Langages formels (11X003) - Automne 2025

5. Algorithmes d'approximation (I)

Enseignant: Arnaud Casteigts (et F. Raynaud) *Assistant: Brian Pulfer*
Moniteurs: T. von Düring & A. Maendly

Que faire lorsqu'on est confronté à un problème NP-difficile ? Plusieurs options s'offrent à nous, parmi lesquelles : 1) le résoudre quand même optimalement, même si la complexité en temps est exponentielle. On parle alors d'algorithme exact ; 2) restreindre le type d'instances qu'on cherche à traiter, les cas particuliers étant parfois plus faciles ; 3) abandonner l'objectif de trouver une solution optimale, tout en cherchant à ne pas trop s'en éloigner.

Les **algorithmes d'approximation** s'inscrivent dans la troisième catégorie : on veut savoir à quel point on peut s'**approcher** de la solution optimale, tout en gardant une **complexité polynomiale** en temps. Cette approche s'adresse généralement aux problèmes NP-difficiles, même si on peut la décliner à d'autres niveaux.

La qualité que l'on peut atteindre varie selon les problèmes. Soit n la taille de l'entrée à traiter. Supposons que l'on a affaire à un problème de minimisation et appelons OPT la valeur de la solution optimale. Voici les trois niveaux d'approximabilité les plus classiques :

- ☺ k-approximation : la solution calculée n'excède pas k fois l'optimum. Par exemple, pour certains problèmes, il existe des algorithmes qui garantissent au plus $2 \cdot OPT$.
- ☺☺ $(1+\epsilon)$-approximation : c'est le meilleur scénario. On peut s'approcher arbitrairement près de OPT, ϵ étant un paramètre à choisir. Bien sûr, plus ϵ est petit, plus cela coûtera cher en temps, mais l'algorithme sera toujours polynomial. Par exemple (dans l'idée), le temps sera en $O(n^5)$ pour $\epsilon = 1/5$, et en $O(n^{12})$ pour $\epsilon = 1/12$, etc.
- ☹ inapproximable : c'est le pire scénario, où quel que soit k, il n'existe aucun algorithme qui garantit une k-approximation en temps polynomial.

La même classification s'applique pour les problèmes de maximisation, en inversant le facteur. Par exemple, une k-approximation garantit une solution de qualité OPT/k.

Il existe d'autres catégories d'approximabilité, mais celles-ci sont les principales. Classifier les problèmes NP-difficiles en fonction de leur approximabilité est un sujet de recherche très actif, qui a connu de nombreux développements au cours des vingts dernières années.

5.1 Deux problèmes emblématiques

Nous allons discuter de deux problèmes qui sont souvent donnés en exemples sur le sujet. Ce sont tous deux des problèmes de graphes.

5.1.1 Couplages (et discussions)

Soit un graphe $G = (V, E)$. Un **couplage** dans G (matching, en anglais) est un ensemble d'arêtes $M \subseteq E$ qui ne se touchent pas (elles ne partagent aucun sommet). Par exemple :

Note : l'ensemble $M = \emptyset$ est un couplage valide, mais pas très intéressant. On cherche généralement à **maximiser** la taille d'un couplage. Il y a deux notions de maximalité qu'il est important de distinguer : couplage maximal (pour l'inclusion) ou couplage maximum (dans l'absolu). Autrement dit :

- Maximale = On ne peut rien lui ajouter
- Maximum = Il n'en existe pas de plus grand.

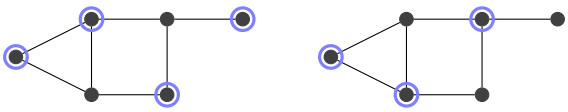
Ce n'est en effet pas pareil, par exemple, les deux solutions ci-dessus sont maximales, mais seulement celle de droite est maximum. Ainsi, une solution maximale n'est pas forcément maximum, mais une solution maximum est toujours aussi maximale. En l'occurrence, les deux versions du problème (Maximal Matching et Maximum Matching) peuvent être résolues en temps polynomial. L'algorithme pour Maximum Matching est assez célèbre et compliqué, il fait d'ailleurs l'objet d'une séance de cours complète en master. [1]. En revanche, pour Maximal Matching, un simple algorithme glouton suffit :

Initialiser $M \leftarrow \emptyset$.
Tant qu'il existe une arête qui ne touche aucune arête de M, on l'ajoute à M.

Nous allons utiliser cet algorithme pour trouver une 2-approx° à un problème NP-difficile.

5.1.2 Couverture par sommets

Soit un graphe $G = (V, E)$. Une **couverture par sommets** de G (vertex cover) est un ensemble de sommets $C \subseteq V$ qui (collectivement) touche toutes les arêtes. Par exemple :



Une solution triviale ici est $C = V$ (pas très intéressant). En général, on cherche à minimiser la taille d'une solution. On peut à nouveau distinguer une solution minimale (dont on ne peut rien enlever) et une solution minimum (aucune solution plus petite n'existe), la seconde étant à nouveau plus forte. Ici, la complexité des deux problèmes est très différente :

1. Pour les curieux : `https://arnaudcasteigts.net/files/graphalgo-6.pdf`

- Minimal Vertex Cover peut être résolu en temps polynomial
- Minimum Vertex Cover est NP-difficile !

