UNIVERSITÉ SORBONNE PARIS NORD
SUP GALILÉE

RAPPORT SAÉ DE L'UE OPTIMISATION

deuxième année cycle ingénieur
spécialité "Informatique"
par

Sanaa KOURICHI
Killian VAN SEUNINGEN

AUTOUR DU TRACÉ D'UN MÉTRO CIRCULAIRE

Rendu le 30 janvier 2026

Enseignants :

Lucas LÉTOCART	professeur d'optimisation combinatoire
Nabil MUSTAFA	professeur de complexité
Pierre FOUILLOUX	professeur d'optimisation linéaire

Nous, soussignés KOURICHI Sanaa et VAN SEUNINGEN Killian, étudiants en deuxième année d'école d'ingénieur à Sup Galilée, déclarons être pleinement conscient que la copie de tout ou partie d'un document, quel qu'il soit, publié sur tout support existant, y compris sur Internet, constitue une violation du droit d'auteur ainsi qu'une fraude caractérisée, tout comme l'utilisation d'outils d'Intelligence Artificielle non-déclarée pour générer une partie de ce rapport ou du code associé. En conséquence, nous déclarons que ce travail ne comporte aucun plagiat, et assurons avoir cité explicitement, à chaque fois que nous en avons fait usage, toutes les sources utilisées pour le rédiger.

Fait à Villetaneuse, le 30/01/2026

Signatures :

KOURICHI Sanaâ VAN SEUNINGEN Killian

TABLE DES MATIÈRES

Liste des figures

Liste des algorithme

1 Introduction

Dans le contexte de nos études en deuxième année cycle ingénieur à l'école Sup Galilée. Dans le cadre de l'UE optimisation composée des matière d' optimisation combinatoire , optimisation linéaire et complexité , il nous a été proposé d'étudier le problème d'anneau de l'étoile . Ce projet a pour objectif de tracer une ligne de métro circulaire . Le but étant de mettre en pratique les méthodes d'optimisation vues en cours.

Ce rapport présente le résolution du problème de l'anneau-étoile (Ring-Star Problem) par la mise en œuvre d'une heuristique , d'une métaheuristique et d'une résolution exacte de la formulation compacte.

Le problème a pour but de déterminer des stations parmi n noeuds et d'affecter les autres nœuds à la station la plus proche, puis, de relier toutes les stations de façon cyclique en minimisant la somme des distance entre stations, et celle entre les stations et leur nœud.

Nous allons découper ce problème en deux problèmes distincts : le p médian et le TSP. Dans un premier temps nous cherchons une solution au problème de p médian pour choisir des stations (les médians), et en répartissant les autres points. Dans un second temps nous effectuons un TSP sur l'ensemble des médians donné par la solution obtenue pour p median . Pour cela nous avons développé une heuristique et une métaheuristique .

Dans ce rapport nous montrerons que le problème de l'anneau étoile est NP-difficile en réduisant le TSP. Puis nous présenterons une heuristique ainsi qu'une métaheuristique . Et pour finir nous effectuerons une analyse comparative.

Pour tout le test nous n'utilisons que des données en distance euclidienne 2D.

2) NP difficulté

2.1 Rappel du problème

Le problème de l'anneau de étoile ou Ring star problem a pour objectif de trouver un sous graphe à coût minimum contenant un cycle (l'anneau) et un ensemble d'arcs

(étoile) tel que chaque nœud appartenant au cycle se voit attribuer les points non-stations dont il est le plus proche . On cherche a minimiser la fonction suivante :

$$Min\ \alpha \sum_{ij \in E} d_{ij} x_{ij} + \sum_{(i,j) \in V \times V} d_{ij} y_{ij}$$

2.2 Rappel du TSP

Le TSP ou Voyageur de commerce est un problème qui consiste à trouver un cycle passant par tous les sommets exactement une fois et de coût minimal. Ce problème est un problème NP-complet car on peut réduire polynomialement le problème de Cycle hamiltonien [1].

2.3 Preuve

Théorème 1. Le problème de l'anneau étoile est NP-difficile.

Preuve 1.

Montrons que l'on peut réduire en un temps polynomial le TSP. C'est-à- dire que le problème d'anneau étoile peut nous dire si le TSP a une solution ou non.

Le TSP cherche une tournée de coût inférieur ou égal à k.

Soit un graph G = (V, E) avec n sommets et m arêtes . Pour l'arête entre le sommet i et j on note cij le coût sur cette arête.

Soit le graph G' = (V' , E') avec les sommets et les arêtes de G . Nous ajoutons toutes les arêtes nécessaires pour que G' soit complet . Chaque arête on ajoute un poids de k+1.

Nous appliquons l'anneau de l'étoile sur G' avec un nombre de station n et les coûts entre chaque sommets sont considérés comme leur la distance euclidienne entre ces derniers.

Sachant que p=n toute les nœuds sont des stations. Par conséquent on a :

$$Coût\ total\ = \sum_{ij \in E} d_{ij} x_{ij}$$

D’une part si l’anneau de l'étoile trouve un cycle de coût inférieur ou égal a k alors il existe un TSP de coût k . De même si l’anneau de l'étoile ne trouve pas de circuit de coût inférieur ou égal a k alors il n'existe pas de TSP .

En effet, l’anneau choisira le circuit de coût le plus petit par définition .

Si le coût est supérieur a k alors soit dans les arêtes de G il n’existe pas de cycle plus petit donc pas de cycle inférieur a k soit il a choisi une arête ajouter dans ce cas cela veut dire que tous les chemins parcourant les arêtes de G sont supérieurs à k.
Soit trouver un chemin qui passe par tous les sommets avec un coût inférieur à k.

