Algorithmique (12X001) - Automne 2023

12. Algorithmes d'approximation (2)

Enseignant: Arnaud Casteigts *Assistant: Brian Pulfer*

Dans ce cours, nous allons voir des exemples de problèmes pour chaque catégorie d'approximation. En fait, il s'agit de plusieurs versions d'un problème que vous connaissez bien, le *voyageur de commerce* (TSP).

Pour rappel, étant donné un ensemble de n villes et une fonction de coûts $c(u, v)$ entre chaque paire de villes, le voyageur de commerce consiste à trouver une tournée de coût total minimum qui visite chaque ville exactement une fois et revient au point de départ. Nous avons déjà parlé de ce problème pour illustrer l'approche gloutonne (avec des garanties plutôt mauvaises) et la programmation dynamique (plutôt bien, même si le temps reste exponentiel). Voyons à quel degré plusieurs versions du problème sont approximables en temps polynomial.

Dans tous les cas, nous appelons OPT le coût d'une tournée optimale.

12.1 TSP général

Dans le cas général, le TSP n'est pas approximable du tout (si P≠NP). Nous ne décrirons pas ce résultat, mais pour information, il peut s'obtenir en montrant que si on pouvait approximer le TSP en temps polynomial à un facteur constant (par exemple), cela permettrait de résoudre le problème du cycle hamiltonien, qui est NP-complet. Nous aurions donc montré au passage que P=NP.

12.2 TSP métrique

Dans la version *métrique* du TSP, on suppose que les coûts satisfont l'inégalité triangulaire : pour toutes villes u, v, w, on a $c(u, v) \leq c(u, w) + c(w, v)$. Autrement dit, il est toujours moins cher d'aller quelque part directement que d'y aller en passant par une ville intermédiaire. Cette hypothèse est réaliste pour de nombreuses applications. Cela nous permet aussi de voir l'instance du problème comme un graphe *complet* dont les arêtes sont pondérées par les coûts.

Le TSP métrique est toujours NP-difficile, mais il admet une 1.5-approximation en temps polynomial. Avant de la présenter, voyons d'abord une 2-approximation plus simple (que l'on étendra ensuite pour obtenir 1.5).

Algorithme de 2-approximation

1. Calculer un arbre couvrant T de poids minimum (MST)
2. Effectuer un parcours en profondeur de T en sautant les villes déjà rencontrées

Explications

Nous savons déjà qu'un MST peut être calculé en temps polynomial, par un algorithme glouton (algorithme de Kruskal, par exemple, c.f. Cours 6). Le parcours en profondeur est également rapide à effectuer.

Pourquoi cela donne-t-il une 2-approximation ? Observons d'abord le fait suivant :

Lemme 12.1. *coût(MST) ≤coût(tournée optimale)*

Preuve. Par l'absurde, si coût(tournée optimale) < coût(MST), alors il suffit d'enlever une arête à la tournée optimale pour obtenir un autre arbre couvrant (en forme de chemin) de coût inférieur à celui du MST, ce qui contredit l'optimalité du MST. □

Il suffit ensuite de réaliser qu'un parcours en profondeur traverse chaque arête deux fois, et les raccourcis que nous prenons ne peuvent qu'améliorer le coût (grâce à l'inégalité triangulaire). On a donc bien :

$$\text{coût(tournée approx)} \leq 2 \cdot \text{coût(MST)} \leq 2 \cdot \text{coût(tournée optimale)}$$

Algorithme de 1.5-approximation (Christofides, 1976)

Avant de donner l'algorithme, définissons deux concepts de graphes utiles :

- Couplage parfait (perfect matching) : couplage qui touche tous les *sommets*
- Cycle eulérien : cycle qui passe exactement une fois par chaque *arête*

Ces deux notions sont indépendantes, mais elles sont toutes deux utilisées dans l'algorithme. Quelques observations : Si un graphe est complet et le *nombre de sommets est pair* (cas qui nous intéressera), alors un couplage parfait existe forcément. De plus, on peut en trouver un de coût minimum en temps polynomial (non décrit ici). Un cycle eulérien existe dans un graphe si et seulement si tous les sommets ont un *degré pair*, et il peut être trouvé en temps polynomial aussi (de manière gloutonne).

