Résolution de Problèmes

Projet

BEROUKHIM Keyvan

Exercice 2

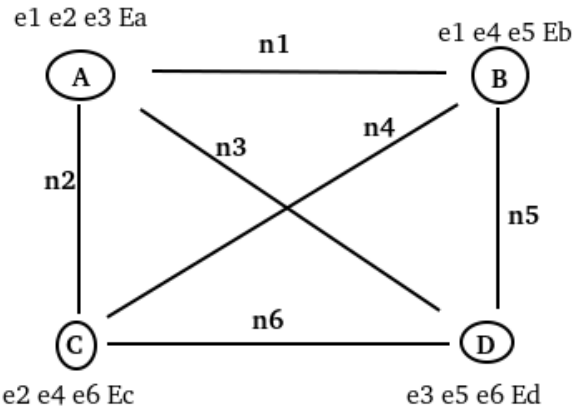
Soit HashCode (HC) le problème de décision associé au problème: "Étant donné un ensemble de vignettes et une borne B, existe-t-il une présentation de score supérieur à B ?"
Pour montrer que le problème est NP difficile, montrons que HC est NP complet.

1) On vérifie en temps polynomial qu'une présentation donnée (le certificat) a un score supérieur à B en sommant la liste des scores.

2) Réduisons le problème NP Difficile du Travelling Salesman Problem (TSP) en une instance de HC.

a) Le problème de décision du TSP est :
"Étant donné G = (V, E) un graphe complet valué, et une borne B, existe-t-il un chemin gamma passant exactement une fois par tous les sommets du graphe (sans revenir au point de départ) tel que le poids de gamma soit inférieur à B ?". On supposera dans cet exercice que la restriction du problème à des valuations entières est aussi NP Difficile. De plus, quitte à ajouter une constante c à toutes les valuations des arcs (et (n - 1) * c à B) , on suppose les valuations strictement positives.

b) Étant donné une instance de TSP {G, B}, on construit l'instance de HC suivante :



- Pour chaque sommet de G, on crée une image.
- Pour chaque arête de G de coût c_i, on crée un ensemble e_i de $n_i = M - c_i$ nouveaux tags distincts, avec M le maximum des valuations des arêtes de G.
- À chaque image, on associe l'union des ensembles des tags associés aux arêtes incidentes au sommet correspondant. De plus, pour chaque image, on associe un ensemble de tag de cardinalité égal au maximum des cardinalités des ensembles précédemment associés (notés Ea Eb Ec et Ed sur le schéma).
- On fixe la valeur de la borne à B’ = (n – 1) * M – B.

Propriété : étant donné deux images et le poids de l’arête correspondante dans le TSP c_i , le score de la transition entre les deux images est $n_i = M – c_i$.

Démonstration :

Soit a et b deux images, le graphe est complet donc a et b partagent une unique arête en commun notée i.

Les tags associés à a et b sont donc de la forme $A = e_1 \cup e_2 \cup \ldots \cup e_{n-1} \cup Ea$ avec ei=e'i.

et $B = e'_1 \cup e'_2 \cup \ldots \cup e'_{n-1} \cup Eb$

On a donc

$A \cap B = e_i$

$A \setminus B = A \setminus e_i$ de cardinalité supérieur à e_i

$B \setminus A = B \setminus e_i$ de cardinalité supérieur à e_i

par définition du score on a donc $s(a, b) = card(e_i) = n_i$.

c) Soit une instance du TSP, montrons l'équivalence des problèmes.

- Supposons que TSP réponde vrai, montrons que HC répond vrai :

Soit gamma un chemin de poids inférieur à B dans TSP. Montrons que la présentation correspondante convient. Soit S le score de la présentation correspondante, on a :

$S = sum(s(a,b))$ (pour chacune des n – 1 transitions (a,b) dans la présentation)

$= sum(M - c_i)$

$= (n - 1) * M - sum(c_i)$

$= (n - 1) * M - c_{gamma}$

$> (n - 1) * M - B$

$= B'$

Donc HC répond vrai.

