INF6102 - Métaheuristiques appliquées au génie informatique

Hiver 2025

Devoir 2
Gestion de points de distribution

Auriane Peter–Hemon - 2310513
Félix Lamarche - 2077446

30 Mars 2025

Table des matières

1 Introduction

Soit $G = (V, E)$ un graphe connecté, non-orienté et sans cycles. Soit $P \subseteq V$ un ensemble de points ramenant un certain revenu. Un de ces noeuds est le centre de distribution dont les camions doivent partir. On cherche le traçage $T \subseteq E$ tel que l'on maximise le revenu donné par les noeuds visités. Ce traçage doit respecter deux contraintes : une contrainte de budget maximal et une contrainte de distance maximale au centre de distribution. On désigne par "noeud-revenu" les noeuds appartenant à P.

2 Algorithme glouton

Dans un premier temps, on génère une solution initiale de manière gloutonne. On commence avec une solution ne contenant que le noeud racine (le centre de distribution).
Un noeud est valide si il n'est pas déjà dans la solution et qu'il peut y être ajouté sans violer les contraintes de budget et de distance.
Pour chaque noeud-revenu valide, on calcule le plus court chemin (en terme de coût) entre ce noeud-revenu et les noeuds dans la solution courante. On note le coût d'ajouter ce noeud à la solution ainsi que le revenu engendré par cet ajout.
On ajoute ensuite le noeud-revenu ayant le ratio $\frac{\text{revenu engendré}}{\text{coût d'ajout}}$ le plus élevé.
On recommence jusqu'à ce qu'il ne soit plus possible d'ajouter des noeuds.

Le pseudo-code est le suivant :

Algorithm 1: solve_profit_nodes_greedy()

```
1: s = 1, T = ∅
2: best_ratio = 1
3: while best_ratio ≠ 0 do
4:    best_ratio = 0, best_node = ∅, best_path = ∅
5:    for all profit-node ∉ s do
6:       if profit-node is valid then
7:          path, cost, dist = get_path_to_solution(profit-node)
8:          if cost / dist > best_ratio then
9:             best_node = profit-node
10:            best_ratio = cost / node
11:            best_path = path
12:         end if
13:      end if
14:   end for
15:   if best_ratio ≠ 0 then
16:      s ← s ∪ {best_node}
17:      T ← T ∪ {best_path}
18:   end if
19: end while
20: Return s, T
```

3 Algorithmes de recherche locale

Pour améliorer les solutions obtenues par l'algorithme glouton, on développe une méthode de recherche locale. Pour cela, on se base sur la méthode développée par Costa et al. [1]

Le principe est le suivant : les solutions générées par l'heuristique gloutonne ont une forme d'arbre (puisqu'on ajoute des chemins allant de la racine à un noeud-revenu). Pour chacune des feuilles de cet arbre, on bloque l'arête connectant la feuille à la solution, puis on relance l'heuristique gloutonne en interdisant de sélectionner cette arête. En pratique, on affecte pour cela un coût infini à cette arête. On sélectionne le meilleur voisin améliorant trouvé.

La modélisation est la suivante :
- **Solution initiale :** solution obtenue grâce à l'heuristique gloutonne décrite précedemment
- **Voisinage :** Les voisins d'une solution sont les solutions ne contenant pas une des arêtes menant à une feuille
- **Évaluation** : la valeur de chaque voisin correspond au revenu total récupéré dans la solution
- **Sélection** : on sélectionne le meilleur voisin améliorant
- **Critère d'arrêt** : on bloque une seule fois chaque arête menant à une feuille et on prend la meilleure solution parmi ce nombre fini d'itérations

Toutes les contraintes sont dures : on ne considère que des solutions valides respectant le budget B et la distance maximale H.

