2020

Rapport de Projet IA54



Groupe : JUSTIN GANIVET - LOÏC MARTIN - YOSEF BAYOUDE

Table des matières

[Introduction 1](#_Toc60601471)

[Présentation du problème 1](#_Toc60601472)

[Representation du probleme 1](#_Toc60601473)

[méthode de REsolution 2](#_Toc60601474)

[Algorithme 2](#_Toc60601475)

[premiere phase : Parcours des fourmis 2](#_Toc60601476)

[Deuxieme phase : Mise à jour du niveau de phéromone 3](#_Toc60601477)

[Réalisation de l’application 4](#_Toc60601478)

[Parseur 5](#_Toc60601479)

[Environnement 6](#_Toc60601480)

[Fourmi 8](#_Toc60601481)

[Paramètres partagés 9](#_Toc60601482)

[Interface graphique 10](#_Toc60601483)

[Résultat 11](#_Toc60601484)

[Difficultés rencontrées 12](#_Toc60601485)

[Conclusion 12](#_Toc60601486)

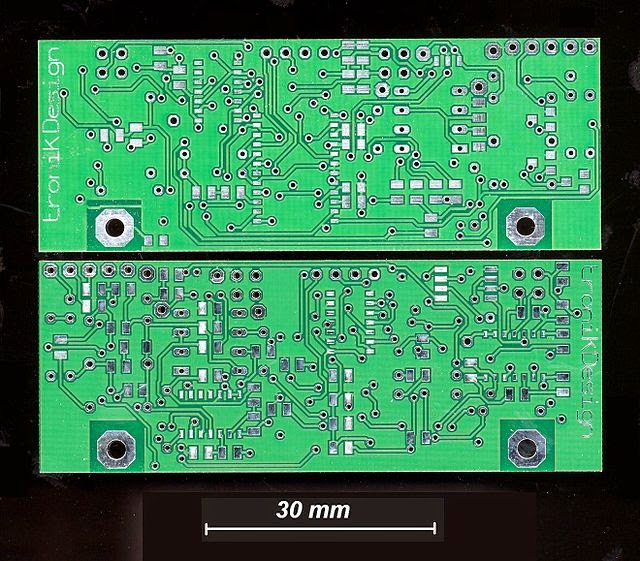
# Introduction

Dans le cadre de notre UV IA54, nous avons dû réaliser, en équipe, une application de simulation multi-agent, utilisant le langage SARL. Pour ce projet nous avons choisi le sujet « Module ACO (Ant Colony optimisation) appliqué au TSP (Traveling Salesaman Problem) ». Le but de cette application est de résoudre le TSP en utilisant des fourmis représentées par des agents, et afficher le chemin le plus court trouvé par cet algorithme.

L’objectif de cet exercice était d’appliquer l’ensemble des notions étudiées en cours telles que le langage SARL, et utilisation d’agents pour résoudre des problèmes.

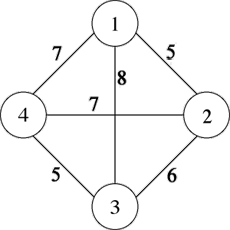
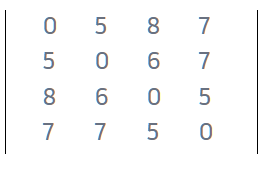
# Présentation du problème

Le TSP ou problème du voyageur de commerce, est un problème d’optimisation, avec l’énoncé suivant : « Un voyageur de commerce doit visiter **toutes** les villes **exactement** une fois, et revenir au point de départ, trouver l’ordre de visites des villes qui minimise la distance totale parcouru. » Ce problème appartient la classe des problèmes de complexité NP-complet. Ce problème sert souvent comme référence pour évaluer la performance d’un algorithme et il est présent dans beaucoup de domaines, comme la transportation de marchandise, ou l’ordre de perçage de trou dans un circuit imprimé par une machine.



