AI09
Problème des tournées sélectives

Jana Rafei - Yousra Hassan

A23
Aziz Moukrim
Université de Technologie de Compiègne

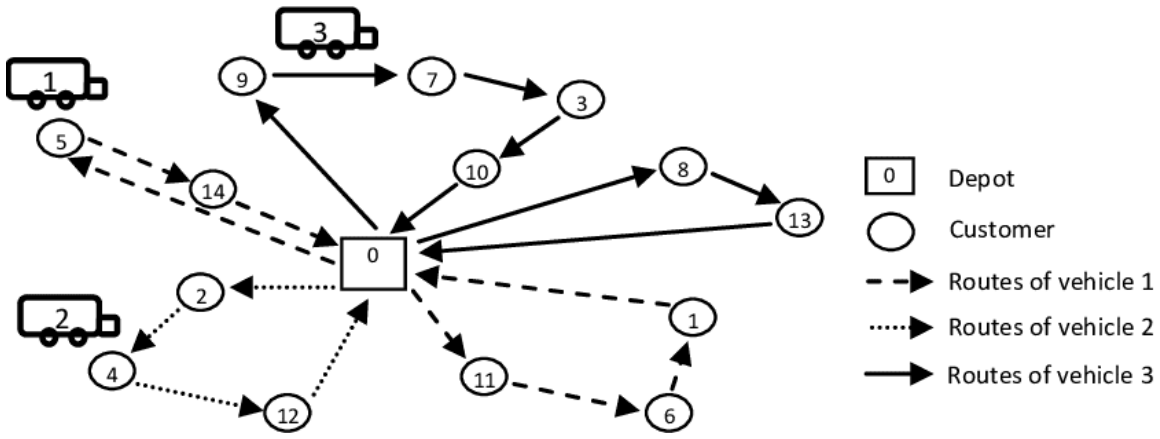
Sommaire

Introduction

Ce projet repose sur le problème des tournées sélectives, décrit dans la littérature sous l'intitulé du Team Orienteering Problem (TOP). Il a pour objectif de déterminer et d'organiser un ensemble optimal de tournées afin de maximiser le profit total collecté des clients desservis. Pour cela, chaque véhicule d'une flotte donnée doit effectuer une tournée entre un point de départ et un point d'arrivée pour récupérer le maximum de profits associés à des clients.

Pour trouver une solution satisfaisante à ce problème, nous allons tenter de formaliser le problème grâce à la formulation linéaire en nombres entiers en exprimant et respectant chaque contrainte impliquée puis présenter les différentes approches de résolution jusqu'à implémenter plusieurs solutions satisfaisantes.

Ce projet a été enrichi par les travaux scientifiques antérieurs, les ressources utilisées sont consignées dans la bibliographie.



Lien

FIGURE 1.1 – Illustration du problème des tournées sous forme de graphe

Formulation linéaire

2.1 Vocabulaire - Données d'entrée

— Ensemble de clients n : c_i, avec $i \in [1,\ n]$
— Ensemble de véhicules m : v_k, avec $k \in [1,\ m]$
— Point de départ : d
— Point d'arrivée : a
— Profit : P_i, associé à un c_i
— Temps de trajet : C_{ij}, associé à un (i, j)
— Temps de parcours maximal : L, commun à tous les véhicules

2.2 Modélisation sous forme de graphe

Ce problème est modélisé par un graphe complet $G = (V, E)$ tel que :

— Sommets : $V = \{1, .., n\}\ U\ \{d, a\}$. Représente tous les points d'arrêts : clients, point de départ et point d'arrivée
— Arêtes : $E = \{\ (i,\ j)\ |\ i, j\ \ \in V\}$. Représente la complétude du graphe car tous les sommets sont liés.

De plus, nous introduisons des raccourcis d'écriture :

— $V^-\ \equiv \{1, .., n\}$ => les clients
— $V^d\ \equiv V^-\ U\ \{d\} \equiv \{1, .., n\}\ U\ \{d\}$ => les clients + départ
— $V^a\ \equiv V^-\ U\ \{a\} \equiv \{1, .., n\}\ U\ \{a\}$ => les clients + arrivée
— $T \equiv \{1, .., m\}$ => les véhicules

et des variables du problème :

— p_{ik} avec $i \in V^-$ et $k \in T$: Associée à un client i, $p_{ik} = 1$ si le client a déjà été visité (et le profit récupéré), $p_{ik} = 0$ sinon. Le k représente le véhicule qui a récupéré le profit.
— t_{ijk} avec $i,\ j \in V^-$ et $k \in T$: Associée à un arc (i, j) et un véhicule k, $t_{ijk} = 1$ si l'arc a été utilisé par le véhicule k, $t_{ijk} = 0$ sinon. Cette variable permettra de simplifier le calcul des coûts de tournées et facilitera grandement le calcul et la formulation.

