Université de
VersaillesSaintQuentinenYvelines

UFR Des Sciences

Algorithme Génétique

Cahier du compte rendu

Dirigé par : KLOUL Leila

Auteurs : AIT SLIMANE Rachid, ANTHENE Nicolas, BRAHIMI Lounes, DIA Mouhamadou Moustapha, DJAMA Samy, HAMENNI Koceila, OKETOKOUN Iqbal, WALSH Mathieu

20 mai 2020

Table des matières

1 Introduction :

Les algorithmes génétiques, initiés dans les années 1970 par John Holland sont des procédés fondés sur des principes et des faits biologiques, principalement le Darwinisme illustrant la survie des espèces. Fortement utilisés dans la résolution de problèmes d'optimisation, l'algorithme commence avec une population composée d'individus représentant des solutions potentiels à notre problème. Ces derniers seront évalués et ce verrons attribué un indice d'adaptation relatif à l'environnement. Par la suite des opérations génétiques leurs seront appliqué « sélection, croisement et mutation ». On recommence alors ce cycle jusqu'à ce qu'une solution satisfaisante soit atteinte.

Afin de consolider nos connaissances acquises durant notre parcours licence informatique à l'uvsq, nous avons choisi pour projet de fin de cycle de réaliser un algorithme génétique générique supervisé par madame Leila Kloul.

Notre algorithme se veut générique, l'application fournira à l'utilisateur les outils nécessaires pour la modélisation de plusieurs problèmes. Notamment pour justifier son efficacité nous avons décidé de faire deux exemples à contexte différents.

Notre application permettra tout d'abord à l'utilisateur de modéliser plusieurs problèmes suivant les données qu'il aura entré ou chargé depuis un fichier sauvegarder au préalable. Notre algorithme tournera alors sur ces informations pour sortir la meilleure solution au problème qui pourra ensuite être sauvegarder dans un fichier de sorti.

Au-delà de cela, nous avons ajouté deux exemples à notre application qui nous permettrons d'illustré l'utilisation de l'algorithme génétique dans différents problèmes : le problème du voyageur de commerce et le problème des huit dames.

Le problème du voyageur de commerce consiste à passer par un ensemble de villes en minimisant la distance du trajet.

Le but du problème des huit dames quant à lui est de placer les reines sur un échiquier sans que celles-ci ne puissent se menacer mutuellement d'après les règles du jeu d'échecs.

L'algorithme génétique propose une recherche d'une solution pour ces problèmes, nettement plus optimal qu'une recherche naïve.

Notre application sera donc codée dans le langage C++ car l'aspect objet est utilisé dans plusieurs facette de notre application par exemple pour l'implémentation des individus, de la population et d'une partie procédurale pour effectuer des opérations de calcul sur ceux-ci.

2 Explication de l'architecture du produit :

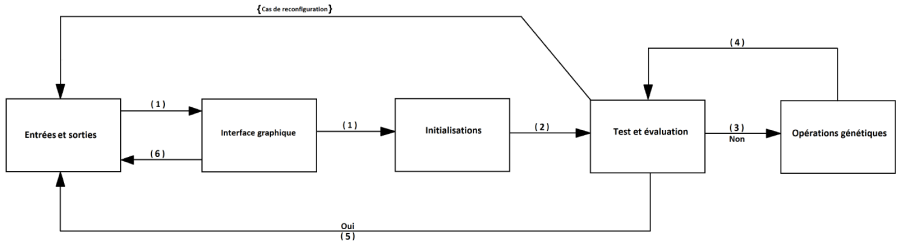


Figure 1 – Organigramme

L'étude des algorithmes génétiques nous a permis de produire cet organigramme composé de 5 principaux modules. Il permet de visualiser les différentes étapes de l'algorithme génétique dans le but de résoudre des problèmes d'optimisation avec une approche évolutionniste.

