



PROJET PTS

.....

***PROBLÈME DU  
VOYAGEUR DE  
COMMERCE***

.....

LATIF WAIL - BADJI KHADIDIATOU -  
DECHANE SANAÂ



## **Remerciements**

*Avant toute chose, il nous paraît opportun de remercier tous ceux qui ont contribué à la réussite de notre étude par leur implication et leur sens de responsabilité.*

*Nous tenons à remercier tout particulièrement et à témoigner notre reconnaissance à M. Boisson, notre tuteur de projet, qui nous a accompagnés le long de cette période de projet, par son suivi régulier et ses conseils pertinents.*

*Mais aussi sur l'organisation et l'avancement de notre étude et sa disponibilité pour répondre aux problèmes que nous avons rencontrés.*

*Nous remercions aussi l'ESILV de nous avoir proposé un projet aussi intéressant.*

# Tables des matières

<i>Remerciements .....</i>	<b>1</b>
<i>Tables des matières.....</i>	<b>2</b>
<i>Introduction.....</i>	<b>3</b>
<i>Présentation du problème.....</i>	<b>4</b>
<b>Etat de l'art.....</b>	<b>5</b>
<b>1- Principe de l'algorithme génétique .....</b>	<b>6</b>
<b>2- Notre algorithme appliqué au problème .....</b>	<b>10</b>
<b>2-1 L'algorithme.....</b>	<b>10</b>
<b>2-2 Observations et résultats .....</b>	<b>16</b>
<b>2.3 API / Interface web .....</b>	<b>18</b>
<b>Planning de notre projet .....</b>	<b>20</b>
<b>Diagramme UML .....</b>	<b>21</b>
<b>Conclusion .....</b>	<b>23</b>
<b>Sources.....</b>	<b>24</b>

## ***Introduction***

Dans le cadre de notre Projet Technique et Scientifique, différents sujets, tous plus différents les uns des autres nous ont été proposés. Nous avons choisi un problème d'optimisation mathématique car l'idée de travailler sur un sujet où existent déjà de nombreuses solutions, et d'être dans une démarche de recherches pour essayer de trouver autre chose de peut-être meilleur nous plaisait beaucoup. Ce problème est le suivant :

### *Traitements du problème du voyageur de commerce par les algorithmes génétiques*

Nous avons passé de nombreuses semaines à faire des recherches sur le PVC (Problème du voyageur de Commerce), sur les algorithmes génétiques, sur ce qui existait, ce qui marchait, ce qui marchait moins bien... Ce jusqu'à obtenir des données suffisantes pour savoir dans quelle direction nous voulions aller.

Nous avons finalement fait le choix d'implémenter notre propre algorithme et de le comparer aux algorithmes déjà existants (performance, temps d'exécution...). Nous avons aussi décidé d'implémenter une interface web afin d'améliorer l'affichage des résultats (comparé à un affichage sur console) mais aussi de faciliter la saisie de données par l'utilisateur.

Concernant le langage de programmation, nous avons choisi le C# car nous trois connaissons ce langage. Ce langage est orienté objet, il va ainsi nous permettre de structurer notre code et faciliter le travail en équipe. Nous l'avons aussi choisi dans le but de monter en compétence dessus (interface web). En complément, nous utiliserons du ReactJs (une bibliothèque de JavaScript) pour l'interface web.

## ***Présentation du problème***

Le PVC (Problème du Voyageur de Commerce) est un problème mathématique bien connu. Le principe est le suivant :

*Un voyageur de commerce doit visiter une et une seule fois un nombre fini de villes et revenir à son point d'origine. Trouver l'ordre de visite des villes qui minimise la distance totale parcourue par le voyageur.*

Si ça a l'air plutôt simple au premier abord, le nombre de trajets possibles pour  $n$  villes est  $n!$  (factorielle  $n$ ). Trouver la meilleure solution s'avère donc bien ardu au fur et à mesure que le nombre de villes à parcourir augmente. Si bien qu'on est obligé de passer par des algorithmes pour le faire.



Néanmoins là encore, trouver la solution n'est pas si simple, car le PVC appartient à la famille des problèmes dits NP-complets (c'est à dire un problème complet pour la classe NP - Non deterministic Polynomial). En théorie de la complexité, un problème NP-complet est un problème de décision vérifiant les propriétés suivantes :

- il est possible de vérifier une solution efficacement (en temps polynomial) ; la classe des problèmes vérifiant cette propriété est notée NP ;
  - tous les problèmes de la classe NP se ramènent à celui-ci via une réduction polynomiale ; cela signifie que le problème est au moins aussi difficile que tous les autres problèmes de la classe NP.

Si on trouve une solution en temps polynomial à l'un de ces problèmes, alors cette solution peut être utilisée pour résoudre, en temps polynomial, tous les problèmes NP-complets, et plus généralement tous les problèmes NP. Il y a même une récompense d'1 million de dollars pour celui qui y arrivera. C'est dire la complexité de la chose.

## ***Etat de l'art***

En faisant nos recherches sur l'algorithme génétique, nous nous sommes aussi intéressés aux autres algorithmes résolvant le problème du voyageur de commerce. Tout d'abord, de manière générale, il existe deux types d'algorithmes pour résoudre un problème :

- Les méthodes exactes permettant d'obtenir une solution optimale à chaque fois, mais le temps de calcul peut être long si le problème est compliqué à résoudre.
- Les méthodes approchées, ou encore appelées heuristiques, permettent quant à elles d'obtenir rapidement une solution approchée, mais qui n'est donc pas toujours optimale. L'algorithme génétique est donc une méthode approchée.

Concernant les autres algorithmes possibles résolvant le problème du voyageur de commerce, il existe par exemple la méthode des colonies de fourmis, les algorithmes gloutons etc.

Ces méthodes sont celles qui ont été principalement utilisées pour résoudre les problèmes de type PVC. Il existe également d'autres méthodes qui sont en réalité des mélanges de plusieurs algorithmes.

Aujourd'hui les algorithmes génétiques sont massivement utilisés pour résoudre ce problème en raison de la qualité de la solution obtenue (même si elle n'est pas optimale) en un temps plutôt correct.

## 1- Principe de l'algorithme génétique

Les algorithmes génétiques sont des algorithmes itératifs – basés sur la répétition – stochastiques – en référence au hasard.