2.4 Conclusion

Nous avons montré qu’il existe une réduction polynomiale du problème de TSP vers le problème de l’anneau étoile. Donc le problème de l’anneau-étoile est NP- difficile

3 Heuristiques

Dans cette partie nous présenterons les heuristiques qui nous permettront de trouver une solution réalisable et optimale localement.

3.1 P-median

Le problème de p médian a pour but de sélectionner p noeuds appelés les médians de manière à minimiser la distance euclidienne entre les sommets et leur médian le plus proche.

3.1.1 Approche théorique

Trois heuristiques ont été implémentés pour ce problème.

3.1.1.1 Heuristique randomisée

Cette heuristique est la plus simple. Nous décidons de choisir de façon aléatoire et uniforme parmi tous les points du problème . A la suite de quoi nous affectons les points à leur médian le plus proche. La qualité de cette heuristique varie selon le tirage.

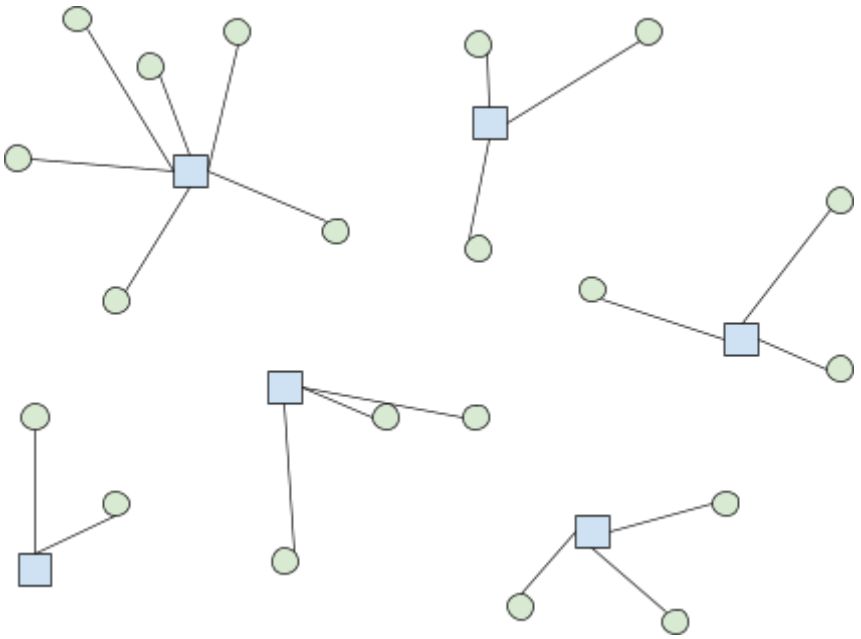

Figure 1 : Illustration du p médian

3.1.1.2 Heuristique rectangle

Pour cette heuristique on découpe le plan en $q \times q$ rectangle. Avec $q = \lceil\sqrt{p}\rceil$.

Puis nous cherchons le point le plus proche du centre de chaque rectangle que l'on choisit comme median . En effet, un rectangle peut être vide. Pour tout rectangle vide on sélectionne le point le plus proche du centre du rectangle. Cette méthode permet une meilleure distribution spatiale des médians.
La qualité de cette heuristique dépend de la répartition des points dans l'espace.

3.1.1.3 Heuristique hybride avec probabilité

Nous avons proposé une approche combinant les deux heuristiques précédentes.
Nous avons fait le choix d'une heuristique qui regroupe les deux précédentes. Initialement nous découpons le plan comme pour la méthode des rectangles (3.1.1.2). Dans chaque rectangle nous choisissons un nœud au hasard. Nous affectons les points au médian . Nous obtenons donc une solution réalisable. Pour améliorer cette dernière, nous avons décidé d'introduire la notion de probabilité. En effet, nous générons des solutions sur le même principe puis calculons le résultat de la fonction optimale. Tous les résultats sont comparés. Les points qui apparaissent fréquemment dans les meilleures solutions sur les p voient leur probabilité d'être tiré augmenter, à l'inverse les points qui

apparaissent souvent dans les mauvaises solutions ont de moins en moins de chances d'être pris.

3.1.2 Algorithme

Algorithme 1: Algorithme de l'heuristique hybride

Entrées : p : nombre de médians
X : la liste des abscisse
Y: la liste des ordonnées
D : Matrice des distances entre chaque point

Sortie : M : matrice qui sur les diagonale contient les median et pour chaque (i,j)associe a i sont median j
C : la liste des indices des médians dans le matrice M .

Initialisation :
M ⟵ [ ]
C ⟵ [ ]
probabilités : liste de taille taille de X dont tout les terme sont égaux 10p
Dictionnaire ⟵ ∅
q ⟵ $\lceil\sqrt{p}\rceil$
listeAleatoire ⟵ [ ]

Début :
Pour itération = 1 à max_iter **faire**
C ⟵ [ ]
Pour k dans rectangle R **faire**
si (k ∉ C) **alors**
| ajouter k dans listeAleatoire probabilité[k] fois
si listeAleatoire ≠ ∅ **alors**
m ⟵ random dans listeAleatoire
ajoute m a C
Si C est trop grand **alors** on choisi p élément au hasard
M ⟵ Affecte au point leur médian le plus proche

