Énumération d'arbres de Steiner évitant les topologies dégénérées

Rapport de projet

Projet final

Achille Baucher, Jérôme Bonacchi,
Bartolomé Heulin & Guillaume Renaut

sous la tutelle d'Hacène Ouzia

Polytech Sorbonne
Spécialité mathématiques appliquées et informatique
Année 5
2020 – 2021

L'ensemble de l'équipe remercie notre tuteur Hacène Ouzia pour l'aide et les conseils qu'il nous a apporté, aussi bien dans le choix de la bibliothèque d'optimisation RFBopt, des conseils graphiques sur l'outil de visualisation ou encore des idées d'heuristiques à employer.

Table des matières

Table des figures V

Liste des tableaux VII

Liste des algorithmes IX

1 Introduction 1

1.1 À l'origine : le problème de Fermat-Torricelli 1

1.2 Le problème de l'arbre de Steiner de longueur minimale 2

1.3 Complexité 4

1.4 Méthodes actuelles 4

1.5 Objectifs du projet 5

2 Réalisation 7

2.1 Interface graphique 7

2.1.1 Visualiser les topologies en trois dimensions . . . 7

2.1.2 Modeler les topologies . . . 7

2.1.3 Parcourir les énumérations . . . 9

2.2 Heuristique de H. Ouzia (« proche en proche ») 10

2.3 Heuristique de scindage hiérarchique 11

2.3.1 Intuition et justification de la méthode . . . 11

2.3.2 Répartition des terminaux dans des groupes . . . 12

2.3.2.1 Fonction de scindage . . . 12

2.3.2.2 Choix de la granularité . . . 13

2.3.3 Première méthode : granularité unitaire . . . 13

2.3.4 Deuxième méthode . . . 15

2.4 Optimisation avec RBFopt 16
2.5 Documentation 17

3 Résultats 19
3.1 Heuristique « de proche en proche » 19
3.2 Heuristique de scindage hiérarchique (granularité 2–3) 22

4 Conclusion 25
4.1 Bilan 25
4.2 Travail futur 26

A Annexes 27

Bibliographie 31

Table des figures

1.1 Problème de Fermat et sa résolution géométrique 2
1.2 Exemple d'une topologie dégénérée pour les terminaux du carré 3
1.3 Exemple d'une topologie optimale pour les terminaux du carré 3

2.1 Arbre de Steiner du cube 8
2.2 Affichage graphique interactif pour modeler des topologies 8
2.3 Affichage graphique pour l'énumération des arbres Steiner dans le cas du cube 10
2.4 Division hiérarchique des terminaux d'un cube, avec un scindage en deux groupes et une granularité 2 12
2.5 Illustration de l'algorithme chou-fleur dans le cas de cinq points 13
2.6 Arbres de Steiner optimaux pour un nombre de terminaux allant de un à cinq 14
2.7 Méthode n°1 pour trois points 14
2.8 Méthode n°1 pour le cube 14
2.9 Méthode n°2 pour la fusion d'un arbre de deux terminaux avec un arbre de trois terminaux 15

3.1 Énumération pour le cube avec l'heuristique « proche en proche » 20
3.2 Énumération pour l'octaèdre avec l'heuristique « proche en proche » 20
3.3 Énumération pour l'icosaèdre avec l'heuristique « proche en proche » 21
3.4 Arbre minimal pour le dodécaèdre 22
3.5 Énumération pour le cube avec l'heuristique de scindage hierarchique 23
3.6 Énumération pour l'octaèdre avec l'heuristique de scindage hiérarchique . . . 23

A.1 Diagramme de classe 27

Liste des tableaux

3.1 Comparaison des longueurs minimales pour chaque méthode et chaque solide platonique 22

Liste des algorithmes

2.1 Heuristique « proche en proche » 11
2.2 Fonction de scindage « chou-fleur » 13
2.3 Fusion de deux arbres de Steiner pour la première méthode 15
2.4 Fusion de deux arbres de Steiner pour la deuxième méthode 16

A.1 Version récursive de la méthode 1 RM1 28
A.2 Scindage hiérarchique récursif SHR 28
A.3 Étape de fusion pour la méthode 1 EFM1 29
A.4 Étape de fusion pour la méthode 2 EFM2 30

1. Introduction

Le projet que nous avons réalisé ce semestre a pour objet d'étude la recherche de l'arbre de Steiner de longueur minimale. Ce problème d'optimisation part d'un ensemble fixé de points dits *terminaux*, en dimension quelconque, formant une figure de départ, par exemple un triangle ou un cube. Le problème consiste à relier par des arêtes tous les points terminaux de la façon la plus courte possible. Pour ce faire, on peut ajouter des points supplémentaires, appelés points de Steiner. Il peut être montré que cet ensemble d'arêtes, de terminaux et de points de Steiner formant le plus cours réseau est un arbre, appelé arbre de Steiner, et en l'occurrence l'arbre de Steiner de longueur minimale.

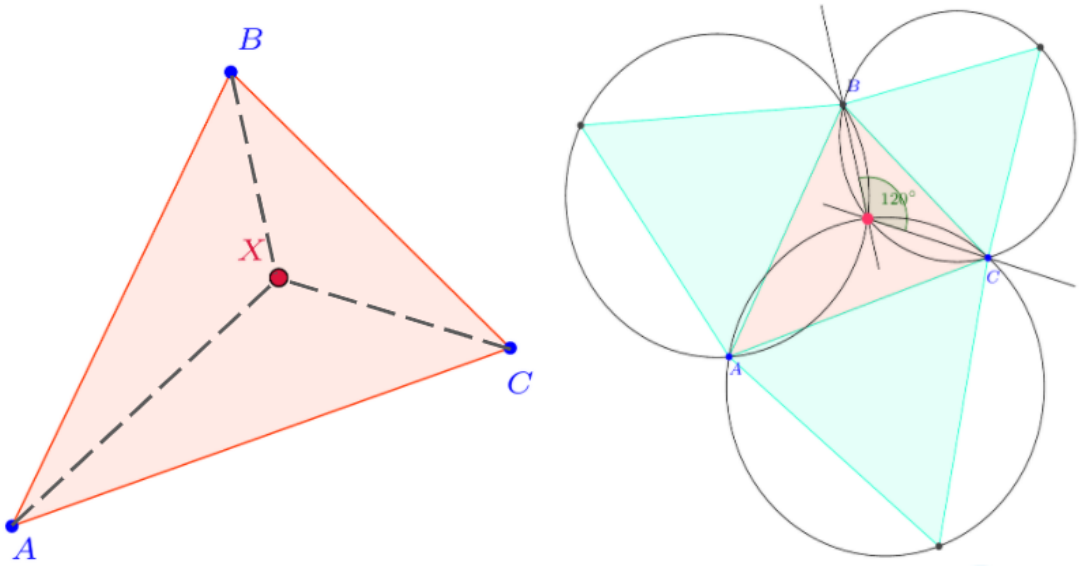
Les arbres de Steiner trouvent leur utilité dans des applications en conception de réseaux, notamment dans les circuits électroniques et les télécommunications, mais aussi dans les serveurs et les centres de calcul partagé. Il y a également des applications en biologie comme pour la modélisation de protéines [1].

1.1 À l'origine : le problème de Fermat-Torricelli

L'origine du problème est attribué à Pierre de Fermat qui au XVII^{e} siècle l'a posé dans une forme similaire à :

> *« Étant donnés trois points dans le plan, trouver un quatrième, tel que la somme des distances aux trois autres soit minimale. »*

Comme le montre la figure 1.1a, nous pouvons exprimer cela en disant qu'il faut trouver un point X dans un triangle ABC de manière à minimiser la somme des distances AX, BX et CX. Ce problème s'applique par exemple lorsque que l'on veut créer un chemin ou un canal qui relie trois villes. Evangelista Torricelli a décrit en 1640 une solution géométrique pour trouver les coordonnées de ce point X. Ce dernier porte le nom de « point de Fermat » ou « point de Torricelli » et sa construction est décrite sur la figure 1.1b.



(a) (b)

Figure 1.1 – Problème de Fermat (a) et sa résolution géométrique (b)

1.2 Le problème de l'arbre de Steiner de longueur minimale

Le problème de Fermat a ensuite été étendu à un nombre fini quelconque de points, donnant naissance au problème de l'arbre de Steiner. Ainsi, le problème de l'arbre de Steiner est la généralisation du problème de Fermat pour un nombre de points et de dimensions arbitraires. Il peut être montré qu'un point de Steiner est le point de Torricelli de son voisinage : le problème de Fermat est la version locale du problème de Steiner [2].

Par la suite, nous résumons certaines propriétés des arbres de Steiner issues, entre autres, de [3]. Une topologie est la configuration des points — les terminaux et les points de Steiner — et des connections sans tenir compte des coordonnées des points. Il a été démontré qu'un arbre de Steiner de longueur minimale possède $n - 2$ points de Steiner pour n terminaux — il y a des exceptions qui ne sont pas traitées dans ce rapport. De plus, tous les terminaux sont des feuilles de cet arbre. Dans celui-ci, les points de Steiner vérifient plusieurs propriétés :

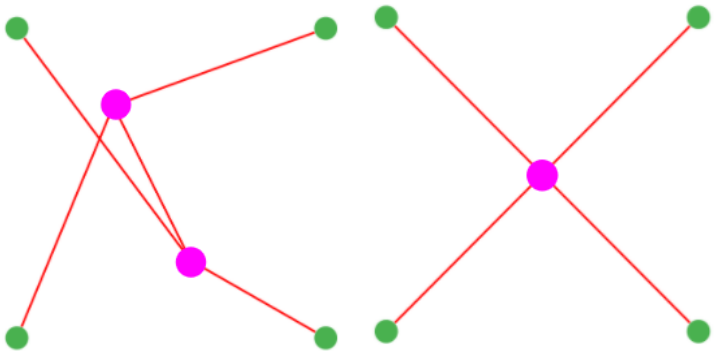
- ils sont liés à trois autres points ;
- leurs trois arêtes sont situées dans un même plan ;
- leurs trois arêtes forment des angles de 120° ;

Il est à noter qu'un arbre de Steiner de longueur minimale peut être partitionné en plusieurs — plus petits — arbres de Steiner disjoints [3].

