Approches de résolution du Team Orienteering Problem (TOP)

AI09 - Méthodes et outils pour l'optimisation et la simulation

Omar ELLOUMI, Julien PILLIS

Décembre 2023

1 Introduction

Le "Team Orienteering Problem" (TOP), est un défi complexe relevant du domaine de l'optimisation combinatoire. Le TOP consiste en la recherche de solutions efficaces pour des équipes de personnes ou de véhicules devant visiter un ensemble de lieux prédéfinis dans un ordre optimal, afin de maximiser la somme des profits de chaque noeud collecté par les individus/véhicules. Cependant, une contrainte majeure de temps/distance s'impose pour chaque élément collecteur : toute tournée ne doit pas dépasser cette contrainte.

Ce problème trouve des applications concrètes dans des domaines tels que la logistique, la planification d'itinéraires pour des équipes de maintenance, ou même la conception de circuits touristiques. Dans ce contexte, la résolution du TOP nécessite des algorithmes avancés et des approches innovantes pour trouver des solutions optimales ou satisfaisantes. Variante des Vehicule Routing Problems with Profits (VRPP) le TOP est un problème NP-difficile. L'explosion combinatoire que le problème engendre ne permet pas de trouver directement la solution optimale. L'utilisation d'heuristiques est alors inévitable, mais permet cependant d'obtenir de très bons résultats.

Au cours de ce rapport, nous vous présenterons la formulation linéaire en nombres entiers du problème afin d'expliciter les contraintes du problème. Dans un second temps, nous discuterons des méthodes de résolutions existantes, puis nous terminerons par la présentation des algorithmes que nous avons implémentés et leurs performances.

2 Formulation linéaire en nombres entiers

Les contraintes du TOP sont nombreuses. Réaliser la formulation linéaire en nombres entier permet d'en prendre connaissance, et de les expliciter pour pourvoir les prendre en compte lors de l'implémentation.

Pour cela, modélisons dans un premier temps les données du problème :

- F : la flotte
- L : le temps de parcours limite pour chaque véhicule
- m : le nombre de véhicules de la flotte
- n : le nombre de clients
- d : le dépôt de départ
- a : le dépôt d'arrivée
- $V = \{1, ..., n\} \cup \{d, a\}$: l'ensemble des clients et des dépôts
- V^- : l'ensemble des clients
- V^d : l'ensemble des clients et du dépôt de départ
- V^a : l'ensemble des clients et du dépôt d'arrivée
- $E = \{(i, j) | i, j \in V\}$: l'ensemble des arcs
- $G = (V, E)$: le graphe complet permettant de modéliser le TOP
- P_i : le profit du passage chez le client i
- $x_i^k \in 0, 1$: variable valant 1 si le profit du client i a été collecté par le véhicule k, 0 sinon
- $t_{i,j}^k$: variable valant 1 si l'arc *(i,j)* est traversé par la tournée/le véhicule k
- $C_{i,j}$: temps de trajet associé à l'arc *(i,j)*

$$max \sum_{i \in V^-} \sum_{k \in F} P_i * x_i^k \tag{1}$$

$$x_i^k \in 0,1 \quad \forall i \in V^- \quad \forall k \in F \tag{2}$$

$$t_{i,j}^k \in 0,1 \quad \forall i \in V, \forall j \in V, \forall k \in F \tag{3}$$

$$\sum_{i \in V^d, j \in V^a \setminus \{i\}} t_{i,j}^k * C_{i,j} \leq L \quad \forall k \in F \tag{4}$$

$$\sum_{j \in V^a} t_{d,j}^k = 1 \quad \forall k \in F \tag{5}$$

$$\sum_{i \in V^d} t_{i,a}^k = 1 \quad \forall k \in F \tag{6}$$

$$\sum_{i \in V^d \setminus \{j\}} t_{i,j}^k = \sum_{l \in V^a \setminus \{j\}} t_{j,l}^k = x_i^k \quad \forall k \in F \quad \forall j \in V^- \tag{7}$$

$$\sum_{k \in F} x_i^k \leq 1 \quad \forall i \in V^- \tag{8}$$

$$\sum_{i \in S, j \in S} t_{i,j}^k \leq |S| - 1 \quad \forall S \subseteq V^-, |S| > 1, \forall k \in F \tag{9}$$

- (1) : Fonction objectif, maximisation de la somme des profits collectés
- (2) : Contrainte d'unicité pour la collecte du profit (intégrité)
- (3) : Contrainte permettant d'indiquer si la tournée k passe par l'arc *(i,j)* (intégrité)
- (4) : Contrainte de temps limite de trajet
- (5) : Toute tournée a strictement un arc partant du départ d (connectivité)
- (6) : Toute tournée a strictement un arc arrivant à l'arrivée a (connectivité)
- (7) : Conservation des flux (connectivité) : tout arc entrant sur un noeud doit avoir un arc sortant de ce noeud (à l'exception des noeud de départ et d'arrivée).
- (8) : Un client ne peut être desservi au plus par un seul flot (contrainte élémentaire d'un problème de tournées)
- (9) : Elimination des sous-circuits pour chaque flot (méthode de coupe)

Avec l'énumération des contraintes, nous nous apercevons que leur nombre est exponentiel. Il est difficilement envisageable d'appliquer une telle méthode sur de grandes instances. Pour cela, nous allons donc étudier différentes approches de résolution basées sur une adaptation des méthodes et algorithmes étudiés en cours pour la résolution des problèmes TSP et VRP. Ainsi, nous pourrons espérer obtenir un solution satisfaisante (mais pas forcément optimale) pour le TOP, en temps polynomial.

3 Méthodes de résolution du TOP

Au cours de ce semestre, nous avons étudié différentes heuristiques de résolution et d'optimisation du TSP (Problème du voyageur de commerce) et des VRP. Nous vous proposons regarder comment ces méthodes pourraient s'adapter au problème du TOP.

