ÉCOLE CENTRALE LYON

Electif Algorithmes collaboratifs et applications

Rapport BE

Résolution du problème "BUS" à l'aide d'un algorithme génétique

Auteur :
M. Valentin VERDON

Encadrant :
M. Alexandre SAIDI

Version du
15 avril 2023

Table des matières

1 Introduction

Le transport en commun est un élément vital dans la plupart des villes du monde, permettant aux citoyens de se déplacer facilement et efficacement d'un endroit à l'autre. Cependant, la planification et la gestion des réseaux de transport en commun peuvent être complexes et coûteuses, surtout lorsqu'il s'agit de planifier les itinéraires et les horaires des bus. Le problème des bus consiste à trouver l'itinéraire optimal pour chaque bus afin de minimiser les coûts et d'optimiser le temps de trajet pour les passagers. Pour résoudre ce problème, une approche prometteuse est l'utilisation d'algorithmes génétiques, une technique de recherche inspirée par l'évolution naturelle qui peut être appliquée à une variété de problèmes d'optimisation. Ce rapport présente l'implémentation d'un algorithme génétique pour la résolution du problème des bus issue de l'article *Multiple Ant Colony Systems for the Busstop Allocation Problem* de **Jasper de Jong** et **Marco Wiering**.

2 Présentation de l'algorithme

L'idée qui se cache derrière notre algorithme est issue de l'observation des fourmis. Cela consiste à envoyer des éclaireurs qui à chaque arrêt vont déposer une quantité constante de phéromones. Ces derniers s'évaporent au cours du temps et sont détectés par les autres fourmis. Ainsi, les fourmis seront attirées vers les chemins qu'ont le plus de phéromones et ces derniers vont continuer à accroître leur quantité de phéromone. Finalement, le chemin qui sera choisi est celui qui aura été le plus emprunter (et donc celui qui semble être le plus performant). Pour plus de détails concernant la logique et les équations derrière cet algorithme, consultez l'ouvrage **[1]**.

Voici le pseudo code qui sera implémenté (issue de l'article cité ci-dessus) :

1 Set $t = 0$
  For $l = 1$ to m do
    Set all edge pheromone levels τ_{ij}^{l} to τ_0

2 For $l = 1$ to m do
    For $k = 1$ to r do
      Choose busstop j_{start}^{l} with probability $p_{sj_{start}^{l}}^{kl}$ given by equation 2
      $J_k = \{j1, ..., jn\} - j_{start}^{l}$
      Update pheromone level of edge (s, j_{start}^{l}) using equation 3

3 For $w = 1$ to $n - 1$ do
    For $k = 1$ to r do
      Choose random q $(0 \leq q \leq 1)$
      If $q \leq q_0$ then
        For $l = 1$ to m do
          Store busstop with its value using equation 1
        Choose busstop j with largest arg max of all stored busstops
      Else
        For $l = 1$ to m do
          Choose and store busstop using equation 2
        Choose busstop j at random from stored busstops
      Set $J_k = J_k - j$ and $S_k = S_k + j$
      Perform local update on last edge according to equation 3

4 For $k = 1$ to r do
    Compute L_k for each solution S_k, and update S_{gb} and L_{gb} if $L_k < L_{gb}$
  For $l = 1$ to m do
    For all edges (i,j) $\in S_{gb}$ do
      Update edge according to equation 4
  Set $t = t + 1$

5 If $t < t_{max}$ then
    Goto step 2

FIGURE 1 – Pseudo code

Concrètement, notre algorithme reçoit une situation avec les coordonnées des différents arrêts de bus, la position dans cette liste de l'arrêt de bus principale, le nombre de lignes que nous souhaitons mettre en place, une liste de passagers avec leur destination de départ et d'arrivée (certains arrêts sont davantage empruntés par rapport à d'autre, il faut donc que notre algorithme prenne cela en compte pour donner la solution la plus optimale). Il prend également un certain nombre d'hyper-paramètres pour enfin retourner une liste contenant les différentes lignes de bus avec les arrêts les constituant (dans l'ordre) ainsi que le temps de parcours moyen d'un passager.

3 Implémentation de la solution

3.1 Présentation du code

Pour implémenter ce programme, nous avons créé deux classes. La première porte sur la situation considérée en prenant comme variable le nombre de lignes de bus, les coordonnées des arrêts, la liste des départs et arrivés des passagers ainsi que l'arrêt de bus principal.

La seconde classe permet de créer une simulation et renvoie une solution pour une situation donnée. Elle prend donc comme paramètres ceux nécessaires à la simulation, soit :

- τ_0 : le taux de phéromone initiale
- r : le nombre de fourmis par ligne de bus (si on numérote les fourmis pour chaque ligne, les groupes de fourmis ayant le même numéro formeront une solution)
- β : paramètre entre 0 et $+\infty$ qui permet de privilégier la visibilité des arrêts de bus ou bien les phéromones
- ρ : le taux d'évaporation des phéromones entre 0 et 1
- α : le pas de progression
- v : la vitesse du bus (comme elle est la même pour chaque ligne, cela ne joue pas dans la résolution, mais on pourrait imaginer un algo qui prend en compte une vitesse différente dans certains secteurs)
- q_0 : paramètre entre 0 et 1 qui régule l'importance relative entre exploration et exploitation
- t_{max} : le nombre d'itérations d'une simulation avant de retourner la solution (doit être assez grand pour que la simulation converge, mais pas trop pour que le programme ne dure pas trop longtemps).

