Rapport de Travaux Pratiques

BE INF
Groupe B1A2

Bureau d'Étude : coloni de fourmis

Élèves :
Antoine Edy
Mathéo Deville

Enseignant :
Alexandre Saidi

22 avril 2023

Résumé

Table des matières

1 Introduction

Dans ce Bureau d'Étude, nous allons essayer de trouver le chemin le plus court entre deux points en utilisant le modèle d'une colonie de fourmi. Pour cela, nous allons mettre en place un environnement composé d'un nid et d'une source de nourriture ainsi que de plusieurs villes reliées entre elles par des routes. Il y aura donc plusieurs chemins possibles entre la source de nourriture et le nid, le but étant de trouver le plus court.

Le code Python est joint à ce rapport (`code_f_DEVILLE_EDY.ipynb`) sous forme de Notebook Jupyter.

Ensuite, nous allons créer une colonie de fourmi qui seront nos agents pour trouver la meilleure route. Ces fourmis seront amener à évoluer pour trouver que les fourmis aient les meilleurs paramètres et remplissent leur objectif. Notre programme sera réparti en 4 classes Python : la classe **Ant** (une fourmi parmi toutes celles de la population), la classe **Civilisation** (qui correspond à l'environnement global), la classe **Ville** (un point d'intersection des routes), la classe **Route** (qui relie les villes) et enfin la classe **Application** qui permet de lancer notre simulation et de mettre en place une interface graphique. Voici le diagramme UML de notre programme :