2.3 Objectif

Maximiser le profit sur une tournée de k véhicules tout en respectant les contraintes imposées, avec $i \in V^-$ et $k \in T$:

$$max \sum_{i \in V} \sum_{k \in T} P_i * p_{ik}$$

2.4 Contraintes

1. Contraintes d'intégrité des variables binaires introduites :

$$p_{ik} \in \{0,\ 1\},\ \forall i \in V^-,\ \forall k \in T \tag{2.1}$$

$$t_{ijk} \in \{0,\ 1\},\ \forall i,\ j \in V,\ \forall k \in T \tag{2.2}$$

2. Les profits de d et a sont nuls :

$$P_d = 0 \tag{2.3}$$

$$P_a = 0 \tag{2.4}$$

3. Les tournées commencent par d et se terminent par a :

$$\sum_{j \in V^a} t_{djk} = 1,\ \forall k \in T \tag{2.5}$$

$$\sum_{i \in V^d} t_{iak} = 1,\ \forall k \in T \tag{2.6}$$

4. Le chemin proposé pour un véhicule doit être cohérent (conservation) :

$$\sum_{i \in V^d \setminus \{x\}} t_{ixk} = \sum_{j \in V^a \setminus \{x\}} t_{xjk} \ = \ p_{xk},\ \forall x \in V^-,\ \forall k \in T \tag{2.7}$$

5. Une seule voiture récupère une et une seule fois le profit (chemin élémentaire) :

$$\sum_{k \in T} p_{ik} \leq 1,\ \forall i \in V^- \tag{2.8}$$

6. La tournée du véhicule k ne doit pas dépasser le coût de valeur L :

$$\sum_{i \in V^d} \sum_{j \in V^a} C_{ij} * t_{ijk} \ \leq \ L \ avec\ i \neq j \tag{2.9}$$

7. Si un client est trop loin (le coût pour y aller est supérieur à L), il n'est visité par aucun véhicule :

$$(C_{di} + C_{ia} \geq L,\ \forall i \in V^-) \implies \sum_{k \in T} p_{ik} = 0\ et\ \sum_{k \in T} t_{ijk} = 0,\ \forall i,\ j \in V^- \tag{2.10}$$

   Cette contrainte est cependant incluse dans l'équation précédente (2.9).

Nous pouvons ajouter aux contraintes précédentes une nouvelle contrainte de manière à optimiser le modèle qui permet de gérer les sous-cycles (pour ne pas boucler inutilement) :

$$\sum_{i,j \ \in \ U,\ i<j} t_{ijk} \ \leq \ |U| \ - \ 1,\ \forall U \subset V,\ |U| \geq 3,\ \forall k \in T \tag{2.11}$$

On prend V et non V^- car on souhaite empêcher tous les sous-cycles, y compris ceux du départ et de l'arrivée.

2.5 Formulation linéaire

Issue des contraintes, la formulation linéaire est :

$$max \sum_{i \in V^-} \sum_{k \in T} P_i * p_{ik} \tag{1}$$

$$\sum_{j \in V^a} t_{djk} = 1, \qquad \forall k \in T \tag{2}$$

$$\sum_{i \in V^d} t_{iak} = 1, \qquad \forall k \in T \tag{3}$$

$$\sum_{i \in V^d \setminus \{x\}} t_{ixk} = \sum_{j \in V^a \setminus \{x\}} t_{xjk} \; = \; p_{xk}, \qquad \forall x \in V^-, \; \forall k \in T \tag{4}$$

$$\sum_{k \in T} p_{ik} \leq 1, \qquad \forall i \in V^- \tag{5}$$

$$\sum_{i \in V^d} \sum_{j \in V^a} C_{ij} * t_{ijk} \; \leq \; L \qquad avec \; i \neq j \tag{6}$$

$$\sum_{i,j \; \in \; U, \; i<j} t_{ijk} \; \leq \; |U| \; - \; 1, \quad \forall U \subset V, \; |U| \geq 3, \; \forall k \in T \tag{7}$$

$$P_d = 0 \tag{8}$$

$$P_a = 0 \tag{9}$$

$$p_{ik} \in \{0, \; 1\} \qquad \forall i \in V, \; \forall k \in T \tag{10}$$

$$t_{ijk} \in \{0, \; 1\} \qquad \forall i, \; j \in V, \; \forall k \in T \tag{11}$$

Approches de résolution

Pour répondre à ce problème d'optimisation, les approches privilégiées sont les méthodes exactes qui permettent de trouver la solution la plus satisfaisante. La formulation linéaire en nombres entiers (FLNE) en est une et à l'avantage de regrouper des algorithmes efficaces et fiables qui sont facilement accessibles. C'est donc une très bonne candidate pour résoudre les problèmes d'optimisation à condition de pouvoir les traduire en contraintes linéaires, ce qui n'est pas toujours possible.

Cependant, au delà de la formalisation du problème, la FLNE n'est pas toujours la solution la plus pertinente car elle peut parfois nécessiter des temps de calcul incompatibles avec les cas d'usages réels, comme c'est le cas ici. En effet, cette méthode ne sera pas la plus adaptée pour notre problème de par les grandes instances qui la composent.