Afin de résumer le programme, et ce, de façon lisible, voisin le diagramme de classe associé :

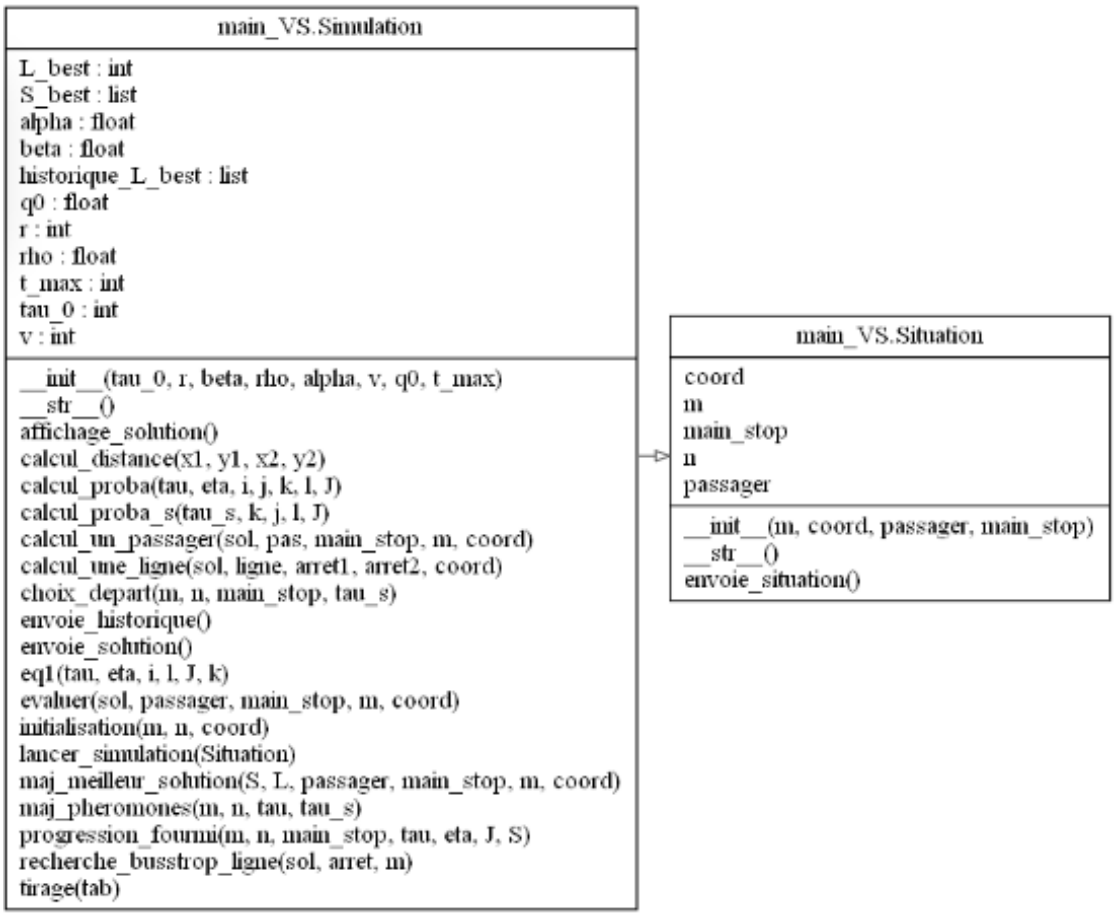

FIGURE 2 – Diagramme UML

On remarquera que notre code suppose que si le passager a besoin de changer de ligne, le changement est immédiat à la main station (pas de temps d'attente).

3.2 Étude sur un exemple

Avant d'étudier en profondeur la performance de notre solution, commençons par vérifier qu'elle fonctionne en s'appuyant sur un exemple.

Prenons comme situation les arrêts de bus suivant avec comme gare principale la station E :

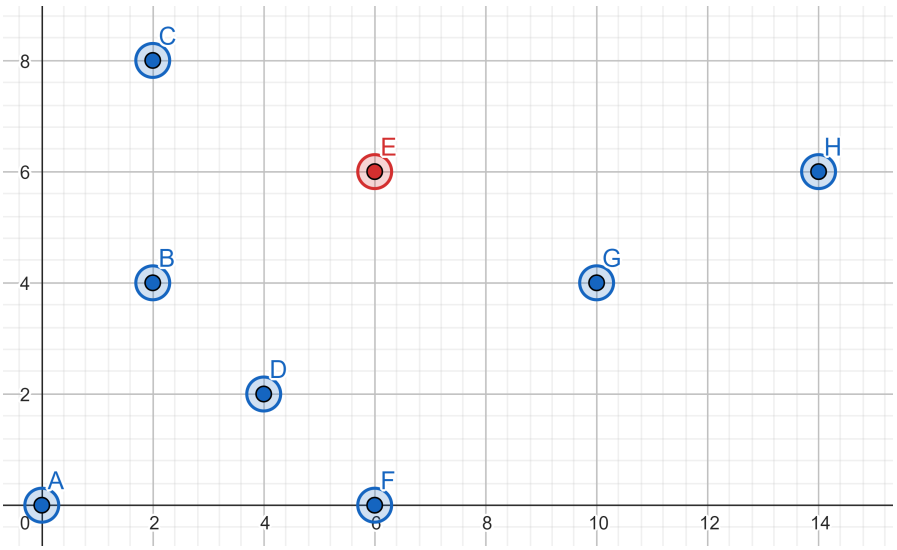

FIGURE 3 – Exemple de Situation

Nous choisissons une répartition aléatoire de 100 passagers entre ces gares (on crée pour cela une petite fonction qui choisit au hasard une gare de départ et d'arrivé en faisant attention que celle-ci soit distinct).