## Representation du probleme

En informatique, ce problème peut être modélisé à l’aide d’un graphe, constitué de sommets et d’arêtes. Le graphe est complet, signifiant que tous les sommets sont reliés entre eux. Ces liaisons sont définies par un poids, dans ce cas, le poids est donné en fonction de la distance physique entre deux sommets, appelé . Le graphe est aussi dit « asymétrique », c’est-à-dire que le poids d’un sommet vers un sommet est égale au poids de vers . Pour tout graphe, une matrice de poids peut être établie, comme on peut voir dans figure ci-dessous.

## méthode de REsolution

Le but du sujet est de résoudre ce problème en utilisant un système multi-agents, en utilisant le module ACO. Ce dernier est un algorithme qui consiste à répliquer le comportement des fourmis dans la nature.

Des fourmis, modélisée chacun par un agent, sont créées et le libéré dans l’environnement, ou le graphe. Ces fourmis, complétement indépendantes, devront traverser le graphe entier en visitant un sommet après l’autre. Elles ne visitent pas sommet plus que une fois, et doivent finir au point d’origine. Lorsque ces fourmis traversent une arête, elles déposent des phéromones, et choisissent leur prochaine destination en fonction du poids donnée à une arête et la quantité de phéromone présente. Plus le poids est petit et la quantité de phéromone est grande, plus il a des chances de prendre ce dernier. Le niveau de phéromone s’évapore au fil du temps, en fonction du chemin le plus désiré (le plus court) traversé par les fourmis. Au fur et à mesure que ces fourmis refont leur passage, la quantité de phéromone dans les arêtes désirées augmentent tandis que celles qui ne sont pas voulu s’évaporeront.

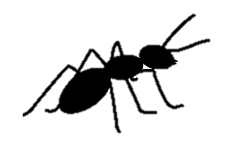
# Algorithme

Rentrons plus dans le détail, dans cette partie nous expliquons l’algorithme, ainsi que ses fonctions mathématiques. Son fonctionnement se résume à deux étapes principales ; le parcours du graphe des fourmis et la mise à jour des phéromones. Ces deux étapes se répètent en boucle autant de fois que nécessaire, plus le nombre de boucle est grand, et plus les fourmis ont la possibilité d’explorer de nouvelles routes et ainsi calculer un chemin plus court, cela déterminera donc la précision du résultat.

## premiere phase : Parcours des fourmis

Tout d’abord on crée un nombre de fourmis , égale au nombre de sommets présents dans le graphe. Cette première phase consiste à faire parcourir le graphe par les fourmis une seule fois, en visitant un sommet exactement une fois et revenir au point d’origine. Lorsqu’une fourmi se déplace d’un sommet à un autre, elle se retrouve face un choix d’arêtes possibles qu’elle peut emprunter.

**?**



La décision de ce choix est probabiliste, cela signifie qu’on ne sait pas quelle décision une fourmi va prendre, mais on connait sa probabilité. Cette probabilité est définie avec la fonction suivante :

Avec :

* : L’intensité de phéromone entre un sommet et
* : La visibilité entre un sommet et , il est égal à , cela signifie que plus une arête est courte, plus cette valeur grande
* : Les sommets lié à par une arête
* et  : Des paramètres qui fixent l’influence de l’intensité de phéromone et la valeur heuristique.

Grace à cette formule, la probabilité qu’une fourmi prenne une arête augmente avec l’intensité de phéromones, tout en gardant en compte la taille de l’arête.

Il faut noter que lors du tout premier passage, il n’y a aucune trace de phéromone ( = 0) dans l’environnement, cela signifie que par défaut les fourmis prendront les sommets les plus prêts l’un après l’autres, et par conséquent n’exploreront pas plus de chemins. Pour éviter cela, lors du premier passage, on remplace par , avec le nombre de fourmis, et étant la longueur du tour généré en utilisant un algorithme « voisin-le-plus-proche » simple.

