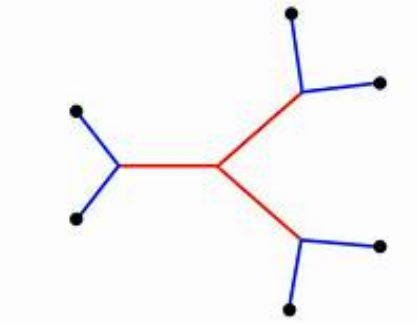
Sorbonne Université

Rapport de projet d'optimisation

MAIN 5

Problème de l'arbre de Steiner Euclidien



Membres :
Ramatoulaye NDIAYE
Harold GALLICE
Paola ALLEGRINI
Mahshid KHEZRI NEJAD

Encadrant :
Hacène OUZIA

18 février 2019

Table des matières

1 Introduction

Le sujet de ce projet porte sur la résolution du problème d'optimisation non linéaire en variables entières suivant. Étant donnés des points dans un espace de dimension fixée (dits terminaux), on cherche d'autres points (dits points Steiner) de telle sorte que la somme des distances des points Steiner aux terminaux soit la plus petite possible. Vu autrement, il s'agit de connecter les points terminaux aux points Steiner de telle sorte que la longueur totale des liens soit minimisée. Ainsi, le graphe ayant pour sommets les points terminaux et points Steiner et pour arêtes les liens entre ces derniers pour lesquels la distance totale est minimisée est nécessairement un arbre.

Le problème pourra donc se définir ainsi : considérant N points donnés (les terminaux) dans $\mathbb{R}^d$, trouver l'arbre de longueur totale (somme des distances Euclidiennes) minimale qui couvre les terminaux et utilisant, éventuellement, des points Steiner. Cet arbre sera appelé arbre de Steiner minimal. L'intérêt d'ajouter des points Steiner est de réduire la distance totale des connexions.

Le problème de l'arbre de Steiner minimal est un problème NP-difficile (voir [4]). Ce problème se manifeste dans plusieurs domaines, par exemple, en phylogénétique (voir [6, 2, 1]) et dans l'étude structurale de certaines protéines (voir [10, 8]). Le projet est scindé en trois parties :

1. Implémentation de l'algorithme d'énumération complète proposé par Smith
   Objectif : Optimiser l'emplacement des points Steiner de manière à trouver un arbre minimal de steiner connectant N sites en dimension D.
2. Conception et réalisation de nouvelles heuristiques ou méta-heuristiques
3. Exploration de l'efficacité de l'approche lagrangienne

2 Présentation du problème

2.1 Définition mathématique

Le problème présenté peut aussi se définir mathématiquement comme un problème de graphe :
On considère un graphe connecté $G = (V, E)$.
P : l'ensemble des indices associés aux points de base,
X : l'ensemble des indices associés aux points Steiner,
V = P∪X l'ensemble des sommets,
On note [i,j] : un bord de G avec i, j ∈ V tel que i<j.
Y = {[i,j] | i ∈P, j∈S} liens terminaux et points Steiner,

Z = {[i,j] | i ∈S, j∈S} liens entre les points Steiner,
E = $Y \cup Z$ tel que $\sum_{e \in E} |e|$ est minimisée (avec $|e|$ la distance euclidenne de l'arête $e \in E$).

2.2 Définitions et propriétés d'un arbre de Steiner minimal

Un arbre minimal de Steiner possède plusieurs propriétés qui sont utilisées par la suite. Vous retrouverez ces propriétés dans le livre de Gilbert et Pollack en 1968 ([3]), ici nous ne ferons que les énoncer.

- Nombre de points Steiner :
  Un arbre de Steiner minimal avec N terminaux possède au plus $N-2$ points Steiner.
- Propriétés d'angle :
  Les arêtes connectées à un même point forment des angles d'au moins 120°. Cela implique que les points Steiner ont exactement trois arêtes incidentes formant des angles de 120°.
- Degré :
  Les points Steiner ont au plus 3 voisins donc ils sont de degré 3. Chaque terminal est de degré 1.

Ces propriétés nous donnent une orientation pour énumérer des solutions réalisables pour ce problème.
On appellera une topologie : une configuration des terminaux et des points Steiner dans laquelle les arêtes connectant les nœuds sont spécifiées mais pas la position géométrique (les coordonnées) des points Steiner. D'où on peut avoir plusieurs configurations possibles des points Steiner.

