README Equipe 1

Membres de l'équipe (Groupe 1) :

- BEKKAR Nassr-Eddine
- BELGHITI JOUHRI Charaf Edine
- VERNHES Adrien

Compilation et Prérequis :

Compilateur C : gcc

Outils : make

Librairies externes : Aucune librairie à installer. Le projet utilise uniquement les librairies standard (<stdlib.h>, <math.h>, etc.). Possible warning lors de la compilation sur MACOS due a la librairie math.h .

Comment exécuter le programme :

Commande “make” dans le terminal

Commande : ./main-m <méthode de calcul> -f <nom de fichier TSPLIB>

-f <nom de fichier TSPLIB>

(diffèrent fichier TSP sont présent dans le répertoire TSP)

-m <méthode de calcul> parmi :

— bf : force brute

— nn : plus proche voisin (nearest neighbour),

— rw : marche aléatoire (random walk),

— 2optnn : 2-opt avec initialisation par le plus proche voisin

— 2optrw : 2-opt avec initialisation par la marche aléatoire,

— ga <nombre d’individus> <nombre de générations> <taux de mutation> : algorithme génétique

générique,

Exemple de compilation :

./main-m bf-f tsp/att10.tsp

Exemple de sortie :

Instance ; Méthode ; Temps CPU (sec) ; Longueur ; Tour

att10.tsp ; bf ; 0.221611 ; 6178 ; [1,3,2,4,10,5,6,7,9,8]

Structure du Projet :

- Les répertoires **bruteforce**, **genetique** et **heuristiques** contiennent les fichiers .c et .h associés à chaque méthode, portant le nom du répertoire correspondant.
- Le **DPX** est implémenté dans le fichier dédié à l'algorithme génétique.
- La **2-OPT** est implémentée dans le fichier associé à la méthode *Random Walk*.
- Le répertoire **python** contient l'ensemble des scripts Python fournis lors de l'appel d'offres, utilisés pour tester le programme.
- Le répertoire **TAD** regroupe toutes les structures de données que nous avons utilisées.
- Le répertoire **TSP** contient les instances TSP sur lesquelles nous avons effectué nos tests.
- Le répertoire **TESTS** contient l'ensemble des résultats obtenus sur les instances du répertoire TSP ; chaque fichier correspond à une instance et regroupe les sorties obtenues avec les différentes méthodes.
  Par exemple, le fichier *att10.tsp* contient les résultats produits par toutes les méthodes.
  Seule la méthode de force brute n'est pas incluse, par souci de temps d'exécution.
  Ces tests ont été réalisés à l'aide du script run_test.py.
- Le fichier refman.pdf du dossier DocumentTechnique/latex contient l'ensemble de la documentation générée avec doxygen.
- Le fichier **run_test.py** permet d'exécuter automatiquement le programme avec différentes méthodes et de rediriger les sorties vers les fichiers du répertoire **TESTS**.
- Le fichier main permet d'exécuter le programme en faisant appel aux différentes méthodes.

Analyse des performances de notre projet :

Pour avoir une vision d'ensemble, nous avons effectué des tests sur différentes tailles d'instances et sur différents types de distances. *Paramètres utilisés pour les tests comparatifs (GA Simple et GA DPX) : Population 100, Générations 500, Mutation 0.1.*

Petites instances (environ 10 villes) :

Comme le montrent les fichiers dans le dossier TESTS, l'algorithme le plus performant sur cette échelle est le Génétique Simple.

- Il trouve les mêmes résultats optimaux que le DPX.
- Son temps d'exécution est plus faible car la variante DPX demande beaucoup plus de calculs pour reconstruire les liens.
- Les algorithmes heuristiques (NN, RW) sont beaucoup plus rapides, mais trouvent des solutions souvent moins optimisées.

Moyennes instances (entre 20 et 60 villes) :

C'est ici que se situe le point de bascule entre les algorithmes.

- Le GA Simple commence à échouer dans sa recherche de l'optimum.
- Le DPX reste parfait et trouve l'optimum avec un temps d'exécution qui reste raisonnable.
- *Observation :* L'algorithme du Plus Proche Voisin (NN) obtient des résultats assez proches du GA Simple, voire meilleurs sur certains fichiers (comme att48.tsp).

Grandes instances (> 100 villes)

Seul le DPX permet une résolution efficace du TSP à cette échelle.

- La durée d'exécution devient importante (plusieurs secondes à plusieurs minutes).
- Comme pour les moyennes instances, le NN reste compétitif face au GA Simple, qui ne parvient pas à converger avec les paramètres par défaut.

Limitations.

- Brute Force : Cet algorithme garantit une solution parfaite, mais sa complexité factorielle rend son temps d'exécution trop élevé. Il n'est pas "efficace" au-delà de 14 villes.
- Génétique Simple : Plus performant que les heuristiques de base sur de grandes instances, il nécessiterait une optimisation fine de ses paramètres, ce qui augmenterait son temps de calcul.
- Variante DPX : C'est la méthode qui s'approche le plus de la solution optimale, mais elle nécessite une grande puissance de calcul.
    - *Note sur les paramètres :* Les meilleurs résultats pour le DPX ont été obtenus avec les paramètres 300 individus / 2000 générations. Avec cette configuration, l'algorithme atteint d'excellents scores mais nécessite un temps d'exécution long (environ 10 minutes pour l'instance att532.tsp).