Nous allons donc résoudre ce problème avec diverses méthodes approchées qui permettront d'obtenir des solutions satisfaisantes. Pour cela, plusieurs méthodes de résolution s'offrent à nous et peuvent engendrer des résultats singulièrement différents en fonction des données entrées.

Au travers des cours, nous avons pu parcourir différentes heuristiques, qui adaptées, peuvent nous donner des résultats appréciables. De plus, la littérature sicentifique nous a permise de composer un portefeuille de méthodes permettant d'obtenir de bons résultats. Nous allons donc au sein de cette partie les détailler puis présenter les solutions retenues pour l'implémentation. L'idée est de pouvoir les comparer entre elles puis comparer nos résultats avec ceux de nos camarades.

3.1 Heuristique de Clarke et Wright

La première heuristique qui a attiré notre attention est celle de Clarke et Wright. En effet, cette heuristique a été spécialement conçue en 1964 pour la résolution du problème des tournées de véhicules. Elle se base sur l'idée de fusionner les tournées générées individuellement pour chaque client en une seule tournée par groupe de clients. Pour notre problème, cette fusion se fait sur la maximisation des gains en considérant les clients avec les plus grands profits. Cette méthode a été notre cible privilégiée compte-tenu de son adaptabilité puisque l'on peut optimiser les fonctions de gains, ce qui peut par la suite significativement impacter nos résultats.

3.2 Heuristique de Gillet et Miller

L'heuristique de Gillet et Miller se base sur l'attribution d'un clients à un véhicule en fonction de sa place géographique. En effet, en fonction de ses coordonnées géographiques, un client sera plus ou moins affecté à une tournée. L'étape suivante sert à améliorer chacune des tournées initiales obtenues à l'aide de la méthode 2-opt pour obtenir des tournées avec un maximum de profit. Cette heuristique peut donner de bons résultats à grand échelle mais n'est pas la plus adaptée à notre problème car l'affectation de clients à une tournée par coordonnée géographique plutôt que par gain peut entraîner

des pertes conséquentes de profit. De plus, elle fonctionne mieux lorsque le départ et l'arrivée sont identiques, ce qui n'est pas toujours le cas.

3.3 Heuristique de Beasley

Cette heuristique, à l'opposé de Clark et Wright, commence par créer une tournée de tous les clients, même les plus loins sans prendre en compte la capacité des véhicules. Le tour géant est calculé en fonction d'heuristiques comme le PVC, Fletcher... Beasley partage ensuite ce tour en plusieurs tournées en fonction de la capacité des véhicules en cherchant le chemin le plus otimal entre deux clients. A l'instar de Clark et Wright, cette heuristique est facilement implémentable et ajustable en fonction des différentes heuristiques prises en compte et elle peut fournir de très bons résultats.

3.4 Heuristique beam search

Cette heuristique est une des solutions que nous avons trouvé dans la littérature scientifique et qui nous a beaucoup plu de par les très bons résultats qu'elle propose. C'est une méthode arborescente qui se limite à examiner seulement un nombre restreint de descendants pour chaque nœud. Elle optimise la recherche en largeur traditionnelle, réduisant ainsi l'espace mémoire et le temps d'exécution nécessaires à son fonctionnement. Cette technique se caractérise donc par son approche gloutonne.

Dans son processus, la recherche en faisceau s'appuie sur l'algorithme de parcours en largeur pour naviguer à travers le graphe. À chaque étape, elle produit tous les successeurs du nœud actuel et les classes en fonction de leur coût heuristique. Toutefois, seulement un nombre prédéfini de ces états, correspondant à la largeur du faisceau, est conservé à chaque niveau. Une largeur de faisceau plus étendue implique moins d'états négligés. Si cette largeur est infinie, la méthode prend en compte tous les états, se transformant ainsi en une recherche en largeur classique. L'efficacité du beam search repose fortement sur l'heuristique utilisée pour évaluer le potentiel de chaque nœud.

3.5 Méthode de recherche locale

3.5.1 Méthode 2-opt

La méthode 2-opt est une technique d'optimisation simple et efficace utilisée pour améliorer les solutions dans le contexte des problèmes de tournées de véhicules. Elle vise à réduire le coût total d'une tournée en échangeant deux arêtes. L'idée est d'éliminer les croisements d'arêtes dans la tournée, ce qui peut réduire la distance totale parcourue. Dans les méthodes que nous avons implémentées, nous avons utilisé cette technique après chaque ajout d'un nouveau client dans une tournée. Ceci est dans le but de minimiser le temps de parcours, permettant ainsi de consacrer plus de temps à la visite des autres clients.

3.5.2 Méthode 3-opt

A l'instar de 2-opt, 3-opt cherche des groupes de 3 arêtes non succesives dans une solution existante et essaie de les réorganiser pour obtenir une solution de meilleure qualité. Cet algorithme est plus lent que le 2-opt mais peut générer des solutions plus intéressantes.

