**Projet mÉtaheuristiques, recherche tabou– 2018**

**Chevasson Raphaël- Gramoli lysa- Turpin AurÉlien**

Table des matières

[I. Formulation du problème 1](#_Toc514344571)

[II. Description de l’algorithme 1](#_Toc514344572)

[III. Choix des paramètres intrinsèques optimaux 2](#_Toc514344573)

[IV. Améliorations apportées 2](#_Toc514344574)

[V. Exemples 3](#_Toc514344575)

[VI. Conclusion 3](#_Toc514344576)

# Formulation du problème

Ce projet consiste à résoudre le problème du voyageur de commerce. Pour cela, nous allons utiliser la méthode de recherche Tabou afin de trouver le chemin le plus court pour le voyageur.

En entrée, nous avons une liste de coordonnées pour chacune des villes.

Exemple : Liste\_Villes =

En début d’algorithme, nous aurons une liste contenant le numéro de chacune des villes. Nous aurons aussi une matrice contenant les distances entre deux villes qui permettra d’optimiser le temps de calcul.

Exemple : Pour les données de l’exemple au-dessus, on aurait :

Liste\_numéro\_Villes = [1,3,2] (les numéros sont uniques mais générés aléatoirement)

En sortie d’algorithme, nous aurons le chemin optimal pour parcourir chacune des villes et nous aurons la distance de ce chemin.

# Description de l’algorithme

Étape 1 : Initialisation

Nous avons créé une matrice au début de l’algorithme afin de calculer toutes les distances dès le départ. Cela nous évite de calculer à chaque itération de l’algorithme toutes les distances.

Nous initialisons ensuite la matrice Tabou à 0. Cette matrice sera symétrique car nous mettrons les permutations interdites entre 2 villes. La taille des lignes et des colonnes sera de la taille de la liste contenant les numéros de villes.

Nous calculons une première distance dans l’ordre où sont les villes dans la liste des numéros de villes.

Étape 2 : Itérations

Le but de cette étape est de trouver le chemin optimal au fil des itérations.

Au début de l’itération, nous appelons une fonction qui s’appelle permutation. Elle va prendre en paramètres la matrice de coordonnées des villes, la matrice Tabou et la liste des numéros des villes. Elle va renvoyer en sortie la permutation qu’il faudra réaliser pour améliorer le chemin qui correspond à la liste contenant le numéro des villes. Cette fonction prend aussi en compte les permutations interdites qui sont dans la matrice Tabou.

Cette fonction calcule les distances entre les villes après une permutation et la compare à la distance initiale. Si cet écart est plus petit que les écarts trouvés auparavant, alors on stocke cette permutation dans une liste. S’il y a plusieurs permutations qui offrent le même écart minimal, alors elles seront stockées dans la même liste et la permutation choisie sera tirée au sort.

Après cette fonction, on réalise ensuite la permutation demandée et on met à jour la matrice Tabou pour interdire pendant 10 tours cette permutation.

On stocke le chemin trouvé avec sa distance à la fin de l’itération. On choisit d’itérer 100 fois.

Étape 3 : Fin de l’algorithme

L’algorithme retournera en sortie le chemin optimal trouvé au bout de 100 itérations ainsi que le chemin.

# Choix des paramètres intrinsèques optimaux

Les paramètres intrinsèques optimaux sont :

* Le nombre d’itérations
* La taille de la mémoire

Étude paramétrique :

# Améliorations apportées

A l’origine, nous calculions toutes les distances à chaque itération. Cette méthode ralentissait considérablement le temps de calcul. Pour pallier cela, nous créons une matrice distance qui sera appelée dans la fonction distance afin d’éviter de répéter les calculs.

# Exemples

# Conclusion