Lepinette Cyril

Optimisation Combinatoire

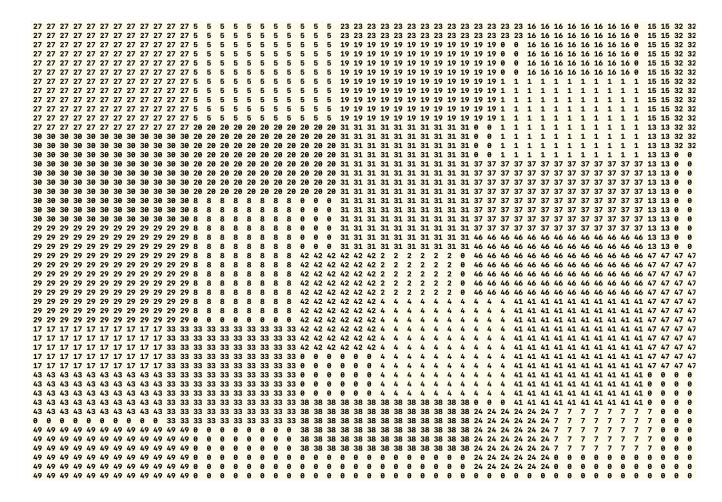
Le projet a été réalisé en C++ et est composé d'un main ainsi que de 3 classes. Des captures d'écran de résultats sont jointes, en plus de celles dans l'interprétation des résultats.

Bâtiment

Un bâtiment est composé d'une hauteur, d'une largeur ainsi que d'un numéro. Le numéro correspond à l'ordre dans lequel il a été ajouté dans la liste (vecteur) de bâtiments à placer sur le terrain. Par exemple, dans l'exemple donnée, l'objet « 3,3 » est le bâtiment 1. Ce numéro est utile pour pouvoir comparer les ordres dans lesquels ont été placés les bâtiments sur le terrain. Par exemple, pour deux tris aléatoires nous pourrions avoir 3-1-2 et 1-3-2. De ce fait, le numéro du bâtiment est commun pour tous les ordres de tri (aléatoire, mais aussi d'espace et d'encombrement). On sauvegarde également sa position (coin supérieur gauche) lorsque celle-ci a été calculée.

Terrain

Un Terrain contient également une largeur et une hauteur. La classe contient deux vecteurs de bâtiments : l'un qui représente les bâtiments qui sont à placer et l'autre représente les bâtiments qui sont placés. Il contient également une « grille » qui permettra l'affichage par la suite. Cette grille est un vecteur de vecteur d'entier. Chaque entier correspond à un numéro de bâtiment. C'est ainsi que nous pourrons ensuite afficher le terrain de la sorte :



O représentant un espace vide.

Différentes fonctions sont présentes, elles sont ici résumées. Voici celles qui placent les bâtiments :

- O PlacerBatiments : si des bâtiments sont à placer, alors pour tout bâtiment à placer on contrôle s'il peut être placer (fonction PlacerBatiment). Si c'est le cas, on met à jour l'aire du terrain utilisée et on la retourne (=le résultat du placement, plus l'aire est grande, mieux c'est)
- o PlacerBatiment: on regarde pour chaque case du terrain x,y si elle est occupée. Si elle est vide alors on regarde si notre bâtiment à placer est en conflit avec d'autres bâtiments déjà placés. (Pour tout bâtiment déjà placé ->fonction enConflitAvec). S'il n'est pas en conflit, alors on indique au bâtiment sa position (exemple, b1 pos(3,4) de largeur 8 et hauteur 7, utile pour contrôler les conflits) puis on appelle AjouterLeBatiment qui l'ajoute à la liste des bâtiments placés dans le terrain.
- o enConflitAvec : Contrôle si un bâtiment n'est pas en conflit avec un autre sur le terrain. Cette fonction utilise également incluDansLeTerrain pour controler que le bâtiment ne dépasse pas du terrain.

Les autres fonctions sont : trier(type de tri), générer(nombre de générations aléatoires), afficher, initialisation etc.

FileHandler

Permet de créer les bâtiments et le terrain à partir d'un fichier.

Main

Contrôle s'il y a des arguments. Si non, il lance à partir du fichier. Si oui, il génère des bâtiments de dimensions aléatoires et compare les heuristique d'espace, d'encombrement et de générations aléatoires.