Ils utilisent la théorie de Darwin sur l'évolution des espèces (d'où leur nom). Elle repose sur trois principes :

- Le principe de variation : Chaque individu au sein d'une population est unique. Ces différences, plus ou moins importantes, vont être décisives dans le processus de sélection.
- Le principe d'adaptation : Les individus les plus adaptés à leur environnement atteignent plus facilement l'âge adulte. Ceux ayant une meilleure capacité de survie pourront donc se reproduire davantage.
- Le principe d'hérité : Les caractéristiques des individus doivent être héréditaires pour pouvoir être transmises à leur descendance. Ce mécanisme permettra de faire évoluer l'espèce pour partager les caractéristiques avantageuse à sa survie.

Ainsi, nous retrouvons dans les algorithmes génétiques ces termes :

- La **population** qui est l'ensemble des solutions envisageables.
- L'**individu** qui représente une solution.
- La **génération** qui est une itération de l'algorithme.

Dans les algorithmes génétiques, l'application des 3 principes de la théorie de Darwin donne naissance à des opérateurs d'évolution des populations :

- Le principe d'adaptation est appliqué par la **sélection**. Cela consiste à choisir les individus les mieux adaptés afin d'avoir une population de solution la plus proche de converger vers l'optimum global. On se base sur un critère appelé **fitness** et dont la valeur représente la qualité de la solution.
- Le principe d'hérité est appliqué par le **croisement (ou crossover)**. Cela consiste à fusionner les particularités de deux individus (les parents), à partir d'un pivot choisi au hasard, afin d'obtenir un ou deux enfants. Il existe aussi une variante, le croisement double, dans lequel on utilise pas un mais deux pivots.

Il permet d'éviter une convergence prématuée de l'algorithme vers un extrémum local.

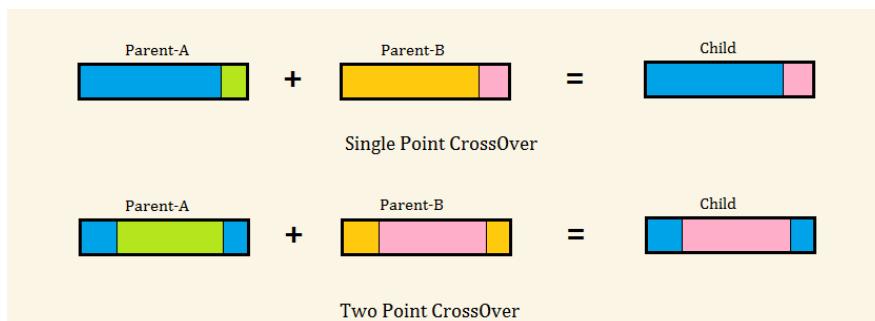


Figure 1: schéma des deux types de croisement

- Le principe de variation est appliqué par la **mutation**. Cela consiste à altérer un gène dans un individu selon un facteur de mutation. Ce facteur est la probabilité qu'une mutation soit effectuée sur un individu. Comme le croisement, il permet d'éviter une convergence prématuée.

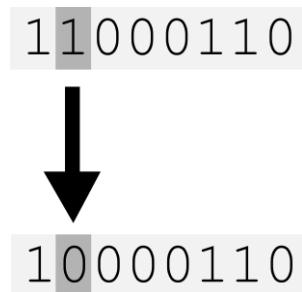


Figure 2: Exemple de mutation sur une population binaire

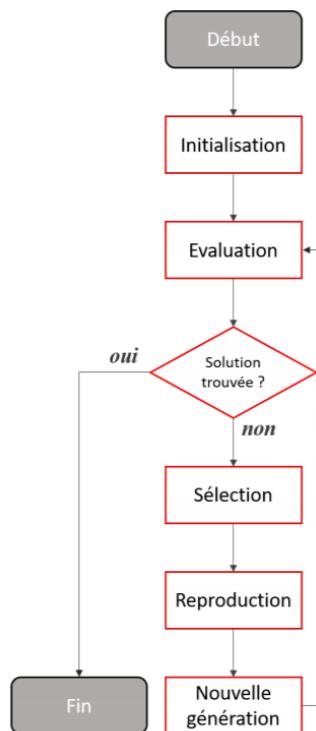


Figure 3 : schéma récapitulatif de l'algorithme génétique

**Dans le cas de notre projet, voici la terminologie utilisée :**

- Un individu -> un trajet
- Une population -> un ensemble de trajets
- Une génération -> une nouvelle population de trajets se constitue, en remplaçant partiellement ou totalement les individus (les trajets) de la génération précédente par les nouveaux individus créés. C'est au bout de plusieurs générations qu'on converge vers une meilleure solution.
- la fitness -> la distance totale (somme des distances à vol d'oiseau entre chaque ville) d'un trajet en kilomètres. Ainsi, plus la valeur est faible, meilleure est la fitness et donc la solution.
- L'évaluation -> quand on évalue si l'individu (le trajet) a une fitness élevée ou non (si l'individu correspond à une bonne solution)
- La reproduction -> La méthode de croisement appliquée aux individus

**Explications des différentes méthodes de sélection pour programmer cet algorithme :**

**• Sélection par la roulette wheel :**

Il s'agit de la méthode la plus fréquemment utilisée. On utilise le principe d'une roue de loterie biaisée. A chaque individu correspond un secteur d'une roue (la roue de loterie). La superficie de ce secteur est proportionnelle à la capacité (fitness) de l'individu à s'adapter : plus un individu a une fonction fitness élevée plus il a de chance d'être sélectionné. A chaque secteur correspond une probabilité cumulée. On tire un nombre aléatoire entre 0 et 1. L'individu sélectionnée est celui dont le secteur est pointé, c'est-à-dire dont la plage de probabilité cumulée contient le nombre tiré au sort. Dans l'illustration ci-dessous, la population est constituée de 6 individus auxquels un secteur de roue a été attribué. On visualise bien que l'individu 3 a moins de chance d'être sélectionné que l'individu 1. Si le hasard tire la valeur 0,5, c'est l'individu 4 qui est désigné puisque la plage de probabilité cumulée lui correspondant se situe entre 0,39 et 0,60.

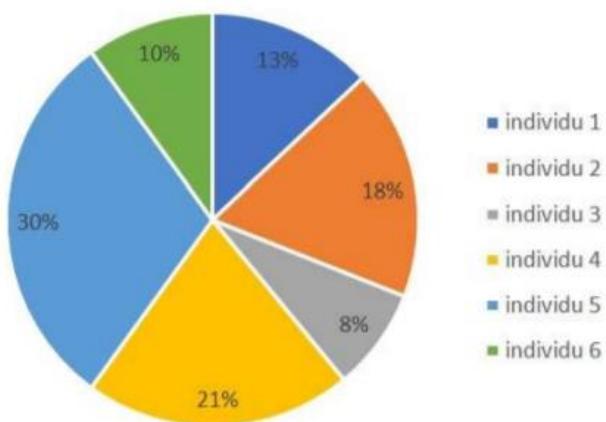


Figure 4 : Sélection par la roulette wheel

- **Sélection par tournoi :**

C'est la méthode donnant en général les meilleurs résultats. Elle possède un paramètre  $T$ , la taille du tournoi. Le principe est le suivant : on effectue un tirage de  $T$  individus dans la population et chaque tirage donne lieu à un combat. L'individu dont la fonction fitness est la plus élevée, donc celui qui a remporté «le combat» est désigné comme parent. On reproduit ce processus autant de fois que nécessite l'obtention des  $n$  nouveaux individus de la 2e génération.