3.1 Heuristiques constructives

3.1.1 Heuristique de Gillett et Miller

L'heuristique de Gillett et Miller peut convenir pour la résolution du TOP. La fin d'une tournée pourrait être déterminée lorsque le chemin du noeud de départ jusqu'à l'arrivée, en passant par le dernier noeud ajouté ne respecte plus la contrainte de temps/distance maximum (le chemin formé serait donc le chemin ne comptant pas ce dernier noeud).
Cependant, le tri par angle polaire croissant des noeuds par rapport au noeud de dépôt ne semble pas être l'heuristique la plus optimale dans le cas général. En effet, l'heuristique est reconnue pour fournir de bons résultats avec un dépôt central mais ce critère ne peut pas être vérifié par une instance du TOP. Nous pouvons nous attendre à obtenir de moins bons résultats qu'une heuristique n'étant pas particulièrement adaptée à un dépôt central.

3.1.2 Heuristique de Clarke & Wright

Cette heuristique semble également convenir pour la résolution du TOP. Pour obtenir une solution du problème, il suffit de ne garder que les n tournées (pour n véhicules) rapportant le plus de profit. L'heuristique de Clarke & Wright est souvent choisie pour sa simplicité, sa rapidité d'exécution et son adaptation à des situations pratiques de routage de véhicules.

3.1.3 Heuristique de Beasley

L'heuristique de Beasley peut aussi s'adapter au TOP. Pour cela, on choisit une heuristique du PVC (par exemple : MI, PLI PPI, PPV, Shamos, Fletcher...). On part de la solution fournie par l'heuristique du PVC choisie. Pour chaque véhicule, tant que la limite L n'est pas dépassée par la distance entre tous les arcs, on essaie d'ajouter le noeud suivant du chemin. On obtient alors plusieurs tournées.

L'heuristique de Beasley est relativement simple à comprendre et à implémenter. Cela peut être avantageux si la simplicité de l'algorithme est une priorité.

Bien que l'heuristique de Beasley ne garantisse pas une solution optimale, elle peut donner de bons résultats dans des situations pratiques et elle est rapide à l'exécution.

3.2 Méta-heuristiques

Les précédentes heuristiques sont efficaces et très rapides. Elles permettent de proposer des solutions en temps polynomial. Malgré tout, bien que les solutions soient généralement satisfaisantes, il arrive qu'elles soient médiocres, notamment sur les grandes instances.

Dans le cas où la recherche d'une solution la plus proche d'une solution optimale soit la priorité (avec acceptation d'un prolongement du temps de calcul), les méta-heuristiques semblent plus adaptées.

Les méta-heuristiques sont des approches algorithmiques puissantes pour résoudre des problèmes d'optimisation complexes. Elles visent à explorer l'espace des solutions de manière efficace, équilibrant l'exploration pour découvrir de nouvelles régions avec l'exploitation pour converger vers des solutions optimales. Ces méthodes sont souvent adaptatives, ajustant dynamiquement leurs stratégies pour améliorer la recherche. Elles offrent des solutions de haute qualité dans des délais raisonnables.

Les méta-heuristiques adaptées au TOP sont nombreuses : Particle Swarm Optimization, Ant Colony Optimization, Genetic Algorithm, Harmony Search, Tabu Search...

Au cours de ce rapport, nous allons nous intéresser à l'optimisation par colonie de fourmis ainsi qu'à l'algorithme génétique.

3.2.1 Optimisation par colonie de fourmis

L'algorithme de colonie de fourmis (Ant Colony Optimization - ACO) est une méta-heuristique inspirée par le comportement des colonies de fourmis lors de la recherche de la meilleure route entre la fourmilière et une source de nourriture. Érigée par Marco Dorigo dans les années 1990, l'ACO a été largement appliqué à divers problèmes d'optimisation combinatoire, mais se prête particulièrement bien au problème du TOP.

Voici les principes de fonctionnement de l'heuristique :

- **Construction de Solutions** : Les fourmis construisent des solutions candidates en suivant des chemins dans l'espace des solutions. Chaque fourmi prend des décisions locales basées sur des informations heuristiques et des traces de phéromones.
- **Phéromones** : Les fourmis déposent une substance chimique appelée phéromone le long de leur chemin. La concentration de phéromones influence le choix des fourmis. Les chemins avec des concentrations plus élevées sont plus attrayants.
- **Évaporation des Phéromones** : Les traces de phéromones s'évaporent avec le temps, évitant que les fourmis ne se fixent trop sur un chemin spécifique. Cela permet une exploration continue de l'espace des solutions.
- **Choix de Chemins** : Les fourmis choisissent leurs chemins en utilisant une combinaison d'informations heuristiques et de la quantité de phéromones. Ce choix est représenté par un tirage alétoire d'un noeud, dont la probabilité de tirage est calculée par une formule. Cette formule prend en compte l'importance relative de la phéromone par rapport à une heuristique (par exemple : la distance entre le noeud de d'arrivée et le noeud tiré).
- **Exploration et Exploitation** : L'ACO équilibre l'exploration (en essayant de nouveaux chemins) et l'exploitation (se concentrant sur les chemins prometteurs). Cela permet de trouver des solutions satisfaisantes, voire optimales, tout en évitant de rester piégé dans des optima locaux.

3.2.2 Algorithme Génétique pour VRP

L'algorithme génétique (AG) est une approche méta-heuristique inspirée par la théorie de l'évolution naturelle. Appliqué au Problème de Tournée de Véhicules (VRP), l'AG vise à trouver une solution optimale ou proche de l'optimale en utilisant des concepts tels que la sélection naturelle, le croisement, et la mutation. Voici les principes fondamentaux de cet algorithme :

- **Initialisation de la Population :** L'algorithme commence par créer une population initiale de solutions candidates. Chaque solution représente une tournée potentielle pour les véhicules, formant ainsi un convoi.
- **Évaluation de la Fitness :** Chaque solution de la population est évaluée en fonction de sa "fitness" ou aptitude. Dans le contexte du VRP, la fitness mesure la qualité de la tournée en termes de profit total et de temps total. Les solutions qui respectent les contraintes de temps et maximisent le profit ont une meilleure fitness.
- **Opérateurs Génétiques :** L'AG utilise trois opérateurs génétiques principaux :

- **Croisement (Crossover) :** Deux solutions parentes sont combinées pour créer des solutions enfants. Pour le VRP, un crossover appelé *Alternating Edges Crossover* est utilisé, où les chemins des parents sont combinés de manière à alterner les arêtes.
- **Mutation :** Des modifications aléatoires sont introduites dans les solutions pour diversifier la population. Différents types de mutations peuvent être appliqués, tels que l'inversion d'un sous-chemin, le changement aléatoire d'un nœud, ou l'échange de positions entre deux nœuds.
- **Sélection :** Les individus de la population sont sélectionnés pour participer au croisement en fonction de leur fitness. Les individus ayant une meilleure fitness ont une probabilité plus élevée d'être sélectionnés, simulant ainsi le principe de sélection naturelle.