Dans une approche énumérative, le but est alors de trouver dans un premier temps une topologie. Si celle-ci vérifie la première propriété ci-dessus alors elle est dite « de Steiner ». Dans un deuxième temps, on doit optimiser cette topologie pour que les arêtes aient une lon-

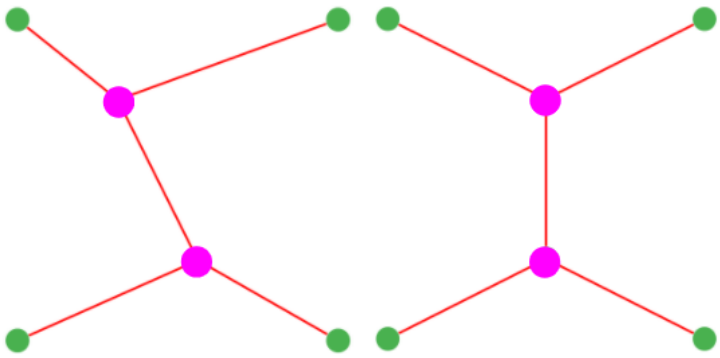
gueur totale minimale. Cela revient à calculer les coordonnées optimales des points de Steiner dans le but de minimiser la longueur totale des arêtes.

Si dans l'arbre de Steiner minimal associé à une topologie, au moins un point de Steiner se superpose à un autre point, la topologie est dite *dégénérée* — les arêtes de longueurs nulles et les points superflus doivent être supprimés. Par exemple, sur la figure 1.2a, on peut voir, sur la topologie T des chevauchements entre les arêtes. Lorsque l'on minimise cette topologie, les deux points de Steiner fusionnent et donnent ainsi un unique point avec quatre voisins. La topologie T est dégénérée contrairement à la topologie $T^\star$ de la figure 1.3b. Dans l'arbre de longueur minimale associé à la topologie $T^\star$, on a bien $n - 2 = 2$ point de Steiner, de degré trois et dont les arêtes forment des angles de 120°. Cette dernière topologie correspond à l'arbre de Steiner de longueur minimale pour le carré.



(a) Topologie dégénérée T (b) Arbre de Steiner de longueur minimale relatif à T

Figure 1.2 – Exemple d'une topologie dégénérée pour les terminaux du carré



(a) Topologie $T^\star$ (b) Arbre de Steiner de longueur minimale relatif à $T^\star$

Figure 1.3 – Exemple d'une topologie optimale pour les terminaux du carré

Dans le plan, il a été montré que l'arbre de longueur minimale, même avec un nombre de terminaux important — de l'ordre de plusieurs milliers, peut être trouvé en un temps raisonnable grâce à GeoSteiner [4]. En effet, les propriétés géométriques utilisées permettent alors

de réduire considérablement la quantité de topologies énumérées. En revanche, les propriétés de cet algorithme ne peuvent être appliquées aux instances de plus haute dimension.

1.3 Complexité

La principale difficulté pour résoudre le problème de l'arbre de Steiner de longueur minimale est que le nombre de topologies de Steiner croît extrêmement rapidement. D'ailleurs, Garey, Graham et Johnson ont prouvé que trouver l'arbre de longueur minimale est NP-dur [5]. Gilbert et Pollak ont montré [6] que le nombre de topologies avec n terminaux et k points de Steiner est :

$$\binom{n}{k+2}\frac{(n+k-2)!}{2^k k!}.$$

Puisque que nous nous intéressons aux topologies pouvant être optimales et que celles-ci contiennent $k = n-2$ points de Steiner, l'expression se simplifie en :

$$\frac{(2n-4)!}{2^{n-2}(p-2)!}.$$

Cette fonction est asymptotiquement en $O(2^n n^n)$. Le temps de calcul va donc très rapidement exploser : au-delà de huit terminaux, il est impossible ou très coûteux de tout énumérer.

1.4 Méthodes actuelles

Actuellement, les topologies peuvent être trouvées de trois manières différentes. Premièrement, en les énumérant, de manière plus ou moins exhaustive suivant les méthodes. En 1992, Smith publia une méthode d'énumération des topologies [7] — sans doute la première pour des topologies en au moins trois dimensions. Le souci principal de cette méthode est qu'elle n'a pas de critères suffisamment restrictifs lors de l'énumération des topologies, résultant en un nombre gigantesque de topologies dont la plupart sont dégénérées. Par la suite, une partie des algorithmes de la littérature s'est fondée sur ce schéma d'énumération et des versions améliorées ont été conçues comme celle présentée dans [8].

Deuxièmement, les topologies peuvent être trouvées en écrivant le problème comme un problème d'optimisation, pouvant ainsi être résolu par des solveurs existants. Pendant ce projet, nous nous sommes intéressés particulièrement à l'article de Ouzia et Maculan qui ont récrit le problème comme un nouveau modèle d'optimisation.

Troisièmement, les topologies peuvent être trouvées en utilisant des heuristiques tentant d'approcher avec des bornes plus ou moins contrôlées les topologies optimales. Ainsi, plusieurs

articles présentent des méthodes différentes dans le but de trouver la meilleure — sans qu'une ne soit meilleure que les autres à notre connaissance. Nous pouvons noter l'utilisation de la méthode *particle swarm* [10], ou bien l'utilisation de la triangulation de Delaunay ou des diagrammes de Voronoï [11], [12].

1.5 Objectifs du projet

Les méthodes actuelles d'énumération sont très coûteuses, notamment l'énumération de Smith qui est exhaustive, et nécessite encore l'optimisation de toutes les topologies énumérées. Notre but dans ce projet est de développer des heuristiques d'énumération de topologies plus efficaces. Nous allons notamment chercher à éviter l'énumération des topologies dégénérées, puisqu'on sait qu'elles ne seront pas optimales, alors qu'elles forment une grande partie de l'ensemble des topologies possibles. En revanche, comme nous n'énumérerons plus l'ensemble de toutes les topologies, il est possible que les topologies optimales ne fassent pas partie de notre énumération non-exhaustive. Nous chercherons néanmoins à ce que la plus courte topologie, parmi notre sélection, ait une longueur suffisamment proche de celle de la topologie optimale.

2. Réalisation

2.1 Interface graphique

L'une des étapes préliminaires de notre projet consistait à développer une interface graphique s'interfaçant avec notre code Python, afin de faciliter notre compréhension, de réaliser des tests et de visualiser nos résultats.

2.1.1 Visualiser les topologies en trois dimensions

Pour se représenter visuellement une topologie et un arbre de Steiner, achevé ou non, nous avions besoin d'un programme permettant de visualiser des points d'un espace euclidien en trois dimensions dans lesquels les homothéties et les rotations sont permises. Notre première approche a été d'utiliser la bibliothèque logicielle Plotly [13] qui permet de faire des graphiques qualitatifs et personnalisables. Suite à divers soucis avec les interfaces suivantes, nous avons décidé d'utiliser Matplotlib [14].

Les points sont représentés par des petites sphères et les liens par des segments. Nous avons représenté les points de Steiner et les terminaux par des couleurs différentes : rouge ou rose pour les premiers et vert pour les seconds. Nous avons aussi choisi de distinguer les arêtes liant deux points de Steiner à celles qui contiennent un point terminal avec ces mêmes couleurs, car ces deux types d'arêtes présentent des caractéristiques différentes. En effet, d'après les propriétés énoncées dans la section 1.2, les terminaux sont censés n'être que des feuilles de l'arbre de Steiner.

2.1.2 Modeler les topologies

Afin de tester pas à pas des idées d'algorithmes, et d'en observer directement le comportement, nous avons besoin de modeler à la main les topologies des arbres de Steiner. Pour ce faire, nous avons rendu l'affichage des topologies interactif, en ajoutant des boutons et la pos-

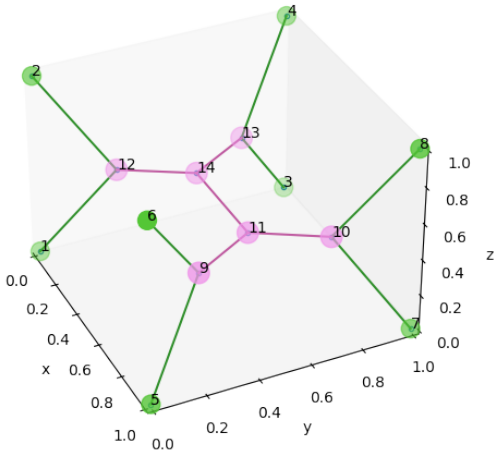


Figure 2.1 – Arbre de Steiner du cube

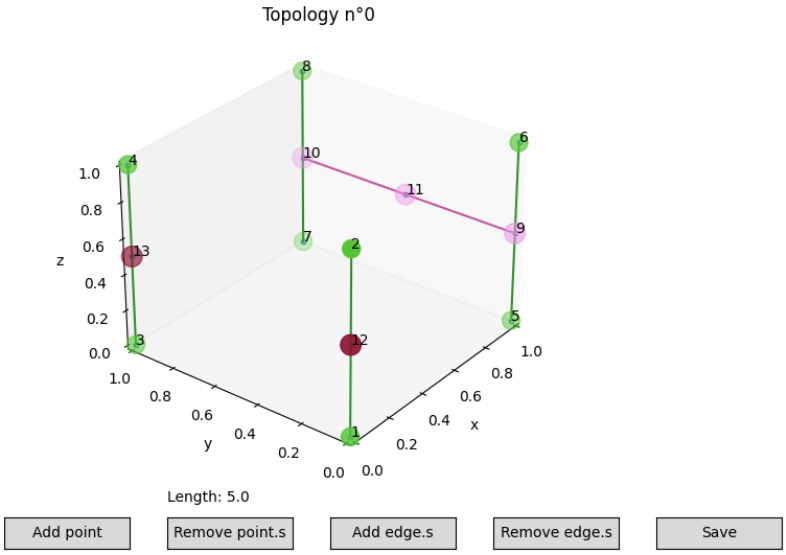


Figure 2.2 – Affichage graphique interactif pour modeler des topologies

sibilité de sélectionner des points sur le graphique. Pour modifier une topologie, nous avons ajouté les options d'ajout et de retrait de points et d'arêtes. En ce qui concerne l'ajout de points, rentrer les coordonnées du nouveau point serait peu visuel et obligerait à les calculer à chaque fois. Nous avons préféré l'option d'ajouter un point à l'isobarycentre des deux ou trois points sélectionnés : cas de figure qui revient souvent dans nos algorithmes de construction d'arbres de Steiner.

