ENSAE ParisTech
Ayoub GHRISS - 1A

Mémoire du projet d'informatique

Algorithme génétique : fourmis

Introduction :

Le but de ce projet est d'implémenter un programme en Python qui permet de chercher un chemin optimal reliant deux points, point de départ (le nid) et un point d'arrivée (la nourriture) dont on ne sait pas la location. Le principe de la recherche doit être basé sur l'algorithme génétique des fourmis.

Plan:

I - Principe d'un algorithme de colonies de fourmis :

Un algorithme de colonies de fourmis est un algorithme itératif à population où tous les individus partagent un savoir commun qui leur permet de guider leurs futurs choix et d'indiquer aux autres individus des directions à suivre. (Dans la suite, on ne s'intéressera pas à détailler la partie graphique de notre programme.)

Cette méthode a pour but de construire les meilleures solutions à partir des éléments qui ont été explorés par d'autres individus. Chaque fois qu'un individu découvre une solution au problème, il enrichit la connaissance collective de la colonie. Ainsi, chaque fois qu'un nouvel individu aura à faire des choix, il pourra s'appuyer sur la connaissance collective pour pondérer ses choix.

Pour reprendre l'explication naturelle : les individus sont des fourmis qui vont se déplacer à la recherche de solutions et qui vont sécréter des phéromones pour indiquer à leurs congénères si un chemin est intéressant ou non. Si un chemin se retrouve fortement « phéromoné », cela signifiera que beaucoup de fourmis l'ont jugé comme faisant partie d'une solution intéressante et que les fourmis suivantes devront la considérer avec intérêt.

Lors la conception d'un tel algorithme, plusieurs problèmes se posent :

- Problème 1 : Eviter de trouver une solution locale : on doit alors permettre à nos fourmis de tenter d'essayer d'autres chemins voisins et permettre ainsi de converger vers un chemin optimal.
- Problème 2 : Assurer que les fourmis ne s'éloignent pas du chemin trouver : les fourmis doivent favoriser les directions contenant plus de phéromones.
- Problème 3 : Permettre aux fourmis d'éviter les anciens chemins et de suivre celui qu'ils viennent de retrouver.

Dans ce qui suit, on déterminera les solutions qu'on a trouvées pour résoudre ces problèmes.

II- L'espace de déplacement :

Notre espace de déplacement est sous forme carré, divisé en cellule carré de taille BlockSize. L'espace est donc de cote résolution*Block Size.*

La cellule s'identifie par ces coordonnées (x,y) et elle peut être en 4 états :

- Contient la nourriture **(self.Food=True).**
- Point de départ **(self.Nest=True).**
- Cellule qui fait part d'un obstacle **(self.obstacle=True).**
- Cellule accessible **(self.Food=self.True=self.obstacle=False).**

Dans notre programme, pour permettre aux fourmis d'explorer d'autres chemins possible et donc éviter une solution locale (problème 1) aucun chemin ne devra être inondé de phéromones et aucun chemin ne devra être totalement invisible, on peut donc aussi contrôler le niveau de phéromone de chaque chemin pour le maintenir entre des bornes minimum (**phero_min**) et maximum (**phero_max**). Un chemin inondé de phéromones masquerait tous les autres à proximité et un chemin pas des tout phéromones ne serait jamais choisi par une fourmi, en conséquence nous devons conserver ces chemins avec des valeurs raisonnables. Ces bornes min et max sont des paramètres de notre programme.

Or, pour permettre d'éviter les anciens chemins, on introduit deux paramètres du système : taux d'évaporation (evaporate) et un paramètre de cellule : la vitesse d'évaporation (evap_time). Donc après evap_time itérations, la phéromone diminue de (1-evaporate).

La quantité de phéromone dans une cellule est contenue dans **self.evaporate.**

```
class Cell:
    def __init__(self, x, y,evaporate,phero_min):
        self.x = x
        self.y = y
        self.Ant=0
        self.pheromone =phero_min
        self.evaporate=evaporate
        self.color = ((0,0,255),(255,0,0))
        self.intensety =0
        self.Food = False
        self.Nest=False
        self.timer=0
        self.obstacle = False
```

On définie ensuite deux méthodes :

def update(self,evap_time,phero_max) : permettant l'évaporation d'une partie de phéromone , et assurant que le phéromone reste dans l'intervalle déterminer par phero_max et phero_min.

def draw(self,screen) : assure l'affichage le couleur de cellule selon son état : phéromone(rouge), Nourriture(vert), Nid (bleu) et obstacle (vert foncé).

