# Rapport de Projet

par ANGELINA Sacha

Université de Lorraine - 2018/2019

# I. Introduction

Le projet porte sur l'implémentation d'un algorithme métaheuristique pour l'optimisation d'un problème classique nommé le problème du Voyageur de Commerce. Une contrainte de temps nous est imposée pour déterminer une solution satisfaisante.

## A) Problème du Voyageur de Commerce

Le problème du Voyageur de Commerce (ou **TSP** - *Travelling Salesman Problem* - en anglais) se formule ainsi : une personne doit effectuer une tournée dans plusieurs villes en minimisant la distance totale parcourue. Pour une liste de villes donnée, la personne doit passer une seule et unique fois par chaque ville et revenir au point de départ. Il s'agit d'un problème d'optimisation classique classé dans les problèmes NP-Complet.

## B) Algorithmes Métaheuristiques

Malgré que les noms se ressemble, il faut différencier un algorithme heuristique - qui utilise une heuristique donc mais qui est spécifique à un problème donnée pour fournir une solution optimal approché - d'un algorithme métaheuristique qui est généralement une méthode stochastique et itérative adaptable à plusieurs problème.

## C) Les données

Les données sont fournis dans des fichiers au format `.tsp.` Chaque ligne donne les coordonnées cartésiennes d'une ville. Les fichiers fournies proviennent de l'université de Heidelberg et sont :

- a280, qui compte 280 villes et a une solution optimale de 2579 ;
- bier127, qui compte 130 villes pour une solution optimale à 118282 ;
- ch130, ayant une solution optimale à 6110 et comporte 130 villes.

Puisque les solutions optimales n'ont été trouvé que tardivement lors du projet, j'ai également créée d'autres fichiers dont je connaissais déjà la solution optimale. Le fichier utilisé possède 600 villes disposé de sorte à former un carré de 600 de périmètre.

# II. Les Algorithmes Étudiés

Dans le cadre de ce projet, plusieurs métaheuristique à population ont été envisagé et mis en place dans le cadre du projet : La colonie de Fourmi, l'algorithme évolutionnaire, et une variante spécifique de cette dernière : l'algorithme évolutionnaire auto-adaptatif.

## A) La colonie de fourmis

Il s'agit d'une métaheuristique à population qui a été notamment mis en place pour la recherche du plus court chemin dans un graphe. Rendant donc l'algorithme tout à fait choisie pour le problème TSP. L'algorithme se base comme son nom l'indique sur les colonies de fourmis.

Le principe est simple, chaque individu représente une fourmi : la fourmi va faire un chemin de manière semi-aléatoire et déposer une quantité de phéromone proportionnel à la qualité de la solution qu'elle propose. Pour la détermination d'un chemin, elle va préférer les portions possédant plus de phéromone mais ne va pas prendre forcément celles avec le plus de phéromones.

Comme l'algorithme a été spécifiquement mis en place pour la recherche du plus court chemin, elle a été envisagé dans un premier temps. Cependant, l'implémentation qui en a résulté ne fournissait pas de solution satisfaisante par rapport aux autres algorithmes choisies et présenté des lacunes qui malgré la limite de temps, continuer à tourner.

## B) Algorithme Évolutionnaire

Il s'agit d'une métaheuristique qui se base sur la théorie de l'évolution de Darwin et donc se base sur les principes de l'hérédité et de la sélection.

Chaque individu est une solution potentielle, c'est-à-dire une liste donnant l'ordre de passage dans les villes. Ces individus seront évalué au vu d'être sélectionner pour ensuite pouvoir générer de nouveaux individus à partir des premiers, en utilisant des opérateur de croisement et de mutation, ces individus remplaceront les anciens. Ce processus de sélection, croisement, mutation se nomme itération ou *évolution*.

Pour ces algorithme, on définit en général soit un nombre d'évolution ou, comme ici, un temps limite.

Dans le cadre de ce projet, un individu est représenté par la liste des villes à parcourir. Et l'initialisation de chaque individu se fait à partir de la fonction `getRandomPath()` fourni dans le projet.

### 1. Première Approche : "Classicisme"

La première approche a été de faire une implémentation "classique" de l'algorithme évolutionnaire.

On effectue une **sélection par tournoi** : pour x individus de la population choisis aléatoirement, on sélectionne celui ayant la meilleur évaluation.