L'interface interactive a d'abord été réalisée avec la bibliothèque logicielle Streamlit [15]. Celle-ci permet d'ajouter des boutons et d'incruster un graphique dans un navigateur internet. Cependant, cette application ne fonctionnant pas chez l'un des membres du groupe, l'équipe a décidé de ne pas continuer de l'utiliser. Nous avons donc tenté de réalisé l'interface avec Dash Plotly [16]. Cette approche semblait prometteuse, mais pour des raisons que nous n'avons pas

examinées — il se peut que ce soit un manque d'optimisation de notre code — notre programme mettait du temps à afficher les graphiques. Nous avons ainsi décidé de ne pas continuer d'utiliser cette bibliothèque. Puisque notre code est en Python, et au vu de notre expérience, nous avons finalement décidé d'utiliser la bibliothèque logicielle Matplotlib [14]. D'une part, elle peut être utilisée pour réaliser les graphiques que nous souhaitons ; d'autre part, on peut créer une interface interactive avec des boutons. De plus, il existe une interface entre cette bibliothèque et les autres bibliothèques que nous utilisons et que nous présenterons par la suite.

2.1.3 Parcourir les énumérations

L'un des enjeux principaux de notre projet est l'énumération de différentes topologies. Chaque algorithme de construction d'un arbre de Steiner passe par une succession d'étapes bien définies au cours desquelles des modifications sont effectuées : ajouts de points de Steiner, ajouts et retraits d'arêtes. À chacune de ces étapes, il peut y avoir plusieurs modifications possibles de la topologie courante : c'est là que se produit l'énumération. Pour représenter cette énumération successive, la structure la plus naturelle nous a paru être celle d'un arbre, défini ci-dessous.

- **Racine :** la topologie initiale, ne contenant que les terminaux et aucun point de Steiner.
- **Nœud :** une étape de construction de l'arbre de Steiner.
- **Séparation :** chaque nœud parent se divise en plusieurs nœuds fils correspondant aux énumérations possibles de l'étape suivante.
- **Feuille :** un arbre de Steiner achevé, ou à un certain nombre d'étapes fixé.
- Le numéro de l'étape est sa distance à la racine.

Pour tester et comprendre le comportement concret des algorithmes implémentés, il est nécessaire de visualiser et de parcourir l'arbre des énumérations des étapes de construction des arbres de Steiner qui a été produit. Nous avons donc choisi de créer une interface contenant deux graphiques juxtaposés, comme montré sur la figure 2.3 : l'arbre des énumérations, à droite, et la topologie associée à une étape sélectionnée sur cet arbre, à gauche. Cet affichage permet de suivre visuellement chacune des étapes de l'algorithme et de parcourir les différentes branches d'énumération.

Un des enjeux rencontrés lors de l'énumération des arbres de Steiner est le caractère exponentiel de l'énumération : à chaque étape, chaque nœud se divise en plusieurs possibilités. Le résultat final d'un algorithme d'énumération contient rapidement beaucoup trop de topologies pour quelles soient représentées lisiblement sur un graphique. Pour palier ce problème lors des tests de nos algorithmes, nous avons ajouté une interaction supplémentaire sur l'arbre des énumérations. Cette option permet de ne pas lancer le solveur directement au démarrage, mais de n'énumérer que les topologies filles d'une topologie sélectionnée sur l'arbre des énumération. Nous pouvons ainsi arriver à construire un arbre de Steiner sans pour autant énumérer toutes les possibilités, en ne choisissant successivement que certaines branches à énumérer.

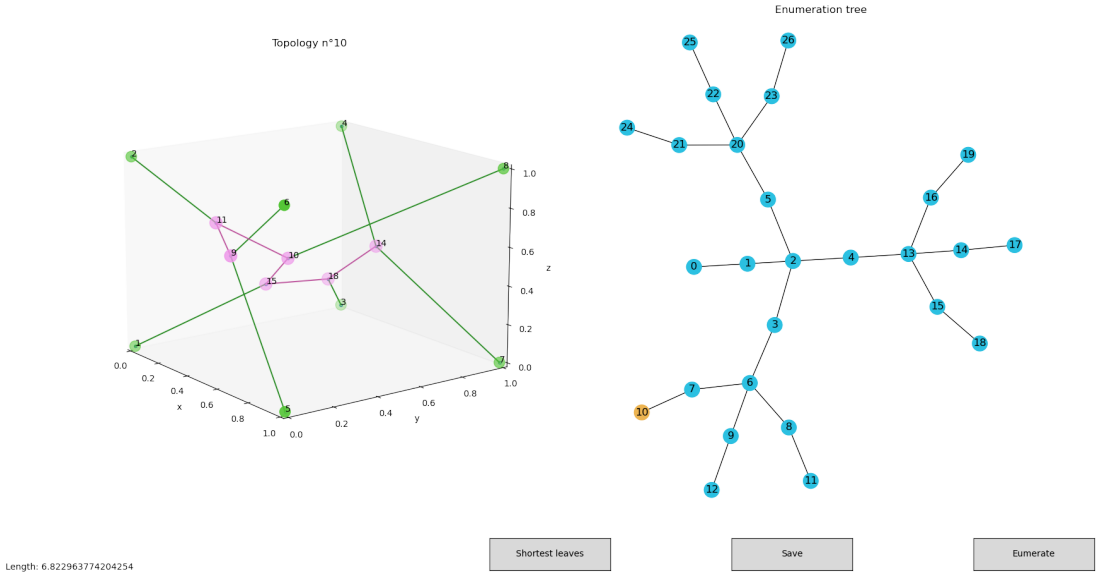


Figure 2.3 – Affichage graphique pour l'énumération des arbres Steiner dans le cas du cube

2.2 Heuristique de H. Ouzia (« proche en proche »)

Le but de cette heuristique est d'éviter les entrecroisements d'arêtes comme sur la figure 1.2a, car ceux-ci produisent des topologies dégénérées. Pour cela, nous partons d'un point terminal et avançons pas-à-pas vers les autres terminaux, de proche en proche. Ainsi, il y a peu de risque de revenir en arrière et de construire des arêtes entremêlées. L'algorithme 2.1 détaille cette heuristique.

Lors de l'ajout d'un point, nous avons le choix de ses coordonnées, c'est-à-dire que nous pouvons par exemple choisir entre le placer comme l'isobarycentre ou le point de Fermat. Nous avons choisi d'utiliser la première possibilité car moins coûteuse en calcul ; néanmoins, la deuxième possibilité est implémentée. À l'étape 7 de l'algorithme, on a le choix entre trois arêtes à retirer. Notre implémentation permet d'énumérer les différentes possibilités, afin de voir les différentes topologies résultantes possibles.

Algorithme 2.1 : Heuristique « proche en proche »

Entrées : Ensemble des terminaux
Sorties : Topologie (points terminaux, points de Steiner et arêtes entre ces points)

```
1 choix (au hasard) d'un point terminal ;
2 sélection des deux plus proches voisins de point;
3 ajout du point de Steiner current entre les 3 points;
4 tant que il reste des terminaux non reliés faire
5     sélection des deux plus proches voisins de current non reliés;
6     ajout d'un point de Steiner new entre les 3 points ;
7     current est de degré 4 : retrait d'une arête (parmi les 3 anciennes);
8     ajout d'un point de Steiner entre le point détaché, new et current;
9     current ← new;
```

2.3 Heuristique de scindage hiérarchique

2.3.1 Intuition et justification de la méthode

Il est assez facile de calculer des arbres de Steiner optimaux quand le nombre de points est assez faible — inférieur à cinq. Nous allons alors supposer dans cette heuristique qu'un arbre de Steiner sur un grand nombre de points peut être considéré comme une combinaison — potentiellement complexe — de plusieurs arbres de Steiner que nous appelons « élémentaires », c'est-à-dire calculés sur de petits groupes de points. Cela semble notamment pouvoir s'appliquer sur des topologies de terminaux où les points forment des groupes bien distincts. Néanmoins, nous supposons dans la suite que cela pourrait aussi fonctionner dans le cas où tous les points seraient proches. Orientés par H. Ouzia, nous avons pu déduire de cette supposition une heuristique pour la construction d'un arbre de Steiner composée des étapes :

1. réunir tous les points au sein de petits arbres élémentaires, dont on connaît la configuration optimale ;
2. combiner ces petits arbres entre eux, en suivant des règles à déterminer.

Si cette façon de construire un arbre était pertinente, ce serait très efficace car nous pourrions utiliser la connaissance des arbres élémentaires pour réduire l'exploration de toutes les énumérations, et ainsi gagner du temps. En effet, tous les groupes de moins de quatre ou cinq points seraient ainsi considérés comme un ensemble, et le nombre de configurations possibles pour relier ces ensembles entre eux est bien plus faible et rapide à calculer. De plus, nous supposons que relier ces ensembles de points peut être perçu comme un autre problème d'arbre de Steiner où il faudrait trouver l'arbre minimal pour relier ces ensembles. Ainsi, dans cette heuristique, nous pourrions utiliser un algorithme hiérarchique récursif qui appliquerait une méthode similaire pour relier d'abord des groupes de points, puis les arbres élémentaires ainsi formés, puis des arbres de plus en plus grands — et de moins en moins nombreux — jusqu'à relier tous les points du problème initial. Nous n'avons pas de justification théorique pour cette méthode mis à part les propriétés énoncées dans [3], cependant il nous paraît utile de la tester

tout de même avec notre outil car les résultats pourraient être intéressants.