Interprétation des résultats

Les heuristique d'espace et d'encombrement donne dans l'ensemble des résultat similaires. Le résultat du nombre de générations aléatoire dépend énormément du nombre de générations faites. Le résultat est globalement en dessous des deux heuristiques précédentes. J'ai dû monter à 10 000 générations (50 bâtiments) pour obtenir des résultats meilleurs que l'espace et l'encombrement. Cependant cela prend beaucoup plus de temps et l'on pourrait se demander s'il ne vaudrait pas mieux calculer toutes les permutations.

```
3000 ordres aléatoire : aire 2393 > aléatoire 2376

Résultat tri sire: 2393
ordre des batiments triés 36 4 5 23 10 14 9 27 39 44 12 17 30 22 31 26 16 7 8 13
ordre des batiments triés 36 4 5 23 10 14 9 27 39 44 12 17 30 22 31 26 16 7 8 13

Début de l'aléatoire:
Génération de 3000 ordres aléatoires
meilleur solution (numéro de batiment trié dans l'ordre): 19 36 23 41 48 6 11 9 31 12 29 14 5 43 17 44 34 0 38 1 13 24 49 45 20 21 37 25 47

Résultat par génération aléatoire: 2376

Affichage:
```

```
10 000 ordres aléatoire : aire 2375 < aléatoire 2410

Résultat tri aire: 2375 ordre des batiments triés 10 27 31 29 21 40 5 30 26 41 9 39 33 4 19 22 46 20 42 17 32 13 3

Résultat tri encombrement: 2375 ordre des batiments triés 10 27 31 29 21 40 5 30 26 41 9 39 33 4 19 22 46 20 42 32 13 17 3

Début de l'aléatoire: Génération de 10000 ordres aléatoires meilleur solution (numéro de batiment trié dans l'ordre): 27 5 23 16 19 1 20 15 31 37 30 29 46 8 42 4 41 17 33 43 38 24 7 49 32 2 13 47 Résultat par génération aléatoire : 2410

Affichage:
```

On pourrait également penser que le meilleur résultat aléatoire donne une liste triée proche de celle des tris d'espaces et d'encombrement. Mais cela ne semble pas être le cas :

```
Aire = 4 3; aléatoire = 1 4 0

Résultat tri aire: 20
ordre des batiments triés 4 3

Résultat tri encombrement: 20
ordre des batiments triés 4 3

Début de l'aléatoire:
Génération de 18000 ordres aléatoires
meilleur solution (numéro de batiment trié dans l'ordre): 1 4 8

Résultat par génération aléatoire: 22

Affichage:

A 4 4 8 0 8 8
A 4 4 8 0 8 8
A 4 4 8 0 8 8
A 4 4 8 0 8 8
A 4 4 8 0 8 8
A 4 4 8 0 8 8
A 4 4 8 0 8 8
A 1 1 4 4 4 8 1 3 3 3 3 3 8
A 1 1 3 3 3 3 3 8
A 1 1 3 3 3 3 3 8
A 1 1 3 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 6 8 8 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 6 8 8 8
A 1 1 3 3 3 3 8 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 1 3 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
A 1 3 3 3 8
```

Génération aléatoire vs toutes les permutations

Les générations aléatoires sont bien lorsque toutes les permutations seraient trop longues à calculer. On pourrait calculer une équation permettant de savoir, suivant le nombre de bâtiments, quand est-ce que toutes les permutations sont plus rentables à faire qu'un certain nombre d'aléatoire. Par exemple, pour 5 bâtiments, il est plus rentable de faire toutes les permutations que de calculer 10 000 fois des permutations aléatoire. On répèterait trop de fois la même permutation pour rien.

Donc si si toutes les permutations sont trop longues à calculer alors on en prend un certain nombre qui sont aléatoires et on retourne la meilleure.

Les heuristiques d'espace et d'encombrement restent les plus rentables en terme de cout/résultat.

Améliorations:

D'un point de vue technique, la mise en place de threads aurait été intéressante. En effet, les calculs sont lents lorsque nous voulons faire beaucoup de permutations ou de génération aléatoires.

Nous aurions pu tester d'autres heuristiques tel qu'alterner les gros bâtiments avec des plus fins dans l'ordre à placer.