Remarque : pour que vous puissiez réaliser les mêmes simulations, nous avons mis le paramètre *random state* = 42.

On gardera les paramètres du modèle de base, à savoir :

paramètre	valeur
τ_0	1
r	10
β	0.5
ρ	0.1
α	0.1
v	1
q_0	0.5
t_{max}	100

TABLE 1 – Paramètres du modèle

Nous obtenons alors le résultat suivant pour un temps de parcours moyen de $L = 9.6$:

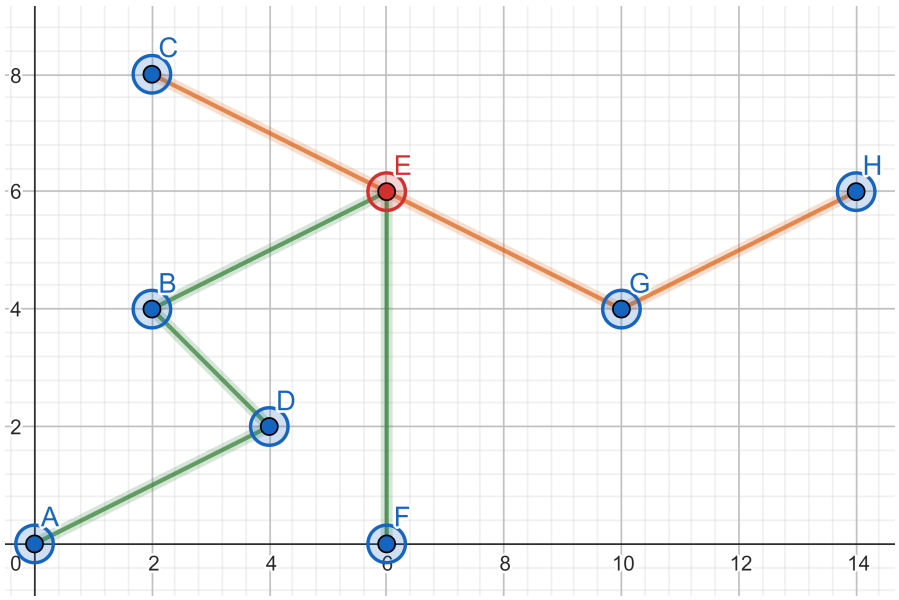

FIGURE 4 – Résultat

Notre programme fonctionne ! Maintenant, penchons-nous un peu plus sur ses performances !

4 Étude des performances

4.1 Temps d'exécution et convergence

Dans un premier temps, nous allons étudier le temps d'exécution de notre algorithme ainsi que vérifier que celui-ci converge bien.

Pour réaliser ces tests, nous allons prendre les paramètres suivants : 3 lignes de bus, 20 arrêts de bus (générés aléatoirement avec la possibilité de fixé le *random state* (ici à la valeur 42)), 100 passagers, et les paramètres de base de la classe *Simulation*. On réalisera 100 simulations afin d'avoir une approximation correcte du temps de calcul.

Remarque : afin de visualiser la situation ainsi que les résultats obtenus par une simulation, nous avons ajouté les fonctions *affichage situation depart* et *affichage situation finale* qui, pour la première, affiche la position des arrêts de bus pour une liste de cordonnées, et pour la seconde affiche les arrêts de bus avec les lignes trouvées par la simulation. (Cette option n'étant pas importante pour

l'étude du sujet, nous ne ferons que l'évoquer et laissons le lecteur tester ces fonctions à l'aide des scripts python fournies en pièce jointe).

Nous obtenons comme résultat un temps d'exécution de **6.70 sec en moyenne**. Ce temps de calcul peut servir de référence si on étudie la complexité du programme.

La complexité peut être écrite sous la forme :

$$\mathcal{O}_{Simulation} = \mathcal{O}_{Initialisation} + \mathcal{O}(t_{max})\mathcal{O}_{uncycle}$$

avec :

$$\mathcal{O}_{uncycle} = (\mathcal{O}_{choixdepart} + \mathcal{O}_{progressionfourmi} + \mathcal{O}_{majmeilleuresolution} + \mathcal{O}_{majpheromones})$$

Le calcul des complexités est résumé dans le tableau suivant :

Fonction	Complexité
initialisation	$\mathcal{O}(m.n^2)$
choix départ	$\mathcal{O}(m.r.n^3)$
progression fourmi	$\mathcal{O}(m.r.n^4)$
maj meilleure solution	$\mathcal{O}(p.r.n^2)$
maj phéromones	$\mathcal{O}(m.n^2)$

TABLE 2 – Résumé des complexités

En prenant comme notation :
— m : nombre de lignes de bus,
— n : nombre d'arrêts de bus,
— r : le nombre de fourmis par ligne,
— p : le nombre de passagers