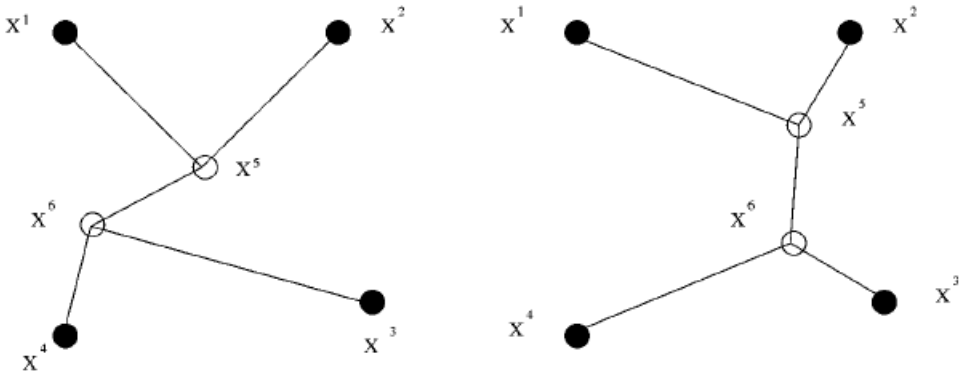


FIGURE 2 – Exemple de 2 arbres appartenant à la même topologie pour 4 points donnés

Une topologie sera dite topologie de Steiner si chaque point Steiner est de degré 3 et chaque terminal est au plus de degré de 3. Une topologie de Steiner où tous les terminaux sont de degré 1 est dite topologie de Steiner complète. Si l'on considère N points terminaux, cette topologie a $N-2$ points Steiner. En dernier, toute topologie de Steiner non-complète peut

être vue comme une topologie de Steiner complète avec la présence d'arêtes de longueur zéro (ce type de topologie est appelée dégénérescente).

On considère alors la recherche de topologies de Steiner complètes. Un algorithme permet d'obtenir une solution exacte du problème en effectuant une recherche exhaustive sur l'ensemble des solutions réalisables c'est à dire sur toutes les topologies d'arbre Steiner possibles : l'algorithme de Smith. Smith n'a utilisé que des topologies Steiner complètes pour traiter ce problème, en utilisant un schéma énumératif intelligent avec une manière de représenter les arbres.

2.3 Modélisation du problème

Le problème à résoudre est le problème d'optimisation ci-dessous :

$$
\begin{aligned}
\min_{s.c} \quad & \sum_{p\in P, q\in X} y_{pq}\|X^q - \zeta^p\|_2 + \sum_{p\in P, q\in X : p<q} z_{pq}\|X^q - X^p\|_2 \\
& \sum_{q\in X} y_{pq} = 1, p \in P, \\
& \sum_{p\in P} y_{pq} + \sum_{p\in X : p<q} z_{pq} + \sum_{p\in X : p>q} z_{qp} = 3, q \in X, \\
& \sum_{p\in X} z_{pq} = 1, q \in X, \\
& \sum_{p\in P} y_{pq} \leq 2, q \in X, \\
& X^p \in \mathbb{R}^d, p \in P, \\
& y_{pq} \in \{0,1\}, p \in P, q \in X, \\
& z_{pq} \in \{0,1\}, p, q \in X, p < q.
\end{aligned}
\tag{1}
$$

Nous avons :

- L'entier N avec $N \geq 3$ le nombre de points terminaux et d la dimension de l'espace.
- les ensembles d'indices $P = \{1, ..., N\}$ et $X = \{1, ..., N-2\}$ associés respectivement aux terminaux et aux points Steiner.
  Les terminaux sont notés ζ_p, $\forall p \in P$ et les points Steiner X^q, $\forall q \in X$.
- Les variables y et z représentent respectivement les liens terminaux - points Steiner et points Steiner - points Steiner. Si la variable vaut 1, une arête lie les deux points sinon il n'y a pas d'arête.

En dernier, nous retrouvons plusieurs propriétés du problème dans nos contraintes.
(1) : chaque terminal est de degré 1, relié à un point Steiner
(2) : chaque point Steiner est de degré 3

(3) : un point Steiner est relié à un seul point Steiner.

Cette formulation du problème n'est pas unique, nous pouvons par exemple citer le modèle mathématique proposé par N. Maculan, Ph. Michleon - [5]).

2.4 Cas particulier de 3 terminaux : solution exacte

Dans le cas particulier où $N = 3$, nous connaissons et savons construire la solution optimale au problème. En effet si les points forment un triangle avec un angle supérieur à 120°, alors les deux côtés plus courts forment l'arbre le plus court.
Dans le cas classique où aucun des angles formés par le triangle des 3 terminaux est supérieur à 120° alors un point Steiner bien positionné et connecté à tous les autres points est la solution optimale. Ce point est appelé le point de Torricelli (ou point de Fermat) du triangle et il peut être définit "à la main" en construisant les triangles équilatéraux externes donnés par chaque côté.