L'**opérateur de croisement** était relativement simple. On sélectionne deux individus.

| | |
|---|---|
| **1-2-3**-4-5-**6** | **5**-4-2-**1-6**-3 |

Pour chacun des individus, on tire une portion de la solution.

| | |
|---|---|
| **2-3-4** | **5-4** |

On crée deux nouveaux individu avec ces portions, puis on ajoute les villes qui ne sont pas dans la solutions partielles dans l'ordre de l'autre parent.

| | |
|---|---|
| 2-3-4-**5-1-6** | 5-4-**1-2-3-6** |

Plusieurs **opérateurs de mutation** ont été testés. Un opérateur simple qui va faire **une permutation** dans la solution. Un qui va **recréer un nouveau chemin** à partir d'un indice tiré aléatoirement (par exemple, la solution est 1-2-3-4-5-6. l'indice 3 est choisi, le résultat de la mutation est 1-2-**6-3-5-4**). Cependant, ces opérateurs présentaient de moins bons résultats que le dernier que je vais introduire.

L'opérateur de mutation finalement retenu se présente ainsi, comme on peut le voir sur la Figure 1 : On sélectionne deux villes de la solution de manière stochastique *a* et *b*, *a* étant toujours avant *b* dans la solution. On note la ville qui suit respectivement *a'* et *b'*. On veille à ce que $a \neq b'$ et $b \neq a'$. Ensuite, on fait en sorte que *b* devienne la ville suivante de *a*, on inverse l'ordre des villes de *b* à *a'* et on termine par remettre dans l'ordre les villes restante à partir de *b'*.

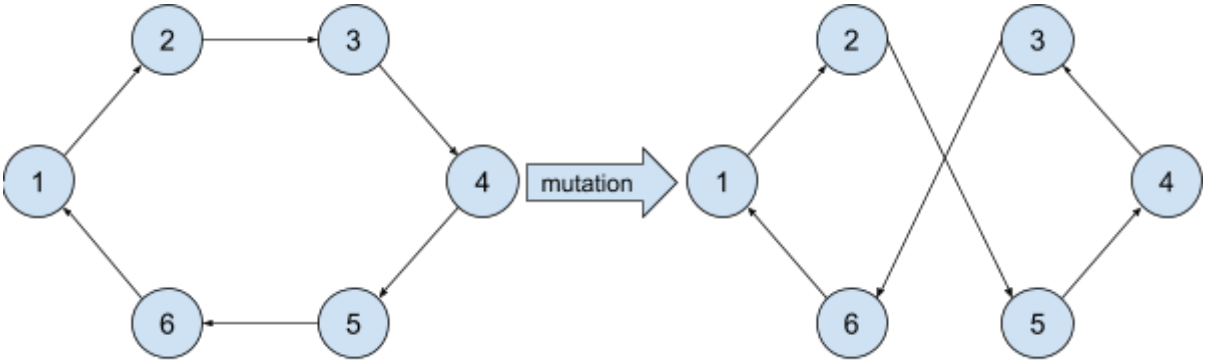



*Figure 1 : Mutation d'un individu*

Après des tests empiriques, les **taux des croisements et de mutations** sont fixés respectivement à 50% et 40%. Laissant ainsi 10% d'élitisme, c'est-à-dire, la copie simple des meilleurs individus. Des meilleurs résultats ont été remarqué pour une population avec 100 individus.

## 2. Deuxième Approche : Adaptativité

La deuxième approche reprend les mêmes opérateurs et le même taux d'élitisme que l'approche précédente. Mais à chaque itération, on modifie les taux de croisement de de mutation.

Dans un premier temps, je suis parti du principe qu'au départ nous avons besoin de faire plus d'exploration, ce qui induit un taux de mutation relativement fort. Mais au fil des évolutions, nous devons favoriser l'exploitation, ce qui induit une augmentation du taux de croisement. C'est ainsi qu'il a été décidé qu'à chaque

itération, on augmenterait le taux de croisement et baisserait celui de mutation. Pour cela, (soit *pc* le taux de croisement, *pe* celui d'élitisme) on fixe un coefficient $\lambda$ et on détermine un $\delta$ tel que :

$$\delta = ((1 - pe) - pc) * \lambda$$

Ensuite, on modifie les taux de croisement et de mutation, respectivement, en ajoutant et retirant $\delta$. Le taux étant initiée respectivement à 50% et 40%.