Cela nous permet d'obtenir la complexité finale de notre algorithme :

$$\mathcal{O}_{Simulation} = \mathcal{O}(t_{max}.m.r.n^2(p + n^2))$$

Remarque : pour réaliser ces calculs, nous avons fait deux approximations :

1. m«n : on suppose que nous avons largement plus d'arrêts que de lignes de bus (cela nous permet de simplifier les sommes n + m en n
2. on suppose que chaque ligne du tableau des arrêts de bus qui reste à visiter est un $\mathcal{O}(n)$ car en moyenne chaque est de longueur $\frac{n}{2}$ soit un $\mathcal{O}(n)$

Cette étude de la complexité nous permet de prendre conscience de l'impact du nombre d'arrêts de bus considéré dans le problème. Cependant, le code pourrait être modifié afin d'améliorer sa complexité.

Nous allons maintenant vérifier la convergence de notre algorithme. Pour cela, nous allons stocker le meilleur temps moyen à chaque itération puis afficher la courbe issue de cette liste.

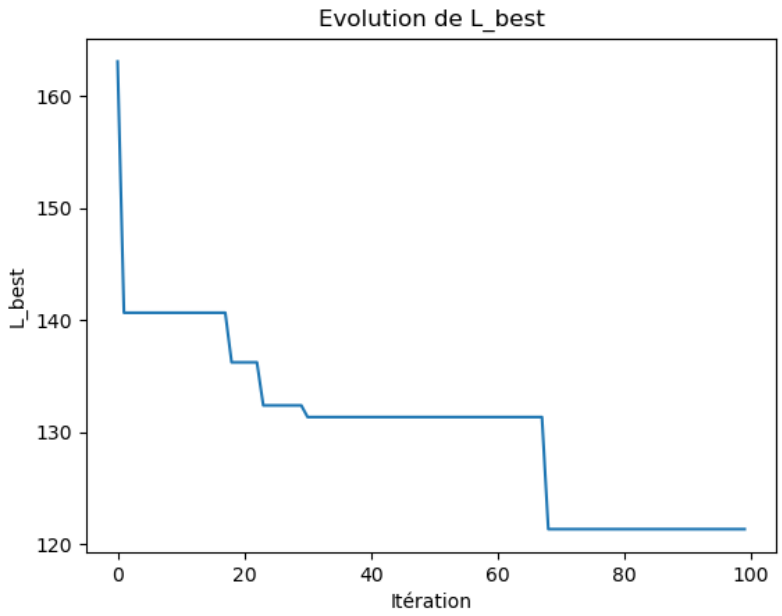

FIGURE 5 – Convergence de l'algorithme

Notre algorithme converge bien vers une solution. On pourrait cependant se demander s'il a fini de converger et si la solution trouvée est la plus optimale.

Si nous réalisons 30 simulations pour une même situation (on reprendra la même situation explicitée dans le paragraphe précédent) et que nous traçons l'histogramme des L_{best} trouvés, nous obtenons ceci :

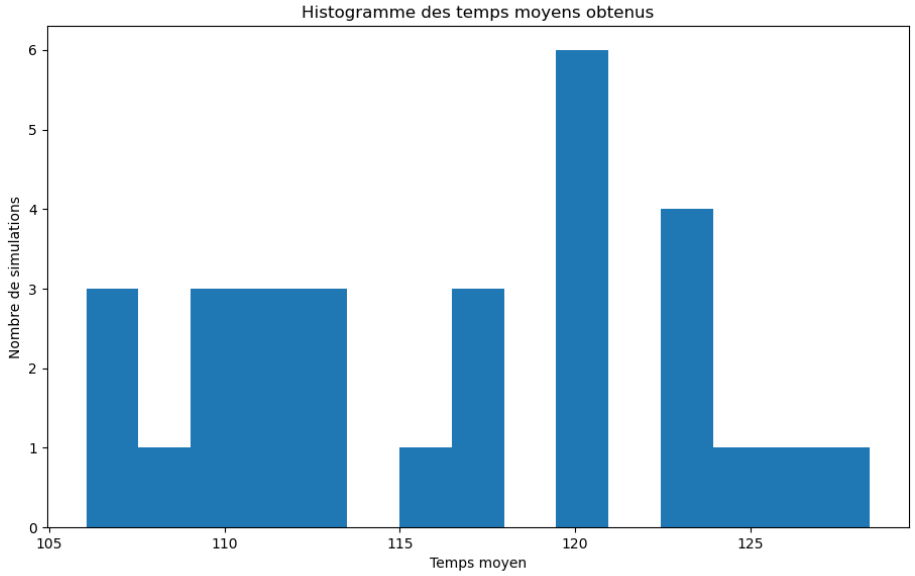

FIGURE 6 – Histogramme des L_{best} pour 30 simulations d'une même situation

Nous obtenons un histogramme avec des scores L_{best} qui ne sont pas tous identiques. Cela est dû au fait que nous ayons un paramètre random dans notre algorithme (nous n'avons pas imposé de graine de génération) ce qui implique différents choix en fonctions des valeurs obtenues. De plus, le problème de bus peut avoir plusieurs minimums locaux, ainsi notre programme peu rester bloqué dans l'un de ces minimums (pour en sortir, il faudra attendre un certain temps (qui sera de plus en plus long, car les phéromones vont favoriser la meilleure solution trouvée).