- **Sélection universelle stochastique :**

Cette méthode est très peu utilisée et possède une très faible variance. On partitionne un segment en tronçons. Chaque individu est associé à un tronçon dont la longueur est proportionnelle à son fitness. On détermine ensuite  $n$  points équidistants sur le segment,  $n$  étant le nombre d'individus devant composer la génération suivante. Les individus pointés par les points dans le segment seront ceux qui figureront dans la nouvelle génération.

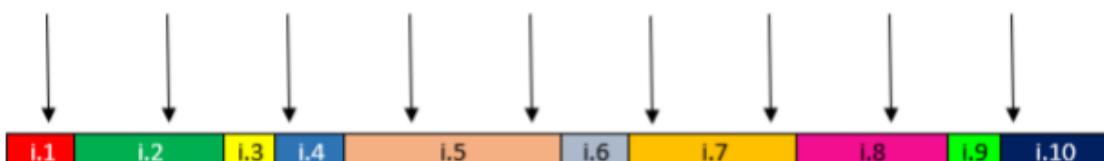


Figure 5 : Sélection universelle stochastique

- **Sélection élitiste :**

C'est la seule méthode de sélection qui soit déterministe. Son principe est de classer par ordre croissant les individus en fonction de leur fitness. Ensuite, on sélectionne un ou plusieurs individus parmi les meilleurs de ce classement qui constitueront la population de parents. On génère ensuite par croisement les individus enfants nécessaires à la constitution de la génération suivante. Les individus les moins performants sont totalement éliminés de la population, et le meilleur individu est toujours sélectionné – on dit que cette sélection est élitiste.

Cette méthode présente une convergence fortement prématurée car on a une variance et une diversité presque nulles : les individus les moins bons n'ont aucune chance de survivre. Elle permet cependant de ne pas perdre des individus avec une fonction fitness élevée.

## 2- Notre algorithme appliqué au problème

### 2-1 L'algorithme

Après avoir décrit en quoi consistait l'algorithme génétique et les possibles méthodes utilisées. Nous allons expliquer ce que nous avons décidé de faire. En effet, concernant les méthodes de sélection, nous avons choisi d'en retenir deux. Il s'agit de la sélection par tournoi (la méthode donnant les meilleurs résultat) et la sélection élitiste (la méthode la plus déterministe et la plus rapide). Ces deux méthodes seront nos méthodes témoins à notre méthode personnelle, afin de comparer les résultats et de voir si notre méthode personnelle donne des bons résultats. Nous utiliserons d'ailleurs la méthode stochastique dans notre méthode personnelle.

Concernant les méthodes de croisement, nous avons choisi le croisement double, car le fait de prendre plusieurs parties d'un même parent nous a semblé plus aléatoire que le croisement simple.

Concernant la méthode de mutation nous l'utiliserons aussi afin d'inverser des villes au sein d'un trajet.

#### Notre base de données :

Concernant la base de données nous avons trouvé une base de données sur internet qui contient toutes les villes de France avec le code postal de chacune et surtout les coordonnées gps (longitude et latitude) de chaque ville. Ce qui nous a permis de calculer la distance à vol d'oiseau entre chaque ville.

Voyons maintenant, en quoi consiste notre méthode personnelle :

#### a) Initialisation

Tout d'abord, en guise de préliminaires, nous avons décidé que plutôt que de choisir aléatoirement la population initiale, ce sera une population déjà assez optimisée (trajets aux fitness pas trop mauvaises). L'idée est de générer la première population par un algorithme que l'on a baptisé AdamEtEve.

#### Comment fonctionne notre algorithme AdamEtEve?

Après de nombreux tests utilisant des algorithmes mis en place en suivant le principe de Dijkstra et glouton, nous avons pu mettre en place un algorithme donnant le plus court chemin permettant de traverser un ensemble de villes en parcourant le moins de distance possible. L'algorithme a été testé sur l'ensemble de données suivant :

"PM" => 776 km,

"PL" => 225 km,

"PO" => 468 km,

"PN" => 388 km,

"ML" => 1000 km,

"MO" => 314 km,

"MN" => 986 km,

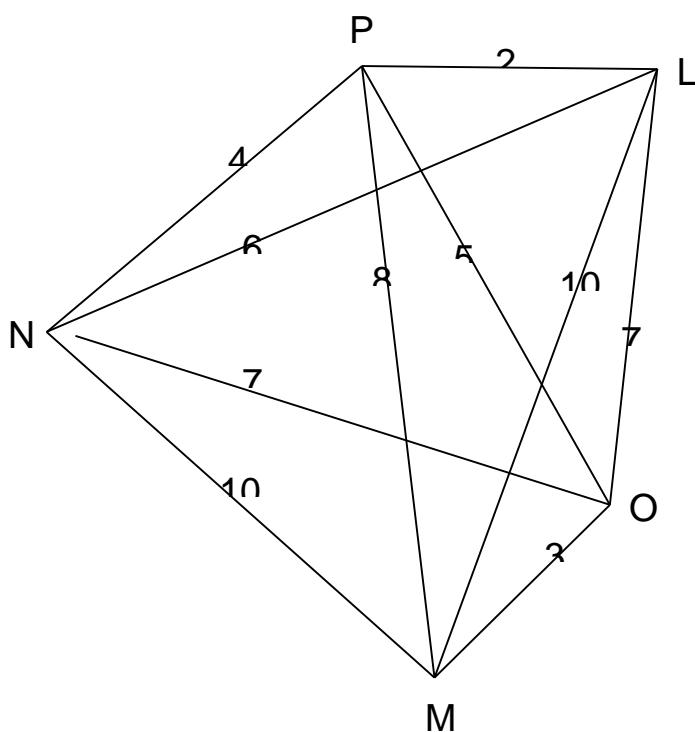
"NL" => 599 km,

"NO" => 685 km,

"LO" => 691 km

avec P= Paris, M=Marseille, L=Lille, O=Lyon, N=Nantes et PM= distance entre Paris et Marseille.