2.3.2 Répartition des terminaux dans des groupes

Pour utiliser cette heuristique, il faut d'abord que chaque terminal appartienne à un petit groupe pour lequel va être créé un petit arbre de Steiner. Puis, il faudra que chaque groupe de terminaux soit associé à d'autres groupes de terminaux avec lesquels les arbres de Steiner seront combinés. Et ainsi de suite, jusqu'à ce que les groupes de groupes contiennent l'entièreté des terminaux. Nous remarquons ici le caractère hiérarchique des groupements. Nous avons donc décidé d'appliquer une division hiérarchique des terminaux. Elle consiste en une division de points en k groupes qui est appliquée hiérarchiquement jusqu'à ce que nous arrivons à la granularité minimale gr_{min}, c'est à dire le nombre de terminaux contenus dans un groupe en dessous duquel on arrête de diviser le groupe. Par exemple, avec une méthode utilisant $k = 2$ et $gr_{min} = 2$, nous scindons les groupes de terminaux en deux groupes jusqu'à que les groupes ne soient constitués plus que de deux terminaux. Sur une topologie avec huit terminaux, par exemple un cube, nous obtenons d'abord deux groupes de deux terminaux, puis deux groupes de deux dans chaque groupe de deux, comme illustré sur la figure 2.4.

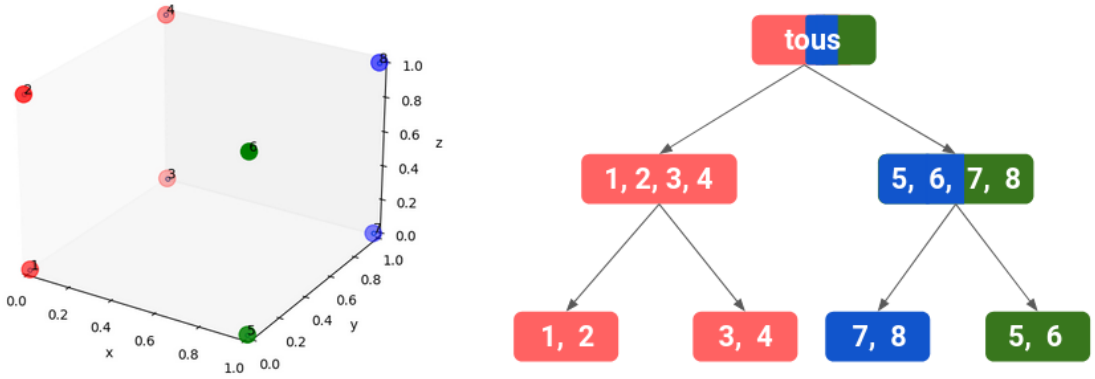


Figure 2.4 – Division hiérarchique des terminaux d'un cube, avec un scindage en deux groupes et une granularité 2

2.3.2.1 Fonction de scindage

La fonction de scindage doit séparer un groupe de terminaux en plusieurs groupes distincts. Nous avons choisi de commencer avec une séparation en deux groupes, par soucis de simplicité. Chacun des deux groupes obtenus par le scindage aura son propre arbre de Steiner élémentaire, et ceux-ci seront ensuite assemblés. Pour éviter au maximum des croisements à l'origine de topologies dégénérées, il faudrait que les deux groupes de terminaux soient bien séparés en terme de position. Pour ce faire nous avons développé un algorithme très simple, mais qui s'est révélé assez efficace sur nos instances. Le principe est de placer successivement dans les deux groupes le terminal le plus loin de l'autre groupe, et ainsi éviter les croisements.

Il est décrit formellement par l'algorithme 2.2 et illustré sur la figure 2.5 pour cinq points. La distance entre un point et un ensemble de points est ici mesurée comme la somme des distances du point aux autres.

Algorithme 2.2 : Fonction de scindage « chou-fleur »

Entrées : L'ensemble des terminaux
Sorties : Deux ensembles distincts de terminaux
1 *chou* et *fleur* sont des ensembles vides ;
2 *chou* ajoute un terminal au hasard ;
3 **tant que** *il reste des terminaux non attribués* **faire**
4 *fleur* choisit le terminal le plus loin des terminaux de *chou* ;
5 *chou* choisit le terminal le plus loin des terminaux de *fleur* ;
6 **retourner** *chou*, *fleur*

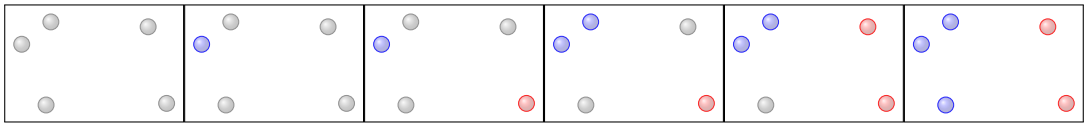


FIGURE 2.5 – Illustration de l'algorithme chou-fleur 2.2 dans le cas de cinq points. L'ensemble chou est bleu et fleur est rouge.

2.3.2.2 Choix de la granularité

La granularité est le nombre de terminaux en dessous duquel un groupe n'est pas divisé. La topologie optimale pour un arbre de STEINER avec ce nombre de terminaux doit être connue, ou très facile à calculer. Nous avons ainsi la certitude que nos arbres élémentaire sont optimaux, ce qui semble essentiel dans cette heuristique. En dessous de cinq terminaux, nous connaissons la topologie optimale pour l'arbre de STEINER — bien qu'il y ait plusieurs possibilités pour l'arbre optimal, comme illustré sur la figure 2.6. Nous n'avons donc essayé que des granularités inférieures ou égales à cinq.

Plus la granularité est petite, plus il y aura d'étapes de fusion, et donc d'énumérations dans le cas où l'on énumère plusieurs possibilités à chaque étape de fusion. Le choix de la granularité doit donc se faire eu égard à la quantité d'énumérations désirée, et de la capacité des topologies obtenues à être ou non dégénérées.

2.3.3 Première méthode : granularité unitaire

Nous avons d'abord implémenté la méthode la plus simple possible pour illustrer cette heuristique. Nous utilisons une granularité unitaire, c'est à dire que la division hiérarchique des groupes s'arrête lorsqu'un groupe ne contient plus qu'un seul point. Chaque groupe de

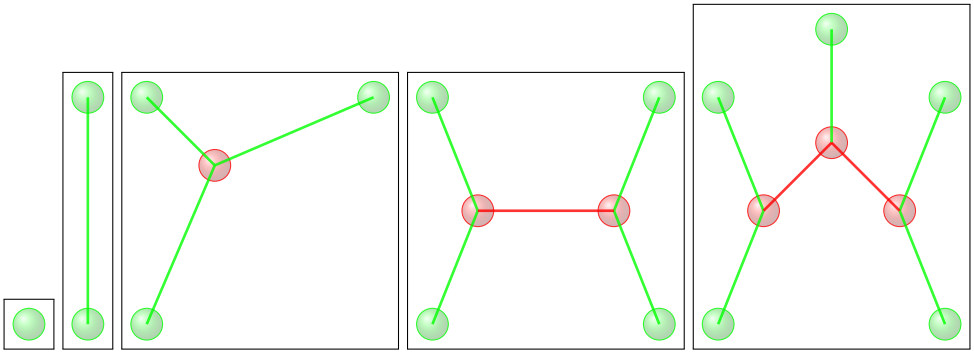


Figure 2.6 – Arbres de Steiner optimaux pour un nombre de terminaux allant de un à cinq

terminaux est représenté par un des point de son arbre de Steiner. La fusion de deux arbres s'effectue en ajoutant un nouveau point de Steiner connecté aux deux points représentants, et placé au milieu. Ce nouveau point devient alors le représentant du nouvel arbre fusionné. À l'initialisation, un groupe d'un point est représenté par ce même point. Comme démontré ci-dessous, les arbres de Steiner ainsi formés contiennent toujours un point de Steiner en trop, c'est à dire $n-1$ si le groupe contient n terminaux. Pour avoir le bon nombre de points de Steiner lors de la dernière fusion, nous ajoutons simplement une arrête entre les deux points représentants au lieu d'ajouter un point de Steiner.

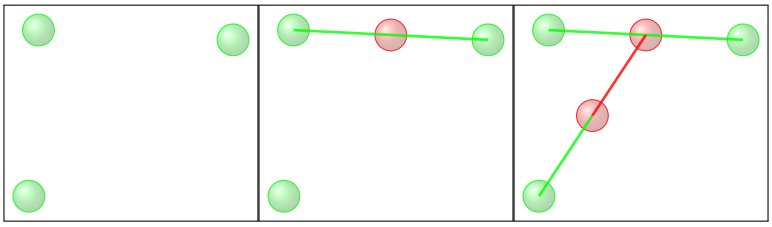


Figure 2.7 – Méthode n°1 pour trois points. De gauche à droite : trois terminaux, fusion des deux terminaux du haut, fusion du terminal du bas avec les terminaux du haut.

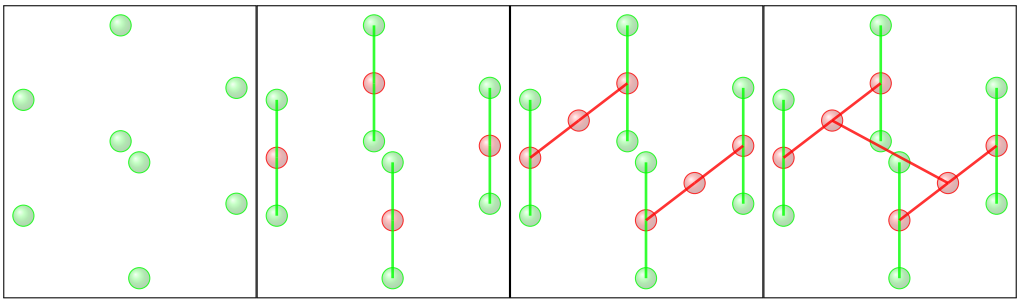


Figure 2.8 – Méthode n°1 pour le cube. De gauche à droite : initialisation, fusion des terminaux, fusion des groupes de deux, dernière étape.