Intéressons-nous un peu plus à la convergence sur des temps plus long, ce graphique montre 3 itérations avec la même situation pour une simulation de paramètre $t_{max} = 1000$

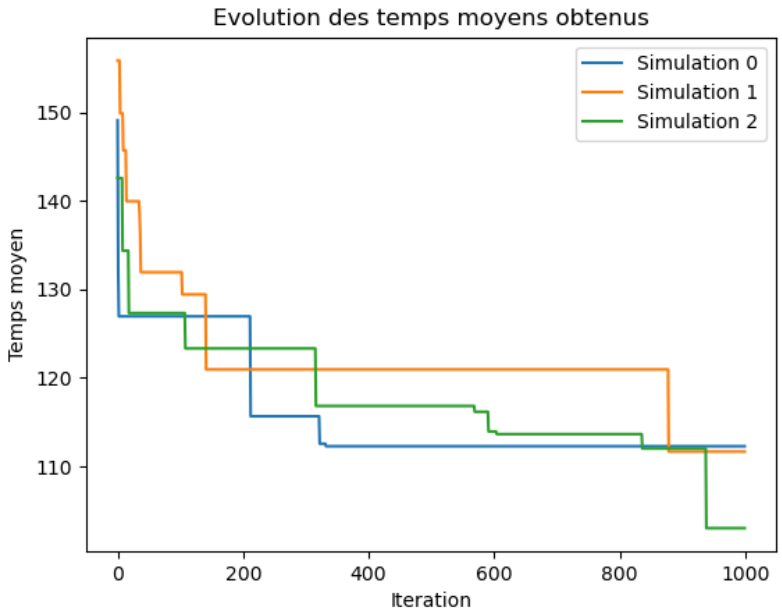

FIGURE 7 – Courbes de convergence des L_{best} avec $t_{max} = 1000$

On remarque que malgré un temps long, nos modèles continus d'être optimisé. Nous n'avons donc pas d'indicateur actuellement qui nous permet de dire que nous avons trouvé la meilleure solution. La seule façon est de laisser plusieurs modèles d'entrainer pendant très longtemps pour s'en approcher. Passons maintenant à l'étude des paramètres de notre modèle dont l'objectif est d'optimiser cette convergence !

4.2 Étude des paramètres de l'algorithme

Dans cette partie, nous allons nous intéresser à l'importance des paramètres de notre modèle et ainsi que leurs impacts sur sa performance et sa vitesse de convergence. Pour cela, nous allons reprendre la même situation et générer des simulations en ne faisant varier qu'un seul paramètre pour étudier son effet sur les résultats. Les paramètres β, ρ, α et q_0 seront étudiés, car ils impactent directement les choix réalisés pas l'algorithme.

Afin de rentrer les graphes plus visibles, nous établirons le code couleur suivant : les courbes seront de plus en plus foncées lorsque la valeur du paramètre étudié augmentera.

4.2.1 Étude du paramètre β : visibilité des arrêts de bus vs phéromones

Le coefficient β intervient en puissance de η qui est le coefficient de visibilité (plus un arrêt de bus est loin de l'arrêt actuel, plus la visibilité sera réduite). Ainsi, si $\beta = 0$, on ignore la visibilité dans notre simulation en ne prenant en compte que les phéromones. Par contre, lorsque β tend vers $+\infty$, nous ne prenons en compte que la visibilité.

Voici l'évolution de L_{best} pour différentes valeurs de β :