- **Élitisme :** Une petite fraction des solutions avec la meilleure fitness, appelées élites, est préservée d'une génération à l'autre. Cela garantit que les solutions de haute qualité ne sont pas perdues au fil des générations.
- **Convergence :** L'AG évolue sur plusieurs générations. À chaque génération, de nouvelles solutions sont créées, évaluées, et mélangées avec la population actuelle. Ce processus se répète jusqu'à ce qu'une condition de convergence soit atteinte ou après un nombre fixe de générations.
- **Recherche Locale :** Pour améliorer la qualité des solutions, des opérations de recherche locale telles que la recherche 2-opt ou 3-opt sont appliquées aux solutions générées par l'AG. Cela vise à optimiser davantage les tournées de véhicules.

L'AG pour le VRP cherche un équilibre entre l'exploration de nouvelles solutions et l'exploitation des solutions prometteuses, tout en respectant les contraintes de temps et en maximisant le profit global du convoi de véhicules.

3.3 Méthodes de recherche locale

3.3.1 2-Opt

On peut tenter d'améliorer la solution obtenue par des méthodes de recherche locale (meilleure solution parmi l'ensemble des solutions que l'on a) comme le 2-Opt. Cependant, telle quelle, la recherche locale du 2-Opt ou Or-Opt n'est dans notre cas pas utile. En effet, les noeuds restent les mêmes, et par conséquent, le profit n'évoluera pas contrairement au temps/distance des tournées. Nous devons donc adapter le 2-Opt. Pour cela, nous pouvons essayer de réduire la durée d'un chemin, puis, de voir si l'insertion de points supplémentaires est possible (Nous utiliserons le PPV). Si un tel cas se présente, nous avons réussi !
Le profit ne pourra être qu'équivalent ou meilleur que celui de chemin non optimisé.

3.3.2 3-Opt

Une fois le 2-Opt implémenté, il est facile de mettre en oeuvre le 3-Opt ou tout autre méthode d'optimisation similaire.

4 Implémentation et Étude qualitative des algorithmes

Notre choix du langage Python pour l'implémentation d'algorithmes, et en particulier des méta-heuristiques, repose sur sa simplicité syntaxique, sa lisibilité et la vaste collection de bibliothèques pour le traitement des données et l'affichage. Ces caractéristiques font de Python un choix pragmatique, accélérant le développement et la résolution des problèmes, tout en offrant une maintenance aisée du code.

Conscients que Python ne soit pas le langage optimal (en temps d'exécution) pour implémenter de tels algorithmes, nous avons avant tout voulu offrir un code clair et une multitude d'heuristiques dans l'objectif de comparer leurs performances. Le temps d'exécution n'est donc pas important, puisque l'écart de performance entre les heuristiques ne change pas d'un langage à l'autre.

Pour obtenir de meilleures performances pour une utilisation réelle, les langages C ou C++ auraient été privilégiés.

Pour ce projet, nous avons implémenté: l'heuristique de Beasley, l'heuristique de Gillett et Miller, l'Optimisation par Colonie de Fourmis ainsi que l'Algorithme Génétique.

Différentes heuristiques du PVC ont également été implémentée pour la construction d'une solution de départ pour l'heuristique de Beasley: le Plus Proche Voisin, l'heuristique de Fletcher, la Meilleure Insertion, La Plus Proche insertion ainsi que la Plus Lointaine Insertion.

Enfin, les recherches locales 2-Opt et 3-Opt ont aussi été implémentées pour l'obtention de meilleures solutions.

4.1 Heuristiques du PVC

Ces heuristiques sont disponibles dans le fichier : $src/algorithms/TSP\_heuristics.py$

Elles ont été implémentées selon leur présentation dans le cours. Cependant, nous devons légèrement adapter l'algorithme au TOP.
Puisque l'on cherche un chemin traversant tous les noeuds de l'instance, entre le point de départ et d'arrivée, nous devons de forcer les points de départ et d'arrivée.

Nous partons du cycle [départ,arrivée] pour PLI, PPI, MI en n'autorisant l'ajout de noeuds uniquement entre le départ et et l'arrivée.

Pour le PPV et Fletcher, il suffit simplement de commencer par le noeud de départ, et de terminer par le noeud d'arrivée.

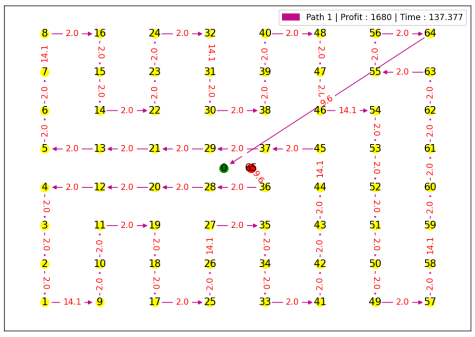
Vous trouverez sur la page suivante un exemple (instance $p.5.4.w$, voir 4.6 Résultats) d'application des heuristiques. :

4.2 Heuristique de Beasley

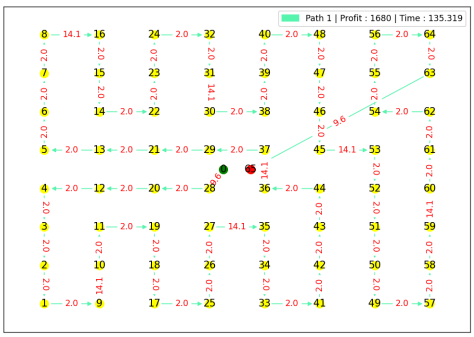
Maintenant que nous avons le chemin de départ pour notre algorithme, nous pouvons lancer l'heuristique sur celui-ci.
L'heuristique est disponible dans le fichier : $src/algorithms/beasley.py$