En connectant le sommet de chacun de ces triangles au terminal opposé, l'intersection des droites nous donne la position du point Steiner comme on peut le voir sur la figure ci-dessous :

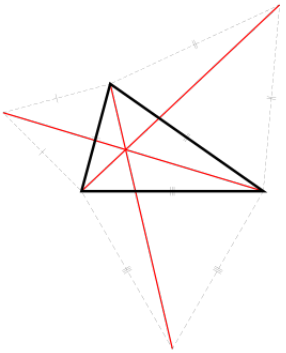


FIGURE 3 – Point de Torricelli

Nous venons de voir que géométriquement nous obtenons la solution exacte cependant algorithmiquement nous ne pouvons pas procéder de la même manière. Dans le cas d'un triangle quelconque nous utiliserons la méthode du gradient pour trouver les coordonnées du point optimal. La fonction à minimiser sera la somme des distances entre les sommets du triangle et le point Steiner.

3 Algorithme de Smith

L'algorithme de Smith est un algorithme intéressant à étudier car il est le premier algorithme qui propose une solution exacte pour des points de dimension $d \geq 3$ et l'ensemble des programmes informatiques connus pour résoudre ce problème se sont basés sur ce code.

3.1 Le principe de l'algorithme

Vous trouverez dans la suite le pseudo-code de l'algorithme de Smith :

Algorithm 1 Smith's Algorithm

Require: X : Set of Coordonates of N point sites in dimension d ($d \geq 3$)
Ensure: min_T : Minimum Steiner Tree
  $Y \leftarrow initialize\_the\_Steiner\_Point\_Coords(X)$
  $T \leftarrow find\_all\_the\_Steiner\_Tree\_Topologies(X)$
  for all t in T **do**
    // To find the shortest possible euclidean embedding of t
    Optimize_Steiner_Coords(t, Y)
  end for
  return min_T = the shortest tree found from the for procedure

Détails :

- Dans le cas où N=3 : la solution exacte est connue et est donc calculée directement sans suivre le pseudo-code.
- La fonction *optimize*() : méthode itérative qui met à jour les coordonnées des points Steiner simultanément à chaque étape. Cette méthode converge en $O(N)$. Le critère de convergence choisit s'appuie sur le calcul de l'erreur dépendant de la taille des angles, une condition d'optimalité est que les angles de l'arbre soient tous $\geq 120°$.

Complexité :
Soit N le nombre de points terminaux et soit K le nombre de points Steiner, d'après Gilbert et Pollak on sait que le nombre de toutes les topologies possibles (l'invariant de boucle **for**) est égal à :

$$\binom{N}{K+2} \frac{(N+K-2)!}{K!2^K}$$

Donc cet algorithme a bien une complexité non-polynomiale.

3.2 Représentation des arbres dans l'algorithme

Dans cette partie nous allons discuter de la génération des arbres dans l'algorithme de Smith. Premièrement on considère les topologies Steiner complètes avec K = N-2 points

Steiner. Dans le cas où l'arbre de Steiner optimal a K < N-2 points Steiner, nous regardons ceci simplement comme une topologie Steiner complète "dégénérée" dans laquelle les points Steiner ont fusionné. Tous les algorithmes pratiques pour trouver l'arbre minimal de Steiner de N terminaux dans un espace de dimension d ont suivi essentiellement le plan suivant :

- Retrouvez toutes les "topologies" des arbres Steiner sur N terminaux et K points Steiner.
- Pour chaque topologie, optimiser les coordonnées des points Steiner afin de trouver l'intégration euclidienne la plus courte possible de cette topologie.
- Éditez l'arborescence la plus courte ainsi trouvée.

Construction de l'arbre On se fixe un nombre de terminaux N. L'emplacement initial des nouveaux points Steiner est une légère perturbation aléatoire du centroïde de ses voisins. Après, on stocke la longueur des arêtes dans un tableau et on retourne la longueur totale.

3.3 Résultats et efficacité de l'algorithme de Smith

Le code en C de l'algorithme de Smith a été donné à la fin de son article [9]. Il a donc été possible de l'implémenter. Une surcouche en python a été créée afin de profiter d'une visualisation de nos arbres de Steiner. On peut donc effectuer le calcul du problème du pentagone (Figure 4) en deux dimensions dont le résultat est connu explicitement. L'algorithme de Smith nous donne deux topologies différentes. Nous avons choisi de représenter l'une des deux. (Figure 5).

x	y
0.5000000	0.2500000
0.3272543	0.4877641
0.0477458	0.3969463
0.0477458	0.1030537
0.3272542	0.0122359

FIGURE 4 – Points du pentagone

x	y
0.5000000	0.2500000
0.3272543	0.4877641
0.0477458	0.3969463

FIGURE 5 – Points Steiner de l'arbre minimal

A l'aide de notre surcouche en Python, une représentation graphique est accessible pour ce problème (Figure 7). Des visualisations sont possibles en 3 dimensions ou pour plus de points. Malheureusement, comme spécifié dans la section précédente, l'algorithme de Smith est une énumération. L'algorithme étant effectué en un temps non-polynomial, le temps de calcul nécessaire pour une topologie dépassant les 12 points devient très lourd. On peut voir sur le graphique 6 que l'algorithme n'est plus utilisable quand on dépasse un certain nombre de points.

