

RAPPORT DE PROJET

Algorithme génétique

Auteurs :

Thibault ODOR

Joan TERIIHOANIA

Jean-Baptiste BES

Lino FAVOT

Superviseur :

Marin BOUGERET

27 mai 2020



Pour l'obtention du DUT Informatique

Université de Montpellier
2019-2020

Table des matières

1	Cahier des charges	5
1.1	Principe général	5
1.1.1	Sélection	7
1.1.2	Croisement	7
1.1.3	Mutation	8
2	Rapport technique	9
2.1	Conceptualisation	9
2.2	Analyse Fonctionnelle	10
2.3	Architecture du code	11
2.4	Manuel d'utilisation	12
3	Gestion de projet	13
3.1	Démarche personnelle	13
3.1.1	Méthode de développement	13
3.1.2	Gestion du code	13
3.1.3	Gestion des taches	14
4	Conclusion	15
5	Bibliographie	16

Remerciements


Nous souhaitons remercier tout particulièrement notre tuteur, BOUGERET Marin ainsi que l'équipe pédagogique de notre formation qui nous a accompagné tout le long de notre projet.

Glossaire

Au sens d'un algorithme génétique :

- *Individu* : Objet algorithmique correspondant à une solution potentielle.
- *Gène* : Donnée innée à un individu, il s'agit de la solution de l'individu. Sa forme peut varier (String, Integer, etc).
- *Élément d'un gène* : Prenant un gène de type chaîne de caractères, un caractère est un élément du gène.
- *Partie d'un gène* : Une partie d'un gène est une section découpée du gène de \mathbf{x} élément.
- *Valeur sélective* : Représentation numérique de la performance d'un individu à la manière d'une note.

Introduction

Au travers de ce document, des termes et expressions cliquables sont disponibles re-dirigeant vers de la documentation en ligne afin d'apporter un support documentaire éventuel aux idées décrites. Ces éléments cliquables sont indiqués par la présence d'une icône les précédant. Par exemple :  [Wikipedia](#).

Dans ce projet, nous produirons un algorithme génétique et étudierons les résultats obtenus afin d'en mesurer la pertinence. Dans cette optique, nous partirons d'un cas établi afin d'en déduire des observations par rapport aux résultats attendus et obtenus. Observations qui nous permettront de mesurer l'utilité d'un algorithme génétique.

L'algorithme génétique n'est pas un algorithme en soit, mais un modèle de conception permettant de mimer le concept d'évolution génétique par génération pour résoudre un problème. Il permet de créer pour un problème donné une solution correspondante la plus optimale possible en s'adaptant à l'environnement fourni.

L'objectif de ce projet est d'une première part, d'instancier le principe des algorithmes génétiques sur un nouveau problème. D'une seconde part, d'ajuster les paramètres de cet algorithme afin d'obtenir la meilleure solution possible.

Le choix des paramètres de la simulation, du nombre d'individus générés et du nombre de cycle notamment, sont des facteurs importants. Leur choix permet d'obtenir un ratio coût-résultat idéal. Un trop grand nombre d'individus et de cycles peut rallonger la durée de calcul inutilement. Un trop faible nombre peut produire des individus qui ne sont pas assez performants.

Enfin, à partir de nos observations, d'**étudier** la pertinence de l'utilisation d'un algorithme génétique pour la résolution d'un problème donné au regard des résultats obtenus. *Est-ce qu'une résolution à la main ou via un algorithme classique donne de meilleurs résultats au regard de sa simplicité et du temps requis à son implantation ?*

Cette situation, bien que relativement simple, peut avoir des applications variées. Qu'il s'agisse d'une simulation naturelle ou d'un environnement économique où les individus sont des entreprises pour en déterminer le comportement le plus efficient pour, par exemple, un résultat lucratif maximal.

En premier lieu, nous allons expliquer les bases générales et le fonctionnement d'un algorithme génétique, par la suite, nous expliquerons la production d'un algorithme génétique pour un cas concret. Nous étudierons ses besoins, ses spécificités et fonctionnalités. Enfin, nous conclurons par les diverses méthodes de gestion utilisée dans le cadre de la production de cet algorithme.

