**IT45 - Optimisation et recherche opérationnelle**

BENEDUCI Marie

18/04/2023

**Table des matières**

[1. Introduction 3](#_Toc132721949)

[2. Travail préparatoire 4](#_Toc132721950)

[1. Compilation et exécution 4](#_Toc132721951)

[2. Calcul et affichage de la matrice des distances 4](#_Toc132721952)

[3. Construction d’une solution suivant l’heuristique « plus proche voisin » 5](#_Toc132721953)

[3. Algorithme de Little 7](#_Toc132721954)

# Introduction

Ce document est un rapport d’analyse sur un TP réalisé en IT45 à l’UTBM. L’objectif de ce dernier est d’implémenter l’algorithme de Little en C pour résoudre le problème du voyageur de commerce (TSP) et de comparer les performances de différentes méthodes d’optimisation du TSP (solveur/Little principalement) au travers de plusieurs exercices guidées.

Vous trouverez dans ce rapport une explication et une analyse de chaque étape de programmation suivant le TP, suivi d’une description, explication et comparaison des différents programmes.

Le code source du projet ainsi que de ce rapport sont disponibles sur mon Github :

<https://github.com/Mxrie2001/IT45-Little>

# Travail préparatoire

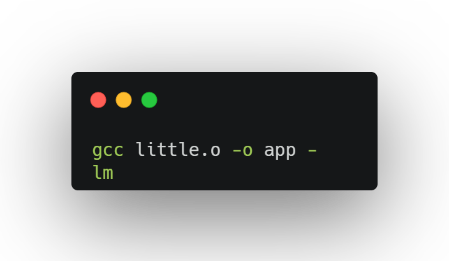
## Compilation et exécution

Dans ce TP, ayant le choix par rapport à la compilation et l’exécution du programme, j’ai choisi de les faire avec les commandes gcc suivantes :

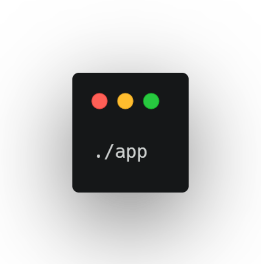
Dans un premier temps il faut créer le fichier .o à partir du .c



Puis créer un fichier exécutable en autorisant toutes les librairies

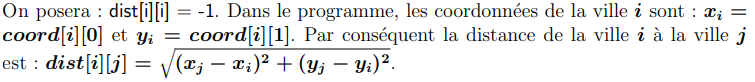


Et enfin, lancer le programme



## Calcul et affichage de la matrice des distances

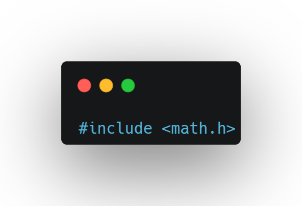
Pour réaliser cette fonction c en se basant sur les calculs de l’énoncé ci-dessous :



Nous arriverons au code suivant :

Une image contenant texte

Description générée automatiquement

Les fonctions pow() et sqrt() étant dans la librairie « math.h », il faut donc l’inclure à notre programme avec . Ces dernières permettent respectivement de mettre à une certaine puissance (ici 2) des données et d’en faire la racine carrée.

## Construction d’une solution suivant l’heuristique « plus proche voisin »

En se basant sur le code et les fonctions déjà existante et en décommentant le code utile, il nous faut, pour réaliser l’heuristique « plus proche voisin » :

* Isoler les villes 2 à 2, la ville de départ(courante) et la prochaine ville
* Vérifier que la ville de destination n’est pas déjà dans le tableau de solution
* Si elle n’est pas dans le tableau de solution et qu’elle est différente de la ville courante, alors si la distance entre ces dernières est inférieure à la distance minimale, la ville de destination entre dans la solution comme prochaine ville
* Sa distance avec la dernière est garder en mémoire comme minimum de distance
* La boucle est répétée jusqu’à ce qu’il n’y ai plus de ville à comparer
* Puis la solution est évalué par la fonction existante evaluation\_solution()
* La fonction de construction d’une solution suivant l’heuristique « plus proche voisin » fini par retenir la meilleure solution grâces aux fonctions déjà existantes et appelées ici.
* La fonction renvoie son évaluation.

