Dipayan Chakraborty
Florent Foucaud
Malika More
Adrien Wohrer

Université Clermont Auvergne
Méthodes d'optimisation
BUT Info 2A
2023-2024

TD3 : Algorithmes heuristiques

VERSION AVEC SOLUTION

Rappel de cours

Un *algorithme heuristique* est un algorithme qui résout un problème donné, mais qui ne calcule pas forcément la solution optimale. L'idée est d'avoir un temps de calcul réduit, au prix de la perte de qualité des solutions.

Un *algorithme d'approximation* est un algorithme heuristique avec une garantie sur la taille de la solution calculée : la taille est au plus un facteur multiplicatif constant par rapport à l'optimal (pour un problème de minimisation) ou au moins un facteur multiplicatif constant par rapport à l'optimal.

Par exemple, pour un algorithme d'approximation à facteur 2, on aura $|A| \leq 2opt$ (où A est la solution calculée par l'algorithme, et opt est la taille de la solution optimale) pour un problème de minimisation, ou bien $|A| \geq \frac{1}{2}opt$ pour un problème de maximisation.

La *relaxation linéaire* d'un PLNE est tout simplement le PL avec le même objectif et les mêmes contraintes, sauf que les variables ont le droit d'être non-entières. La technique de l'*arrondi* de la relaxation linéaire consiste à trouver une solution pour un PLNE selon les étapes suivantes :

1. Résoudre la relaxation linéaire (PL) du PLNE
2. Prendre toutes la variables qui ont au moins une certaines valeur (par exemple 0.5, cela dépend du problème), et arrondir à l'entier supérieur.
3. Pour les autres variables, on arrondit à l'entier inférieur

Par exemple, si les variables sont entre 0 et 1, cela consiste à mettre à 1 certaines variables et à 0 les autres. Selon le porblème, cela peut donner une solution du PLNE qui n'est pas trop mauvaise. Par exemple, nous avons vu en cours que pour le problème de *couverture par sommets* d'un graphe, cela donne un algorithme d'approximation à facteur 2.

Un *algorithme glouton* est un algorithme qui construit une solution petit à petit, en faisant à chaque étape un choix qui semble le moins mauvais (mais sans regarder le problème complet dans sa globalité). Un tel algorithme est normalement efficace en temps de calcul, mais ne renvoie pas forcément une bonne solution.

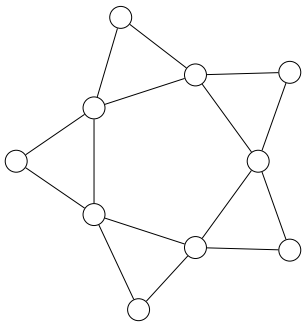
Par exemple, nous avons vu en BUT1 (cours de graphes) des algorithmes gloutons pour construire des arbres couvrants de poids minimum (Kruskal et Prim) ou des flots maximum (Ford-Fulkerson) qui calculent toujours une solution optimale. Mais également des algorithmes gloutons de coloration qui ne calculaient pas de coloration optimale.

Cette année, nous avons vu en cours que l'algorithme glouton pour la couverture par sommets d'un graphe est certes efficace en temps, mais peut être très mauvais en termes de la taille de la solution calculée.

Exercice 1 (Ensembles dominants dans les graphes : arrondi de la relaxation linéaire)**.**

Un *ensemble dominant* dans un graphe $G = (V, E)$ est un ensemble $D \subseteq V$ de sommets de G tel que chaque sommet qui n'est pas dans D a au moins un voisin dans D. On souhaite minimiser la taille de la solution (c'est-à-dire le nombre de sommets sélectionnés).

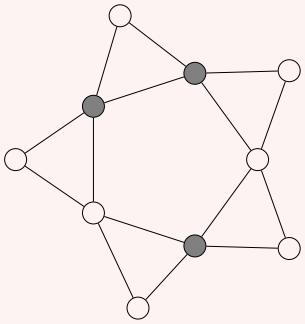
Ce problème modélise par exemple l'installation d'hôpitaux dans un territoire : on souhaite que chaque ville ait soit un hôpital, soit un hôpital dans (au moins) une ville voisine, tout en minimisant le coût total de construction.