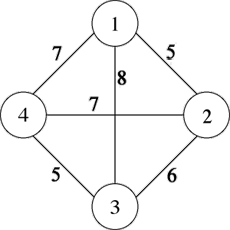
## Deuxieme phase : Mise à jour du niveau de phéromone

La deuxième phase, consiste à mettre à jour le niveau de phéromone déposé dans le graphe, ceci aurait pu être fait en même temps que le parcours des fourmis, mais pour des raisons de performance, il se fait à après chaque parcours de graphe par toutes les fourmis.

En premier lieu, on diminue le niveau de phéromone dans **toutes** les arêtes par une constante. Le calcul est le suivant :

Avec , le taux d’évaporation de phéromone, ayant une valeur entre 0 et 1. Cette première démarche évite que le niveau de phéromone s’accumule à l’infini, et permet aux fourmis « d’oublier » les arêtes indésirables.

Finalement, on ajoute des phéromones aux arêtes visités pas toutes les fourmis durant le dernier tour de façon suivante :

Avec , la somme de toutes les arêtes du tour effectué par une fourmi . Cette formule donne ainsi plus d’importance aux arêtes qui ont permis aux fourmis de faire les tours les plus courts. 

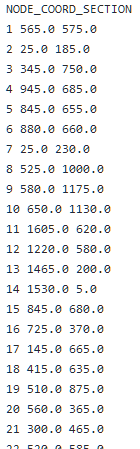
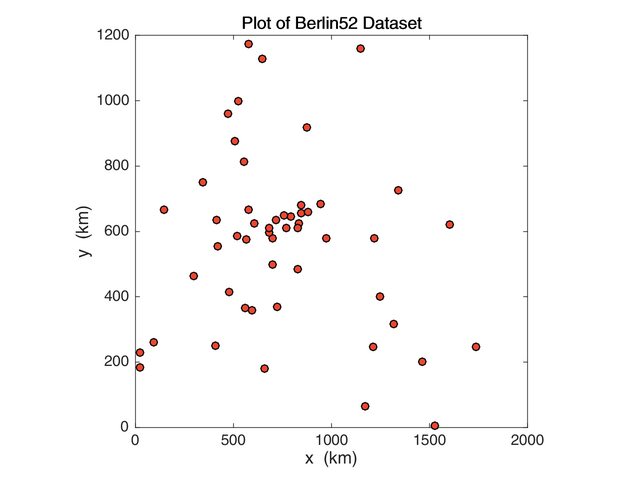
# Réalisation de l’application

Notre application a été développé en SARL, un langage de programmation orienté agent, ainsi que du Java et AWT pour l’interface graphique. Elle peut être découpé en plusieurs éléments :

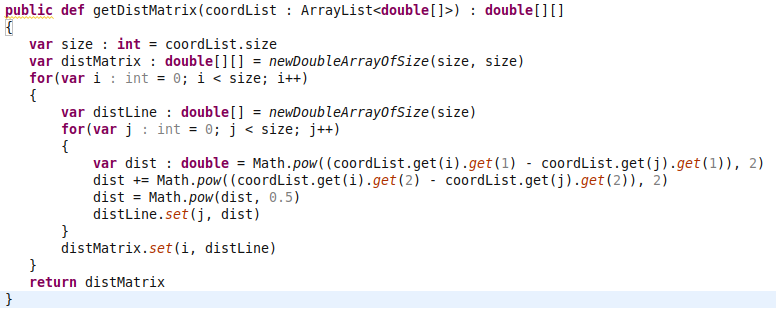
* Le parseur
* Environnement
* Fourmis
* Paramètres partagés
* Interface graphique

## Parseur

Afin de pouvoir exécuter cet algorithme sur un ensemble des graphes variés, nous avons créé un « parseur ». Ce parseur nous permet d’interpréter un fichier de type *.txt*, donné en argument au lancement de l’application, vers un graphe que notre application pourra comprendre. Ce parseur sera donc le point d’entrer de notre application.