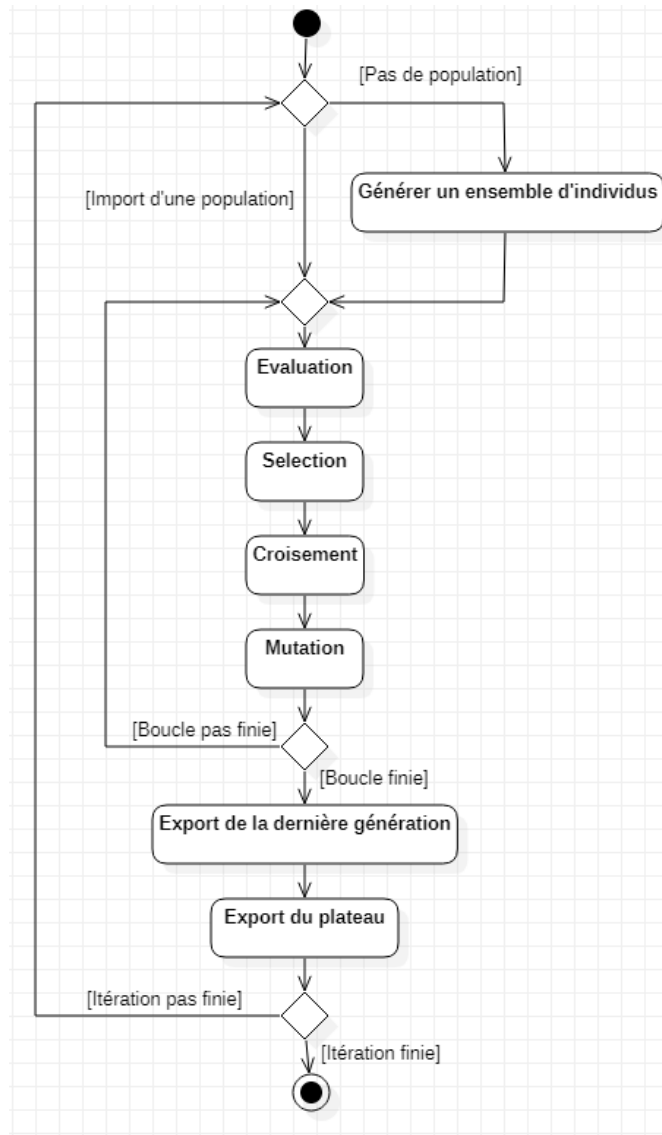
1 Cahier des charges

1.1 Principe général

Dans un environnement (**problème**), des individus (**solutions**) sont confrontés afin de déterminer lequel a le plus de valeur. *On peut considérer que la solution de chaque individu est un gène G qui leur est inné et qu'ils peuvent transmettre, croiser et muter.*

L'algorithme génétique se base sur une évolution par sélection naturelle. Le principe de \boxtimes sélection naturelle est basé sur la théorie de \boxtimes Darwin. Ce principe repose sur la base d'une valeur sélective définie à chaque individu parmi une masse plus ou moins diffuse pour obtenir, sur une période donnée, une souche détenant les paramètres les mieux adaptés à un environnement sélectionné sur des critères d'évaluation déterminés. L'individu **a** faisant parti d'une population d'une génération à un instant **t** et pouvant passer ses "gènes" **G** à un individu **b** fils de la génération à l'instant **t+1** en passant par une phase d'évaluation, sélection, croisement, mutation puis réinsertion.

Nous pouvons représenter ce schéma de réalisation par le suivant :



Il est important de garder en tête que l'algorithme et toutes les phases qui le composent, notamment les phases de sélection, mutation et croisement, doivent être optimisés. Etant donné qu'un algorithme génétique se base sur une population avec un nombre élevé pour avoir un résultat adéquat.

Important : Afin de préserver un potentiel d'évolutivité, l'algorithme ne doit ni être eugéniste (↔ Eugénisme) ni élitiste. Les individus de la population, qu'ils soient faiblement ou moyennement évalués ne doivent pas être supprimés pour conserver une certaine diversité au sein de la population et éviter une stagnation. Par la même occasion, la sélection des individus à croiser et muter ne doit pas totalement ignorer les individus faiblement ou moyennement évalués.