3.6 Choix final d'implémentation

Après avoir balayé plusieurs méthodes de résolutions, nous avons décidé d'en implémenter deux : l'heuristique de **Clarke et Wright**, et l'heuristique **beam search**, les deux **améliorées par 2-opt**.

Implémentation de la solution

4.1 Solutions implémentées - nos choix

Vous pouvez retrouver notre code ici : **dépôt Git**.

Nous avons donc décidé d'implémenter deux solutions pertinentes en python, qui correspondent à notre problème :

— Heuristique de Clark et Wright + 2-opt (exécuter main_clarke_wright.py)
— Beam search + 2-opt (exécuter main_beam.py)

Ces dernières ont été implémentées en python. Les instances utilisées pour faire tourner nos algorithmes peuvent être récupérées ici : **data**.

4.2 Heuristique de Clark et Wright + 2-opt

4.2.1 Idée de fonctionnement de l'algorithme

1. Nous initialisons la 'marguerite' : création de n tournées correspondant à la prise en charge d'un seul client. Le chemin est donc : [(d, n_i), (n_i, a)].
2. Nous créons ensuite une liste déterminant les économies réalisées entre chaque couple client, utile pour réaliser les fusions par la suite. Ces économies sont triées par ordre décroissant dans une liste. La fonction `SavingList`, qui génère cette liste, est conçue pour prendre en compte à la fois le gain de temps et le profit, tout en accordant une priorité plus importante au profit par rapport au gain de temps. Ainsi, entre deux arcs offrant le même gain de temps, celui avec le plus grand profit sera choisi en premier.
3. Pour chaque arc, nous vérifions si choisir cet arc entre deux clients est bel et bien une économie. Si oui, nous effectuons la fusion et intégrons l'arc dans la tournée, sinon nous gardons la tournée telle qu'elle est.
4. S'il y a fusion, nous optimisons la tournée obtenue grâce à la méthode de recherche locale 2-opt.
5. Finalement, nous récupérons les m tournées générant le plus de profit : ce sont les solutions du problème.

4.2.2 Complexité

La complexité de la fonction `clarke_wright` varie en fonction du nombre de nœuds et d'arcs dans le graphe. Dans le meilleur des cas, où les opérations à l'intérieur de la boucle principale sont minimisées, la complexité est de l'ordre de $O(n^2)$.

Dans le pire des cas, si l'on suppose que la boucle `while` s'exécute pour chaque paire d'arcs potentiels dans un graphe complet, le nombre d'itérations de cette boucle peut atteindre $O(n^2)$. Étant donné que l'opération la plus coûteuse à l'intérieur de cette boucle, l'optimisation des tournées (`opt_2`), a également une complexité de $O(n^2)$, la complexité globale de la fonction dans le pire des cas pourrait s'élever à $O(n^4)$.

4.2.3 Résultats

Les résultats sont consignées dans le fichier : output_clark_wright_all.csv. Il regroupe les instances de tous les sets de données. Pour analyser la performance de notre algorithme, nous avons également enregistré les temps d'exécution de notre programme. Ces résultats proviennent de l'exécution des algorithmes sur un ordinateur sous Windows 11, avec 16 GB RAM et un processeur de 1,20 GHz.

4.3 Beam search + 2-opt

Bien que l'heuristique précédemment mise en œuvre a abouti à une solution, notre objectif était d'atteindre des résultats plus proches de la solution idéale. Cette ambition nous a guidées vers l'exploration de l'algorithme de recherche en faisceau BSCBP (Beam Search for Computing the Best Paths).

4.3.1 Idée de fonctionnement de l'algorithme

Comme décrit dans la section 3.4, la méthode parcourt un arbre, chaque nœud de cet arbre représentant l'état d'avancement d'une tournée.

Voici la procédure de cet algorithme :

1: Let B be the set containing the nodes at a given level of the tree;
2: Let B_{off} be the offspring nodes (descendants of nodes in B);
3: **for** $k = 1$ to m **do**
4: $\eta_0 \leftarrow \{P^+ = \{v_0\}, P^- = V'\}$ (the root node)
5: Set η_0.score $\leftarrow 0$ and η_0.time $\leftarrow 0$;
6: Set $B \leftarrow \{\eta_0\}$ and $\ell \leftarrow 0$;
7: $\eta^* \leftarrow \eta_0$; (the best solution found for the k^{th} path)
8: **while** $(B \neq \emptyset)$ **do**
9: Branch out of each node $\eta_{\ell_j} \in B$ and create the offspring nodes B_{off};
10: $\ell \leftarrow \ell + 1$;
11: **for each** nodes $\eta_{\ell_j} \in B_{off}$ **do**
12: Apply the 3-opt local optimization on the partial path P_j^+ of the node in order to try to decrease η_{ℓ_j}.time;
13: **end for**
14: Remove from B_{off} the nodes that will violate the T_{max} constraint if adding the end point v_{n+1};
15: **if** $P^- = \emptyset$ for a node $\eta_{\ell_j} \in B_{off}$ **then**
16: Add vertex v_{n+1} (end point) to that path and compute the total time and score;
17: **if** (η_{ℓ_j}.score$>$ η^*.score) **then**
18: $\eta^* \leftarrow \eta_{\ell_j}$;
19: Remove η_{ℓ_j} from B_{off};
20: **end if**
21: **end if**
22: Sort nodes in B_{off} according to the selection criterion ρ and keep only the $\max(\omega, |B_{off}|)$ first nodes, remove the other nodes from B_{off};
23: $B \leftarrow B_{off}$;
24: $B_{off} \leftarrow \emptyset$;
25: **end while**
26: Assign to $P_k \in \mathbb{P}$ the path P^+ stored in node η^*;
27: Update set V' by removing the vertices used in path P_k;
28: **end for**

