**Projet mÉtaheuristiques, recherche tabou– 2018**

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

Table des matières

[I. Formulation du problème 1](#_Toc514371587)

[II. Description de l’algorithme 2](#_Toc514371588)

[III. Choix des paramètres intrinsèques optimaux 3](#_Toc514371589)

[IV. Améliorations apportées 5](#_Toc514371590)

[V. Exemples 5](#_Toc514371591)

[VI. Conclusion 6](#_Toc514371592)

# 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. Cette liste sera le chemin initial que l’algorithme va suivre et effectuer dessus des permutations. 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 = [1,2,3]

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 le temps d’interdiction pour une permutation et le nombre d’itérations.

Nous initialisons le chemin à partir du nombre de villes.

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 contenant les distances entre les 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. 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 un nombre de tours fixé au début de l’algorithme 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

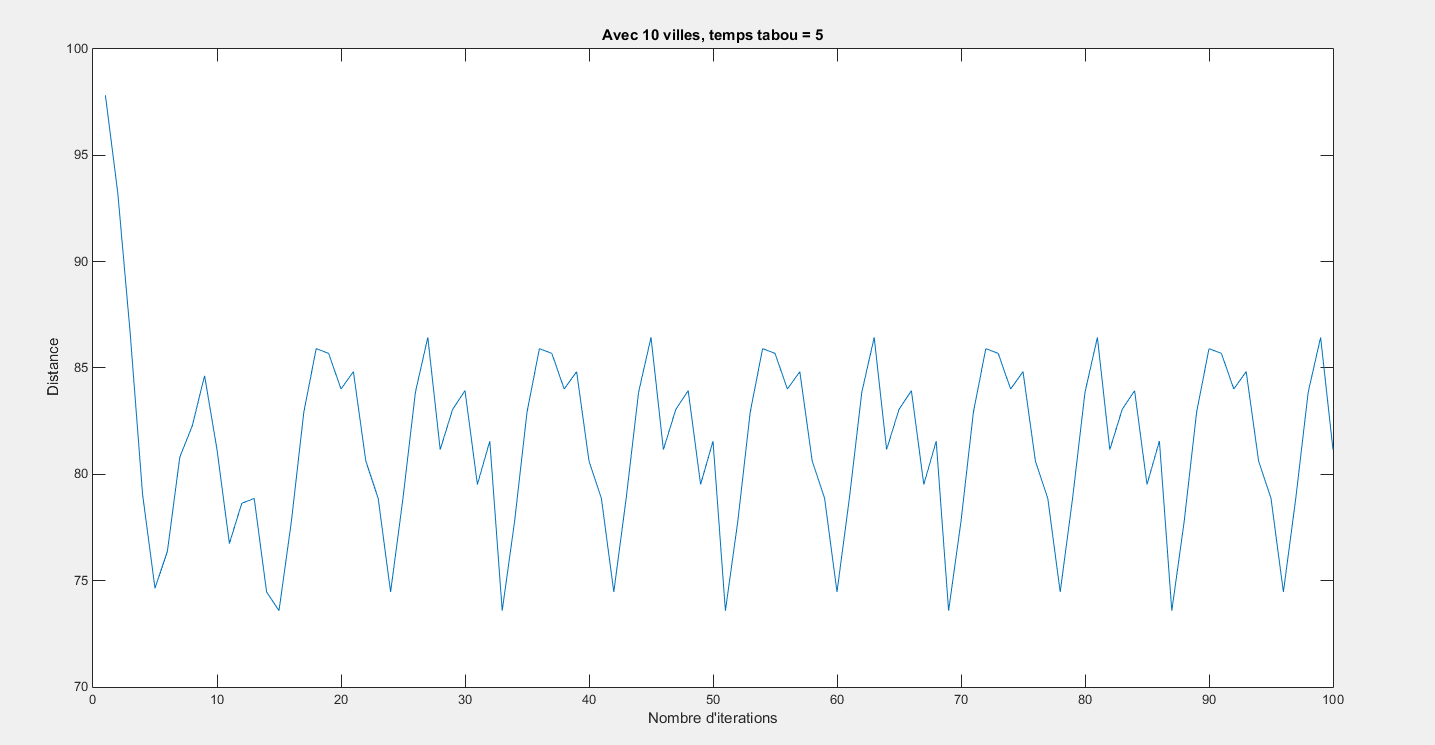
L’algorithme retournera en sortie le chemin optimal trouvé au bout de 100 itérations ainsi que le chemin.

