*Application de l’algorithme de fourmis au problème de Voyageur de Commerce*

## **1. Introduction :**

### 1.1 Définitions :

Données fournies :

* Un ensemble fini X de noeuds représentant des villes
* un ensemble fini *U* = {(*i,j*)|*i,j* ∈ *X*} d’arcs reliant les noeuds de X
* un ensemble de constantes *dij* représentant la longueur de chaque arc (i,j) ∈ U , c’est-à-dire la distance séparant la ville i de la ville j (avec i,j ∈ X).

On imagine alors un voyageur de commerce qui doit réaliser sa tournée en visitant une et une seule fois l’ensemble X des villes. Il souhaite déterminer la suite de villes qui minimisera la distance qu’il a à parcourir.

Formalisation:

* la longueur d’un circuit *µ* est la somme des longueurs des arcs qui le composent, soit :

* le TSP (Traveling Salesman Problem) est le problème consistant à trouver un circuit hamiltonien de longueur minimale sur le graphe G = (X,U).

Le voyageur de commerce est un problème NP-complet. On peut appliquer un des algorithmes méta heuristiques tel l’algorithme de fourmi.

## **2. le Système des fourmis :**

### 2.1 Conventions

Les variables que nous devons définir pour comprendre la suite sont les suivantes:

* *bi*(*t*) (*ou i* ∈ *X*) le nombre de fourmis dans la ville i à l’instant t
* *m* leur nombre total, invariant dans le temps
* *τij*(*t*) la valeur (quantité de phéromone)de *τij* , à l’instant t
* *n* = |*X*|, le nombre de villes
*  la visibilité d’une ville j quand on est placé sur la ville i, invariante dans le temps.

Précisons maintenant le comportement de l’ensemble de la colonie. A tout instant t, chaque fourmi choisit une ville de destination selon un choix défini. Toutes les fourmis se placent à l’instant *t*+1 dans une ville de leur choix. On appelle une itération de l’algo, l’ensemble de déplacements de l’ensemble de la colonie entre l’instant *t* et l’instant *t*+1. Ainsi après *n* itérations, l’ensemble de la colonie aura effectué un circuit hamiltonien sur le graphe. De cette manière toutes les fourmis commenceront et finiront leur tour en même temps.

2.2 Choix d’implémentation :

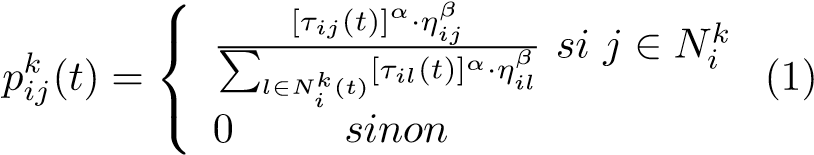
On précise que chaque fourmi a une mémoire implémentée par une liste de villes déjà visitées.

Cela permet de garantir qu’aucune fourmi ne visitera deux fois une même ville au cours de sa recherche.

La mémoire de chaque fourmi est vidée lorsqu’elles ont terminé leur cycle.

Choix des transitions :

Une fourmi k placée sur la ville i à l’instant t va choisir sa ville j de destination en fonction de la visibilité ηij de cette ville et de la quantité de phéromones τij(t) déposée sur l’arc reliant ces deux villes. Ce choix sera réalisé de manière aléatoire, avec une probabilité de choisir la ville j donnée par :



où l’on défini l’ensemble Nik comme étant l’ensemble des villes que la fourmi k, placée sur la ville i, n’a pas encore visité à l’instant t dans le cycle courant. α et β sont deux paramètres qui contrôlent l’importance relative entre phéromones et visibilité. Ainsi si α est égal à 0, le choix se fera uniquement en fonction de la visibilité (si β est différent de zero).

Mise à jour des phéromones :

A la fin de chaque cycle (chaque fourmi a parcouru les n sommets qui composent le graphe), les variables des phéromones sont mises à jour selon la formule :

τij(t + n) = ρ · τij(t) + ∆τij(t) (2)

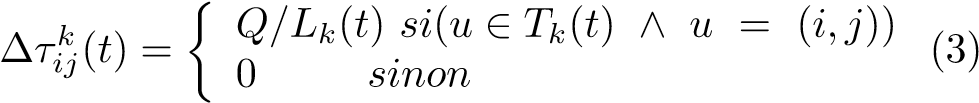
où ρ ∈ [0,1[ est un coefficient qui définira la vitesse d’évaporation des phéromones sur les arcs entre l’instant t et l’instant t+n , et où ∆τij(t) représente la quantité de phéromone déposée par les fourmis dans ce même intervalle de temps sur l’arc (*i,j*).

Le choix de ρ est important, en effet si ρ se rapproche trop de 1, on observe un effet de stagnation des phéromones sur les arcs, ce qui implique des inconvénients tel que le fait de voir les mauvaises solutions persister.