4.2.1 Algorithme Classique

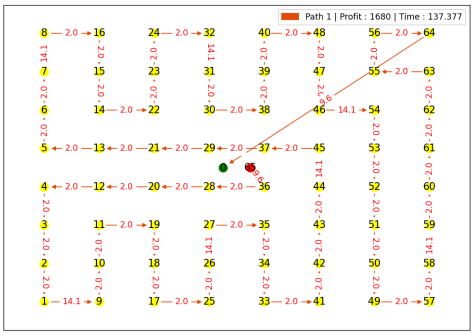
Le recherche de solution se fait par la méthode : $beasley\_top(graph : Graph, heuristic, localSearch)$. En entrée, la fonction récupère le graphe (objet Graph) représentant l'instance à étudier, l'heuristique du PVC à appliquer ainsi que la recherche locale à utiliser.



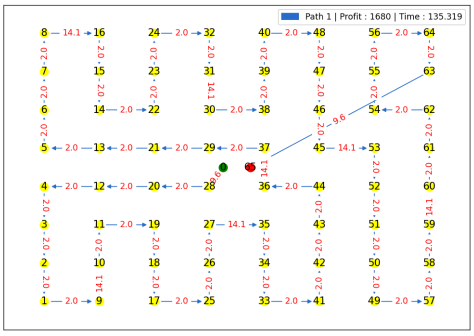
(a) Plus Proche Voisin $O(n^2)$



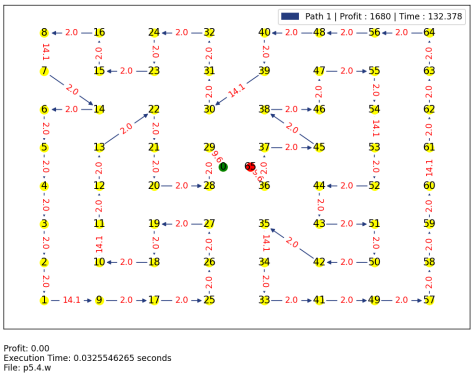
(b) Plus Lointaine Insertion $O(n^2)$



(c) Plus Proche Insertion $O(n^2)$



(d) Meilleure Insertion $O(n^2)$



(e) Heuristique de Fletcher $O(n^4)$

Figure 1: Différentes heuristiques

- Construction d'un chemin passant par tous les noeuds (heuristique PVC)
- Construction du graphe auxiliaire
- Génération d'une solution compatible au problème (méthode *gen_conv*())
- Recherche locale effectuée sur chaque tournée de la solution, afin de maximiser les profits

Complexité de l'algorithme implémenté : $O(n^4)$ avec 2-Opt (2-Opt simple avec boucle while)

4.2.2 Algorithme de recherche multiple

Une seconde version de l'algorithme est proposée par la méthode *beasley_top_optimized*(*graph* : *Graph*). Cette méthode teste toutes les heuristiques du PVC et de recherche locale, ainsi que l'inversion des noeuds de départ et d'arrivée (autorisé car le graphe n'est pas orienté). Cette méthode retourne la meilleure solution trouvée.

4.3 Heuristique de Gillett et Miller

Cette heuristique est disponible dans le fichier : *src/algorithms/gillettMiller.py*

L'algorithme de cette heuristique reprend le format de l'implémentation de celle de Beasley.

4.3.1 Algorithme Classique

Le recherche de solution se fait par la méthode : *gillett_miller_top*(*graph*, *localSearch*) . En entrée, la fonction récupère le graphe (objet Graph) représentant l'instance à étudier, ainsi que la recherche locale à utiliser.

- Tri des noeuds par angle polaire croissant
- Pour chaque noeud de ce tableau, on recherche des tournées possibles dont la recherche commence par ce noeud
- Génération d'une solution compatible au problème (méthode *gen_conv*())
- Recherche locale effectuée sur chaque tournée de la solution, afin de maximiser les profits
- Si la solution est meilleure que celle enregistrée, on l'enregistre

Remarque : chercher une solution pour chaque noeud permet d'augmenter les chances d'obtenir une meilleure solution. Bien évidemment, cela a un impact considérable sur la complexité de la solution retenue.
Complexité de l'algorithme implémenté : $O(n^5)$ avec 2-Opt (2-Opt simple avec boucle while)

4.3.2 Algorithme de recherche multiple

De la même manière que pour l'heuristique de Beasley, une seconde version de l'algorithme est proposée par la méthode *gillett_miller_top_optimized*. Cette méthode teste toutes les heuristiques du PVC et de recherche locale, ainsi que l'inversion des noeuds de départ et d'arrivée (autorisé car le graphe n'est pas orienté). Cette méthode retourne la meilleure solution trouvée.

4.4 Optimisation par colonie de fourmis

Cette heuristique est disponible dans le fichier : *src/algorithms/aco.py*

4.4.1 Objectif

Bien que les heuristiques implémentées auparavant fournissent une solution, nous voulions obtenir des résultats se rapprochant plus de la solution optimale. Nos recherches nous ont conduits à l'algorithme des colonies de fourmis (ACO) qui semblaient être un bon compris entre performance et temps de calcul (pour rappel, nous implémentons en Python). Nous avons donc fait le choix d'implémenter celui présenté par Liangjun Ke, Claudia Archetti et Zuren Feng (2008).

