

Rapport mini-projet de programmation en langage C++

Router placement, Round Final Google Hash 2017

**EHRESMANN Nicolas - HADDAD Mehdi - OUTHIER Arthur
RITTNER Neil - SOYER Mathieu**

2017/2018

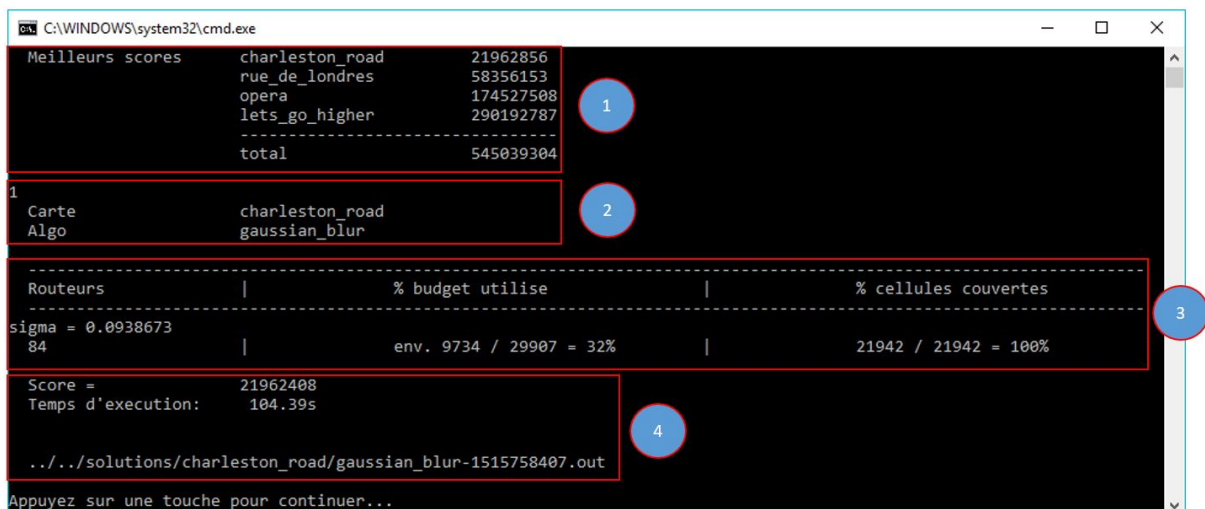
Introduction

Pour cette 4ème année à Polytech Nantes, il nous a été demandé de faire un projet de groupe en C++. Il consiste à réaliser l'algorithme demandé lors de la phase finale du Google Hash Code 2017. Cette épreuve consiste, étant donné un plan de construction, de décider où placer des routeurs sans fil et comment les connecter pour maximiser la couverture et minimiser les coûts, tout en obéissant à quelques règles.

Pour cela nous allons voir dans un premier temps l'interface pour les développeurs de la partie stratégie (qui s'occupe du moteur de résolution). Puis dans un second temps, les algorithmes qui ont été développés. Et pour finir nous parlerons de la partie arbitre et moteur de temps de résolution, et de ses quelques spécificités.

I. Interface pour les développeurs de la stratégie

Lors de ce projet, les développeurs de la partie stratégie ont mis au point la petite console suivante, indépendante de la partie arbitre. Cette interface permet aisément de suivre l'exécution d'une stratégie sur une carte et de pouvoir ainsi comparer facilement 2 exécutions. On affiche le score à titre indicatif, sous réserve que le fichier d'output soit validé par l'arbitre.



```

C:\WINDOWS\system32\cmd.exe
Meilleurs scores   charleston_road   21962856
                  rue_de_londres   58356153
                  opera         174527508
                  lets_go_higher  290192787
                  -----
                  total         545039304

1 Carte           charleston_road
Algo              gaussian_blur

-----
Routeurs          |          % budget utilise          |          % cellules couvertes
-----
sigma = 0.0938673 | 84 | env. 9734 / 29907 = 32% | 21942 / 21942 = 100%
Score =           21962408
Temps d'execution: 104.39s

../solutions/charleston_road/gaussian_blur-1515758407.out

Appuyez sur une touche pour continuer...
  
```

Légende :

- (1) Meilleur score pour chaque carte et score total.
- (2) Carte sur laquelle l'algorithme est lancé et nom de l'algorithme
- (3) Affichage du nombre de routeurs connectés au backbone, du budget utilisé et du nombre de cellules couvertes.
- Lorsque l'algorithme utilisé est "gaussian_blur", nous imprimons aussi le paramètre sigma du flou gaussien pour en garder une trace.
- (4) Score, temps d'exécution et fichier solution généré

II. Les Algorithmes développés

1. Algorithme Random

Dans un premier temps nous avons développé un algorithme au comportement simpliste dans l'optique d'avoir un résultat comme base.

Comme son nom l'indique, cet algorithme se contente de choisir aléatoirement une cellule cible non couverte et d'y placer un routeur. On continue jusqu'à couverture totale des cellules ou épuisement du budget. Les scores sont donc très variables. Ce qu'il faut voir c'est que cet algorithme ne nécessitant pas de parcourir beaucoup de fois la carte, il est très rapide à exécuter.