1. Essayer de trouver une solution qui vous semble bonne dans le graphe ci-dessus (entourer les sommets choisis). Quelle est sa taille ?
2. Exprimer le problème sous forme de programme linéaire en nombre entiers (PLNE), pour tout graphe $G = (V, E)$. On utilisera une variable x_v pour chaque sommet v. On pourra utiliser la notation $N[v]$ qui symbolise le *voisinage fermé* d'un sommet v, c'est-à-dire v avec tous ses voisins.
3. On résout la *relaxation linéaire* de ce PLNE (le PL avec les mêmes spécifications, mais avec des valeurs non-entières). Trouver (à la main) une solution non-entière du PL pour le graphe de la figure, qui est meilleure que la solution entière.
4. Un solveur nous donne la solution optimale z^* de la relaxation linéaire. On construit à partir de z^*, un ensemble dominant D en sélectionnant tous les sommets v de G tels que dans z^*, $x_v \geq 0.2$ (c'est la technique de l'*arrondi* de la relaxation linéaire). En utilisant votre solution obtenue à la question précédente, quel ensemble D obtenez-vous ?
5. Montrer que si dans le graphe G, tout sommet a au plus 4 voisins (c'est le cas sur le graphe de la figure), alors toute solution D obtenue par l'arrondi de la relaxation linéaire est un ensemble dominant.
6. Montrer que la taille de D est au plus $5 \left(\sum_{x_v \text{ tel que } x_v > 0 \text{ dans } z^*} x_v \right)$.
7. En déduire que la de D est au maximum 5 fois la taille minimum d'un ensemble dominant de G (cette méthode donne donc un algorithme d'approximation à facteur 5).
8. Comment adapter cette technique pour un graphe où chaque sommet a au plu k voisins ?

Solution.

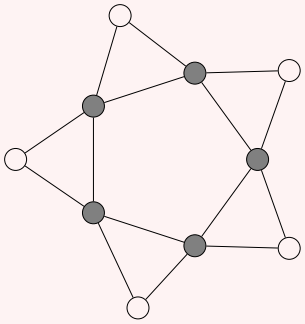
1. On peut prendre trois sommets (voir sommets grisés sur la figure)



2.

$$\begin{array}{lll} \text{minimiser :} & \sum_{v \in V} x_v & \\ \text{tel que :} & \left(\sum_{w \in N[v]} x_w \right) & \geq \quad 1 \quad \forall v \in V \\ & x_v & \leq \quad 1 \quad \forall v \in V \\ & x_v & \geq \quad 0 \quad \forall v \in V \\ & x_v & \in \quad \mathbb{N} \quad \forall v \in V \end{array}$$

3. Une solution possible du PL est de mettre toutes les variables à $1/3$. Cela donne un coût total de $10 \times 1/3 = 3$ (pas mieux que la solution entière). C'est bien une solution car chaque sommet a au moins 3 sommets dans son voisinage fermé.
   Une meilleure solution est de mettre toutes les variables correspondant à des sommets du cycle intérieur à 0.5 (les autres à 0), pour un coût total de $5 \times 0.5 = 2.5$. Les contraintes sont bien respectées car les sommets de degré 2 ont chacun deux voisins, chacun avec une valeur de 0.5, soit une somme sur leur voisinage fermé à 1 ; les sommets de degré 4 ont deux voisins plus eux-même, avec donc une somme sur leur voisinage fermé à $1.5 > 1$.
4. Avec la solution où les sommets du cycle intérieur sont à 0.5, D est égal à l'ensemble de ces cinq sommets. Ce n'est pas optimal, mais ce n'est pas trop mauvais non plus. Si on avait pris la solution où toutes les variables reçoivent $1/3$, on aurait obtenu $D = V$: tous les sommets.



5. Comme pour tout sommet v, on a une contrainte qui dit que la somme des x_w sur les quatre sommets w dans $N[v]$ est au moins 1, par le principe des tiroirs, au moins l'un de ces cinq sommets (appelons-le t) satisfait $x_t \geq 0.2$ dans z^*. Ce sommet sera sélectionné dans D. Donc, le sommet v sera dominé par t dans D. Donc D est bien un ensemble dominant.
6. On a :

$$\begin{aligned} |D| &= \sum_{x_v \text{ tel que } x_v \geq 0.2 \text{ dans } z^*} 1 \\ &\leq 5 \sum_{x_v \text{ tel que } x_v \geq 0.2 \text{ dans } z^*} x_v \end{aligned}$$

   car les valeurs des variables x_v que l'on a sommées sont entre 0.2 et 1, et $5 \times 0.2 = 1$.