S ⟵ $\sum_{(i,j)\in V\times V} d_{ij} y_{ij}$
Dictionnaire [C] ⟵ S
C_meilleur ⟵ min Dictionnaire selon S
C_pire ⟵ max Dictionnaire selon S
Pour i ∉ C_meilleur ∩ C_pire **faire**
Si i ∈ C_meilleur **alors**
| probabilité[i] + = 1
Sinon

probabilité[i] - = 1

3.1.3 Affichage

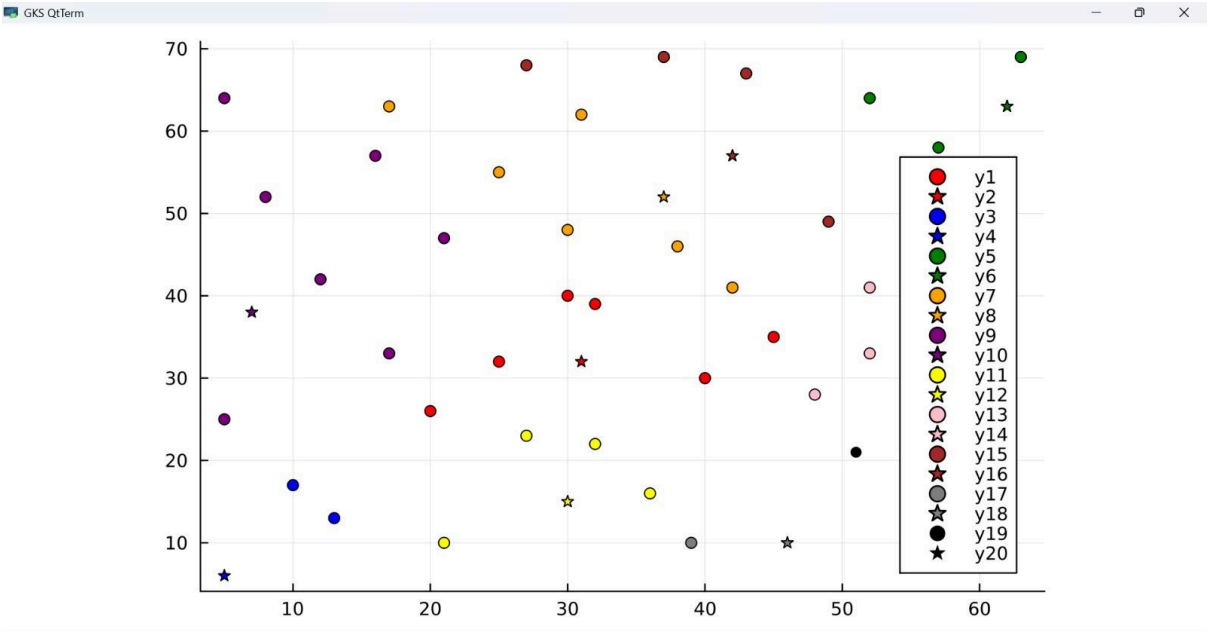

Figure 2 : affichage de l'algorithme 1

3.1.4 Conclusion

L'heuristique hybride permet de trouver une solution réalisable en un temps limité. L'heuristique hybride est plus stable que l'approche aléatoire et que la méthode des rectangles.

3.2 TSP

Nous avons considéré une heuristique d'améliorations locals de type k-OPT, dont la plus performante est Lin-Kernighan

3.2.1 Approche théorique

3.2.1.1 Heuristique du plus proche voisin

On commence par prendre le sommet 1 . A chaque itération on visite le sommet le plus proche non visité (non marqué). Lorsque tous les sommets sont visités , on retourne au sommet de départ.

Cette heuristique donne toujours un cycle hamiltonien.

3.2.1.2 Amelioration local : k-OPT

Une fois un cycle initial trouvé, on applique une amélioration locale.

2-OPT

Partant d'une solution, on supprime deux arêtes non adjacentes dans le cycle, puis on reconstruit un cycle réalisable. On fait ceci pour chaque couple d'arêtes et on conserve la meilleure solution.

Nous allons jusqu'à 4-OPT ici soit l'échange de quatre arêtes.

3.2.1.3 Heuristique de Lin-Kernighan

L'heuristique de Lin-Kernighan est connue comme l'une des plus efficaces pour le problème de TSP.

Une fois arrivé à un 2-opt stable, on applique 3-opt sur la solution courante. En cas d'amélioration, on retourne à 2-opt, sinon on passe à 4-opt.
On applique ce procédé jusqu'à ce que 4-opt soit stable.

3.2.2 Algorithme

Algorithme 2: Algorithme de l'heuristique Lin Kernighan

Entrées : D : Matrice des distances entre chaque point

Sortie : S : La liste des point dans l'ordre de la solution du TSP

Initialisation :

S ⟵ Plus_proche_voisin(D)(1)

coût ⟵ calcul_cout(S,D)(2)

Début :

Repeter

S' ← 2-OPT(S, D)

S'' ← 3-OPT(S', D)

si (S' = S'') **alors**
S''' ← 4-OPT(S'', D)
si (S'' = S''') **alors**
S ← S'''
sortir de la boucle
sinon
S ← S''

S ← formater(S)(3)

---

1) Plus_proche_voisin(D) : Construis un tour initiale
2) Calculer_cout(S,D) : retourne la longueur totale du tour
3) Formater(S) : normalise le représentation de la solution : place 1 au début

3.2.4 Conclusion

Avec cette heuristique la taille du voisinage s'ajuste dynamiquement. Elle donne un TSP assez proche de l'optimum en un temps relativement rapide en pratique. En effet malgré une complicité théorique dans le pire cas n'est pas polynomiale en pratique pour un nombre de point inférieur à 100 l'algorithme converge rapidement.