Le pseudo-code est le suivant :

Algorithm 2: solve_local_search()

```
s, T = solve_profit_nodes_greedy()
best_solution = (s, T)
for all edges(i,j) incident to a leaf vertex do
    previous_cost = c_ij
    c_ij = ∞
    s, T = solve_profit_nodes_greedy() (with modified cost)
    if r(s, T) > r(best_solution) then
        best_solution = (s, T)
    end if
    c_ij = previous_cost (restore original cost)
end for
Return best_solution
```

Analyse du voisinage : Le voisinage contient des résolutions de l'instance avec l'algorithme greedy tel qu'une des arêtes de la meilleure solution a été retirée du problème en lui donnant un coût infini. Ce voisinage a une taille de M, donc il y a une solution voisine pour chacune des arêtes présente dans la meilleure solution. Ce voisinage n'est pas connecté, puisqu'on ne peut que retirer une seule arête pour la recherche Greedy, il pourrait y avoir un mouvement optimal nécessitant que deux arêtes soient retirées dans un cas où un noeud a trois arêtes entrantes et que la solution optimale comprend l'arête entrante que l'algorithme greedy croit être le pire choix. En effet, l'algorithme greedy implique que le meilleur choix à chaque itération est choisie et retirer une seule arête peut ne pas suffir pour que l'algorithme greedy générant le voisinage puisse faire les choix optimaux. De plus, seule les arêtes immédiatement connectées à un noeud de profit sont altérées, ce qui implique que plusieurs chemins ne seront pas générés par l'algorithme greedy, générant des chemins optimaux que pour atteindre un seul noeud de profit à la fois, et non un chemin pouvant être plus optimal pour atteindre plusieurs noeuds si pour générer ce chemin, une arête n'étant pas directement connecté à un noeud de profit doit être bloqué. Sélectionner un voisin est également coûteux, puisque chaque voisin implique une nouvelle recherche gloutonne, la fonction a une complexité asymptotique de $O(N * P^2(M + NlogN))$ (expliqué en plus de détails dans la section sur la complexité asymptotique).

On obtient les résultats moyens suivants :

Méthode utilisée	**InstanceA**	**InstanceB**	**InstanceC**	**InstanceD**	**InstanceE**	**InstanceF**
Méthode gloutonne	681	1204	2735	3758	344	648
Recherche locale	694	1204	2778	3822	345	648

Ainsi que les temps d'exécution moyens :

On constate que la recherche locale permet d'obtenir de meilleurs résultats que la méthode gloutonne. Cependant, elle est longue à exécuter sur les grandes instances car elle relance plusieurs fois entièrement

Méthode utilisée	InstanceA	InstanceB	InstanceC	InstanceD	InstanceE	InstanceF
Méthode gloutonne	0.04	0.11	1.10	3.52	14.44	54.34
Recherche locale 1	0.20	0.74	17.43	59.11	277.82	128.34

la méthode gloutonne.

4 Algorithme de recuit simulé

Un algorithme de résolution de recherche locale par recuit simulé a été ajouté. Comme la méthode de recherche locale développée précedemment est lente, on développe une seconde méthode que l'on utilise dans l'algorithme de recuit simulé.

La modélisation est la suivante :
- **Solution initiale :** solution obtenue grâce à l'heuristique gloutonne décrite précédemment
- **Voisinage :** Le voisinage est composé des solutions ayant un noeud-revenu de profit ajouté ou retiré par rapport à la solution courante. Si un noeud est retiré, les noeuds et arêtes liant ce noeud-revenu à un autre noeud-revenu de la solution sont retirés également. De nouveaux noeuds de profits sont ajoutés en trouvant le chemin le moins coûteux reliant ce noeud de profit à notre solution courante.
- **Évaluation** : la somme des revenus des noeuds présents dans la solution.
- **Sélection** : on sélectionne aléatoirement des voisins et le premier voisin améliorant ou qui est accepté aléatoirement selon la probabilité de sélection du recuit simulé est sélectionné. On retire uniquement les noeuds-revenus qui sont des feuilles (car sinon, ils sont reliés à au moins un autre noeud-revenu hors dépôt)
- **Critère d'arrêt** : on s'arrête lorsque l'on a atteint un certain nombre d'itérations sans amélioration ou un certain nombre d'itérations sans que la oslution courante ait été modifiée ou que le temps de recherche est écoulé

