4.2.7. *Si* $\Pi \in$ *NPO et* $\Pi' \in$ *PTAS et* $\Pi \leq_I \Pi'$ *avec* $\alpha = \beta = 1$ *de* Π *vers* Π' *alors* $\Pi \in$ *PTAS*

Démonstration. Soit $\varepsilon > 0$.

Supposons que $\mathcal{O} = \mathcal{O}' = \min$. Posons $\varepsilon' = \varepsilon$. Puisque $\Pi' \in$ PTAS, alors il existe une approximation de rapport $(1+\varepsilon')$ pour Π'. D'après le Théorème 4.2.5, il existe une approximation de rapport $1+\varepsilon' = 1+\varepsilon$ pour Π.

Supposons que $\mathcal{O} = \mathcal{O}' = \max$. Posons $\varepsilon' = \varepsilon$. Puisque $\Pi' \in$ PTAS, alors il existe une approximation de rapport $(1-\varepsilon')$ pour Π'. D'après le Théorème 4.2.5, il existe une approximation de rapport $1-\varepsilon' = 1-\varepsilon$ pour Π.

Supposons que $\mathcal{O} = \min$ et $\mathcal{O}' = \max$. Posons $\varepsilon' = \min(\frac{1}{2}, \frac{\varepsilon}{2})$. Puisque $\Pi' \in$ PTAS, alors il existe une approximation de rapport $(1-\varepsilon')$ pour Π'. D'après le Théorème 4.2.5, il existe une approximation de rapport $\frac{1}{1-\varepsilon'}$ pour Π. Or $\frac{1}{1-\varepsilon'} \leq 1+2\varepsilon'$ puisque $\varepsilon' \leq \frac{1}{2}$. Donc il existe une $1+2\varepsilon' \leq 1+\varepsilon$ approximation pour Π.

Supposons que $\mathcal{O} = \max$ et $\mathcal{O}' = \min$. Posons $\varepsilon' = \varepsilon$. Puisque $\Pi' \in$ PTAS, alors il existe une approximation de rapport $(1+\varepsilon')$ pour Π'. D'après le Théorème 4.2.5, il existe une approximation de rapport $\frac{1}{1+\varepsilon'}$ pour Π. Or $\frac{1}{1+\varepsilon'} \geq 1-\varepsilon'$. Donc il existe une $1-\varepsilon$ approximation pour Π. □

Prenons pour exemple les problèmes de couverture par ensembles et de l'arborescence de Steiner.

Problème : Problème de l'arborescence de Steiner (Directed Steiner Tree en anglais)

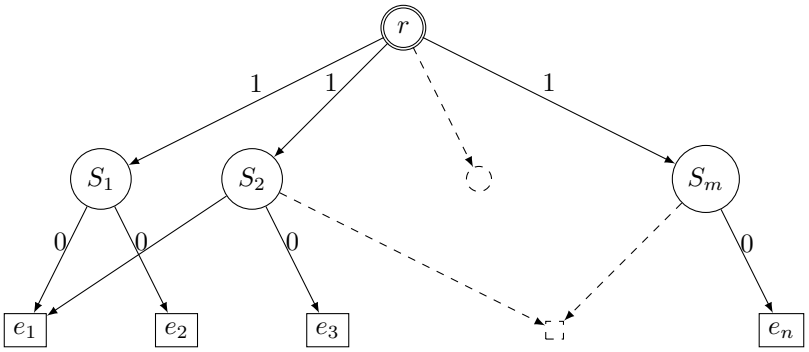
Instance :
- Un graphe orienté $G = (V, A)$ contenant n nœuds et m arêtes.
- Un nœud $r \in V$ appelé *racine*
- Un ensemble de k nœuds $X \subset V$ appelés *terminaux*.
- Une fonction de poids $\omega : A \to \mathbb{N}$.

Solution réalisable : Une sous-arborescence T de G enracinée en r et couvrant chaque terminal de X.

Optimisation : Minimiser le poids $\omega(T) = \sum_{a \in T} \omega(a)$.

Théorème 4.2.8. *Set Cover $\leq_I$ Directed Steiner Tree*

Démonstration. Soit une instance $(\mathcal{X}, \mathcal{S})$ de Set Cover où $\mathcal{X}$ et $\mathcal{S}$ sont respectivement les éléments de l'instance et les ensembles qui permettent de les couvrir. On construit une instance $(G, r, X, \omega) = f(\mathcal{X}, \mathcal{S})$ de l'arborescence de Steiner ainsi : G est un graphe contenant un nœud r, un nœud v_S pour chaque ensemble $S \in \mathcal{S}$ et un nœud v_e pour chaque élément $e \in \mathcal{X}$. On relie r à tous les nœuds v_S et, pour tout $S \in \mathcal{S}$, on relie v_S à v_e si $e \in S$. Enfin, on pose $\omega(r, v_S) = 1$ pour tout ensemble S et $\omega(v_S, v_e) = 0$ pour tout $e \in S$. Enfin, on pose $X = \{v_e, e \in \mathcal{X}\}$.



Connaissant une couverture C de $\mathcal{X}$, alors on peut construire une arborescence enracinée en r et couvrant X de poids $|C|$ en utilisant tous les arcs (r, v_S) pour $S \in C$ et, pour chaque $e \in \mathcal{X}$, un arc (v_S, v_e) où $S \in C$ et $e \in S$. On sait qu'un tel arc existe puisque C couvre $\mathcal{X}$. Inversement, connaissant une arborescence T couvrant $\mathcal{X}$, enracinée en r de poids $\omega(T)$, on peut construire une couverture C de $\mathcal{X}$ contenant $\omega(T)$ ensembles en sélectionnant tous les ensembles S tels que $(r, v_S) \in T$. On sait que C est une couverture car, pour tout $v_e \in X$, il existe un chemin de r vers v_e dans T, et donc un chemin de r vers le prédécesseur de v_e dans ce chemin. Soit v_S un tel prédécesseur, le seul chemin qui relie r à v_S est l'arc (r, v_S). Donc $S \in C$. Or il existe un arc reliant v_S à v_e donc $e \in S$. On pose $g(T) = C$ cette dernière construction.

Notons ici que f et g se calculent en temps polynomial.

Des deux constructions du paragraphe précédent, on peut déduire que le poids $|C^*|$ d'une solution optimale de $(\mathcal{X}, \mathcal{S})$ est égal au poids d'une solution optimale $\omega(T^*)$ de $f(\mathcal{X}, \mathcal{S})$. On vérifie donc $\omega(T^*) \leq \alpha|C^*|$ avec $\alpha = 1$. De plus pour toute solution réalisable T de $f(\mathcal{X}, \mathcal{S})$, on a $\omega(T) = |g(T)|$ donc $|g(T)| \leq \beta\omega(T)$ avec $\beta = 1$. On a donc bien une réduction isofacteur entre le problème de Couverture par ensemble et celui de l'arborescence de Steiner. □

Cette réduction prouve que le problème de l'arborescence de Steiner est au moins aussi difficile à approcher que le problème de couverture par ensembles. On sait par exemple qu'il existe une constante c tel que le problème de couverture par ensembles ne peut être approché avec un rapport meilleur que $c \log(k)$ où k est le nombre d'éléments à couvrir. On peut donc en déduire de la réduction précédente que le problème de l'arborescence de Steiner ne peut être approché avec ce même rapport où k est le nombre de terminaux.

Réduction GAP

La réduction GAP est une réduction clairement différente des autres puisqu'elle se base sur le fait que savoir approcher un problème avec un certain rapport permettrait de résoudre en temps polynomial un problème NP-Complet. On conclue donc l'impossibilité d'approcher certains problèmes sauf si P = NP. C'est une réduction qu'on a déjà aperçue à plusieurs reprises dans le cours, avec les théorèmes 1.2.1 et 2.2.2.

On rappelle pour la suite qu'un problème de décision peut être défini avec un ensemble $\mathcal{L}$ d'instances et une partition de ces instances en deux sous-ensembles $\mathcal{L}^Y$ et $\mathcal{L}^N$ qui sont les instances respectivement positives et négatives.

Définition 30. Il existe une GAP-Réduction du problème de **décision** $\Pi = (\mathcal{L}, \mathcal{L}^Y, \mathcal{L}^N)$ NP-Complet vers le problème $\Pi' = (\mathcal{I}', \mathcal{S}', \mathcal{M}', \mathcal{O}')$ s'il existe deux réels α et g et une transformation polynomiale $f : \mathcal{I} \to \mathcal{I}'$ tels que :

- Si $\mathcal{O}' = \min$,
  - si $x \in \mathcal{L}^Y$, $\mathcal{M}'^*(f(x)) < g(x)$;
  - si $x \in \mathcal{L}^N$, pour tout $y \in \mathcal{S}'(f(x))$, $\mathcal{M}'(f(x), y) \geq \alpha \cdot g$
- Si $\mathcal{O}' = \max$,
  - si $x \in \mathcal{L}^Y$, $\mathcal{M}'^*(f(x)) > g(x)$;
  - si $x \in \mathcal{L}^N$, pour tout $y \in \mathcal{S}'(f(x))$, $\mathcal{M}'(f(x), y) \leq \alpha \cdot g$

Remarque 15. Comme pour les autres réductions, on peut tout à fait imaginer cette définition avec α et g qui sont des fonctions.

Théorème 4.2.9. *S'il existe une GAP-Réduction (f, α, g) de Π vers Π' alors Π' ne peut être approché avec un rapport α si $P \neq NP$.*

Démonstration. Supposons qu'il existe un algorithme d'approximation polynomial $\mathcal{A}$ de rapport α pour Π'. Soit une instance x de Π. Construisons l'algorithme $\mathcal{B}$ suivant : il calcule $x' = f(x)$ puis $y' = \mathcal{A}(x')$. Si $\mathcal{O}' = \min$, il répond ensuite OUI ou NON selon que $\mathcal{M}'(x', y') < \alpha \cdot g$ ou non. Si $\mathcal{O}' = \max$, il répond OUI ou NON selon que $\mathcal{M}'(x', y') > \alpha \cdot g$ ou non. On peut noter que $\mathcal{B}$ est un algorithme polynomial.

- Si $\mathcal{O} = \min$, et si x est positif alors $\mathcal{M}'^*(x') < g$, donc $\mathcal{M}'(x', y') \leq \alpha \mathcal{M}'^*(x') < \alpha \cdot g$. Donc, l'algorithme $\mathcal{B}(x)$ répond OUI. Sinon $\mathcal{M}'(x', y') \geq \alpha \cdot g$ et $\mathcal{B}(x)$ répond NON.
- Si $\mathcal{O} = \max$, et si x est positif alors $\mathcal{M}'^*(x') > g$, donc $\mathcal{M}'(x', y') \geq \alpha \mathcal{M}'^*(x') > \alpha \cdot g$. Donc l'algorithme $\mathcal{B}(x)$ répond OUI. Sinon $\mathcal{M}'(x', y') \leq \alpha \cdot g$ et $\mathcal{B}(x)$ répond NON.

Dans les deux cas, $\mathcal{B}$ permet de résoudre en temps polynomial le problème Π ce qui n'est pas possible, sauf si P = NP. □

Pour compléter les deux exemples des autres chapitres, prouvons ici un théorème qui prouve l'inapproximabilité du problème de coloration minimum.

Théorème 4.2.10. *Soit K un entier et $\varepsilon \in [0; \frac{K+1}{K} - 1]$. Soit $\Pi \in NPO$ un problème de minimisation tel que le problème suivant Π' soit NP-Complet : connaissant une instance x, est-ce que $\mathcal{M}^*(x) \leq K$?*[5] *Alors il existe une GAP-Réduction $(f, \frac{K+1}{K} - \varepsilon, g)$ de Π' vers Π.*

Remarque 16. Le théorème 4.2.10 s'applique également aux problèmes de maximisation avec une réduction GAP pour laquelle $\alpha = \frac{K}{K+1} + \varepsilon$.

Démonstration. Posons f la fonction identité et $g = \frac{K+1}{\alpha}$. Si x est une instance positive de Π' alors $\mathcal{M}^*(x) \leq K$, et sinon, $\mathcal{M}^*(x) \geq K + 1$. En remarquant que $K = \frac{K+1}{\frac{K+1}{K}} < \frac{K+1}{\alpha} = g$ et que $K + 1 = \alpha g$, on obtient bien la GAP-réduction recherchée. □

Corollaire 4.2.1. *Pour tout $0 < \varepsilon \leq \frac{1}{3}$ Il n'existe pas de $\frac{4}{3} - \varepsilon$ approximation polynomiale pour le problème de coloration minimum.*

Démonstration. On obtient immédiatement ce résultat en utilisant la NP-Complétude du problème de 3-coloration et le théorème 4.2.10. □

Les réductions L, Isofacteur et GAP, ou des variantes de ces réductions, sont celles que l'on retrouve le plus couramment dans la littérature. Elles sont intuitives et permettent de démontrer assez facilement des résultats d'inapproximabilité.

5. On notera qu'il s'agit de Π_D mais la valeur de K est fixée, elle ne dépend pas de l'instance.

Réduction PTAS

Présentons une dernière réduction, nommé réduction PTAS. C'est une réduction qui a été introduite notamment pour créer une réduction qui ne transforme pas nécessairement une solution optimale en solution optimale comme c'est le cas pour la réduction L. L'idée est que, si on ne se restreint pas à de telles réductions, on pourra trouver d'autres relations entre problèmes d'optimisations qui permettront de trouver d'autres résultat d'approximabilité ou d'inapproximabilité. On peut noter que la réduction isofacteur, quand α ou β est différent de 1, n'oblige pas ce lien entre les solutions optimales.

