SIMULATION ET OPTIMISATION **ISC - HEIG-VD**
J.-F. Hêche, E. Rappos **Automne 2025**

Travail pratique **Heuristiques du plus proche voisin pour le TSP**

Objectifs

L'objectif de ce travail pratique est de programmer les heuristiques « Plus proche voisin », notée NN pour *Nearest Neighbor*, et « Plus proche voisin par les deux bouts », notée DENN pour *Double Ended Nearest Neighbor*, pour le problème du voyageur de commerce symétrique.

Les performances des deux heuristiques seront ensuite évaluer et comparer sur plusieurs jeux de données fournis.

Travail de programmation à effectuer

Vous devez compléter les sources fournies afin de mettre en œuvre les deux heuristiques NN et DENN dont les pseudocodes sont rappelés en tables 1 et 2.

Algorithme 1 Heuristique NN (« Plus proche voisin »)

Données : Un ensemble S de $n \geq 3$ villes, une fonction de distance symétrique $d : S \to \mathbb{R}_+$ et une ville de départ $s \in S$.

Résultat : Une tournée passant une et une seule fois par chacune des n villes.

```
1: procedure NN(S, d, s)
2:     Partir d'une chaîne ne contenant que la ville s
3:     tant que il reste au moins 2 villes à ajouter à la solution faire
4:         Déterminer la ville t, hors solution, la plus proche de s et l'ajouter après s
5:         Poser s = t
6:     fin tant que
7:     Ajouter la dernière ville après s et fermer la chaîne pour obtenir une tournée
8:     Retourner la tournée construite
9: fin procedure
```

Vous veillerez à la structure de vos mise en œuvre qui devront implémenter l'interface `ObservableTspConstructiveHeuristic` mais, également à leur efficacité et, en particulier, à leur complexité dans le pire des cas qui devra être en $O(n^2)$ pour un problème comptant n villes.

Départage des égalités

Dans chaque heuristique, des égalités peuvent apparaître lors de la recherche de la prochaine ville à ajouter à la tournée. Selon le mode de départage de ces égalités, il est possible d'obtenir des solutions différentes mais toutes correctes pour une même ville de départ.

Algorithme 2 Heuristique DENN (« Plus proche voisin par les deux bouts »)

Données : Un ensemble S de $n \geq 3$ villes, une fonction de distance symétrique $d : S \to \mathbb{R}_+$ et une ville de départ $s \in S$.

Résultat : Une tournée passant une et une seule fois par chacune des n villes.

1: **procedure** DENN(S, d, s)
2: Déterminer la ville t la plus proche de s et créer une chaîne avec s et t
▷ la ville s est la tête de la chaîne et la ville t sa queue
3: **tant que** il reste au moins 2 villes à ajouter à la solution **faire**
4: Déterminer la ville v, hors solution, la plus proche de s ou de t
5: Ajouter v à la chaîne actuelle et mettre à jour s ou t selon le cas
6: **fin tant que**
7: Ajouter la dernière ville, entre s et t, afin de fermer la tournée
8: Retourner la tournée construite
9: **fin procedure**

Afin de simplifier les comparaisons, vous retiendrez systématiquement la ville de plus petit numéro lorsque plusieurs villes seront la plus proche d'une extrémité de la tournée en construction et vous effectuerez systématiquement un ajout après la tête s plutôt que l'extrémité t en cas d'égalité dans l'heuristique DENN.

Format des données

Les jeux de données sont stockés dans des fichiers texte d'extension `dat`. La première ligne de chaque fichier contient un seul entier égal au nombre n de villes de l'instance qui doit être au moins égal à 3. Les n lignes suivantes contiennent chacune l'identifiant d'une ville suivi de ses deux coordonnées cartésiennes. Les identifiants sont des entiers allant de 0 à $n-1$ et ils apparaissent dans l'ordre croissant dans chaque fichier. Les coordonnées cartésiennes de chaque ville sont des entiers (normalement positifs ou nuls).

La distance entre deux villes est égale à la distance euclidienne entre les points du plan correspondant aux deux villes mais arrondie, par excès, à l'entier le plus proche.

La classe `TspData` fournie permet la lecture, le stockage et l'exploitation d'un jeu de données à partir d'un fichier respectant le format précédent.

Travail d'analyse à effectuer

Vous évaluerez les deux heuristiques sur les six jeux de données fournis avec l'archive des sources à compléter. Pour chacun d'eux, la longueur de la tournée optimale est donnée dans la table qui suit.

Nom de l'instance	pcb442	att532	u574	pcb1173	nrw1379	u1817
Longueur optimale	50 778	86 729	36 923	56 892	56 638	57 201

Pour chaque instance et chaque heuristique vous calculerez les longueurs des tournées obtenues en utilisant successivement chaque ville de l'instance comme ville de départ. À partir de

ces résultats vous calculerez et afficherez dans la console les statistiques de base (minimum, moyenne, maximum) pour les performances absolues mais également relatives (par rapport à la longueur de la tournée optimale) de chaque heuristique. Vous afficherez également le temps total passé à calculer toutes les tournées, pour chaque instance et chaque heuristique.

Modalités et délais

- ▷ Le travail de programmation est à effectuer par groupe de deux, en Java, version 21 ou 23.
- ▷ L'archive contenant les sources du projet et les jeux de données à étudier, est disponible sur le site Cyberlearn du cours.
- ▷ Vous devez rendre une archive (au format `zip`) contenant toutes les sources de votre projet complété. Vous prêterez une attention toute particulière aux commentaires de vos implémentations des deux heuristiques. Votre archive contiendra également un fichier texte (ou markdown) contenant les statistiques sur les performances des deux heuristiques pour chacun des jeux de données fournis.
- ▷ Vous devez rendre votre travail sur Cyberlearn au plus tard le **dimanche 30 novembre 2025** (avant minuit).