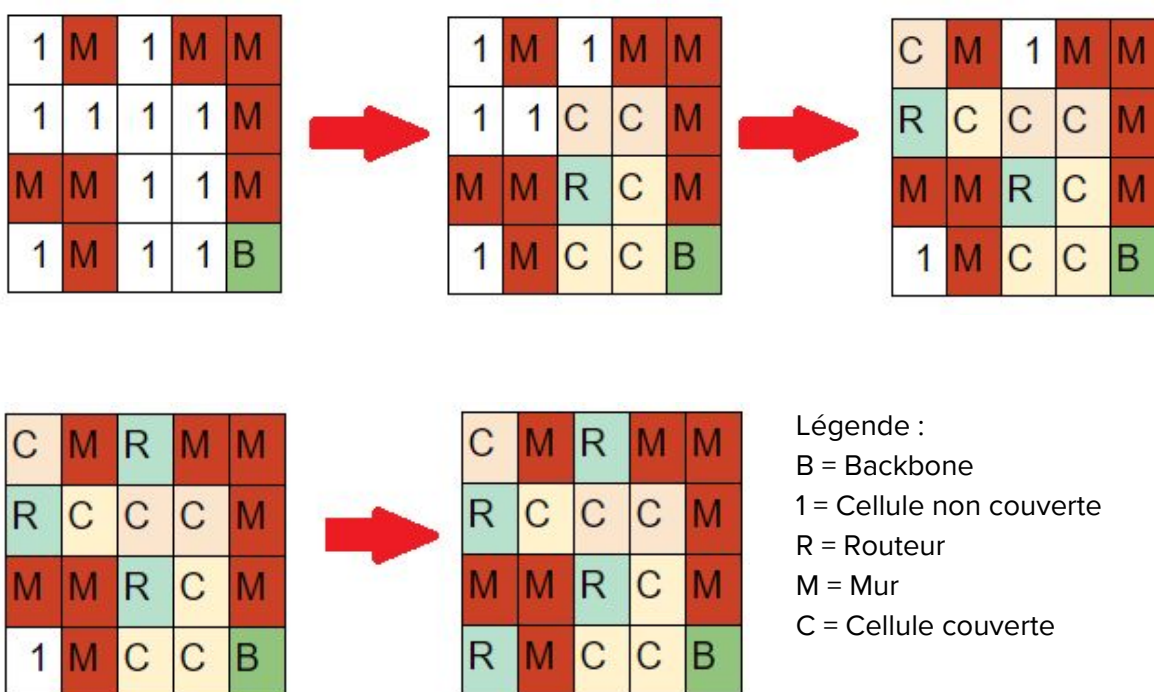
2. Algorithme BigCase

Après avoir réussi la mise en place d'un algorithme sans réelle stratégie, nous avons essayé de pousser la réflexion un peu plus loin. Nous avons implémenté un parcours de la totalité de la carte à chaque placement du routeur pour déterminer sur quelle case le routeur rapporterait le plus de points.

Dans cet algorithme, on ne prend pas en compte la distance entre les routeurs, ni le câblage. Cet algorithme qui semblait légèrement meilleur que le placement aléatoire ne se révéla pas probant, le temps d'exécution étant très long et le résultat obtenu n'étant que très peu meilleur voir moins bien que certains placements aléatoires.

Ces problèmes rencontrés nous ont poussé à nous tourner vers une autre solution. Cet algorithme a un avantage, il n'y a pas d'aléatoire ce qui fait que toutes les exécutions donnent le même résultat. Il est donc inutile d'exécuter plusieurs fois l'algorithme. Mais cet avantage peut aussi être un défaut car on ne pourra pas obtenir des scores meilleurs que d'autres.

Exemple :



3. Algorithme Gaussian-Blur

Cette méthode est celle qui nous a donné les meilleurs résultats. Tous nos meilleurs scores ont été réalisés avec celle-ci.

/!\ Temps approximatifs:

- Charleston Road : 2 min
- Rue de Londres : 10 min
- Opera : 1h45
- Lets go higher : 3h

Nous stockons une matrice de même dimension que la carte dans laquelle:

- 1 : les cellules cibles pas encore couvertes
- -1 : le reste (les cellules cibles déjà couvertes, les murs, les cellules vides)

Nous effectuons un flou gaussien sur cette matrice. Le paramètre sigma est tiré aléatoirement entre 2 bornes définies dans le code afin d'obtenir un flou différent et donc un fichier solution différent d'une exécution à l'autre. Plus sigma est grand, plus le flou sera fort.

Pour effectuer ce flou sur la matrice constituée de 1 et de -1, nous effectuons une convolution entre cette matrice et un noyau gaussien de même dimension que le rayon d'un routeur.

Par exemple, pour sigma = 1 et un routeur de rayon = 2, cela donne un noyau gaussien de dimension 5.