Définition 31. Soient $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O})$ et $\Pi' = (\mathcal{I}', \mathcal{S}', \mathcal{M}', \mathcal{O}')$ deux problèmes de NPO.
Une réduction PTAS de Π vers Π', notée $\Pi \leq_{PTAS} \Pi'$, est une triple transformation f, g, c :
— $f : \mathcal{I} \times]0,1[\to \mathcal{I}'$ avec $f(x, \varepsilon)$ calculable en temps polynomial par rapport à $|x|$
— $g : \mathcal{S}'(f(x)) \times]0,1[\to \mathcal{S}(x)$, avec $g(y', \varepsilon)$ calculable en temps polynomial par rapport à $|y'|$
— $c :]0;1[\to]0;1[$
— Si $\mathcal{O} = \min$, $\mathcal{O}' = \min$, $\frac{\mathcal{M}'(f(x),y')}{\mathcal{M}'^*(f(x))} \leq 1 + c(\varepsilon) \Rightarrow \frac{\mathcal{M}(x,g(y',\varepsilon))}{M^*(x)} \leq 1 + \varepsilon$
— Si $\mathcal{O} = \max$, $\mathcal{O}' = \max$, $\frac{\mathcal{M}'(f(x),y')}{\mathcal{M}'^*(f(x))} \geq 1 - c(\varepsilon) \Rightarrow \frac{\mathcal{M}(x,g(y',\varepsilon))}{M^*(x)} \geq 1 - \varepsilon$
— Si $\mathcal{O} = \min$, $\mathcal{O}' = \max$, $\frac{\mathcal{M}'(f(x),y')}{\mathcal{M}'^*(f(x))} \geq 1 - c(\varepsilon) \Rightarrow \frac{\mathcal{M}(x,g(y',\varepsilon))}{M^*(x)} \leq 1 + \varepsilon$
— Si $\mathcal{O} = \max$, $\mathcal{O}' = \min$, $\frac{\mathcal{M}'(f(x),y')}{\mathcal{M}'^*(f(x))} \leq 1 + c(\varepsilon) \Rightarrow \frac{\mathcal{M}(x,g(y',\varepsilon))}{M^*(x)} \geq 1 - \varepsilon$

Théorème 4.2.11. *Si $\Pi \in$ NPO et $\Pi' \in$ PTAS et $\Pi \leq_{PTAS} \Pi'$ alors $\Pi \in$ PTAS*

Démonstration. Supposons qu'il existe un PTAS $\mathcal{A}_\varepsilon$ pour Π'. Soit $\varepsilon > 0$. Il suffit, pour calculer une $1 + \varepsilon$ approximation d'une instance x de Π, de calculer $x' = f(x, \varepsilon)$, puis de calculer $y' = \mathcal{A}_{c(\varepsilon)}(x')$ et de renvoyer $y = g(y', \varepsilon)$. Par définition de $\mathcal{A}_{c(\varepsilon)}$, la prémisse de la dernière propriété de la réduction PTAS est vérifiée, donc $\mathcal{M}(x, y) \leq (1 + \varepsilon) \cdot \mathcal{M}^*(x)$ si $\mathcal{O} = \min$ et $\mathcal{M}(x, y) \geq (1 - \varepsilon) \cdot \mathcal{M}^*(x)$ si $\mathcal{O} = \max$. □

4.3 Notions de complétude

À l'aide des différentes réductions, on peut utiliser, comme pour la théorie de la complexité classique, les notions de difficulté et de complétude.

Définition 32. Soit $\mathcal{C}$ un classe d'approximabilité et un problème Π, alors Π est $\mathcal{C}$-Difficile sous la réduction $\leq_R$ si, pour tout $\Pi' \in \mathcal{C}$, on a $\Pi' \leq_R \Pi$.

Définition 33. Soit $\mathcal{C}$ un classe d'approximabilité et un problème Π, alors Π est $\mathcal{C}$-Complet sous la réduction $\leq_R$ s'il est $\mathcal{C}$-Difficile sous cette réduction et si $\Pi \in \mathcal{C}$.

Définition 34. Soit $\mathcal{C}$ un classe d'approximabilité, et $\leq_R$ une réduction entre problèmes d'optimisation, alors $\leq_R$ préserve l'appartenance à la classe $\mathcal{C}$ si, pour tout $\Pi \in \mathcal{C}$ et tout problème Π' d'optimisation tel que $\Pi' \leq_R \Pi$ on a $\Pi' \in \mathcal{C}'$.

Cette notion de préservation est importante pour justifier l'intérêt de chercher des problèmes complets. De plus, il servira juste après pour définir les problèmes intermédiaires, c'est-à-dire des problèmes qui sont *entre deux classes.*

Lemme 4.3.1. *Soient $\mathcal{C}$ et $\mathcal{C}'$ des classes d'approximabilité telles que $\mathcal{C}' \subsetneq \mathcal{C}$ et $\leq_R$ une réduction entre problèmes d'optimisation préservant l'appartenance à $\mathcal{C}'$. Soit Π un problème $\mathcal{C}$-Complet sous $\leq_R$, alors $\Pi \notin \mathcal{C}'$.*

Démonstration. Supposons que $\Pi \in \mathcal{C}'$. Pour tout problème Π_2 de $\mathcal{C}$, on a $\Pi_2 \leq_R \Pi$ puisque Π est $\mathcal{C}$-Complet sous $\leq_R$. Puisque cette réduction préserve l'appartenance à $\mathcal{C}'$, on a $\Pi_2 \in \mathcal{C}'$. Donc $\mathcal{C} \subset \mathcal{C}'$, ce qui implique l'égalité entre des deux classes, ce qui est exclu par hypothèse. □

Ce lemme est une manière de prouver qu'un problème n'appartient pas à la classe PTAS. Il *suffit* pour cela de montrer qu'il est dans APX et qu'il est complet pour cette classe sous une réduction préservant l'appartenance à PTAS (la réduction L, la réduction iso-facteur avec $\alpha = \beta = 1$ ou la réduction PTAS).

On peut noter que les 3 définitions et le résultat ci-dessus se généralisent sans problème à des problèmes qui ne sont pas des problèmes d'optimisation. Par exemple, la NP-Complétude classique est définie sous la réduction polynomiale de Karp, et cette réduction préserve l'appartenance à P et à NP (et à toutes les autres classes *classiques* de la théorie de la complexité). Et tout problème NP-Complet n'est pas dans P si P $\subsetneq$ NP.

On va montrer ici l'existence de 2 problèmes, de minimisation et de maximisation qui sont respectivement complets pour la moitié des problèmes de NPO sous toutes les réductions présentées dans ce cours.

À l'instar du problème de satisfiabilité qui a été dans les premiers problèmes prouvés NP-Complet, les problèmes mentionnés ici sont des problèmes de satisfiabilité de poids maximum et minimum.

Problème : Satifiabilité de poids minimum/maximum (Minimum/Maximum Weighted Satisfiability)

Instance :
- Une formule booléenne φ sur n variables $X = \{x_1, x_2, \ldots, x_n\}$
- Des poids $\omega : X \to \mathbb{N}$

Solution réalisable : Une affectation des variables à vrai ou faux qui rende la formule vraie. On note $a_i \in \{1, 0\}$ l'affectation de la variable x_i (1 si x_i est vrai et 0 sinon).

Optimisation : Minimiser/Maximiser le poids de l'affectation : $\sum_{i=1}^{n} \omega(x_i)a_i$

Intéressons nous ici à un point intéressant : il est possible que, pour une instance donnée, il n'y ait pas de solution réalisable. Plus important : la recherche de l'existence d'une solution réalisable dans ce problème est équivalente au problème SAT. Ainsi, si P $\neq$ NP, il n'est pas possible de déterminer en temps polynomial si une telle solution existe. En utilisant une preuve similaire à celle du théorème 1.2.1, on peut montrer qu'il n'existe pas d'approximation polynomiale pour ce problème, quel qu'il soit.

On arrive donc à un petit problème vis-à-vis des définitions de réduction proposées plus haut. Que faire si l'instance de départ ne possède pas de solution réalisable ? On serait alors embêté pour utiliser la fonction de retour g qui transforme les solutions du problème d'arrivée en solutions du problème de départ. De plus, les conditions faisant intervenir $\mathcal{M}^*(x)$ ou $\mathcal{M}'^*(f(x))$ n'ont pas de sens s'il n'y a pas de solution réalisable. Pour les mêmes raisons, si l'instance d'arrivée n'a pas de solution réalisable, il sera difficile de produire une solution de l'instance de départ. Cependant, des problèmes sans solution existent et il faut bien qu'il soit possible de les réduire les uns aux autres, sans quoi aucun problème ne pourra être NPO-Complet. On va donc compléter les 3 réductions avec les conditions suivantes :

Définition 35 (Complétion des définitions de réduction L, isofacteur et PTAS)**.** Si on souhaite réduire un problème $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O})$ au problème $\Pi' = (\mathcal{I}', \mathcal{S}', \mathcal{M}', \mathcal{O}')$ avec une réduction L, isofacteur ou PTAS faisant intervenir les fonctions f et g, connaissant une instance $x \in \mathcal{I}$:

- si $\mathcal{S}(x) \neq \emptyset$, alors $\mathcal{S}'(f(x)) \neq \emptyset$, et les autres propriétés de la réduction doivent être vérifiées
- si $\mathcal{S}(x) = \emptyset$, alors soit $\mathcal{S}'(f(x)) = \emptyset$ soit f ne renvoie rien (ou renvoie le mot vide ou une erreur) aucune autre propriété n'a à être satisfaite (puisqu'elles n'ont pas de sens).

Le cas où f ne renvoie rien permet de réduire des problèmes où certaines instances n'ont pas de solution vers des problèmes où toutes les instances ont des solutions. On pourrait se dire qu'on pourrait alors se passer de l'autre possibilité, le cas où f renvoie une instance sans solution. Cependant, le cas où f ne renvoie rien doit se faire en temps polynomial. Et, comme mentionné précédemment, il est possible qu'il soit NP-Difficile de savoir si $\mathcal{S}(x) = \emptyset$. Donc la possibilité pour f de malgré tout renvoyer une instance $f(x)$ permet à f de renvoyer une sortie en temps polynomial même dans ce cas. On va voir ci-dessous un exemple de cas de la sorte.

On peut maintenant gérer sans crainte des problèmes où certaines instances n'ont pas de solution. Pour montrer la complétude des problèmes de satisfiabilité de poids maximum ou minimum, on va utiliser deux autres problèmes intermédiaires liés à l'exécution de machines de Turing non déterministes. La fin de la preuve réutilisera les techniques de la preuve de Cook et Levin montrant que SAT est NP-Complet.

Problème : Exécution bornée d'une machine de Turing de poids minimum/maximum

Instance :
- Une machine de Turing M non déterministe
- Une entrée x pour M
- Un entier p unaire
- Des poids $\omega : [\![-p; p]\!] \to \mathbb{N}$

Solution réalisable : Une exécution de M sur l'entrée x se terminant sur un état acceptant en p itérations.

Optimisation : On rappelle que chaque case de la bande est indicée avec un entier de $\mathbb{Z}$.
On veut minimiser/maximiser la somme des poids des cases de la bande de M contenant un 1 après la dernière itération. Autrement dit, si on pose z_i le numéro inscrit sur la case $i \in \mathbb{Z}$ à la dernière itération, avec $z_i = 0$ si la case contient un blanc, on veut optimiser $\sum_{i=-p}^{p} \omega(i) \cdot z_i$

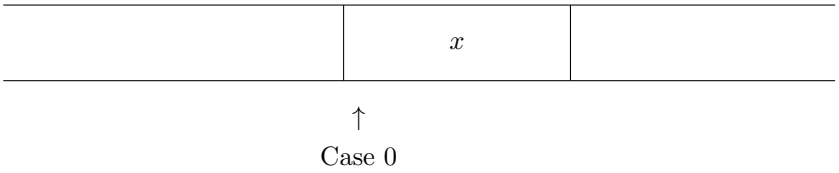
Remarque 17. La bande d'une machine de Turing est infinie, mais puisque la machine ne peut effectuer plus de p itérations, seules les cases $-p$ à p peuvent contenir un 1.

Lemme 4.3.2. *Soit $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O}) \in NPO$. Si $\mathcal{O} = \min$ (respectivement $\mathcal{O} = \max$), il existe une L-réduction avec $\alpha = \beta = 1$, une réduction isofacteur avec $\alpha = \beta = 1$ et une réduction PTAS avec $c(\varepsilon) = \varepsilon$ de Π vers le problème d'exécution bornée d'une machine de Turing de poids minimum (respectivement maximum).*

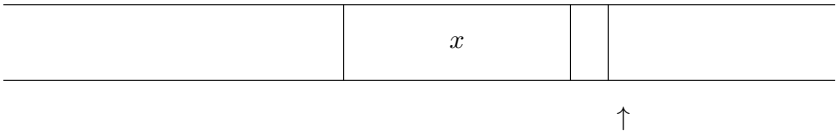
Démonstration. On veut donc, à partir de Π, créer une instance (M, x, p, ω) d'exécution bornée d'une machine de Turing de poids minimum/maximum en temps polynomial satisfaisant les conditions des trois réductions. Pour cela, une condition sera que toute solution y d'une instance de Π est associée, de manière bijective, à une solution réalisable de (M, x, p, ω) de même poids que y.