Cet algorithme est le suivant :

Initialize the parameters, heuristic information η and the
pheromone trails τ
$s_{ib} \leftarrow Null$
$s_{gb} \leftarrow Null$
$N_{ni} \leftarrow 0$
iteration $\leftarrow 0$
while *iteration* is less than N_C do
  for $i = 1$ to n_a do (n_a is the number of ants)
    $s_i \leftarrow$ **ConstructSolution**(τ, η) (see section 3.2)
    $s_i \leftarrow$ **LocalSearch**(s_i) (see section 3.4)
  end for
  $s_{ib} \leftarrow \arg\max(F(s_1), F(s_2), \cdots, F(s_{n_a}))$
  if $F(s_{ib}) > F(s_{gb})$
    $s_{gb} \leftarrow s_{ib}$
    $N_{ni} \leftarrow 0$
  else
    $N_{ni} \leftarrow N_{ni} + 1$
  end if
  PheromoneUpdate(τ, s_{ib}, s_{gb}, N_{ni}) (see section 3.3)
  iteration $\leftarrow$ *iteration* + 1
end while

Figure 2: ACO proposé par L.Ke, C.Archetti et Z.Feng

- F correspond à la fonction de qualité d'une solution
- s_{ib} correspond à la meilleure solution de l'itération courante
- s_{gb} correspond à la meilleure solution de trouvée pour le moment
- N_{ni} correspond au nombre d'itération effectuées pour le renouvellement des phéromones

Pour rappel, voici les principes utilisés par l'algorithme :

- **Construction de Solutions** : Les fourmis construisent des solutions candidates en suivant des chemins dans l'espace des solutions. Chaque fourmi prend des décisions locales basées sur des informations heuristiques et des traces de phéromones.

- **Phéromones** : Les fourmis déposent une substance chimique appelée phéromone le long de leur chemin. La concentration de phéromones influence le choix des fourmis. Les chemins avec des concentrations plus élevées sont plus attrayants.
- **Évaporation des Phéromones** : Les traces de phéromones s'évaporent avec le temps, évitant que les fourmis ne se fixent trop sur un chemin spécifique. Cela permet une exploration continue de l'espace des solutions.
- **Choix de Chemins** : Les fourmis choisissent leurs chemins en utilisant une combinaison d'informations heuristiques et de la quantité de phéromones. Ce choix est représenté par un tirage alétoire d'un noeud, dont la probabilité de tirage est calculée par une formule. Cette formule prend en compte l'importance relative de la phéromone par rapport à une heuristique (par exemple : la distance entre le noeud de d'arrivée et le noeud tiré).
- **Exploration et Exploitation** : L'ACO équilibre l'exploration (en essayant de nouveaux chemins) et l'exploitation (se concentrant sur les chemins prometteurs). Cela permet de trouver des solutions satisfaisantes, voire optimales, tout en évitant de rester piégé dans des optima locaux.

Nous avons essayé de reproduire à l'identique les méthodes décrites dans l'article. Nous avons également pu ré-utiliser l'algorithme du 2-Opt que nous avions déjà implémenté, puisqu'il se comporte identiquement à celui utilisé.

Complexité de l'algorithme implémenté : $O(n\_iterations * n\_ants * n^4)$

4.4.2 Paramètres

L'implémentation utilisée pour les tests présentés dans l'article est effectuée en C++. Pour 20 fourmis, le nombre d'itération est fixé à 2000. Ces valeurs sont trop élevées pour notre implémentation et induirait de nombreuses dizaines de minutes de calcul. Nous avons donc fait le choix d'ajuster les paramètres heuristiques.

Afin de réduire le temps de calcul, mais d'obtenir tout de même des solutions satisfaisantes, nous avons ajusté le nombre d'itérations à 20, et le nombre de fourmis à 10. Le temps de calcul est alors considérablement réduit. Cependant, cette réduction a un impact fort sur le dépôt de phéromones. Nous avons donc augmenter le taux d'importance accordé à l'exploration (β) et réduit l'importance de l'information heuristique de θ_{ij} (dans γ) afin de considérer davantage le gain plutôt que la distance entre les noeuds.

Nos différents tests ont montré que les valeurs des variables suivantes permettent d'obtenir de bons résultats :

- **n_ants** : int = 10 : Le nombre de fourmis de la colonie
- **n_iterations** : int = 20 : Le nombre d'itérations/de voyage d'une fourmi
- **Ncycles** : int = 3 : Le seuil de réinitialisation des phéromones
- α : float = 1 : Importante des phéromones (coefficient)
- β : float = 30 : Importance de l'information heuristique (coefficient)
- γ : float = 0.05 : Influence de l'angle pour l'information heuristique (coefficient)
- **evaporation_value** : float = 0.98 : La variable pour calculer le taux d'évaporation
- P_{best}: float = 0.05 : Probabilité de construire la meilleure solution quand les phéromones ont convergée vers Tmin ou Tmax

Nous avons ainsi pu obtenir des solutions pouvant être moins bonnes que celles promises par l'article, mais (parfois) bien meilleures que celles trouvées par les heuristiques précédentes. De plus, les temps de calcul sont en moyenne 2 à 3 fois plus courts (et jusqu'à 10 fois) que ceux présentés dans l'article, pour les mêmes instances de test. Cela laisse penser que cette adaptation est beaucoup plus rapide en C++/C, et que notre objectif est atteint.

4.5 Algorithme génétique

Cette heuristique est disponible dans le fichier : *src/algorithms/genetic.py*

4.5.1 Objectif

L'objectif de l'implémentation de cet algorithme était de tester et expérimenter la théorie de l'évolution naturelle appliquée au Problème de Routage de Véhicules (VRP). L'objectif final était d'obtenir un meilleur rapport entre le profit total et le temps d'exécution.

4.5.2 Fonctionnement de l'Algorithme Génétique

- **Initialisation de la Population** : L'algorithme génétique débute en créant une population initiale de plans de livraison. Ces plans représentent différentes manières d'organiser la livraison pour un véhicule donné. Par exemple, une population pourrait être constituée de chemins tels que [[1,6,3,9,12,32,50], [1,32,12,42,50], [1,4,31,21,44,50], ...]. Chaque chemin doit respecter certaines contraintes, notamment le respect du temps total ne dépassant pas Tmax, l'absence de nœuds redondants dans le même chemin, et le début du chemin au nœud de départ et la fin au nœud d'arrivée. De plus, pour optimiser la recherche, les couples de nœuds dépassant Tmax ont été écartés.

- **Évaluation de la Fitness** : Chaque plan de livraison est évalué en fonction de sa qualité, mesurée par une fonction de fitness. Cette fonction calcule le profit total d'un chemin donné, par exemple : [1,32,31,44,55,60]. Si le chemin dépasse Tmax, la fonction retourne 0 pour signaler qu'il est défectueux, sinon elle calcule normalement le profit entre les nœuds.

