Déploiement de fibres optiques

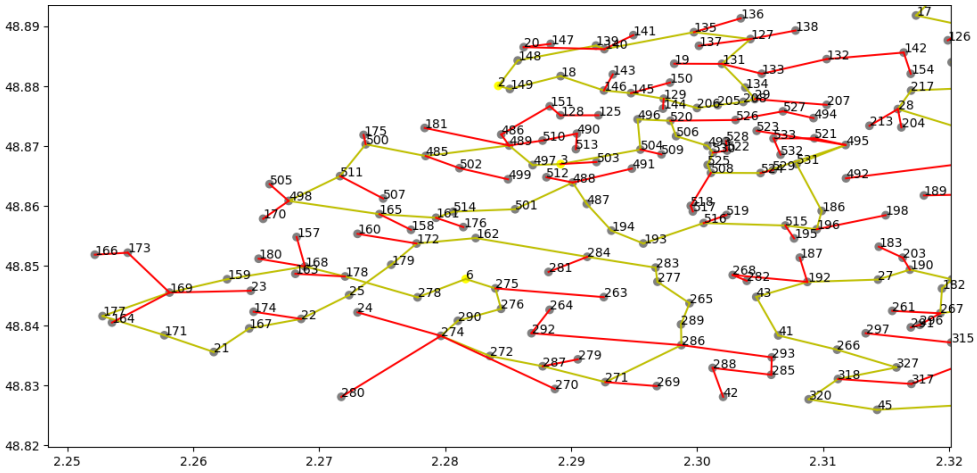
Hugo LAURENÇON, Clément LASUEN, Quentin SPINAT

Janvier 2019

1 Visualisation des données

Avant de programmer des heuristiques qu'on espère relativement performantes, il a fallu acquérir de l'intuition afin de voir quel type de solution donnait des bons scores.

Une première chose a été de visualiser les solutions pour obtenir une idée de la répartition des points de distribution parmi les antennes. Nous avons réalisé un programme en python pour ceci, et dont le résultat pour Paris est présent ci-dessous (image coupée pour la visibilité).



2 Choix des chaînes

On s'est aperçu que le choix des boucles était déterminant, et qu'une fois ces boucles choisies, on peut déterminer les chaînes convenablement de façon algorithmique. Pour cela :

• Récupérer la liste des sommets qui ne sont pas déjà impliqués dans une boucle ;

• Ordonner la liste de cette façon : On prend le premier élément. On parcourt toute la liste à partir du deuxième élément et on note le sommet tel que la distance pour passer du premier élément à lui est minimale. On met ce sommet en deuxième position. On recommence pour le troisième élément et ceci jusqu'à la fin de la liste. On met en fait à la suite des sommets qui sont proches entre eux ;

• Pour le premier sommet s_1 de la liste triée, on parcourt tous les sommets de toutes les boucles et on note l'élément b_1 tel que la distance pour passer de b_1 à s_1 est minimale. On crée une chaîne qui part de b_1 et qui continue vers s_1. Pour le deuxième sommet s_2, on va regarder la chaîne où a été placé le dernier élément, en l'occurence s_1. Sur cette chaîne, il est possible qu'il n'y ait plus de place, dans ce cas, on associe le coût c_a de cette action d'augmenter la chaîne de s_2 à l'infini. Sinon, on associe le coût c_a égal à la distance entre s_1 et s_2. On va maintenant parcourir tous les sommets de toutes les boucles et noter l'élément b_2 tel que la distance pour passer de b_2 à s_2 (on pose c_b égale à cette distance) est minimale. On compare donc le coût c_a de continuer la chaîne précédente au coût c_b de créer une nouvelle chaîne et on effectue l'action de coût minimal. On répète jusqu'à la fin de la liste.

La complexité totale est en $O(n^2)$ avec n le nombre total de sommets.

3 Choix des boucles

3.1 Cadre de travail

Pour le choix des boucles, nous avons choisi de nous limiter à faire des groupes. Nous avons créé autant de groupes qu'il y a de points de distribution. Pour chaque antenne, on va regarder la distance entre cette antenne et tous les points de distribution. On note le point de distribution tel que la distance est minimale et on place l'antenne dans le groupe de ce point de distribution. La complexité est de $O(dt)$ avec d le nombre de points de distribution et t le nombre d'antennes.

L'idée est alors de créer une boucle de taille variable par groupe, et d'ensuite en déduire les chaînes.

Cela est une hypothèse qui nous prive d'explorer de nombreuses solutions, mais intuitivement ces solutions n'ont pas l'air très bonnes, et on y gagne énormément en temps dans le parcours des solutions quand on fait une heuristique.

Nous avons codé en C++. Nous avons utilisé trois heuristiques différentes à la suite pour obtenir les solutions proposées, c'est pourquoi nous allons présenter chacune des ces heuristiques avant de donner l'algorithme final.

3.2 Première heuristique : On change les boucles aléatoirement

Le but est de ne pas partir d'une solution trop mauvaise pour le recuit simulé, mais de façon à avoir quand même une grosse marge de manoeuvre. En effet, une solution trouvée aléatoirement possède de nombreux défauts et donc on est loin d'être dans un minimum local comme on a plus de change de l'être après un certain nombre d'itérations en recuit simulé.

On fixe d'abord un groupe. On va prendre aléatoirement un entier entre 2 et 30 qui sera la taille de la boucle pour ce groupe. On va prendre aléatoirement les éléments qui seront dedans parmi tous les éléments du groupe. On trie la boucle de la même façon qu'on a trié la chaîne de l'algorithme pour calculer les chaînes. On calcule les chaînes avec l'algorithme décrit ci-dessus. Si cette solution est meilleure que l'actuelle, on change. On répète ceci un certain nombre d'itérations k en ne changeant donc que les éléments du groupe choisi. Une fois ceci fini, on change de groupe jusqu'à tous les faire. La complexité est en $O(dkn^2)$.