Puisque Π est dans NPO, il existe une machine non déterministe qui résout Π_D en temps polynomial, décrite dans la preuve du théorème 2.1.2. On va réutiliser une variante de cette machine ici :

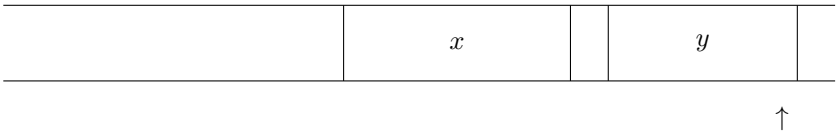
1. La machine prend en entrée un nombre binaire encodant une instance x de Π. On notera également x ce nombre binaire. On représente ci-dessous la bande, une flèche indique la position de la tête. On suppose que la première case de x est indicée avec l'entier 0. Si ce n'est pas le cas, la machine déplace x à cet emplacement.



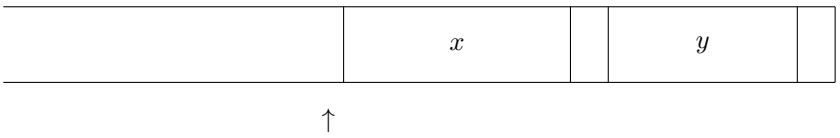
2. La machine vérifie que x est une instance de Π sinon elle répond NON. Elle place ensuite la tête à droite de x. Cette vérification se fait en temps polynomial.



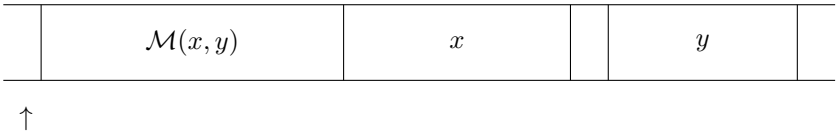
3. Toute solution réalisable de x est de taille bornée par un polynôme. La machine génère de manière non déterministe un nombre binaire y de taille polynomiale.



4. La machine vérifie que y encode bien une solution réalisable de x, sinon elle répond NON. Elle place ensuite la tête à gauche de x. Dans la suite, on identifiera le nombre binaire y et la solution réalisable de $\mathcal{S}(x)$ associée. Cette vérification se fait en temps polynomial.



5. La machine calcule $\mathcal{M}(x, y)$ et l'écrit dans la bande en temps polynomial. Elle l'écrit sur les cases à gauche de x. Ce calcul se fait en temps polynomial.



6. La machine répond OUI.

Quelle que soit sa réponse, cette machine s'exécute en temps polynomial. Soit q un polynome tel que $q(|x|)$ est une borne supérieure du temps de calcul de cette machine. L'entier p de notre instance d'exécution bornée d'une machine de Turing de poids minimum/maximum est fixé à $q(|x|)$.

On peut voir que, en temps $q(|x|)$, la machine répond OUI à partir du moment où une solution réalisable existe et il existe une et une seule exécution possible de la machine se terminant sur un état acceptant pour chaque solution réalisable $y \in \mathcal{S}(x)$. À l'issue de la dernière itération, $\mathcal{M}(x, y)$ est écrit sur la bande. Par construction, ce mot est écrit sur les cases $-1, -2, \ldots, -p$. On pose $\omega(-i) = 2^{i-1}$ pour $i \in [\![1; p]\!]$. Pour tout autre entier i, on pose $\omega(i) = 0$. On a donc immédiatement que le poids de cette exécution de la machine est $\mathcal{M}(x, y)$.

Pour plus de simplicité, dans la suite, on note $\Pi' = (\mathcal{I}', \mathcal{S}', \mathcal{M}', \mathcal{O}' = \mathcal{O})$ le problème d'exécution bornée d'une machine de Turing de poids minimum/maximum. Considérons donc maintenant la fonction f qui effectue les opérations ci-dessus pour transformer une instance $x \in \mathcal{I}$ en instance $(M, x, p, \omega) \in \mathcal{I}'$. Cette transformation est polynomiale. On peut voir que $\mathcal{S}(x) = \emptyset \Leftrightarrow \mathcal{S}'(f(x)) = \emptyset$, on respecte donc bien les conditions additionnelles des réductions dans le cas où il n'y a pas de solution.

Considérons également la fonction g qui, connaissant une solution $E \in \mathcal{S}'(M, x, p, \omega)$, c'est-à-dire une exécution de la machine M pour l'entrée x se terminant sur un état acceptant en moins de p itérations, va lire le nombre binaire y écrit sur la bande et renvoyer la solution réalisable de $\mathcal{S}(x)$ associée. Pour tout $y \in \mathcal{S}(x)$ de poids $\mathcal{M}(x, y)$, il existe une exécution $E \in \mathcal{S}'(M, x, p, \omega)$ de la machine qui écrit y sur la bande et qui est de poids $\mathcal{M}'(M, x, p, \omega, E) = \mathcal{M}(x, y)$. Inversement, pour toute exécution $E \in \mathcal{S}'(M, x, p, \omega)$, il existe une solution réalisable $y \in \mathcal{S}(x)$ de poids $\mathcal{M}(x, y) = \mathcal{M}'(M, x, p, \omega, E)$. On peut donc identifier les solutions réalisables de chaque problème avec des solutions de l'autre problème et de même poids. Les solutions optimales ont donc la même valeur : $\mathcal{M}^*(x) = \mathcal{M}'^*(M, x, p, \omega) = \mathcal{M}'^*(f(x))$. De plus $\mathcal{M}'(f(x), E) = \mathcal{M}(x, g(E))$. Ces deux égalités prouvent l'existence d'une L-réduction avec $\alpha = \beta = 1$ et une réduction isofacteur avec $\alpha = \beta = 1$ de Π vers Π'.

Pour la réduction PTAS, un détail technique nous empêche d'utiliser f et g tels quel puisque, formellement ces fonctions doivent dépendre d'un autre paramètre ε. Il suffit donc d'ajouter que f et g ignorent ce nouveau paramètres et on obtient bien une réduction PTAS avec $c(\varepsilon) = \varepsilon$.

□

Définition 36. On nomme min-NPO et max-NPO les classes des problèmes de minimisation et de maximisations de NPO.

Théorème 4.3.1. *Exécution bornée d'une machine de Turing de poids minimum est* min-*NPO-Complet sous les réductions L, isofacteur et PTAS. Exécution bornée d'une machine de Turing de poids maximum est* max-*NPO-Complet sous les réductions L, isofacteur et PTAS.*

Démonstration. Ceci découle immédiatement du Lemme 4.3.2 et de la définition de complétude. □

Pour la suite de la preuve, il faut transformer nos problèmes intermédiaires en problème de satisfiabilité de poids minimum ou maximum avec les mêmes réductions. Sans aller jusqu'à refaire la preuve du théorème de Cook-Levin, on va en réexpliquer quelques détails :

- Cette preuve permet de montrer que SAT est un problème NP-Complet.
- Pour cela elle montre qu'on peut encoder une machine de Turing avec une formule booléenne de taille polynomiale par rapport à celle de la machine et au temps qu'elle passe à calculer. La formule est satisfiable si et seulement si la machine s'arrête sur un état acceptant.
- Dans notre formule, on encode, pour chaque itération k, la i^{e} case de la bande avec 3 variables t_{ik0}, t_{ik1} et $t_{ik\emptyset}$ signifiant qu'à l'itération k, sur la case i, il y a un 0, un 1 ou un blanc. De manière similaire, d'autres variables encodent, pour chaque itération, la case où se trouve la tête de lecture et l'état courant de la machine. Chaque set de variables correspond à une photo de l'état de la machine à chaque itération, de

la première jusqu'à la dernière. De nombreuses clauses existent pour relier entre elles ces variables, pour s'assurer que la cohérence de l'exécution de la machine, par exemple un XOR relie les variables t_{ik0}, t_{ik1} et $t_{ik\emptyset}$ pour éviter que deux symboles soient écrits sur la même case au même moment. Une autre clause s'assure qu'il existe au moins une itération où un état acceptant est atteint. D'autres clauses permettent par exemple de choisir une transition valide et de relier les variables des itérations avant et après que la transition ait été utilisée. Les variables étant indicées par l'itération k, on comprend bien que le nombre d'itérations de la machine doit être borné par un polynôme si on souhaite que la transformation se fasse en temps polynomial.

On utilise donc la même techniques pour transformer une instance (M, x, p, ω) d'exécution bornée d'une machine de Turing de poids minimum/maximum en instance (φ, X, ω') de satisfiabilité de poids minimum/maximum. Connaissant le poids $\omega(i)$ de la case i, on pose $\omega'(t_{ip1})$. Pour tout autre variable de X, le poids ω' est nul. La variable t_{ip1} étant vraie si et seulement si, à l'itération p, la case i est remplie avec le symbole 1, le poids de l'affectation des variables sera égal au poids de l'exécution de la machine. On obtient donc, comme dans la preuve du lemme 4.3.2, une identification des solutions réalisables des deux problèmes avec des poids identiques. Ainsi, on peut construire une réduction L, Isofacteur ou PTAS entre les deux problèmes.

Théorème 4.3.2. *Minimum Weighted Satisfiability est* min-*NPO-Complet sous les réductions L, isofacteur et PTAS. Maximum Weighted Satisfiability est* max-*NPO-Complet sous les réductions L, isofacteur et PTAS.*

Ce théorème ne suffit pas à prouver que ces problèmes sont NPO-Complets. Pour cela, il serait nécessaire de prouver qu'ils se réduisent l'un à l'autre. Une réduction est donnée dans le premier livre indiqué en introduction mais utilise la réduction AP, non présentée dans ce cours, montrant donc que ces problèmes sont NPO-Complets sous la réduction AP. Un point important est que la réduction PTAS est une généralisation de la réduction AP. Il existe donc une réduction PTAS de Minimum Weighted Satisfiability vers Maximum Weighted Satisfiability et inversement. On peut donc en déduire le résultat suivant :

Théorème 4.3.3. *Minimum Weighted Satisfiability et Maximum Weighted Satisfiability sont NPO-Complets sous la réduction PTAS.*

À partir de tous ces résultats, on peut par exemple montrer la NPO-Complétude d'autres problèmes comme le voyageur de commerce (dans un graphe quelconque sans inégalité triangulaire) sous la réduction PTAS. Il faut noter que de bons candidats à la NPO-Complétude sont des problèmes pour lesquels il est NP-Complet de trouver des solutions réalisables. Un autre point important est que ces problèmes doivent avoir des poids et, dans les réductions, ces poids sont exponentiellement grands (ici on a utilisé des poids de l'ordre de 2^i pour la case i). Il semble en effet que l'existence d'un problème sans poids, ou dont les poids sont polynomialement bornés et qui soit NPO-Complet soit peu probable car elle implique en particulier que la hiérarchie polynomiale soit finie [6].

On peut enfin noter que sous les réductions L, isofacteur et PTAS, il n'y a malheureusement pas, à ma connaissance de résultat marquant de NPO-Complétude. Plus exactement, aucun résultat n'est mentionné dans les sources de ce poly. On peut noter toutefois quelques autres résultats connus cités dans ces sources. On rappelle que $O(2^{n^c})$-APX et $O(n^c)$-APX sont respectivement les problèmes que l'on peut approcher avec un rapport exponentiel et polynomial (attention, ces classes possèdent des problèmes de minimisation et de maximisation. Dans le cas d'un problème de maximisation, il faut que le rapport soit inversé donc, formellement, il faudrait écrire $O(n^c)$-APX $\cup$ $O(n^{-c})$-APX).

Théorème 4.3.4. *Si on se restreint aux instances où il existe au moins une affectation des variables qui satisfait la formule et qui peut être calculée en temps polynomial, le problème Minimum (respectivement Maximum) Weighted Satisfiability est* min-$O(2^{n^c})$*-APX-Complet (respectivement* max-$O(2^{n^c})$*-APX-Complet) sous les réductions L, isofacteur et PTAS.*

Démonstration. Voir exercice 33. □

Théorème 4.3.5. *Le problème Max3SAT est APX-Complet sous la réduction PTAS.*

Théorème 4.3.6. *Les problèmes Stable Maximum et Clique Maximum sont $O(n^c)$-APX-Complet sous la réduction PTAS.*

Enfin, un dernier résultat intéressant et cohérent avec l'appartenance de Bin Packing à la classe PTAS_∞. On va d'abord définir la notion de problème intermédiaire.

6. Cette hiérarchie est un ensemble de classes généralisant les classes P, NP et Co-NP `https://fr.wikipedia.org/wiki/Hiérarchie_polynomiale`.

Définition 37. Soient $\mathcal{C}$ et $\mathcal{C}'$ des classes d'approximabilité telles que $\mathcal{C}' \subsetneq \mathcal{C}$ et $\leq_R$ une réduction entre problèmes d'optimisation préservant l'appartenance à $\mathcal{C}'$, un problème Π est $\mathcal{C}$-intermédiaire si $\Pi \in \mathcal{C} \backslash \mathcal{C}'$ et si Π n'est pas NP-Complet.

Théorème 4.3.7 ([7]). *Le problème Bin Packing est APX-Intermédiaire (relativement à la classe PTAS) sous la réduction PTAS.*

Ces résultats citent la réduction PTAS car il s'agit de la réduction la plus générale qui existe. Ainsi, il existe beaucoup plus de problèmes qui se réduisent les uns les autres sous la réduction PTAS que sous les autres réductions qui sont plus strictes. La réduction AP, précédemment mentionnée fait aussi partie des réductions pour lesquels beaucoup de résultats ont été montrés, elle est moins générale et donc plus simple à utiliser que la réduction PTAS.

D'autres résultats pour d'autres réductions peuvent également être trouvés dans les sources de ce poly.

4.4 Un mot sur le théorème PCP et ses conséquences

Terminons sur le résultat le plus fort de l'histoire de l'approximabilité polynomiale et qui a permis de prouver de nombreux résultats d'inapproximabilité. Ces résultats se basent essentiellement sur des réductions GAP après avoir défini et prouvé une nouvelle caractérisation de NP.

Certificats et vérifieurs probabilistes

Une définition classique de NP se base sur la notion de certificat et de vérifieur.