1.1.1 Sélection

L'algorithme sélectionne un certain nombre d'individu (ou part de la population définie) arbitrairement. Les individus sélectionnés subiront les phases suivantes de croisement et mutation. La façon dont les individus sont sélectionnés peut varier :

1. **Sélection aléatoire simple** : On sélectionne les individus de manière totalement aléatoire. Elle permet d'obtenir une sélection difforme et variée et d'ainsi varier les individus qui suivront.
2. **Sélection aléatoire pondérée** : On attribue à chaque individu une **valeur sélective** (calculée par la fonction d'évaluation) puis, en se basant sur celle-ci, on sélectionne les individus de manière aléatoire. Aussi, un individu dont la valeur est plus élevée aura plus de chance d'être sélectionné. *Il est important de garder en tête que l'algorithme ne doit pas être "sur-sélectif". Il n'est donc pas pertinent de rendre la valeur des individus les moins évalués nulle afin de leur donner une chance d'être sélectionné, évoluer et éventuellement s'améliorer.* Elle permet d'obtenir une sélection difforme et variée tout en privilégiant les gènes les plus efficaces sans ignorer les autres.

Note : La méthode de sélection peut différer en fonction du résultat escompté. On privilégiera dans ce projet la méthode de **sélection aléatoire pondérée**.

1.1.2 Croisement

Parmi la population sélectionnée par la phase précédente, l'algorithme effectue un croisement de gène. Celui-ci peut se faire entre deux individus ou plus (principalement 2). Cette phase permet de produire une population (dont les individus sont principalement bien évalués) dont les gènes sont croisés et diversifiés par rapport à la population de départ. *La persistance des individus dont les performances sont faibles ou moyennes leur permet de s'améliorer grâce aux individus les mieux évalués.*

Note : Usuellement, la population d'entrée est supprimée et remplacée par la population de sortie. Il faut donc que cette phase produise pour des croisements entre **n** individus, **n** individus croisés afin de garder une population au même nombre que celle d'entrée.

Les méthodes de croisement d'individus existent sous plusieurs formes :

1. **Croisement aléatoire simple** : Chaque élément (ou partie) du gène des **n** individus sont parcourus afin de produire un individu. L'algorithme choisit aléatoirement l'élément (ou partie) de l'individu A, B, etc. Cela produit alors un nouveau gène mêlant les éléments (ou parties) de l'individu A, B, etc. Cette opération est effectuée **n** fois afin de produire une population de quantité équivalente.

2. **Croisement aléatoire hiérarchisé (ou pondéré)** : On considère que chaque type de gène dispose d'un coefficient (ou **valeur sélective** à la manière d'un individu). Ce dernier est utilisé comme base afin de pondérer le choix entre les gènes des individus croisés. Un gène à coefficient élevé a des chances plus élevées d'être sélectionné. Il en existe deux formes :
 - **Statique** : Ici, les gènes disposent d'une hiérarchie (dont les valeurs sont fixées arbitrairement au départ de la simulation). Chaque élément du gène des individus possède un coefficient qui identifie son niveau par rapport à un autre. Par exemple, la hiérarchie $G > B > N$ (dont les coefficients sont $3 > 2 > 1$) où $B=2$ est supérieur à $N=1$ mais inférieur à $G=3$. De la même manière qu'une méthode de sélection aléatoire pondérée, l'algorithme choisi un élément aléatoirement entre l'individu A et B où l'élément dont la valeur hiérarchique est supérieure a plus de chance d'être sélectionnée.
 - **Dynamique** : Cette méthode reprend le fonctionnement de la méthode de croisement aléatoire pondéré statique mais dans laquelle la hiérarchie des gènes n'est pas statique mais dynamique. C'est-à-dire qu'elle évolue à chaque génération et chaque population en fonction des gènes de cette dernière. Ainsi, une population dans laquelle le gène G est dominant verra sa hiérarchie telle que $G > \dots$. Bien entendu, il est possible d'effectuer un *décalage* entre une génération n et une génération $n+k$ tel que : $h(n)$ de $p(n)$ est utilisée sur $p(n+1)$ où h (hiérarchie), p (population), n (numéro de génération) et k (nombre de génération de décalage). Le dynamisme d'une hiérarchie peut se décliner sous deux formes :
 - **Dynamisme général** : Une population possède une hiérarchie commune. Celle-ci est recalculée au début de la phase de croisement en fonction de la quantité du gène au sein de la population totale ou sélectionnée.
 - **Dynamisme unitaire** : Pour une population avec un gène de k élément, il existe k hiérarchie commune recalculée au début de la phase de croisement pour chaque kème élément e du gène G (noté $e[k]$) qui dispose d'une hiérarchie de l'élément k notée $h(e[k])$ dont le dynamisme évolue en fonction de la quantité des gènes de la population du kème élément. *Soit la hiérarchie $h(e[1])$, la hiérarchie de l'élément 1.*

1.1.3 Mutation

La population résultant de la phase de croisement est alors soumise à la phase de mutation. Durant cette dernière, chaque élément (ou partie) de leur gène a la possibilité de muter (*Les chances de mutation sont par usage très faibles dans les alentours de 0.05%*) en un autre gène déterminé aléatoirement. Cette phase a pour but de donner une chance d'évolution et de diversification à la population.