3.3 Anneau étoile

L'heuristique du problème de l'anneau-etoile est la combinaison des heuristique de p médians et de TSP.

3.3.1 Approche théorique

3.3.1.1 Version en cascade

Dans un premier temps nous sélectionnons les médians en appliquant l'heuristique hybride. Dans un second temps on convertit la matrice D en une matrice qui contient uniquement les distances entre les médians à qui on applique le facteur alpha. Dans un dernier temps, on applique l'heuristique de colonie de fourmis sur les médians .

3.3.1.2 Version imbriquer

Dans ce cas nous n'attendons pas que la méta heuristique du puisse finisse pour appliquer la méta heuristique du TSP dessus. Nous allons à chaque résultat intermédiaire du p médian appliquer le TSP même si la solution n'est pas la meilleure.

3.3.2 Affichage

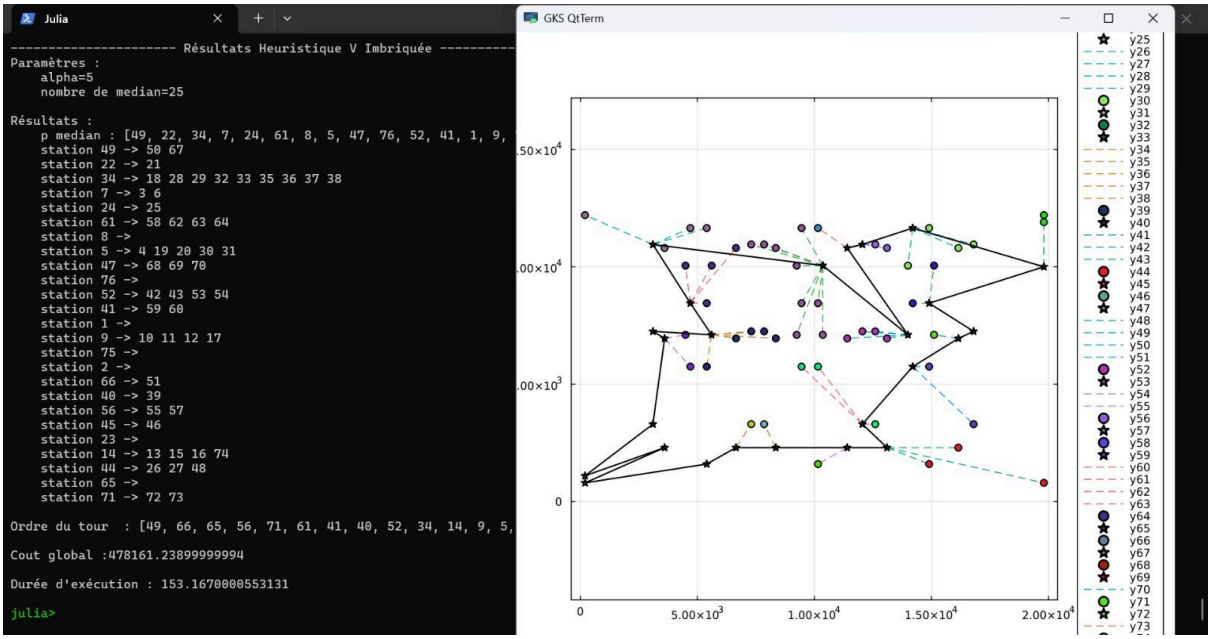

Figure 3 : affichage de l'heuristique avec alpha à 5 sur une instance de 76 points

3.3.3 Algo

Algorithme 3: Algorithme de l'heuristique Anneau Étoile non imbriquer

---

Entrées : p : nombre de médians
X : la liste des abscisse
Y: la liste des ordonnées
D : Matrice des distances entre chaque point

Sortie : M : matrice qui sur les diagonale contient les median et pour chaque (i,j)associe a i sont median j
C : la liste des indices des médians dans le matrice M .

T : liste des noeud dans l'ordre du TSP
cout : le cout de la solution

Initialisation :

M , C ← heuristique_hybride(p , X, Y , D) (algorithme 1)
D' ← matrice(p , p)

Début :

Pour i de 1 à p **faire**
Pour j de 1 à p **faire**
si (i = j) **alors**
D'[i,j] ← D[1,1]
sinon
D'[i,j] ← D[C[i],C[j]]
T_brut ← lin_kernighan (D') (algorithme 2)
T ← [ ]
Pour i de 1 à p **faire**
T[i] ← C[T_brut[i]]
coût ← calcul_global(α , M , D , T_brut , D')

Algorithme 4 : Algorithme de l'heuristique Anneau Étoile imbriquer

---

Entrées : p : nombre de médians
X : la liste des abscisse
Y: la liste des ordonnées
D : Matrice des distances entre chaque point
nombreDIteration
alpha

Sortie : M : matrice qui sur les diagonale contient les median et pour chaque (i,j)associe a i sont median j
C : la liste des indices des médians dans le matrice M .
tableauTSP : liste des noeud dans l'ordre du TSP
cout : le cout de la solution
coutTotal
duree

Initialisation :

debut ⟵ temps_courant()
n ⟵ taille(D,1)
M ⟵ matrice n×n remplie de 0
C ⟵ vecteur vide
probabiliter ⟵ vecteur de taille n, initialisé à 10×p
q ⟵ arrondir_sup(√p)
equarex ⟵ (max(X) − min(X)) / q
equarey ⟵ (max(Y) − min(Y)) / q

Début :

Pour iteration de 1 à nombreDIteration **faire**
Pour g de 1 à p−1 **faire**
Dic ⟵ dictionnaire vide
Pour chaque rectangle (i,j) dans la grille q×q **faire**
Tant que le nombre de stations C est trop petit :
Choisir aléatoirement une ville dans le rectangle, en tenant compte des probabilités.
Si la ville n'est pas déjà dans C **alors** l'ajouter à C.
Si longueur(C) < p alors
Ajouter les villes les plus probables jusqu'à atteindre p stations.
Sinon Si longueur(C) > p **alors**
Choisir aléatoirement p stations parmi C, en gardant au moins la ville 1.