- **Sélection des Élites** : Les plans les mieux évalués, appelés élites, sont préservés pour garantir la conservation de solutions de haute qualité au fil des générations.

- **Croisement (Crossover)** : L'algorithme génétique utilise l'opérateur de croisement AEX (Alternating Edges Crossover) pour combiner des caractéristiques de deux plans de livraison (parents) et créer de nouveaux plans (enfants). AEX interprète un chromosome comme un cycle dirigé d'arcs, et les arcs du parent sont choisis alternativement pour former le cycle enfant.

  Par exemple, supposons que les deux parents soient p1 = (5 1 7 8 4 9 6 2 3) et p2 = (3 6 2 5 1 9 8 4 7).

  La procédure commence par choisir l'arc 5 $\rightarrow$ 1 de p1 comme premier arc. Ainsi, l'enfant est initialisé comme c = (5 1 *******).

  Ensuite, l'arc de p2 sortant de 1 est ajouté, c'est-à-dire 1 $\rightarrow$ 9. Ainsi, l'enfant devient c = (5 1 9 ******).

  Ensuite, l'arc de p1 sortant de 9 est ajouté, etc. Après quelques étapes, l'enfant partiellement formé est le suivant : c = (5 1 9 6 2 3 * * *).

  L'arc sortant de 3 devrait être choisi à partir de p2, mais un tel choix est infaisable car il fermerait le cercle trop tôt.

  Pour éviter cette situation, l'un des sommets restants non visités est choisi au hasard, par exemple 7. Ainsi, l'enfant devient c = (5196237 * *).

  À partir de ce point, la procédure ordinaire peut être reprise en choisissant par exemple l'arc 7 $\rightarrow$ 8 de p1, puis 8 $\rightarrow$ 4 de p2. L'enfant complet ressemble alors à c = (519623784).

- **Mutation** : Certains plans de livraison subissent des mutations aléatoires, introduisant ainsi une diversité dans la population. Ces mutations peuvent inclure des ajustements tels que l'inversion de

l'ordre des arrêts ou l'échange entre véhicules.

Cette fonction prend en entrée un chemin, une probabilité de mutation, et éventuellement un opérateur de mutation spécifique. Si la probabilité générée aléatoirement est inférieure à la probabilité de mutation et que la longueur du chemin est suffisante, une mutation est appliquée en utilisant un opérateur spécifique ou choisi aléatoirement parmi cinq options (bit flip, random resetting, swap, scramble, inversion). Chaque opérateur modifie le chemin d'une manière particulière, comme inverser l'ordre des gènes, échanger des positions, ou changer aléatoirement certaines valeurs. Le chemin résultant est ensuite validé pour garantir qu'il commence et se termine par des nœuds spécifiques, avant d'être renvoyé.

- **Optimisation Locale** : À chaque génération, une optimisation locale est appliquée pour affiner les trajets des véhicules et améliorer la qualité des solutions.
- **Nouvelle Génération** : La nouvelle population est formée en combinant les élites, les descendants du croisement, et les individus mutants. Ce processus est répété sur plusieurs générations.

4.5.3 Paramètres de Configuration

- **Taille de la Population** : Le nombre d'individus dans la population. Une taille plus grande offre une exploration plus approfondie, mais avec un coût computationnel accru.
- **Nombre de Générations** : Le nombre d'itérations de l'algorithme. Un nombre plus élevé peut permettre une convergence vers des solutions optimales, mais avec un temps d'exécution plus long.
- **Taux d'Élitisme** : La proportion des meilleurs individus préservés à chaque génération pour favoriser la stabilité des solutions de haute qualité.
- **Probabilité de Mutation** : La chance qu'un individu subisse une mutation. Une probabilité plus élevée encourage l'exploration de nouvelles solutions.
- **Opérateurs de Mutation** : Différents types de mutations influençant la diversité génétique de la population.

À la fin de la génération de N itérations, une vaste population de solutions émerge. L'étape suivante consiste à sélectionner les meilleurs chemins parmi cette population, ceux qui maximisent le profit tout en respectant les contraintes du problème. Pour cette tâche, nous avons développé la fonction gen_conv. Initialement, nous créons un dictionnaire appelé nodes_paths, attribuant à chaque nœud du graphe les chemins qui le traversent, accompagnés de leur profit respectif. Ensuite, nous constituons une liste nommée best_solutions, qui parcourt ce dictionnaire pour enregistrer le meilleur chemin associé à chaque nœud, en termes de profit. Enfin, nous itérons à travers best_solutions pour sélectionner les meilleurs nbVehicules chemins, en veillant à ce qu'ils présentent des nœuds uniques pour chaque trajet.

L'algorithme génétique a une complexité temporelle approximative de $O(\textit{générations} * \textit{taille\_population} * (\textit{nombre\_de\_nœuds})^2)$. Il utilise des opérations d'initialisation, de sélection, de croisement, de mutation et de réparation pour évoluer vers une solution optimale respectant les contraintes de temps et maximisant le profit.