Afin de faciliter la compréhension de l'algorithme nous avons mis en place un graphique pondéré afin de représenter les villes et nous avons divisé par 100 leur distance afin de faciliter les calculs.



### Étape 1:

Prendre le sommet dont la somme des distances de ces villes voisines est la plus faible.

$$P : 2+8+5+4 = 19$$

$$L : 2+10+7+6 = 25$$

$$O : 7+3+7+5 = 22$$

$$M : 10+8+10+3 = 31$$

$$N : 6+4+7+10 = 27$$

Paris sera donc notre point de départ car sa distance de ces villes voisines est la plus faible.

## Étape 2:

Nous cherchons maintenant la ville suivante pour cela nous allons comparer les villes voisines entre elles en utilisant 3 critères : la distance avec la ville actuelle, la distance de la ville voisine la plus lointaine, ainsi que la distance de la ville voisine la plus proche hormis les villes déjà visitées.

Par exemple pour notre premier cas, la ville actuelle est Paris donc ces villes voisines sont :

ville voisine	distance avec Paris	distance de la ville voisine la plus proche (hormis les villes déjà visitées)	distance de la ville voisine la plus lointaine (hormis les villes déjà visitées)
Lille (L)	2	6	10
Lyon (O)	5	3	7
Nantes (N)	4	6	10
Marseille (M)	8	3	10

Ayant ces informations nous allons comparer les villes les unes aux autres afin de garder la plus avantageuse. Pour cela nous allons utiliser un delta que nous obtiendrons en soustrayant les distances représentées sur le tableau précédent par exemple :

Ici le delta entre Lille et Lyon est  $2-5 + 6-3 + 10-7 = 3 > 0$  donc Lyon est une meilleure solution que Lille donc nous allons garder Lyon et la comparer aux autres villes tant que Lyon a un meilleur delta.

Delta entre Lyon et Nantes :  $5-4 + 3-6 + 7-10 = -5 < 0$  donc Lyon est une meilleure solution, donc on garde Lyon.

Delta entre Lyon et Marseille :  $5-8 + 3-3 + 7-10 = -6 < 0$  donc on garde Lyon.

Notre trajet est donc PO.

Dans le cas d'un delta = 0 la meilleure solution est la solution la plus proche de la ville actuelle. Si la distance avec la ville actuelle est égale alors nous choisirons la ville dont la distance avec la ville la plus proche est la plus faible.

Nous allons faire tourner cette algorithme jusqu'à arriver au deux dernières villes.

Ainsi, on recommence l'étape 2 pour trouver la ville suivante :

ville voisine	distance avec la ville actuelle	distance de la ville voisine la plus proche (hormis les villes déjà visitées)	distance de la ville voisine la plus lointaine (hormis les villes déjà visitées)
Lille (L)	7	6	10
Nantes (N)	7	6	10
Marseille (M)	3	10	10

Ici en calculant les différents deltas nous obtenons comme meilleur solution M car les deltas sont égaux on prend donc la ville la plus proche.

Notre trajet est désormais : POM

Il ne reste plus que deux villes. Le delta sera calculé en utilisant la distance avec la ville actuelle, et la distance avec la ville de départ.

ville voisine	distance avec la ville actuelle	distance de la ville de départ
Lille (L)	10	2
Nantes (N)	10	4

Ici cela semble logique la prochaine ville sera Nantes car la distance entre Nantes et la ville de départ (Paris) est plus loin que Lyon.

Notre trajet est donc : POMNL

### Étape 3:

Ajouter la ville de départ à la fin du trajet :

Notre trajet final est donc :  $POMNLP = 2592 \text{ km}$  qui est le résultat obtenu par notre algorithme testant toute les possibilités de trajet et retournant le plus court chemin.

- 1) On fait tourner l'algorithme AdamEtEve créé par nous-mêmes (voir ci-dessus) et qui nous renvoie un trajet d'une fitness avantageuse (2 fois).
- 2) Les deux trajets récupérés constitueront nos 2 parents.
- 3) Nous les faisons se reproduire entre eux à un haut taux de mutation pour créer notre population initiale de n trajets.

#### b) Sélection

Une fois la population initiale obtenue, nous allons procéder à la prochaine étape : Le Croisement.

- 1) On commence par trier les trajets par fitness. (*Rappel : la fitness est la distance en kilomètres, donc on cherche la meilleure fitness qui sera la plus petite*).
- 2) On divise ensuite la population en deux. Les 15% meilleurs constitueront notre première sous-population (notons PopA) et les 85% restants la population la sous-population B (notons PopB).



### c) Croisement

L'idée est de faire se reproduire les bons individus avec les moins bons. Contrairement à la méthode élitiste où l'on ne garde que les meilleurs, nous avons décidé de procéder ainsi car il se peut qu'un parent a priori mauvais possède tout de même un gène (une partie de trajet) puissant. Ainsi, on garde une chance de récupérer de bons éléments.

- 3) Chaque individu de PopA se reproduit avec un individu de PopB. L'heureux élu est déterminé par une sélection stochastique sur PopB pour accroître la part de hasard. Le croisement se fait avec un taux de mutation élevé pour éviter de stagner.
- 4) On répète l'opération autant que nécessaire pour obtenir le nombre n d'individus désiré.

Ainsi, on a obtenu la population de n individus de la 1ère génération !

Pour créer les générations suivantes, on reprend à partir de la sélection (l'initialisation ne se fait qu'une seule fois au tout début).

## 2-2 Observations et résultats

Voici ci-dessous quelques graphiques afin de comparer les performances de notre méthode personnelle et celles des méthodes témoins dites classiques (Élitiste et Tournoi) en faisant varier quelques paramètres. Pour les 3 graphiques nous avons fait varier chaque paramètre selon l'augmentation du nombre de villes.

Pour le premier graphique (figure 6) nous avons comparé le temps d'exécution. Et nous pouvons voir que même si le temps d'exécution de notre méthode personnelle augmente de plus en plus en augmentant le nombre de villes, il reste toujours inférieur à celui de la méthode Tournoi. Nous pouvons aussi observer qu'il est plus moins égal à celui de la méthode Élitiste. Ce sont donc des résultats très satisfaisants pour nous.

