Rapport stratégique : Projet c++

Dans ce rapport du projet Google Hashcode 2017 en C++, nous vous présentons d'abord rapidement les étapes de l'élaboration du projet, puis nous vous expliquons les différentes stratégies testées et enfin nous revenons sur les problèmes rencontrés.

Elaboration du projet

Après le lancement du projet, nous avons fait une première réunion pour discuter de l'implémentation du projet. Nous nous sommes répartis les tâches et nous avons discuté des différentes stratégies possible.

Après avoir implémenter la partie solution, nous avons pu nous lancer dans la phase stratégies et faire les premiers tests. Nous avons ensuite essayé d'améliorer les résultats obtenus en testant de nouvelles stratégies, tout en limitant les temps de résolution pour les maps les plus grandes.

Dans un troisième temps, nous nous sommes attelés à l'implémentation de l'arbitre des scores et du temps.

Enfin, une fois la meilleure stratégie choisie et les arbitres finis, nous avons pu mettre en commun nos solutions et faire les dernières corrections.

Les stratégies

Avant de parler des stratégies du placement des routeurs, nous vous présentons la stratégie du câblage des routeurs. Nous avons utilisé cette stratégie dès le début. On fait une approximation de l'arbre de Steiner grâce à la méthode Takahashi.(ajouter un cable entre un nouveau points et le points le plus proche dans l'arbre des points déjà connectés).

Notre première stratégie a été de placer aléatoirement les routeurs. C'est une stratégie naïve qui permettait d'avoir les premiers résultats.

Puis nous avons essayé la stratégie Sring qui consistait à placer les routeurs à intervalle constant.

Ensuite nous avons essayé une dernière beaucoups plus optimisée:

Dans une matrice qui a la même taille que la map, chaque coefficient de cette matrice indique le nombre de cellules cibles “pas encore couvertes” si on plaçait un routeur à cet endroit.

On place un routeur à la position Pi qui correspond à la valeur maximale dans cette matrice. On met à jours la matrice, une partie uniquement, plus précisément le carré de côté 2*R + 1 centré sur Pi.

On répète le processus tant qu'il rest des coefficients non vides et que le coût des routeurs ne dépasse pas le budget.

Une fois les routeurs placés, on les connecte au backbone, re-calcule le coût pour prendre en compte le prix des câbles.

Tant qu’il le budget n’est pas respecté on enlève le dernier routeur placer, et on reconnecte tout le réseaux.

Cette stratégie est gloutonne (dans l’aspect placement des routeurs), car on prend le meilleur d’abord.

Piste d’amélioration:

-Si a une étape donnée y a plusieurs meilleurs ex aequo on peut en choisir un au hasard.
-Ajouter du génétique à cet aglo:

L’idée de privilégier les positions qui permettent de maximiser la couverture locale parait raisonnable, on peut combiner ça avec du génétique comme suit:

Un individu (solution) est une liste d’entiers:

choix_max[i] = j;

à l'itération i de l’aglo on a choisi le j-eme max parmi les ex aequos.

les crossovers se font de manières classique en choisissant un point dans la liste et puis permuter avec un autre individu.

pour faire les mutations on altère des valeurs au hasard.

Problèmes rencontrés

Un des plus gros problèmes rencontrés furent les corrections finales. En effet nous avions mal interprété les consignes et étions partis dans la mauvaise direction au sujet de l'arbitre. Nous avions fait d'un côté l'arbitre de temps, qui pouvait être traité à part, alors que l'arbitre des scores avaient une grosse partie du code en commun avec la partie solution. Nous avons donc dû créer une librairie afin que l'arbitre de temps puisse utiliser l'arbitre des scores.

Annexes:
performances: score/temps