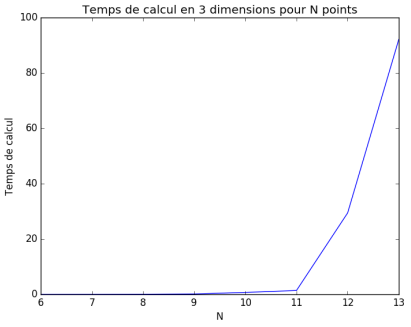


FIGURE 6 – Temps de calcul en 3 dimensions de l'algorithme de Smith

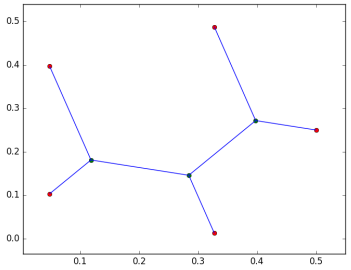


FIGURE 7 – Arbre Steiner minimal pour le problème de pentagone

Script Python : Le script python créé prend en argument un fichier de données .dat qui contient dans l'ordre : le nombre de terminaux, la dimension et les coordonnées de ces points. Ce script appelle ensuite le code en C de Smith qui extrait différentes topologies de solution du problème au cours de l'énumération. Après extraction, le script python stocke chacune des topologies dans un nouveau dossier dans des fichiers ".topo" contenant les terminaux, les points Steiner trouvés et les liens entre les points. La meilleure topologie trouvée est également dessinée dans un fichier ".png". Un script draw.py est également disponible pour afficher la visualisation de la figure de n'importe quel fichier .topo.

L'algorithme de Smith étant performant jusqu'à une douzaine de points, il est possible de l'utiliser pour des figures connues. Vous pouvez trouver à la suite la simulation en trois dimensions pour le cube et le tétraèdre. A chaque fois seront affichées la moins bonne topologie donnée par l'algorithme de Smith et la meilleure.

3.3.1 Arbre Steiner minimal du cube

x	y	z
0	0	0
0	0	0.577735
0	0.57735	0
0	0.57735	0.57735
0.57735	0	0
0.57735	0	0.57735
0.57735	0.57735	0
0.57735	0.57735	0.57735

FIGURE 8 – Points du cube

Topologie	Longueur totale
Moins bonne	3.6180668224210537
Meilleure	3.5774449549905585

FIGURE 9 – Longueurs des différentes topologies trouvées

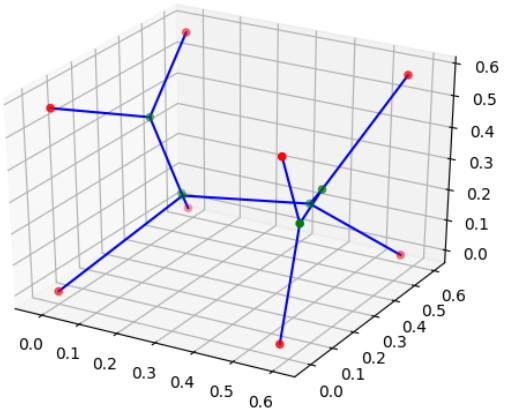


FIGURE 10 – Moins bonne topologie du cube

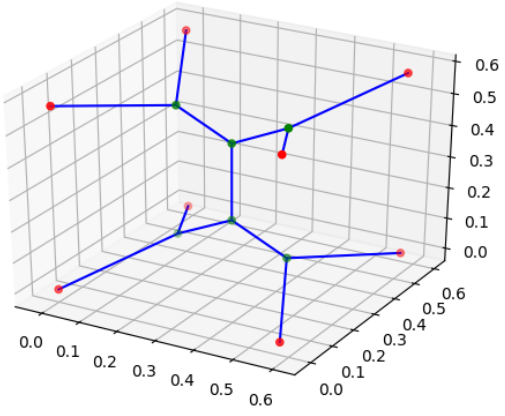


FIGURE 11 – Meilleure topologie du cube

3.3.2 Arbre Steiner minimal du tétraèdre

x	y	z
0.0	0.0	0.0
1.0	0.0	0.0
0.5	0.86602540378	0.0
0.5	0.28867513459	0.81649658092

FIGURE 12 – Points du tétraèdre

Topologie	Longueur totale
Moins bonne	2.4391575887441546
Meilleure	2.4391575887432886