5.2 Une 2-approximation pour la couverture minimum

Soit `minimal_matching()` une fonction qui exécute l'algorithme glouton précédent. On peut l'utiliser pour construire une couverture par sommet, comme suit :

```
vertex_cover(G) :
    C ← ∅
    M ← minimal_matching(G)
    Pour chaque arête uv de M :
        Ajouter u à C
        Ajouter v à C
    Renvoyer C
```

Il se trouve que cet algorithme est une 2-approximation pour Minimum Vertex Cover. Commençons par vérifier que la solution renvoyée est une couverture valide.

Lemme 5.1. *Chaque arête de G est couverte par au moins un sommet de C*

Preuve. Par l'absurde, supposons qu'il existe une arête uv qui n'est pas couverte. Cela implique que ni u ni v ne touche une arête du couplage utilisé (sinon, au moins l'un d'eux aurait été sélectionné). Mais alors, on aurait pu ajouter l'arête uv à ce couplage, qui n'était donc pas maximal (contradiction). □

Notons OPT la taille d'une solution optimale pour Minimum Vertex Cover.

Lemme 5.2. $|C| \leq 2 \cdot OPT$

Preuve. Par définition, une couverture par sommets doit couvrir toutes les arêtes de G. Elle doit donc, en particulier, couvrir toutes les arêtes d'un couplage maximal M. Mais ces arêtes étant disjointes, il faut au moins un sommet distinct pour couvrir chacune. Autrement dit, $|M| \leq OPT$. Via l'algorithme, on a $|C| = 2 \cdot |M| \leq 2 \cdot OPT$. □

Conclusion : L'algorithme ci-dessus garantit donc une 2-approximation pour Minimum Vertex Cover, alors qu'une solution exacte prend (présumément) un temps exponentiel.

Nous avons exploité ici un exemple de **dualité** entre deux problèmes, à savoir :

$$|\texttt{Maximal Matching}| \leq 2\times |\texttt{Minimum Vertex Cover}|$$

Beaucoup d'algorithmes d'approximation sont basées sur ce type de relations entre deux problèmes, l'un étant facile et l'autre non. On appelle cela des relations min-max.

5.3 Sac à dos

Dans le problème du sac à dos (KNAPSACK), une instance consiste en un ensemble de n objets qui ont chacun une valeur et un poids, ainsi qu'un poids total maximum (capacité du sac à dos). Par exemple : $\{(100, 20), (120, 30), (10, 10), (60, 10)\}$ et une capacité de 50. Le but est de trouver un sous-ensemble de valeur totale maximum sans dépasser la capacité.

Nous avons déjà vu dans un cours précédent [2] qu'une modification légère de l'algorithme glouton donne aussi une 2-approximation pour ce problème.

L'algorithme (pour rappel). Commencer par nettoyer l'instance de départ pour enlever tous les objets qui dépassent à eux seuls le poids total autorisé (ces objets ne servent à rien). On applique ensuite l'algorithme glouton comme indiqué précédemment, mais cette fois, on s'arrête dès qu'un objet x dépasse le seuil. À ce moment là, notre solution gloutonne cumule une certaine valeur V. L'objet refusé a lui-même une valeur v_x. Si $v_x < V$, on prend la solution gloutonne (en l'état), sinon, on prend l'objet x seul.

5.4 Pour aller plus loin

Un algorithme qui fournit une $(1+\epsilon)$-approximation est appelé un PTAS (polynomial-time approximation scheme, en anglais). Si un PTAS n'existe pas mais qu'on peut garantir une k-approximation (pour un certain k), alors le problème est dans la classe APX. Les problèmes qui n'admettent pas de PTAS sont dit APX-hard. Autrement dit, ils sont "au moins aussi difficiles" à approximer que n'importe quel problème dans APX.

Pour montrer qu'un problème est APX-hard, on procède généralement par réduction, comme pour montrer qu'un problème est NP-hard. Ici, cependant, on a besoin d'une réduction plus spécifique, qui préserve le ratio d'approximation lors de la transformation. Par exemple, on prend une instance d'un problème X (que l'on sait APX-hard) et on la transforme en une instance du problème Y, en montrant que si on arrive à approximer cette dernière, cela nous donne une approximation pour la première. Pouvoir approximer Y contredirait donc le fait que X est APX-hard, d'où le fait que Y est aussi APX-hard. Cela peut être utilisé par exemple pour montrer que la 2-approximation connue pour VERTEX COVER (présentée plus haut) est optimale, sauf si P=NP.

Quelques exemples de problèmes inapproximables sont CLIQUE ou INDEPENDENT SET.

2. `https://arnaudcasteigts.net/files/algo-glouton-1.pdf`