 🡪 

Le fichier contient un nombre de sommets, suivi par leurs coordonnées, son format issu de la librairie « TSPLIB ». Ces informations sont alors converties vers une liste de points, suite auxquelles on calcule le poids des arêtes entres chaque sommet. Elles sont calculées en fonction de leurs distance physique à l’aide de la méthode suivante :

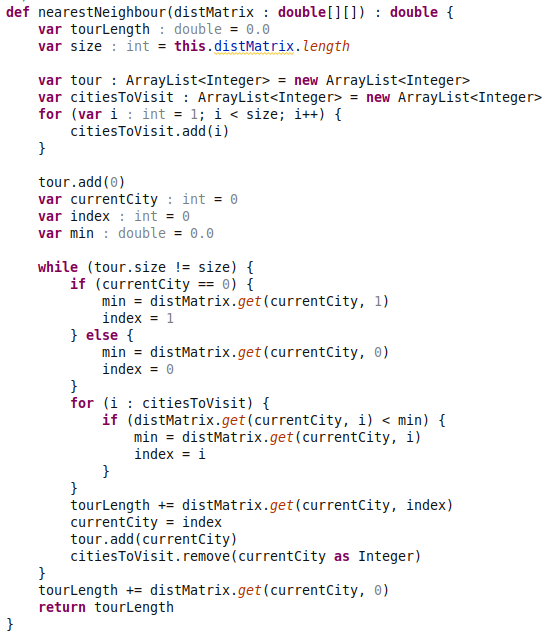


Avec cela notre application dispose du graphe complet.

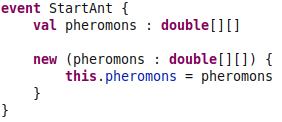
## Environnement

L’environnement est modélisé par un agent. Son rôle est de représenter le graphe, ainsi que ses propriétés, comme le niveau de phéromones et les résultats trouvés. De plus, cet agent se charge de plusieurs taches pour assurer le bon fonctionnement du programme.

Premièrement, l’agent environnement contient une méthode permettant de calculer , en effet ce dernier sera nécessaire pour attribuer la valeur du premier tour. Cette étape se trouve en dehors de la boucle principale.

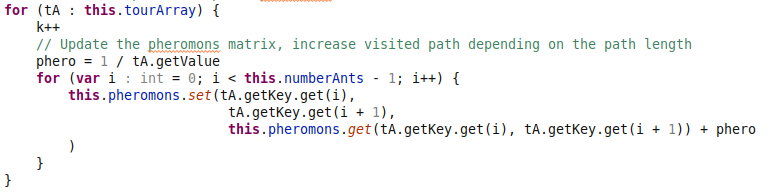


Pour commencer la boucle, l’environnement envoi le signal de départ aux les fourmis, grâce à l’event *StartAnt*. Cet event contient les niveaux de phéromones associé à chaque arête afin que les fourmis puissent prendre leurs décisions.

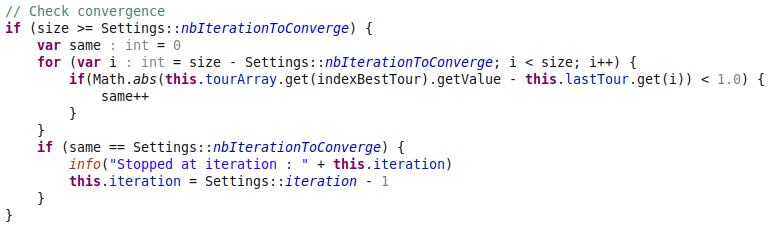


Une fois que toutes les fourmis ont finit leur tour, elles renvoient leur résultat avec un l’event *TourFound* contentant l’ordre des sommets visités et la taille du tour*.* Durant cette étape l’environnement attend une réponse de toutes les fourmis. Avec les réponses de toutes les fourmis à sa disposition, l’environnement peut commencer une de ses taches principales ; la mise à jour des phéromones.