III- les fourmis :

On définit chaque fourmis par une classe 'ant'.

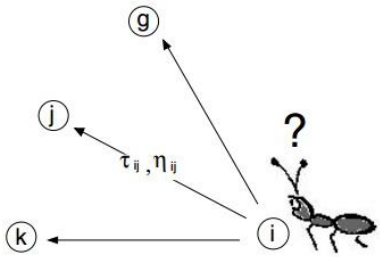
```
class ant:
    def __init__(self,iAnt,alpha,beta,cellGrid,resolution,blockSize,NestLocation,phero_min,distance):
        self.x,self.y = NestLocation[0], NestLocation[1]
        self.alpha=alpha
        self.beta=beta
        self.iAnt=iAnt # l'identifiant de la colonie
        self.CarryFood = False
        self.distance=0
        self.way=[]
        self.away=(0,0)
        self.phero_max=phero_min
        self.back=False
        self.direction=directions.initial_direction((self.x,self.y),cellGrid,resolution,beta,alpha)
```

Nos fourmis sont programmées en deux parties : Déplacement et mémorisation.

1- Le déplacement des fourmis :

Le choix du chemin se fait avec le même principe que dans la nature. Dans la nature, la phéromone est une substance qui permet de créer une piste chimique, sur laquelle les fourmis s'y retrouvent. En effet, d'autres fourmis peuvent détecter les phéromones grâce à des capteurs sur leurs antennes. Les phéromones ont un rôle de marqueur de chemin : quand les fourmis choisissent leur chemin, elles ont tendance à choisir la piste qui porte la plus forte concentration de phéromones. D'autre part, les odeurs peuvent être utilisées par les autres fourmis pour retrouver les sources de nourritures trouvées par leurs congénères. Les fourmis passant à proximité vont avoir tendance à suivre, de façon plus ou moins directe, cette piste.

Et donc, une fourmi ***k*** placée sur la cellule i à l'instant t va choisir sa ville ***j*** de destination en fonction de la visibilité n_{ij} de cette ville et de la quantité de τ_{ij}phéromones 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 :

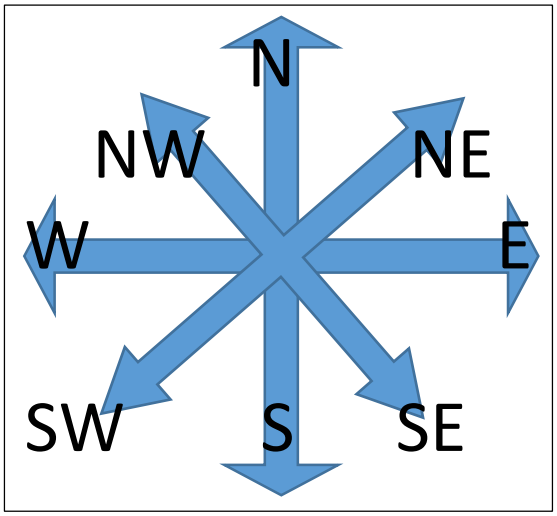


$$P_{i,j}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \epsilon N_i^k(t)} [\tau_{il}(t)]^{\alpha} \cdot \eta_{il}^{\beta}} & si\ j \epsilon N_i^k \\ 0 & sinon \end{cases}$$

- où l'on définit l'ensemble N_i^k 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.

Dans notre programme N_i^k est determiné en fonction de la direction de la fourmi. Ainsi, chaque fourmis prend initialement une direction tel que N_i^k contient toute les directions (N,S,E,W,NE,NW,SE,SW). n_{ij} présente la distance entre la cellule occupée et celle qu'on veut y arrivé, donc $n_{ij} = \sqrt{2}$ si on passe par diagonale, et 1 sinon.

- **α** et **β** sont deux paramètres qui contrôlent l'importance relative entre phéromones et visibilité. On va déterminer le rôle de ces deux paramètres ultérieurement.
- Quand la fourmi quitte le nid, elle ne peut prendre que les directions qui fassent une angle moins de 90 degré avec sa direction précédente. Ainsi, une fourmis venant du sud (direction N) ne peut prendre que les directions : NW, N, NE.
- Chaque fourmi arrivant au nid est réinitialiser (self.back).
- Quand la fourmi parcours s'éloigne des dernières traces de phéromones par une distance supérieure à Away il mort. Away est un paramètre du système, et la fourmi contient un paramètre self.away qui permet de calculer cette distance.
- La paramètre self.CarryFood permet de séparer l'état de recherche de l'état de réussite.



