Quadratic Assignment Problem

Ce projet a été réalisé en langage Java. L'ensemble des sources se trouvent dans le dossier fourni avec ce rapport : il s'agit d'un projet Java lançable avec un IDE tel que IntelliJ ou Eclipse. La méthode main se trouve dans le fichier Main.java.

Les instances de Taillard :

Fichier sources des instances de Taillard

Les instances de Taillard sont composés de la taille n = |L| = |P| du QAP (nombre d'emplacements ou nombre d'équipements), deux matrices symétriques (le poids et la distance entre deux éléments est évidemment non-orientés) de taille n x n qui indiquent la distances entre deux locations et le poids entre deux équipements.

Implémentation de la modélisation du problème

```
10      private Map<Integer, Integer> locationWithFacility;
11      private int[][] distances;
12      private int[][] weights;
13      private int size;
14
15      private int optimalSolution;
```

Attributs de la classe QAP

.

locationWithFacility représente la bijection qui associe une location à un équipement. Deux tableau pour gérer les distances et les poids, la taille et enfin la solution optimale données par le site.

Le voisinage

```
80      public boolean isOptimal() throws Exception {
81          int sum = sum();
82          if (sum < optimalSolution)
83              throw new Exception("Solution can't be better than optimal one");
84          return sum == optimalSolution;
85      }
```

méthode qui permute deux équipements

On utilise la permutation (symétrique) de deux équipements pour définir le voisinage.
On définit la taille d'un voisinage v pour un équipement qui pourra permuter avec les éléments proches :
Par exemple : n = 12, |v| = 4, l'équipement 1 pourra permuter avec les éléments du voisinage v(1) = { 11, 0, 2, 3}
Le voisinage V regroupe le voisinage de chaque élément :
*Pour tout i appartenant à n V = Union v(i) donc |V| < n * |v|*
car si j appartient à v(i) et i à v(j) alors c'est la même permutation (symétrie)

Méthodes heuristiques

Recuit Simulé

Paramètres

Le Recuit Simulé prend 3 paramètres: la température initiale, la variation de la température et le nombre maximal d'itérations. La température correspond aux chances d'accepter une solution sélectionnée lors d'une itération si cette solution est moins bonne: plus la température est importante et plus les chances sont élevées. La température initiale correspond à la température utilisée pour la première itération.

La température diminue au cours du temps: cette diminution étant dans notre cas linéaire, à chaque itération on multiplie la température par le coefficient de variation entré en paramètre, compris entre 0 et 1. Plus le coefficient est proche de 1, plus la diminution sera lente.

Enfin, le nombre maximal d'itérations indique au bout de combien d'itérations on s'arrête pour renvoyer la solution trouvée.

Fonctionnement

L'algorithme s'exécute dans la classe SimulatedAnnealing.java héritée de la classe Algorithms, qui contient tous les éléments communs aux deux algorithmes: entre autres un objet QAP qui a pour but de représenter le problème dans son état actuel à un moment donné: emplacements des équipements, poids… La classe SimulatedAnnealing est simple à utiliser: elle contient la méthode execute qui définit tout l'algorithme, et qui prend en entrée les trois paramètres décrits plus haut.

Dans le cadre de l'algorithme, la solution que l'on sélectionne à chaque itération est totalement aléatoire; cependant dans notre code, d'une exécution de l'algorithme à l'autre le générateur d'aléatoire est toujours initialisé avec la même graine: cela permet de ne pas obtenir des résultats différents si l'on exécute deux fois la même instance de Taillard avec les mêmes paramètres.

Tabou

Paramètres

Le paramètre principal de la méthode Tabou est la taille de la liste Tabou, elle peut changer significativement de chemin exploré car ce facteur à un impact sur liberté de choix (plus la taille de la liste est grande plus des chemins seront banni).

Le second est le voisinage qu'on peut faire varier qui joue aussi un grand rôle dans le nombres de possibilités à partir d'un noeud du chemin. Il est aussi le principal facteur de la complexité temporelle de l'algorithme.