Définition 38. NP est l'ensemble des problèmes de décision $\Pi = (\mathcal{L}, \mathcal{L}^Y, \mathcal{L}^N)$ tels qu'il existe un polynôme p et un algorithme polynomial déterministe $\mathcal{V}$ tels que :

- $\mathcal{V}$ prend en entrée une instance x de Π et un mot binaire, et renvoie OUI ou NON
- pour toute instance $x \in \mathcal{L}^Y$ de Π, il existe un mot binaire c de taille $p(|x|)$ tel que $\mathcal{V}(x, c)$ répond OUI
- pour toute instance $x \in \mathcal{L}^N$ de Π, pour tout mot binaire c de taille $p(|x|)$, $\mathcal{V}(x, c)$ répond NON

Le mot binaire c est appelé un *certificat* et l'algorithme $\mathcal{V}$ est appelé *Vérifieur*. Par exemple, pour le problème SAT, le certificat consiste en l'ensemble des affectations des variables et le vérifieur en le calcul de la formule une fois les variables remplacées par leurs affectations ; si la formule est satisfiable, il existe un certificat pour lequel le vérifieur répond OUI ; si la formule n'est pas satisfiable, quelle que soit l'affectation qu'on donnera au vérifieur, il répondra NON. Pour le problème de recherche d'une chaîne hamiltonienne dans un graphe non orienté, le certificat consiste à donner une suite de n nœuds du graphe et le vérifieur vérifie que cette suite est une chaîne élémentaire ; si une chaîne hamiltonienne existe, pour au moins un certificat, le vérifieur répondra OUI ; sinon quelle que soit la suite de nœuds fournies, cette suite ne sera pas une chaîne élémentaire.

Cette définition permet de se passer de la définition à base de machines de Turing, bien qu'elle soit en réalité identique. Le certificat représente les choix non déterministes que doit faire la machine de Turing et le vérifieur correspond à l'exécution de la machine en suivant les choix du certificat. Une machine non déterministe accepte un mot si et seulement s'il existe une suite de choix qui mène à un état acceptant, c'est-à-dire si et seulement s'il existe un certificat qui sera accepté par le vérifieur, on retrouve donc bien la définition ci-dessus.

On peut se demander s'il est toujours nécessaire de lire tout le certificat. Si on souhaite répondre de manière sûre, il semble difficile de se passer de l'ensemble du certificat. Clairement, il faut lire l'ensemble des nœuds d'une suite de n nœuds pour s'assurer qu'il s'agit bien d'une chaîne hamiltonienne et il faut lire l'ensemble des affectations des variables pour pouvoir vérifier si la formule est satisfaite par ces affectations. Pourtant, quand la réponse est NON, il semblerait que ce ne soit pas le cas. Par exemple, si on me donne une suite de nœuds d'un graphe, alors en lisant seulement 2 informations, je peux dire qu'il ne s'agit pas d'une chaîne hamiltonienne, il me suffit de vérifier si la suite ne contient pas deux nœuds identiques ou s'il n'y a pas deux nœuds successifs qui ne sont pas reliés. À partir du moment où j'ai trouvé ces deux nœuds, je n'ai pas besoin de lire le reste du certificat. Se pose alors la question de savoir comment trouver ces deux nœuds ? C'est le hasard qui va nous les indiquer. C'est ainsi qu'est venue l'idée des classes PCP (Probabilistic Checkable Proofs)

Définition 39. PCP$[r(n), q(n)]$ est l'ensemble des problèmes de décisions $\Pi = (\mathcal{L}, \mathcal{L}^Y, \mathcal{L}^N)$ tels qu'il existe un polynôme p et un algorithme probabiliste polynomial $\mathcal{V}$ tels que :

- $\mathcal{V}$ prend en entrée une instance x de Π et un mot binaire, et renvoie OUI ou NON
- $\mathcal{V}$ génère $r(|x|)$ bits aléatoire pour effectuer son calcul
- $\mathcal{V}$ lit au plus $q(|x|)$ bits du certificat

7. P. Crescenzi, R. Silvestri, and L. Trevisan. Natural complete and intermediate problem in approximation classes. Manuscript, 1994.

— pour toute instance $x \in \mathcal{L}^Y$ de Π, il existe un mot binaire c de taille $p(|x|)$ tel que $\mathcal{V}(x, c)$ répond OUI avec une probabilité égale à 1
— pour toute instance $x \in \mathcal{L}^N$ de Π, pour tout mot binaire c de taille $p(|x|)$, $\mathcal{V}(x, c)$ répond NON avec une probabilité supérieure à $\frac{1}{2}$

Par exemple une caractérisation simple de NP est la suivante :

Théorème 4.4.1. *NP* $= \bigcup_{d \in \mathbb{N}} PCP[0, n^d]$

Démonstration. Si on reprend la définition de la classe PCP, on obtient que $\mathcal{V}$ génère 0 bits aléatoires, il devient donc un algorithme polynomial déterministe, il répondra donc OUI ou NON avec une probabilité de 1. On voit aussi que le vérifieur peut lire un nombre polynomial de bits du certificats. Puisque le certificat est de taille polynomiale, le vérifieur peut lire tout le certificat. On obtient donc exactement la définition de NP présentée plus haut. □

Le résultat impressionnant et inattendu mentionné en début de partie est le théorème PCP.

Théorème 4.4.2. *NP* $= PCP[O(\log(n)), O(1)]$

(Plus formellement $\mathrm{PCP}[O(\log(n)), O(1)] = \bigcup_{d_1, d_2 \in \mathbb{N}} \mathrm{PCP}[d_1 \log(n), d_2]$)

Avant d'esquisser un léger morceau de la preuve de ce théorème, remarquons à quel point cette caractérisation est surprenante : elle annonce qu'en lisant au plus un nombre constant de bits du certificat, le vérifieur va se tromper au plus dans la moitié des cas où la réponse est NON et ne jamais se tromper dans le cas où la réponse est OUI.

Morceau de preuve. L'inclusion de la classe de droite dans NP est plus simple que l'inverse. Et, étant donnée la taille de la preuve et les nombreuses techniques qu'elle fait intervenir, on se contentera de prouver cette partie. Le reste de la preuve pour les lecteurs courageux se trouve dans les sources du poly.

Soient $d_1, d_2 \in \mathbb{N}$ et $\Pi \in \mathrm{PCP}[d_1 \log(n), d_2])$. Soit une instance x de Π. Si x est positive, alors il existe au moins un certificat c de taille $p(|x|)$ tel que, quelque soit la suite aléatoire de bits générés, le vérifieur $\mathcal{V}(x, c)$ répond OUI. Si x est négative, alors pour tout certificat c de taille $p(|x|)$, pour au moins la moitié des suites aléatoires de bits générés, le vérifieur $\mathcal{V}(x, c)$ répond NON.

Un point crucial est que le nombre de suites de $d_1 \log(n)$ bits aléatoires est $2^{d_1 \log(n)} = n^{d_1}$. Autrement dit, il y a un nombre polynomial de séquences aléatoires que peut générer le vérifieur $\mathcal{V}$. Considérons l'algorithme $\mathcal{V}'$ déterministe suivant : pour chacune des n^{d_1} séquences, $\mathcal{V}'(x, c)$ va simuler $\mathcal{V}(x, c)$ comme s'il générait la séquence en question. Si, pour au moins la moitié des séquences, $\mathcal{V}(x, c)$ répond OUI, alors $\mathcal{V}'(x, c)$ répond OUI. Sinon, dès que $\mathcal{V}(x, c)$ répond NON, $\mathcal{V}'(x, c)$ répond NON.

On a donc bien un vérifieur polynomial déterministe $\mathcal{V}'$ pour Π, donc $\Pi \in$ NP. □

Quelques exemples dans le sens du théorème PCP

Nous allons illustrer, aux travers d'exemples, comment, en lisant un nombre constant de bits de la preuve, il est possible de ne pas se tromper quand la réponse est OUI, et de ne pas toujours se tromper quand la réponse est NON.

Exemple 9. Considérons le problème 3SAT, nous avons une formule booléenne avec n variables $(x_1, x_2, \ldots, x_n)$ avec m clauses $(C_1, C_2, \ldots, C_m)$. Chaque clause doit être satisfaite pour satisfaire la formule. Chaque clause contient exactement 3 littéraux. Considérons l'algorithme $\mathcal{V}$ suivant qui prend en entrée une formule et un nombre binaire de taille n correspondant à une possible affectation des n variables, 1 si la variable est vraie et 0 sinon. $\mathcal{V}$ commence par générer $\log(m)$ bits aléatoires. Ces bits forment un nombre k entre 1 et m. Il va ensuite inspecter la clause k, supposons sans perte de généralité que $C_k = (x_i \vee \bar{x_j} \vee x_l)$. Le vérifieur lit ensuite les bits i, j et l du certificat. Si ces bits sont respectivement 0, 1 et 0 alors $\mathcal{V}$ répond NON. Sinon il répond OUI.

Si l'instance est positive, alors il existe bien un certificat permettant au vérifieur de répondre toujours OUI. La formule étant satisfiable, il existe une affectation des variables qui satisfait toutes les clauses. Ainsi, quelle que soit le nombre aléatoire k généré avec les $\log(m)$ bits, le certificat indiquera qu'au moins un des trois littéraux de la clause est vrai, et donc le vérifieur répondra OUI. Par contre, si l'instance est négative, quel que soit le certificat, au moins une clause n'est pas satisfaite. S'il s'agit de la clause C_k inspectée par le vérifieur, il répondra NON. Sinon il est possible que le vérifieur réponde OUI. La probabilité de tomber sur une clause non satisfaite est au moins $\frac{1}{m}$.

On a donc *presque* la preuve que 3SAT est dans $\mathrm{PCP}[O(\log(m)), O(1)]$. La difficulté consiste à élever cette probabilité à $\frac{1}{2}$.

Exemple 10. Considérons le problème de chaîne hamiltonienne, nous avons un graphe $G = (V, E)$ avec n nœuds et m arêtes. Considérons l'algorithme $\mathcal{V}$ suivant qui prend en entrée une formule et un nombre binaire de taille n^2 correspondant à une suite de n nœuds. Chaque tranche de n bits contient un seul 1 au milieu de $n-1$ bits égaux à 0 et décrit un nombre entre 1 et n. Ainsi la i^{e} tranche décrit un nœud v_i de G. Par exemple, le certificat correspondant à la suite v_2, v_1, v_3 s'écrirait avec 3 tranches de 3 nœuds 010-100-001. On suppose pour simplifier que $\mathcal{V}$ ne reçoit pas de certificat malformé, avec par exemple une ligne ne contenant que des 0 ou une ligne contenant 2 fois un bit égal à 1.

$\mathcal{V}$ commence par générer deux nombres aléatoires k_1 et k_2 entre 1 et n^2 (avec $4\log(n)$ bits). Il va ensuite inspecter les bits k_1 et k_2 du certificat. Si un de ces deux bits est nul ou si $k_1 = k_2$, il répond OUI. Sinon ces deux bits correspondent à des nœuds v_1 et v_2. Si $v_1 = v_2$, il répond NON. Si les deux bits ne sont pas sur des tranches successives du certificat, il répond OUI. Enfin si (v_1, v_2) est une arête de E, il répond OUI, sinon il répond NON.

Si l'instance est positive, alors il existe bien un certificat permettant au vérifieur de répondre toujours OUI. Le graphe étant hamiltonien, il existe une suite de n nœuds qui forment une chaîne élémentaire du graphe. Supposons que le certificat encode cette suite de nœuds. Considérons les deux nombres aléatoires générés par $\mathcal{V}$, soient un de ces nombres ne correspondent pas à des nœuds (le bit vaut 0) et $\mathcal{V}$ répond OUI. Soit ils correspondent à deux nœuds v_1 et v_2. On ne peut pas avoir $v_1 = v_2$ puisque la chaîne encodée par le certificat est élémentaire. Si les deux bits ne sont pas dans des tranches successive, le vérifieur répond OUI. Enfin, s'ils le sont, alors les deux nœuds correspondant sont reliés puisque le certificat encode une chaîne.

Si l'instance est négative, alors quel que soit le certificat donné au vérifieur, il répondra NON avec une probabilité $\frac{1}{n^4}$. En effet, le graphe n'étant pas hamiltonien, pour toute séquence de n nœuds encodée par le certificat, la séquence contient au moins 2 nœuds identiques ou deux nœuds successifs qui ne sont pas reliés par une arête dans le graphe. La probabilité pour que k_1 et k_2 renvoient à ces deux nœuds est supérieure à $\frac{1}{n^4}$.

Il est possible d'arriver à un résultat similaire sans forcer le certificat à être bien formé mais la preuve est technique et sort du cadre de ce que l'on souhaite montrer avec cet exemple.

Exemple 11. Considérons enfin le problème de savoir si un graphe orienté, dont le degré sortant des nœuds est inférieur à une constante d, possède un noyau. Soit un graphe $G = (V, A)$ orienté avec n nœuds et m arêtes, un noyau de G est un sous-ensemble V' de V tel que V' est un stable dans G et tout nœud de V est soit dans V' soit possède un successeur dans V'. On pose $V = (v_1, v_2, \dots v_n)$. Considérons l'algorithme $\mathcal{V}$ suivant qui prend en entrée un graphe et un nombre binaire de taille n encodant V' : le i$^{\text{e}}$ bit du certificat vaut 1 si et seulement si le $v_i \in V'$.