2- Mémoire des fourmis :

Une fois une fourmi a trouvé la nourriture, on distingue deux cas :

- La fourmi est la première à trouver la nourriture : on mémorise le chemin est la longueur du chemin (la méthode remember) et on ajoute des phéromones dans le chemin en fonction de la longueur (la méthode : go_back).
- La fourmi n'est la première à arriver, si son chemin améliore la solution, on le mémorise et en fait comme précédemment, sinon, on garde l'ancien chemin et on le retrace.

Une fois cela est fait, la fourmi est réinitialisée.

La phéromone de l'ancien chemin s'évapore, et chaque fourmi arrivée permet de renouveler le phéromone et d'éviter la disparition du chemin.

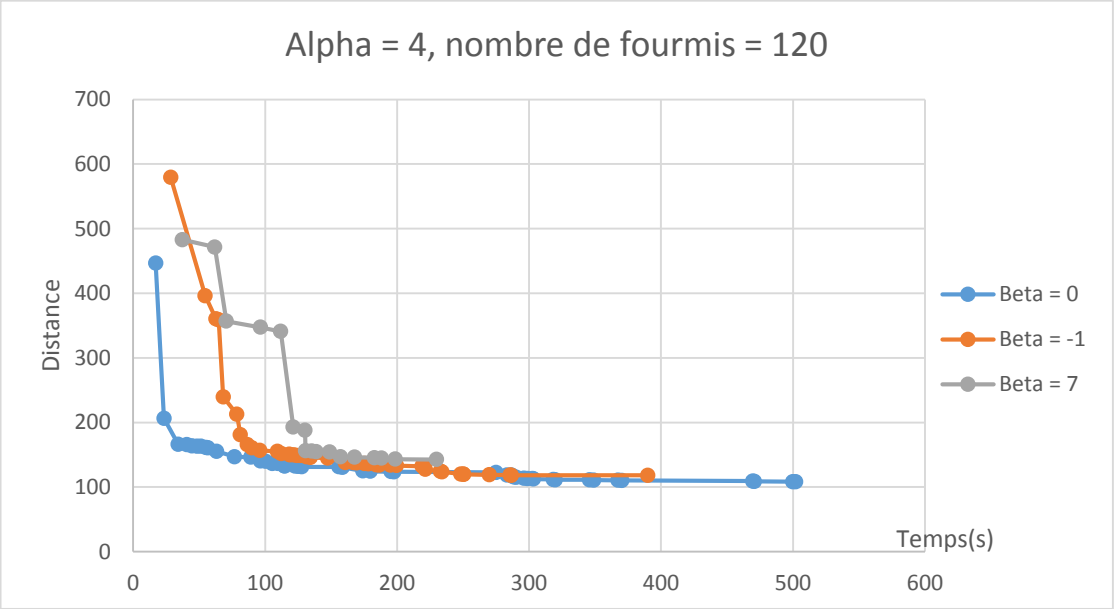
Dans la suite on va interpréter les paramètres du système.

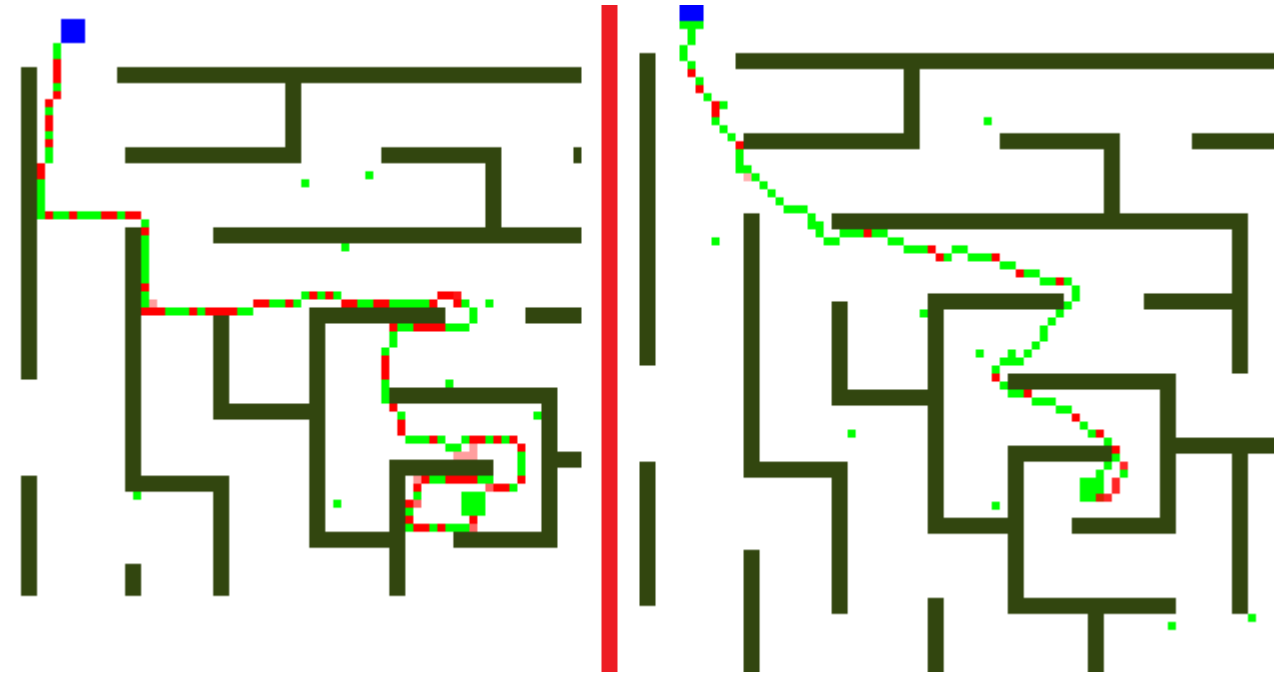
IV- Fonctionnement du programme.

