EILCO INFO3 — Mineure IA

Résolution de problèmes – TP1

Novembre 2024

L'objectif de ce TP est de concevoir, développer et expérimenter des méthodes d'optimisation heuristiques pour le problème TTP *travelling thief problem*.

Le TP sera réalisé en binôme. Le code à écrire en Python, C ou C++, doit être facile à exécuter : compilation et/ou exécution immédiate(s) (utiliser des librairies classiques). Vous rendrez une archive nommée `nom1-nom2.zip`, contenant les sources ainsi qu'un mini-rapport.

1 TTP

Le TTP est une combinaison du problème du voyageur de commerce (*travelling salesman problem* - TSP) et le problème du sac à dos (*knapsack problem* - KSP). Ces deux problèmes sont introduits dans les sections suivantes.

1.1 TSP

Étant donné un ensemble de villes, le TSP consiste à trouver le circuit "optimal" qui relie toutes les villes, sans passer plus d'une fois par la même. L'optimalité se rapporte généralement à une distance parcourue, ou un coût, à minimiser. Le TSP est connu comme étant un problème NP-difficile : on ne connaît pas d'algorithme permettant de trouver une solution exacte en un temps polynomial. Néanmoins, de nombreuses applications réelles dans différents domaines (transport, ordonnancement, logistique) se rapportent à un problème de TSP, ou se doivent de résoudre le TSP comme sous-problème. Plus formellement, le TSP peut être défini par un graphe complet $G = (V, E)$, où $V = \{v_1, v_2, ..., v_n\}$ est un ensemble de noeuds (les villes) et $E = \{[v_i, v_j], v_i, v_j \in V\}$ est un ensemble d'arcs. À chaque arc $[v_i, v_j] \in E$ est affecté un coût positif $c(i, j)$, donné par une matrice de coûts. Le but est de trouver la permutation cyclique simple π de taille n qui minimise la fonction suivante (la partie droite correspond au retour à la ville de départ) :

$$f(\pi) = \sum_{i=1}^{n-1} c\Big(\pi(i), \pi(i+1)\Big) \; + \; c\Big(\pi(n), \pi(1)\Big)$$

1.2 KSP

Pour le KSP, un sac à dos doit être rempli d'objets tout en respectant une contrainte de poids maximum. Chaque objet j a une valeur $b_j \geq 0$ et un poids $w_j \geq 0$ où $j \in \{1, .., m\}$. Le vecteur de décision binaire $z = (z_1, .., z_m)$ définit si chaque objet est sélectionné ou non. L'espace de recherche de ce problème contient 2^n combinaisons et le but est de maximiser le profit $g(z)$:

$$g(z) = \sum_{j=1}^{m} z_j b_j \quad \text{, tel que } \; \sum_{j=1}^{m} z_j w_j \leq Q$$

1.3 TTP

Le TTP combine les deux sous-problèmes définis ci-dessus. Le voleur peut récupérer des objets dans chaque ville qu'il visite. Les objets sont stockés dans son sac à dos. De manière plus détaillée, chaque ville π_i fournit un ou plusieurs objets, qui peuvent être récupérés (ou non) par le voleur. Voici l'interaction entre les sous-problèmes : la vitesse du voleur pendant son déplacement dépend du poids w du sac à dos qu'il transporte (le voleur ralenti donc au fur et à mesure que son sac s'alourdi). Cette vitesse vaut :

$$v(w) = v_{max} - \frac{w}{Q} * (v_{max} - v_{min})$$

Ainsi, le temps de parcours du voleur vaut :

$$d(\pi) = \sum_{i=1}^{n-1} \frac{c\Big(\pi(i), \pi(i+1)\Big)}{v(w(i, \pi, z))} + \frac{c\Big(\pi(n), \pi(1)\Big)}{v(w(n, \pi, z))}$$

sachant que $w(i, \pi, z)$ correspond au poids du sac du voleur après avoir visité la ième ville du parcours π (z étant un vecteur binaire indiquant pour chaque objet si le voleur l'emporte ou non).
Enfin, la fonction objectif vise à maximiser la valeur des objets emportés, tout en cherchant à limiter la durée du parcours du voleur (R, le *renting ratio*, est donné dans l'instance) :