FIGURE 4.1 – BSCBP proposé par A.Bouchakhchoukha, T. Menouer et N.Sukhija

— B correspond à liste des nœuds à un niveau donné d'un arbre.
— μ correspond au noeud de l'arbre.
— B_{off} correspond à la liste des nœuds fils de μ.
— m est le nombre maximal de tournées à créer.

— V' est la liste des clients non visités par toutes les tournées.
— P^+ correspond à la liste des clients visités à un état μ.
— P^- correspond à la liste des clients non visités à un état μ .
— μ^* correspond au meilleur état trouvé ayant le plus de profit.
— w correspond au largeur du faisceau qui est le nombre de noeuds à exploiter à chaque niveau de l'arbre.
— p est le critère de sélection des noeuds fils.

Le critère de sélection des nœuds fils est le suivant : le sommet le plus proche est choisi comme prochain point à visiter. En cas d'égalité de distance entre plusieurs sommets, celui offrant le profit le plus élevé est privilégié. À chaque niveau de l'arbre, l'algorithme s'efforce d'explorer les w meilleurs nœuds, selon ce critère de sélection. Le parcours s'arrête lorsqu'il devient impossible d'ajouter d'autres nœuds à la fin de la tournée, en raison des contraintes de temps. Enfin, les nœuds feuilles sont évalués en fonction du profit total du parcours, et l'état générant le profit le plus élevé est sélectionné pour la tournée.

4.3.2 Complexité

La complexité temporelle de Beam Search est $O(w * d)$, tel que 'w' est le nombre de noeuds visités à chaque niveau de l'arbre et 'd' est le profondeur de l'arbre, c'est à dire le nombre maximale d'étapes que l'algorithme a efféctué. Cela signifie qu'à chaque niveau de l'arbre, jusqu'à w nœuds sont explorés, et ce processus se répète pour chaque niveau jusqu'à la profondeur maximale 'd'.

4.3.3 Résultats

Les résultats de cet algorithme seront consignés dans des fichiers csv. Il suffit pour cela d'entrer le nom du fichier dans lequel vous souhaitez obtenir les résultats dans la variable *nom_fichier_csv*. Vous choisissez ensuite le set de données et les fichiers qui vous intéressent. Dans le dépôt git, nous avons par exemple le fichier résultat : output_beam_all_p1.csv.

4.4 Discussion autour des approches

Pour répondre au problème des tournées, nous avons balayés une multitude de méthodes afin de nous rapprocher de la solution exacte. Appliquées sur des instances différentes, ces dernières nous ont offerts des résultats exploitables qui nous permettront de conclure sur la performance des algorithmes impliqués.

Nous avons décidés de comparer les résultats de quatre différents algorithmes :
— Heuristique de Gillet et Miller
— Heuristique de Beasley
— Heuristique de Clark and Wright
— Beam search

Les résultats des deux premiers ont été récupérés des algorithmes de nos camarades Omar Elloumi & Julien Pillis qui les ont testés sur les mêmes instances. Les deux autres algorithmes ont été effectués par nos soins. Tous les algorithmes ont été optimisés, tous utilisent la méthode de recherche locale 2-opt, Beasley étant en plus fortement optimisé.

Les résultats sont présentés dans les tableaux ci-dessous :
— Les instances sont à récupérer dans le dossier **data**
— L : temps de parcours limite pour chaque véhicule.
— n : nombre de clients considérés
— m : nombre de véhicules considérés (rangé par couleur)

Conjectures

Pour pouvoir comparer les résultats, nous avons fait varier le nombre de clients (32 ou 66), le nombre de véhicules (2, 3 ou 4), ainsi que les temps de parcours limites.

Plusieurs points sont à noter. Premièrement, pour des petits temps limites de parcours (L) avec 32 clients, les résultats sont quasiment identiques. L'écart se creuse alors à mesure que L augmente, où l'algorithme doit faire des choix entre les meilleurs compromis profit/temps, pour des clients situés un peu plus loin.