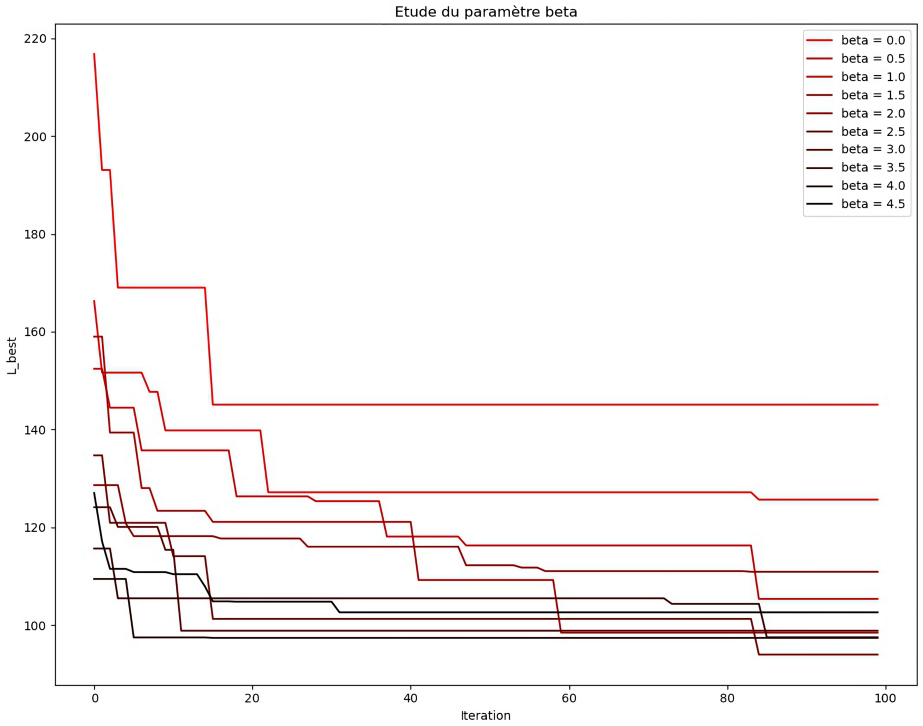

FIGURE 8 – Série de tests n°1

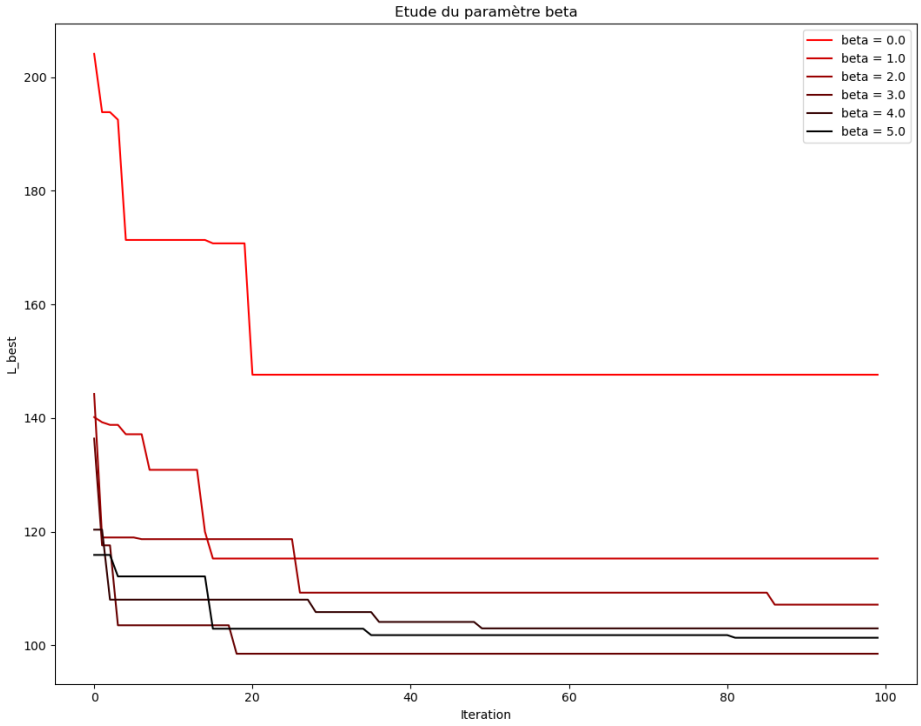

FIGURE 9 – Série de tests n°2

Nous remarquons donc que lorsque nous utilisons uniquement les phéromones, notre algorithme converge très lentement et donne des résultats assez mauvais. Dès lors que nous mettons un peu de visibilité, nous obtenons radicalement de meilleures performances. À partir de $\beta = 3$, nous en voyons plus trop les différences. Il ne faut pas oublier que le caractère stochastique de notre algorithme ne nous permet pas de fixer un β limite pour avoir la certitude d'avoir de bonnes performances.

Nous avons également remarqué qu'au-delà de $\beta = 6$, le programme met beaucoup de temps à converger. Cela s'explique par la gestion de l'arrêt principale lorsqu'il est l'arrêt le plus proche de la fourmi alors que celle-ci est déjà passée par là. Les proba favorisent le choix de cet arrêt, mais comme la fourmi doit passer par un arrêt qu'elle n'a pas déjà vu, on réitère le tirage jusqu'à trouver une solution. Il faudrait donc modifier l'algorithme en implémentant une nouvelle gestion de l'arrêt principale pour que la fourmi ne puisse plus le choisir

lorsqu'elle y est déjà passé, même si le reste de sa colonie ne l'a pas encore vue (ce qui implique modifier la distribution de probabilité).

Ainsi, pour avoir de bonnes performances rapidement, nous vous conseillons de mettre le paramètre β dans une gamme de valeur entre 1 et 5 selon votre projet, la taille de votre réseau de bus ainsi que les contraintes temporelles.

4.2.2 Étude du paramètre ρ : taux d'évaporation des phéromones

ρ correspond au tau d'évaporation des phéromones (entre 0 et 1) d'une même colonie lors de la progression de la colonie. Ainsi, plus le tau d'évaporation est élevé, plus on va converger rapidement vers une solution lors de la progression des fourmis et plus généralement lors de la progression de l'algorithme (celle-ci ne sera pas forcément la plus optimale).

Voici l'évolution de L_{best} pour différentes valeurs de ρ :