$$f(z) = (\sum_{i=1}^{n} z_j b_j) - R * d(\pi)$$

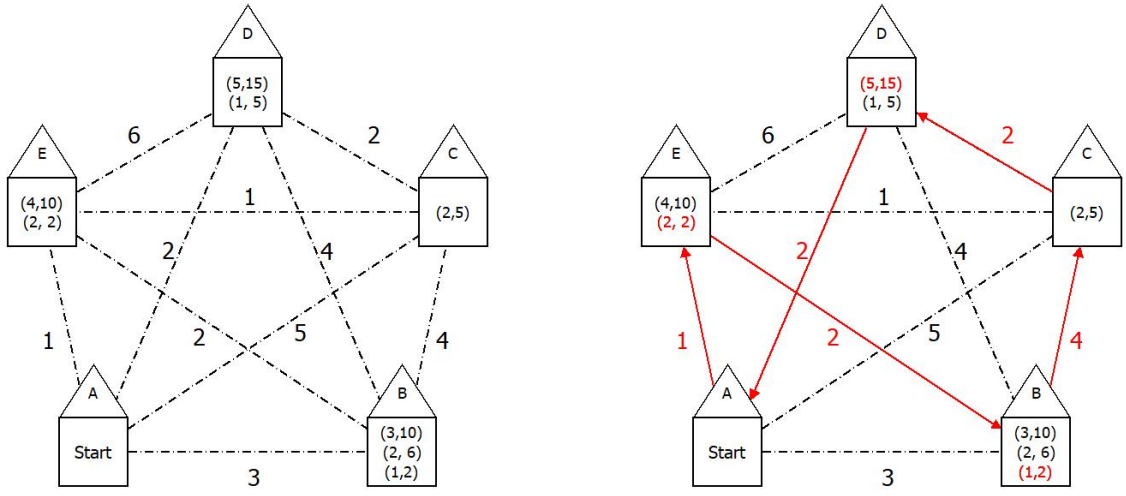

Figure 1: A gauche : Une instance TTP, avec les distances sur les arêtes et le couple (valeur, poids) des objets sur les sommets. On considère ici Q = 20kgs (charge maximale), On suppose que $R = 1$. $V_{max} = 30$km/h et $V_{min} = 10$km/h.
A droite : Une solution TTP (représentation : parcours "AEBCD", objets "01-001-0-10"). On a $v(w) = v_{max} - \frac{w}{Q} * (v_{max} - v_{min}) = 30 - \frac{w}{20} * (30 - 10) = 30 - w$. La durée de parcours résultant de la solution vaut donc : $1/30 + 2/28 + 4/27 + 2/27 + 2/12 \approx 0.49$. Le poids total est de 19kgs (solution valide), le gain vaut 8. On a donc : $f(z) = 8 - 1 * 0.49 \approx 7.51$.

2 Instances

Nous allons utiliser 12 instances pour nos expérimentations : 6 sont construites à partir d'un TSP à 52 sommets, les 6 autres à partir d'un TSP à 280 sommets. Le fichier `best-known.csv` contient les meilleures valeurs connues pour ces instances.
Attention, les valeurs données pour les villes sont des coordonnées, il faut donc calculer les distances entre chaque paire de villes en calculant leur distance (Euclidienne).

3 Fonctionnalités à écrire

Question 3.1 : Écrire un parseur permettant de lire le fichier d'instance passé en paramètre et de construire la matrice de distances (valeurs arrondies par excès).

Question 3.2 : Développer une fonction d'évaluation, capable de calculer la valeur d'une solution (permutation + chaîne de bits) passée en paramètre. La fonction retourne 0 si la solution n'est pas réalisable.

Question 3.3 : Proposez une procédure de génération aléatoire d'une solution (de préférence valide).

Question 3.4 : Proposez une procédure de génération d'un voisin d'une permutation : le `swap` (échange la position de deux villes) et le `two-opt` (supprime deux arrêtes du parcours et reconnecter les deux sous-tours obtenus) d'une solution (de préférence valide).
De même pour une chaîne de bits : `flip` modifie la valeur d'un bit, `replace` échange un 1 avec un 0 de la chaîne de bits.

Question 3.5 : Développer deux types de stratégies de mouvement : meilleur voisin améliorant, ou premier voisin améliorant. Comparer les résultats obtenus par votre algorithme de hill-climbing en fonction du choix de la stratégie du mouvement.
De même, tester les deux variantes avec différents voisinages : `swap+flip`, `two-opt+flip`, `swap+flip+replace`, `two-opt+flip+replace`.

Question 3.6 : Proposer une stratégie afin de poursuivre le processus de recherche même lorsque votre algorithme de hill-climbing atteint un optimum local (vous êtes entièrement libres sur le choix de votre approche).

Question 3.7 : Fournir un mini-rapport. Celui-ci doit contenir quelques lignes indiquant comment compiler et exécuter, ce qui a été réussi, et une description de la variante proposée. Vous donnerez les résultats obtenus lors de vos expériences (par exemple en limitant à 10 millions d'évaluations, et moyennant sur 10 exécutions par méthode; comparaison avec le "best known" - analyse des résultats en quelques lignes).