$\mathcal{V}$ commence par sélectionner aléatoirement un nœud v_i avec $\log(n)$ bits. Il vérifie si $v_i \in V'$ ou non avec le certificat. Si oui, il vérifie pour chacun des successeurs de v_i si ce successeur est dans V', auquel cas il répond NON. Sinon il répond OUI. Si $v_i \notin V'$, il vérifie pour chacun des successeurs si $v_i \in V'$. Si aucun n'est dans V' alors il répond NON et sinon il répond OUI. Dans les deux cas, l'algorithme a lu $d+1$ bits du certificat.

Si l'instance est positive, alors il existe bien un certificat permettant au vérifieur de répondre toujours OUI. Le graphe possédant un noyau V', supposons que le certificat encode ces nœuds. Quel que soit le nœud aléatoire choisi par $\mathcal{V}$, il sera soit dans V' et n'aura pas de successeur dans V' par définition du noyau, soit il ne sera pas dans V' et aura donc un successeur dans V' par définition du noyau. Le vérifieur répond donc OUI.

Si l'instance est négative, alors quel que soit le certificat donné au vérifieur, il répondra NON avec une probabilité $\frac{1}{n}$. En effet, le graphe ne possédant pas de noyau, pour tout sous-ensemble V' de nœuds de G encodé par le certificat, V' contient soit un nœud possédant un successeur dans V', soit un nœud ne contenant aucun successeur. La probabilité, dans les deux cas, pour que ce nœud fautif soit aléatoirement choisi est supérieure à $\frac{1}{n}$.

Il est possible de se passer de la contrainte sur le degré de G mais cela rend la preuve très technique et moins compréhensible.

Conséquences du théorème PCP

Le théorème PCP nous permet de prouver, en particulier, que Max-SAT n'est pas dans la classe PTAS. De cette preuve découlent, à l'aide d'une suite de réductions, d'autres résultats d'inapproximabilité comme celle de Clique maximum, Stable maximum, Couverture par ensembles, Couverture par sommets, Arbre de Steiner, ...

On va terminer ce cours avec le premier résultat. L'inapproximabilité de Clique et Stable est donnée en exercice.

Problème : Satisfiabilité maximum ou Max-SAT (Maximum Satisfiability en anglais)

Instance :
— m clauses disjonctives $C_1, C_2, \ldots, C_m$.
$C_j = (l_1^j \vee l_2^j \vee \cdots \vee l_p^j)$, où l_i^j est un littéral x_k ou $\bar{x_k}$.

Solution réalisable : Une affectation des variables à vrai ou faux, c'est-à-dire $a_1, a_2, \ldots, a_n \in \{\top, \bot\}$

Optimisation : Maximiser le nombre de clauses satisfaites.

Théorème 4.4.3. *Il existe une GAP-Réduction de tout problème de NP vers Max-SAT avec α qui est une constante vérifiant $\alpha < 1$.*

Démonstration. Soit $\Pi = (\mathcal{L}, \mathcal{L}^Y, \mathcal{L}^N)$ un problème de décision de NP. D'après le théorème 4.4.2, $\Pi \in \text{PCP}[O(\log(n)), O(1)]$. Donc il existe un vérifieur $\mathcal{V}$ qui prend en entrée une instance x de Π de taille n et un certificat c, qui génère $r(n) = O(\log(n))$ bits aléatoires, qui lit $q(n) = q$ bits du certificat et qui renvoie OUI si $x \in \mathcal{L}^Y$ pour au moins un certificat et NON si $x \in \mathcal{L}^N$ pour tout certificat pour au moins la moitié des séquences aléatoires possibles. On va montrer qu'on peut, à l'aide du vérifieur, construire une instance de Max-SAT qui satisfait toutes les clauses si $x \in \mathcal{L}^Y$ mais qui n'en satisfait qu'une faction $\alpha < 1$ sinon.

Rappelons, comme évoqué dans l'esquisse de la preuve du théorème 4.4.2 que le nombre de séquences aléatoires possibles est $2^{r(n)} = 2^{d \cdot \log(n)} = n^d$ qui est polynomial. On peut supposer sans perte de généralité que c est de taille au plus qn^d. En effet, le vérifieur lit q bits de la preuve pour chacune des séquences aléatoires, donc si le certificat est plus grand que qn^d, il existera nécessairement des bits qui ne seront jamais lus par le vérifieur [8]. Enfin, on supposera que le vérifieur lit toujours exactement q bits de c, quitte à ignorer une partie des bits lus.

Construisons maintenant une instance de Max-SAT avec ces considérations. Pour chaque bit du certificat, on génère une variable z_i, qui sera vrai si le i^{e} bit du certificat vaut 1 et fausse si le i^{e} bit du certificat vaut 0 : on a donc qn^d variables. On veut maintenant générer des clauses qui vont encoder la question suivante : existe-t-il un certificat qui soit accepté par le vérifieur avec une probabilité de 1 ? Pour générer les clauses, on va donc simuler $\mathcal{V}$ sur l'ensemble des n^d séquences aléatoires possibles. Pour chaque séquence, on va regarder les bits du certificats qui sont lus. Pour chaque valeur possible de ces bits, le vérifieur peut répondre OUI ou NON. S'il répond NON, il faut empêcher le certificat de prendre ces valeurs là, on va donc ajouter une nouvelle clause. On détaillera cette étape à la fin. Considérons pour me moment que, pour chaque séquence r de bits aléatoires, on a construit au plus 2^q clauses et on note $C(r)$ ces clauses. On complète la formule avec des clauses trivialement satisfiable de sorte à en avoir exactement $2^q n^d + 1$.

Dans la suite, on notera $(\mathcal{I}', \mathcal{S}', \mathcal{M}', \max)$ le problème Max-SAT. On note f la transformation ci-dessus d'une instance x de Π en instance $f(x) \in \mathcal{I}'$. Posons $g(x) = 2^q n^d$. On peut noter que $g(x)$ est le nombre de clauses de $f(x)$. On pose enfin $\alpha = (2^q n^d + 1 - \frac{n^d}{2})/(g(x)) = (2^q n^d + 1 - \frac{n^d}{2})/2^q n^d < 1$ pour des valeurs assez grandes de n.

Supposons maintenant que $x \in \mathcal{L}^Y$, alors il existe un certificat c qui est accepté par le vérifieur quelle que soit la séquence de bits aléatoire lue par le vérifieur. Soit z la solution de $\mathcal{S}'(f(x))$ qui consiste à affecter z_i à vrai si et seulement si le i^{e} bit de c vaut 1. Alors toutes les clauses de $f(x)$ sont satisfaites. On a donc $\mathcal{M}'^*(f(x)) = 2^q n^d + 1 = g(x)$. Si, au contraire $x \in \mathcal{L}^N$, alors, pour tout certificat, c est refusé pour au moins la moitié des séquences de bits aléatoires. Cela signifie que, quelle que soit l'affectation z des variables, pour une séquence r de bits aléatoire sur deux, on a au moins une clauses de $C(r)$ qui est fausse. Donc on a au moins $\frac{n^d}{2}$ clauses qui sont fausses. Le nombre de clauses satisfaites est donc $\mathcal{M}'(f(x), z) \leq 2^q n^d + 1 - \frac{n^d}{2} = \alpha \cdot g(x)$. Et donc on a bien une GAP-Réduction avec un rapport α.

Montrons enfin comment construire $C(r)$. Soient $i_1, i_2, \ldots, i_q$ les indices des q bits lus par le vérifieur [9]. Il y a 2^q tuples de valeurs possibles pour ces bits, ce nombre est constant et on peut donc les énumérer. Pour chaque tuple $t = (t_1, t_2, \ldots, t_q)$, on simule le vérifieur. Si le vérifieur répond NON, on rajoute une clause disant que les bits $i_1, i_2, \ldots, i_q$ ne doivent pas être égaux à t. Par exemple, si $t = (0, 0, \ldots, 0)$, la clause sera $C = (z_{i_1} \vee z_{i_2} \vee \cdots \vee z_{i_q})$. Ainsi, on force le certificat à mettre au moins un de ces bits à 1. Autre exemple, si $t = (0, 1, 0, 1, \ldots, 0, 1)$, alors la clause sera $C = (z_{i_1} \vee \bar{z_{i_2}} \vee z_3 \vee \bar{z_4} \vee \cdots \vee z_{i_{q-1}} \vee \bar{z_q})$, ainsi soit un des bits impair vaut 1, soit un des bits pair vaut 0. Si le vérifieur répond OUI, on ne fait rien. On a donc au plus 2^q

8. Plus exactement, à partir de cette remarque, on peut prouver qu'il existe un vérifieur $\mathcal{V}'$ qui ne lit que les qn^d premiers bits de c.

9. On a ici une petite approximation de la réalité. Le 2e bit lu peut tout à fait dépendre de la valeur du premier bit lu. Si par exemple l'algorithme $\mathcal{V}$ peut décider de lire le 2e bit si le premier vaut 0 et le 3e bit à la place si le premier vaut 1. Cela ne change pas fondamentalement la suite de la preuve.

clauses ajoutées à la formule pour chacune des n^d séquences aléatoires. On a donc bien une formule de taille polynomiale. □

Corollaire 4.4.1. *Si P ≠ NP, Max-SAT n'est pas dans PTAS.*

Démonstration. D'après le théorème 4.4.3 et le théorème 4.2.9, si P ≠ NP, il existe une constante $\alpha < 1$ telle que MAX-SAT ne peut être approché avec un rapport supérieur à α, donc Max-SAT ne peut être dans PTAS sous cette condition. □

Parmi les résultats qui ont suivi cette preuve, les théorèmes suivantes ont été prouvés :

Corollaire 4.4.2. *Si P ≠ NP, le problème max-3SAT n'est pas dans PTAS.*

Corollaire 4.4.3. *Si P ≠ NP, le problème de l'arbre de Steiner n'est pas dans PTAS.*

Corollaire 4.4.4. *Si P ≠ NP, le problème de couverture par sommets n'est pas dans PTAS.*

Corollaire 4.4.5. *Si P ≠ NP, pour tout $\varepsilon > 0$, le problème de couverture par ensembles ne peut être approché avec un rapport inférieur à $(1 - \varepsilon)\log(n)$.*

Corollaire 4.4.6. *Si P ≠ NP, pour tout $0 < \varepsilon < \frac{1}{2}$, le problème de clque maximum ne peut être approché avec un rapport meilleur que n^{ε}.*

4.5 Conclusion

Cette partie a présenté un tout petit morceau de tout ce qui peut être dit sur l'inapproximabilité. Ce sujet pourrait à lui seul justifier un cours entier tellement il est vaste. Si les deux premières parties s'intéressent à démontrer que des problèmes n'ont pas d'approximation polynomiale avec tel ou tel rapport, les deux dernières vont plus loin et s'intéresse directement aux classes d'approximabilité et aux conséquences des réductions sur l'existence de problèmes complets pour ces classes. Un point important à noter est que ces derniers résultats ont aussi des impacts sur la théorie de la complexité et montre le lien étroit qui existe entre difficulté d'approximation, complétude dans les classes d'approximabilité et propriétés des classes de la théorie de la complexité (P, NP, Co-NP, Hiérarchie polynomiale, ...). Il ne faut donc pas voir l'approximabilité comme une simple extension de la théorie de la complexité. Tout est lié et le moindre résultat dans l'une des branches de cette théorie peut avoir des conséquences dans toutes les autres branches.

Un autre point à ne pas oublier est qu'inapproximabilité ne rime pas avec insolvabilité. De même qu'un problème NP-Difficile n'est pas insoluble, un problème inapproximable pourra toujours être attaqué par des solveurs de programmation linéaire, des méthodes de branchement, des méta-heuristiques ou des heuristiques adaptées, bien qu'on ne pourra pas prouver de garantie de performances pour ces méthodes. Plus exactement, l'inapproximabilité est une bonne justification de l'utilisation de ces méthodes, puisqu'elles sont alors notre dernier recours pour produire des solutions à nos problèmes.

Chapitre 5

Conclusion

Dans ce cours, j'ai cherché à proposer d'une manière pédagogique l'approximabilité polynomiale en arrangeant et réordonnant les sources que j'utilise et espère qu'il vous aura plu. Des cours d'approximabilité existent en de nombreux exemplaires sur internet, beaucoup reprennent les mêmes sources que moi, mais il ne me semble pas qu'il existe de cours qui les condensent comme je l'ai fait.

J'aimerai noter ici un point fondamental de la théorie de l'approximabilité qui en fait toute son intérêt : trouver une approximation polynomial pour un problème ou un résultat d'inapproximabilité est extrêmement difficile. Aujourd'hui, en théorie de la complexité classique, il n'est pas très difficile de démontrer des preuves de NP-Complétude pour de nouveaux problèmes. Les preuves se ressemblent et la grande variété des problèmes NP-Complets fait qu'il est souvent (je dis bien *souvent*, certains problèmes résistent très bien quand on cherche à savoir s'ils sont polynomiaux ou non) possible de trouver des problèmes proches de ceux qu'on cherche à prouver NP-Complets. On peut rapidement voir un lien entre 3SAT, Partition, Sac à dos, Voyageur de commerce, Set Cover, ... et un nouveau problème. C'est un jeu pour lequel on obtient facilement de l'expérience. De même prouver la polynomialité d'un problème passe souvent par des techniques similaires telles que la programmation dynamique ou la résolution de programmes linéaires (en nombre réels). En approximabilité c'est tout autre chose car le moindre changement peut avoir un impact énorme sur l'approximabilité d'un problème et une astuce peut rapidement ne plus fonctionner. Tous les algorithmes, toutes les techniques sont variées. Même pour un même problème, on pourra trouver pléthore d'algorithmes pour le résoudre, chacun avec son analyse et son rapport d'approximation. De l'autre côté, construire une réduction polynomiale qui préserve l'approximabilité d'un problème est chose plus difficile que faire une *simple* réduction de Karp. La profusions de réductions qui existent ne rendent pas non plus très simple la recherche d'inapproximabilité. Cette difficulté fait tout le plaisir de personnes acharnées prêtes à passer des semaines ou des mois sur le même problème pour en ressortir un algorithme ou une preuve.