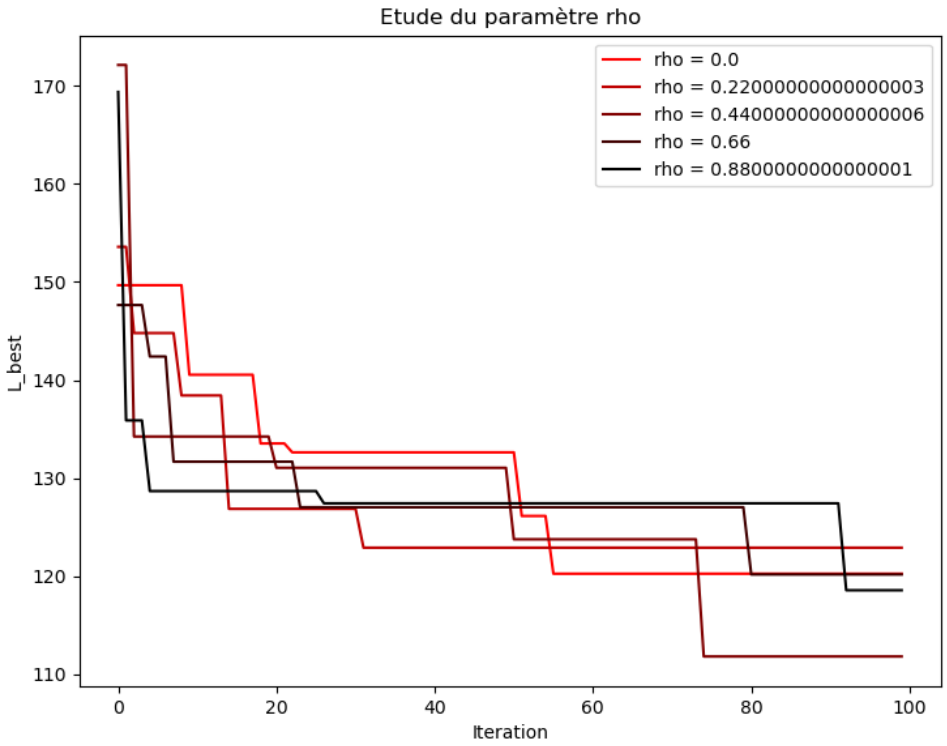

FIGURE 10 – Courbes de convergence des L_{best} en faisant varier ρ

Ces courbes confirment notre étude, les courbes les plus foncées converge plus rapidement vers un plateau alors que les courbes décroissent plus lentement, cependant elles ont moins de chance de tomber dans des minimums locaux. Nous recommandons l'utilisation d'un tau vers 0.5 qui semble être un bon compromis.

4.2.3 Étude du paramètre α : tau de progression

Le tau de progression met à jour les phéromones lorsque les colonies ont fini de progresser. L'équation faisant intervenir α valorise la meilleure solution en lui ajoutant aux trajets parcourus des phéromones. On s'attend donc à avoir une convergence plus rapide du L_{best} pour des taux de progression plus important.

Voici l'évolution de L_{best} pour différentes valeurs de α (on a réduit $t_{max=}$ à 50 car l'impact du tau de progression est nettement plus visible au début de l'algorithme) :

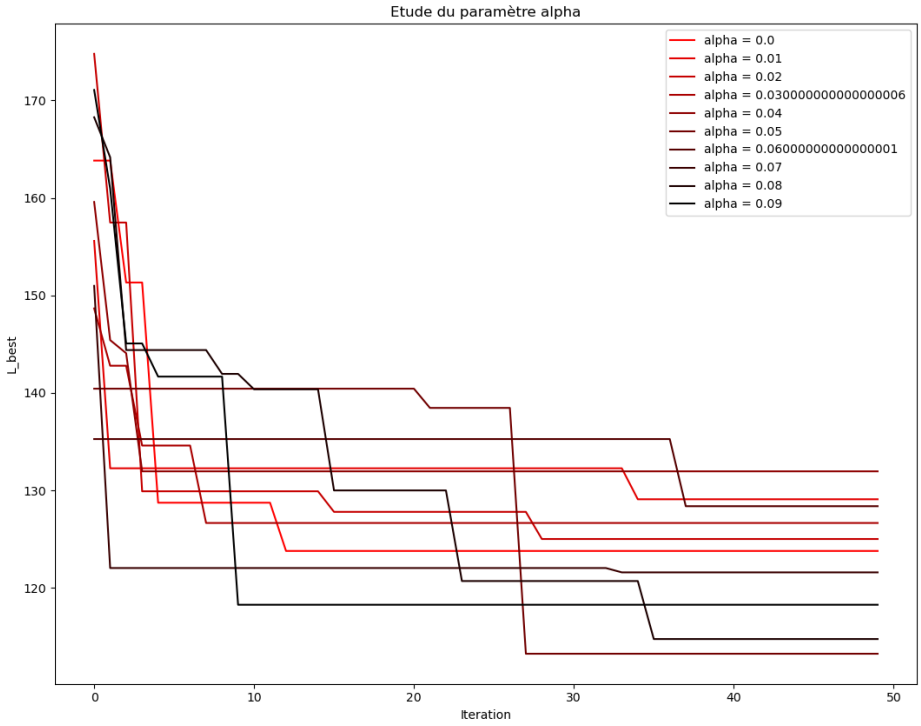

FIGURE 11 – Courbes de convergence des L_{best} en faisant varier α

Ces courbes confirment ce que nous avions prévu. Il faut cependant faire attention à ne pas mettre un tau de progression de 1, car cela entrainerait des taux de phéromones nuls ce qui provoque des erreurs dans le calcul des probabilités.

Tout comme le paramètre ρ, on veillera à ne pas mettre un tau de progression trop important, car on risquerait de s'enfermer dans un minimum local avec des probabilités assez faible pour en sortir (ce qui se traduit par des grands plateaux sur nos courbes de convergence).