2 Rapport technique

2.1 Conceptualisation

Afin d'obtenir les résultats escomptés par l'objectif de ce projet, il est opportun de réaliser un exercice. Avant de produire un algorithme génétique généralisant les cas d'utilisation d'une évolution, nous pouvons nous consacrer à la réalisation d'un cas concret et spécifique afin de l'utiliser comme base du cas général. Comme cas concret, nous prendrons le suivant :

Environnement : Nous disposons d'un plateau de $W \times W$ cases. Chaque case peut contenir *Pièce* ou pas (*Ces pièces sont au nombre $nbPièce$ réparties aléatoirement sur le plateau*). D'une position initiale X du pion *Pion* et d'un entier n .

Problème : *Quel est l'enchaînement de mouvement/pas M à partir de la position X qui permet de récupérer le plus de *Pièce* avec n pas ?*

Mouvement : Un mouvement M peut être **Haut**, **Bas**, **Gauche**, **Droite** relatif à la position actuelle de *Pion* et à l'axe du plateau. Un mouvement M est valide tant qu'il ne fait pas sortir le pion en dehors des limites du plateau.

Individu : Chaque *individu* de la population contiendra une solution/gène G : une suite de n caractère(s) chacun représentant un mouvement/pas de type M .

Evaluation : Chaque *individu* sera évalué et obtiendra un capital d'évaluation ou **valeur sélective** qui sera calculée en fonction du nombre de *Pièce* qu'il aura récupéré, de la distance parcourue (nombre de pas). Au cours du développement, la question des enchaînements de mouvement menant dans les murs du plateau. Trois hypothèses sont envisageables :

- Le sanctionnement par un score -infini. *Qui aura pour effet de rendre les individus ayant les dits enchaînements comme des individus ayant des chances d'être sélectionnées nulles. Néanmoins, comme précisé plus haut dans ce document, il est opportun de conserver tout individu comme conserver le potentiel évolutif de la population.*
- Le sanctionnement par un score -1. *Qui aura pour effet de rendre les individus concernés moins efficaces. De plus, ils ne seront pas écartés de la sélection.*
- Aucun sanctionnement. *En effet, selon notre méthode d'évaluation, le score des individus est retiré du **nombre de pas**. Ceci est en soit une sanction, étant donné qu'un mouvement dans un mur équivaut à un pas perdu que l'individu n'utilisera pas pour obtenir une pièce.*

Sélection : Une sélection est faite à chaque génération. La sélection récupère 30% de la population de façon aléatoire, plus un individu est mieux noté plus il aura de chance d'être sélectionné.

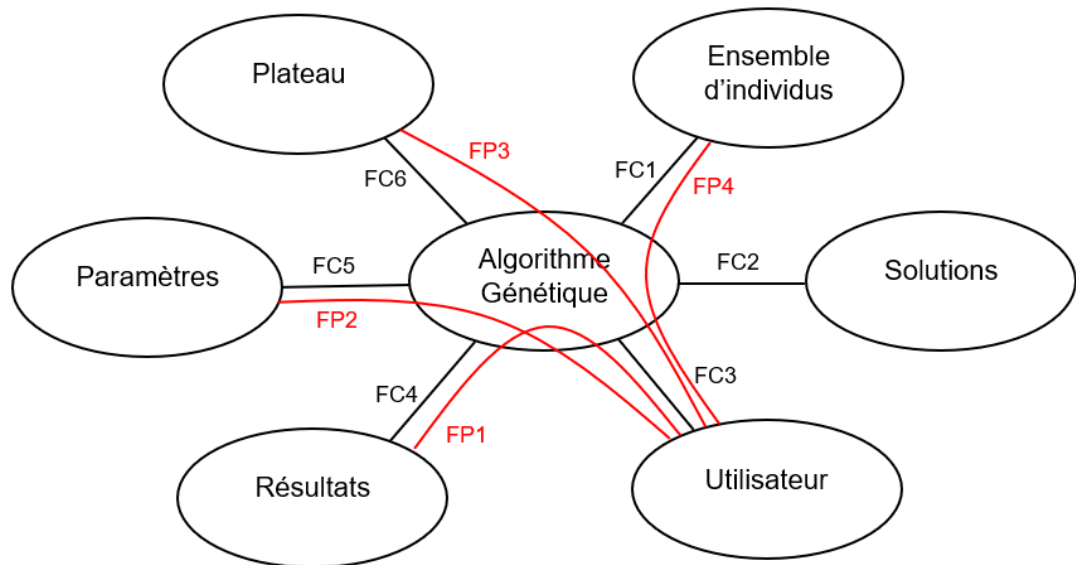
Croisement : Lors d'un croisement de deux individus, deux nouveaux individus fils