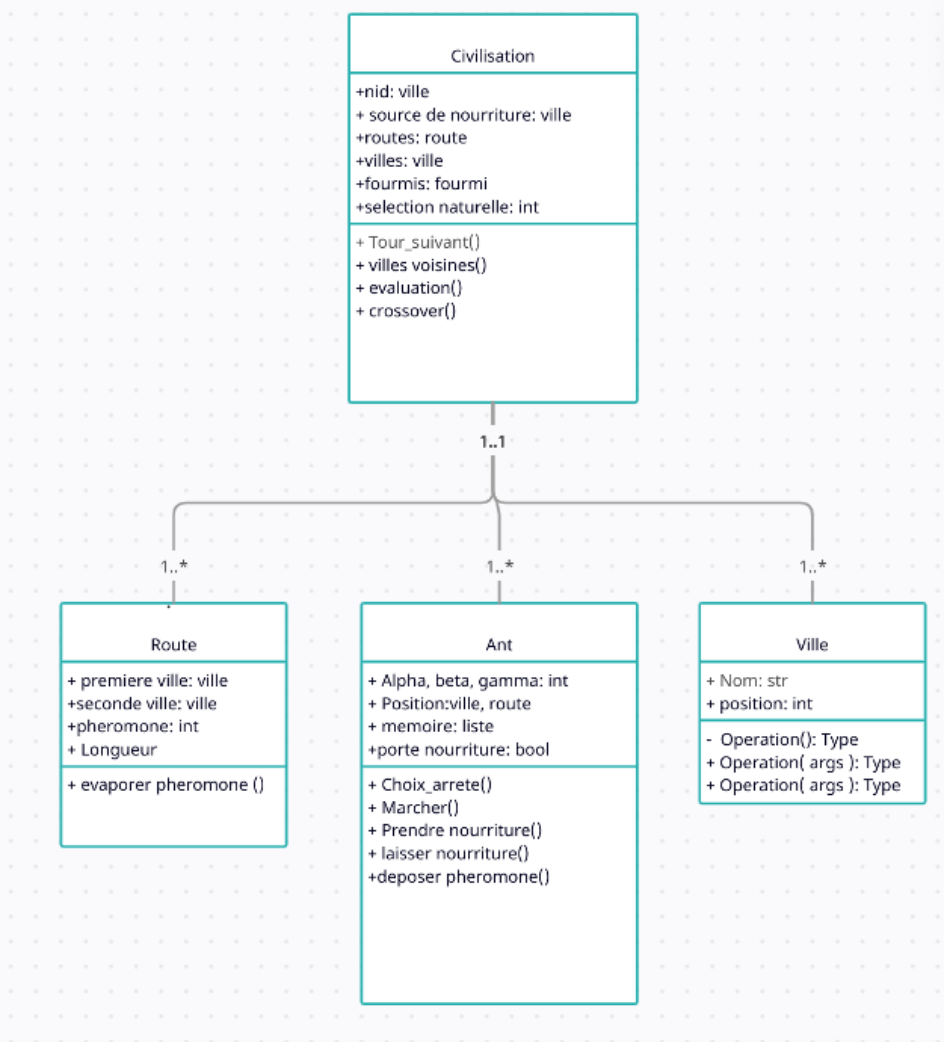


FIGURE 1 – Diagramme UML

2 Classes Route et Ville

Les classes route et ville permettent de créer des éléments routes et villes pour construire "monde" dans lequel notre colonie de fourmi va évoluer.

Le premier élément est une ville, elle n'a que deux attributs, son nom et sa position.

L'élément route est lui un peu plus complexe. Une route nécessite pour sa construction deux villes qui en sont les extrémités, puis une longueur et un niveau de phéromone. En effet, ce niveau est important, car c'est lui qui influe la décision de la fourmi à choisir une route ou une autre. Ce niveau va être modifié par la présence de fourmi qui en dépose en marchant sur cette route et par un phénomène d'évaporation qui fait diminuer le niveau de phéromone sur la route à chaque tour. La seule méthode de la classe correspond à cette tâche et est basée sur la formule suivante :

$$niveau_{t+1} = niveau_t * (1 - \tau)$$

avec τ une constante inférieure à 1.

Voici un exemple de "monde" créer avec ces deux classes :

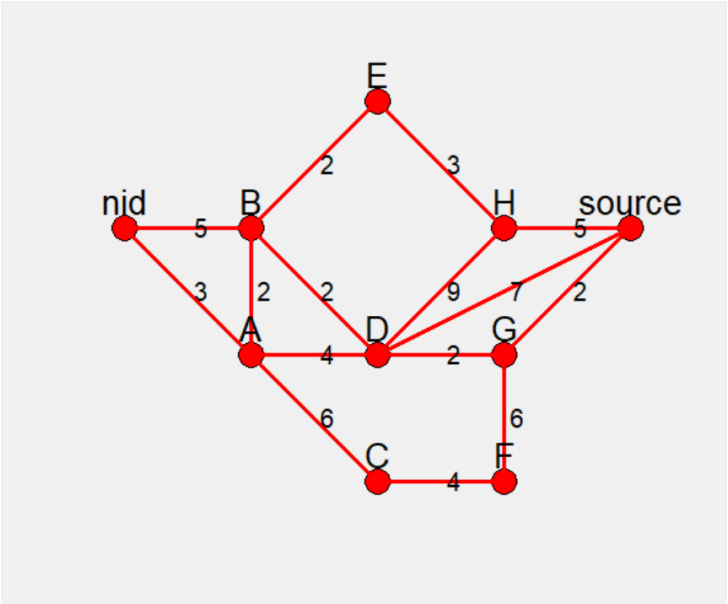


FIGURE 2 – Exemple de monde constitué de route et de ville

3 Classe Ant

La classe Ant nous permet de créer des fourmis. Une fourmi a de nombreux attributs :

- 3 paramètres, α, β, γ, qui entrent en compte dans le choix de la route,
- un booléen, *porte_nourriture*, qui permet de savoir si la fourmi porte de la nourriture,
- une mémoire, *mmoire*, pour savoir quelle ville elle a visité et ainsi lui éviter de tourner en rond. Cette mémoire et remise à zéro à chaque arrivée au nid ou à la source de nourriture,
- la position, *position*, pour connaître la position de la fourmi sur la route et savoir lorsqu'elle a atteint la ville suivante,

— le nombre de pas effectué depuis son départ du nid ou de la source de nourriture, ce paramètre est, lui aussi, utile pour le dépôt de phéromone,
— les paramètres *score* et *score_max* qui considère le nombre de pas effectué lors d'un aller-retour. Le paramètre *score_max* correspond au plus petit score effectué par la fourmi, et donc le meilleur. Il permet de sélectionner les meilleures fourmis pour la partie algorithme génétique.