Tout d'abord, nous avons le module Entrée sortie qui regroupe les données entrées par l'utilisateur, nécessaires pour la configuration initiale, la modélisation du problème, et également le

chargement ou la sauvegarde d'une configuration pour une utilisation ultérieur. Il est relié au module Interface graphique permettant à l'utilisateur d'entré les différentes informations. Ce dernier interagi avec l'utilisateur en montrant le déroulement de l'algorithme génétique grâce à la modélisation d'un problème ou encore, le choix d'un exemple prédéfini. Les informations recueillis sont ensuite transmises au module Initialisation permettant l'interprétation des données entrée par l'utilisateur pour la création de la population initiale comportant les différents individus et leurs gènes respectifs initialisés aléatoirement. Le module Test et évaluation quant à lui, permettra d'interpréter sur les différents gènes, une équation que l'utilisateur aura entrée afin de vérifier si la population testée est satisfaisante et arrêter l'exécution de notre algorithme. Enfin, nous avons le module Opération Génétique qui regroupe toutes les principales opérations importantes pour tourner notre algorithme génétique à savoir la sélection, le croisement et la mutation. Tous ces modules seront coordonnés grâce à l'interface graphique implémenté grâce à la bibliothèque QT

3 Description du fonctionnement :

Cette section sera consacrée à l'étude du fonctionnement de notre application. Pour cela , des points techniques seront abordé puis nous proposerons des exemples pour illustré le fonctionnement de notre application.

3.1 Fonctionnement des modules :

3.1.1 Entrée Sortie :

Ce module se charge de tout ce qui touche à l'écriture des données. Il reçoit les données du module interface graphique ou d'un fichier de configuration sauvegardé au préalable. Une fois que les informations entrées ont été vérifiées, elles sont envoyées au module Initialisation pour la configuration et la modélisation du problème. Il permet également, à par le chargement de fichier, de sauvegarder dans un fichier Latex une configuration pour une utilisation ultérieur mais aussi de générer un fichier de sorti en Latex décrivant l'évolution des individus de génération en génération.

3.1.2 Interface Graphique :

Ce module permettra de visualiser le déroulement de l'algorithme génétique depuis la création de la population principale jusqu'à ces générations futures. Il permettra à l'utilisateur de modéliser un problème grâce aux données (taille de la population initiale, type des gènes, nombre de gènes, un type de sélection) qu'il pourra entrer dans les différents champs disponibles, ou de choisir l'exécution de l'algorithme sur un exemple prédéfini. L'utilisateur aura également la possibilité d'entrée l'équation pour l'évaluation, un intervalle pour la valeur des gènes, la condition d'arrêt du test, un nombre maximum d'itérations, le nombre d'individus à sélectionnés, un taux de croisement et un taux de mutation. Il permettra également l'arrêt de la simulation pour soit produire et afficher les statistiques donnant un aperçu correspondant à l'évolution des générations, ou encore une reconfiguration des entrées etc. Nous avons choisi d'utiliser la bibliothèque Qt pour l'implémenter car elle est simple à utiliser, générique et elle donne beaucoup de liberté pour son utilisation.

3.1.3 Initialisation :

Ce module traduit les informations entrées par l'utilisateur pour constituer la population initiale. Il se chargera également de mettre à la disposition des autres modules les outils nécessaires pour l'exécution de l'algorithme génétique sur les différents problèmes que l'utilisateur voudra résoudre. Il se compose de deux principales classes à savoir : **Individu** et **Gene** contenant tous les gènes constituant un individu. Ces deux classes seront utilisée pour la création à part entière d'un objet « individu » qui sera membre de la population.

3.1.4 Test et évaluation :

Symbole	signification	Exemple
a	gène 1	
b	gène 2	
c	gène 3	
d	gène 4	
w	gène 5	
x	gène 6	
y	gène 7	
z	gène 8	
+	Addition	a+b
-	Soustraction/chiffre négatif	a-b ou bien (-a)
*	Multiplication	a*b
/	Division	a/b
%	Modulo	a%b
(	parenthèse ouvrante	
)	parenthèse fermante	
>	entier supérieur le plus proche	>(a)
<	entier inférieur le plus proche	<(b)
^	puissance	x^y
fcos	Cosinus	fcos(a)
fcosh	Cosinus hyperbolique	fcosh(b)
facos	Arc cosinus	facos(x)
fsin	Sinus	fsin(z)
fsinh	Sinus hyperbolique	fsinh(a)
fasin	Arc sinus	fasin(b)
ftan	Tangente	ftan(x)
ftanh	Tangente hyperbolique	ftanh(z)
fatan	Arc tangente	fatan(y)
fln	Logarithme népérien	fln(a)
flog	log2	flog(b)
flogx	log10	flogx(c)
fr	racine carré	fr(x)
frc	racine cubique	frc(y)
fva	valeur absolue	fva(-a)
fe	exponentielle	fe(b)
&	ET logique	a&b
|	Ou logique	x|y
fx|	xor	afx|b
f~	not	f~a
f~|	nor	xf~|y
f~&	nand	af~&b