sont créés avec les mouvements de leurs parents. Les mouvements sont choisis un par un de façon aléatoire entre les deux parents. Les mouvements peuvent être choisis deux par deux, trois par trois, etc ...

Mutation : Chaque *individu* de la population a une chance de 5% de muter `mutate_chance` (*ce taux peut être modifié par l'utilisateur*). Ce taux est appliqué à chacun de ces mouvements de sa solution et cela transforme ce mouvement en un autre mouvement de façon aléatoire.

2.2 Analyse Fonctionnelle

Afin de mieux visualiser les fonctionnalités du programme. Nous avons réalisé un diagramme pieuvre qui permet d'illustrer les fonctions contraintes (FC) et les fonctions principales (FP) du programme. Ainsi que de les définir par une phrase.



FP1 : L'utilisateur doit pouvoir avoir accès aux résultats stockés à la fin d'une simulation.

FP2 : L'utilisateur doit pouvoir modifier les paramètres de la simulation.

FP3 : L'utilisateur doit pouvoir donner un plateau.

FP4 : L'utilisateur doit pouvoir donner un ensemble d'individus à tester.

FC1 : Récupérer un ensemble d'individus.

FC2 : Retourner le ou les individu(s) solution(s).

FC3 : Afficher graphiquement le ou les individu(s) solution(s) à l'utilisateur.

FC4 : Stocker le ou les individu(s) solution(s) dans un fichier texte nommé en fonction de la date et de l'heure.

FC5 : Modifier les paramètres de la simulation.

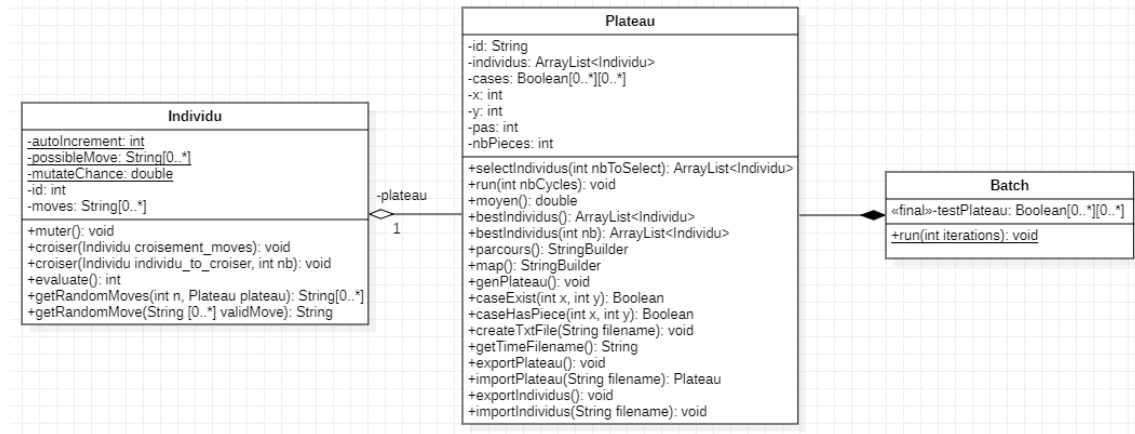
FC6 : Récupérer un plateau.