Initialisation L'arbre de Steiner d'un terminal est simplement ce terminal, on a donc bien $0 = 1 - 1$ points de Steiner.

Hérédité On suppose qu'on a un groupe de p terminaux avec $p-1$ points de Steiner et un groupe de q terminaux avec $q-1$ points de Steiner. Lors de la fusion, on ajoute un

seul point de Steiner, et on a donc $(p-1)+(q-1)+(1) = p+q-1$ points de Steiner pour $p+q$ terminaux.

Algorithme 2.3 : Fusion de deux arbres de Steiner pour la première méthode

Entrées : deux arbres de Steiner avec un point en trop, leurs représentants
Sorties : un arbre de Steiner avec un point en trop, son représentant

1 Récupérer a et b les points représentants des deux arbres;
2 Créer un point de Steiner s au milieu a et b;
3 Connecter s à a et à b;
4 s devient le représentant du nouvel arbre de Steiner ;

2.3.4 Deuxième méthode

Nous présentons ici notre deuxième méthode pour assembler deux arbres de Steiner. Si les arbres contiennent p et q terminaux chacun, ils ont alors $p-2$ et $q-2$ points de Steiner respectivement. L'arbre fusionné contiendra $p+q$ terminaux et doit contenir $p+q-2$ points de Steiner, deux de ces points doivent donc être ajoutés lors de la fusion. Notre méthode lie deux arbres en ajoutant un nouveau point de Steiner sur une arête de chaque arbre, que nous appelons arête représentante, puis connecte ces nouveaux points, comme illustré sur la figure 2.9. Ajouter un point de Steiner sur une arête signifie enlever cette arête, et connecter le point de Steiner aux deux terminaux de l'arête. Le nouveau point est placé initialement au milieu des deux terminaux, même s'il sera amené à changer de position lors de la minimisation de l'arbre.

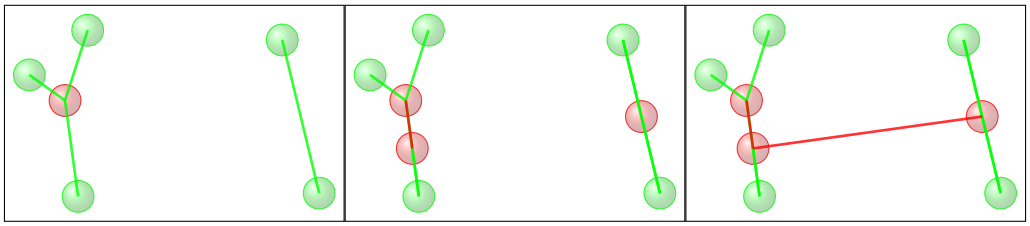


Figure 2.9 – Méthode n°2 pour la fusion d'un arbre de deux terminaux avec un arbre de trois terminaux. De gauche à droite : Les deux arbres de Steiner, ajout d'un point de Steiner au milieu des arêtes représentantes, fusion des deux arbres

La principale difficulté de cette méthode réside dans le choix des arêtes représentantes des deux arbres de Steiner à fusionner. Une première idée serait d'énumérer toutes les possibilités. Si un arbre de Steiner a p arêtes et l'autre q, on aura donc $p \times q$ fusions possibles. Le nombre d'énumération explose ainsi très rapidement. Pour pallier ce problème, il serait préférable d'introduire un critère de sélection. Une possibilité serait de choisir l'arbre de Steiner fusionné le plus court. Cependant, cela impliquerait de minimiser tous les arbres possibles à chaque nouvelle énumération, ce qui serait bien trop coûteux en temps de calcul. Un autre critère de sélection serait la taille de l'arbre non minimisé, et donc la taille de la nouvelle arête

Algorithme 2.4 : Fusion de deux arbres de Steiner pour la deuxième méthode

Entrées : A et B, deux arbres de Steiner
Sorties : une énumération d'arbres de Steiner

```
pour chaque arête (a1, a2) représentante de A faire
    pour chaque arête (b1, b2) représentante de B faire
        Créer un nouvel arbre de Steiner incomplet contenant A et B ;
        Créer as un point de Steiner au milieu de a1 et a2;
        Connecter as à a1 et a2;
        Créer bs un point de Steiner au milieu de b1 et b2;
        Connecter bs à b1 et b2;
        Connecter as et bs;
        Ajouter l'arbre obtenu aux énumérations;
```

puisque c'est la seule chose qui varie entre deux possibilités de fusion. C'est le critère que nous avons choisi, et à chaque énumération les m arbres les plus courts sont générés, avec m une limite arbitraire. D'autres critères de sélection auraient pu être intéressants à mettre en œuvre. Nous avons pensé par exemple à éviter les topologies dégénérées en ne sélectionnant pas les arêtes dont un terminal est *derrière* une autre arête, mais nous avons eu des difficultés à formaliser une notion géométrique pertinente de *derrière*.

Pour cette méthode, plusieurs granularités sont possibles, nous avons essayé avec cinq : nous obtenons des groupes minimaux de taille quatre et cinq ; puis avec trois, nous obtenons des groupes minimaux de taille deux et trois.

2.4 Optimisation avec RBFopt

Une fois les topologies énumérées avec nos heuristiques, nous avons besoin de les optimiser afin de détecter les topologies dégénérées et afin de mesurer la longueur de chaque arbre optimisé. Pour cela nous avons utilisé RBFopt [17]. RBFopt est une bibliothèque d'optimisation, conseillée par Hacène Ouzia qui est adaptée à notre situation puisqu'elle est capable d'optimiser un problème sans connaître sa fonction objectif sous une forme explicite. Elle n'a besoin que d'une fonction qui *évalue* l'objectif à chaque itération. Dans notre problème, l'objectif est de minimiser la longueur de l'arbre obtenu à partir d'une topologie.

Nous avons utilisé un objet prédéfini de RBFopt appelé « boîte noire » (*blackbox*), lequel prend un certain nombre d'arguments en paramètres :

- la dimension du problème ;
- les bornes inférieures et supérieures des variables, pour chaque dimension ;
- le type des variables — entier ou réel ;

— la fonction évaluation de l'objectif.

Il y a également un certain nombre de paramètres que l'on peut définir, notamment le nombre maximum d'itérations et le pas de précision à partir desquels la fonction s'arrête.

À chaque tour d'itération, la boîte noire va calculer un vecteur x, correspondant aux nouvelles coordonnées des points de Steiner, et demander à la fonction objectif de l'évaluer. Dans un premier temps, la fonction d'évaluation que nous avons définie va mettre à jour, dans la structure de données qui stocke l'arbre, les coordonnées des sommets. Dans un deuxième temps, la fonction va calculer la longueur de l'arbre et renvoyer cette valeur à la boîte noire.

2.5 Documentation

Puisque ce projet met en œuvre des méthodes nouvelles et pas simples à comprendre au premier abord, nous avons documenté soigneusement notre code. Nous avons ensuite généré une documentation avec l'utilitaire Sphynx [18] qui permet de la générer au format *pdf* et *html*. Par ailleurs, nous avons réalisé un diagramme de classe, figure A.1, présenté en annexe A.

3. Résultats

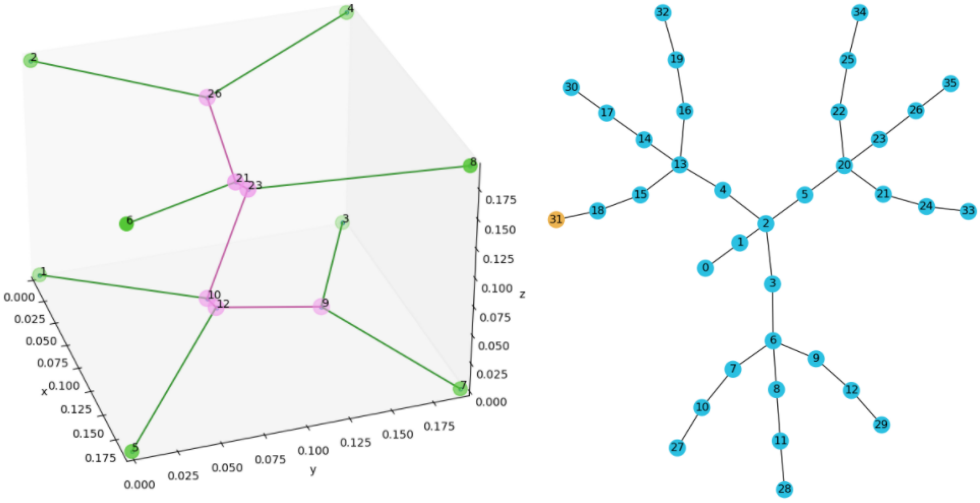
Pour tester nos méthodes, nous avons énuméré les topologies engendrées par nos heuristiques sur des solides platoniques : le cube, l'octaèdre, l'icosaèdre et le dodécaèdre. Nous avons utilisé les instances de solides platoniques d'Ouzia et Maculan dans [9]. Pour chaque méthode et chaque solide, nous avons compté le nombre de topologies dégénérées obtenues. Nous avons ensuite optimisé avec RBFopt les topologies avec les longueurs les plus courtes. Nous avons pris une tolérance de $1{,}0 \cdot 10^{-8}$ pour réaliser les optimisations. Les longueurs ont été comparées avec celles obtenues dans [9]. Plus précisément, nous avons comparé nos résultats avec la méthode $R_{1.1}$ (relaxation 1 du modèle 1) dont les résultats ont été obtenus après trois heures. Dans cette partie, nous avons décidé de présenter nos résultats pour l'heuristique « de proche en proche » et de scindage hiérarchique avec une granularité de 2-3.

3.1 Heuristique « de proche en proche »

Pour le cube, figure 3.1, les neuf topologies engendrées sont dégénérées. La topologie optimale n'est donc pas engendrée. La topologie minimale obtenue a une longueur de 1,209 478 ce qui est 1,004 fois plus grand que celle obtenue dans [9]. Notons que notre topologie minimale est dégénérée car certaines propriétés énoncées dans la partie 1.2 ne sont pas respectées. Certains points de Steiner semblent fusionner. Contrairement à notre arbre minimal, l'arbre d'Ouzia et Maculan ne semble pas quant à lui dégénérée, les propriétés semblent vérifiées.