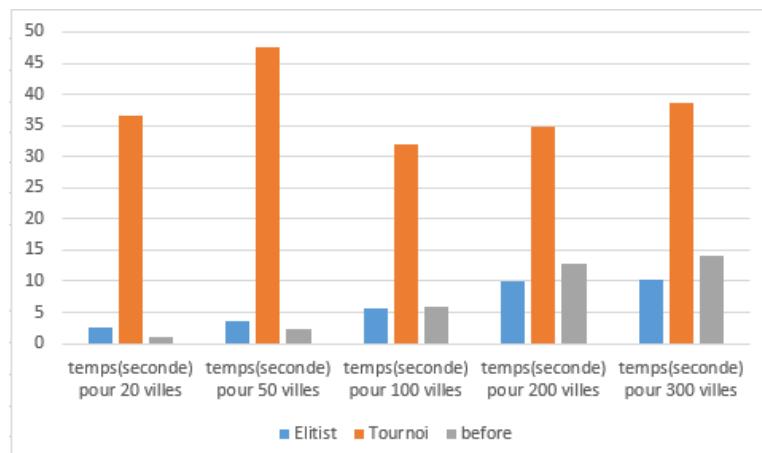


Figure 6

Le second graphique (figure 7) compare le nombre de kilomètres d'un trajet selon le nombre de villes sélectionnées. Nous pouvons constater que plus le nombre de villes est élevé (pour 200 et 300 villes) plus le trajet obtenu par notre méthode est court. Ce sont donc des résultats très satisfaisants pour nous.

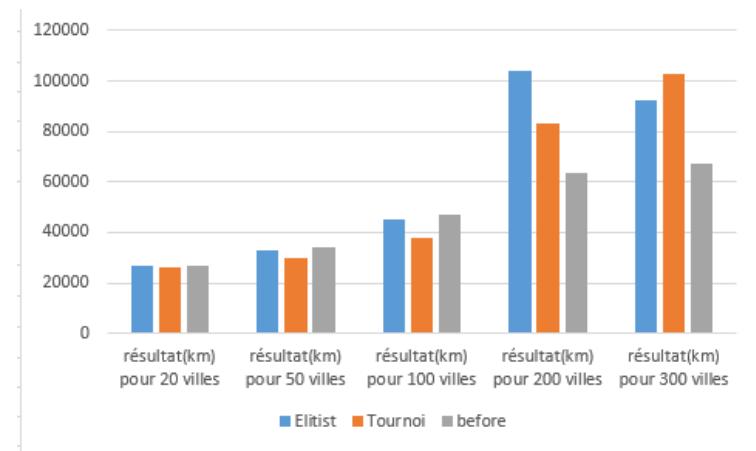


Figure 7

Le troisième graphique quant à lui (Figure 8) compare le nombre de générations au bout duquel on obtient la meilleure solution. On peut voir que la méthode Élitiste converge plus vite vers une solution, ce qui est normal car c'est la plus déterministe. Mais notre méthode converge plus vite vers une solution que la méthode Tournoi. C'est un mauvais signe car cela veut dire que cela converge vers une solution locale qui n'est pas forcément une solution optimale. Or la méthode Tournoi donne de meilleur résultat que pour 20, 50 et 100 villes car elle converge moins vite (Figure 7). Cependant avec plus de 200 villes notre méthode obtient de meilleur résultat, ceci peut s'expliquer par notre méthode de sélection qui est plus aléatoire que celle de la méthode Tournoi. Donc avec plus de 200 villes il est préférable d'utiliser notre méthode.

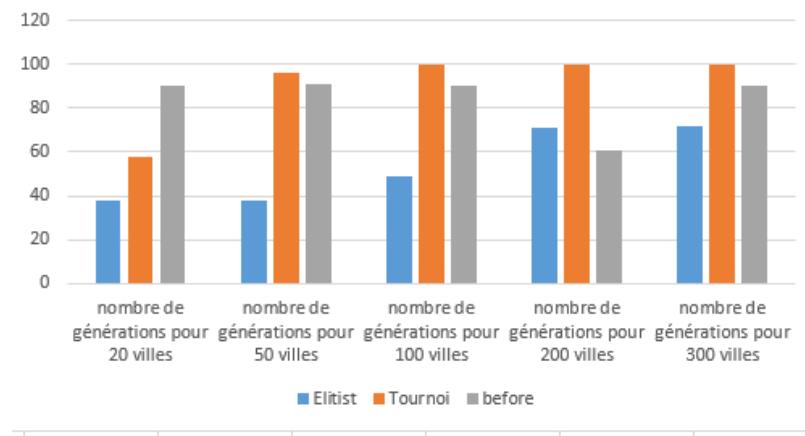


Figure 8

## 2-3 API / Interface web

Une fois le back end développé et fonctionnel, nous avons décidé de mettre en place une interface web afin de faciliter la lecture des résultats et la saisie des paramètres.

Pour cela, nous avons développé une API avec plusieurs Endpoints (contrôleurs) permettant d'accéder à l'application à l'aide de requêtes http (Ajax) ce qu'on appelle communément une RestFull API. Pour ce faire, nous avons mis en place 3 contrôleurs. Le premier permettant d'obtenir un token de session, la liste des méthodes de sélection disponibles et pour finir les facteurs de mutation par défaut pour chacune des méthodes. Le second permettant d'avoir une liste de suggestions de 5 villes grâce à une chaîne de caractères (lors de la saisie utilisateur) et le dernier permettant d'avoir les résultats de l'algorithme en utilisant une méthode de sélection spécifique ainsi qu'un facteur de mutation qui seront sélectionnés via l'interface web.

Cette interface web s'adresse à toute personne qui veut comparer nos méthodes.

Comment fonctionne -t-elle ?

Notre interface web est simple à utiliser afin de faciliter à l'utilisateur la comparaison des différentes méthodes. Tout d'abord il doit sélectionner les villes pour construire son trajet ou en cochant directement la case "Select random cities" en entrant le nombre de villes souhaité). Ensuite il sélectionne les méthodes qu'il souhaite comparer avec le taux de mutation de son choix. Il clique alors par la suite sur le bouton "compare" (voir Figure 9) et les résultats s'affichent !

Cities to visite :

Add a city

Cormoranche-sur-Saône (01290)Tossiat (01250)Replonges (01620)Corcelles (01110)Faramans (01800)