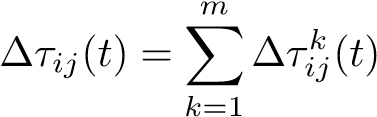
De même, choisir ρ ≈ 0 implique une évaporation trop rapide des phéromones, donc amène la fourmi à un choix dépendant uniquement de la visibilité des nœuds.

Quantités de phéromones déposées :

Appelons *Tk*(*t*) = (*uk*1, *..., ukq*) le tour réalisé par la k-ème fourmi dans l’intervalle de temps [*t, t*+*n*], et *Lk*(*t*) sa longueur. *Tk*(*t*) *(et donc Lk(t))* s’obtient en analysant la mémoire de la fourmi. Soit ∆*τijk* (*t*), la quantité de phéromones déposée par cette fourmi sur l’arc (i,j) dans ce même intervalle de temps. On le définit ainsi :



où Q est un constante. On voit bien ici que les phéromones sont régulés en fonction de la qualité de la solution obtenue car plus *Lk*(*t*) est faible plus l’arc sera mis à jour en phéromones. On peut maintenant définir le ∆*τij*(*t*) de la formule de mise à jour des phéromones ainsi :

 (4)

## **3. Fonctionnement de l’algorithme :**

Nous allons maintenant préciser les différentes étapes de l’algorithme au cours du temps.

Initialisation de l’algorithme. Les éléments s’agencent de la manière suivante au début de l’algorithme :

1. les *m* fourmis sont réparties aléatoirement sur les *n* villes.
2. Pour chaque fourmi, la liste qui modélise sa mémoire contient sa ville de départ.
3. Les pistes de phéromones sont initialisées comme suit : *tij*(0) = *c* , où *c’*est une petite constante positive, qui ne peut être nulle (sinon, il y a un problème lors du calcul de (1))

Fin d’un cycle. Après *n* itérations, nous sommes à l’instant *t*, toutes les fourmis ont terminé leur tour, chacune a une liste "mémoire" pleine et est revenue à sa propre ville de départ. À ce moment :

1. Chaque fourmi calcule sa valeur *Lk*(*t*) .
2. Les variables *τijk* (*t*) sont calculées conformément à la formule (1).
3. Les variables de phéromone *τij*(*t*) sont mises à jour suivant la formule (2). En d’autres termes, la fourmi refait son tour en sens inverse tout en déposant des phéromones.
4. On observe quelle fourmi a trouvé le tour de longueur minimum (i.e. on recherche la fourmi *k* telle que ). Si ce tour est meilleur que le meilleur tour jusqu’ici, on le mémorise.
5. Les mémoires des fourmis (liste des villes visitées) sont effacées.
6. Les fourmis recommencent un nouveau tour, toujours au départ de la ville sur laquelle elles avaient été placées au début de l’algorithme.

Fin de l’algorithme. On arrête l’algorithme après un nombre de cycles égal à une constante *NCmax*. Si à partir d’un instant, toutes les fourmis font le même tour, l’algorithme s’interrompt : on est dans une situation de stagnation où le programme arrête de chercher des alternatives. L’algorithme donne en retour le meilleur tour mémorisé.

Les paramètres permettant de caractériser complètement une instance de AS sont repris dans les tuplet : *< m,p,α,β,Q,NCmax >* ,où 0 ≤ *p <* 1,*α* ≥ 0, *β* ≥ 0 et *c >* 0.

### 3.1 Complexité :

Afin de se faire une idée, nous divisons l’algorithme en 5 sections pour calculer la complexité :

1. initialisation.
2. un cycle de l’algorithme.
3. fin du cycle et calcul des dépôts de phéromone.
4. Evaporation des phéromones.
5. boucle de l’algorithme.

Procédons étape par étape :

1. On a une complexité *O*(|*L*| + *m*) = *O*(*n*2 + *m*), puisque l’on a supposé une interconnexion totale entre les villes.
2. La complexité est *O*(*n*2 · *m*), puisque les opérations de calcul de la ville suivante nécessite un balayage de l’intégralité des villes.
3. La complexité est *O*(|*L*| + *m* · |*L*|) = *O*(*m* · |*L*|) = *O*(*n*2 · *m*).
4. La complexité est *O*(|*L*|) = *O*(*n*2).
5. Le test de stagnation est de complexité *O* (*n* · *m*) (on doit comparer les tours de *m* fourmis, chaque tour ayant une longueur de *n* éléments).

La complexité globale est obtenue en additionnant la complexité de l’étape 1, au produit du nombre total de cycle (soit *NCmax* ) par la complexité globale des étapes 2 à 5. La complexité globale étant celle maximale, soit *O*(*n*2∗*m*) . La complexité générale de l’algorithme devient donc :

*O*(*n*2 + *m* + *NCmax* · *n*2 · *m*)

soit :Algo-complexity=*O*(*NCmax*·*n*2*m*)

Il faut noter cependant que cette formule ne nous dit rien sur le nombre d’étapes qui sont effectivement nécessaires pour atteindre l’optimum : on pourrait atteindre celui-ci bien avant les *NCmax* cycles.