Concernant les méthodes de cette classe, elle en possède 5. Il y a la méthode *prendre_nourriture* qui consiste simplement à prendre une unité de nourriture pour la transportée. Cette méthode est employée lorsque la fourmi arrive à la source de nourriture. Réciproquement nous avons également *laisser_nourriture* qui consiste à laisser l'unité de nourriture une fois revenu au nid. Une fourmi ne peut porter qu'une unité de nourriture à la fois et doit en avoir pour pouvoir déposer de la nourriture au nid.

Ensuite il y a la méthode *marcher* qui permet à la fourmi d'avancer d'un pas et ainsi d'avancer de ville en ville. A chaque pas, il faut que la fourmi dépose des phéromones pour dire à ses congénères qu'elle a emprunté cette route. Pour cela il y a la méthode *deposer_pheromone* qui permet de remplir cette tâche. Elle intervient à chaque fois que la fourmi fait un pas sur une route. Concernant le niveau déposé il suit cette formule :

$$niveau_{t+1} = niveau_t + \frac{1}{L_{t+1}}$$

avec L_{t+1} la longueur parcouru depuis la dernière venu au nid de la fourmi.

La dernière méthode est la plus complexe, c'est celle qui permet de choisir la route que va emprunter la fourmi quand elle est dans une ville. La méthode fonctionne en deux parties, soit la fourmi choisit la route avec le plus de phéromone soit elle choisit une route au hasard. Ce choix est aléatoire. Dans le premier cas, la fourmi "regarde" toutes les routes disponibles et compare à l'aide de ses paramètres α et β quelles routes à le plus de phéromone. La longueur de la route entre aussi en jeu. Le choix est fait à partir de la formule suivante :

$$\max[taux_i^{\alpha} * (\frac{1}{L_i}^{\beta * C})]$$

avec $taux_i$ le niveau de phéromones sur la route i et L_i la longueur de cette route. Ainsi, nous pouvons voir que chaque paramètre fait peser un facteur différent plus ou moins lourd dans la balance.

4 Classe Civilisation

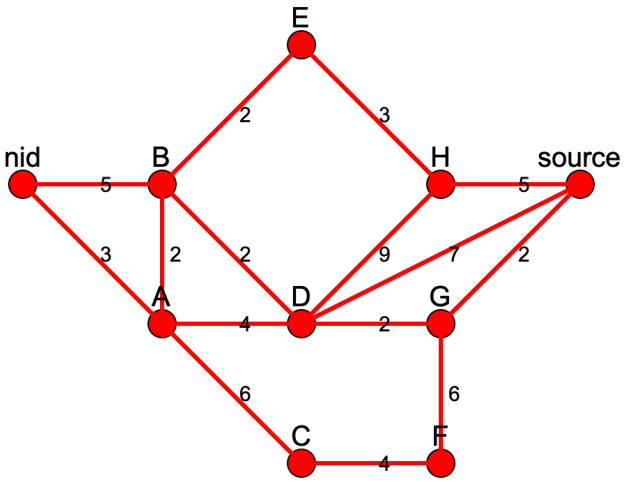
La classe Civilisation est la classe mère de notre projet. Elle est composée d'instance des classes suivantes ville, route et ant. Elle possède aussi la méthode *tour_suivant* qui mène chaque fourmi à faire les actions nécessaire selon sa position. De plus c'est aussi cette méthode qui lance le processus d'évolution génétique si c'est le moment, c'est à dire lorsque le nombre de tours est suffisant. Le processus d'évolution génétique est détaillé dans l'une des parties suivantes.

Cette classe a une dernière méthode qui s'appelle *villes_voisines*, qui prend en argument une ville et nous donne en sortie toutes les villes qui sont reliées par une route à notre entrée.

Ainsi pour créer une civilisation à partir de cette classe il faut une liste de villes, de routes et de fourmis. Ensuite la méthode *tour_suivant* s'occupe de tout.