# Choix des paramètres intrinsèques optimaux

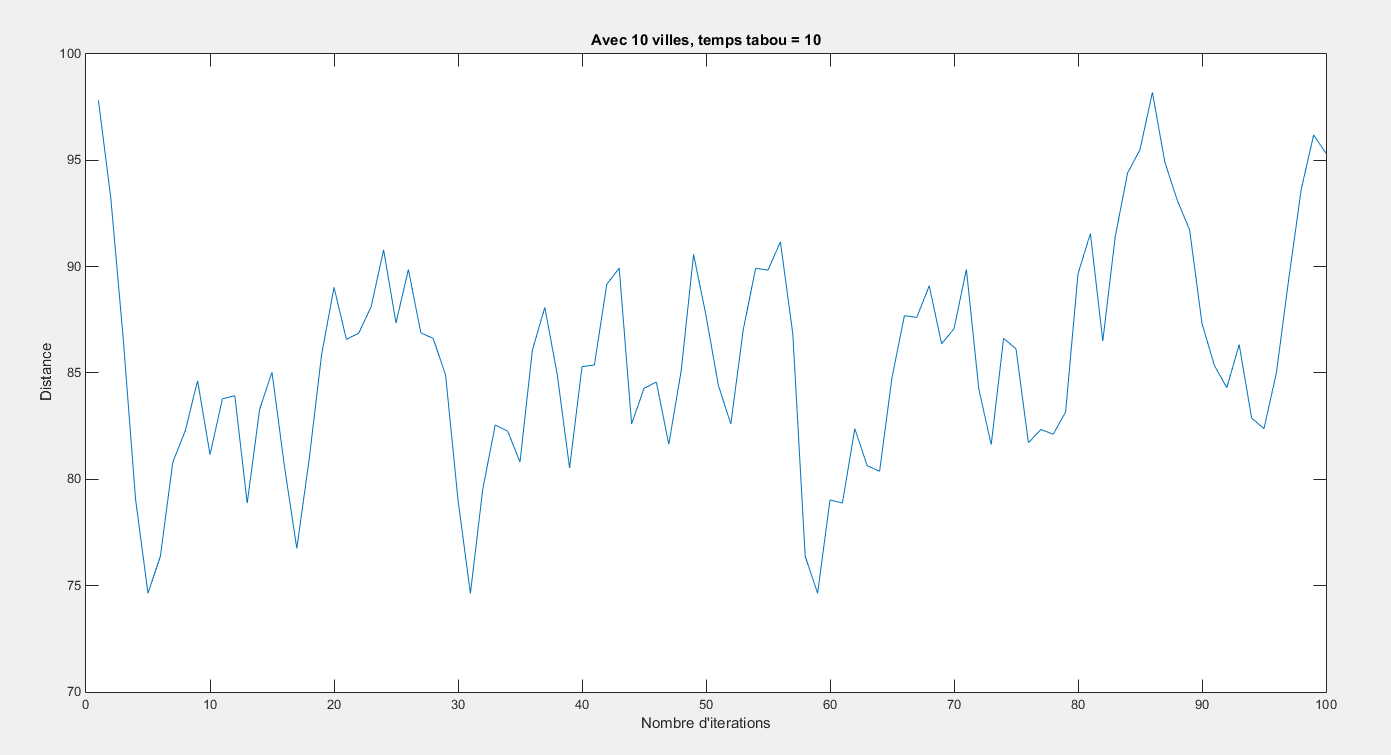
Les paramètres intrinsèques optimaux sont :

* Le nombre d’itérations : Plus il y a d’itérations, plus la probabilité de trouver le chemin optimal augmente. Il faut faire néanmoins attention à ne pas avoir un nombre trop élevé d’itérations qui peuvent conduire à des opérations inutiles voire à des cycles. Après observations, nous avons choisi de prendre 30 itérations pour 10 villes.
* Le nombre d’itérations où la permutation est interdite : suffisamment grand pour éviter les cycles, nous avons choisi de prendre taille\_tabou = 5 pour 10 villes.