FIGURE 1 – Listes des symboles utilisés pour l'évaluation

Ce module permet d'analyser la structure des données entrer par l'utilisateur et de traduire sur les différents gènes, l'équation qu'il aura entrée. A chaque individu sera attribué une note (indice) correspondant à sa capacité d'adaptation dans le problème à résoudre. L'algorithme testera à chaque itération si la génération satisfaisante est atteinte. Si c'est le cas, la simulation s'arrêtera et affichera les statistiques corresponds aux résultats, sinon, l'algorithme continuera son exécution jusqu'à ce que le nombre maximal d'itération soit atteint. On notera également l'utilisation de la classe individu à ce niveau car un individu pourrait être évalué plus d'une fois avec à chaque fois une modification de la valeur de son indice.

3.1.5 Opération Génétique :

Ce module contient toutes les principales opérations constituant l'algorithme génétique. Il est composé de : La sélection consistant à choisir parmi les 03 différents types de sélections proposés, les individus de la population qui deviendront les parents de la prochaine génération.

Le croisement, qui consiste à diviser la population sélectionnée en deux sous-populations. On appliquera alors sur chacun des individus de celles-ci la reproduction (enjambement) donnant finalement un couple de parents formé de deux nouveaux individus (enfants) dont leurs caractéristiques sont issues des parents des sous-populations.

La mutation consistant à tirer un ou plusieurs gènes d'un individu puis de le(s) modifié(s) aléatoirement.

Toute cette partie constitue essentiellement la partie modélisation du problème de notre projet. Plusieurs outils sont mis en place à ce niveau pour permettre à l'utilisateur de contrôler la simulation. Nous avons également décidé de montrer l'utilisation des algorithmes génétique grâce à deux exemples.

Comme premier exemple pour montrer la généricité de notre algorithme, nous avons choisi la résolution du **problème du voyageur de commerce** . Ce module possède une classe unique VoyageurDeCommerce qui est à la fois l'interface et l'algorithme traitant le problème.

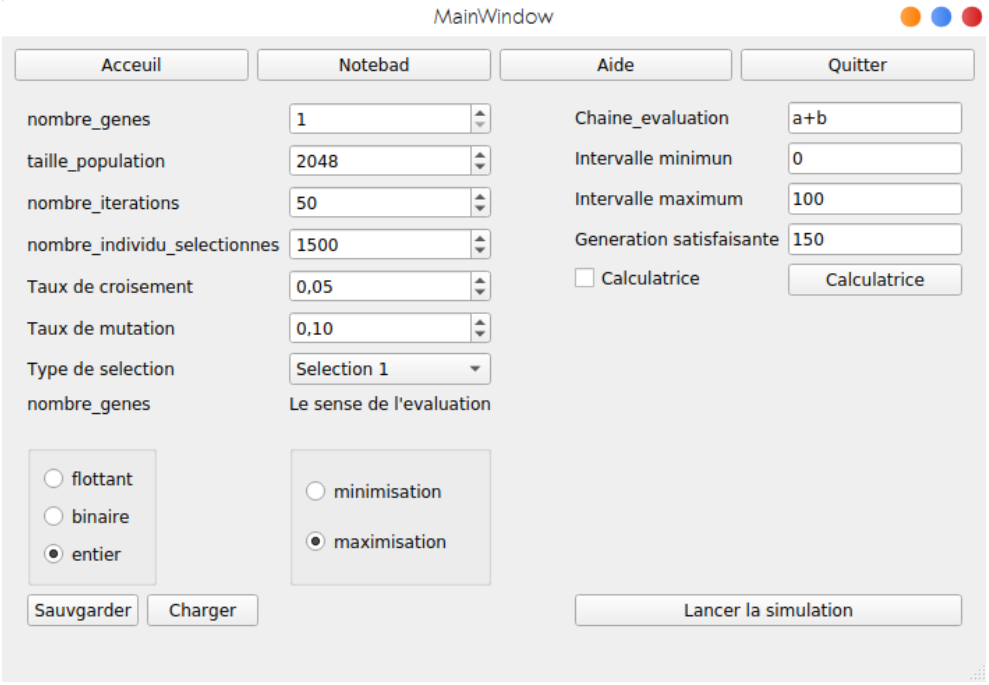
Notre deuxième exemple s'est porté sur **le problème des Huit Dames** qui consiste à trouver une combinaison de position pour 8 reines d'un jeu d'échecs. Pour ce faire, nous avons divisé le module en deux parties, la première représentant la partie algorithmique du problème (ProblemeDesHuitDames) et la seconde représentant ça partie graphique (BoxEchiquier, Echiquier, interfaceHuitDames, piece).