5 L'évolution graphique de l'environnement

Nous avons décidé d'ajouter une visualisation des routes à l'aide de Tkinter comme expliqué précédemment. Voici un environnement de base :



(a) Environnement au temps $t = 0$

(b) Boutons de l'interface qui permettent d'avancer dans la simulation

Les 4 boutons possèdent les fonctions respectives :
— +1 pas : avancer la simulation d'une itération
— +10 pas : avancer de 10 itérations
— +inf : aller jusqu'au bout de la simulation (1000 iterations dans notre cas)
— Afficher : afficher le taux de phéromone de chaque route

Il nous a semblé pertinent d'ajouter une manière de voir quelles routes sont les plus empruntés, à l'aide d'un code couleur. Ainsi, à chaque itération, on va colorer les routes en fonction de leur taux de phéromone. Un nombre faible de phéromones (proche de 0) équivaut à une route en jaune, alors qu'une route plus empruntée sera rouge.

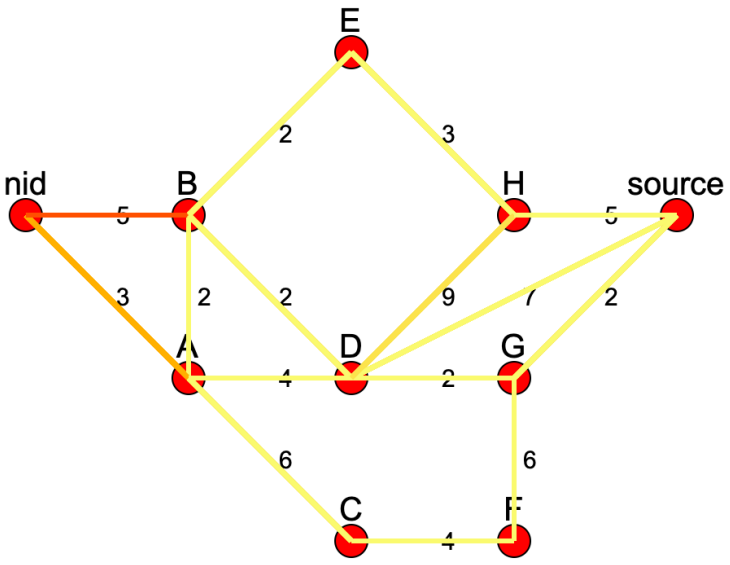


FIGURE 4 – Environnement au bout de 10 itérations

En cliquant sur le bouton "Afficher" :

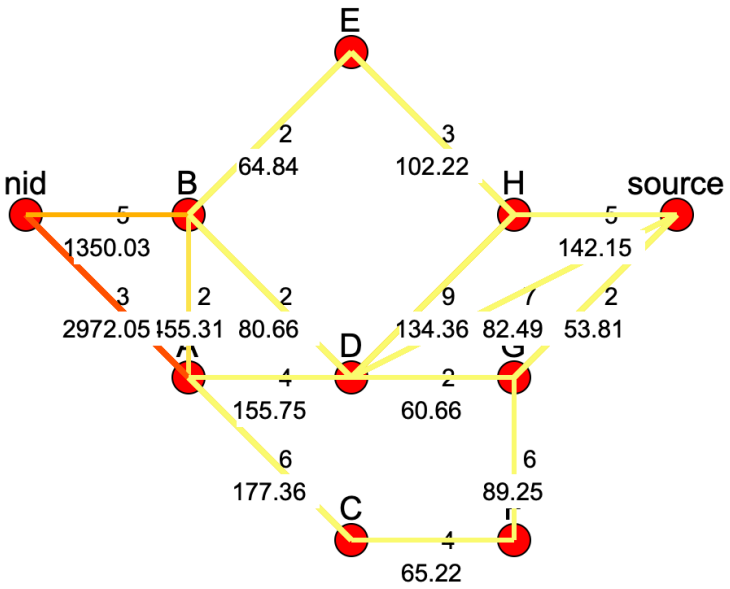


FIGURE 5 – Environnement au bout de 300 itérations avec les taux de phéromone

On remarquera la cohérence entre les taux de phéromone et le chemin le plus court : les fourmis empruntent bien majoritairement le meilleur chemin pour aller du nid à la source et pour revenir.