Bien entendu, trouver ces résultats ne s'arrête pas là. Comme évoqué dans le chapitre précédent, une inapproximabilité peut être complétée par le développement d'algorithmes heuristiques sans garantie pour malgré tout résoudre le problème (en particulier s'il a un intérêt pratique). De même une approximabilité n'a pas non plus le dernier mot. Elle offre certe en temps polynomial de bonnes solutions mais cela ne signifie pas qu'une heuristique bien recherchée ne fera pas mieux sur la majorité des instances. En bref, en ce qui concerne les problèmes industriels, l'approximation polynomiale ne nous décharge pas d'effectuer de l'évaluation des performances de nos algorithmes. Elle constitue cependant un point de départ important à la conception d'algorithmes heuristiques. De même que ne pas chercher à savoir si un problème est polynomial avant de le résoudre à coup de programmation linéaire, ne pas chercher à savoir si un problème est approximable est possiblement un oubli dommageable.

Même en cas de forts résultats d'inapproximabilité sur les problèmes, des alternatives avec garantie existent. On peut en citer plusieurs :

- les algorithmes d'approximation en temps faiblement exponentiel, pour lequel on s'autorise à sacrifier un peu de temps exponentiel si cela permet de réduire le rapport d'approximation ;
- les algorithmes d'approximation sur machine quantique, en supposant que ces machines parviennent un jour à se développer au niveau de nos machines déterministes, ce genre d'algorithme peut avoir un intérêt certain ;
- la complexité paramétrées ou l'approximabilité paramétrée qui visent à mieux étudier les instances de nos problèmes pour en extraire les paramètres naturellement petits et qui influence grandement sur le temps de calcul de nos algorithmes.

Annexe A

Exercices

Dans cette section sont donnés des exercices. La première partie donne les intitulés des exercices, sans aucun indice ni question intermédiaire. Ces exercices sont donc, à priori, difficiles. La partie suivante donne des indices ou découpe l'exercices en sous-questions.

A.1 Approximations polynomiales

Dans cette section, les exercices portent essentiellement sur les techniques vue dans les chapitres 1 et 3.

Sujets

Exercice 1
Le problème Max SAT consiste, connaissant une formule booléenne sous forme normale conjonctive, à trouver une affectation des variables qui satisfait un nombre maximum de clauses. Montrer que ce problème est $\frac{1}{2}$-approximable en temps polynomial.

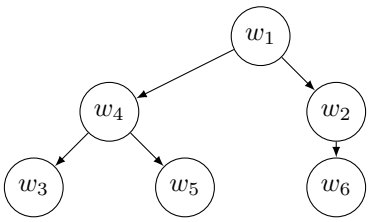
Exercice 2
On considère le problème Sous-graphe sans circuit maximum dans lequel, connaissant un graphe orienté $G = (V, A)$, on recherche un sous-graphe $G' = (V, A')$ sans circuit et possédant le plus d'arcs possible. Montrer qu'il existe une $\frac{1}{2}$-approximation polynomiale pour ce problème.

Exercice 3
On considère le problème de Couverture par sommets (Vertex Cover en anglais) : connaissant un graphe $G = (V, E)$ non orienté, trouver un sous-ensemble de nœuds V' de taille minimum incident à toutes les arêtes de E. Montrer qu'un algorithme calculant un couplage maximal peut être adapté pour construire une 2-approximation polynomiale pour ce problème. *On rappelle qu'un couplage est un ensemble d'arêtes ne partageant pas d'extrémité.*

Exercice 4
On considère le problème de l'arborescence de Steiner (Directed Steiner Tree en anglais) : connaissant un graphe orienté $G = (V, A)$, un nœud $r \in V$ racine, des nœuds $X \subset V$ terminaux et des poids $\omega : A \to \mathbb{N}^*$ sur les arcs, chercher une arborescence T reliant r à X et dont la somme des poids des arcs est minimum. Montrer qu'il existe une k-approximation polynomiale pour ce problème, où $k = |X|$.

Exercice 5
On considère le problème d'ordonnancement Minimum-Makespan suivant : connaissant m machines identiques et n tâches de durées $t_1, t_2, \ldots, t_n$, attribuer chaque tâche i à une machine $1 \leq M_i \leq m$ telle que $\max T_j$ est minimum si on note $T_j = \sum_{i:M_i=j} t_i$ le temps total d'exécution sur la machine j. Montrer que ce problème est 2-approximable.

Exercice 6
Dans le problème Binary Tree Knapsack, on dispose, comme pour le problème de sac à dos d'un sac de volume V et de n objets de volumes $v_1, v_2, \ldots, v_n$ d'utilisé $u_1, u_2, \ldots, u_n$. En entrée est également donnée une arborescence T binaire avec n nœuds $w_1, w_2, \ldots, w_n$. On peut mettre les objets dans le sac, mais, lorsqu'on sélectionne le i^{e} objet pour le mettre dans le sac, alors il faut aussi mettre dans le sac tout objet j tel que w_j est sur le chemin reliant la racine de T à w_i. Par exemple, dans l'image ci-dessous, choisir l'objet 5 impose de choisir également l'objet 1 et l'objet 4. Notez que l'objet 1 est toujours dans le sac, sauf s'il est vide. On veut maximiser la somme des utilités des objets dans le sac, de sorte que la somme des volumes de ces objets ne dépasse pas V. Montrez qu'il existe un FPTAS pour ce problème (pour tout $\varepsilon > 0$, il existe un algorithme d'approximation polynomial de rapport $(1 + \varepsilon)$ dont le temps de calcul augmente polynomialement avec $\frac{1}{\varepsilon}$).



Exercice 7
Le problème de Stable maximum consiste, connaissant un graphe $G = (V, E)$ non orienté à trouver un stable de taille maximum dans G, c'est à dire un ensemble $V' \subset V$ dont les nœuds ne sont pas voisins deux à deux. Montrer qu'il existe un PTAS pour ce problème si on se restreint aux graphes planaires (pour tout $\varepsilon > 0$, il existe un algorithme d'approximation polynomial de rapport $(1 + \varepsilon)$).

On admettra le résultat suivant : si $k \in \mathbb{N}$ et G est un graphe k-outerplanar [1], alors il existe un algorithme en temps $O(8^k n)$ qui construit un stable de taille maximum de G.

Exercice 8
On considère le problème de 2-(Sac à dos) maximum suivant : connaissant n objets de volume $v_1, v_2, \ldots, v_n$ et 2 sacs de tailles V_1 et V_2, mettre le plus d'objets possible dans les deux sacs sans dépasser le volume d'aucun sac. Comme dans le problème de sac à dos, les objets sont insécables. Montrez qu'il existe un algorithme d'approximation polynomial de rapport absolu 1 pour ce problème.

Exercice 9
On considère le problème d'équilibrage suivant : n objets d'utilité $u_1, u_2, \ldots, u_n$. On souhaite partitionner ces objets en p ensembles $L_1, L_2, \ldots, L_p$. On note $U_j = \sum_{i \in L_j} u_i$ la somme des utilités des objets dans L_i et U la somme de toutes les utilités. On veut minimiser $\frac{\max U_i - \min U_i}{U}$.

On admettra qu'il existe un algorithme pseudo-polynomial $\mathcal{A}$ pour résoudre exactement ce problème, en temps $O(npU^5)$. Montrez que, pour tout $\varepsilon > 0$, il existe un algorithme d'approximation polynomial de rapport absolu ε pour ce problème.

Exercice 10
Le problème Set splitting consiste, connaissant un ensemble X, une sous-ensemble $S \subset 2^X$ de parties de X, à trouver une partition $X_1 \uplus X_2$ de X maximisant le nombre d'ensembles de S possédant à la fois des éléments de X_1 et de X_2. Montrer qu'il existe une $\frac{1}{2}$-approximation polynomiale probabiliste pour ce problème qui peut être déterminisée.

Exercice 11
On considère, comme dans le chapitre 3 le problème Max SAT avec des poids.

Montrer que l'algorithme suivant est une $\frac{3}{4}$-approximation probabiliste pour le problème :

1: Produire une solution avec l'algorithme 13 du chapitre 3.
2: Produire une solution en mettant chaque variable à vrai avec une probabilité $\frac{1}{2}$.
3: Renvoyer la meilleure des deux solutions.

Exercice 12
Proposez une 2-approximation polynomiale déterministe d'arrondi de programme linéaire pour le problème suivant généralisant le problème de couverture par sommet.

Soit $G = (V, E)$ un graphe non orienté. L'objectif est de couvrir toutes les arêtes. On peut pour cela poser des jetons sur les nœuds. Chaque arête e est associée à un entier d_e. Pour couvrir e, il faut poser au total d_e jetons sur les deux nœuds incidents à e. Chaque nœud v est associé à un poids ω_v. Chaque jeton posé sur v coûte ω_v. Il faut minimiser le coût total des jetons posés sur le graphe.

Exercice 13
Remontrez le résultat de l'approximation de l'exercice 3 en utilisant l'approche Dual Fitting.

Exercice 14
Le problème de multicoupe minimum dans un arbre consiste, connaissant un arbre $T = (V, E)$ et k paires de nœuds $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ et des capacités sur les arêtes $c : E \to \mathbb{R}^+$, à trouver un sous-ensemble d'arêtes E' de E séparant s_i de t_i pour $i \in [\![1; k]\!]$ et dont la somme des capacités est minimum.

On considère la modélisation par un programme linéaire suivante où p_i est l'unique chaîne de T reliant s_i et t_i.

1. Un graphe k-outerplanar est planaire et, pour chaque nœud, on peut tracer au stylo une courbe qui le relie à la face extérieure en croisant au plus k arêtes. Autrement dit, un graphe où tout nœud est sur la face extérieure est (1-)outerplanar. Un graphe qui serait outerplanar si on retire les nœuds de sa face extérieure est 2-outerplanar. Et ainsi de suite...

Programme linéaire (PP9) : Programme linéaire pour le problème de multicoupe dans un arbre

Minimiser	$\sum_{e \in E} c(e) d_e$			
s.c.	$\sum_{e \in p_i} d_e$	$\geq$	1	$\forall i \in [\![1; k]\!]$
	d_e	$\in$	$0, 1$	$\forall e \in E$

Ecrire le dual relâché de (PP9). On notera f_i la variable duale associée à la contrainte du couple (s_i, t_i). Considérer ensuite l'algorithme suivant :

1: Orienter T depuis n'importe quel nœud r
2: Fixer f à 0 et E' à $\emptyset$.
3: Ordonner les couples (s_i, t_i) en fonction de la distance de r à la chaîne p_i, par distance décroissante
4: **Pour** $i \in [\![1; k]\!]$ dans l'ordre précédemment construit **Faire**
5: Augmenter f_i jusqu'à saturer une contrainte
6: Ajouter à E' les arêtes dont la contrainte est saturées
7: **Pour** $e \in E'$ par ordre inverse d'ajout à E' **Faire**
8: **Si** $E' \backslash e$ est une coupe **Alors**
9: Supprimer e de E'

Montrer, en utilisant un schémas primadual, que cet algorithme est une 2 approximation polynomiale.

Indices

Indice de l'exercice 1
Connaissant l'affectation où toute variable est vraie, on suppose que cette affectation satisfait strictement moins de la moitié des clauses, cherchez comment transformer cette affectation pour en satisfaire plus de la moitié des clauses.

Indice de l'exercice 2
On rappelle qu'un graphe sans circuit possède un ordre topologique de ses nœuds. Utiliser une méthode similaire à celle de l'exercice 1.

Indice de l'exercice 3

1. Montrer que, connaissant un couplage M maximal, les extrémités de M forment une solution réalisable.
2. On note V^* une solution optimale. Montrer que $|M|$ est une borne inférieur de $|V^*|$.
3. En déduire le résultat.

Indice de l'exercice 4
On note $\omega(H)$ la somme des poids des arcs de H, pour tout sous graphe H de G.

1. Soit T^* une solution optimale. Soit $v \in X$, montrer que le poids d'un plus court chemin de r à v est une borne inférieure de $\omega(T^*)$.
2. Montrer que si un graphe $G' \subset G$ possède un chemin de r vers v pour tout $v \in X$, il est possible d'en déduire en temps polynomial une solution réalisable T de poids inférieur à $\omega(G')$
3. En déduire le résultat

Indice de l'exercice 5

1. Montrer que $\frac{1}{m}\sum_{i=1}^{n} t_i$ est une borne inférieure de la valeur de la solution optimale.
2. Montrer que l'algorithme glouton qui place tour à tour les tâches 1 à n sur la machine dont le temps total d'exécution est minimum est une 2-approximation polynomiale

Indice de l'exercice 6

1. Montrer qu'il existe un algorithme pseudopolynomial pour résoudre ce problème dont la complexité dépend de n et de l'utilité des objets. On utilisera la fonction suivante : $g(i, u)$ est le volume minimum nécessaire que l'on doit utiliser pour atteindre exactement l'utilité u avec les objets descendant de i dans T ; cette fonction vaut $+\infty$ si l'utilité u est inatteignable.
2. En utilisant la même technique que pour le problème de sac à dos, construire un FPTAS pour ce problème.

Indice de l'exercice 7
On considère un plongement planaire du graphe dans le plan. On note V_1 les nœuds de la face extérieure, V_2 les nœuds de la face extérieure si on retire V_1, ..., V_i les nœuds de la face extérieure si on retire V_1, V_2, ..., V_{i-1} et ainsi de suite jusqu'à ne plus avoir de nœud dans le graphe.