- De même, supposons que HC réponde vrai, montrons que TSP répond vrai.

Soit gamma un chemin de poids supérieur à B' dans HC. Montrons que le chemin correspondant convient. Soit C le coût du chemin correspondant, on a :

$C = sum(c(a,b))$ (pour chacune des n – 1 transitions (a,b) dans le chemin)

$= sum(M - s_i)$

$= (n - 1) * M - sum(s_i)$

$= (n - 1) * M - S_{présentation}$

$< (n - 1) * M - B'$

$= B$

Donc TSP répond vrai.

Donc HC est un problème NP Difficile.

Exercice 3

La méthode gloutonne plus rapide implémentée consiste à :

- **trier** les vignettes horizontales par ordre croissant de **nombre de tags**.
- trier les vignettes verticales par ordre décroissant de nombre de tags.
- commencer par placer les vignettes horizontales puis les vignettes verticales comme dans la méthode proposée. Cependant, on ne cherche pas la meilleure vignette parmi toutes les vignettes restantes mais seulement dans une **fenêtre de taille réduite**. La complexité est alors O(n*w) avec n le nombre de vignettes et w la taille de la fenêtre (au lieu de O(n*n) pour la méthode précédente). Afin de pouvoir enlever les vignettes en O(1) on stocke les vignettes dans une **liste chaînée**.

Exercice 4

On implémente une technique de **recuit simulé** : pour un nombre d'itération donné en paramètre, on tire au hasard deux vignettes que l'on permute si la permutation augmente le score ou avec une probabilité décroissante en la perte et le numéro d'itération sinon.
Notons que ces améliorations n'augmentent le score des solutions obtenues par l'algorithme glouton de **quelques pourcents seulement**.

Exercices 5 & 6

On modélise ce problème en **PLNE** de la même manière que le TSP. Cependant, on maximise la somme des poids des arêtes au lieu de les minimiser. De plus, afin de chercher le chemin le plus grand plutôt que circuit le plus grand, on rajoute un **sommet fictif** reliés à tous les sommets et de poids entrants et sortants nuls. Une fois le circuit de poids maximal trouvé, on enlève le sommet fictif, ce qui nous permet d'obtenir le chemin de poids maximal.
Cette méthode est bien trop lente pour résoudre des grandes instances (comme lovely_landscapes).

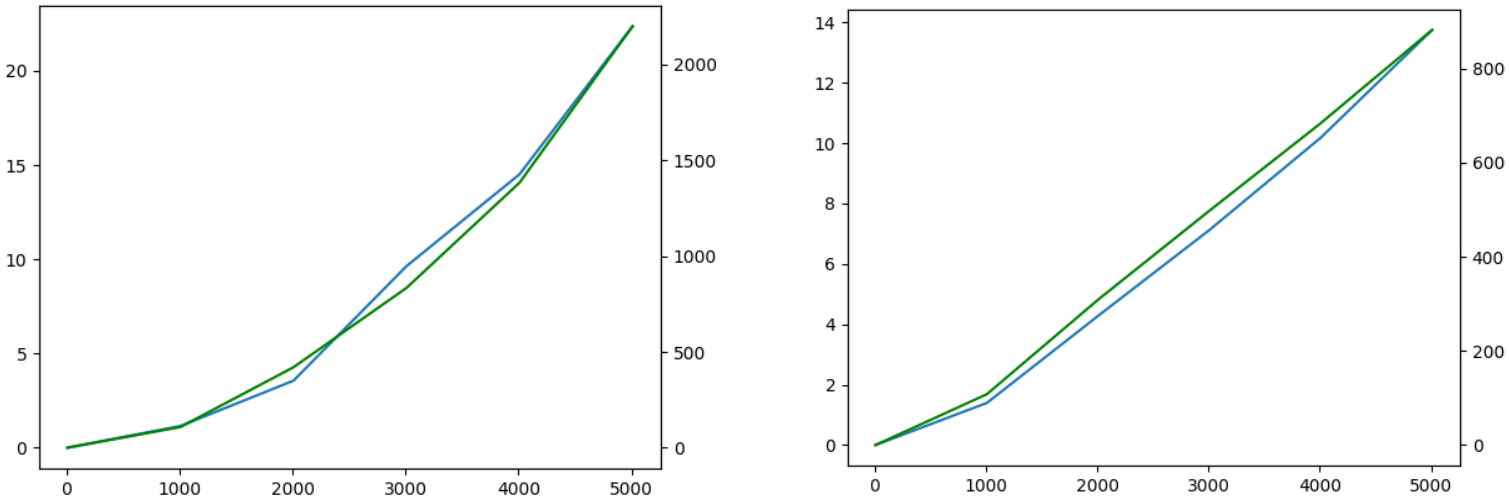
La méthode **d'arrondi** est plus rapide mais cela ne suffit pas pour résoudre des grandes instances.