Analyse du voisinage : Le voisinage a une taille de P voisins, il y a autant de voisins que de noeuds de profits. Le voisinage respecte les deux contraintes dures : la solution ne dépasse jamais son budget alloué B, et aucun noeud ayant une distance au dépôt de plus que la distance maximale permise H n'est ajouté. Il n'y a aucune contrainte molle.
Le voisinage n'est pas connecté, parce que le voisinage consiste aux solutions adjacents ayant un noeud de profit ajouté, et son chemin le moins coûteux pouvant le relier à notre solution courante, et retirer un noeud de profit feuille et son chemin relié. Ce qui implique que des chemins optimales pour atteindre plusieurs noeuds de profits simultanément ne seront jamais générés. Par exemple, il peut exister deux noeuds de profits ayant un chemin commun moins coûteux pour les atteindre communément, mais qui serait plus coûteux pour en atteindre une seule. Ainsi, le chemin moins coûteux sera toujours généré pour atteindre une des deux noeuds de profits en premier, et le chemin commun optimale ne pourra jamais être généré, puisqu'il implique de produire un chemin sous-optimal pour ensuite devenir optimal. Notre voisinage, en pratique, obtient de bons résultats, mais peut rester coincer dans des minimums locaux, puisqu'on peut choisir d'ajouter et de retirer des noeuds de profits de façon sous optimalle, mais pas les chemins les reliant à notre solution courante, qui eux seront toujours le chemin ayant le plus petit coût pour ajouter le dit noeud de profit.
Pour trouver le chemin reliant le noeud de profit à notre solution, l'algorithme de Djikstra est utilisé pour trouver le chemin avec le plus petit coût respectant nos contraintes dures. La génération du voisinage a une complexité asymptotique de $O(P * (M + NlogN))$

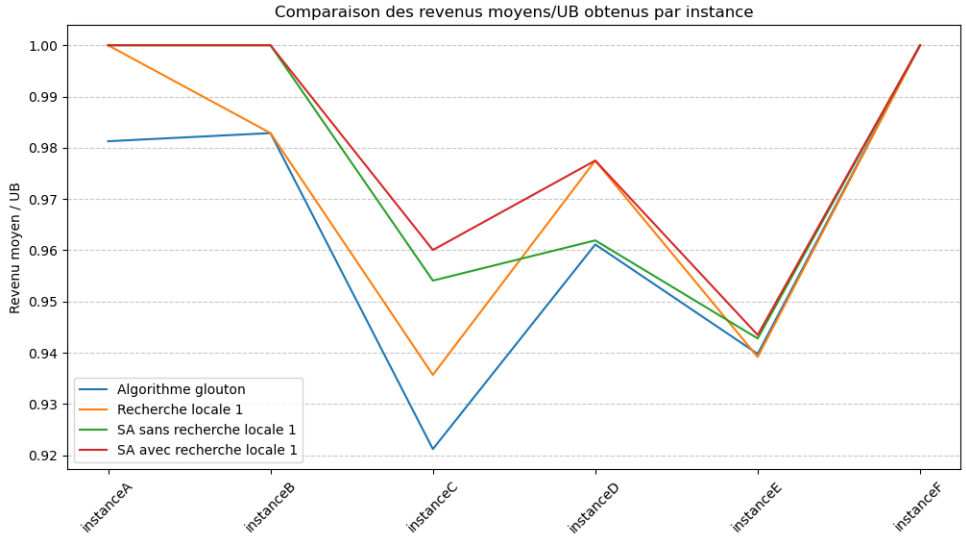
Améliorations supplémentaires : Plusieurs mécanismes supplémentaires ont été ajouté pour tenter d'améliorer la recherche. Pour éviter des mouvements répétitifs, une mémoire des noeuds de profits récemment ajoutés ou retirés de la solution sont sauvegardés pour éviter des cycles (liste tabou). Tant qu'il y a du temps de recherche, la recherche peut redémarrer avec la meilleure solution courante trouvée pour le moment, et pour diversifier la recherche, un nombre aléatoire de noeuds de profits et les chemins les liant au reste de la solution sont retirés de cette solution pour obtenir une solution initiale plus diversifiée.

Paramètres : Les paramètres choisies ont été choisie de façon expérimentale. De très hautes valeurs ont

été choisies pour avoir une bonne diversification au départ des solutions. Le facteur de refroidissement est de 0.99, et la température se base sur le nombre de noeuds dans l'instance pour une température de départ de $250N$. Cette haute température permet de bouger aléatoirement au départ pour ensuite lentement décroitre, puisque retirer un noeud de profit est toujours une dégradation de notre solution courante.

