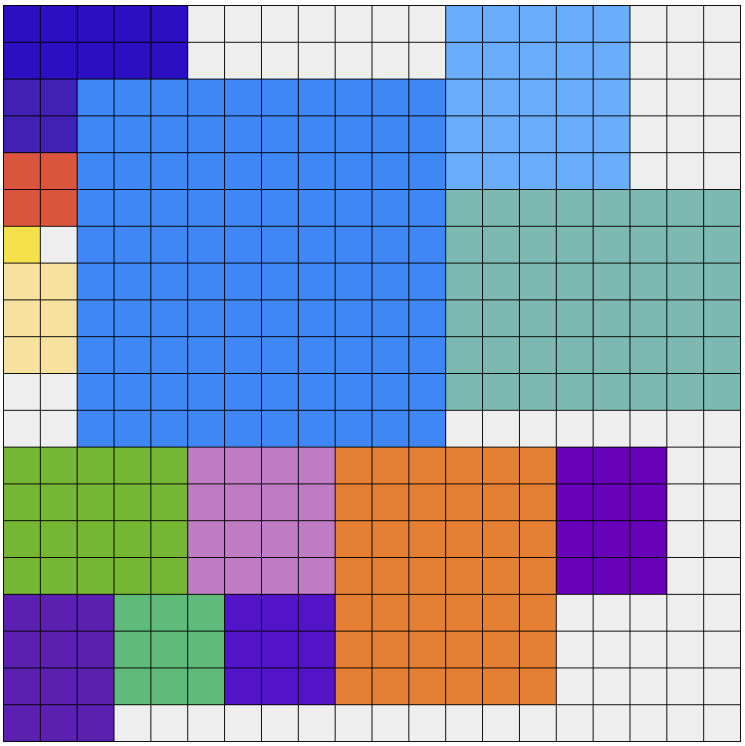
Rapport d'optimisation linéaire et combinatoire

Projet :

Implémentation d'algorithmes d'optimisation des placements de bâtiments dans une ville.



(Screen shoot d'une instance après glouton)

Etudiants :

-Maxence Marot

-Quentin Maignan

Tables des matières :

1 – Présentation du problème et de l'objectif à optimiser

A – Notre problème

Ce projet d'optimisation consiste à placer des bâtiments, qui sont représentés par des positions de départs en X et Y et des longueurs et largeurs prédéfinies, dans une ville représentée par une matrice d'entier. Dans cette ville, pour chaque case[x][y], nous avons le numéro du bâtiment qui occupe cette case ou 0, si aucun bâtiment n'est sur la case. Nous avons pour contrainte que tous les bâtiments soient reliés avec le bâtiment numéro 1 qui représente la mairie de la ville.

Nous disposons de 5 instances de ville dans lesquelles la taille de la ville ainsi que tous les bâtiments nous sont imposés.

B – Notre objectif

Notre objectif est de maximiser l'air total des bâtiments dans la ville en veillant à ce que tous les bâtiments soient bien reliés avec la marie. Nous allons donc pouvoir calculer un score de ville qui sera la somme des aires des bâtiments placés dans la ville. Plus ce score sera élevé, plus notre algorithme aura été performant.

2 – Présentation des algorithmes de résolution

A – Links

Ce premier algorithme n'est pas un algorithme de résolution à proprement parlé, mais il va nous permettre de vérifier que tous les bâtiments sont bien reliés à la mairie. Cet algorithme va explorer les cases adjacentes à notre mairie pour voir si tous les bâtiments qui sont placés sur les cases de la ville sont atteignable par la route ou bien, directement collés à notre mairie.

B – Glouton

Cet algorithme dit « glouton » est le cœur de notre projet. En effet c'est celui-ci qui va réaliser le placement des bâtiments dans la ville en respectant la contrainte de connexion entre les bâtiments et la marie en appelant links.