1- Les paramètres du déplacement :

Commençant d'abord par déterminer l'influence des paramètres α et β. Quand la valeur du paramètre α est grande on a moins de fourmis qui s'éloignent des traces de phéromones, quant au β, une grande valeur de β pousse les fourmis à prendre des chemins de plus en plus droits, et d'éviter de passer par la diagonale.

Le graphe suivant calcule la longueur du chemin trouvé pour de différentes valeurs de beta





Beta = 7 Beta = 0

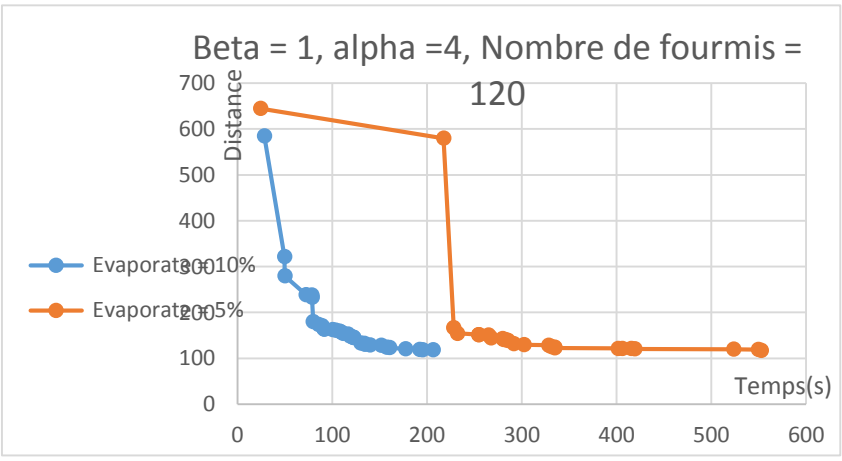
On constate que pour beta =7, les fourmis se content d'une solution locale, car elles parcourent toujours des chemins droits et évitent de passer en diagonale. Cependant, pour beta=0, les fourmis ne distinguent pas entre les directions par visibilités. Pour beta =-1, les fourmis favorisent le passage en diagonale. Cependant, pour beta = 0, la convergence semble plus rapide. On peut aussi comparer graphiquement le comportement des fourmis pour différentes valeurs de beta.

Alpha = 0.5 Alpha = 7

On constate qu'une valeur faible d'alpha disperse les fourmis et donc il est moins probable qu'une fourmi trouve un meilleur chemin. Or pour une grande valeur d'alpha, les fourmis se trouvent piégés dans une solution locale du problème et les fourmis ne quittent jamais le premier chemin distingué. D'après les tests faits, une valeur d'alpha voisine de 4 est optimale.

2- L'évaporation :

Le taux d'évaporation semble influencer la vitesse de convergence. Cela peut paraitre évident, car les fourmis perdent rapidement les traces de phéromones et sont donc obliger à recommencer jusqu'au point on le nombre des fourmis qui arrive à la destination est grand, pour que les phéromones s'accumulent et ne disparaissent rapidement. Cela est inversement vrai pour le temps d'évaporation, les fourmis ne peuvent distinguer entre l'ancien et le nouveau chemin, car les quantités de phéromones sont presque égaules, et la convergence est lente.