Le nom d'itération maximum permet la terminaison accélérée de l'algorithme même si d'autre méthode le font s'arrêter :

- il retourne dans un même état (solution et liste tabou identiques), cela prouve la terminaison mais elle peut être très lente.
- il obtient la solution optimale qui n'est pas suffisante pour la terminaison de l'algorithme.

Il peut évidemment avoir un impact néfaste sur l'obtention de la solution puisqu'il peut potentiellement arrêter un algorithme sur la bonne voie.

Fonctionnement

```
98 @    public int[] executeMultipleTsizes(int[] Tsizes, int neighbourhoodSize, int maxIterations) throws Exception {
99
100         int[] solutions = new int[Tsizes.length];
101         for (int i = 0; i < Tsizes.length; i++)
102             solutions[i] = execute(Tsizes[i], maxIterations, neighbourhoodSize).sum();
103         return solutions;
104     }
105
106 @    public int[] executeMultipleNeighbours(int Tsize, int[] neighbourhoodSizes, int maxIterations) throws Exception {
107
108         int[] solutions = new int[neighbourhoodSizes.length];
109         for (int i = 0; i < neighbourhoodSizes.length; i++)
110             solutions[i] = execute(Tsize, maxIterations, neighbourhoodSizes[i]).sum();
111         return solutions;
112     }
```

Méthodes utilisées pour construire les graphes

Le code de l'algorithme Tabou est commenté, il se trouve dans la Classe Tabou.java

```
43              for (int i = 0; i < size; i++) {
44                  for (int j : neighbourhood(neighbourhoodSize,i)) {
45                      if (!T.contains(new Pair<>(i, j))) {
```

Parcours du voisinage

Pour approfondir en détail le parcours du voisinage, le premier for() itère chaques équipements de l'échantillon. Pour chacun on va voir tous les équipement voisins avec qui il

peut permuter (2ème for()) et enfin on vérifie que la permutation est autorisée par la liste Tabou.

```
114     /**
115      *
116      * @param demiSize neiberhood size : node will have size * 2 neighbours : size for each side
117      * @param node
118      * @return List of neigbours
119      */
120     private List<Integer> neighbourhood(int demiSize, int node) {
121
122         int seq;
123         final int QAPsize = qap.getSize();
124         demiSize += demiSize % 2;
125         if (demiSize > QAPsize)
126             demiSize = QAPsize;
127         List<Integer> neighbours = new ArrayList<>();
128         for (int i = 0; i < demiSize; i++)
129             if ((seq = modulo( i: node + sequence(i), QAPsize)) > node)
130                 neighbours.add(modulo(seq,QAPsize));
131         return neighbours;
132     }
133
134     /**
135      *
136      * @param i
137      * @return
138      */
139     private int sequence(int i) {
140         final int QAPsize = qap.getSize();
141         final boolean positif = i % 2 == 0;
142         return (positif ? 1 : - 1) * (i/2 + 1);
143     }
```

Méthode pour définir le voisinage

Pour une meilleur optimisation le voisinage des permutations ne prend que des valeurs paires inférieure à la taille de l'échantillon. On crée une liste de permutations de la manière décrite précédemment (cf : Le voisinage)

Résultats

Le Recuit Simulé

Pour le Recuit Simulé nous avons essayé de tester le plus de combinaisons de paramètres possibles afin de trouver la meilleure combinaison pour chaque instance de Taillard. De manière générale, nous avons observé que :

- les résultats sont meilleurs quand le coefficient de variation de la température est supérieur ou égal à 0.9, ce qui correspond à une diminution lente de la température
- la température initiale doit être comprise entre 0 et 25000, car au-delà trop de mauvaises solutions sont acceptées
- pour le nombre maximum d'itérations, 1000 semble être un nombre idéal. Nous avons choisi ce nombre pour coïncider avec la méthode Tabou et pouvoir mieux comparer les résultats. Augmenter le nombre d'itérations ne change pas grand chose. En effet, lors d'itérations supplémentaires la température a déjà bien baissé, ce qui laisse peu de chances de partir sur une moins bonne solution et donc de sortir d'un minima local.