6 Algorithme génétique

Comme nous l'avons vu précédemment, les paramètres α et β sont très importants en particulier dans le choix de la route à prendre. Mais quelles valeurs doivent-ils prendre ? Pour le savoir, nous allons procéder en plusieurs étapes pour que notre colonie de fourmis ait des paramètres optimaux. En effet, tous les 100 tours, nous allons modifier la génération actuelle. Les méthodes qui concernent cette partie sont dans la classe civilisation.

Dans un premier temps, il faut sélectionner les meilleurs et les pires fourmis de la colonie. Les meilleurs sont les fourmis qui ont effectué l'aller-retour entre le nid et la source de nourriture en faisant le moins de pas. Cette valeur est stockée dans $score\_max$ qui est un attribut que chaque fourmi possède. Ainsi, il faut simplement comparer cette valeur pour toutes les fourmis et stocker l'identifiant des fourmis qui ont le meilleur et le pire $score\_max$.

Une fois la sélection effectuée, il faut faire subir une mutation à notre colonie de fourmi. Pour cela, nous allons moyenner les paramètres α et β des meilleures fourmis. Une fois cette étape effectuée, nous changeons les paramètres des pires fourmis par les valeurs moyennes légèrement modifiées. Cela permet aux fourmis les moins bonnes de devenir meilleur.

6.1 Un premier essai peu concluant

Malheureusement les résultats ne sont pas vraiment concluants, les valeurs se rapprochent de 5, mais ne convergent pas vraiment comme nous pouvons le voir sur la courbe suivante ou nous traçons l'évolution de α (en orange) et β (en bleu) sur 300 générations.

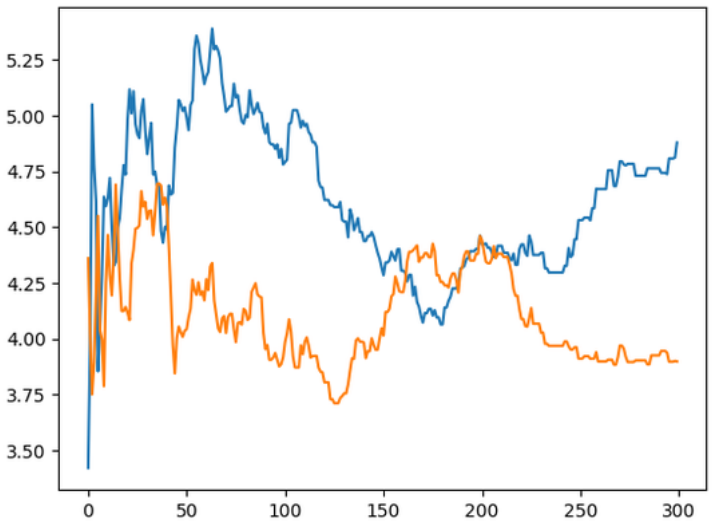


FIGURE 6 – Évolution de α (bleu) et β (orange) au cours des mutations

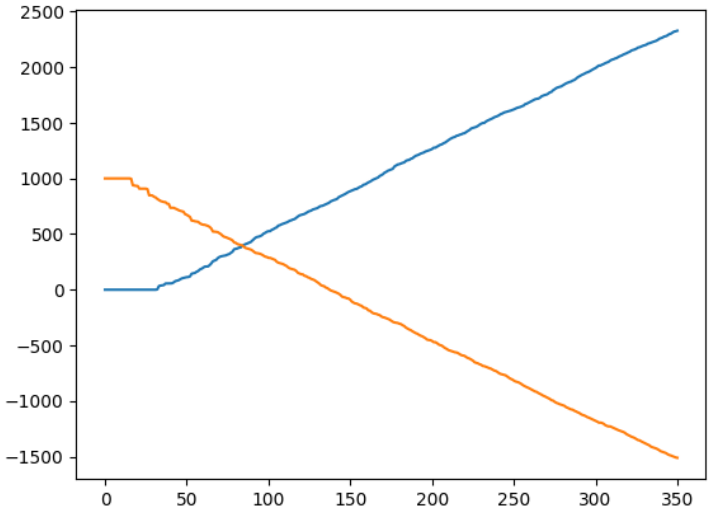


FIGURE 7 – Évolution de la nourriture dans la source (orange) et dans le nid (bleu) en fonction des itérations

On remarque que la nourriture passe de la source au nid de manière linéaire avec le temps : il n'y a pas une évolution satisfaisante du niveau des fourmis, elles ne progressent pas assez avec le temps.