2.3 Architecture du code

L'architecture du code se base sur la logique orientée objet du langage Java qui sera utilisé pour le développement de l'algorithme génétique. La relation entre les objets **Individu** avec leur **Plateau** se base sur un modèle de responsabilité **MEDIATOR**. La classe **Plateau** englobant les **Individus** dans une liste de manière individuelle et les rend indépendants les uns des autres. Au niveau des modèles de construction, **Individus** se base sur un modèle **BUILDER**, sa construction ayant lieu au niveau du **Plateau** et ce dernier étant instancié et construit dans la classe **Batch** (ou celle de l'utilisateur).

- **Plateau**;
- Et **Individu**.

Les relations des classes peuvent être exprimées avec le schéma suivant :



Au début du projet, il avait été jugé opportun de créer une classe intermédiaire qui se chargerait du contrôle de chaque étape de vérification : **Mouvement**. Mais après réflexion, la création de cette classe rendait la classe **Individu** inutile, étant donné que presque toutes, si ce n'est toutes, les méthodes de **Individu** appelaient les méthodes de **Mouvement**. Il est donc préférable de se tenir aux deux classes ci-dessus pour le moment.

2.4 Manuel d'utilisation

Cette production a été développée et testée dans un environnement Windows 10 sous Java 11 avec l'application IntelliJ IDEA. Aucun autre environnement n'a encore été testé. Pour exécuter cet algorithme, compilez les classes contenues dans le dossier `src/main/java/fr/montpellier/iut` puis exécutez la classe `Main`. Un utilitaire d'exécution de simulation s'affichera, il ne vous restera plus qu'à suivre les instructions.

3 Gestion de projet

3.1 Démarche personnelle

3.1.1 Méthode de développement

Nous avons opté pour la méthode agile qui se déroule sur une période bouclée d’une semaine. A la fin de chaque sprint, nous effectuons une réunion avec notre tuteur afin de discuter de nos avancées, des éventuelles modifications et améliorations à faire et de la suite du travail. Cette méthode de travail correspond à notre contexte de développement où nous développons dans le but d’observer un résultat plutôt que d’essayer d’obtenir un résultat. Ainsi, nous pouvons, afin nos observations, faire des modifications et réfléchir sur l’algorithme afin de l’améliorer.

3.1.2 Gestion du code

Qu’il s’agisse de la phase de versionning ou de travail collaboratif, plusieurs plateformes ont été utilisées afin de permettre un partage efficace du code. En outre, nous avons utilisé, utilisons ou avons tenté d’utiliser des applications telles que :

- **Github** (principalement) ;
- **Floobits** (plateforme de codage collaboratif en temps réel) ;

Néanmoins, après avoir essayé la plateforme *Floobits*, nous avons observé que cette méthode de travail et de fonctionnement n’était pas adapté à notre méthode de fonctionnement et n’incluait pas un suivi d’historique des changements qu’offrait Github.

Nous avons utilisé également un package implémenté dans Java dans notre code ou afin de tester l’application : *Junit* pour produire des tests unitaires et vérifier le fonctionnement de nos méthodes et fonctions.

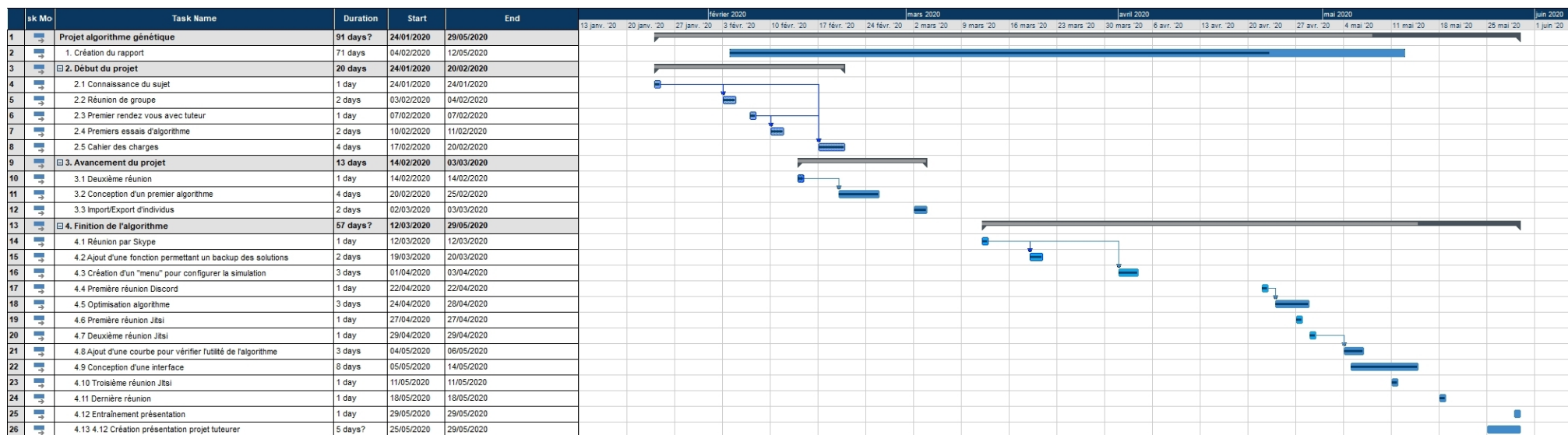
Au départ du projet, nous avons débuté le code en créant des dossiers séparés au sein du dépôt afin de pouvoir créer une version de notre projet de manière individuelle. Après quelques semaines, nous avons rapidement conclu qu’il s’agissait d’une mauvaise idée, car menant à de trop grandes différences et un travail de synchronisation supplémentaire qui aurait pu être évité. Après cette conclusion, nous avons donc refactorisé le code dans une version unique et commune sur laquelle chaque membre aura des tâches à faire spécifiques, en profitant de cette refactorisation pour mettre en commun nos idées.