Voici l'algorithme de Christofides :

1. Calculer un arbre couvrant T de coût minimum (MST)
2. Trouver les sommets de degrés *impairs* dans T, et calculer un couplage parfait C de coût minimum entre ces sommets dans le graphe d'origine.

3. Faire l'union de C et T (vu comme un multigraphe, s'il y a deux fois les mêmes arêtes)
4. Trouver un cycle eulérien dans cette union, qui n'a que des degrés pairs
5. Parcourir le cycle eulérien en sautant les sommets déjà visités.

Explications

Dans tout graphe (et à fortiori, dans T), le nombre de sommets de degré *impair* est forcément *pair* (lemme des "poignées de main"). Il existe donc un couplage parfait entre ces sommets là dans le graphe d'origine (et à fortiori, un couplage C de coût minimum). Si on fait l'union $T \cup C$, tous les sommets auront un degré pair dans cette union (soit ils avaient déjà un degré pair dans T, soit leur degré est devenu pair grâce à l'ajout du couplage). Il existe donc forcément un cycle eulérien dans $T \cup C$.

Facteur d'approximation de 1.5

La tournée finale consiste à effectuer un cycle eulérien dans $T \cup C$ en sautant les sommets déjà visités. Grâce à ces raccourcis, chaque arête de $T \cup C$ est empruntée au plus une fois. Le coût total est donc inférieur ou égal au coût total des arêtes de $T \cup C$. Ce coût est lui-même inférieur au total des coûts de T et des coûts de C. Nous avons déjà vu que le coût de T est inférieur à OPT. Il suffit donc pour conclure de montrer que le coût de C est inférieur à $0.5 \cdot OPT$. En fait, on peut même montrer quelque chose de plus fort, à savoir que le coût d'un couplage parfait minimum dans le graphe de départ tout entier est inférieur à la moitié du coût d'une tournée optimale (c'est plus fort, là encore, grâce à l'inégalité triangulaire).

Lemme 12.2. *coût(couplage parfait minimum)* $\leq 0.5 \cdot$ *coût(tournée optimale)*

Preuve. Toute tournée optimale définit un couplage parfait, en prenant une arête sur deux le long de la tournée. Elle en définit un autre : les autres arêtes de la tournée. La somme des coûts de ces deux couplages est exactement égale au coût de la tournée, donc l'un des deux ne dépasse pas la moitié (et à fortiori, un couplage parfait minimum non plus). □

12.3 TSP euclidien

Le version *euclidienne* du TSP est un cas particulier de version *métrique*, dans lequel chaque ville est spécifiée par des coordonnées dans le plan (ou plus généralement dans un espace à d dimensions) et le coût entre deux villes correspond à leur distance euclidienne. Le TSP euclidien est toujours NP-difficile, mais cette fois, il admet une $(1+\epsilon)$-approximation, ce qui signifie qu'on peut s'approcher autant qu'on veut de la solution optimale, en payant un temps de calcul qui reste polynomial en n, mais qui augmente en fonction de l'erreur ϵ souhaitée. Un exemple classique est l'algorithme d'Arora (1996), qui permet cela en temps

$n^{O(1/\epsilon)}$ pour des instances en deux dimensions (l'algorithme se généralise en payant un peu plus en dimensions supérieures).

L'idée générale est de décomposer l'espace récursivement en petites sections résolues séparément, puis recollées par de la programmation dynamique. Cet algorithme n'est pas au programme, mais je vous recommande les notes de cours suivantes si vous souhaitez approfondir : `https://theory.epfl.ch/osven/courses/Approx13/Notes/lecture4-5.pdf`