Une **solution moins efficace mais plus rapide** implémentée est de couper l'instance en de petites sous-instances, de les résoudre indépendamment, puis de concaténer les résultats.

Exercice 7

Les graphiques suivants représentent en bleu le temps d'exécution en seconde. La courbe verte, si présente, représente le score obtenu sur l'instance « lovely landscapes » et l'échelle correspondante est inscrite à droite.

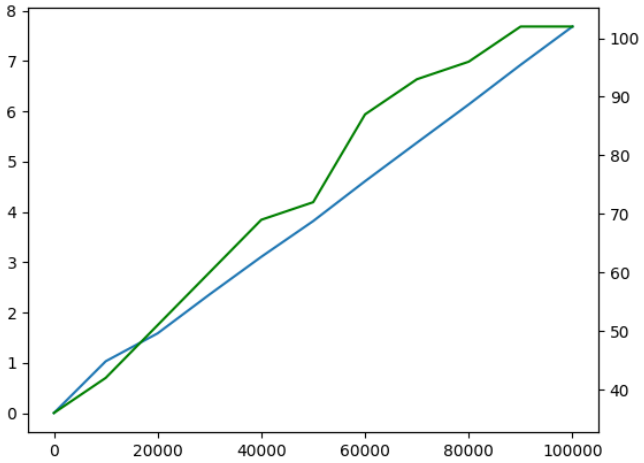
1) Algorithmes gloutons



Scores obtenus par l'algorithme glouton classique à gauche et glouton fenêtré à droite
en fonction de la taille de l'instance

La complexité et le score évoluent de la même manière: l'algorithme **glouton** a une complexité et un score **quadratiques** en la taille de l'instance. La version **fenêtrée** a une complexité et un score quadratiques en la taille de la fenêtre mais **linéaires** en la taille de l'instance.