1. Soit $k \in \mathbb{N}$ et $j \in [\![0; k]\!]$ et $\bar{V_j} = \{V_i | i \equiv j \bmod k+1\}$, on note $\bar{G_j}$ le graphe G privé des nœuds de $\bar{V_j}$, montrer que $\bar{G_j}$ est k-outerplanar.
2. Soit V^* un stable maximum de G, montrer qu'il existe $j \in [\![0; k]\!]$ tel que $|\bar{V_j} \cap V^*| \leq \frac{|V^*|}{k+1}$. En déduire qu'il existe $j \in [\![0; k]\!]$ tel que le stable maximum de $\bar{G_j}$ possède au moins $\frac{k \cdot |V^*|}{k+1}$ nœuds.
3. En déduire un PTAS pour le problème de Stable maximum dans les graphes planaires.

Indice de l'exercice 8
Considérez l'algorithme suivant : trier les objets du plus petit au plus grand, on suppose donc $v_1 \leq v_2 \leq \cdots \leq v_n$ placer les dans l'ordre dans le premier sac jusqu'à ce qu'on ne puisse plus mettre d'objet, recommencer avec le second sac.

1. Montrer que s'il existe une solution réalisable avec r objets alors $\sum_{i=1}^{r} v_i \leq V_1 + V_2$.
2. Soit r le dernier objet placé dans un sac par l'algorithme. Si $n \geq r+2$, montrer que $V_1 + V_2 < \sum_{i=1}^{r+2} v_i$.
3. En déduire que cet algorithme est un algorithme d'approximation polynomial de rapport absolu de 1.

Indice de l'exercice 9

1. Considérer $K = \frac{\varepsilon U}{2n}$ et l'instance où toutes les utilités u_i sont remplacées par $\lfloor \frac{u_i}{K} \rfloor$. Et appliquer $\mathcal{A}$ à cette instance. Montrer que cet algorithme est polynomial.
2. On note $L_1, L_2, \dots L_p$ la solution calculée par $\mathcal{A}$, c'est-à-dire la solution optimale dans l'instance où les utilités sont réduites. Noter que cette solution est une solution des deux instances (avec les utilité u_i et avec les utilités réduites). Cependant, l'ensemble L_j maximisant (respectivement minimisant) $\sum_{i \in L_j} u_i$ n'est pas nécessairement l'ensemble maximisant (resp. minimisant) $\sum_{i \in L_j} \lfloor \frac{u_i}{K} \rfloor$.

   On note

$$p_A = \arg\min(\sum_{i \in L_j} u_i) \qquad U_A = \sum_{i \in L_{p_A}} u_i$$
$$p_B = \arg\max(\sum_{i \in L_j} u_i) \qquad U_B = \sum_{i \in L_{p_B}} u_i$$
$$p_C = \arg\min(\sum_{i \in L_j} \left\lfloor \frac{u_i}{K} \right\rfloor) \qquad U_C = \sum_{i \in L_{p_C}} \left\lfloor \frac{u_i}{K} \right\rfloor$$
$$p_D = \arg\max(\sum_{i \in L_j} \left\lfloor \frac{u_i}{K} \right\rfloor) \qquad U_D = \sum_{i \in L_{p_D}} \left\lfloor \frac{u_i}{K} \right\rfloor$$

   Montrer que $U_B - U_A \leq K \cdot (U_D - U_C) + K \cdot n$.
3. On note $L_1^*, L_2^*, \dots L_p^*$ une solution optimale (dans l'instance d'origine).
   On note

$$p_E = \arg\min(\sum_{i \in L_j^*} u_i) \qquad U_E = \sum_{i \in L_{p_E}^*} u_i$$
$$p_F = \arg\max(\sum_{i \in L_j^*} u_i) \qquad U_F = \sum_{i \in L_{p_F}^*} u_i$$
$$p_G = \arg\min(\sum_{i \in L_j^*} \left\lfloor \frac{u_i}{K} \right\rfloor) \qquad U_G = \sum_{i \in L_{p_G}^*} \left\lfloor \frac{u_i}{K} \right\rfloor$$
$$p_H = \arg\max(\sum_{i \in L_j^*} \left\lfloor \frac{u_i}{K} \right\rfloor) \qquad U_H = \sum_{i \in L_{p_H}^*} \left\lfloor \frac{u_i}{K} \right\rfloor$$

   Montrer que $U_D - U_C \leq U_H - U_G$.
4. Montrer que $U_H - U_G \leq \frac{1}{K}(U_F - U_E) + n$.
5. En déduire le résultat demandé.

Indice de l'exercice 10

1. On suppose dans un premier temps que tout ensemble est de taille au moins 2. Quelle est la probabilité p_s qu'un ensemble $s \in S$ possède un élément de X_1 et un élément de X_2 si on construit la partition en mettant chaque élément de X dans X_1 ou X_2 avec une probabilité $\frac{1}{2}$.
2. Soit m^* la valeur d'une solution optimale. En déduire que cet algorithme construit une solution dont l'espérance de la valeur dépasse $\frac{1}{2}m^*$.
3. Généraliser ce résultat au cas où il existe des ensembles de taille 0 ou 1.
4. Utiliser la technique utilisée avec Max SAT au chapitre 3 pour déterminiser cet algorithme.

Indice de l'exercice 11

1. Quelle est la probabilité q_j qu'une clause C_j ne soit pas satisfaite avec la seconde solution ?
2. Soit z° une solution optimale du programme relâché (RSATP7). Soit p_j la probabilité que la clause C_j soit satisfaite avec la première solution. Montrer que $p_j + q_j \geq \frac{3}{2} z_j^\circ$. Considérez deux cas selon la taille des clauses.

3. Quelle est la moyenne des espérances des deux solutions ? Montrer que cette moyenne est supérieure à $\frac{3}{4}$ fois le poids de z°.
4. Conclure.

Indice de l'exercice 12

1. Ecrire un programme linéaire. Il faut se baser sur les programmes linéaires pour le problème de couverture par ensembles. Chaque nœud est associé à une variable entière, il y a une contrainte par arête.
2. Montrer que, dans une solution réalisable de la relaxation linéaire du programme, pour chaque arête $e = (u, v)$, au moins une des variables associées à u ou v vaut au moins $\frac{d_e}{2}$.
3. En déduire une 2-approximation polynomiale.

Indice de l'exercice 13

1. Ecrire le dual du programme suivant

Programme linéaire (PP10) :

Minimiser	$\sum_{v \in V} x_v$			
s.c.	$x_u + x_v$	$\geq$	1	$\forall e \in E$
	x_v	$\in$	$\{0; 1\}$	$\forall v \in V$

On notera y les variables duales. On peut remarquer qu'un s'agit d'un programme linéaire pour calculer un couplage maximum de G.

2. Montrer que l'algorithme suivant construit une solution réalisable du dual et du primal : tant que x n'est pas solution du primal, trouver une contrainte $x_u + x_v \geq 1$ violée, fixer $x_u = x_v = 1$ et $y_{uv} = 1$. Montrez qu'à l'issue de l'algorithmique x et y sont des solutions rélisables du primal et du dual
3. Montrer enfin que le poids de x est inférieur au double du poids de y et conclure.

Indice de l'exercice 14

1. Ecrire le programme dual.
2. Soit v le nœud le plus proche de r dans la chaîne p_i. Montrer que, si $f_i \neq 0$, l'algorithme ne peut pas sélectionner dans E' plus d'une arête entre s_i et v. De même pour t_i et v.
3. Montrer que E' et f vérifient les conditions d'écart complémentaires primales et duales relâchées avec $\alpha = 1$ et $\beta = 2$.
4. Conclure

A.2 Classes d'approximabilité

Dans cette section, les exercices portent essentiellement sur les techniques vue dans les chapitres 1 et 3.

Sujets

Exercice 15

1. Ecrire les problèmes suivants sous forme de problème d'optimisation en précisant le quadruplet $\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O}$.
   a) (1-Centre). Connaissant un graphe $G = (V, E)$ non orienté, un nœud $v \in V$, on note $d(v, s)$ la taille d'une plus court chaîne de v à s. Soit un graphe $G = (V, E)$, trouver le nœud $s \in V$ qui minimise $\max_v(d(v, s))$.
   b) (Plus petite sous-chaîne). Soit un ensemble $s_1, s_2, \ldots, s_n$ de chaînes de caractères sur un alphabet binaire. Trouver la plus petite chaîne de caractère binaires de taille inférieure à $\sum_{i=1}^{n} |s_i|$ qui contient toutes ces chaînes comme sous-chaînes.
   c) (Distance d'édition à une matrice inversible). Soit une matrice M de $n \times n$ entiers, trouver une matrice P de taille $n \times n$ inversible et la plus proche possible de M au sens de la norme 1 (la somme des différences terme à terme des deux matrices).
   d) (Bloqueur de dominant). Connaissant un graphe $G = (V, E)$, on rappelle qu'un ensemble dominant est un ensemble voisin de tous les nœuds de V (chaque nœud a au moins un voisin dans le dominant). Un dominant est minimum si sa taille est la plus petite possible parmi tous les dominants de G. Soit un graphe $G = (V, E)$ non orienté et un entier k, trouver un ensemble $S \subset V$ de taille $|S| \leq k$ qui intersecte le plus d'ensemble dominants minimum possibles.
2. Montrer que le problème 1-Centre est dans PO, que le deuxième est dans NPO, que le troisième n'est pas dans NPO et le quatrième n'est pas dans NPO si $P \neq NP$. On admettra qu'on peut encoder les objets mathématiques classiques (entiers, rationnels, graphes, matrices, listes, ...) et reconnaître cet encodage en temps polynomial.

Exercice 16
En reprenant les problèmes des exercices de la section précédentes, dans quelle classe d'approximabilité pouvez vous mettre ces problèmes compte tenu des approximations polynomiales démontrées ? On admettra que tous ces problèmes sont dans NPO ou, si ce n'est pas le cas, qu'une variante très proche, où seul l'ensemble des solutions réalisables est réduit, est dans NPO.

Exercice 17
Soit $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \mathcal{O}) \in$ NPO dont le problème de décision est NP-Difficile. On suppose que, pour toute entier $a \in \mathbb{N}$, il existe une transformation polynomiale $f_a : \mathcal{I} \to \mathcal{I}$ telle que, pour toute instance $x \in \mathcal{I}$, si on pose $x' = f_a(x)$, il existe une solution $y \in \mathcal{S}(x)$ de mesure $\mathcal{M}(y) = m$ si et seulement s'il existe une solution $y' \in \mathcal{S}(x')$ de poids $\mathcal{M}(y') = a \cdot m$.

Montrer que, sauf si P = NP, alors, $\Pi \notin$ AAPX.

Exercice 18
Soit $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \min) \in$ NPO. On suppose que $M(x, y) \in \mathbb{N}^*$ pour tout $x \in \mathcal{I}$ et $y \in \mathcal{S}$. Pour tout entier $k \in \mathbb{N}^*$, on définit le problème $k - \Pi_D$ qui consiste, connaissant une instance $x \in \mathcal{I}$, à déterminer si $\mathcal{M}^*(x) \leq k$.

— Montrez que si P $\neq$ NP et, pour tout $k \in \mathbb{N}$, $k - \Pi_D$ est NP-Complet alors $\Pi \notin$ APX.
— Montrer que si $\Pi \in$ PTAS alors pour tout $k \in \mathbb{N}^*$, le problème $k - \Pi_D$ est dans P.

Exercice 19
On définit la classe ln-PTAS comme étant l'ensemble des problèmes Π de minimisation de NPO tels que, pour tout $\varepsilon > 0$, Π est approximable en temps polynomiale avec un rapport $\ln((1 + \varepsilon) \cdot n)$ (où n est la taille de l'instance). Soit $\Pi \in$ ln-PTAS, montrez que s'il existe une constante k telle que, pour toute instance x de Π, $\mathcal{M}^*(x) \leq k$, alors $\Pi \in \ln - \text{APX}_\infty$.

Exercice 20
Montrer que AAPX $\subset$ FPTAS$_\infty$. En déduire que, si P $\neq$ NP, alors FPTAS$_\infty \not\subset$ PTAS. Montrer enfin qu'il existe des problèmes NP-Difficiles au sens fort qui sont dans FPTAS$_\infty$.

Exercice 21
On note $A = (n \to \min(1, |\cos(n)| \cdot n)) - \text{APX}$, $B = (n \to \min(1, |\sin(n)| \cdot n)) - \text{APX}$. Montrez qu'il existe $c < 1$ tel que $A \cap B \subset (n \to c \cdot n) - \text{APX}$.

Exercice 22
Soit un problème $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \min) \in NPO$ tel que, pour tout $x \in \mathcal{I}$, $\mathcal{S}(x) \neq \emptyset$ et tel qu'il existe un algorithme polynomial $\mathcal{A}$ qui, connaissant $x \in \mathcal{I}$ renvoie une solution réalisable de x. Montrez qu'il existe $c \in \mathbb{N}$ tel que $\Pi \in (n \to 2^{n^c}) - \text{APX}$.

Exercice 23

1. On considère un problème de maximisation $\Pi = (\mathcal{I}, \mathcal{S}, \mathcal{M}, \max)$. On définit le problème de minimisation Π' suivant : soit $x_1, x_2, \ldots, x_n$ des instances de Π, trouver x_i qui minimise $\mathcal{M}^*(x_i)$. Montrez que si $\Pi \in \mathcal{APX}$ alors il existe une constante r et un algorithme polynomial qui, connaissant une instance x' de Π' renvoie une solution de valeur au plus r fois la valeur optimale de x'.
2. Pourquoi ne peut-on pas dire que Π' est dans APX ?