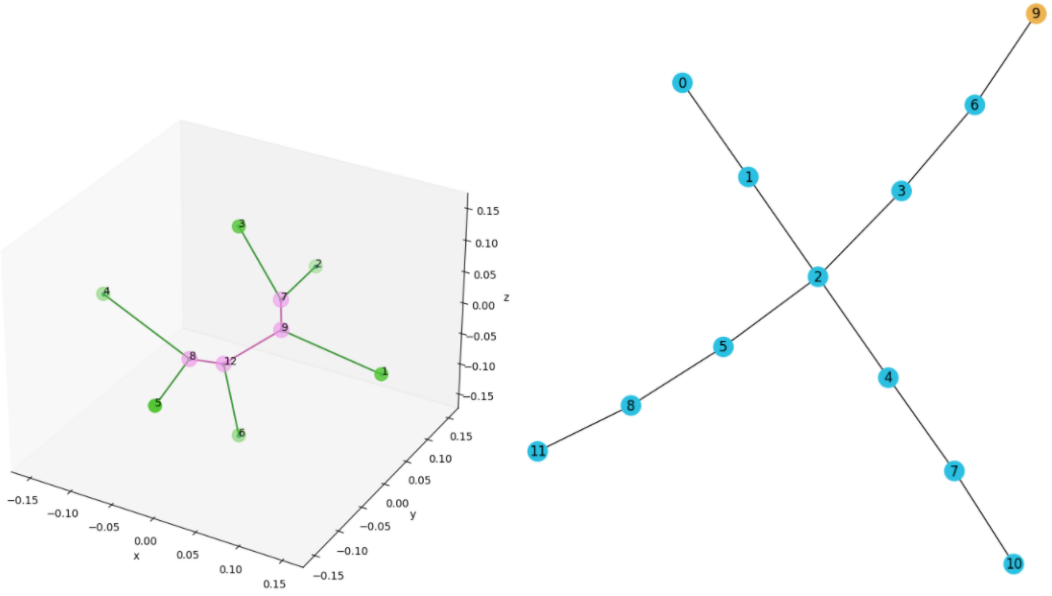
Pour l'octaèdre, figure 3.2, aucune des trois topologies engendrées ne semblent dégénérées. L'arbre de longueur minimale a une longueur de 0,956 004 ce qui est 1,014 fois plus court que celui obtenu dans [9].

Pour l'icosaèdre, figure 3.3, nous n'avons optimisé qu'une seule topologie : la topologie avec la plus petite longueur avant optimisation. Nous avions obtenu 81 topologies, ainsi toutes les optimiser aurait pris du temps. La longueur de la topologie minimale est de 1,666 935 après optimisation ce qui est 1,011 fois plus long que celle obtenue dans [9].



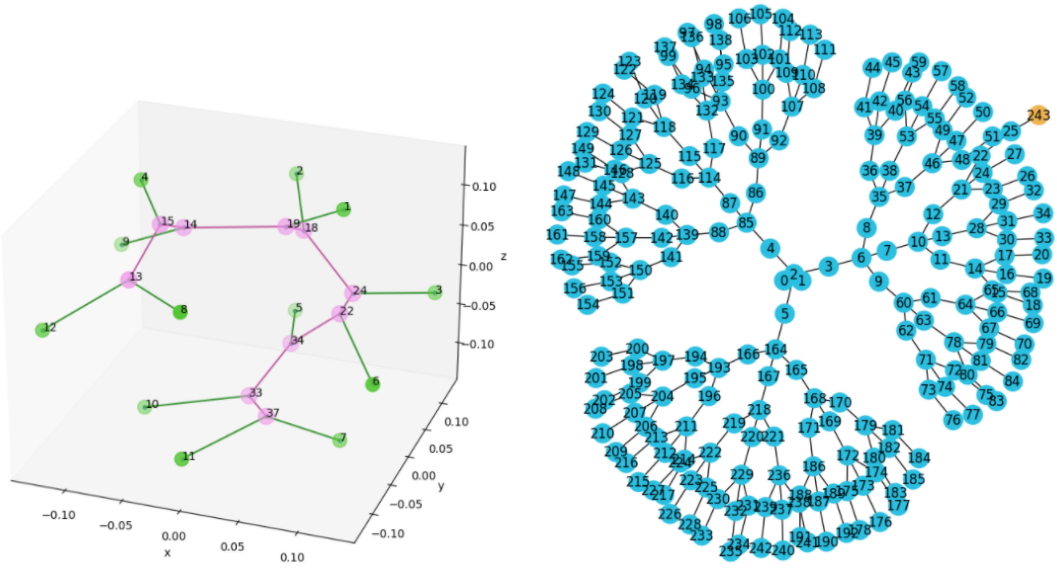
(a) Arbre minimal (b) Arbre d'énumération

Figure 3.1 – Énumération pour le cube avec l'heuristique « proche en proche »



(a) Arbre minimal (b) Arbre d'énumération

Figure 3.2 – Énumération pour l'octaèdre avec l'heuristique « proche en proche »



(a) Arbre minimal (b) Arbre d'énumération

FIGURE 3.3 – Énumération pour l'icosaèdre avec l'heuristique « proche en proche »

Pour le dodécaèdre, figure 3.4, nous n'avons pas affiché l'arbre d'énumération car il y avait trop de topologies engendrées — 6 561 exactement. Nous avons donc optimisé seulement la topologie avec la plus petite longueur avant optimisation. La topologie minimale a une longueur de 2,372 723 ce qui est 1,039 fois plus long que celle dans [9].

Il est d'une grande importance de noter que pour obtenir nos énumérations et nos arbres minimaux, il ne nous fallait pas plus de trente minutes dans le pire des cas, et nous obtenons des valeurs assez proches de celles de [9].

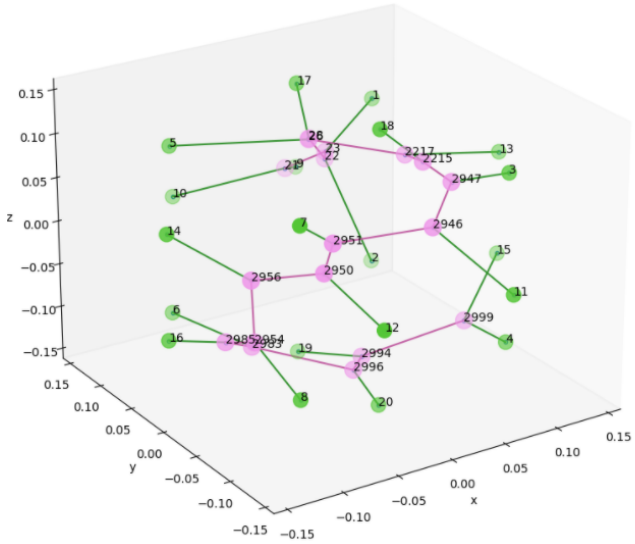


Figure 3.4 – Arbre minimal pour le dodécaèdre

3.2 Heuristique de scindage hiérarchique (granularité 2–3)

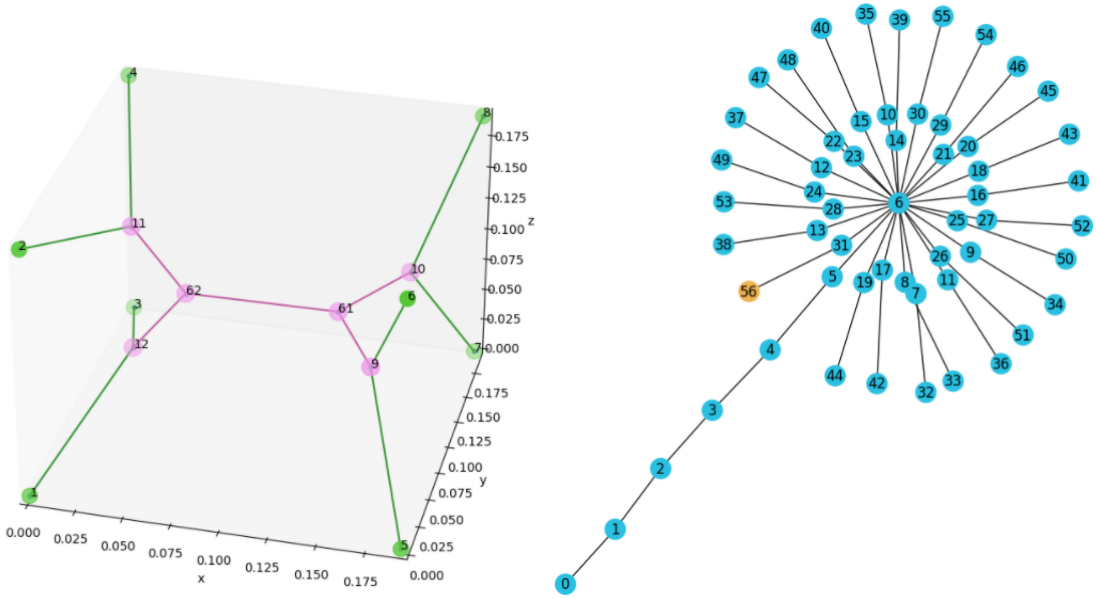
Pour le cube, figure 3.5, on obtient 23 topologies et seule une d'entre elles semble non dégénérée. Celle-ci semble correspondre à l'arbre optimal. Sa longueur est de 1,192 504 ce qui est 1,002 6 fois plus court que celle obtenue dans [9].

Pour l'octaèdre, figure 3.6, comme pour l'heuristique de « proche en proche », nous obtenons neuf topologies et aucune d'entre elles ne semble dégénérée. La topologie minimale a une longueur de 0,956 005 ce qui est 1,014 fois plus court que celle dans [9].