FIGURE 13 – Longueurs des différentes topologies trouvées

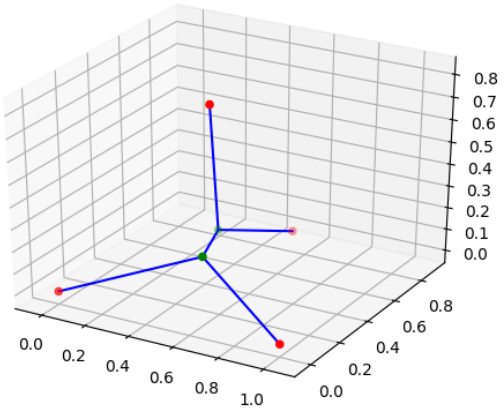


FIGURE 14 – Moins bonne topologie

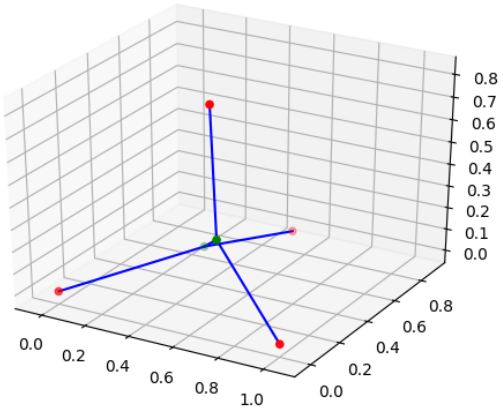


FIGURE 15 – Meilleure topologie

4 Résolution approchée

4.1 Algorithmes de colonies de fourmis

Les algorithmes d'optimisation par colonies de fourmis sont des algorithmes d'optimisation discrets qui s'inspirent de l'observation du comportement d'alimentation des colonies de fourmis. En particulier la façon dont les fourmis peuvent trouver les chemins les plus courts entre les sources de nourriture et leur nids.

En marchant des sources de nourriture au nid et vice versa, les fourmis déposent sur le sol une substance appelée phéromone, formant ainsi une traînée. Les fourmis peuvent sentir les phéromones et sont particulièrement sensibles à la concentration de phéromones lors du choix de leur chemin. La traînée de phéromones permet ainsi aux fourmis de retrouver leur chemin vers la source de nourriture (ou vers le nid). De plus, cette substance peut être utilisée par d'autres fourmis pour trouver l'emplacement des sources de nourriture trouvées

par leurs compagnons de nidification.

Il a été démontré expérimentalement que cette traînée de phéromones suivant le comportement peut donner lieu, une fois employée par une colonie de fourmis, à l'émergence de chemins les plus courts. En d'autres termes, lorsqu'il y a plusieurs chemins disponibles entre le nid et une source de nourriture, une colonie de fourmis peut être en mesure d'exploiter les traces de phéromones laissées par les fourmis individuellement pour découvrir le chemin le plus court entre le nid et la source de nourriture.

Pour réaliser cet objectif, les fourmis partent aléatoirement par des chemins différents vers la source de nourriture. Elles déposent sur leur route une quantité de phéromones constante. Cette quantité de phéromones s'évapore avec le temps. Le chemin choisi par les fourmis évolue avec la concentration de phéromones sur chaque chemin disponible. Si la concentration s'évapore avec le temps, les chemins les plus courts seront les chemins avec le plus de concentration car les fourmis sont plus concentrées sur ces chemins. Ainsi avec le temps la quantité en phénomène s'accélèrera car plus de fourmis choisiront ce chemin et la concentration augmentera, on aura convergence de choix vers le chemin le plus court.

4.1.1 Adaptation au problème de l'arbre de Steiner minimal

Cette implémentation a été largement inspirée par l'article de Monsieur Prossegger et Monsieur Bouchachia [7]. L'idée est de tirer un terminal de départ, de trouver le chemin le plus court vers un terminal à l'aide de l'optimisation par colonie de fourmis, puis réitérer l'algorithme vers les autres terminaux, avec comme points de départ le chemin parcouru depuis le début. La colonie ne sera donc pas un point mais un ensemble de point et la fourmi peut partir de n'importe lequel de ces points.

Ainsi nous obtenons les 3 points les plus proches. Nous les marquons et créons le point Steiner de ces 3 points, qui sera le point de Torricelli. Nous recherchons des terminaux proches à l'aide de nos colonies de fourmis et dès que nous obtenons 2 points non marqués et un marqué, nous créons le point Steiner.

4.1.2 Implémentation

Notre code a été fait en python disponible sur github (`https://github.com/HaroldGal/fourmi_algo`) et reprend les algorithmes du papier [7]. Le nombre de fourmis a été fixé à 100 et le nombre de mise à jour de phéromone à 20. L'évaporation est fixée à 0.5 et les phéromones initiaux et incrémentés à 0.2.