Pour les tournées avec peu de véhicules, la méthode de **Clark and Wright** est la plus adaptée car elle propose quasiment à tous les coups la meilleure tournée. La méthode de **Beasley** offre également de très bons résultats et devance Clark and Wright lorqu'on augmente le nombre de véhicules pour un même nombre de clients.

De manière générale, les algorithmes **Beasley, Clark and Wright et Beam Search** sont efficaces car la tournée avec le meilleur gain remporté dépend vraiment de l'instance, les résultats étant assez proches pour 32 clients.

Par ailleurs, nous pouvons remarquer que si nous augmentons le nombre de clients, la méthode **Beam Search** sera alors de loin la plus adaptée car c'est avec celle-ci cette fois que l'on obtient quasiment à tous les coups les meilleures tournées. Cela se fait pourtant au détriment du temps d'exécution qui n'est plus instantané, et qui peut s'élever jusqu'à une demi-minute pour certaines instances testées. Beam Search peut être adapté pour concilier meilleur gain et meilleur temps d'exécution. En effet, l'avantage avec Beam search est que l'on peut gérer sa précision pour pouvoir adapter les temps à nos contextes réels d'utilisation. L'algorithme se base sur le facteur w, qui permet de limiter le nombre de noeuds exploités pendant son parcours. Ainsi, plus w_{max} sera élevé, plus le résultat se rapprochera du résultat exact, tout en augmentant bien-sûr les temps d'exécution. Dans notre implémentation, nous avons posé que $w_{max} = 10$, et que donc w ne pourra pas excéder w_{max}.

Le choix de la méthode dépend donc des contraintes posées par le problème, ce qui nécessitera la mise en place de simulations pour pouvoir dégager la meilleure tournée. Notons que, sans surprise, l'heuristique de Gillet et Miller n'offre pas de très bon résultat sur ce problème.