4.2.4 Étude du paramètre q_0 : exploration vs exploitation

Le paramètre q_0 est utilisé lors du choix de la méthode progression utilisé par la fourmi. Celle-ci peut être en mode exploitation ($q_0 = 1$) dans lequel elle va prendre un chemin qui semble être le meilleur compromis entre distance et tau de phéromone (voir l'équation (1) p.4 dans la référence **[1]**. L'autre mode est l'exploration biaisée ($q_0 = 0$) dans lequel la fourmi va choisir son prochain

arrêt suivant une distribution de probabilité dans lequel la distance et le tau de phéromone influe.

Voici l'évolution de L_{best} pour différentes valeurs de q_0 :

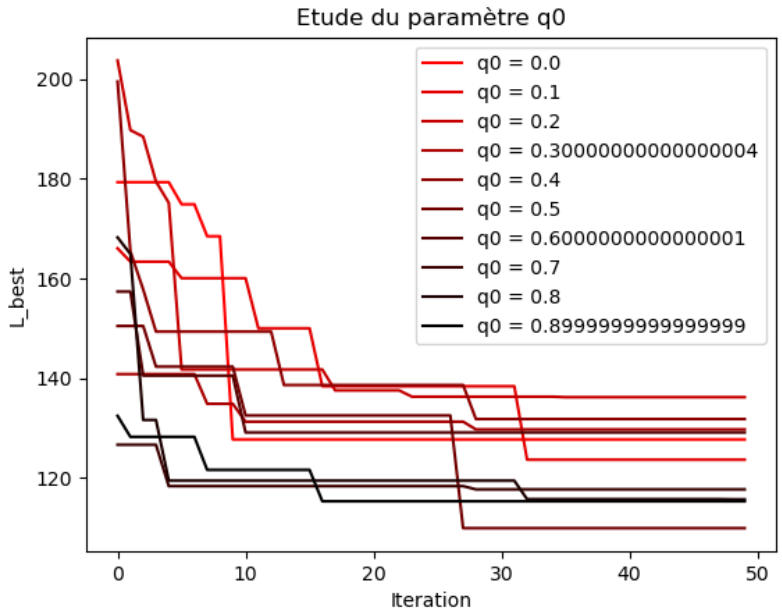

FIGURE 12 – Courbes de convergence des L_{best} en faisant varier q_0

La première remarque est que le mode d'exploitation ne fonctionne pas lorsqu'il est tout seul. En effet, le choix du prochain arrêt ainsi que la gestion présentée dans l'article de l'arrêt de bus principale implique que celui-ci doit être dans la liste des arrêts de bus non visité par les K-fourmis s'il n'a pas été visité par tous les K-fourmis. Or si une K-fourmi l'a déjà visité et que celui-ci reste l'arrêt de bus le plus prometteur (prise en compte de la visibilité et des phéromones) alors, on entre dans un blocage de l'algorithme qui tournera en boucle dans le vide.

Néanmoins, nous pouvons voir que plus le paramètre q_0 augmente (tends vers l'exploitation) meilleur sont les résultats. Cependant, on remarque si on se contente de ce mode de sélection, ils stagnent assez vite et ne bénéficieront pas de l'exploration pour sortir de leur minimum local. Il faut donc trouver un coefficient pour bénéficier de la convergence rapide de l'exploitation puis de l'exploration.

5 Conclusion

Finalement, nous avons réussi à proposer une résolution du problème "BUS" à partir d'un algorithme génétique. Nous avons également pu l'étudier en profondeur afin d'analyser l'impact de chacun des paramètres et donc de trouver des compromis pour avoir une solution acceptable le plus rapidement possible.

Nous pouvons également apporter des idées d'amélioration de l'algorithme. L'une d'elle consiste à déposer des phéromones uniquement sur les meilleures solutions et non au fur et à mesure comme c'est le cas actuellement.

Il faudrait également revoir la gestion des arrêts de bus visités par les fourmis en mettant non pas seulement une liste commune aux fourmis, mais peut-être en ajoutant un indicateur associé à chaque fourmi permettant de savoir si celle-ci est déjà passé par l'arrêt de bus principal pour ne pas le prendre en compte dans les calculs de cette fourmi.

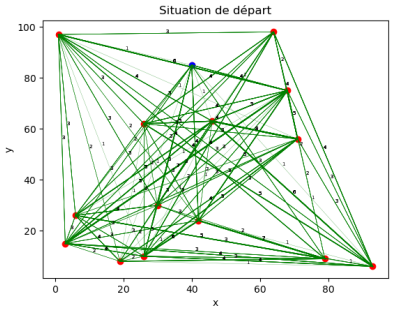

FIGURE 13 – Exemple de situation générée

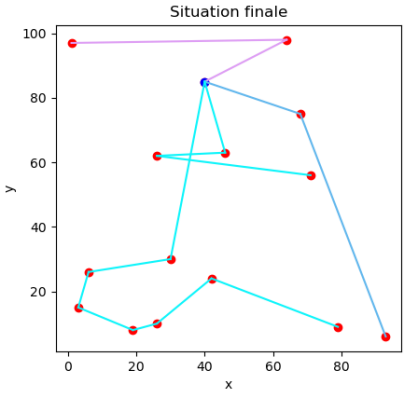

FIGURE 14 – Exemple de solution générée graphiquement par notre algorithme

6 Bibliographie

[1] : de Jong, J. H., & Wiering, M. A. (2001). *Multiple ant colony systems for the busstop allocation problem*. In Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation (pp. 717-724). ACM.