Select random cities

Selection Methodes

SelectBefore

Mutation Factor : 1%

Elitist

Mutation Factor : 30%

X

Tournament

Mutation Factor : 1%

X

+ Compare

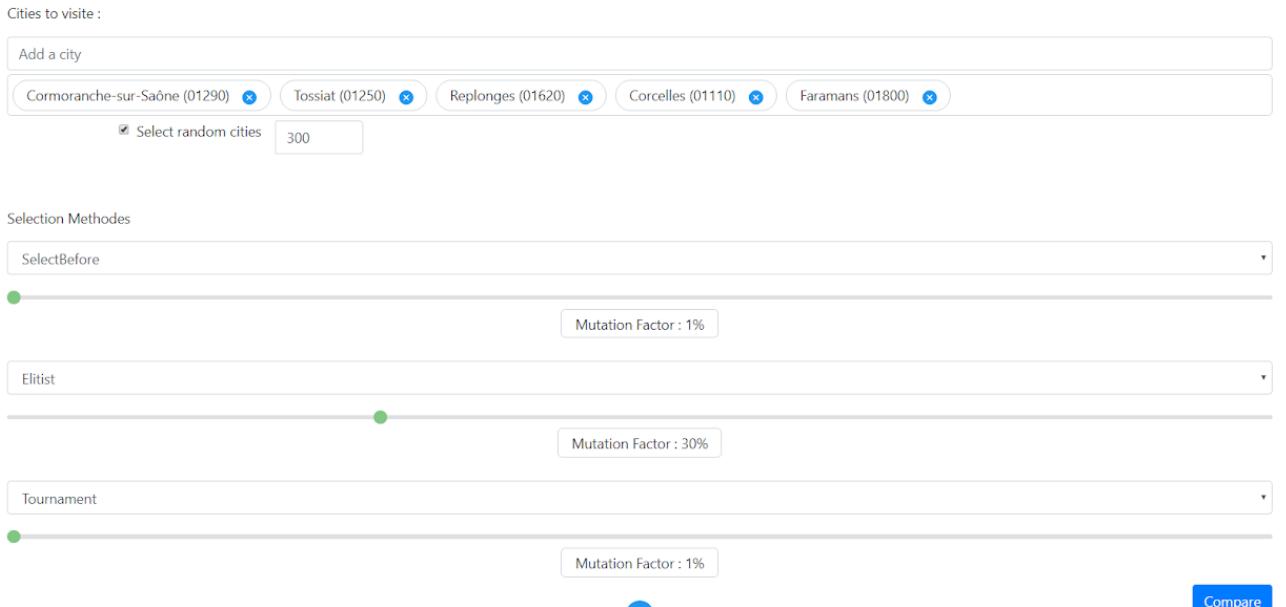
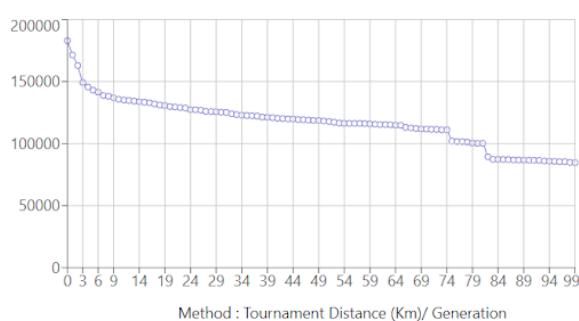
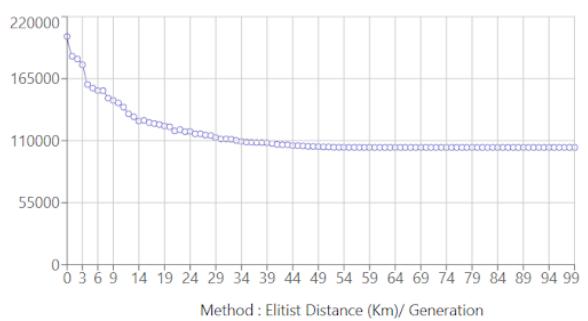
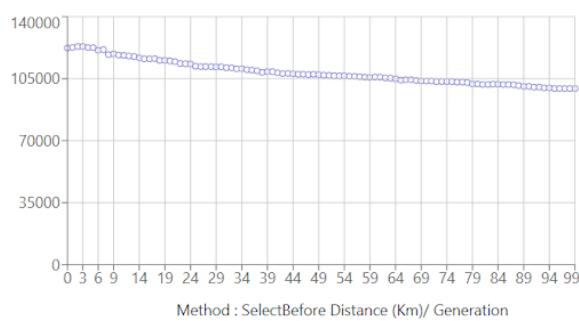


Figure 9

Pour pouvoir mieux appréhender les résultats, nous avons décidé de les afficher sous forme de graphiques. Ils décrivent l'avancée du nombre de générations en fonction du nombre de km du trajet total. Ci-dessous un exemple de résultats pour 300 villes aléatoires :

## Find your way.

Compare different algorithms to find the shortest route for your travel.

