Résolution du problème du voyageur de commerce

Équipe 14

- Thomas CHUNG-HOW
- Sidrit VEJSELI
- Max ROGEL
- Hichem SANHAJI

Description

Le projet consiste à résoudre le problème du voyageur de commerce à l'aide d'un programme codé en C, et ce à l'aide de plusieurs méthodes différentes.

Le code principal se trouve dans `/src/main.c`.

Le code en Python pour les tests se trouve dans `/test`.

Correspondance aux exigences et qualité de conception

Le programme respecte pleinement les exigences demandées en proposant une approche générique du problème du voyageur de commerce. En effet, non seulement les points sont génériques, mais les distances le sont également, ce qui permet de travailler non seulement avec des distances numériques classiques, mais aussi avec d'autres formes de distances telles que, par exemple :

- distances lexicographiques,
- distances polaires,
- distances complexes,
- distances multidimensionnelles,
- …

Une interface générique a été mise en place et est directement accessible à l'utilisateur. Celle-ci est réutilisable, indépendante de toute implémentation concrète des points et des distances, et peut être adaptée ou étendue sans modifier le cœur du programme.

Le projet respecte les principes du Clean Code :

- utilisation correcte du tas (heap) avec une gestion rigoureuse de l'allocation et de la libération de la mémoire,
- prise en compte de la libération de la mémoire même en cas d'interruption du programme,
- respect de l'encapsulation et d'une séparation claire des responsabilités.

Le code peut être analysé à l'aide de Valgrind et ne présente aucune fuite mémoire.

Compilation

```
make
```

Exécution

```
./bin/main
```

Paramètres

Paramètre	Description
`-f <fichier>`	Spécifie le chemin du fichier `.tsp` contenant les données du problème.
`-m <méthode>`	Définit la méthode de résolution à utiliser.
`-o <fichier>`	(Optionnel) Spécifie le fichier de sortie pour enregistrer les résultats.
`-c`	(Optionnel) Affiche la tournée canonique.
`-h`	Affiche l'aide.

Méthodes de résolution (pour `-m`)

Méthode	Description	Complexité
`bf`	Force brute : explore toutes les permutations possibles. Exact mais très lent.	$O(n!)$
`nn`	Plus proche voisin (nearest neighbour) : construit un chemin en choisissant à chaque étape la ville la plus proche.	$O(n^2)$
`rw`	Marche aléatoire (random walk) : génère un chemin aléatoire.	$O(n)$
`2optnn`	2-opt avec initialisation par le plus proche voisin : améliore le chemin initial en décroisant toutes les arrêtes.	$O(n^3)$
`2optrw`	2-opt avec initialisation par la marche aléatoire : idem mais à partir d'un chemin aléatoire.	$O(n^3)$
`ga <nb_individus> <nb_générations> <taux_mutation>`	Algorithme génétique générique : populations d'individus évoluant avec sélection, croisement et mutation.	$O(n^2)$
`gadpx <nb_individus> <nb_générations> <taux_mutation>`	Algorithme génétique avec DPX.	$O(n^2)$
`all`	Toutes les méthodes sauf la force brute.	

Exemples d'exécution

Exemple 1 — Résolution de `att12.tsp` par la méthode de force brute :

```
./bin/main -f data/tsp/att12.tsp -c -m bf
```

Exemple 2 — Résolution de `gr431.tsp` par la méthode génétique générique :

```
./bin/main -f data/tsp/gr431.tsp -c -m ga 20 20 .3
```

Lancement des tests Python

Des tests unitaires sont fournis pour valider le bon fonctionnement du programme principal `main.c`.

Préparation de l'environnement de test

Avant de lancer les tests, il faut configurer l'environnement Python :

```
./test/setup_env.sh
source ./venv/bin/activate
```

Exécution des tests

Une fois l'environnement activé, lancez les tests avec :

```
python3 ./test/test_tsp_c.py
```

Désactivation de l'environnement virtuel

```
deactivate
```

Utilisation des fichiers python

Affichage avec Python

L'affichage interactif représente graphiquement le déroulement des algorithmes en :

- montrant les décroisements à chaque itération pour l'algorithme 2-opt,
- affichant l'ensemble des individus d'une génération (y compris le meilleur) pour l'algorithme génétique.

```
python3 ./test/affichage_interactif.py
```

Analyse statistique des performances de chaque algorithme

Cette commande exécute, pour tous les fichiers ayant un opt.tour correspondant (solution optimale prouvée), l'ensemble des algorithmes de recherche de la meilleure permutation (à l'exception de l'algorithme de force brute) et génère un fichier au format .csv contenant :

- les temps d'exécution,
- les distances obtenues,
- les différences en pourcentage entre la meilleure distance théorique et le résultat trouvé.

```
python3 ./test/analyse_performance.py
```

Affichage des résultats de l'analyse de performance

Après avoir lancé la commande précédante, cette commande permet d'afficher des histogrammes représentant les performances de l'ensemble des algorithmes implémentés, notamment en termes de temps d'exécution et de qualité des solutions.

```
python3 ./test/analyse_performance.py
```