Ce travail est accompli en utilisant la procédure et les formules mathématique vus précédemment, l’environnement commence par réduire l’intensité des phéromones dans toutes les arêtes et augmente celles traversés par les fourmis.



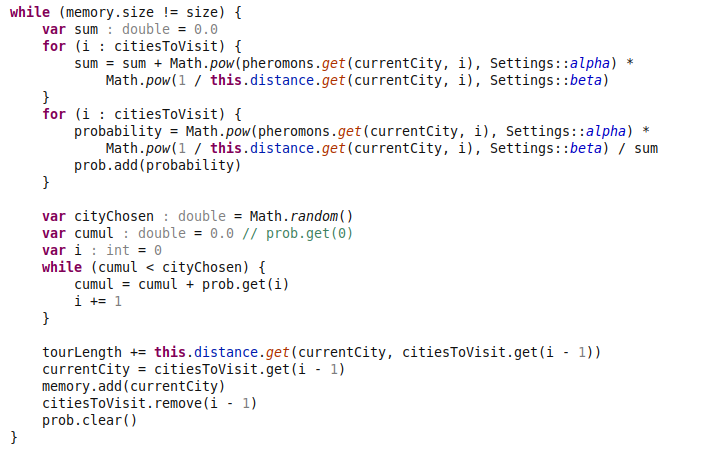
Finalement, l’agent vérifie après chaque tour si les résultats convergent, le nombre de résultat qui doivent converger avant qu’il s’arrête est défini dans le fichier des paramètres partagés, et donc modifié facilement si besoin. Au contraire, si le résultat ne converge pas, l’agent fait à nouveau appel aux fourmis pour ainsi répéter la boucle.



## Fourmi

Les fourmis sont également représentées par des agents. Leur rôle est très basique, puisqu’elles doivent seulement visiter tous les sommets exactement une fois et revenir au point d’origine.

Pour y parvenir elles disposent d’une mémoire avec tous les sommets déjà visités, ainsi que tous les sommets à visiter. Pour chaque arête qu’elles peuvent emprunter, sa probabilité est calculée et finalement les fourmis prennent leur décision en fonction du résultat.



Etant donné les fourmis sont des agents, elles sont complètement indépendantes et s’exécutent en parallèle, de plus dans notre application elles ne communiquent pas entre elles. Finalement quand tous les sommets ont été visités, elles renvoient leur chemin à l’environnement, pour pouvoir effectuer le calcul des phéromones.