Indices

Indice de l'exercice 15

1. Pas besoin d'indice ici, il s'agit d'appliquer le cours.
2. a) Attention, trouver un algorithme polynomial pour résoudre le problème de construction n'est pas suffisant.
   b) Appliquez la définition du cours
   c) Combien y a-t-il de solutions réalisables d'une instance ?
   d) Combien de temps prend le calcul de la mesure ?

Indice de l'exercice 16

Reprenez le cours.

Indice de l'exercice 17

Reprenez le principe de la démonstration du théorème 2.2.6, page 38.

Indice de l'exercice 18

Montrer que si $k - \Pi_D$ est NP-Complet, alors si $P \neq NP$, il existe un rapport r_k dépendant uniquement de k tel que Π n'est pas approximable en temps polynomial avec un rapport r_k. Déduire de ce résultat la réponse aux deux questions.

Indice de l'exercice 19

Pour quelles valeurs de ε peut-on montrer que $\ln((1+\varepsilon)\cdot n)\mathcal{M}^*(x) < \ln(n)\mathcal{M}^*(x) + 1$?

Indice de l'exercice 20

Que devient la définition de la classe FPTAS_∞ quand ε vaut 0 ? Pour la suite, réfléchissez à un problème évoqué dans le cours qui n'a pas de PTAS mais qui est dans AAPX.

Indice de l'exercice 21

1. Cherchez une borne supérieure de $\min(|\sin(n)|, |\cos(n)|)$.
2. En déduire la réponse à la question

Indice de l'exercice 22

1. Quel sont les poids minimum et maximum d'une solution réalisable d'une instance x de Π ?
2. En déduire que $\mathcal{A}$ est l'approximation recherchée.

Indice de l'exercice 23

1. Si une c-approximation polynomiale existe pour Π, montrer que si on applique l'approximation sur chacune des instances x_i de l'instance, on obtient une solution $\frac{1}{c}$ approchée.
2. Quel est le premier critère à respecter pour être dans APX ?

A.3 Inapproximabilité

Dans cette section, les exercices portent essentiellement sur les techniques vue dans les chapitres 4.

Exercice 24
Montrer que le rapport de l'algorithme proposé dans l'indice de l'exercice 3 est serré.

Exercice 25
Montrer que le rapport de l'algorithme proposé dans l'indice de l'exercice 4 est serré.

Exercice 26
On considère le problème de coloration minimum dans un graphe quelconque : connaissant un graphe $G = (V, E)$, colorier ses sommets de sorte que deux voisins n'aient pas la même couleur. On applique l'algorithme suivant : on définit un ordre arbitraire des sommets, on colorie ensuite chaque nœud dans l'ordre, avec la première couleur disponible. Cet algorithme est une $\frac{d+1}{d}$-approximation polynomiale où d est le degré maximum des nœuds de G. Montrer que ce rapport est serré.

Exercice 27
On considère les problèmes de Clique maximum et de couverture par sommets qui consistent, connaissant un graphe $G = (V, E)$ à trouver une clique de taille maximum dans le graphe et à trouver une couverture des arêtes par les sommets de taille minimum dans G. Connaissant un graphe $G = (V, E)$, instance de clique maximum, on définit $f(G)$ comme étant le graphe $G' = (V, F)$ complémentaire de G (où toutes les arêtes et non-arêtes ont été inversées). Connaissant une couverture par sommet V' de G', on définit $g(V') = V \backslash V'$.

Montrer que f et g sont polynomiales, que $g(V')$ renvoie bien une solution réalisable de G et qu'on vérifie la contrainte $|\mathcal{M}(G, g(V')) - \mathcal{M}^*(G)| = |\mathcal{M}'(f(G), V') - \mathcal{M}'^*(f(G))|$ où $\mathcal{M}$ et $\mathcal{M}'$ sont respectivement les mesures associées aux problèmes de clique et de couverture par sommets. Montrer que, toutefois, ces fonctions ne permettent pas de construire une L-réduction de Clique maximum vers Couverture par sommets avec $\alpha = \beta = 1$. Pour quelles valeurs de β et α la réduction fonctionne-t-elle ?

Exercice 28
Le problème de coupe maximum consiste, connaissant un graphe $G = (V, E)$, à partitionner V en deux ensembles V_1 et V_2 de sorte à maximiser le nombre d'arêtes reliant V_1 et V_2. Le problème Max 3SAT consiste en la restriction de Max SAT à des instances où chaque clause ne contient pas plus de 3 littéraux. Montrer qu'il existe une L-réduction de coupe maximum vers Max 3SAT.

Exercice 29

1. Le problème de stable maximum consiste, connaissant un graphe $G = (V, E)$, à trouver un stable de taille maximum dans G. Montrer qu'il existe une réduction isofacteur de Max SAT vers Stable maximum avec $\alpha = \beta = 1$. En déduire une preuve que Stable maximum et Clique maximum n'ont pas de PTAS sauf si P = NP.
2. Soit k un entier. Considérer la fonction f_k suivante qui prend en entrée un graphe $G = (V, E)$ et renvoie le graphe $G_k = (V_k, E_k)$ produit cartésien de G par lui-même k fois. Autrement dit, G_k contient un nœud pour chaque k-tuple $(v_1, v_2, \ldots, v_k)$ de nœuds de V et relie 2 k-tuples $(v_1, v_2, \ldots, v_k)$ et $(v'_1, v'_2, \ldots, v'_k)$ par une arête si $(v_i, v'_i) \in E$ ou si $v_i = v'_i$. Montrer, à l'aide de cette transformation et de la question précédente, que Clique Maximum n'est pas dans APX sauf si P = NP.

Exercice 30
Montrez que la réduction isofacteur, dans le cas restreint où α et β sont constants, est transitive.

Exercice 31
On considère le problème d'équilibrage suivant : n objets d'utilité $u_1, u_2, \ldots, u_n$. On souhaite partitionner ces objets en p ensembles $L_1, L_2, \ldots, L_p$. On note $U_j = \sum_{i \in L_j} u_i$ la somme des utilités des objets dans L_i et U la somme de toutes les utilités. On veut minimiser $\max U_i - \min U_i$.

Montrer qu'il existe une GAP-Réduction depuis le problème de partition d'entiers (connaissant n entiers $x_1, x_2, \ldots, x_n$, trouver une partition de ces entiers en deux ensembles de sommes égales) vers ce problème.

Exercice 32
Le problème de chemins disjoints consiste, dans un graphe orienté, connaissant 4 nœuds s_1, t_1, s_1, t_2, à savoir s'il existe 2 chemins P_1, P_2 nœuds-disjoints tels que P_i relie s_i à t_i. Il s'agit d'un problème NP-Complet.

On considère le problème d'optimisation suivant : connaissant un graphe orienté complet $G = (V, A)$, trois nœuds r, x_1, x_2 de V et des poids $\omega : A \to \mathbb{N}^+$ sur les arcs, trouver un chemin élémentaire démarrant en r, passant par x_1 et x_2 dans n'importe quel ordre et de poids minimum.

Montrer qu'il existe une réduction GAP du premier problème vers le second.

Exercice 33
On considère le problème Maximum Weighted Satisfiability-FS où on se restreint aux instances où chaque clause contient le même littéral x. Ainsi, toute instance possède une solution réalisable consistant à mettre la variable x à VRAI. Montrer que ce problème est $O(2^{n^c})$-APX complet.

Exercice 34
Montrer qu'il existe un vérifieur probabiliste polynomial pour le problème suivant qui, si la réponse est OUI, accepte au moins un certificat avec une probabilité de 1, et sinon refuse chaque certificat avec une probabilité au moins $\frac{1}{n^2}$. Ce vérifieur devrait générer au plus un nombre logarithmique de bits aléatoires et lire au plus un nombre constant de bits du certificat.

Soient 3 ensembles X, Y, Z disjoints, tous les 3 de taille n et un ensemble M de triplets de $X \times Y \times Z$. Peut-on couvrir tout X, Y, Z avec des triplets de M disjoints ? Autrement dit, existe-t-il n triplets de M disjoints ?

Indices

Indice de l'exercice 24
Trouver une instance où ne pendre que la moitié des nœuds d'un couplage maximum permet de couvrir l'ensemble du graphe.

Indice de l'exercice 25
Trouver une instance où l'arborescence possède un plus court chemin distinct de la racine vers chaque terminal, mais où dans l'arbre optimal, le chemin de la racine vers chaque terminal n'est pas un plus court chemin.

Indice de l'exercice 26

1. Montrer que le nombre chromatique d'un cycle impair est 3.
2. Considérer un cycle impair de taille $2n+1$ et rajouter deux arêtes à ce cycle de sorte qu'il soit toujours 3-coloriable, qu'il soit de degré maximum 3 et qu'il existe un ordre des nœuds où l'algorithme glouton présenté dans le sujet de l'exercice propose une 4 coloration
3. En déduire le résultat.

Indice de l'exercice 27

1. Montrer que si V' est une couverture par sommet de G alors $V - V'$ est un stable de G, en déduire les réponses à la première partie.
2. Chercher un exemple de graphe où la première contrainte de la L réduction n'est pas vérifiée.

Indice de l'exercice 28
Considérer la fonction f suivante qui connaissant un graphe G, produit une instance de Max 3-SAT : ajouter une variable x_v par nœud v et une variable x_e par arête e, et créer 5 clauses par arête $e = (u, v)$:
— une clause x_e
— quatre clause décrivant les 4 cas possibles pour l'arête, par exemple $x_u \vee \bar{x_v} \vee x_e$ signifiant que u et v ne sont pas dans le même ensemble et donc qu'on comptabilise l'arête e

1. Montrer que f est polynomiale
2. Connaissant une affectation a des variables, comment la transformer, en temps polynomial, en affectation a' qui satisfait au moins 4 clauses par arête ?
3. Définir une fonction g polynomiale, qui, connaissant une affectation a, renvoie une solution réalisable $g(a)$ de G.
4. On note $\mathcal{M}$ et $\mathcal{M}'$ les mesures associées aux problèmes de la coupe max et de Max 3-SAT. Montrer que $\mathcal{M}(G, g(a)) = \mathcal{M}'(f(G); a) - 4|E|$ où E sont les arêtes de G.
5. On suppose admis que $\mathcal{M}^*(G) \geq \frac{|E|}{2}$, déduire que f et g forment une L-réduction avec $\alpha = 9$ et $\beta = 1$.

Indice de l'exercice 29

1. Considérer la fonction f suivante qui, connaissant une instance x de Max SAT : pour chaque clause, construire une clique contenant un nœud par littéral de la clause. Puis relier deux nœuds correspondant à des littéraux de deux clauses distinctes s'ils sont la négation l'un de l'autre.
   a) Montrer qu'il existe une affectation des variables de x qui satisfait m clauses si et seulement s'il existe un stable de $f(x)$ de taille m.
   b) En déduire une fonction g telle que f et g forme une réduction isofacteur de Max SAT vers Stable maximum avec $\alpha = \beta = 1$.
   c) En déduire que Stable maximum n'a pas de PTAS sauf si P = NP.
   d) Montrer qu'un graphe de taille n a un stable de taille m si et seulement si son complémentaire a une clique de taille n
   e) En déduire que Clique maximum n'a pas de PTAS sauf si P = NP.
2. a) Montrer que G_k contient une clique de taille m^k si et seulement si G contient une clique de taille m.
   b) En déduire que s'il existait une r-approximation polynomiale pour Clique maximum, avec $r > 1$, alors on pourrait l'utiliser sur G_k pour produire une solution $r^{\frac{1}{k}}$-approchée de G.
   c) En choisissant bien k, montrer que cette r approximation peut donc servir à produire une $(1+\varepsilon)$-approximation pour le problème de clique maximum, pour tout ε.

Indice de l'exercice 30
Appliquez simplement les définitions. On peut choisir des problèmes de minimisation, maximisation ou un mélange des deux.

Indice de l'exercice 31
Considérer le cas où $p = 2$.

Indice de l'exercice 32
Connaissant une instance G du problème de chemins disjoints, rajouter une racine r prédécesseur de s_1, un terminal x_2 successeur de t_2 et un terminal x_1 successeur de t_1 et prédécesseur de s_2. On note G_2 ce graphe. Montrer que G possède deux chemins disjointes de s_1 à t_1 et de s_2 à t_2 si et seulement si G_2 possède un chemin élémentaire démarrant en r et couvrant x_1 et x_2. Utiliser ensuite une technique similaire à celle du problème de voyageur de commerce pour prouver la réduction GAP.

Indice de l'exercice 33

1. En utilisant le résultat de l'exercice 22, montrer que ce problème est dans $O(2^{n^c})$-APX.
2. Soit un problème Π de NPO, montrer que s'il n'existe pas d'algorithme polynomial pour déterminer si une instance de Π possède une solution réalisable, alors $\Pi \notin O(2^{n^c})$-APX
3. Avec le théorème 4.3.3, on sait qu'il existe une L-réduction (f, g) avec $\alpha = \beta = 1$ de tout problème Π de $O(2^{n^c})$-APX vers Maximum Weighted Satisfiability. Compléter f pour construire une L-réduction (f', g) avec $\alpha = \beta = 1$ de tout problème Π de $O(2^{n^c})$-APX vers Maximum Weighted Satisfiability-FS. (il n'est pas nécessaire de connaître f et g dans le détail pour répondre à cette question).

Indice de l'exercice 34
Considérer l'encodage suivant : m lignes de $3n$ bits, chaque ligne contient 3 tranches de n bits et correspond à un triplet, et chaque tranche contient exactement un bit à 1 correspondant à l'élément de X, Y ou Z qui est dans le triplet. Appliquez ensuite un vérifieur similaire à celui des exemples du cours. On supposera que le vérifieur ne reçoit que des certificats bien formés.