7. On obtient donc

$$\begin{aligned} |D| &\leq 5 \sum_{x_v \text{ tel que } x_v \geq 0.2 \text{ dans } z^*} x_v \\ &\leq 5 \sum_{x_v \text{ tel que } x_v > 0 \text{ dans } z^*} x_v \\ &\leq 5opt(PL) \\ &\leq 5opt(PLNE) \\ &= 5opt \end{aligned}$$

8. La valeur d'arrondi passe à $1/(k+1)$. On obtient une approximation de facteur $(k+1)$.

Exercice 2 (Algorithme glouton pour le problème du sac à dos)**.**

On rappelle ce problème étudié au TD2 : un randonneur veut remplir son sac à dos de capacité (poids maximal) W en maximisant l'utilité totale des objets qu'il emporte avec lui. Les objets disponibles sont $O_1, \ldots, O_n$ et chaque objet O_i a une utilité u_i, un poids p_i, et le nombre maximum d'exemplaires autorisés est m_i.

Dans l'exemple du TD2, on avait $W = 22$ et les objets à disposition étaient :

	objet	utilité	poids	maximum autorisé
O_1	barres énergétiques	8	3	10
O_2	pull	12	4	2
O_3	carte	9	3	1
O_4	gourde	15	3	2
O_5	tente	26	13	1

1. Donner un algorithme glouton qui construit pas à pas une solution S_i avec $S_0 = \emptyset$ (l'ensemble vide). A chaque étape $i = 1, 2, 3, ...$ on choisit, parmi tous les objets qui rentrent encore dans le sac et qu'on a encore le droit de choisir (en plus de ceux déjà choisis dans S_{i-1}), celui qui a la plus grande utilité.
2. Tester l'algorithme sur l'exemple. Cela donne-t-il une bonne solution ? Pourquoi ?
3. Améliorer l'algorithme glouton en prenant en compte le ratio u_i/p_i des objets.
4. Calculer le ratio de chaque objet u_i/p_i et le tester sur l'exemple. Cet algorithme sera-t-il toujours optimal ? Pourquoi ?

Solution.

1.
   - $S = \emptyset$
   - Tant qu'il existe un objet disponible qui rentre encore dans le sac :
     - Trouver l'objet O_i disponible, qui rentre dans le sac, et qui maximise l'utilité u_i
     - $S = S \cup \{O_i\}$
   - Renvoyer S

   Note : comment trouver l'objet qui maximise l'utilité ? → On ajoute une boucle de calcul du macimum qui parcourt l'ensemble des objets.

2. L'algorithme va sélectionner : O_5 (tente), O_4 (gourde), O_4 (encore une gourde), et enfin O_3 (carte). Le poids total est $13+3+3+3=22$, pour une utilité totale de $26+15+15+9=65$.

   Ce n'est pas optimal, car on peut prendre 2 gourdes, 1 pull, 1 carte et 3 barres pour 22kg et une utilité de 75. En effet, l'algorithme peut prendre un objet très utile mais aussi très lourd (comme la tente).

3. - $S = \emptyset$
   - Tant qu'il existe un objet disponible qui rentre encore dans le sac :
     - Trouver l'objet O_i disponible, qui rentre dans le sac, et qui maximise le ratio u_i/p_i
     - $S = S \cup \{O_i\}$
   - Renvoyer S

4. Ratio des objets : $u_1/p_1 = 8/3 = 2.67$; $u_2/p_2 = 12/4 = 3$; $u_3/p_3 = 9/3 = 3$; $u_4/p_4 = 15/3 = 5$; $u_5/p_5 = 26/13 = 2$.

   L'algorithme va donc commencer par les objets O_4 (gourdes) qui ont le meilleur ratio, et en prendre le maximum possible (deux). Cela prend 6kg. Ensuite, l'algorithme va choisir les objets de ratio 3 (O_2 et O_3, pulls et carte) et va prendre deux objets O_2 ($4+4=8kg$) et un O_3 (3kg). Finalement, l'algorithme va sélectionner une barre pour 3kg. L'algorithme se termine car il n'y a plus de place, on obtient une solution d'utilité $15+15+12+12+9+8=71$ pour $3+3+4+4+3+3=20kg$.

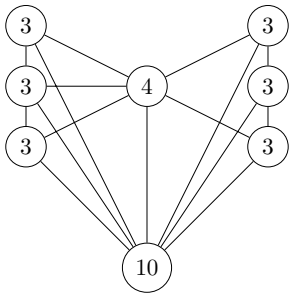
   Donc, l'algorithme n'est toujours pas optimal, car il existe une solution d'utilité 75. Dans cet exemple, on voit qu'il n'utilise pas bien la place, il aurait mieux valu prendre un objet moins utile en plus (barre) à la place d'un objet un peu plus utile (pull) mais plus lourd, et qui empêche de prendre un objet supplémentaire.

