1 Voyageur de commerce

Dans sa version optimisation, le problème du voyageur de commerce consiste, étant donné un ensemble de n villes séparées par des distances données, à trouver le plus court circuit qui relie toutes les villes.

Voici un exemple de donnée.

	A	B	C	D
A	∞	5	7	8
B	10	∞	16	10
C	7	5	∞	2
D	10	11	17	∞

TABLE 1 – Une matrice de distances possible pour un problème de voyageur de commerce avec quatre villes

On se propose d'écrire deux heuristiques différentes pour le TSP, que l'on comparera expérimentalement.

Une solution pour le TSP est une permutation des villes. On demande de construire deux heuristiques qui vont construire cette permutation de manière différente. Les deux heuristiques sont d'abord décrites, le travail à réaliser est décrit plus bas.

1.1 Construction itérative par ajout du plus proche

La première heuristique construit cette permutation ville par ville en partant du début. Elle peut s'écrire de la manière suivante :

1. Initialiser une permutation avec le premier sommet.
2. A chaque étape choisir la plus proche ville non encore sélectionnée et l'ajouter au tour.
3. S'arrêter lorsque l'on a sélectionné toutes les villes.

Dans cette méthode, on a à toute étape un chemin de longueur inférieure ou égale à n reliant la première ville à une autre. L'algorithme s'arrête lorsque le chemin est de taille $n-1$. Il ne faut pas oublier d'ajouter le retour à la ville de départ.

Amélioration de la méthode : tester toutes les villes de départ et garder la meilleure solution.

1.2 Construction par ajout d'arcs

Une autre heuristique peut être utilisée en considérant le TSP comme un problème de graphe. La matrice de distances est alors considérée comme une matrice d'adjacence. On peut alors construire le circuit en ajoutant des arcs itérativement.

1. Trier les arêtes par valeur croissante.
2. A chaque étape, ajouter au tour l'arête courante si elle ne crée par de circuit de taille inférieure à n.

Dans cette méthode, on a à toute étape une liste d'arcs sélectionnés, non obligatoirement consécutifs. Il faut s'assurer que : deux arcs sélectionnés ne partent du même sommet, que deux arcs sélectionnés n'arrivent pas au même sommet, et qu'on ne crée pas de sous-tour (circuit de taille au plus $n-1$).

1.3 Travail à réaliser

Question 1. **[A programmer]** Implémenter ces deux heuristiques (voir ci-dessous *implémentation*).

Question 2. Comparer les deux heuristiques sur le jeu de test donné (voir ci-dessous *implémentation*).

Implémentation

Les méthodes à implémenter sont à écrire dans les classes
— `DecreasingArcHeuristic.java` et
— `InsertHeuristic.java`.

Seule la méthode `double computeSolution(double[][] matrix, List<Integer> solution);` est imposée par l'interface. Vous pouvez ajouter les méthodes et les constructeurs de votre choix.

Vous pourrez comparer vos méthodes avec la classe `TestTSP` qui permet de lire des fichiers de données, et tester des heuristiques en les lançant et en calculant la moyenne des valeurs obtenues.

Un exemple d'utilisation (que vous pouvez éditer) est contenu dans le fichier `MainTSP.java`.

Les données sont contenues dans le répertoire `data/instances`. Pour le debug, vous pouvez utiliser les instances jouet de `data/toy_instances`. Les données sont entrées comme des coordonnées dans le plan, les classes fournies permettent de transformer ces données en matrices de distance.

2 Problème de CVRP

2.1 Rappel du problème

On s'intéresse au problème de tournées de véhicules avec contraintes de capacités.

Données :
— un ensemble de clients à visiter. Chaque client i a une demande d_i
— une matrice de distances entre les clients
— un ensemble infini de véhicules identiques de capacité C
— un unique dépôt 0

Objectif : trouver des tournées pour les véhicules qui minimisent la distance totale parcourue tout en visitant tous les sommets avec une demande non nulle. On notera que l'on n'est pas obligé d'utiliser tous les véhicules.

2.2 Heuristique

On se propose d'utiliser l'heuristique de Clarke et Wright.

Algorithme 1 : Algorithme de Clarke et Wright

Créer une liste vide L de tournées ;
pour *chaque client* i **faire**
| créer une nouvelle tournée contenant uniquement i et l'ajouter à L
fin
while *il reste deux tournées concaténables* **do**
| fusionner les deux tournées de L qui maximisent le gain de la concaténation (saving)
end

Note : pour une implémentation efficace :
— on peut calculer a priori tous les gains possibles en fusionnant une tournée finissant par i et une tournée commençant par j puisqu'ils ne changent pas au cours de l'algorithme
— on peut obtenir le meilleur gain à chaque fois en utilisant la structure de tas (file de priorité)
— il faut créer toutes les structures de données nécessaires pour ne pas générer de la complexité non nécessaire

2.3 Code fourni

On vous fournit le parser Java qui permet de constuire une instance de CVRP à partir d'un fichier de données (`VRPinstance.java`). La documentation est donnée sur moodle.

On vous fournit en plus une implémentation simplifiée de liste chaînée qui permet de concaténer des listes en temps constant (répertoire `util`).

2.4 Travail à réaliser

Question 3. Implémentez l'heuristique de Clarke et Wright pour le CVRP.
Question 4. Testez votre heuristique sur les instances fournies.

2.5 Giant tour

On se propose d'implémenter l'heuristique du "giant tour". Pour cela, on commence par résoudre avec une méthode heuristique ou exacte le problème de voyageur de commerce sur l'instance de CVRP. On découpe ensuite la tournée en plusieurs tours à l'aide de la programmation dynamique.

Algorithme 2 : Algorithme du tour géant

Résoudre le problème de TSP correspondant à la matrice de distances du VRP ;
La solution est une permutation $v_1 = 0, v_2, \dots v_n$;
Construire un sommet par client ;
Ajouter un arc entre un sommet v_i et v_j $(j > i)$ s'il existe une tournée réalisable passant par les sommet $v_{i+1}, \dots, v_j$. Le coût de l'arc est le coût de la tournée;
Résoudre le problème du chemin de plus petite valeur entre v_1 et v_n ;

Question 5. Implémentez l'heuristique du giant tour pour le CVRP.
Question 6. Testez votre programme sur les instances de CVRP.

3 À rendre

Vous devez rendre :
— un document répondant aux différentes questions posées dans le TD
— un rapport suivant le format décrit ci-dessous
— une archive contenant toute l'arborescence du projet, notamment
  — les sources
  — les fichiers .class (y-compris celles fournies)
  — les instances

L'enseignant correcteur doit pouvoir lancer l'exécution du programme en l'état, sans avoir à modifier de chemins ou ajouter de fichiers.

4 Plan du rapport

Le rapport devra impérativement suivre le plan suivant.
— **Introduction.** Brève introduction décrivant le projet.
— **Travail réalisé.** Devra décrire ce qui est implémenté et ce qui ne l'est pas, et donner les détails intéressants d'implémentation (structures de données utilisées, astuces pour améliorer le temps de calcul, etc.).
— **Résultats numériques.** Devra rapporter dans un tableau les résultats obtenus par toutes les méthodes sur tous les jeux d'essai de (temps de calcul et valeur de solution). Des commentaires sur les résultats seront les bienvenus.
— **Conclusions.** Commente le travail réalisé, ce qui pourrait être amélioré, les principales difficultés rencontrées.