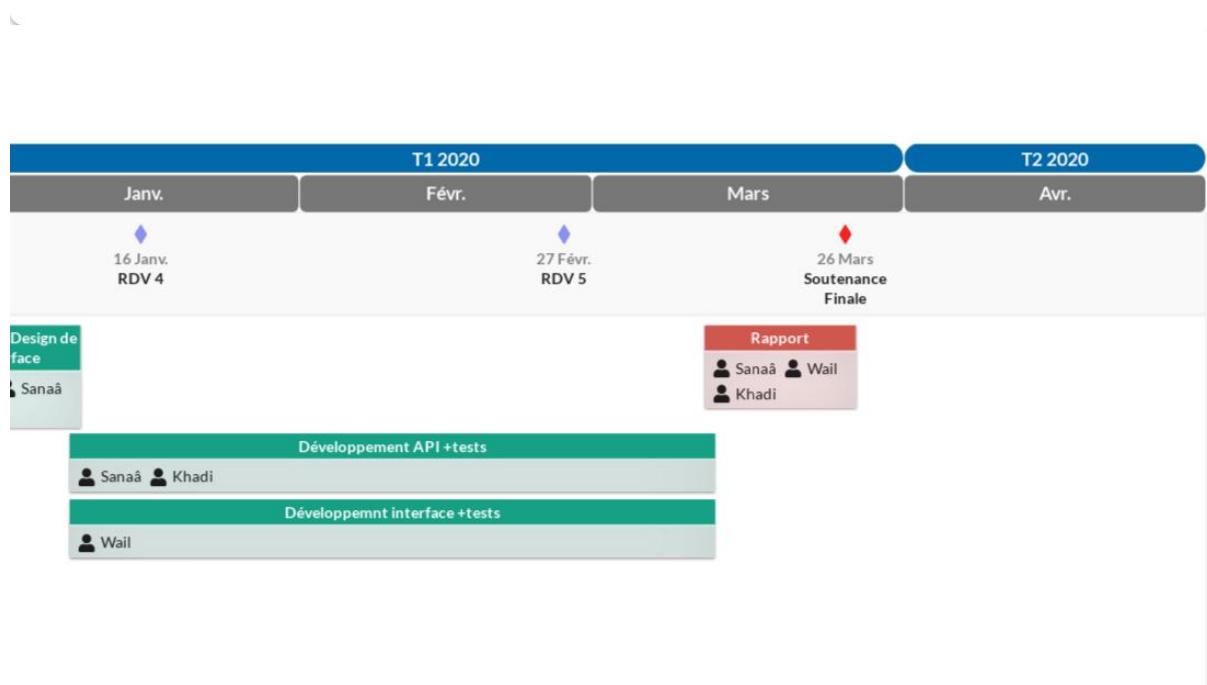
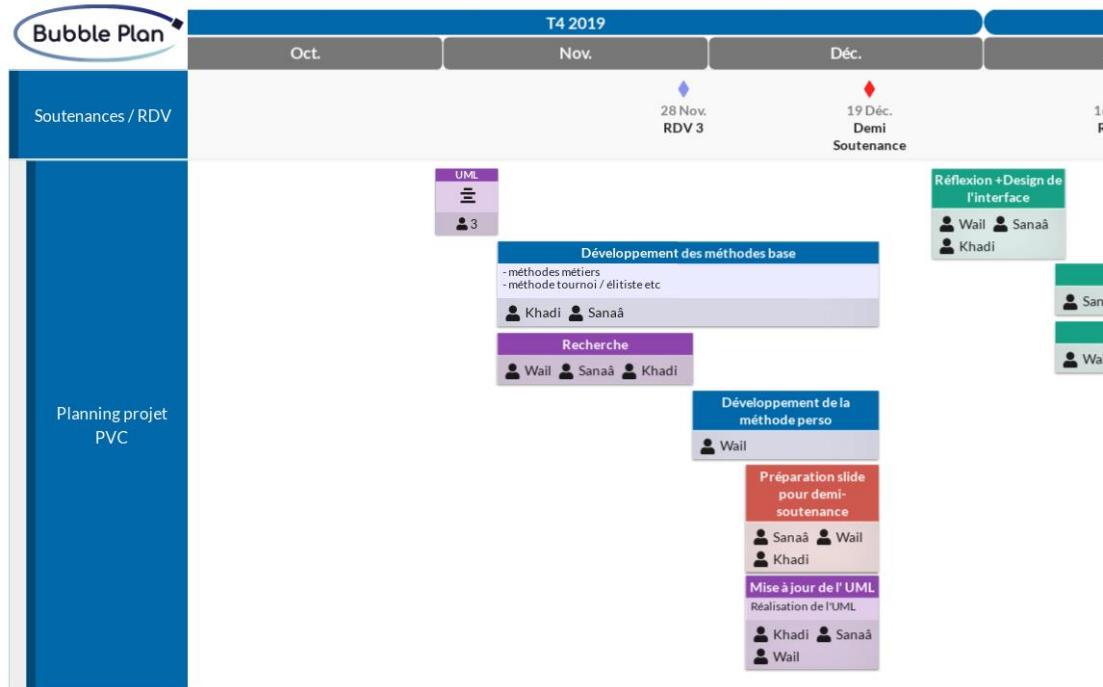


## Planning de notre projet :

Ci-dessous notre planning (image divisée en 2 afin de faciliter la lecture).

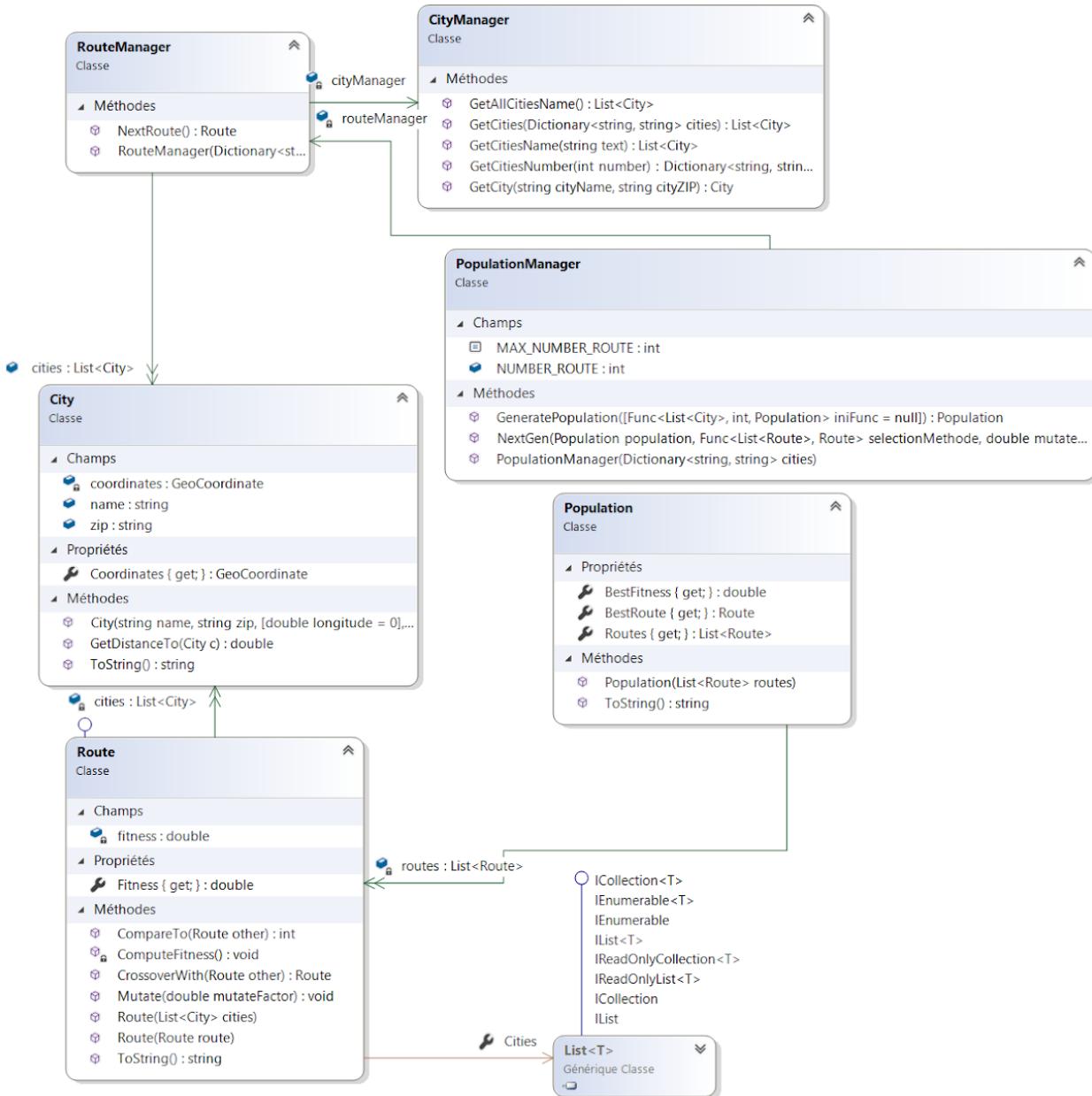
Nous l'avons élaboré dès les premières semaines de projet sur le conseil de Mr BOISSON pour avoir une vue d'ensemble et nous repérer au fil du temps. Nous avons essayé de nous répartir les tâches de la façon la plus efficace et équitable possible.