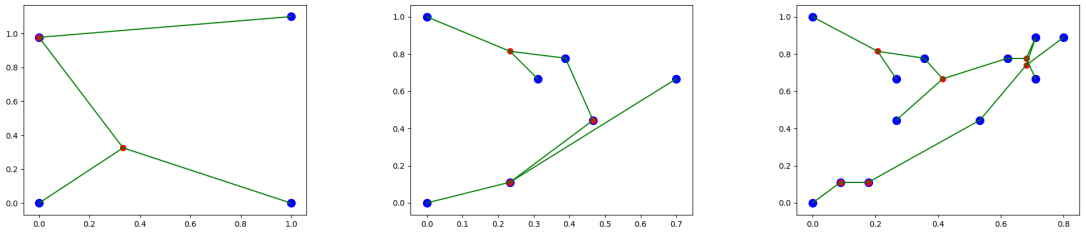


FIGURE 16 – Tests de l'algorithme des fourmis avec 4, 7 et 12 terminaux

Topologie 1	
x	y
0.0	0.0
1.0	0.0
0.0	1.0
1.0	1.1

Topologie 2	
x	y
0.0	1.0
0.0	0.0
0.5	0.5
0.4	0.8
0.7	0.7
0.2	0.1
0.3	0.7

Topologie 3	
x	y
0.0	1.0
0.0	0.0
0.5	0.5
0.4	0.8
0.7	0.7
0.2	0.1
0.3	0.7
0.12	0.15
0.3	0.5
0.7	0.9
0.8	0.9
0.6	0.8

TABLE 1 – Coordonnées des points pour 3 topologies test

4.1.3 Comparaison avec Geosteiner

Logiciel Geosteiner
Geosteiner est un logiciel 2D permettant de trouver l'arbre de steiner minimal en 2D et pouvant aller jusqu'à 10000 points. Nous n'avons plus les mêmes propriétés qu'avant par exemple les N-2 points Steiner. Il n'est pas non plus exploré en 3D. Cet outil nous permettra de comparer les résultats de nos heuristiques.

Les résultats de la comparaison sont disponibles dans le tableau 2. On remarque que l'heuristique ne donne pas de résultats incohérents mais ses performances ne sont pas au niveau de celles de Geosteiner. De plus, le temps de calcul de recherche de plus courts chemins par les fourmis est long, l'algorithme étant très gourmand.

Topologie	Longueur geosteiner	Longueur fourmis
1	2.783398407683427	2.94784564012049
2	1.800461125071304	2.35827508223738
3	2.183571204973291	2.8984881664840825

TABLE 2 – Comparaison des résultats pour différentes topologies

4.2 Algorithme d'optimisation par essaim d'abeilles

Les algorithmes d'optimisation par essaim d'abeilles sont des algorithmes itératifs permettant la recherche d'un optimum grâce à des déplacements d'abeilles. Ces abeilles ont des coordonnées et tentent d'optimiser une fonction objective. Elles peuvent se déplacer suivant un gradient, mais aussi de façon plus aventureuse pour éviter les minimums locaux. De plus elles sont capables de communiquer entre elles et ainsi éviter les faibles minimums locaux. Ces méta-heuristiques sont très puissantes.

4.2.1 Adaptation au problème de Steiner

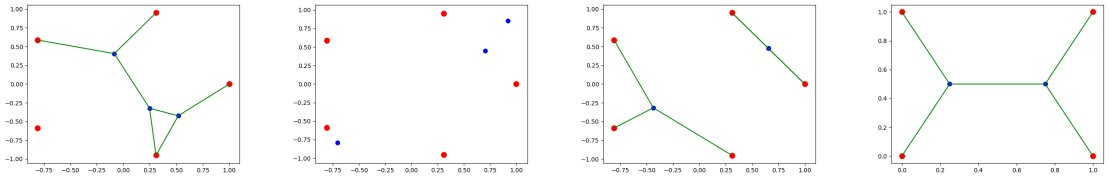
Dans le problème de Steiner les abeilles communiqueront entre elles pour conserver un angle de 120 degrés et espérant ainsi obtenir une topologie correcte obtenant des points Steiner intéressants.

4.2.2 Implémentation

Le code a été implémenté en python. A chaque itération il suffit d'un mouvement d'une abeille pour demander le déplacement de toutes les autres abeilles. Le pseudo code est détaillé dans l'algorithme 2 ci-dessous.

Algorithm 2 Trouver les points Steiner

Require: T liste des coordonnées des terminaux, N nombre de terminaux
  Initialiser *abeilles*, une liste de N-2 abeilles, aux coordonnées random entre le min et max de T de chaque dimension
  while $abeilles^t \neq abeilles^{t-1}$ **do**
    for abeille in abeilles **do**
      $Proches \leftarrow$ 3 terminaux ou abeilles plus proches de abeille
      Déplacer abeille pour avoir le bon angle avec $Proches$
    end for
  end while



(a) (b) (c) (d)

FIGURE 17 – Différents tests de l'algorithme

L'algorithme n'est pas fini car un autre projet porte dessus. Nous finirons cette étude au cours donc de ce projet futur. Nous avons néanmoins repéré quelques problèmes évidents visibles dans la figure 17.

Le cas (a) montre qu'un terminal peut se trouver complètement isolé car les abeilles et les autres terminaux obtiennent un certain équilibre. Dans le cas (b) et (c), les abeilles initiales sont très éloignées les unes des autres et forment donc des sous cycles. Mais nous pouvons voir que nous obtenons la solution exacte pour le cas (d), ce qui est encourageant pour l'amélioration de l'algorithme et le projet à venir.