Dans un deuxième temps, voyant que les résultats n'étaient pas satisfaisant par rapport à la première approche, j'ai envisagé de fixer un seuil $\varepsilon$ de sorte à adapter $\delta$ selon $\Delta$ qui dépend du meilleur résultat de la dernière population et la population courante.
Soit *rp* le résultat de l'évaluation de la population précédente et *rc* le résultat de l'évaluation courante, alors :

$$\Delta = \frac{(rp - rc)}{rp}$$

Ainsi, si la valeur absolue de $\Delta$ est inférieur ou égal à $\varepsilon$, alors on estime que la population stagne et on réinitialise les taux à leur valeur donné au départ. Sinon, si $\Delta$ est inférieur à $-\varepsilon$, alors on estime que la population à fait pire qu'avant et on diminue le taux de croisement (et augmente celui de mutation), sinon $\Delta$ est donc supérieur à $\varepsilon$ et alors la population "est meilleur" et on augmente le taux de croisement (et diminue celui de mutation). Ce que l'on va ajouter ou retirer au taux est calculé de la façon suivante (Soit *pci* le taux de croisement initial) :

$$\delta = \begin{cases} (pci - pc) * \lambda, & \text{si } \Delta < -\epsilon \\ ((1 - pe) - pc) * \lambda, & \text{si } \Delta > \epsilon \end{cases}$$

On ajoute et retire respectivement au taux de croisement et au taux de mutation $\delta$, si la valeur absolue de $\Delta$ est supérieur à $\varepsilon$.

En parallèle de cette méthode, j'ai pensé qu'il serait peut-être plus logique de modifier les taux en fonction de l'écart de la solution. La méthode dépendrait aussi d'un $\varepsilon$. Ainsi, on aurait :

$$\delta = \begin{cases} (pci - pc) * \Delta, & \text{si } \Delta < -\epsilon \\ ((1 - pe) - pc) * \Delta, & \text{si } \Delta > \epsilon \end{cases}$$

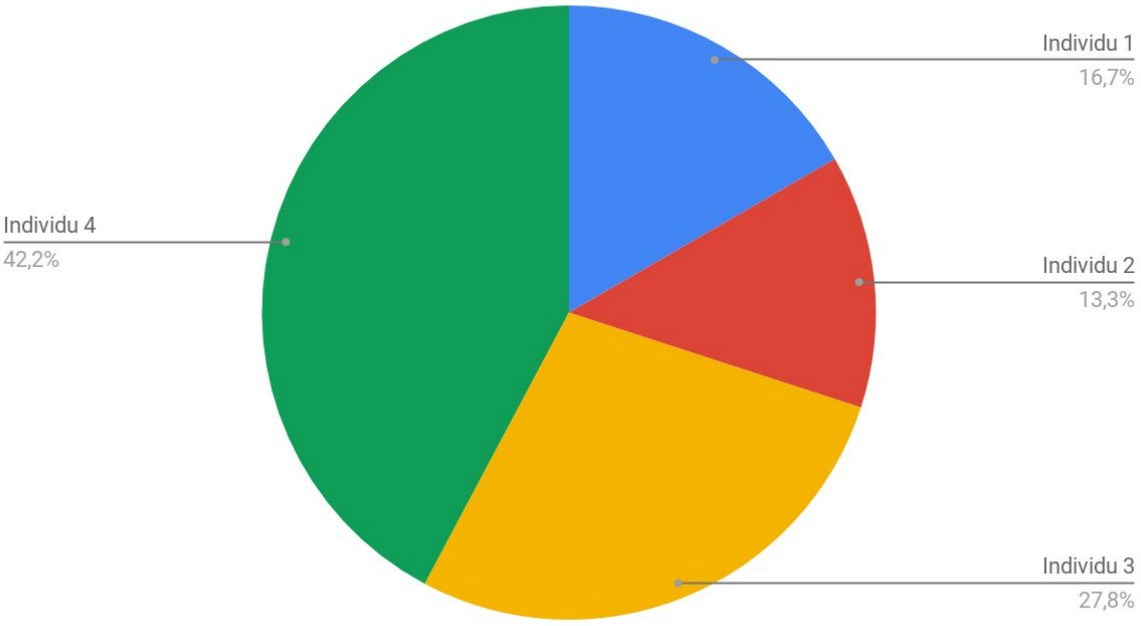
En définitive, d'après les tests empiriques entrepris pour déterminer les deux dernières méthode de modification des taux, on est arrivé à la conclusion que les deux approches fournissent des résultats sensiblement équivalents, avec cependant un infime avantage pour la méthode avec un seuil et utilisant le $\lambda$. Ainsi, plus tard il seront confrontés aux deux autre approche d'algorithmes évolutionnaires.