N.B. Grace aux « print » nous pouvons avoir un aperçu dans la console des résultats et vérifier que notre fonction fonctionne bien.

Une image contenant texte

Description générée automatiquement

Au niveau du code, nous arrivons donc à cela :

Une image contenant texte

Description générée automatiquement

# Algorithme de Little

Une image contenant texte

Description générée automatiquement

Pour la réalisation de cette fonction en suivant toutes les étapes de l’énoncé et du code donné :

* Dans un premier temps nous vérifions si le nombre d’itération correspond au nombre de ville à évaluer, si c’est le cas, nous construisons une solution, la fonction s’arrête. Sinon la fonction continue
* Ensuite nous créons une copie de la matrice des distances
* On place un 0 sur chaque ligne et colonne de la matrice et on récupère la somme des minimums enlevés grâce à la fonction reduct\_matrix()

Une image contenant texte

Description générée automatiquement

Ici nous allons réduire la matrice à traiter en cherchant le minimum, vérifiant qu’il n’est pas dans la diagonale en temps que -1. Si le minimum =0 nous ne faisons rien. S’il est différent, on le soustrait a la ligne ou colonne de la matrice jusqu’à obtenir un 0 par ligne ou colonnes (en fonction de l’appel de la fonction).

* On vérifie si la solution est optimale
* On recherche le 0 avec la plus grande pénalité grâce à la fonction count\_regrets()

Une image contenant texte

Description générée automatiquement

On va chercher le minimum de la ligne et de la colonne du 0 en question en l’excluant et on additionne les résultats pour avoir la pénalité.

* Ensuite on stock la ligne et la colonne du 0 avec la pénalité maximale dans la ville de départ et d’arrivée
* On vérifie que le max des pénalités n’est pas -1
* On fait les modifications nécessaires sur la copie de la matrice des distances.
* On modifie ensuite la matrice des distances pour voir les possibilités alternatives.

N.B. cette fonction sera lancé en boucle pour avoir le résultat optimal de l’algorithme Little avec les n itérations nécessaires.

# Expérimentation

## Test sur jeux de données de dimension faible (6 et 10 villes)

**Résultat terminal :**

Pour Les 6 premières villes :

Une image contenant calendrier

Description générée automatiquement Une image contenant texte

Description générée automatiquement

Pour les 10 premières villes :

Une image contenant calendrier

Description générée automatiquement Une image contenant texte

Description générée automatiquement

En comparant les résultats du terminal au résultat donnés dans l’énoncé, nous pouvons conclure que l’algorithme fonctionne bien, que ce soit au niveau de la recherche de solution optimale ou encore dans ses différentes étapes au niveau des itérations pour obtenir ici borne minimum, 1er zéro ayant la pénalité la plus forte pour le départ et l’arrivée

## Evolution du temps de recherche quand la dimension du problème augmente

Afin de comparer au mieux les résultats, et d’utiliser le fichier. tsp, une fonction de lecture de fichier avec remplissage de tableau à été rajouté au code :

Une image contenant texte

Description générée automatiquement

De même pour évaluer le temps d’exécution, la fonction clock() est utilisée.



Comparaison :

|  |  |
| --- | --- |
| Nombre de villes | Temps d’exécution (en s) |
| 6 | 0.000278 |
| 11 | 0.009417 |
| 16 | 8.750216 |
| 21 | 153.756221 |
| 26 |  |
| 31 |  |
| 36 |  |
| 41 |  |
| 46 |  |
| 51 |  |

Comparaison

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre de villes | Little C - Temps d’exécution (en s) | Little GLPK - Temps d’exécution (en s) | Little + C - Temps d’exécution (en s) |
| 6 | 0.000278 |  | 0.000184 |
| 11 | 0.009417 |  | 0.009839 |
| 16 | 8.750216 |  | 7.694668 |
| 21 | 153.756221 |  | 147.864480 |
| 26 |  |  |  |
| 31 |  |  |  |
| 36 |  |  |  |
| 41 |  |  |  |
| 46 |  |  |  |
| 51 |  |  |  |