La principale idée sera de forcer une condition de l'arbre Steiner minimal, obliger les terminaux à être reliés à un et un unique point Steiner. En marquant ainsi les terminaux déjà visités, nous pourrions obtenir de nouvelles solutions certainement réalistes et une heuristique prometteuse.

5 Approche Lagrangienne

5.1 Rappel du principe de la relaxation Lagrangienne

Le but de cette partie est de résoudre le problème à l'aide de la technique de la relaxation Lagrangienne. Cette technique consiste à supprimer des contraintes difficiles. Celles-ci sont ensuite intégrées dans la fonction objective et elles pénaliseront la fonction si elles ne sont pas respectées.

Ainsi, si l'on considère le problème de base du type :

$$\begin{aligned} &\underset{y}{\text{Min}} \quad f(x) \\ &\text{s.c :} \qquad\quad x \in X \cap F \end{aligned}$$

X et F sont les ensembles représentant les contraintes d'inégalité du type $g(x) \leq 0$.

La relaxation lagrangienne s'écrit : $\Theta(\pi) = min\{L(x,\pi) : x \in F, \pi \geq 0\}$.

Le vecteur π correspond aux multiplicateurs de Lagrange associés aux contraintes d'inégalité et L représente **la fonction de Lagrange** qui s'exprime ainsi :

$$L(x,\pi) = f(x) + \sum_{i=1}^{n} \pi_i g_i(x)$$

Le dual Lagrangien est défini comme suit :

$$\begin{aligned} &\underset{y}{\text{Max}} \quad \Theta(\pi) \\ &\text{s.c :} \qquad\quad \pi \geq 0 \end{aligned}$$

Rappelons également que la valeur d'une solution optimale de la relaxation continue constitue un minorant pour la valeur d'une solution optimale du problème primal.
La convexité de la fonction de Lagrange L assure l'existence d'un unique minimum Θ. Par ailleurs, la relaxation lagrangienne Θ étant concave, on a bien un unique maximum qui correspond au dual lagrangien.

Dans notre cas, les contraintes à relâcher sont celles qui imposent que tous les points Steiner soient de degré 3 :

$$\sum_{p \in P} y_{pq} + \sum_{p \in X : p < q} z_{pq} + \sum_{p \in X : p > q} z_{qp} - 3 = 0, q \in X \tag{2}$$

De plus nous devons avoir, comme vu précédemment, une fonction objective Lagrange différentiable. Pour ce faire, nous changeons la fonction f(x). Au lieu de considérer les distances entre les points, nous allons considérer les distances élevées au carré. Nous avons donc pour f(x) :

$$\sum_{p \in P, q \in X} y_{pq} \|X^q - \zeta^p\|_2{}^2 + \sum_{p \in P, q \in X : p < q} z_{pq} \|X^q - X^p\|_2{}^2 \tag{3}$$

5.2 Principe de l'algorithme de sous-gradient

Le principe de l'algorithme de sous-gradient est le suivant :

- On part d'un gradient initialement nul.
  Multiplicateur de Lagrange π initialisé au vecteur nul
- On résout la relaxation Lagrangienne

- On détermine une direction de descente à l'aide des contraintes :
  - Si celle-ci est nulle, on a atteint le minimum local
  - Sinon, on résout le problème principal et met à jour la valeur du gradient.

De manière plus générale, au niveau de l'implémentation, l'algorithme se divise en deux grandes étapes :

1. Il faut résoudre la relaxation lagrangienne. La solution x^k obtenue est utilisée lors de l'initialisation de la méta-heuristique.

2. Puis on résoud le problème principal avec une méta-heuristique.
   On répète la procédure jusqu'à atteindre la solution optimale ou la condition d'arrêt.

5.3 Implémentation de la méthode de sous-gradient en Matlab

On cherche à résoudre dans cette question la relaxation continue du problème (1) à l'aide de l'algorithme de sous gradient en ne dualisant que la contrainte 3.

Algorithm 3 Subgradient Method

Require: f_obj, f_lagrange, g , π_0
  Initialize $theta = f\_lagrange(\pi)$
  Initialize $LB = -\infty, k = 0, \hat{\pi} = \pi^k, stop = false$
  while not stop **do**
    x, theta = min($f\_lagrange(\pi^k)$)
    for y_i in y **do**
      $y_i = g_i(x)$
    end for
    if $y == 0$ **then**
      stop = true
    end if
    Update π^k :
    Calculate λ_k (step size)
    $\pi^{k+1}(i) = \pi^k(i) + \lambda_k y(i)/||y||$
    $x, \hat{theta} = min(f\_lagrange(\pi^{k+1})$
    if $\hat{theta} == theta$ **then**
      Stop=true
    else if $theta \geq LB$ **then**
      LB=
    end if
    $k = k + 1$
  end while
  return x , LB

Malheureusement, notre code Matlab n'est pas assez optimisé pour présenter des résultats en un temps convenable. Nous avons arrêté l'exploration de cette approche et avons choisi de découvrir les différentes heuristiques présentées précédemment.