6.2 Un second essai qui l'est davantage

On décide cette fois-ci de faire plus de mutations (toutes les 30 itérations), et de modifier les valeurs prises par α et β (entre -5 et 5 pour les deux variables). On obtient la courbe suivante :

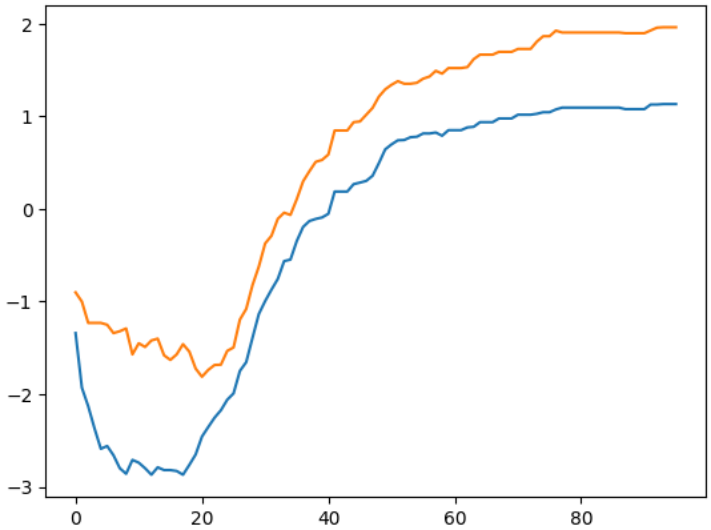


FIGURE 8 – Évolution de α (bleu) et β (orange) au cours des mutations

Plus précisément, on trouve que :

$$\begin{cases} \alpha \longrightarrow 1.0831 \\ \beta \longrightarrow 2.2824 \end{cases}$$

Qui semble être les valeurs optimales. On trouve en effet que les fourmis remplissent le nid plus rapidement avec ces valeurs qu'avec d'autres.

Valeurs de α et β	Nourriture dans le nid	Itération
Valeurs optimales (α = 1.0831 et β = 2.2824)	1000	211
Valeurs aléatoires (α = -1 et β = -1)	1000	323

Soit une diminution de $\frac{323-211}{323} = 34.67\%$, ce qui est un résultat satisfaisant.

7 Conclusion

Nous avons étudié et implémenté un algorithme de recherche du plus court chemin basé sur la méthode de colonie de fourmis. Cette méthode s'inspire du comportement des fourmis lors de leur recherche de nourriture pour trouver le chemin le plus court entre deux points. Notre implémentation de cet algorithme a montré des résultats prometteurs comme vu en fin de la dernière partie. L'algorithme est globalement capable de trouver des chemins optimaux pour passer d'un point de départ à un point d'arrivée, et les fourmis s'améliorent avec le temps grâce à l'algorithme génétique mis en place.

Cependant, notre algorithme présente encore quelques limites, qui sont principalement dues aux paramètres de la simulation. En effet, nous n'avons pas réussi à trouver des intervalles optimaux pour plusieurs variables, notamment τ le taux d'évaporation de phéromone, N_{pop} la population de fourmi, $New_{generation}$ le nombre d'itérations avant d'actualiser les fourmis, ou encore, dans une moindre mesure, l'intervalle de α et β ($[-5, 5]$ ayant été choisis empiriquement). Ces paramètres ont une forte influence sur la qualité de la simulation et sont des points à étudier plus en profondeur.

En fin de compte, cet algorithme de colonie de fourmis que nous avons mis en place est intéressant pour résoudre des problèmes de plus court chemin, mais il est important de prendre en compte ses limites et de les surmonter pour obtenir des résultats optimaux.