				Gillet et Miller + 2-opt		Beasley optimisé++		Clark and Wright + 2-opt		Beam Search + 2-opt	
instances	L	n	m	Profit	Tps exécution	Profit	Tps exécution	Profit	Tps exécution	Profit	Tps exécution
Set_32_234/p1.2.b	5.0	32	2	15	0,026	15	0,160	15	0.001	15	0.001
Set_32_234/p1.2.c	7.5	32	2	20	0,043	20	0,150	20	0.001	20	0.002
Set_32_234/p1.2.d	10.0	32	2	30	0,064	25	0,170	30	0.002	30	0.013
Set_32_234/p1.2.e	12.5	32	2	40	0,048	45	0,160	40	0.006	40	0.1008
Set_32_234/p1.2.f	15.0	32	2	65	0,083	70	0,170	70	0.0089	70	0.3115
Set_32_234/p1.2.g	17.5	32	2	75	0,106	90	0,170	75	0.0108	85	0.9655
Set_32_234/p1.2.h	20.0	32	2	110	0,180	100	0,180	85	0.0116	110	1.5382
Set_32_234/p1.2.i	23.0	32	2	120	0,257	120	0,190	120	0.0124	110	3.1939
Set_32_234/p1.2.j	25.0	32	2	125	0,241	140	0,190	115	0.0142	140	5.5137
Set_32_234/p1.2.k	27.5	32	2	140	0,265	155	0,200	160	0.016	160	6.6445
Set_32_234/p1.2.l	30.0	32	2	155	0,300	175	0,210	175	0.0166	180	7.9953
Set_32_234/p1.2.m	32.5	32	2	175	0,387	180	0,220	200	0.0271	180	10.2406
Set_32_234/p1.2.n	35.0	32	2	190	0,400	205	0,230	230	0.0249	220	12.0809
Set_32_234/p1.2.o	36.5	32	2	190	0,413	210	0,240	240	0.0217	230	12.8319
Set_32_234/p1.2.p	37.5	32	2	190	0,432	215	0,240	240	0.0243	230	13.6089
Set_32_234/p1.2.q	40.0	32	2	220	0,491	245	0,260	245	0.023	235	14.6916
Set_32_234/p1.2.r	42.5	32	2	225	0,537	220	0,260	275	0.0281	235	18.8384
Set_32_234/p1.3.c	5.0	32	3	15	0,026	15	0,170	15	0.0021	15	0.001
Set_32_234/p1.3.d	6.7	32	3	15	0,026	15	0,150	15	0.0011	15	0.0
Set_32_234/p1.3.e	8.3	32	3	30	0,054	30	0,160	30	0.001	30	0.024
Set_32_234/p1.3.f	10.0	32	3	40	0,087	40	0,160	40	0.0021	40	0.034
Set_32_234/p1.3.g	11.7	32	3	50	0,055	50	0,160	50	0.0037	50	0.1377
Set_32_234/p1.3.h	13.3	32	3	70	0,103	65	0,160	70	0.006	60	0.3635
Set_32_234/p1.3.i	15.3	32	3	85	0,102	95	0,170	95	0.0081	85	0.9125
Set_32_234/p1.3.j	16.7	32	3	100	0,137	100	0,170	95	0.0134	100	0.8975
Set_32_234/p1.3.k	18.3	32	3	120	0,136	125	0,180	115	0.0141	125	1.2571
Set_32_234/p1.3.l	20.0	32	3	150	0,214	145	0,180	120	0.014	145	1.6852
Set_32_234/p1.3.m	21.7	32	3	165	0,212	160	0,190	130	0.013	165	2.6764
Set_32_234/p1.3.n	23.3	32	3	170	0,238	180	0,190	175	0.0141	175	3.2609
Set_32_234/p1.3.o	24.3	32	3	175	0,244	180	0,190	195	0.0147	190	4.9657
Set_32_234/p1.3.p	25.0	32	3	180	0,246	195	0,190	170	0.015	190	4.5713
Set_32_234/p1.3.q	26.7	32	3	200	0,265	220	0,200	205	0.016	205	8.9181
Set_32_234/p1.3.r	28.3	32	3	205	0,278	230	0,200	215	0.016	220	8.6292
Set_32_234/p1.4.d	5.0	32	4	15	0,026	15	0,150	15	0.004	15	0.0
Set_66_234/p5.2.b	5.0	66	2	20	0,266	20	1,480	20	0.002	20	0.005
Set_66_234/p5.2.c	7.5	66	2	40	0,286	45	1,470	40	0.006	50	0.0548
Set_66_234/p5.2.d	10.0	66	2	65	0,361	80	1,490	70	0.006	80	0.1535
Set_66_234/p5.2.e	12.5	66	2	120	0,463	160	1,480	160	0.0181	180	1.3262
Set_66_234/p5.2.f	15.0	66	2	170	0,646	210	1,690	160	0.032	195	2.306
Set_66_234/p5.2.g	17.5	66	2	310	1,282	310	1,530	220	0.0638	320	7.3175
Set_66_234/p5.2.h	20.0	66	2	370	1,368	370	1,590	240	0.074	410	11.3023
Set_66_234/p5.2.i	22.5	66	2	405	1,589	400	1,600	380	0.0702	465	23.3151
Set_66_234/p5.2.j	25.0	66	2	445	1,804	540	1,630	580	0.0814	560	24.2376
Set_66_234/p5.2.k	27.5	66	2	495	2,281	600	1,700	650	0.0818	670	36.8288
Set_66_234/p5.2.l	30.0	66	2	500	2,565	665	1,750	650	0.0812	770	40.3585
Set_66_234/p5.2.m	32.5	66	2	570	3,046	735	1,990	650	0.0829	860	62.9305
Set_66_234/p5.2.n	35.0	66	2	715	3,866	840	2,000	810	0.0863	815	49.8554
Set_66_234/p5.2.o	37.5	66	2	715	4,029	880	1,900	840	0.0955	900	67.6064
Set_66_234/p5.2.p	40.0	66	2	715	4,233	945	1,960	1140	0.0957	1150	69.243
Set_66_234/p5.2.q	42.5	66	2	715	4,640	1000	2,490	1140	0.1175	1195	101.9715
Set_66_234/p5.2.r	45.0	66	2	815	5,309	1090	2,170	1140	0.0966	1060	99.595
Set_66_234/p5.3.b	3.3	66	3	15	0,165	15	2,380	15	0.0021	15	0.0021
Set_66_234/p5.3.c	5.0	66	3	20	0,239	20	1,460	20	0.0031	20	0.004
Set_66_234/p5.3.d	6.7	66	3	55	0,331	60	1,650	60	0.0041	50	0.0427
Set_66_234/p5.3.e	8.3	66	3	70	0,405	90	1,850	95	0.0041	95	0.0998
Set_66_234/p5.3.f	10.0	66	3	95	0,474	110	1,510	105	0.007	110	0.1775
Set_66_234/p5.3.g	11.7	66	3	165	0,634	175	1,540	120	0.0186	175	0.9211
Set_66_234/p5.3.h	13.3	66	3	195	0,651	220	1,540	240	0.0164	260	1.6538
Set_66_234/p5.3.i	15.0	66	3	270	0,879	270	1,530	240	0.0338	310	4.2012
Set_66_234/p5.3.j	16.7	66	3	460	1,384	405	1,560	315	0.0471	470	7.5455
Set_66_234/p5.3.k	18.3	66	3	465	1,662	495	1,550	360	0.0641	455	9.6123
Set_66_234/p5.3.l	20.0	66	3	555	1,722	540	1,610	360	0.0732	595	15.8225
Set_66_234/p5.3.m	21.7	66	3	555	1,867	580	1,660	580	0.0742	650	17.8957
Set_66_234/p5.3.n	23.3	66	3	590	1,976	635	1,650	645	0.0731	755	25.546
Set_66_234/p5.3.o	25.0	66	3	665	2,404	790	1,680	845	0.0759	830	26.8522
Set_66_234/p5.3.p	26.7	66	3	740	2,698	865	1,750	975	0.0803	990	35.0116
Set_66_234/p5.3.q	28.3	66	3	745	2,672	910	1,720	975	0.0815	985	44.8509
Set_66_234/p5.3.r	30.0	66	3	745	2,979	1030	1,820	975	0.0827	1010	71.3456
Set_66_234/p5.4.c	3.8	66	4	20	0,215	20	1,460	20	0.0041	20	0.005
Set_66_234/p5.4.d	5.0	66	4	20	0,215	20	1,470	20	0.0041	20	0.0045
Set_66_234/p5.4.e	6.2	66	4	20	0,216	20	1,440	20	0.004	20	0.006
Set_66_234/p5.4.f	7.5	66	4	80	0,542	80	1,460	80	0.0049	80	0.0685
Set_66_234/p5.4.g	8.8	66	4	80	0,498	125	1,480	120	0.0081	140	0.1791
Set_66_234/p5.4.h	10.0	66	4	115	0,601	140	1,520	140	0.005	140	0.2131
Set_66_234/p5.4.i	11.2	66	4	190	0,859	190	1,530	180	0.0126	240	0.7886
Set_66_234/p5.4.j	12.5	66	4	240	0,854	300	1,530	320	0.0171	340	1.9289
Set_66_234/p5.4.k	13.8	66	4	260	0,822	295	1,720	320	0.016	340	1.9655
Set_66_234/p5.4.l	15.0	66	4	340	0,113	345	1,530	320	0.0383	390	4.8526
Set_66_234/p5.4.m	16.2	66	4	500	1,260	495	1,570	450	0.0459	530	13.4177
Set_66_234/p5.4.n	17.5	66	4	615	1,908	620	1,610	440	0.0631	620	15.4075
Set_66_234/p5.4.o	18.8	66	4	620	1,930	670	1,620	540	0.0597	650	13.2776
Set_66_234/p5.4.p	20.0	66	4	725	1,999	710	1,600	480	0.0709	740	26.0506
Set_66_234/p5.4.q	21.2	66	4	710	2,130	725	1,630	670	0.0724	860	30.1972
Set_66_234/p5.4.r	22.5	66	4	775	2,231	770	1,600	760	0.0771	905	28.7138