Exercice 3 (Algorithme glouton pour l'ensemble dominant)**.**

On reprend le problème de l'ensemble dominant vu à l'exercice 1. Cette fois-ci cependant, les sommets du graphe ont chacun un coût (le coût est noté $c(v)$ pour le sommet v). On souhaite minimiser le coût total de la solution (c'est-à-dire la somme des coûts des sommets sélectionnés).

Dans l'exemple des hôpitaux, les coûts représentent les coûts de construction d'un hôpital dans les différentes villes.

Sur la figure, les nombres sur les sommets représentent les coûts.

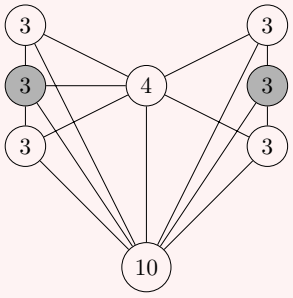


1. Trouver une solution (à la main) aussi bonne que possible pour le graphe de la figure. Quel est son coût ?
2. Modifier le PLNE de l'exercice 1 pour qu'il prenne en compte les coûts.
3. Donner un algorithme glouton qui construit pas à pas une solution, en choisissant à chaque étape un sommet qui couvre le plus de nouveaux sommets.
4. Tester l'algorithme sur l'exemple. Cela donne-t-il une bonne solution ? Pourquoi ?

5. Améliorer l'algorithme glouton en prenant en compte le ratio "nombre de nouvelles villes couvertes / coût" des sommets choisis.
6. Calculer les ratios initiaux et tester sur l'exemple. Cet algorithme sera-t-il toujours optimal ? Pourquoi ?

Solution.

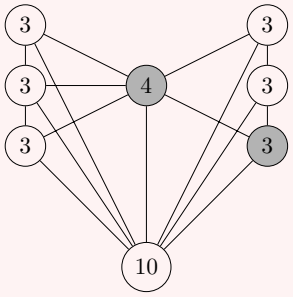
1. Une solution de coût 3+3=6 est de prendre les deux sommets grisés sur la figure ci-dessous :



2.

$$\begin{array}{lll} \text{minimiser :} & \displaystyle\sum_{v\in V} c(v)x_v & \\ \text{tel que :} & \left(\displaystyle\sum_{w\in N[v]} x_w\right) \geq 1 & \forall v \in V \\ & x_v \leq 1 & \forall v \in V \\ & x_v \geq 0 & \forall v \in V \\ & x_v \in \mathbb{N} & \forall v \in V \end{array}$$

3.
   - $S = \emptyset$
   - Tant qu'il existe un sommet non dominé :
     - Trouver le sommet v qui contient le plus de sommets non encore dominés dans son voisinage fermé (y compris lui-même)
     - $S = S \cup \{v\}$
   - Renvoyer S
4. L'algorithme va sélectionner le sommet qui est voisin de tous les autres, pour un coût de 10.
   Cet algorithme ne calcule pas forcément la solution optimale, car l'algorithme va favoriser des sommets qui ont beaucoup de voisins mais qui peuvent être trop coûteux.
5.
   - $S = \emptyset$
   - Tant qu'il existe un sommet non dominé :
     - Trouver le sommet v qui maximise le ratio "sommets non encore dominés dans son voisinage fermé / $c(v)$"
     - $S = S \cup \{v\}$
   - Renvoyer S
6. Ratio initial des sommets : $8/10 = 0.8$, $7/4 = 1.75$, $5/3 \approx 1.66$, $4/3 \approx 1.33$.
   L'algorithme va donc commencer par le sommet de ratio 1.75 (de degré 6). Ce sommet domine presque tous les autres, sauf un. Parmi les sommets restants, ceux qui ne dominent pas le dernier sommet non dominé ont un ratio de 0, les autres, un ratio de 1/3 ou de 1/10. L'algorithme va donc choisir un sommet de coût 3 (ratio 1/3) qui domine le dernier sommet. Cela donne une solution de coût total $4 + 3 = 7$.



Donc, l'algorithme n'est toujours pas optimal, car il existe une solution de coût 6. On voit qu'il faudrait parfois faire des choix non-gloutons pour trouver la solution optimale.

Le PLNE donnerait une solution optimale, mais en temps exponentiel...