Nous pouvons remarquer que cette heuristique de scindage hiérarchique énumère plus de topologies que l'heuristique de « proche en proche ». Par conséquent, son temps d'énumération est plus long. Le tableau 3.1 résume tous les résultats obtenus pour chaque méthode et chaque solide platonique.

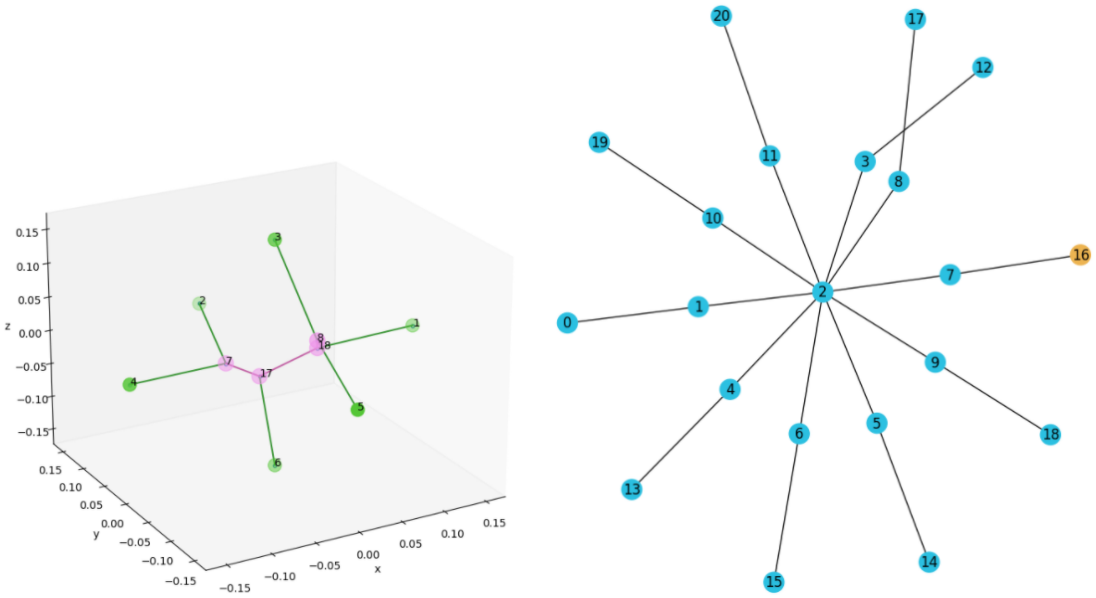
Polyèdre	« proche en proche »	scindage hiérarchique	Ouzia et Maculan [9]
Octaèdre	0,956 004	0,956 005	0,969 439
Cube	1,209 478	1,192 504	1,204 798
Icosaèdre	1,666 935	-	1,633 776
Dodécaèdre	2,372 723	-	2,283 626

Tableau 3.1 – Comparaison des longueurs minimales pour chaque méthode et chaque solide platonique



(a) Arbre minimal (b) Arbre d'énumération

FIGURE 3.5 – Énumération pour le cube avec l'heuristique de scindage hierarchique



(a) Arbre minimal (b) Arbre d'énumération

FIGURE 3.6 – Énumération pour l'octaèdre avec l'heuristique de scindage hiérarchique

4. Conclusion

4.1 Bilan

Ce projet étudiait les arbres de Steiner avec, comme objectifs principaux, de proposer des heuristiques d'énumération plus rapides, d'éviter les topologies dégénérées, d'obtenir des résultats proches des optimaux attendus. Au cours de ce projet, nous avons pu produire deux grands types d'énumérations, ainsi qu'un outil graphique.

L'outil graphique est fonctionnel et à été décliné en trois variantes selon l'utilisation souhaitée : observation simple, manipulation de topologie et visualisation des énumérations.

Les résultats produits par les heuristiques testées sont encourageants. Concernant la première heuristique, on peut voir que l'on a fortement réduit le nombre de topologies énumérées. La proportion de dégénérées dans l'énumération varie suivant le solide étudié, mais même dans le cas du cube où il n'y a que des topologies dégénérées, celles-ci donnent finalement une longueur de topologie très proche de la valeur optimale attendue. Ainsi, même dans le cas où nous énumérons des topologies dégénérées, il semble que les arbres qui en résultent soient « acceptables ».

Concernant la deuxième heuristique, elle aussi réduit fortement le nombre de topologies énumérées. Elle a permis d'obtenir la meilleure topologie possible dans le cas du cube comme de l'icosaèdre. On peut donc supposer qu'elle est plus généralisable que la première heuristique, qui avait donné un résultat très différent pour ces deux solides en terme de topologies dégénérées.

De manière générale ces deux heuristiques ont donné de bons résultats, proches des optimaux existants dans le cas de l'octaèdre et du cube, en un temps bien plus court que l'énumération de Smith et en énumérant beaucoup moins de topologies. De même, les résultats sont obtenus plus rapidement qu'avec la méthode d'optimisation d'Hacène Ouzia [9]. Les objectifs de ce projet ont donc été remplis.

4.2 Travail futur

Bien que nous ayons développé des idées intéressantes pour résoudre — au moins de manière approchée — des instances simples d'arbres de Steiner, il reste de nombreuses tâches que nous n'avons pas pu mener. Tout d'abord, nous souhaiterions comparer plus en détail les topologies créées par nos méthodes et celles créées par l'énumération de Smith [7]. Plus précisément, il faudrait comparer la durée d'énumération, la longueur de l'arbre le plus court ou encore le ratio de topologies dégénérées. Puis, nous aimerions faire de même avec la méthode d'Ouzia et Maculan, ainsi que d'autres méthodes comme celles présentées dans [8], [11], [12], [19]-[22].

Ensuite, un notre point capital sur lequel il faudrait s'attarder est l'optimisation de notre code. En effet, nous avons fait face à ses limites lorsque nous avons essayé d'exécuter notre dernière méthode sur le dodécaèdre et qu'il en était incapable. L'optimisation concernerait la réduction du coût en temps et en mémoire. Par ailleurs, pour que notre méthode soit la plus efficace possible, nous devons trouver des critères permettant de dissocier des topologies non dégénérées — voire optimales dans l'idéal — des topologies dégénérées. De tels critères pourraient permettre de détecter une topologie en cours de création dont on saurait d'avance quelle ne sera pas pertinente. En concevant une énumération à partir de ces critères, nous pourrions réduire le nombre de topologies énumérées, par la génération de coupes. En outre, nous pensons que, de part son fonctionnement, RBFopt n'est pas l'outil le plus adapté pour optimiser les topologies et qu'il faudrait donc trouver un programme plus efficace. De plus, notre interface graphique fondée sur Matplotlib a montré ses limites en terme de performance, de personnalisation et de commodité. Ainsi, nous devrions utiliser un outil plus puissant et plus personnalisable. Plusieurs possibilités comme Dash VTK [23], Mayavi [24] ou bien PyVista [25] seraient à étudier.

A. Annexes

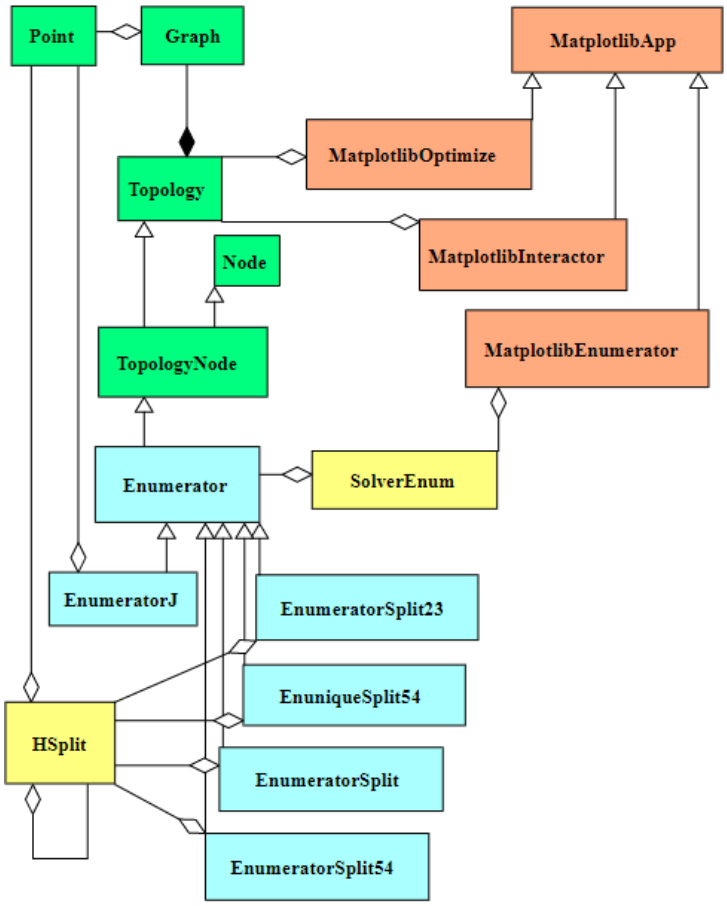


FIGURE A.1 – Diagramme de classe

L'algorithme A.1 est une version récursive de la première méthode.

L'algorithme A.2 forme un arbre de scindage de terminaux, et marque les nœuds associés aux groupes possédant moins de terminaux que la granularité indiquée commes des feuilles.

L'algorithme A.3 est une étape de la version non récursive de la méthode n°1.

