Rapport projet Algorithmique Avancée

Explication des différents programmes

I - Résolution heuristique

L'algorithme de résolution heuristique est composé de **4 boucles**.

- La première boucle permet d'itérer jusqu'à que tous les points soient couverts.
- La deuxième et troisième séries d'itération permettent d'avoir le couple (antenne, point antenne).
- La dernière loop remplit les points couverts par l'antenne étudiée dans la collection "tmpCouverts"
  Ensuite cette boucle fait le max(antenne_iter, meilleurAntenne) en fonction d'un ratio calculé. Ce ratio est le suivant :
  "`currentRatioAntenne = tmpCouverts.size / antenne.cout;`"

Enfin on enlève les points qui n'étaient pas couverts de la collection `nonCouverts`, regroupant tous les points non couverts. Puis, on set / reset les variables servant pour choisir la meilleure antenne et l'algorithme garde en mémoire où et quelle antenne est choisie pour quel point.

II - Résolution exacte

Comme pour le grand "I", l'algorithme va stocker la solution dans la variable `select[antennes][points_antenne]`.
Le but est de minimiser le coût de placement des antennes, tout en ayant tous les points couverts par le réseau.
Ainsi, les contraintes sont les suivantes :

- Au moins 1 antenne au total
- Tous les points couverts par une antenne, ce qui veut dire :
  - Pour tous les points, au moins une antenne à bonne distance
  - Visible entre eux, le point et l'antenne où le point est choisi ou non d'être positionné.

III - Générateur de .dat : "genPoints.c"

Quelles sont les règles de génération de points et de la matrice de visibilité ?

Les prémices de développement, en dehors de ceux donné par l'énoncé, sont les suivantes :

- Avoir une matrice de point qui représente une forme qui est la plus carrée possible
- Avoir une matrice de visibilité qui choisit les si les points sont visibles entre eux en fonction d'une probabilité, d'un pourcentage.

Ainsi, il fallait développer un programme capable de trouver les multiples, les plus égaux possible.
Un struct à été écrit pour simplifier l'envoi de données en paramètre des fonctions gérant la résolution de ce problème :

```
typedef struct modular_array {
    int **matrice;
    int taille;
    int taille_bloc;
    int n_multiples;
} modular_array;
```

C'est un struct contenant une matrice. Elle possède tous les multiples du chiffre représentant le nombre de points total. D'autre part, la structure de données stocke les variables permettant d'utiliser cette matrice correctement.
La matrice contient pour chaque indice i, dans `matrice[i][_]` deux chiffres, les multiples, stockés dans `j` (ou `_` ici).
Subséquemment, la taille du tableau à deux dimensions de données est : `[taille][2]`. Il est incrémenté de `5` en `5`, par défaut, en cas de taille maximale atteinte (soucis de performances des `realloc()`).

L'explication des fonctions est documentée dans le fichier.

Enfin, la matrice de visibilité est "calculée" via l'appel d'une fonction, qui donne en fonction d'un pourcentage 1 ou 0. Cette fonction est appelée pour chaque point de la matrice.

IV - Script de génération de tests automatisés

Le script de génération de tests automatisés permet de compter le temps d'exécution de l'heuristique et de l'algorithme de résolution exacte. Le nombre de points augment de `n_points à n_points * 2 * trials` pour les deux algorithmes. Il y a donc `trials * 2` tests.

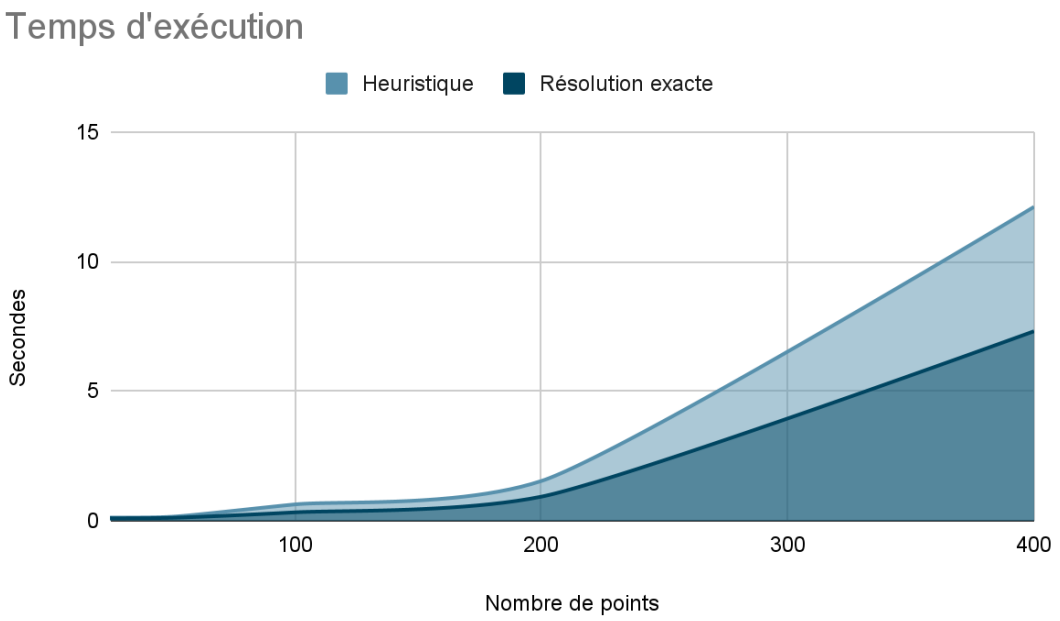
Chaque test est mesuré en temps d'exécution, et affiché sur la console.

Un message affiche le temps d'exécution total des deux algorithmes pour le même nombre de points donné, et les mêmes matrices de visibilité (même srand). Même si les algorithmes n'ont pas les mêmes données exactes, cela ne change pas grand-chose dans la mesure où il y a de nombreux essais pour le même nombre de points donné entre les deux programmes.

Explication des résultats

V - Conclusion

Lors des tests de temps d'exécution entre la méthode de résolution avec l'heuristique et avec la résolution OPL exacte, nous avons constaté une différence de temps d'un facteur 1.5 à 2. Cependant, c'est l'heuristique qui est plus lente que la résolution OPL, ce qui est étonnant. En effet le but du projet et de l'heuristique est de gagner du temps sur l'exécution afin d'avoir une solution, peut-être non optimale, dans un temps raisonnable. Hors, ici, avec cette heuristique d'une grande complexité (4 boucles imbriquées) cet objectif n'est pas atteint.



L'heuristique est pire, en tout point de vue, que l'algorithme écrit en OPL de résolution exacte. Il donne souvent une moins bonne solution, et dans un temps d'exécution plus long.

En effet, concernant la qualité des résultats de l'heuristique, on a pu constater qu'ils sont moins bons d'environ 40%. Pour 100 points de calculs, l'heuristique donnait environ 7000 en coût. Le script exact d'OPL en donnait 5000.

Des solutions d'optimisation peuvent être envisagées pour l'algorithme heuristique.
Réduire le nombre de boucles, changer de langage de programmation, explorer uniquement les points autour de la portée de l'antenne, etc. Ces solutions peuvent faire la différence et rendre l'exploration de résolution heuristique utile pour ce problème sur un très grand set de données.