Ainsi, pour chaque instance, on a testé toutes les températures initiales de 0 à 25000 avec un pas de 100, et tous les coefficients de variation de 0.9 à 0.99 avec un pas de 0.01. Choisir des pas plus faibles augmente le nombre de combinaisons à tester et donc le temps d'exécution, et ne permet pas forcément d'obtenir des meilleurs résultats.

Compte tenu de l'étendue des combinaisons testées, nous n'afficherons pour chaque instance que la meilleure fitness trouvée, avec la température initiale et la variation de température ayant permis d'obtenir ce résultat :

Instance	Température initiale	Variation de température	Fitness associée	Solution optimale de référence
12	1400	0.99	224416	224416
15	13600	0.99	388214	388214
17	9200	0.96	498656	491812
20	9200	0.94	719388	703482
25	15300	0.91	1198612	1167256
30	2500	0.99	1871396	1818146
35	10300	0.99	2516216	2422002
40	12000	0.94	3303208	3139370
50	21700	0.98	5226450	4938796
60	5400	0.99	7653044	7205962
80	1400	0.97	14389024	13499184
100	7800	0.98	22439606	21044752

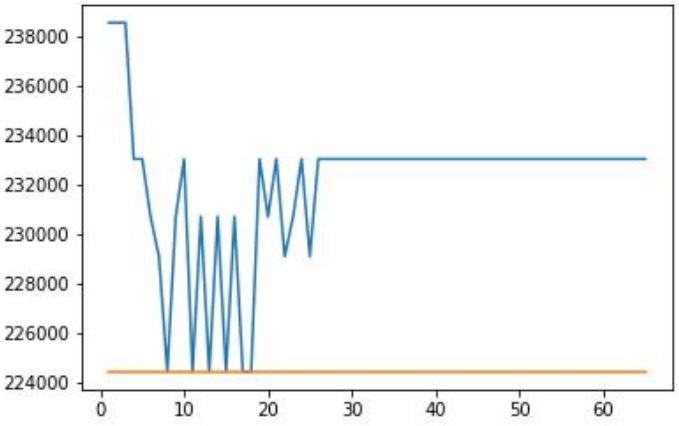
On peut voir que la température initiale et la variation de température idéales ne varient pas de manière linéaire. C'est pour cela qu'il est difficile d'effectuer une bonne estimation de ces valeurs avant d'appliquer l'algorithme. D'autant plus que les résultats ne dépendent pas uniquement de ces variables, mais également du facteur aléatoire lors de la sélection du voisin et de la chance d'y aller si c'est une moins bonne solution.

Tabou

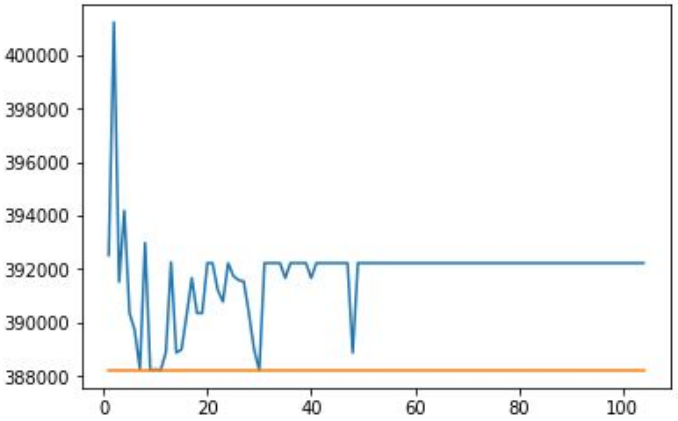
Solutions en fonction de la taille de la liste Tabou