6 Conclusion

Dans ce projet nous avons découvert la résolution d'un problème d'optimisation avec des contraintes linéaires et une fonction objective non linéaire, non convexe et non différentiable. Pour résoudre le problème donné nous avons appris plusieurs méthodes qui donnent soit un résultat exact soit un résultat approché.

Dans un premier temps, nous avons commencé par l'algorithme de Warren-Smith qui a déjà été implémenté en C et qui donne une solution exacte du problème. Ce qui est intéressant à propos de cet algorithme est qu'il s'applique à n'importe quelle dimension donnée contrairement aux autres méthodes. Nous nous sommes rendu compte de la complexité de l'algorithme et de ses limites ; en effet, le temps de calcul en 3 dimensions augmente très rapidement en fonction du nombre de terminaux.
Nous avons ensuite essayé de résoudre le problème avec la relaxation lagrangienne en relâchant les contraintes de degré 3. Nous avons utilisé la méthode du sous-gradient implémentée en Matlab. Avec cette méthode on approche la solution à chaque itération de la relaxation. Le problème avec cette méthode est que le temps d'exécution du programme est trop grand (par rapport aux autres méthodes approchées) pour nous fournir des résultats intéressants.
Nous avons aussi découvert quelques heuristiques comme essaim d'abeilles et surtout l'algorithme de colonies de fourmis qui ont été implémentées sous python. Au vu du temps d'exécution, ces méthodes sont efficaces en comparaison avec la relaxation lagrangienne. En plus, les résultats obtenus par l'algorithme de fourmis sont assez précis par rapport aux résultats exacts. Concernant l'algorithme par essaim d'abeille, il ne nous fournit pas toujours une solution réalisable mais reste une méthode prometteuse à approfondir.

7 Organisation du travail

Pour le partage des fichiers, on a utilisé un google drive avec une adresse mail créée uniquement pour ce projet ainsi qu'un répertoire git pour mettre à jour les fichiers qui peuvent être modifiés par plusieurs personnes en même temps.
Les méthodes étudiées et les heuristiques ont été proposées par M. Ouzia dans les réunions hebdomadaires pendant le semestre. Ces choix nécessitaient le plus souvent un travail de recherche de la méthode la plus adaptée qui nous incitait tous à s'impliquer davantage de même que la compréhension des algorithmes à travers diverses documentations. En fonction du travail à effectuer on se concertait d'abord pour définir les étapes à suivre et se répartir les taches tout au long du projet.
Plus précisément Harold s'est consacré aux méthodes d'implémentation des méta-heuristiques en python et la visualisation des résultats. Ramatoulaye s'est chargée de la synthèse des documentations dans le rapport ainsi que la comparaison avec geosteiner et les tests de Warren Smith avec Paola et Mahshid. Ces dernières ont aussi établi la relaxation lagrangienne du problème sous Matlab et les tests de solveurs.

Références

[1] M. Brazil, D. A. Thomas, B. K. Nielson, P. Winter, C. Wulff-Nilsen, and M. Zachariasen. A novel approach to phylogenetic trees : d-dimensional geoemtric steiner trees. *Networks*, 53(2) :104–111, 2009.

[2] L. L. Cavalli-Sforza and A. W. F. Edwards. Phylogenetic analysis : Models and estimation procedures. *Evolution*, 21 :550–570, 1967.

[3] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1) :1–29, 1968.

[4] Garey M. R., Graham R. L., and D. S. Johnson. Complexity of computing steiner minimal trees. *SIAM J. Appl. Math.*, 32(4) :835–859, June 1977.

[5] Nelson Maculan, Philippe Michelon, and Adilson E. Xavier. The euclidean steiner tree problem in rn : A mathematical programming formulation. *Annals of Operations Research*, 96(1) :209–220, Nov 2000.

[6] F. Montenegro, J.R.A. Torreão, and N. Maculan. Microcanonical optimization for the euclidean steiner problem in $\mathbb{R}^n$ with application to phylogenetic inference. *Physical Review E*, 68 :056702–1– 056702–5, 2003.

[7] Markus Prossegger and Hamid Bouchachia. Ant colony optimization for steiner tree problems. pages 331–336, 01 2008.

[8] J. M. Smith, Y. Jang, and M. K. Kim. Steiner minimal trees, twist angles, and the protein folding problem. *Proteins : Structures, Functions, and Bioinformatics*, 66(4) :889–902, 2007.

[9] Warren D. Smith. How to find steiner minimal trees in euclideand-space. *Algorithmica*, 7(1) :137–177, Jun 1992.

[10] C. Stanton and J. M. Smith. Steiner trees and 3-d macromolecular conformation. *Informs journal on computing*, 16 :470–485, 2004.