5 Résultats

Pour que l'on puisse comparer les instances de manière lisible, on affiche $\frac{\text{revenue moyen obtenu}}{UB}$ pour chaque instance et chaque méthode.



"SA avec recherche locale 1" désigne l'algorithme de recuit simulé lancé avec comme solution initiale la solution obtenue avec la recherche locale 1. "SA sans recherche locale 1" désigne l'algorithme de recuit simulé lancé avec comme solution initiale la solution obtenue avec l'algorithme glouton.

On peut voir que l'algorithme de recuit simulé donne toujours de meilleurs résultats que la méthode gloutonne. Les meilleurs résultats sont obtenus pour la méthode de recuit simulé initialisé avec la méthode de recherche locale 1. On parvient donc à s'extraire de maximum locaux dans lesquels on était coincés avec la méthode gloutonne.

L'algorithme de recuit simulé initialisé avec la méthode de recherche locale 1 nous permet systématiquement d'obtenir le meilleur résultat obtenu toutes méthodes confondues. Ces résultats sont les suivants :

	InstanceA	**InstanceB**	**InstanceC**	**InstanceD**	**InstanceE**	**InstanceF**
Meilleur revenu obtenu	694	1225	2853	3822	348	648
Revenu moyen	694	1225	2850,37	3822	346.25	648
Ecart type	0.00	0.00	2.02	0.00	2.04	0.00
Tps d'exécution moyen (s)	0.44	1.27	21.37	67.40	309.20	189.17

Pour les instances C, D et E, on n'obtient pas le meilleur résultat possible. On reste donc coincé dans des minima locaux.

6 Analyse de complexité

Recherche du chemin le plus court pour rajouter un noeud de profit :
`get_additional_path_to_node()` : $O(M + NlogN)$.
Nous utilisons l'algorithme de Djikstra pour trouver le chemin le moins coûteux entre le noeud de profit et n'importe quel noeud de notre solution courante.

Recherche du chemin d'un noeud feuille pouvant être retirer :
`get_path_to_remove_node(node)` : $O(N)$.
L'algorithme ne peut retirer que des noeuds feuilles. Chaque noeud intermédiaire étant relié au noeud cible n'étant pas un noeud de profit se fait retirer s'il n'a qu'un seul voisin.

Recherche Greedy :
`solve_proft_nodes_greedy` : $O(P^2(M + NlogN))$.
La recherche greedy ajoute toujours le meilleur choix courant de noeud de profit à sa solution. Pour cela, il doit itérer sur chacun des noeuds de profits pas présentement dans la solution et trouver leur chemin le plus court avec la fonction `get_additional_path_to_node()`, qui a une complexité de $O(M + NlogN)$. Ces chemins peuvent changer à chaque fois que la solution courante change, ils doivent donc être recalculés à chaque nouveau noeud de profit ajouté.

Recherche locale méthode 1 :
`solve_profit_nodes_local_search()` : $O(NP^2(M + NlogN)$.
Cette recherche fait autant de recherches greedy que de noeuds feuilles, qui peut aller jusqu'à N, elle a donc une haute complexité asymptotique.

Recherche de Simulated Annealing :
`solve_profit_nodes_simulated_annealing`$temp, cooling_rate, tabu_length$: $O(P(M + NlogN))$.
À chaque itération, on retire ou ajoute un noeud de profit de la solution courante si possible en générant le chemin optimal a retiré ou ajouté pour chacun des noeuds de profit jusqu'à la sélection du premier voisin améliorant ou du voisin étant sélectionné selon la probabilité de sélection. Le critère d'arrêt étant un nombre d'itérations sans changement ou amélioration, l'algorithme peut avoir un temps d'exécution assez long bien que sa complexité asymptotique soit faible.

Références

[1] Alysson M. Costa, Jean-François Cordeau et Gilbert Laporte. "Fast heuristics for the Steiner tree problem with revenues, budget and hop constraints". en. In : *European Journal of Operational Research* 190.1 (oct. 2008), p. 68-78. ISSN : 03772217. DOI : 10.1016/j.ejor.2007.06.012. URL : https://linkinghub.elsevier.com/retrieve/pii/S0377221707005516 (visité le 29/03/2025).