Construire la matrice de distances entre les p stations.
Résoudre le TSP sur ces stations avec linKernighan → tableauTSPbrute.
Remonter le tour sur les villes réelles : tableauTSP[i] ⟵ C[tableauTSPbrute[i]].

Réinitialiser M : M[i,i] ⟵ 1 pour chaque station i ∈ C.
Pour chaque ville non-station, l'attribuer à la station la plus proche.

S ⟵ calculGlobal(alpha, M, D, tableauTSPbrute, matriceDesDistanceReduite)

Si iteration < nombreDIteration **alors**
Mémoriser C avec son coût S dans Dic.
C1 ⟵ ensemble C correspondant au meilleur coût.
C2 ⟵ ensemble C correspondant au pire coût.
Mettre à jour les probabilités :
Pour chaque ville différente entre C1 et C2,
augmenter la probabilité de celle dans C1,
diminuer celle dans C2.

Réinitialiser M avec le C final, puis attribuer chaque ville à sa station la plus proche.

coutTotal ⟵ calculGlobal(alpha, M, D, tableauTSP, D)

fin ⟵ temps_courant()

duree ⟵ fin − debut

3.3.4 Conclusion

La combinaison des deux méta heuristiques nous permet d'obtenir une solution réalisable satisfaisante pour les petites instances.

4 MétaHeuristique

4.1 P-median

Pour la métaheuristique du p médian nous avons fait le choix d'implémenter une recherche tabou.

4.1.1 Approche théorique : Recherche tabou

La recherche tabou part d'une solution fourni par l'heuristique hybride. Après avoir généré tous les voisins d'un median . Parmi cette liste de voisins, on choisit le meilleur voisin même si ce dernier dégrade la solution. On ajoute l'échange de ces deux nœuds dans la liste tabou. On itère jusqu'à un certain nombre d'itération.

4.1.2 Algorithme

Algorithme 5: Algorithme tabou

Entrées : p : nombre de médians
X : la liste des abscisse
Y: la liste des ordonnées
D : Matrice des distances entre chaque point

Sortie : M : matrice qui sur les diagonale contient les median et pour chaque (i,j)associe a i sont median j
C : la liste des indices des médians dans le matrice M .

Début :

M, C ⟵ heuristique_hybride(p, X, Y, D)

```
S ← cout_total(M, D)
tabuListe ← ∅
 tabuIteration ← 10
 stop ← faux

 Tant que stop = faux faire
        voisins ← generer_voisin(M, p, D, tabuListe)

         meilleur_S ← S
        meilleur_M ← copie(M)
        meilleur_C ← copie(C)
        meilleur ← faux

        Pour k de 1 à taille voisins faire
                (M_voisin, C_voisin, echange) ← voisins[k]
                S_voisin ← cout_total(M_voisin, D)

                 estTabou ← faux
                 Pour t de 1 à longueur(tabuListe) faire
                        Si tabuListe[t] = echange alors
                                estTabou ← vrai

                estMeilleure ← faux
                Si S_voisin < meilleur_S alors
                        estMeilleure ← vrai

                Si (estTabou = faux) et (estMeilleure = vrai) alors
                        meilleur_S ← S_voisin
                        meilleur_M ← copie(M_voisin)
                        meilleur_C ← copie(C_voisin)
                        meilleur_echange ← echange
                        meilleur ← vrai

        Si meilleur = vrai alors
                M ← copie(meilleur_M)
                 C ← copie(meilleur_C)
                S ← meilleur_S

                 ajouter meilleur_echange dans  tabuListe
                Si taille tabuListe > tabuIteration alors
                         supprimer la tête de tabuListe
                        stop ← vrai

Retourner M, C
```

4.2 TSP

4.2.1 Colonie de fourmis

Comme métaheuristique nous avons fait le choix d'une colonie de fourmis. L'algorithme se base sur un principe de phéromone et de visibilité des arcs .

L'idée est de construire un ensemble de fourmis qui vont effectuer des tours sur le graph en passant par toutes les villes et de renforcer les arcs de chemin les plus courts en déposant davantage de phéromones. Concrètement chaque fourni par d'une ville (un nœud du graph) . Elle se déplace en fonction de la quantité de phéromone présente sur l'arc et la visibilité (l'inverse de la distance) de l'arc pondérée par un paramètre β. Une fois tous les tours construits , les phéromones sont mises à jour en tenant compte de la qualité du cycle. On réitère.