## Paramètres partagés

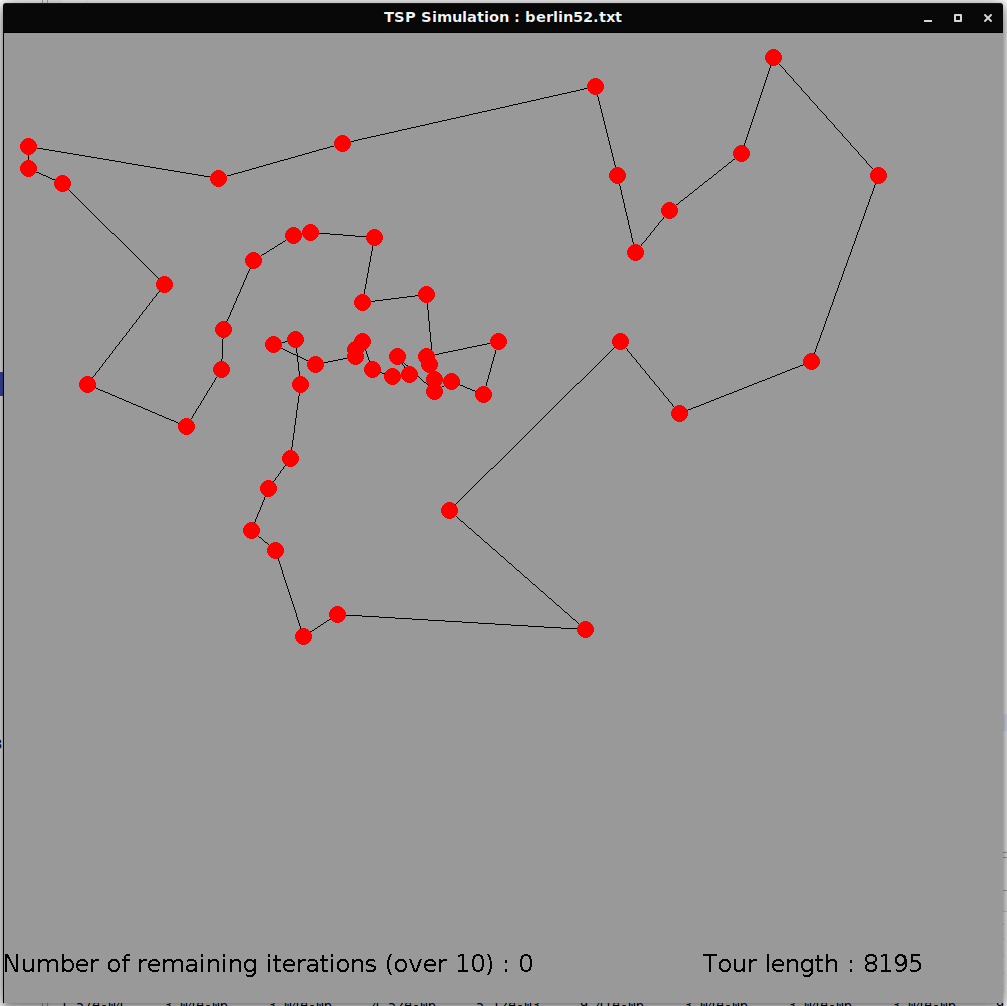
Afin de fixer les valeurs nécessaires, tout en les rendant facilement modifiable si souhaité, nous avons décidé de mettre ces variables dans un fichier commun nommé « Settings ». De cette manière, nous pouvons changer les valeurs rapidement pour expérimenter, ou bien privilégier la performance / précision.



C’est dans ce fichier aussi que nous paramétrons les conditions du résultat, en effet, nous pouvons y indiquer le nombre d’itération que nous voulons, ainsi que le nombre de résultat qui convergent vers une valeur avant que le programme s’arrête. Ceci nous permet par exemple de pousser l’algorithme à chercher plus de chemins même si cela prend plus d’itération, ou bien d’avoir un grand nombre d’itérations qui convergent pour valider le résultat.