Paramètres :

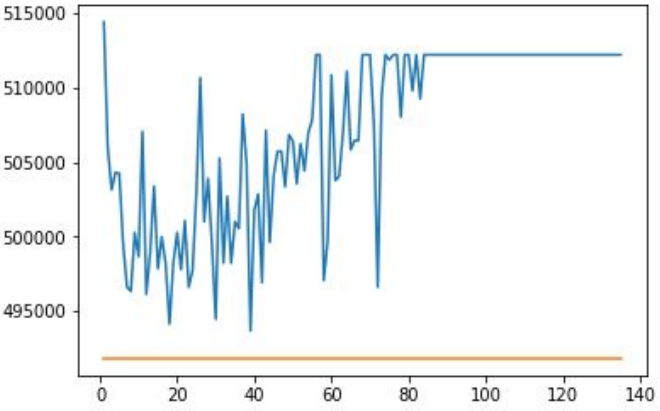
- Maximum d'itérations : 1000
- Voisinage de n éléments |V| = n * (n - 1) / 2



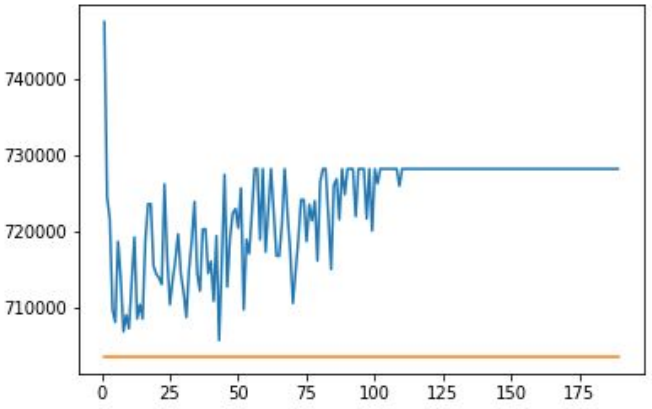
Taillard 12



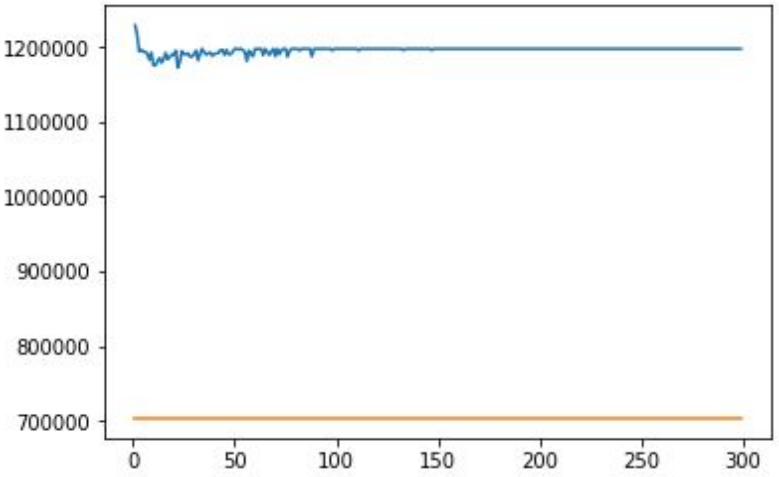
Taillard 15



Taillard 17



Taillard 20



Taillard 25

Légende :

La courbe en bleu représente les solutions de l'algorithme Tabou en fonction de la taille de la liste Tabou.