4.2.2 Algorithme

Algorithme 6 : Algorithme de la Colonie de Fourmis pour le TSP

Entrées :
nbIterations : nombre d'itérations de l'algorithme
tauxEvap : taux d'évaporation des phéromones
coeffPhero : coefficient de dépôt de phéromones
alpha : poids de la phéromone dans le calcul de probabilité
beta : poids de la visibilité dans le calcul de probabilité
matriceDesDistances : matrice des distances entre les villes

Sorties :
meilleurChemin : liste des villes dans l'ordre de la tournée trouvée
meilleurCout : coût de ce tour
laDuree : temps d'exécution de l'algorithme

Initialisation :
leDebut ⟵ temps_courant()
nbVilles ⟵ nombre_de_lignes(matriceDesDistances)
visibiliteVilles ⟵ initVisibilites(nbVilles, matriceDesDistances)
nbFourmis ⟵ nbVilles
tourDesFourmis ⟵ matrice_entiers(nbFourmis, nbVilles, 0)

```
   meilleurChemin ← algoPlusProcheVoisin(matriceDesDistances)
   meilleurCout ← calculCoutTotalTSP(meilleurChemin, matriceDesDistances)

Début :

   Pour i de 1 à nbIterations faire
      pheromonesArcs ← initPheromones(nbVilles)
      villesDispo ← matrice_entiers(nbFourmis, nbVilles, 1)

   Pour ville de 1 à nbVilles faire
         tourDesFourmis[ville, 1] ← ville
         villesDispo[ville, ville] ← 0


      Pour j de 1 à nbVilles − 1 faire
         pheromonesArcs ← alterationPheromones(pheromonesArcs, tauxEvap)

         Pour fourmi de 1 à nbFourmis faire
            tabVillesDispo ← extraire_ligne(villesDispo, fourmi)
            villeActuelle ← tourDesFourmis[fourmi, j]

            tabProba ← calculTableauDeProba(villeActuelle, tabVillesDispo,
                  visibiliteVilles, pheromonesArcs, alpha, beta)

            numChoix ← nombre aléatoire entre 0 et 1
            numChoix ← numChoix arrondi  5 chiffres après la virgule

            raffineVillesDispo, raffineTabProba ← filtrerLesZeros(tabVillesDispo,
tabProba)

            indice ← 1
            nbVillesDispo ← longueur(raffineVillesDispo)
            somme ← 0.0

            Tant que indice ≤ nbVillesDispo et somme ≤ numChoix faire
               somme ← somme + raffineTabProba[indice]
               indice ← indice + 1


            choix ← raffineVillesDispo[indice − 1]

            tourDesFourmis[fourmi, j + 1] ← choix
            villesDispo[fourmi, choix] ← 0
```

```
        Pour k de 1 à nbFourmis faire
            tourDeLaFourmi ⟵ extraire_ligne(tourDesFourmis, k)

            coutDeCeTour ⟵ calculCoutTotalTSP(tourDeLaFourmi,
matriceDesDistances)

            Pour arc de 1 à nbVilles − 1 faire
                pheromonesArcs[tourDeLaFourmi[arc], tourDeLaFourmi[arc + 1]] ⟵
                    pheromonesArcs[tourDeLaFourmi[arc], tourDeLaFourmi[arc + 1]]
                    + coeffPhero × (1 / coutDeCeTour)


            pheromonesArcs[tourDeLaFourmi[dernier], tourDeLaFourmi[premier]] ⟵
                pheromonesArcs[tourDeLaFourmi[dernier], tourDeLaFourmi[premier]]
                + coeffPhero × (1 / coutDeCeTour)

            Si coutDeCeTour < meilleurCout alors
                meilleurChemin ⟵ tourDeLaFourmi
                meilleurCout ⟵ coutDeCeTour

    laFin ⟵ temps_courant()
    laDuree ⟵ laFin − leDebut

    meilleurChemin ⟵ formaterLesSolutions(meilleurChemin)
```

4.2.3 Paramétrage

Cette métaheuristique dépend de plusieurs paramètres : le nombre d'itération , le taux d'évaporation , le coefficient de dépôt , γ et β

Afin de les fixer au mieux, nous avons testé différents paramètres sur différentes instances. En faisant varier ces paramètres, on vérifie le coût du tour et le temps d'exécution.

α influence les phéromones et β la visibilité des arcs . En effet la probabilité de passer de la ville i a la ville j est de :

$$\tau_{ij}^{\gamma} \times \eta_{ij}^{\beta}$$

avec τ_{ij} est la quantité de phéromone et η_{ij} la visibilité.
Nous remarquons que fixer γ à 1 on obtient une mémoire suffisante sans trop dépendre du premier tour.

En revanche β accentue l'effet de distance, il favorise donc fortement les chemins les plus courts . Cela est visible sur notre tableau.

Pour le taux d'évaporation le choisir trop faible nous enfermera très vite sur quelque arcs alors que trop elever il nous empêchera d'engranger les meilleur solution.

Sur la base de nos observations un taux d'évaporation entre 0,3 et 0,6 selon les instances est le choix le plus judicieux .

Une fois ces paramètres fixés on remarque que le coefficient de dépôt de phéromone n'a pas beaucoup d'incidence.

Dans cent premières itérations l'amélioration de la solution est significative . Au-delà de 100 itérations la différence n'est plus importante. On se limite donc à 1000 itérations.

4.2.4 Conclusion

La colonie de fourmis permet de trouver une solution réalisable en explorant l'espace et en favorisant les chemins les plus courts.

4.3 Anneau étoile

4.3.1 Approche théorique

4.3.1.1 Premiere approche

Dans un premier temps nous avons décidé d'optimiser simultanément le p médian et le TSP avec une boucle d'amélioration récursive.

4.3.1.2 Version imbriquer

Dans ce cas nous n'attendons pas que la méta heuristique du puisse finisse pour appliquer la méta heuristique du TSP dessus. Nous allons à chaque résultat intermédiaire du p médian appliquer le TSP même si la solution n'est pas la meilleure.