4.6 Résultats

Fichier	ProfitGM	TempsGM	ProfitBeasley	TempsBeasley	ProfitACO	TempsACO	ProfitAG	TempsAG
p1.2.b.txt	**15**	0.026	**15**	0.16	**15**	0.027	**15**	9.98
p1.2.c.txt	**20**	0.043	**20**	0.15	**20**	0.041	15	0.00
p1.2.d.txt	**30**	0.064	25	0.17	25	0.067	25	0.47
p1.2.e.txt	40	0.048	45	0.16	45	0.157	**65**	0.49
p1.2.f.txt	65	0.083	70	0.17	**80**	0.212	65	0.46
p1.2.g.txt	75	0.106	**90**	0.17	85	0.358	80	0.44
p1.2.h.txt	**110**	0.180	100	0.18	**110**	0.389	100	0.38
p1.2.i.txt	120	0.257	120	0.19	120	0.448	**135**	0.36
p1.2.j.txt	125	0.241	**140**	0.19	125	0.454	135	0.41
p1.2.k.txt	140	0.265	155	0.20	**160**	0.468	155	0.43
p1.2.l.txt	155	0.300	**175**	0.21	**175**	0.494	150	0.39
p1.2.m.txt	175	0.387	180	0.22	**200**	0.526	170	0.46
p1.2.n.txt	190	0.400	205	0.23	**225**	0.534	185	0.45
p1.2.o.txt	190	0.413	210	0.24	**235**	0.562	195	0.45
p1.2.p.txt	190	0.432	215	0.24	**235**	0.571	205	0.43
p1.2.q.txt	220	0.491	245	0.26	**255**	0.549	235	0.55
p1.2.r.txt	225	0.537	220	0.26	**265**	0.546	140	0.60
p1.3.c.txt	**15**	0.026	**15**	0.17	**15**	0.029	**15**	9.10
p1.3.d.txt	**15**	0.026	**15**	0.15	**15**	0.029	**15**	9.22
p1.3.e.txt	**30**	0.054	**30**	0.16	**30**	0.058	25	0.00
p1.3.f.txt	**40**	0.087	**40**	0.16	30	0.084	35	1.40
p1.3.g.txt	**50**	0.055	**50**	0.16	**50**	0.159	45	1.41
p1.3.h.txt	70	0.103	65	0.16	70	0.252	**90**	1.32
p1.3.i.txt	85	0.102	95	0.17	100	0.400	**105**	1.36
p1.3.j.txt	100	0.137	100	0.17	**105**	0.443	90	1.34
p1.3.k.txt	120	0.136	**125**	0.18	120	0.451	**125**	1.24
p1.3.l.txt	**150**	0.214	145	0.18	140	0.526	**150**	1.08
p1.3.m.txt	165	0.212	160	0.19	155	0.619	**170**	1.01
p1.3.n.txt	170	0.238	180	0.19	170	0.551	**185**	1.07
p1.3.o.txt	175	0.244	180	0.19	175	0.562	**190**	1.07
p1.3.p.txt	180	0.246	**195**	0.19	190	0.560	180	1.10
p1.3.q.txt	200	0.265	**220**	0.20	210	0.569	200	0.97
p1.3.r.txt	205	0.278	230	0.20	**235**	0.590	205	1.04
p1.4.d.txt	**15**	0.026	**15**	0.15	**15**	0.032	**15**	7.58
p1.4.e.txt	**15**	0.026	**15**	0.15	**15**	0.032	**15**	9.96
p1.4.f.txt	**25**	0.048	**25**	0.16	**25**	0.049	**25**	0.01
p1.4.g.txt	**35**	0.064	**35**	0.16	**35**	0.080	30	0.00
p1.4.h.txt	40	0.079	**45**	0.16	40	0.099	40	3.29
p1.4.i.txt	55	0.089	**60**	0.16	**60**	0.178	55	3.16
p1.4.j.txt	70	0.075	70	0.16	75	0.247	**105**	3.18
p1.4.k.txt	90	0.121	90	0.17	100	0.292	**115**	2.90
p1.4.l.txt	105	0.120	115	0.18	120	0.308	**125**	3.00
p1.4.m.txt	120	0.121	125	0.18	120	0.522	**130**	2.89
p1.4.n.txt	**150**	0.136	135	0.18	135	0.545	140	2.77
p1.4.o.txt	**155**	0.155	**155**	0.18	145	0.530	**155**	2.69
p1.4.p.txt	165	0.183	160	0.18	**165**	0.517	160	2.49
p1.4.q.txt	175	0.213	170	0.18	180	0.612	**195**	2.38
p1.4.r.txt	190	0.205	**200**	0.19	**200**	0.663	**200**	2.07
p5.2.b.txt	20	0.266	20	1.48	20	0.098	**25**	0.00
p5.2.c.txt	40	0.286	45	1.47	45	0.180	**55**	0.00
p5.2.d.txt	65	0.361	80	1.49	75	0.248	**85**	0.70