Algorithme A.1 : Version récursive de la méthode 1 RM1

Entrées : Ensemble E des terminaux, booléen `dernière_étape` indiquant si c'est la dernière étape
Sorties : Point représentant de l'arbre

```
si |E| = 1 alors
    v = seul élément de E;
    retourner v;
Scinder E en deux groupes E1 et E2;
v1 = RM1(E1, dernière_étape=Faux);
v2 = RM1(E2, dernière_étape=Faux);
si dernière_étape alors
    Connecter v1 et v2;
    Fin de l'algorithme;
sinon
    Créer un point de Steiner s au milieu de v1 et de v2;
    Lier s à v1 et à v2;
    retourner s;
```

Algorithme A.2 : Scindage hiérarchique récursif SHR

Entrées : Ensemble E des terminaux, granularité g
Sorties : Racine de l'arbre de scindage

```
Créer un nœud N associé à l'ensemble E;
si |E| ≤ g alors
    Marquer N comme une feuille;
    retourner N;
Scinder E en deux groupes E1 et E2;
N1 = SHR(E1, g);
N2 = SHR(E2, g);
Associer N à ses enfants N1 et N2;
retourner N;
```

Algorithme A.3 : Étape de fusion pour la méthode 1 EFM1

Entrées : Noeud N de l'arbre de scindage

si N *est une feuille* **alors**
    EFM1(parent(N));
sinon si *Un enfant* N_x *de* N *n'est pas une feuille* **alors**
    EFM1(N_x);
sinon si *Les deux enfants* N_a *et* N_b *de* N *sont des feuilles* **alors**
    Récupérer v_a et v_b les représentants de N_a et N_b;
    si N *est la racine de l'arbre de scindage* **alors**
        Connecter v_a et v_b;
        Fin de l'algorithme;
    sinon
        Créer un point de Steiner s au milieu de v_a et de v_b;
        Lier s à v_a et à v_b;
        Marquer s comme représentant de N;

Pour la méthode 2, décrite dans l'algorithme A.4, l'implémentation est plus complexe, car on doit énumérer différentes possibilités et former un arbre d'énumération. Ainsi, un nœud de l'arbre de scindage, correspondant à un groupe de terminaux, peut être associés à plusieurs arbres de Steiner possibles, et donc à plusieurs types de représentants selon la branche sur laquelle on se trouve dans l'arbre des énumérations. Donc, chaque nœud de l'arbre d'énumération doit posséder ses propres représentants des groupes. Il doit aussi être associé à un nœud de l'arbre de scindage, dont il s'occupe de la fusion des deux groupes.

Algorithme A.4 : Étape de fusion pour la méthode 2 EFM2

Entrées : Noeud N_{enum} de l'arbre d'énumération et son nœud associé N_{scind} de l'arbre de scindage

```
Soit E l'ensemble des terminaux associé à N_scind;
si |E| ≤ 3 alors
    Créer un nœud fils N_e de N_enum, avec les mêmes représentants et feuilles;
    si |E| = 2 alors
        Dans N_e, connecter les deux terminaux de E avec une arête e;
        N_e marque e comme seule arête représentante pour N_scind ;
    sinon si |E| = 3 alors
        Dans N_e, créer un point de Steiner s au barycentre des points de E;
        Connecter s aux trois points de E;
        N_e marque comme arêtes représentantes de N_scind les trois arêtes formées;
    N_e marque N_scind comme feuille ;
si N_enum marque N_scind comme une feuille alors
    Associer parent(N_scind) à N_enum;
    EFM2(N_enum);
sinon si Un enfant N_s de N_scind n'est pas marqué comme une feuille par N_enum alors
    Associer N_s à N_enum;
    EFM2(N_enum);
sinon si Les deux enfants N_a et N_b de N_scind sont des feuilles alors
    Récupérer l_a et l_b les ensembles d'arêtes représentantes que N_enum attribue à N_a et
     N_b;
    pour (a_1, a_2) dans les arêtes l_a faire
        pour (b_1, b_2) dans les arêtes l_b faire
            Créer un nœud fils N_e de N_enum, avec les mêmes représentants et feuilles;
            Dans N_e, créer un point de Steiner v_a au milieu de a_1, a_2;
            Dans N_e, lier v_a à a_1 et à a_2;
            Dans N_e, créer un point de Steiner v_b au milieu de b_1, b_2;
            Dans N_e, lier v_b à b_1 et à b_2;
            N_e marque N_scind comme une feuille ;
            N_e marque l'ensemble des arêtes de son arbre de Steiner comme
             représententantes de N_scind;
```

Bibliographie

[1] J. M. Smith, Y. Jang et M. K. Kim, « Steiner minimal trees, twist angles, and the protein folding problem, » eng, *Proteins, structure, function, and bioinformatics*, t. 66, n° 4, p. 889-902, 2007, issn : 0887-3585.

[2] D. Cieslik, *Steiner Minimal Trees*. Springer Science+Business Media Dordrecht : Springer US, 1998, isbn : 978-0-7923-4983-9. doi : 10.1007/978-1-4757-6585-4. (visité le 21/02/2021).

[3] M. Fampa, « Insight into the computation of Steiner minimal trees in Euclidean space of general dimension, » eng, *Discrete Applied Mathematics*, 2019, issn : 0166-218X.

[4] D. Warme, P. Winter et M. Zachariasen. (2 jan. 2017). « GeoSteiner, » adresse : http://www.geosteiner.com (visité le 21/02/2021).

[5] M. R. Garey, R. L. Graham et D. S. Johnson, « The Complexity of Computing Steiner Minimal Trees, » eng, *SIAM journal on applied mathematics*, t. 32, n° 4, p. 835-859, 1977, issn : 0036-1399.

[6] E. N. Gilbert et H. O. Pollak, « Steiner Minimal Trees, » eng, *SIAM journal on applied mathematics*, t. 16, n° 1, p. 1-29, 1968, issn : 0036-1399.

[7] W. D. Smith, « How to find Steiner minimal trees in euclidean d -space, » *Algorithmica*, t. 7, n° 1, p. 137-177, 1er juin 1992, issn : 1432-0541. doi : 10.1007/BF01758756. (visité le 21/02/2021).

[8] R. Fonseca, M. Brazil, P. Winter et M. Zachariasen, « Faster Exact Algorithms for Computing Steiner Trees in Higher Dimensional Euclidean Spaces, » 2014.

[9] H. Ouzia et N. Maculan, « Mixed Integer Nonlinear Optimization Models for the Euclidean Steiner Tree Problem in R^d, » working paper or preprint, sept. 2019, adresse : https://hal.sorbonne-universite.fr/hal-02293105.

[10] W. W. Costa, M. L. Rocha, D. N. Prata et P. L. Moreira, « Application of Enhanced Particle Swarm Optimization in Euclidean Steiner Tree Problem Solving in RN, » in *Computational Intelligence, Optimization and Inverse Problems with Applications in Engineering*, Cham, Switzerland : Springer, 26 sept. 2018, p. 63-85, isbn : 978-3-319-96432-4. doi : 10.1007/978-3-319-96433-1_4. (visité le 21/02/2021).

[11] J. E. Beasley et F. Goffinet, « A delaunay triangulation-based heuristic for the euclidean steiner problem, » *Networks*, t. 24, n° 4, p. 215-224, 1er juil. 1994, issn : 0028-3045. doi : 10.1002/net.3230240404. (visité le 21/02/2021).

[12] J. W. Van Laarhoven et J. W. Ohlmann, « A randomized Delaunay triangulation heuristic for the Euclidean Steiner tree problem in $\mathfrak{R}$ d, » *J. Heuristics*, t. 17, n° 4, p. 353-372, 1er août 2011, issn : 1572-9397. doi : 10.1007/s10732-010-9137-z. (visité le 21/02/2021).

[13] plotly. (18 fév. 2021). « Plotly Python Graphing Library, » adresse : https://plotly.com/python (visité le 21/02/2021).

[14] J. D. Hunter, « Matplotlib : A 2D graphics environment, » *Computing in Science & Engineering*, t. 9, n° 3, p. 90-95, 2007. doi : 10.1109/MCSE.2007.55.

[15] streamlit. (21 fév. 2021). « Streamlit — The fastest way to create data apps, » adresse : https://www.streamlit.io (visité le 21/02/2021).

[16] plotly. (21 fév. 2021). « Dash Overview, » adresse : https://plotly.com/dash (visité le 21/02/2021).

[17] A. Costa et G. Nannicini, « RBFOpt : an open-source library for black-box optimization with costly function evaluations, » *Mathematical Programming Computation*, t. 10, août 2018. doi : 10.1007/s12532-018-0144-7.

[18] Sphinx Authors. (21 fév. 2021). « Sphinx, » adresse : https://www.sphinx-doc.org/en/master (visité le 21/02/2021).

[19] A. Grodet et T. Tsuchiya, « Reorganizing topologies of Steiner trees to accelerate their elimination, » *arXiv*, 11 nov. 2015. eprint : 1511.03407. adresse : https://arxiv.org/abs/1511.03407v4 (visité le 21/02/2021).

[20] L.-A. Gottlieb et Y. Bartal, « Near-linear time approximation schemes for Steiner tree and forest in low-dimensional spaces, » *arXiv*, 7 avr. 2019. eprint : 1904.03611. adresse : https://arxiv.org/abs/1904.03611v1 (visité le 21/02/2021).

[21] V. L. do Forte, F. M. T. Montenegro, J. A. de Moura Brito et N. Maculan, « Iterated local search algorithms for the Euclidean Steiner tree problem in n dimensions, » *Int. Tran. Oper. Res.*, t. 23, n° 6, p. 1185-1199, 1er nov. 2016, issn : 0969-6016. doi : 10.1111/itor.12168. (visité le 21/02/2021).

[22] M. Fampa et K. M. Anstreicher, « An improved algorithm for computing Steiner minimal trees in Euclidean d-space, » *Discrete Optimization*, t. 5, n° 2, p. 530-540, 2008, In Memory of George B. Dantzig, issn : 1572-5286. doi : https://doi.org/10.1016/j.disopt.2007.08.006. adresse : https://www.sciencedirect.com/science/article/pii/S1572528607000473.

[23] plotly. (21 fév. 2021). « dash-vtk, » adresse : https://github.com/plotly/dash-vtk (visité le 21/02/2021).

[24] P. Ramachandran et G. Varoquaux, « Mayavi : 3D Visualization of Scientific Data, » *Computing in Science & Engineering*, t. 13, n° 2, p. 40-51, 2011, issn : 1521-9615.

[25] C. B. Sullivan et A. Kaszynski, « PyVista : 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK), » *Journal of Open Source Software*, t. 4, n° 37, p. 1450, mai 2019. doi : 10.21105/joss.01450. adresse : https://doi.org/10.21105/joss.01450.