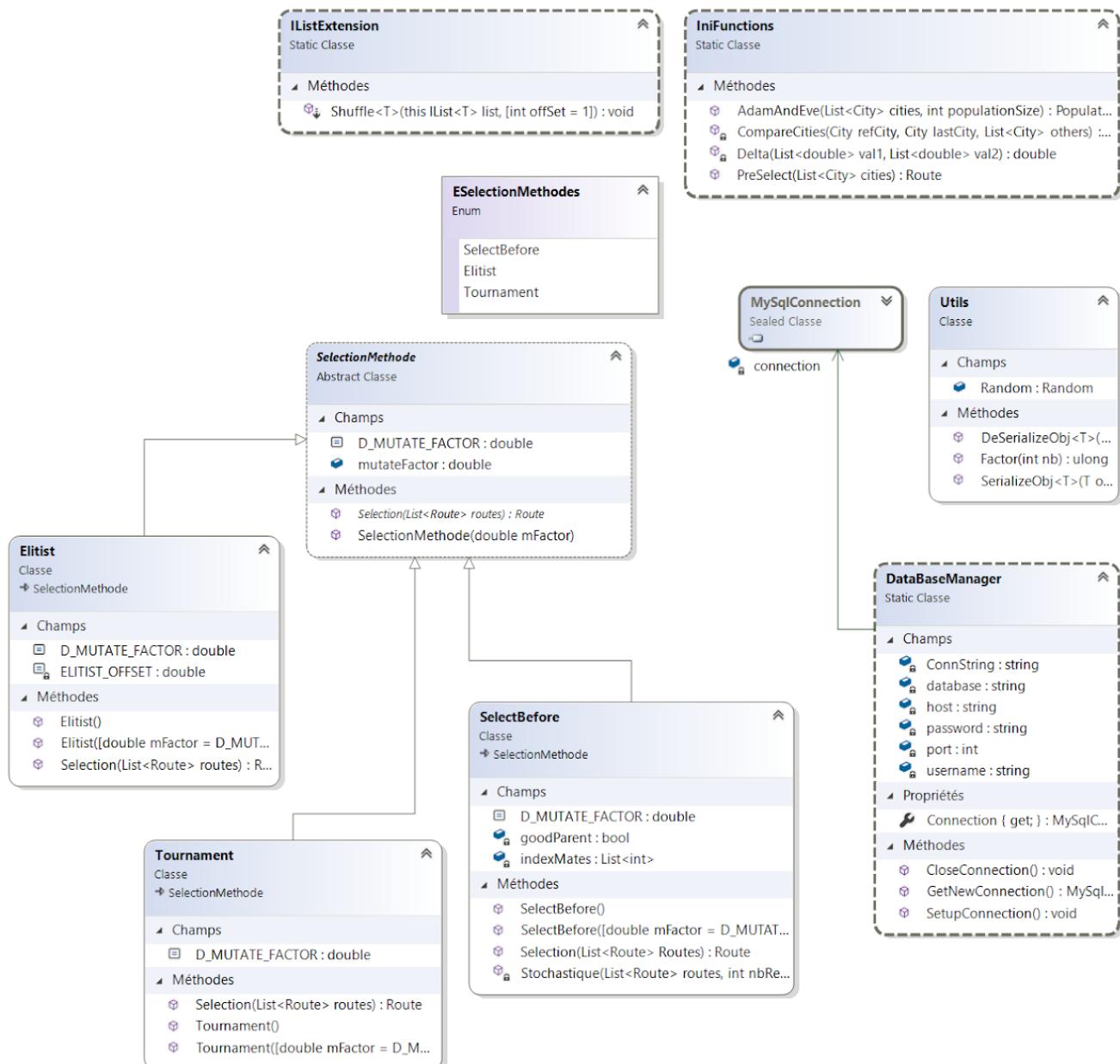
Planning PVC



## Diagramme UML

Ci-dessous notre diagramme UML (image divisée en 2 afin de faciliter la lecture).





## **Conclusion**

Après une année passée sur ce projet, nous sommes plutôt fiers du résultat. Nous ne nous attendions certainement pas à en apprendre autant. Ce projet fut un véritable projet de recherche. Les premières semaines passées à faire de la veille technologique nous ont paru longues, mais auront été tout sauf inutiles. Elles ont servi à forger une base solide pour notre solution.

Il s'agissait donc de la partie la plus importante, mais la partie développement n'était pas non plus à négliger. Nous avons d'ailleurs parfois eu du mal à nous répartir les tâches car la partie développement de nos méthodes témoins et de notre méthode personnelle a duré plus longtemps que prévu. En effet, en codant nous sont apparus certains problèmes que nous n'avions pas anticipés, mais à chaque fois nous avons su trouver une solution adéquate.

Ce projet a permis à chacun d'entre nous d'améliorer nos compétences individuelles mais aussi collectives. Nous nous tenions constamment informés de ce que modifiaient les uns ou les autres. Surtout, dès lors que l'un de nous était en difficulté sur une tâche, les autres étaient prêts à intervenir et à aider pour plus d'efficacité.

Finalement, à partir d'un énoncé aux apparences banales se cachait un véritable projet d'équipe qui a été enrichissant et formateur pour chacun de nous.

## Sources

[https://fr.wikipedia.org/wiki/Algorithme\\_g%C3%A9n%C3%A9tique](https://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique)

<https://khayyam.developpez.com/articles/algo/genetic/>

[http://igm.univ-mlv.fr/~dr/XPOSE2013/tleroux\\_genetic\\_algorithm/fonctionnement.html](http://igm.univ-mlv.fr/~dr/XPOSE2013/tleroux_genetic_algorithm/fonctionnement.html)

<https://hal-enac.archives-ouvertes.fr/hal-00934534/file/284.pdf>

<https://interstices.info/glossaire/algorithme-genetique/>

[http://souqueta.free.fr/Project/files/TE\\_AG.pdf](http://souqueta.free.fr/Project/files/TE_AG.pdf)

<https://fr.mathworks.com/discovery/genetic-algorithm.html>

<https://interstices.info/le-probleme-du-voyageur-de-commerce/>

[http://labo.algo.free.fr/pvc/algorithme\\_colonie\\_de\\_fourmis.html](http://labo.algo.free.fr/pvc/algorithme_colonie_de_fourmis.html)