p5.2.e.txt	120	0.463	160	1.48	**180**	0.580	**180**	0.50
p5.2.f.txt	170	0.646	210	1.69	**240**	0.880	230	0.48
p5.2.g.txt	310	1.282	310	1.53	315	1.473	**325**	0.45
p5.2.h.txt	370	1.368	370	1.59	**395**	1.729	385	0.43
p5.2.i.txt	405	1.589	400	1.60	**455**	1.690	450	0.46
p5.2.j.txt	445	1.804	540	1.63	**580**	1.837	490	0.46
p5.2.k.txt	495	2.281	600	1.70	**670**	2.058	565	0.49
p5.2.l.txt	500	2.565	665	1.75	**800**	2.232	680	0.58
p5.2.m.txt	570	3.046	735	1.99	**845**	2.030	705	0.57
p5.2.n.txt	715	3.866	840	2.00	**920**	2.336	835	0.58
p5.2.o.txt	715	4.029	880	1.90	**1010**	2.467	865	0.66
p5.2.p.txt	715	4.233	945	1.96	**1150**	2.562	950	0.71
p5.2.q.txt	715	4.640	1000	2.49	**1180**	2.541	975	0.74
p5.2.r.txt	815	5.309	1090	2.17	**1245**	2.627	1095	0.92
p5.2.s.txt	840	6.726	1160	2.22	**1295**	2.709	1120	0.89
p5.2.t.txt	870	6.245	1185	2.33	**1335**	2.834	1180	0.91
p5.2.u.txt	995	7.801	1285	2.55	**1405**	2.720	1310	0.99
p5.2.v.txt	995	8.251	1370	2.81	**1490**	2.883	1385	1.08
p5.2.w.txt	1020	8.624	1425	2.72	**1530**	3.022	1440	1.05
p5.2.x.txt	1055	9.139	1455	2.75	**1580**	2.934	1475	1.09
p5.2.y.txt	1195	10.828	1535	2.89	**1615**	2.870	1525	1.16
p5.2.z.txt	1195	11.259	1630	3.05	1640	2.876	**1655**	1.30
p5.3.b.txt	**15**	0.165	**15**	2.38	**15**	0.20	**15**	0.00
p5.3.c.txt	20	0.239	20	1.46	20	0.107	**25**	0.00
p5.3.d.txt	55	0.331	**60**	1.65	**60**	0.231	45	0.00
p5.3.e.txt	70	0.405	90	1.85	95	0.245	**100**	0.00
p5.3.f.txt	95	0.474	110	1.51	110	0.319	**115**	1.53
p5.3.g.txt	165	0.634	175	1.54	185	0.756	**220**	1.39
p5.3.h.txt	195	0.651	220	1.54	260	0.794	**270**	1.37
p5.3.i.txt	270	0.879	270	1.53	**335**	1.218	325	1.31
p5.3.j.txt	460	1.384	405	1.56	**465**	1.562	425	1.27
p5.3.k.txt	465	1.662	495	1.55	475	1.974	**495**	1.19
p5.3.l.txt	555	1.722	540	1.61	**580**	2.325	565	1.15
p5.3.m.txt	555	1.867	580	1.66	**645**	2.347	640	1.18
p5.3.n.txt	590	1.976	635	1.65	**730**	2.376	695	1.04
p5.3.o.txt	665	2.404	790	1.68	**870**	2.382	750	1.15
p5.3.p.txt	740	2.698	865	1.75	**990**	2.502	825	1.10
p5.3.q.txt	745	2.672	910	1.72	**1065**	2.613	915	1.24
p5.3.r.txt	745	2.979	1030	1.82	**1110**	2.969	970	1.34
p5.3.s.txt	845	3.150	1030	1.82	**1165**	2.641	1010	1.32
p5.3.t.txt	855	3.336	1130	1.95	**1235**	2.720	1130	1.41
p5.3.u.txt	1025	4.116	1240	1.86	**1295**	2.670	1195	1.50
p5.3.v.txt	1070	4.384	1305	1.95	**1375**	2.675	1275	1.54
p5.3.w.txt	1070	4.498	1375	1.96	**1465**	2.700	1365	1.60
p5.3.x.txt	1070	4.749	1400	2.03	**1490**	2.932	1400	1.78
p5.3.y.txt	1070	4.901	1465	2.08	**1560**	2.626	1440	1.86
p5.3.z.txt	1145	5.277	1470	2.26	**1585**	2.541	1470	1.73
p5.4.c.txt	**20**	0.215	**20**	1.46	**20**	0.127	**20**	0.00
p5.4.d.txt	20	0.215	20	1.47	20	0.115	**25**	0.00
p5.4.e.txt	20	0.216	20	1.44	20	0.115	**25**	0.00
p5.4.f.txt	80	0.542	80	1.46	80	0.278	**85**	0.00
p5.4.g.txt	80	0.498	125	1.48	140	0.366	**160**	0.00
p5.4.h.txt	115	0.601	140	1.52	140	0.366	**175**	3.25
p5.4.i.txt	190	0.859	190	1.53	235	0.673	**245**	3.17

p5.4.j.txt	240	0.854	300	1.53	**340**	0.928	335	3.13
p5.4.k.txt	260	0.822	295	1.72	340	0.941	**350**	2.94
p5.4.l.txt	340	1.113	345	1.53	**425**	1.480	410	2.85
p5.4.m.txt	500	1.260	495	1.57	**535**	1.949	505	2.47
p5.4.n.txt	615	1.908	620	1.61	**620**	2.446	615	2.21
p5.4.o.txt	620	1.930	**670**	1.62	640	2.446	665	2.07
p5.4.p.txt	725	1.999	710	1.60	**745**	2.786	735	2.49
p5.4.q.txt	710	2.130	725	1.63	800	2.828	**810**	2.11
p5.4.r.txt	775	2.231	770	1.60	**875**	2.749	860	1.84
p5.4.s.txt	775	2.319	900	1.65	**995**	3.178	900	1.80
p5.4.t.txt	885	2.491	1000	1.81	**1160**	2.795	1000	2.33
p5.4.u.txt	970	3.009	1035	1.68	**1285**	2.765	1050	2.29
p5.4.v.txt	975	2.974	1140	1.75	**1300**	2.835	1150	2.21
p5.4.w.txt	990	3.118	1215	1.74	**1350**	2.857	1240	2.58
p5.4.x.txt	990	3.240	1280	1.86	**1420**	2.792	1280	2.54
p5.4.y.txt	1100	3.486	1250	1.86	**1450**	2.788	970	2.60
p5.4.z.txt	1125	3.546	1415	1.84	**1485**	5.35	1390	2.73

Remarque : Utilisation de la fonction *beasley_top_optimized* (toutes les heuristiques appliquées sauf Fletcher) et de *gillett_miller_top*

D'après nos résultats, nous pouvons émettre l'hypothèse que l'ACO est meilleure sur des instances où le Tmax est important, mais l'AG plus performant sur les Tmax moyen. Sur des Tmax faibles, les heuristiques de Beasley et de Gillett et Miller semblent plus adaptées car plus rapides en execution, tout en garantissant généralement un résultat optimal par rapport aux autres algorithmes.

- Les instances utilisées se trouvent ici

5 Conclusion

En résumé, chaque méthode abordée présente ses avantages et inconvénients, en particulier en fonction de la taille des instances et des contraintes temporelles. Certains fichiers, comme Set_32_234/p1.4.a.txt avec un Tmax de seulement 1.4 secondes, ont posé des défis difficiles à surmonter dans le temps imparti. Le choix de la meilleure approche dépend ainsi des contraintes spécifiques du problème, incluant le temps de calcul disponible et la qualité de la solution requise. Les résultats obtenus mettent en lumière la diversité des techniques heuristiques et de recherche locale (2-opt & 3-opt). Notamment, des méthodes telles que l'algorithme de la colonie de fourmis ont démontré une proximité avec la solution optimale. Cette exploration a enrichi nos connaissances, révélant de nouveaux algorithmes et des approches innovantes pour optimiser les problèmes, soulignant l'importance de la collaboration en équipe pour repousser les limites de la résolution de problèmes complexes tels que le Problème de Routage de Véhicules (VRP).

6 Bibliographie

- Route first—Cluster second methods for vehicle routing, JE Beasley, 1983
- Ants can solve the team orienteering problem, Liangjun Ke, Claudia Archetti, Zuren Feng, 2008
- Algo génétique : crossover
- Algo génétique : mutation