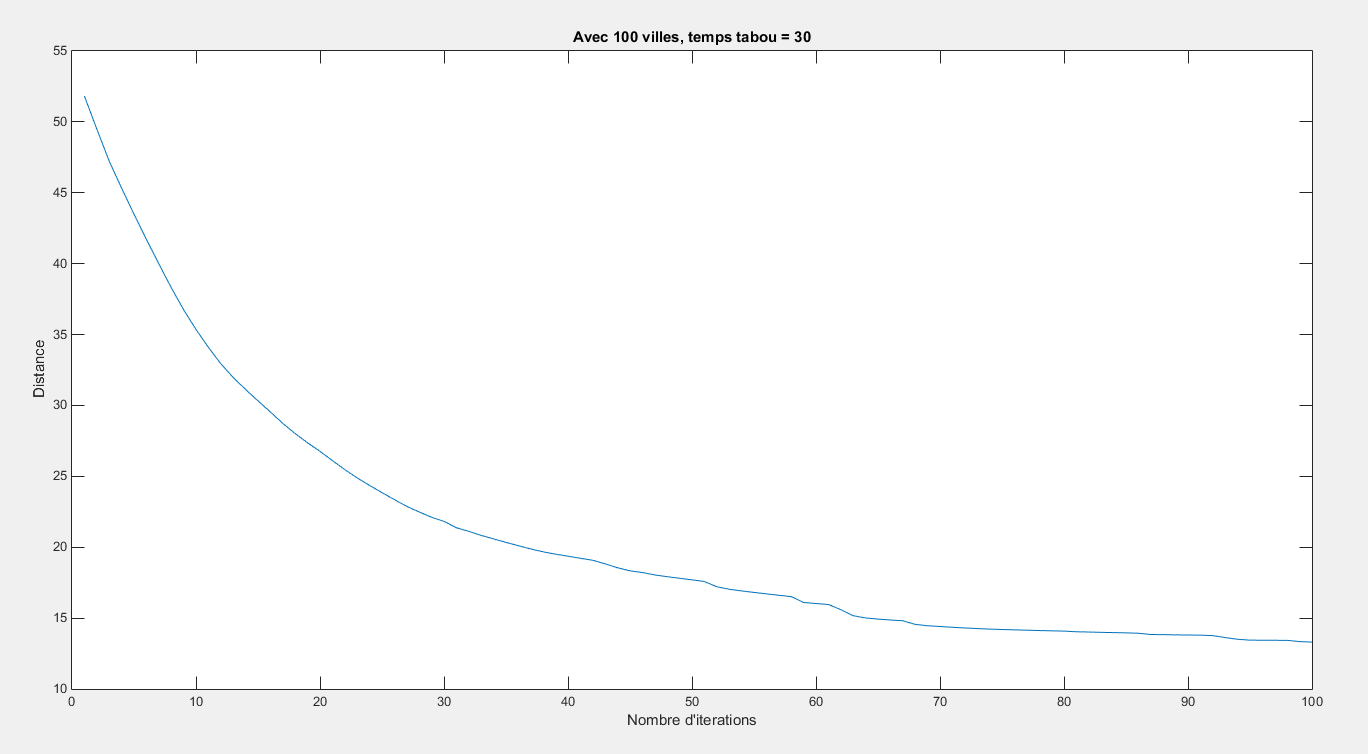
Étude paramétrique :



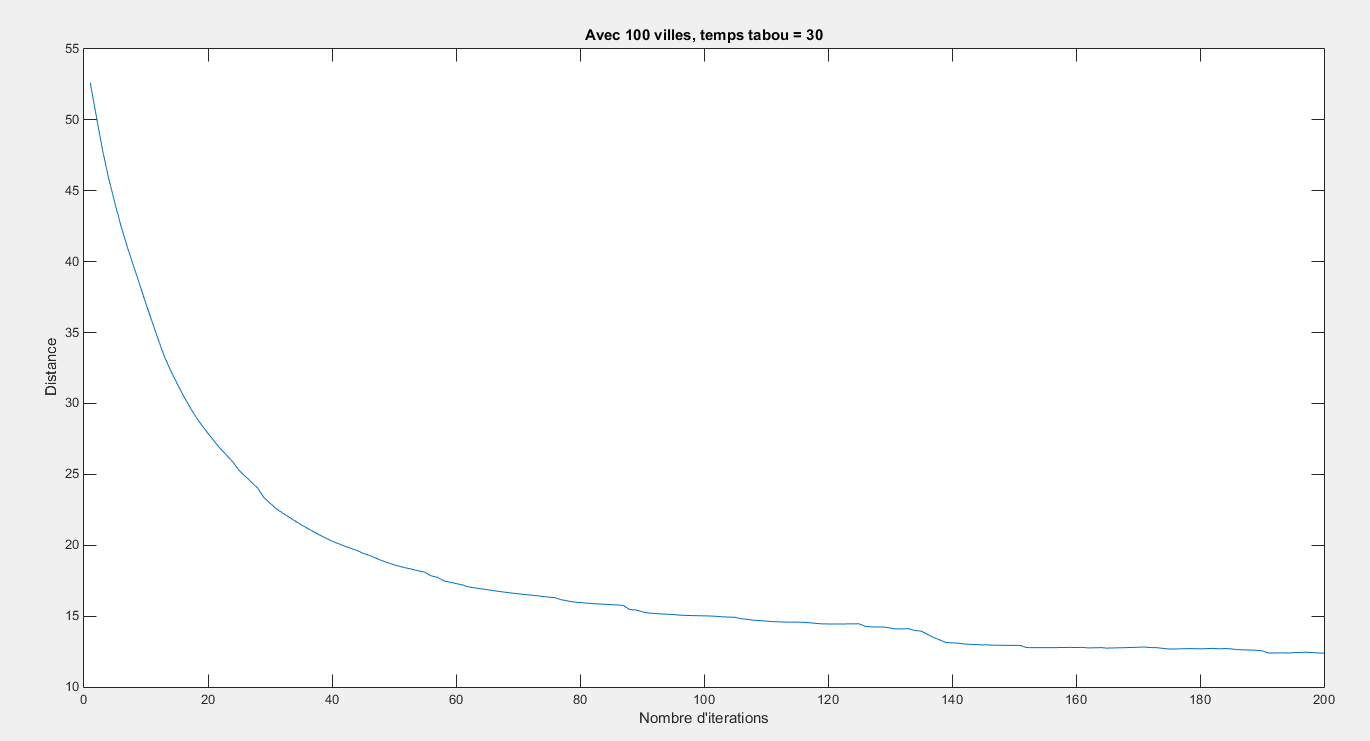
Avec un nombre d’itérations trop grand, on observe l’apparition de cycles. Il faut donc l’adapter au nombre de villes qu’on a en paramètres.