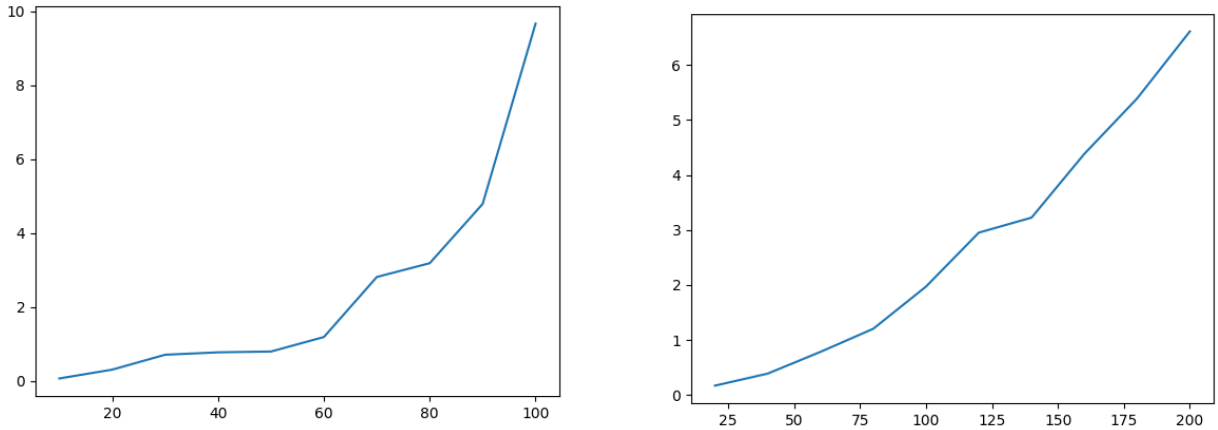
2) Recherche locale



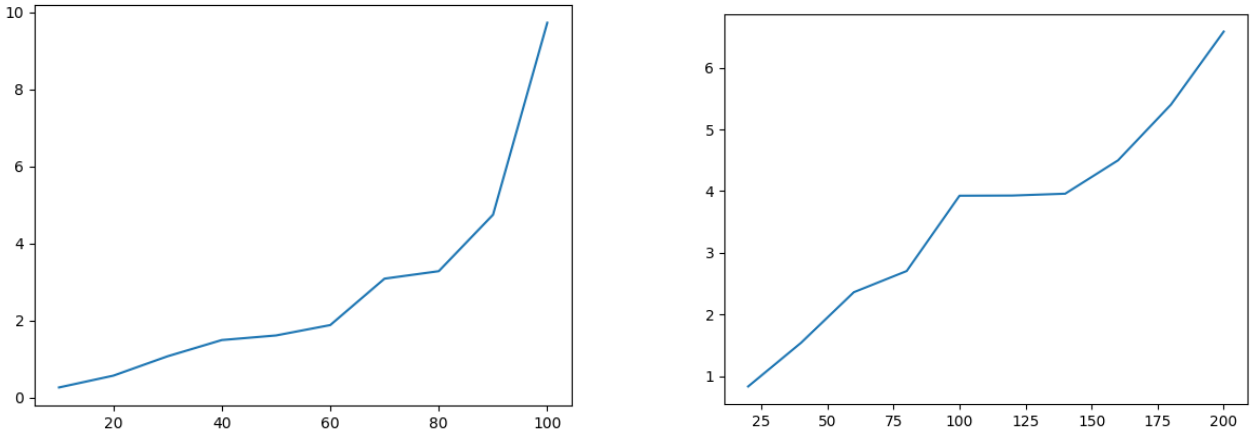
Score obtenu par recuit simulé en fonction du nombre d'itérations

Le simulated annealing a une complexité linéaire en le nombre d'itérations, il semble être utile quelle que soit la solution initiale (du moment qu'elle n'est pas trop bonne)

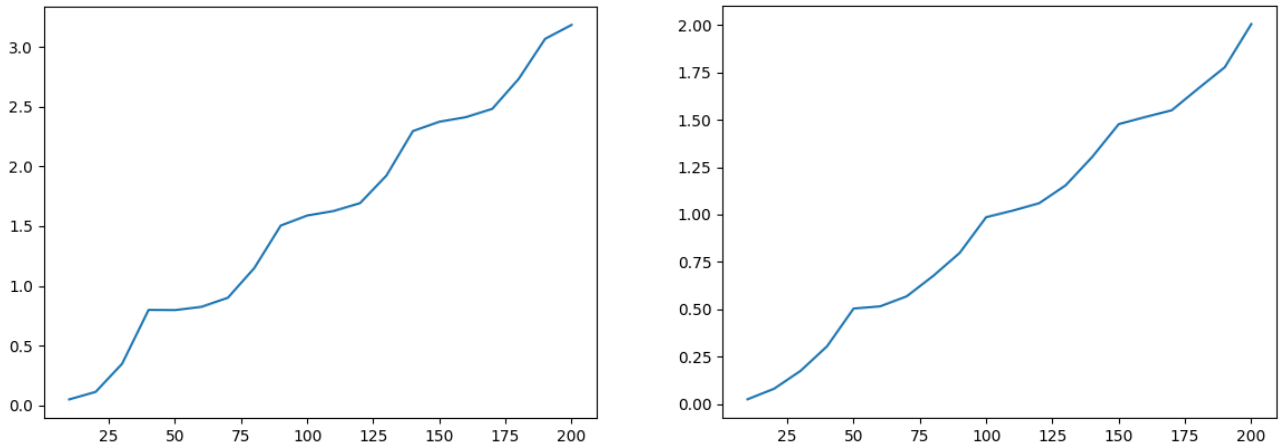
c) Programmation linéaire



PLNE normal à gauche et PL relâché à droite
appliqués à des instances de tailles croissantes