Également afin d’obtenir un résultat visuel et attrayant pour notre interface nous avons utilisé les packages *JFrame* (Pour la gestion des fenêtres) et *Jpanel* (Afin de dessiner les solutions).

3.1.3 Gestion des taches

Utilisation du logiciel de cartographie mentale et de gestion de projet **MindView** afin de répartir efficacement les taches tout le long du projet.

Ci-dessous le diagramme GANTT permettant de visualiser les différentes étapes de notre projet :



4 Conclusion

Nous avons réussi à produire une application basée sur le concept d’algorithmie génétique afin de tenter de résoudre le cas concret décrit ci-dessus. Malheureusement, les résultats obtenus sont loin d’être ceux escomptés. En outre, nous obtenions un résultat qui rendait l’utilisation d’un algorithme génétique peu pertinent par rapport à une résolution algorithmique conventionnelle voir même une résolution à la main.

Le très grand nombre de paramètres modifiables et de phases influants sur le résultat de la simulation peut être la cause de ces mauvais résultats.

En effet, le résultat de la simulation pouvait différer drastiquement avec une modification mineure des variables (*phases et paramètres*) montrant une très haute sensibilité du résultat par rapport à ces dernières. Nous avons sous-estimé la recherche des paramètres optimaux et n’avons par conséquent pas attribué assez de temps à cette partie.

Néanmoins, nous estimons que ce projet et cette expérience nous a permis d’acquérir des connaissances et des compétences clés de l’algorithmie génétique. Non seulement afin de développer un esprit d’observation et de réflexion aux différents aspects inhérents à ce concept, mais également nous a mis en face de la difficulté à obtenir un résultat satisfaisant escompté au début du projet.

Nous avons également pu abordé la programmation d’une interface graphique, chose que nous n’avons pas pu voir durant notre formation. Ce projet nous a permis de mettre en pratique les compétences de programmation orientée objet (*structures de données : HashSet* pour l’optimisation) et surtout d’avoir une première expérience concrète avec les logiciels de versionning (Github) et de collaboration (Floobits).

5 Bibliographie

Il est possible de coupler cet algorithme à un réseau de neurones artificiels ou ↗ [Neural Network](#) basé sur les idées du psychologue ↗ [Franck Rosenblatt](#). Notamment pour le calcul des poids de neurones.

Quelques lectures peuvent être trouvées à ce sujet dont (en anglais) :

- ↗ ["Deep Learning with Darwin: Evolutionary Synthesis of Deep Neural Networks"](#) de M. J. Shafiee, A. Mishra et A. Wong ;
- ↗ ["Evolutionary Robot Behaviors Based on Natural Selection and Neural Network"](#) dans *Artificial Intelligence Applications and Innovations* de M. Bramer et V. Devdizic ;
- ↗ ["Evolving Neural Networks"](#) et ↗ ["A biology-inspired neural network evolving through natural selection"](#) de Souvik Das ;
- Ou encore, un jeu de simulation ↗ ["The Bibites"](#) développé par Léo Caussan dont le développement peut être suivi depuis sa chaîne ↗ [YouTube](#).