FIGURE 4.2 – Comparaison des profits et temps d'exécution pour quatres techniques distinctes en fonction de paramètres donnés

Conclusion

Le problème des m-tournées sélectives est un ancien problème d'optimisation qui peut être approché par de multiples heuristiques développées à cet effet. En effet, faisant partie du groupe des problèmes NP-difficile, nous n'avons toujours pas trouvé d'algorithmes capables de le résoudre en temps polynomial. C'est pour cette raison que nous faisons appel à des méthodes approchées qui tentent de déterminer le meilleur résultat.

Pour répondre à ce problème, nous l'avons formalisé en FLNE mais avons été bloquées par les temps d'exécution de cette méthode exacte. Nous avons donc implémenté plusieurs algorithmes approchés qui ont présenté des résultats très satisfaisants. Cependant, ces résultats sont intrinsèquement liés aux contraintes imposées qui vont impacter la qualité des résultats. Il faut donc choisir la méthode la plus adaptée au problème en fonction des paramètres ; soit en ayant de l'expérience, soit en effectuant des simulations comme nous l'avons fait dans ce projet afin de pouvoir dégager la meilleure tournée.

Sur des instances à échelle industrielle, nous pouvons affirmer que l'algorithme le plus intéressant est le Beam Search optimisé avec la méthode 2-opt. Les temps d'exécution peuvent être un peu amélioré en utilisant des langages performants avec des accès directs en mémoire par exemple (comme en C ou en C++) bien que c'est principalement la complexité de l'algorithme qui influe sur les temps d'exécution. Il faut donc choisir et faire un compromis entre la performance, la précision des résultats et les temps d'exécution qui peuvent être importants. Avec cet algorithme, la précision peut être améliorée en augmentant le facteur w_{max} ou en implémentant la méthode de recherche locale 3-opt (plus efficace que 2-opt mais plus lente).

Bibliographie

Thèses scientifiques desquels nous nous sommes inspirées pour ce projet :
— El-Hajj, R., Dang, D., & Moukrim, A. (2013). Un algorithme de branch-and-cut pour la résolution du problème de tournées sélectives. ResearchGate.
— Khemakhem, M. (2008, 1 février). Heuristiques pour un problème de m-Tournées sélectives.
— Chabchoub, H. (2017). Le probleme de m-tournées séléctives : une approche basée sur la méthode des centres mobiles. fsegs.
— Isoart, N. (2021, 19 novembre). The traveling salesman problem in constraint programming.
— A hybrid heuristic for the team orienteering problem. (2017, août 1). IEEE Conference Publication | IEEE Xplore.
— Roozbeh I., Ozlen M. , Hearne J.(2016, 19 août). A heuristic scheme for the Cooperative Team Orienteering Problem with Time Windows