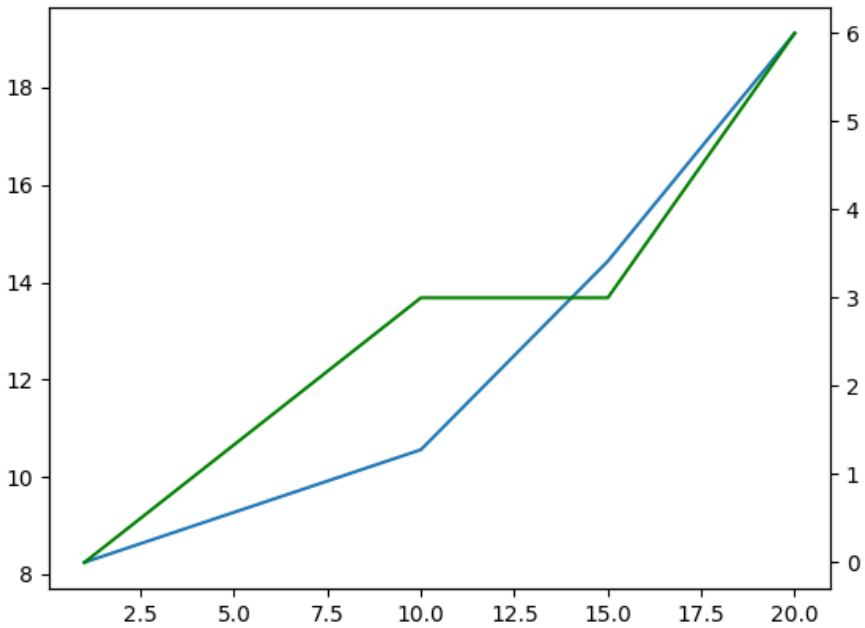


PLNE de taille 100 à gauche, PL relâché de taille 200 à droite
résolus par sous-instances de tailles croissantes



PLNE classique à gauche, PL relâché à droite de tailles croissantes
résolus par des sous-instances de taille fixée à 50

Les PL atteignent leurs limites vers des instances d'une centaine d'éléments. Le PLNE classique a une allure exponentielle et le PL relâché une allure quadratique.
En coupant le problème en sous-instances, le temps évolue de la même manière que précédemment en la taille des sous-instances et linéairement en le nombre de sous-instances. Pour une taille de sous-instance fixée, le temps n'augmente que linéairement en la taille de l'instance globale.



PL relâché sur une instance de taille 5000
résolu par des sous-instances de taille croissante

Malgré la relaxation linéaire, le programme linéaire **ne passe pas à l'échelle**: en 20 secondes on atteint un score de 6 alors que dans la même situtation l'algorithme glouton obtient un score de 2000.

Exercices 8

Avec notre version d'algorithme glouton suivi de recuit simulé, on obtient un **score total de 1 018 893**.

1 - taille instance 4
glouton temps 0.0 score 1
descente temps 4.5 score 2

2 - taille instance 80000 (que des horizontales)
glouton temps 23.7 score 1794
descente temps 8.6 score 1866

3 - taille instance 1000
glouton temps 0.3 score 1403
descente temps 10.1 score 2381

4 - taille instance 90000
glouton temps 178.9 score 431835 (early cuts: 55863)
descente temps 51.8 score 433032 (+ 0.3 %)

5 - taille instance 80000 (que des verticales)
glouton temps 796.1 score 478933 (early cuts: 2)
descente temps 254.7 score 581612 (+ 21.4%)

Pour utiliser le code, exécuter la commande :
python3 main.py