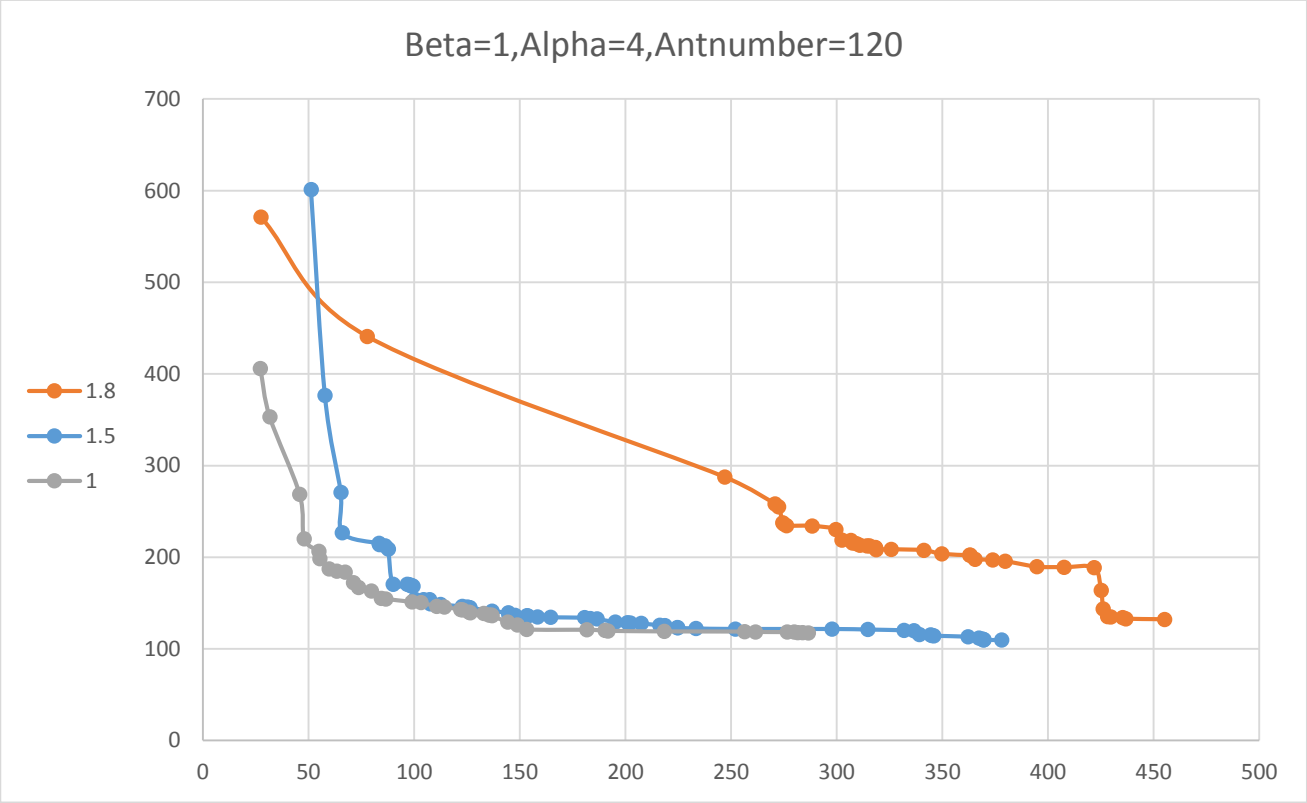


3- Dépôt des phéromones :

On choisit une fonction du dépôt de phéromone de la forme :

$$Q = \frac{Quantité}{longueur\ du\ chemin^{a}}$$

On compare la convergence pour a=1, 1.5 et 1.8 avec Quantité =10^6. Quand la quantité est plus grande, la convergence est plus rapide. Cependant, la phéromone ne dépasse jamais la valeur phero_max, ainsi, déposer la valeur phero_max semble optimale. Cela est confirmé par la nature, en fait, les fourmis déposent la même quantité lors de leur déplacement, et donc la distance qu'une fourmi parcourt n'influence pas la quantité de phéromone qu'elle dépose.



V- Conclusion :

On peut conclure quant aux paramètres optimisant notre programme :

- Valeur d'alpha proche de 4.
- Beta nulle.
- Evaporation d'environ 3%.
- Quantité déposé maximale.

Toutefois, les paramètres optimaux dépendent aussi de la forme de la carte où les fourmis se trouvent : la taille de la surface, les obstacles, et la distance entre nid et nourriture.