Noyau gaussien :

0.003765	0.015019	0.023792	0.015019	0.003765
0.015019	0.059912	0.094907	0.059912	0.015019
0.023792	0.094907	0.150342	0.094907	0.023792
0.015019	0.059912	0.094907	0.059912	0.015019
0.003765	0.015019	0.023792	0.015019	0.003765

Carte :

1	1	-1	1	1
1	1	-1	-1	-1
1	1	-1	1	1
-1	-1	1	-1	1
1	1	1	1	-1

Résultat de la convolution Noyaux gaussien ** Carte :

0.396303	0.281579	0.005925	0.001979	0.086665
----------	----------	----------	----------	----------

0.466011	0.318845	-0.046678	-0.094031	0.023097
0.299079	0.229248	0.034838	0.039434	0.131671
0.123227	0.185569	0.206422	0.185467	0.156475
0.164287	0.281579	0.329015	0.211589	0.053417

Nous récupérons la valeurs la plus grande dans la matrice résultante de cette convolution : 0,466011

Ensuite nous récupérer les coordonnées des cases contenant ces valeur : ici, seule la case de coordonnées (1, 0) contient la cette valeur.

Si on avait eu plusieurs points respectant cette valeur maximale, on aurait alors procédé à d'autres filtres:

- on prend les positions qui un maximum de nouvelles cellules.
- on prend les positions qui engendreront le moins de coût de câblage.
- on prend les positions qui sont les plus proches de murs, de cellules vides ou de cellules déjà couvertes

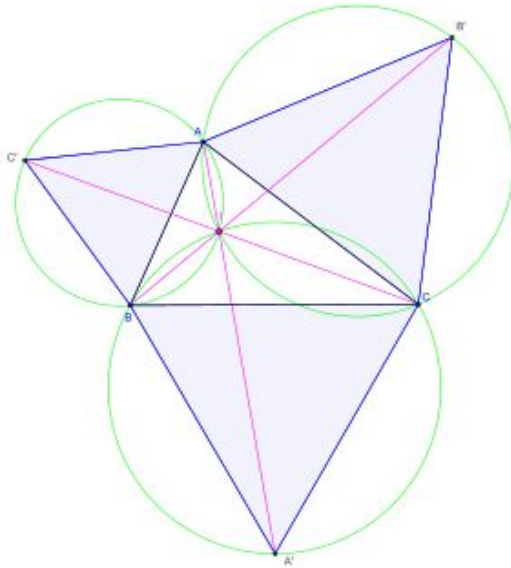
Si il reste plusieurs positions possibles après cela, alors on en choisit une aléatoirement, de sorte à avoir un fichier solution différent à chaque lancement de l'exécution.

4. Câblage

Le repère n'étant pas cartésien, il est équivalent de se déplacer en diagonale, en horizontale ou à la verticale. De ce fait, nous favorisons le déplacement en diagonale car qui permet de se rapprocher plus rapidement de la destination. De plus, dans beaucoup de cas, une seule droite diagonale permet de relier plusieurs routeurs à la fois.

Nous déterminons l'arbre minimal couvrant les routeurs afin d'optimiser les connexion inter-routeurs.

Ensuite, nous parcourons aléatoirement les routeurs. Pour chaque routeur, nous récupérons les 2 routeurs les plus proches de lui. Ces 3 routeurs forment alors un triangle ABC.



Source : Wikipédia

Ensuite nous cherchons à trouver le point de Fermat (barycentre). Le point de Fermat d'un triangle ABC est le point I pour lequel la somme $IA + IB + IC$ des distances aux trois sommets du triangle est minimale. (Définition Wikipédia).

Ce point de Fermat est calculable si tous les angles sont inférieurs à 120° . Dans le cas où cette contrainte n'est pas respectée, nous approximations le barycentre en calculant le centre de gravité du triangle.

Si cette contrainte est respectée, on cherche les point A', B' qui composeront les triangles équilatéraux ABC' et ACB'.

On trace ensuite les droites (AA') et (BB').

Le point de Fermat noté I correspond à l'intersection de ces 2 droites.

III. Arbitre et moteur de temps de résolution

L'arbitre est sous forme d'exécutable et prend deux paramètres: Le répertoire des exécutables des stratégies et le nom de fichier de sorti. Pour le fonctionnement de cet arbitre, ce deuxième paramètre est aussi le nom de la map que l'on souhaite tester. Le moteur de temps va ensuite tester chaque stratégie sur la map 5 fois et obtenir un temps d'exécution moyen. Les résultats sont écrits dans un fichier de sorti avec pour chaque stratégie le temps moyen et le score obtenu mesuré selon la fonction arbitre checkSolution.

Conclusion

Durant notre projet nous avons effectué des algorithmes en augmentant peu à peu la complexité de ceux-ci. Nous avons créé des algorithmes qui parfois se sont révélés peu intéressants mais à la fin de ce projet, nous avons un algorithme efficace qui est l'algorithme de flou gaussien. Nous sommes dans les meilleurs scores du classement sur le site et nous

avons plusieurs méthodes permettant d'obtenir des résultats. Notre arbitre est fonctionnel également et nous avons une interface agréable et utilisable pour la stratégie et l'arbitre.