Il est plus intéressant d'optimiser groupe par groupe plutôt que tout en même temps, car si on fait tout en même temps, on peut tomber sur une solution meilleure sur certains groupes mais moins bonnes sur d'autres, ce qui fait compenser et il faut alors plus longtemps pour obtenir le même coût final.

3.3 Deuxième heuristique : Recuit simulé

Ceci est l'heuristique principale. Comme pour toutes les heuristiques, on va optimiser groupe par groupe. On commence par le groupe 1, puis le groupe 2, jusqu'au dernier groupe, avant de revenir au groupe 1, puis au groupe 2, etc... On fait ceci r fois.

On va tout d'abord coder une fonction voisin. On va prendre en paramètre une solution et un indice (l'indice du groupe que l'on optimise). On va comparer 3 voisins et renvoyer le meilleur.

On prend la boucle actuelle, on y prend un élément au hasard et on l'échange avec un élément au hasard parmi les éléments du groupe non inclus dans la boucle. On trie la boucle, calcule les chaînes, et on obtient un voisin v_1.

On prend la boucle actuelle, on y supprime un élément au hasard. On trie la boucle, calcule les chaînes, et on obtient un voisin v_2.

On prend la boucle actuelle, on y ajoute un élément pris au hasard parmi les éléments du groupe non présents dans la boucle. On trie la boucle, calcule les chaînes, et on obtient un voisin v_3.

On renvoie alors le voisin de meilleur coût.

Pour le recuit simulé, on passe de voisin en voisin avec une probabilité donnée comme dans le cours, avec la température $T = \frac{1000}{log(i)}$ avec i l'itération à laquelle on est. D'après Wikipédia, une telle température permettrait de converger en probabilité vers l'optimum global.

Pour un groupe fixé, on part donc de la solution initiale et on fait p itérations. On recommence ceci k fois (on repart de la même solution initiale à chaque fois, mais vu qu'il y a de l'aléa, il est intéressant de recommencer plusieurs fois et de garder la meilleure solution). On peut ainsi prendre la meilleure solution et passer au groupe suivant. On fait ceci jusqu'au dernier groupe. On recommence le tout r fois.

La complexité est en $O(rdkpn^2)$.

3.4 Troisième heuristique : Approche naïve

Après avoir optimisé en recuit simulé, nous avons remarqué que nous n'étions pas tout à fait dans un minimum local. En effet, il existe énormément de voisins pour chaque solution et il est plus rapide de passer à un voisin ayant un coût plus élevé (même si la probabilité peut être faible) que de tomber directement sur un voisin qui améliore le coût.

C'est pourquoi il est intéressant d'utiliser une approche naïve même après un récuit simulé. L'approche naïve consiste à ne passer à un voisin que s'il est meilleur. On utilise la même fonction voisin que pour le recuit simulé.

La complexité est en $O(dpn^2)$.

4 Algorithme final et résultat

Pour Grenoble, la solution proposée a été trouvée entièrement à la main à l'aide d'une visualisation des données. Nous obtenons un score de 1991.

Pour Paris et Nice, nous avons tout d'abord appliqué la première heuristique, pour laquelle on a passé le résultat à la deuxième, pour laquelle on a passé le résultat à la troisième. Nous obtenons des scores pour Paris et Nice de respectivement 24940 et 6269.

5 Qualité des résultats obtenus

Pour juger la qualité des résultats obtenus, nous avons formulé le problème sous la forme d'un problème linéaire en nombres entiers pour lequel nous avons fait un relaché linéaire.

On note V l'ensemble des sommets, n le nombre de sommets, d_{uv} la distance entre $u \in V$ et $v \in V$, t_u qui vaut 0 si $u \in V$ est un point de distribution, et 1 si c'est un terminal.

$$
\begin{aligned}
\min \quad & \sum_{u,v \in V} d_{uv} r_{uv} \\
\text{sc} \quad & r_{uv} \geq 0, \ \forall u, v \in V, \\
& r_{uv} \leq 1, \ \forall u, v \in V, \\
& r_{uv} + r_{vu} \leq 1, \ \forall u, v \in V, \\
& \sum_{u \in V} r_{uv} \geq t_v, \ \forall v \in V, \\
& \sum_{u \in V} r_{uv} \leq 1 + n(1 - t_v), \ \forall v \in V, \\
& \sum_{u \in V} r_{vu} + \sum_{u \in V} r_{uv} \geq 1 - t_v, \ \forall v \in V.
\end{aligned}
$$

Ce problème n'est pas exactement le problème proposé car certaines contraintes étaient difficiles à formuler. Les problèmes sont les suivants : on peut avoir une boucle de taille supérieure à 30 et qui ne contient pas de point de distribution ; et la taille des chaînes peut être supérieure à 5. Le reste est vérifié. Cependant, cela fournit quand même une borne inférieure. Les r_{uv} représentent le fait que u est relié à v ou non. Si u est relié à v, alors v n'est pas relié à u (3ème condition). La 4ème et la 5ème condition sont facultatives pour les points de distribution, mais elles forcent le fait de ne passer qu'une seule fois par antenne. On peut passer plusieurs fois sur un point point de distribution (plusieurs boucles). On peut ne pas partir d'un point de distribution (une chaîne arrive sur ce point), on peut ne pas arriver vers un point de distribution (pas de boucle) mais il ne peut pas y avoir les deux à la fois (6ème condition).

Pour Grenoble et Nice, nous n'avons pas fait de relaché linéaire et nous avons trouvé 1856 et 5402. Pour Paris, nous avons fait un relaché linéaire et nous obtenons 15849.