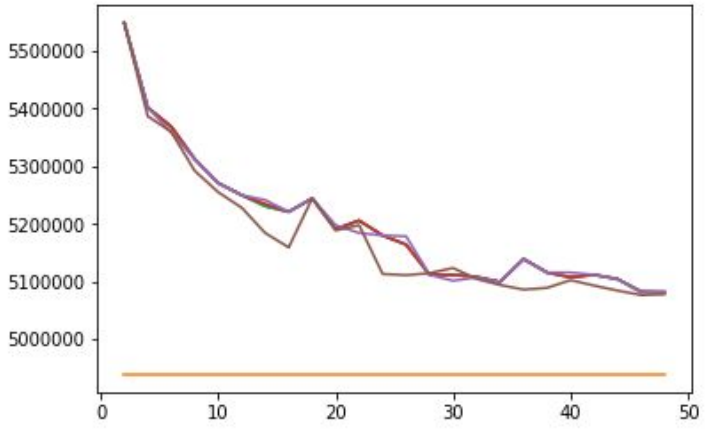
La courbe en orange représente la solution optimale.

On remarque que lorsque la taille de l'échantillon grandit les résultats s'éloigne de la solution optimale, l'instance de Taillard de taille 25 montre que l'algorithme Tabou n'est plus efficace pour des échantillons de taille conséquente.
En ce qui concerne les plus petits échantillons il est bien adapté, pour les instances de Taillard 12 & 15 il trouve même des solutions optimales pour certaines valeurs de la taille de la liste Tabou.
On peut conjecturer un intervalle grossier de taille de la liste Tabou donnant de bonnes solutions : La taille de la liste doit être au moins supérieur à 5 et inférieure à n/2.

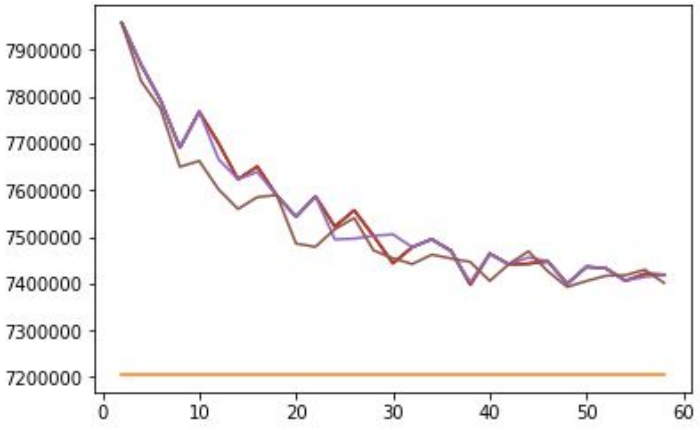
Solutions en fonction de de la taille de voisinage

Paramètres :

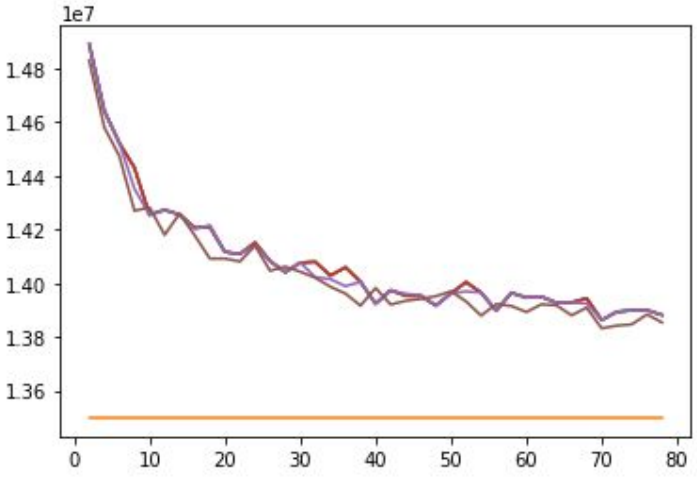
- Maximum d'itérations : 1000
- Voisinage de n éléments |V| = n * (n - 1) / 2