4.3.1.3 Comparaison

Nous testons les deux versions et comparons les résultats ainsi que les valeurs de α. Nous cherchons à savoir quelle version donne les meilleurs résultats et sur quelle version α a une vraie incidence.

Cela nous permettra de choisir une approche ainsi que de fixer une valeur de alpha.

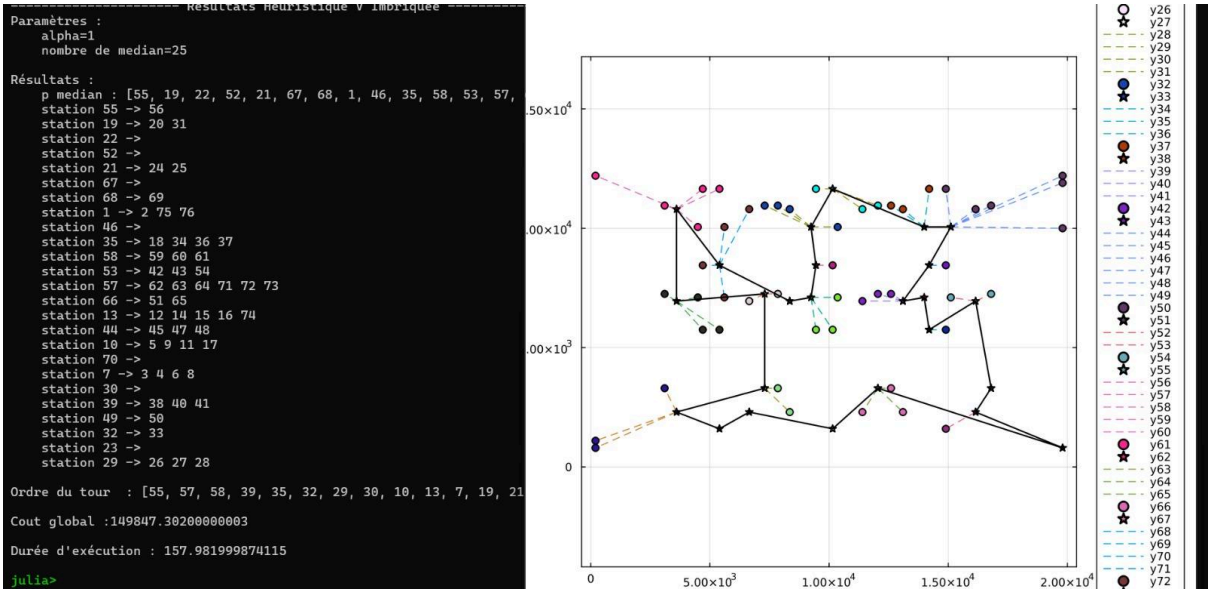

Figure 4 : L'algorithme imbriquer avec alpha a 5 et 76 point

Suite à nos expérimentations nous somme arriver a la conclusion que la métaheuristique était plus performante avec un alpha à 5.

4.3.2 Algorithme

Algorithme 7 : Algorithme Métaheuristique Anneau Étoile imbriquer

---

Entrées : p : nombre de médians
X : la liste des abscisse
Y: la liste des ordonnées
D : Matrice des distances entre chaque point

nbIterations : nombre d'itérations de l'algorithme
tauxEvap : taux d'évaporation des phéromones
coeffPhero : coefficient de dépôt de phéromones
alpha : poids de la phéromone dans le calcul de probabilité
beta : poids de la visibilité dans le calcul de probabilité

alpha_cout

Sortie : M : matrice qui sur les diagonale contient les median et pour chaque (i,j)associe a i sont median j
C : la liste des indices des médians dans le matrice M .
tableauTSP
coutTotal
duree

Initialisation :
debut ⟵ temps_courant()
(M, C) ⟵ heuristic_hybride(p, X, Y, D)
S ⟵ coutGlobalFromPMedian(p, C, M, D, nbIterations, tauxEvap, coeffPhero, alpha, beta, alpha_cout)
tabuListe ⟵ liste vide
tabuiteration ⟵ 10
stop ⟵ faux

Début :
Tant que stop = faux **faire**
Générer une liste de voisins de la solution (M, C) en changeant quelques stations ou affectations.
Pour chaque voisin (M_voisin, C_voisin, echange) **faire**
Si echange est tabou alors passer au suivant.
Calculer S_voisin ⟵ coutGlobalFromPMedian(p, C_voisin, M_voisin, D, nbIterations, tauxEvap, coeffPhero, alpha, beta, alpha_cout)
Si S_voisin < meilleur_S alors
Mémoriser ce voisin comme meilleur.

Si un meilleur voisin non tabou a été trouvé **alors**
Mettre à jour M, C, S avec ce voisin.
Ajouter echange à tabuListe.
Si tabuListe dépasse tabuiteration **alors** retirer le plus ancien.
Sinon
stop ⟵ vrai

Construire le tour final des stations avec antColony puis calculer coutTotal.
fin ⟵ temps_courant()
duree ⟵ fin − debut

4.3.4 Conclusion

Les heuristique nous permette de sortir de minimum locaux. Elle donne donc de meilleures solutions.

