Rapport Mini-Projet C++

Google Hash Code 2017 - Polytech Nantes

Nicolas QUINQUENEL Laouenan LE CORGUILLE
Antonin ARQUEY Andréas PASTOR Mathis GUILLET

Vendredi 12 Janvier 2018

1 Introduction

1.1 Contexte

Dans le cadre de nos études d'ingénieur en spécialité informatique à l'école Polytech' Nantes, nous avons réalisé un projet par groupe de 5 avec le langage de programmation C++.

Ce mini-projet est basé sur le sujet du concours Hashcode 2017 du géant américain Google. Le principe du projet est d'optimiser la couverture d'une surface WiFi sur une carte donnée, à l'aide de routeurs, de câbles et de backbones.

1.2 Objectifs

A l'aide d'un plan donné en entrée, poser des routeurs et les connecter au backbone dans le but de maximiser le couvrement et minimiser le coût.

2 Organisation

Afin d'optimiser le temps de travail, nous nous sommes divisés en deux groupes

2.1 Groupe de résolution

- **Antonin ARQUEY** Parser et câblage
- **Andréas PASTOR** Stratégie de résolution et pose des routeurs

2.2 Groupe arbitre

- **Nicolas QUINQUENEL** Fonctions de validation et rédaction du rapport
- **Laouenan LE CORGUILLE** Fonctions de validation de l'arbitre
- **Mathis GUILLET** Moteur des temps d'exécutions

3 Stratégie de résolution

Pour pouvoir déposer les routeurs que l'on souhaite mettre sur notre solution, nous utilisons une approche gloutonne qui calcule des optimisations locales.

Pour chaque point de la carte nous calculons une distance au plus proche câble ainsi que le potentiel wifi de la cellule.

Ensuite, nous posons un routeur au niveau de la cellule qui a un potentiel wifi maximal, et une distance au plus proche câble minimale. Suite à cela nous re-calculons le potentiel wifi pour l'ensemble des cellules dans la zone d'action du routeur que l'on vient de poser et nous répétons ce processus jusqu'à ce que le budget sois épuisé.

Cette méthode est assez rapide grâce à l'optimisation suivante : nous avons recalculé le potentiel wifi seulement pour les cellules infectées par la couverture du nouveau routeur déposé.

La méthode de câblage est inspirée de la diagonalisation, en effet une diagonale permet de parcourir plus de trajet que des câbles horizontaux et verticaux. L'algorithme utilise des équations linéaires pour calculer les croisements entre deux diagonales.

Cabler en diagonale permet aussi de répandre plus de câble sur le bâtiment et ainsi pouvoir optimiser les cablage à venir. L'algorithme de cablage possède une complexité faible ce qui permet de le lancer à chaque pose de routeur et ainsi obtenir une solution.

A la fin de la pose des routeurs, on supprime tous les câbles puis on refait tourner l'algorithme de cablage, souvent une solution plus optimale est trouvée et on peut ensuite reposer quelques routeurs avec l'argent économisé.

4 Fonctionnement de l'arbitre

4.1 Instructions pour tester l'arbitre

Placer les fichiers d'entrée dans le même dossier que l'exécutable de l'arbitre.

Exécuter l'arbitre en ligne de commande en précisant le chemin du répertoire contenant les exécutables des différentes stratégies et le nom du fichier de sortie qui sera généré dans le dossier courant.

4.2 Fonctionnement

L'arbitre vérifie la validité des exécutables de résolution situés dans le dossier dont le chemin est passé en paramètre. Pour cela il itère sur chacun des exécutables et va effectuer la résolution 5 fois afin de calculer le temps moyen.

Pour le calcul du temps moyen de résolution, c'est le temps CPU sur lequel on se base. On se sert donc de std::clock() qui permet de connaître combien de temps les ressources sont allouées au programme par le système d'exploitation.

A la fin de chacune des exécutions de résolution sur les différents fichiers en entrée, nous allons vérifier la validité de la résolution fournie par le programme de résolution. Si jamais une anomalie est détectée, le temps retenu sera de 0, et l'exécution des tests sur la stratégie donnée s'arrêtent pour passer à la résolution avec la stratégie suivante.

Pour vérifier la validité d'une solution, nous récupérons les données des fichiers de résultat via un parser afin de pouvoir en juger la validité et d'en calculer le score. Nous vérifions notamment la bonne syntaxe du fichier, que tous les routeurs soient connecté au backbone, qu'aucun routeur ne soit placé sur un mur et que le budget est respecté.