## Interface graphique

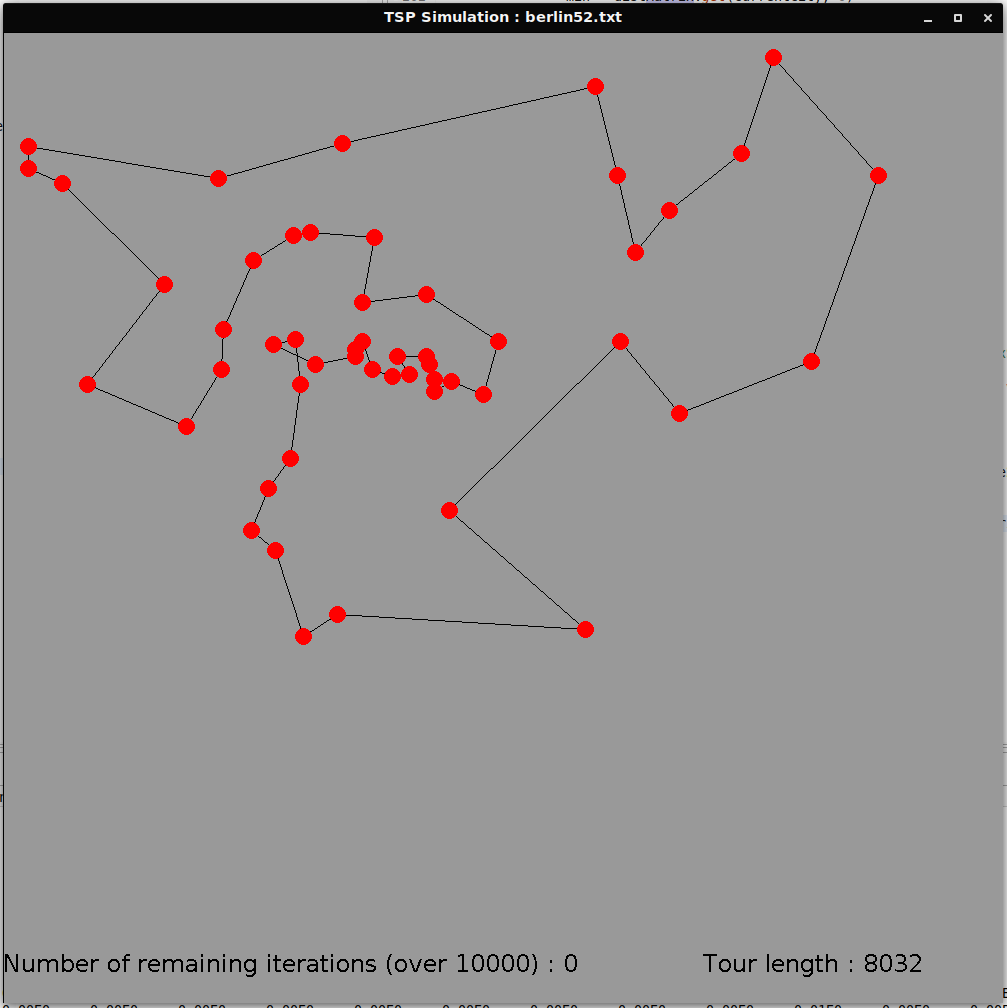
Notre application contient aussi une interface graphique basique, créé en utilisant la librairie AWT, qui permet de visualiser les chemins emprunter par les fourmis en temps réel.



On peut voir affiché tous les sommets, le nombre de boucle restant, ainsi que la longueur du tour trouvé.

# Résultat

Le résultat de notre travail est un programme fonctionnel qui utilise l’algorithme ACO en SARL pour le TSP avec une interface graphique du résultat. Pour le lancer, il suffit de mettre en argument le fichier .txt de la structure du graphe et le programme se démarre tout seul.



Quant aux solutions obtenues, le programme nous donné des résultats positifs, dans plusieurs cas nous étions à moins de 10% du résultat optimal avec de bonnes performance, puisque le résultat convergeait en quelques secondes. Nous avons remarqué que lorsqu’on modifiait les paramètres dans le fichier « settings », il nous était possible d’atteindre de meilleurs résultats avec peu d’effet sur la performance. Etant donné que nous avons à faire à des probabilités, il est aussi important de noter qu’il n’est pas possible de reproduire le même résultat à chaque coup, puisqu’il y a un facteur de chance dans la recherche de la meilleure solution.

# Difficultés rencontrées

L’utilisation de SARL était parfois compliquée, en effet les ressource sur internet sont plutôt limités, cela signifie que quand on se trouver face à un problème nous devions souvent le résoudre par nos propres moyens, par exemple, en réutilisant du code fourni dans les TP.

# Conclusion

Pour conclure, la réalisation de ce projet s’est bien déroulée. Ce dernier nous a permis de mettre en application une nouvelle approche de résoudre les problèmes, notamment la programmation orienté agent. Nous avons vu des exemples en TP et nous avons pu mettre en application nos connaissances. Nous pouvons à présent les appréhender plus facilement.

Nous avons également pu étudier un nouveau langage, SARL et nous nous sommes aussi rendu compte du potentiel de ce dernier.