5 Formulation compacte

Nous avons tenté d'implémenter la formulation compacte donnée dans le sujet, mais nous ne sommes pas parvenu à la faire fonctionner : un problème réalisable était donné comme irréalisable. Le seul problème qui semble apparaître est dans la logique du flot z, que nous ne sommes pas parvenu à résoudre.
Nous avons donc tenté une approche moins performante mais plus simple : MTZ.
Cette formulation fonctionne avec le problème de 14 points (avec p=4), plus lentement pour celui de 51 points (avec p=12) mais nous obtenons un résultat (environ quarante minutes). Au-delà, nous ne sommes pas parvenu à un quelconque résultat.
Par ailleurs, cette formulation semble mieux tourner avec HiGHS qu'avec GLPK.

$$Min \sum_{ij \in E} d_{ij} x_{ij} + \sum_{(i,j) \in V \times V} d_{ij} y_{ij}$$

$$\sum_{i \in V} y_{ii} = p \tag{1}$$

$$\sum_{j \in V} y_{ij} = 1 \qquad \forall i \in V \tag{2}$$

$$y_{ij} \leq y_{jj} \qquad \forall (i,j) \in V \setminus \{j\} \times V \tag{3}$$

$$\sum_{ij \in \delta(i)} x_{ij} = 2 y_{ii} \qquad \forall i \in V \tag{4}$$

$$\sum_{j \in V \setminus \{1\}} z_{1j} = p - 1 \tag{5}$$

$$\sum_{j \in V \setminus \{i\}} z_{ji} = \sum_{j \in V \setminus \{1,i\}} z_{ij} + y_{ii} \qquad \forall i \in V \setminus \{1\}, \tag{6}$$

$$z_{ij} + z_{ji} \leq (p-1) x_{ij} \qquad \forall i \in V, j \in V \setminus \{1,i\} \tag{7}$$

$$x_{ij} \in \{0,1\} \qquad \forall ij \in E$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in V \times V$$

$$z_{ij} \in [0, p-1] \qquad \forall (i,j) \in V \times V \setminus \{1,j\}$$

$$y_{11} = 1$$

$$y_{1j} = 0 \qquad \forall j \in V \setminus \{1\}$$

6 Comparaison des méthodes

Afin d'évaluer l'efficacité de nos méthodes de résolution, nous devons comparer les optimaux obtenues par résolution exacte avec la formulation compacte. Nous décidons de comparer les trois méthodes sur une instance de 51 nœuds.
Nous représentons la comparaison sous forme de tableau.

Méthode	**Qualité solution (coût total)**	**Garantie d'optimalité**	**Temps de calcul (ordre de grandeur)**	**Scalabilité quand n augmente**	**Paramétrage / mise en œuvre**
Heuristique	Moyenne : dépend fortement du choix initial des stations et de la qualité du 2-opt/3-opt/4-opt.	Aucune	Très faible (quelques dixièmes de seconde sur 51 points).	Bonne : reste utilisable pour des tailles importantes car tout est en temps polynomial.	Simple, peu de paramètres (nombre d'itérations de voisinage, choix de l'heuristique p-médian).code.pdf
Métaheuristique	Bonne : les meilleurs coûts sont proches des meilleures heuristiques de tournée, parfois meilleurs, mais avec une forte variabilité selon α,β,évaporation, etc.	Aucune	Faible à modérée (1–3 s pour 51 points avec 125 itérations, selon les paramètres).	Bonne : plus coûteuse que l'heuristique simple mais toujours utilisable sur des instances moyennes, surtout si on limite le nombre d'itérations et de fourmis.	Paramétrage délicat : nécessite de choisir itérations, taux d'évaporation, coefficient de dépôt, α,β, et d'analyser les résultats.

PLNE compacte	Très bonne : fournit une borne inférieure et souvent la solution optimale pour 51	Oui (si le solveur va jusqu'à la fin de l'énumération)	Élevé mais raisonnable sur 51 points et des valeurs de p modestes ; peut devenir important dès que n	Faible : le nombre de variables/contraintes croît, et le temps peut exploser pour des instances plus	Mise en œuvre conceptuelle claire, mais dépend fortement du solveur MIP et des réglages (coupes, temps limite, etc.).

7 Conclusion

Ce projet a permis d'expérimenter différentes approches de résolution pour le problème de l'anneau étoile.

Le problème de l'anneau étoile est un problème NP-difficile car il existe une réduction polynomiale du TSP .

Nous avons implémenté une heuristique qui dans un premier temps résout le problème de p médian avec une heuristique hybride qui utilise une heuristique randomisé à qui on ajoute un aspect probabiliste combiner à la méthode des rectangle et dans un second temps donne une solution réalisable pour le problème de TSP sur les médians obtenu précédemment . Cette heuristique donne une solution réalisable satisfaisante sur les instances de petite taille.

Nous avons aussi implémenté une métaheuristique qui comme pour l'heuristique résout consécutivement le problème de p médian avec une métaheuristique tabou qui cherche à visiter toute les solution en sortant des minimum locaux et le TSP avec une métaheuristique colonie de fourmis qui simule le comportement des fourmis pour favoriser les chemin les plus court.

Afin de comparer cela nous avons implémenté la formulation compacte grâce au solveur HIGHS.

BIBLIOGRAPHIE

A. COSTANZO , T.V LUONG , G. MARILL Optimisation par colonies de fourmis [travaux] (citer page 19)

Jean-Christophe Gay. Résolution du problème du p-médian, application à la restructuration de bases de données semi-structurées. [these]

P. HANSEN Heuristic solution of the multisource Weber problem as a p-median problem

Github [internet] : Project in Java to solve the Travelling Salesman Problem (TSP). An approximation solution with self-organizing maps (SOM) is proposed.
https://ansegura7.github.io/TSP/ (Page consultée le 04/01/2026)