## 3. Troisième approche : Auto-adaptativité

La Troisième approche consiste rendre la population auto-adaptative. Ainsi, les taux de mutation et de croisement font parti des gènes d'un individu. Cette approche a été inspiré par [1].

Pour effectuer des croisement, on utilise la méthode de **sélection par roulette** (ou Roulette Wheel Selection - RWS, en anglais). C'est à dire qu'on attribue une portion de disque à chaque individu de la population (voir Figure 2), puis on fait tourner la roulette, l'individu sélectionné est celui qui tombe sur le point fixe.

Figure 2 : Exemple de RWS




Pour déterminer la portion qu'un individu va prendre, j'ai décidé que cela dépendra de son évaluation par rapport à la somme des évaluations de la population, multiplié par son taux de croisement :
Soit $p_k$ la portion de l'individu $k$, $s_k$ l'évaluation de l'*k-ième* individu, et $tc_k$ le taux de croisement :

$$p_k = (\sum_{i=1}^{N} s_i - s_k) * tc_k$$

Le résultat du croisement, à la possibilité de muter défini par son taux de mutation.

Les opérateurs de croisements et de mutations sont identiques que les précédentes approches pour la partie de la solution, cependant, pour les taux, il a été décidé de les changer ainsi :

- **Pour le croisement**, on utilise la même méthode pour les différents taux, on nomme $te_i$ le taux (croisement ou mutation) de l'enfant *i*, et $tp_i$ le taux du parent, et $\alpha$ un réel aléatoire entre [0 ; 1] :

$$te_1 = tp_1 * \alpha + tp_2 * (1 - \alpha)$$
$$te_2 = tp_1 * (1 - \alpha) + tp_2 * \alpha$$

- **Pour la mutation**, on utilise la même méthode pour les différents taux, Soit *tm* le taux résultant de la mutation, *tmp* le taux provenant de l'individu qui va muté, et $\alpha$ un réel aléatoire entre [0 ; 1] :

$$tm = tmp * \alpha$$

A l'initialisation de la population, on fixe des bornes pour les taux de croisements et de mutations pour que chaque individu ait un taux aléatoire bornée. La taille de la population étant fixé à 100.

## 4. Résultats

Après plusieurs tests, qui se déroulaient sur différents fichier avec plusieurs exécutions, et en faisant varier les différent paramètres comme la taille de la population et les différents taux, il en est sortie que la dernière approche présente les moins bons résultats et que les deux autres approches, à nombre d'individu égale, était au coude à coude bien qu'une différence semblait donner l'avantage à l'évolution adaptatif.

Cependant, les résultats ne sont pas tranchés et mériteraient une plus longue étude, notamment pour réussir à paramétrer correctement la troisième approche. Beaucoup de facteur rentre en compte pour paramétrer ces méthodes. Peut-être que la méthode auto-adaptative présenterait de meilleur résultat en changeant la méthode de sélection ou en affinant plus les bornes.

# III. Algorithme Choisie

Il est décidé en fin de compte, de présenter pour le tournoi “Bourdieu avait raison” qui utilisera donc la deuxième approche où l’on modifie les taux de croisement proportionnellement à la différence entre les résultats de deux populations. Les paramètres utilisé suivant :

| Nombre d’individu | Taux de croisement* | Taux de mutation* | Taux d’élitisme | Nombre tournoi | $\varepsilon$ |
|---|---|---|---|---|---|
| 1’000 | .10 | .80 | .10 | 8 | 0.001 |

*utilisé à l’instanciation et pour la réinitialisation

# IV. Conclusion

Le projet portant sur l’optimisation du problème du voyageur de commerce a permis de mettre en oeuvre et de tester plusieurs algorithmes méta-heuristique sur un problème que ce soit la Colonie de Fourmis, les algorithmes évolutionnaires ou bien le recuit simulé (qui n’a pas été évoqué dans ce rapport car les autres algorithmes ont rapidement fourni de meilleurs résultat). Ce projet a servi ainsi à mieux appréhender ces algorithmes abordés dans le cadre du cours.

L’étude des algorithmes évolutionnaires fut très intéressant, et il reste frustrant de ne pas pouvoir les étudier plus amplement du fait qu’il y a beaucoup de paramètres à définir.

# V. Bibliographie

[1] Maxim A. Dulebenets, Masoud Kavoosi, Olumide Abioye and Junayed Pasha, *A Self-Adaptive Evolutionary Algorithm for the Berth Scheduling Problem: Towards Efficient Parameter Control* [mdpi.com]
[2] Les cours de Métaheuristique de A.Blansché