Pour faire cet algorithme, nous allons itérer sur toutes les cases de notre matrice. Dès qu'une case sera marquée comme vide `matrice[y][x] == 0`, nous allons essayer de placer un bâtiment (le premier de notre liste de bâtiments qui puisse rentrer dans cette case, c'est-à-dire qu'il ne sorte pas des bornes de la ville et qu'il ne soit pas non plus sur un autre bâtiment). Une fois un bâtiment placé, nous allons regarder s'il n'a pas détruit de liaisons avec d'autre bâtiments précédemment placés, (s'il a détruit des liens, alors nous passerons simplement à un autre bâtiment). Si le bâtiment a bien été placé alors nous passons simplement à la case suivante. De fait, aucun bâtiment bien placé ne sera déplacé ou retiré.

C – Les variantes du glouton

Nous avons 4 variantes du glouton. Ces 4 variantes sont appelées dans la même méthode glouton. Pour les différencier, nous avons rajouté un attribut de méthode qui va pouvoir effectuer un tri spécifique avant de réellement commencer le glouton.

`if(sortType.equals("air"))` : alors nous trions nos bâtiments pour que les bâtiments qui comptent le plus de cases soient mis en premiers dans la liste des bâtiments. De ce fait, les plus gros bâtiments seront placés en premier dans la ville.

`else if(sortType.equals("congestion"))` : alors nous trions nos bâtiments par congestion c'est-à-dire que les bâtiments dont la longueur + la largeur est la plus élevé, seront placés en premier.

`else if(sortType.equals("random"))` : et finalement un dernier tri dit random, pour que les bâtiments soient triés de façon aléatoire.

D – Best glouton

Cet algorithme est dit « le meilleur » car il va itérer sur tous les placements possibles de la mairie, avant de faire le glouton pour voir avec quelle position de la mairie le score est le meilleur.

Nous pouvons appeler best Glouton avec les 4 types de glouton précédemment définis.

3 – Etudes des résultats

A – les résultats du glouton « de base »

Tri=none	Test 1	Test 2	Test 3	Test 4	Test 5	av.	Best Glouton
Instance 1 max 251	231	217	225	231	215	223,8	251
Instance 2 max 456	320	326	309	322	326	320,6	344
Instance 3 max 804	796	796	782	819	796	797,8	829
Instance 4 max 579	382	368	355	358	368	366,2	385
Instance 5 max 620	358	358	349	358	358	356,2	367

Dans ce tableau, vous pouvez voir nos scores pour l'algorithme du glouton en mode « sans tri ». Nous avons 5 tests pour chaque instance pour avoir une moyenne de plusieurs runs de notre algorithme vu que la marie est placée aléatoirement dans le coin supérieur gauche.

Nous pouvons voir que notre algorithme Best glouton est toujours le meilleur. En effet, c'est normal puisque le max de nos tests est une borne inferieur à notre algorithme du best glouton.

4 – Discussion / Conclusion

A – Ce qu'il nous manque

Pour ce projet d'optimisation, il nous manque l'algorithme de « Branch and Bound ». Malheureusement, il nous a manqué un peu de temps pour l'implémenter.

Nous avons aussi quelques problèmes d'algorithme, tel que le links qui ne marche pas comme voulu de temps en temps et nous n'avons pas su corriger le problème. Tous nos tris de bâtiments qui sont censés représenter les 3 autres variantes du glouton ne marchent pas non plus, pour une mystérieuse raison, ou raison mystérieuse à vous de choisir.

Mise à part ces deux petits soucis nous avons réussi à faire un projet plus ou moins complet et une interface graphique plutôt ludique.

Nous aurions aussi pu faire en sorte que nos algorithmes soient mieux optimisés, par exemple dans le glouton nous passons en revu toutes cases. Nous aurions pu avoir une liste de coordonnées où il est possible de placer nos bâtiments pour perde moins de temps à faire, pour chaque bâtiment, des vérifications de placement. Il y aurait pas mal de petites optimisations comme celle-ci pour faire moins d'itérations dans nos fors, pour gagner en rapidité

B – Ce qu'on a appris

Avec ce projet nous avons pu, pour la première fois, réaliser un algorithme d'optimisation. Nous nous sommes rendu compte que dans un contexte simple comme celui de ce projet, ce genre d'algorithme peut être facile à mettre en place et puissant. Demain, si nous avons un problème de ce

genre nous serions plus enclins à implémenter cet algorithme glouton, et pourquoi pas si le temps le permet cette fois ci, le « Branch and Bound »