On remarque aussi que lorsque la durée d’interdiction pour une permutation est trop grand (ici c’est égal au nombre de villes), il y a des risques de ne pas aboutir au minimum.



Pour 100 villes, il semblerait que 100 itérations soit trop faible car la distance n’a pas encore atteint de minimum, il faut donc prendre un nombre d’itérations plus grand.



A 200 itérations, il semblerait qu’un seuil a pu être franchi et que la fonction s’est stabilisée. Les coordonnées ayant été générées aléatoirement, il se peut que les villes que leurs coordonnées soient proches, d’où la faible distance.

# 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 avons voulu créer une matrice distance qui sera appelée dans la fonction distance afin d’éviter de répéter les calculs.

On a aussi amélioré la fonction distance : Au lieu de séparer en deux fonctions, nous avons fusionné en une seule, ce qui a réduit le temps d’exécution.

Nous pourrions aussi générer aléatoirement le temps d’interdiction des permutations pour la matrice taboue.

# Exemples

Nous avons pris comme exemple 10 villes avec les coordonnées suivantes que nous avons comparé avec d’autres groupes pour vérifier la cohérence.

Coordonnées : [-6.4400212542671671, -1.4271601981054403, -3.6626197115742221 -2.0061618392897262, 8.7636945696786626, -9.7118148512161433,

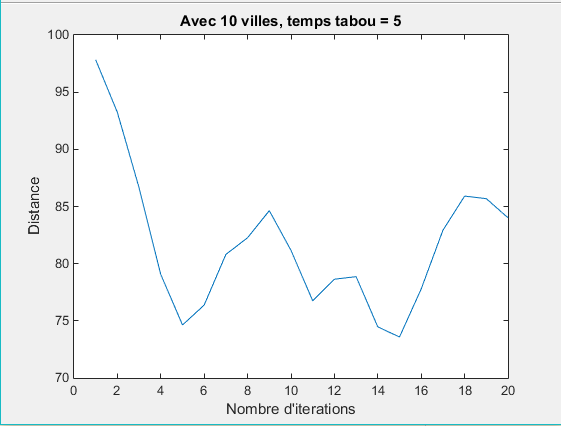
-9.2905548801729978, -7.904169509722589, 1.7480813523593284, 4.8424502818772233;

-3.423109843514216, -9.90426800580618, 6.4200000066541563, -2.3991456316755215, 8.8967270303777148, -8.768145288895564, 8.895629225105818, 1.7406874682312825,-3.341865349713653 1.9537220186720656];

Ordre attendu : 0 [4 9](https://www.facebook.com/groups/1485796531683197/) [2 6](https://www.facebook.com/groups/1485796531683197/) [1 8](https://www.facebook.com/groups/1485796531683197/) [7 3](https://www.facebook.com/groups/1485796531683197/) 5 10 0

Distance attendue : 73.5-74 selon les groupes

Nous obtenons alors :





Cet exemple nous permet de valider la fonctionnalité de l’algorithme.

# Conclusion

On peut conclure que la méthode de recherche tabou a l’avantage de tendre assez rapidement vers le minimum global avec un temps d’exécution rapide. Cependant, si le nombre d’itérations et le temps d’interdiction sont mal réglés, il y a rapidement l’apparition de cycles ou un minimum qui ne pourra plus être atteint. On peut essayer d’améliorer cela, en utilisant l’aspiration afin d’autoriser exceptionnellement certaines permutations afin d’avoir un meilleur minimum plus rapidement. L’autre inconvénient est que cette méthode ne converge pas vers le minimum global, lorsqu’il y a des cycles.