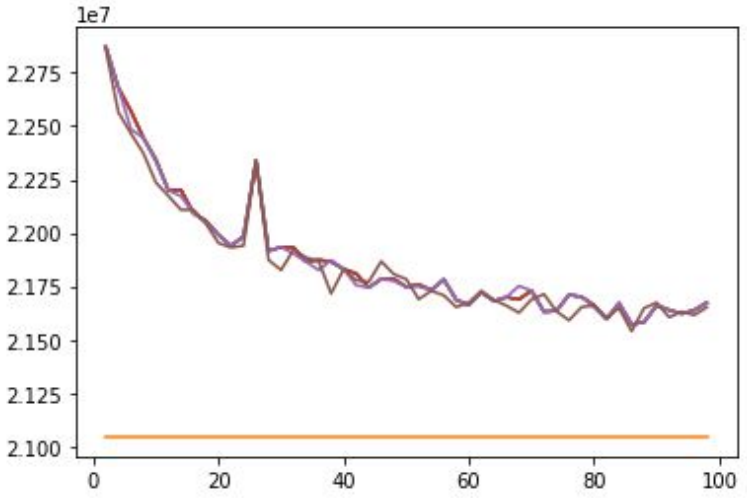
Taillard 50



Taillard 60



Taillard 80



Taillard 100

Légende :
Les courbes de différentes couleurs hors orange sont les solutions de l'algorithme Tabou en fonction de la taille du voisinage (plus précisément le nombre de permutations possible pour chaques éléments). Chaque courbes à une taille de la liste Tabou différente.
La courbe en orange représente la solution optimale.

Taillard 50	Taillard 60	Taillard 80	Taillard 100
200	500	700	990
100	200	500	500
75	100	100	100
50	50	50	50
10	10	10	10

Tout d'abord en augmentant le voisinage, les solutions s'améliorent cependant si la complexité est proportionnelle au nombre de permutations possibles (C(Tabou) = O(m * n^2 * c * p)) la qualité des solutions semble plutôt suivre une sorte d'hyperbole décroissante qu'une fonction affine : On peut voir que la courbe à une forte pente pour des valeurs basses mais se stabilise après n/2 et plus l'échantillon est grand mieux elle se stabilise. On peut donc en déduire que diminuer fortement le nombre de permutation (de n à n/2 par exemple) réduira fortement la complexité temporelle (de 50% par exemple) en obtenant des solutions peu altérés par rapport à un voisinage plus grand.

Comparaison des méthodes

Complexité

Soit n la taille de l'échantillon, et m le max d'itérations, en choisissant le plus grand voisinage |V| = n * (n - 1) / 2. c est une constante petite égale au nombres d'itérations dans la boucles.
C(Tabou) = m * (n * |V|) * c = 0.5 * m * (n^3 - n^2) * c = **O(m * n^3 * c)**
Si on choisit un voisinage plus petit en laissant pour chaque élément p permutations possible : |V| < n * v
C(Tabou) = m * (n * |V|) * c = m * n^2 * p * c = **O(m * n^2 * p * c)**

c' est une constante plus grande que c au nombres d'itérations dans la boucles.
C(Tabou) = m * c' = **O(m * c')**

La complexité du recuit simulé est bien meilleure et ne dépend pas de l'échantillon.
C(Recuit) = m = **O(m)**

Efficacité

Pour ce problème, de manière générale la méthode Tabou est plus efficace. En effet, les paramètres à prendre en compte sont un peu moins nombreux étant donné qu'il s'agit d'entiers, et pour chaque instance de Taillard la meilleure solution que nous trouvons avec Tabou a une meilleure fitness que la meilleure solution que nous trouvons avec le recuit simulé.

En revanche Tabou est plus long à exécuter que le recuit simulé, étant donné que pour ce dernier, à chaque itération le voisin est choisi de manière aléatoire: il n'y a pas de comparaisons à faire avec les autres voisins et la complexité est donc moins grande. Cette rapidité du recuit simulé est compensé par le fait que la recherche des paramètres est plus longue et déterminante: si on n'a pas d'idée des meilleurs paramètres alors Tabou est une meilleure méthode, mais si l'on sait quels paramètres utiliser il vaut mieux partir sur le recuit simulé.