Notre algorithme étant générique, ces exemples utilisent tous deux les modules définis plus haut pour s'exécuter. Tous ces modules seront coordonnés grâce à l'interface graphique implémenté à l'aide bibliothèque QT.

3.2 Fonctionnement des différentes parties de l'application :

3.2.1 Modélisation du problème :

Cette partie permet essentiellement à l'utilisateur de modéliser un problème par les configurations qu'il pourra saisir dans les différents champs de l'interface grâce aux outils mis à sa disposition.

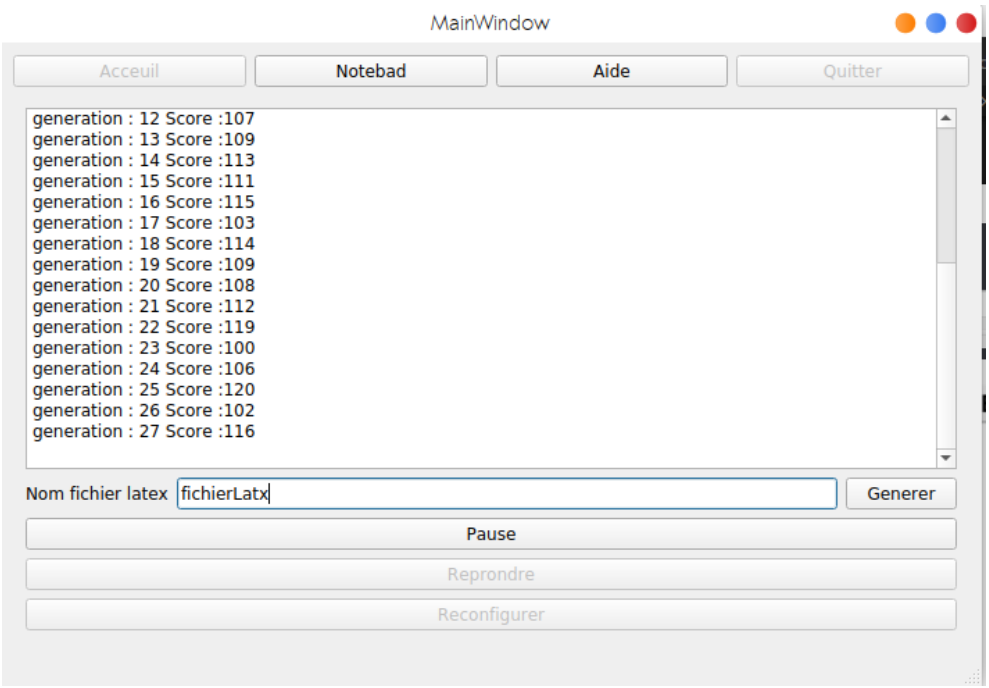


Les données entrées seront utilisées pour la création de la population initiale et l'exécution de l'algorithme sur le problème à résoudre. Une configuration par défaut est implémentée pour l'exécution. Si l'utilisateur souhaite toutefois la concevoir manuellement, il aura la possibilité de saisir une équation pour l'évaluation qui permettra de noter la capacité d'adaptation de nos individus dans le problème définit, la taille de la population(le nombre d'individu), le nombre d'itérations maximal(dans le cas où la génération satisfaisante n'est pas obtenue, cela nous évitera d'être enlacé dans une boucle infinie), le choix du type de sélection parmi les 3 proposées(sélection par rang, sélection par tournois et sélection par roulette), le nombre d'individus a sélectionné, le nombre de gènes limité à 8, le type des gènes(entier, flottant et binaire) accompagné d'un intervalle de valeur que pourront prendre les gènes, un taux pour le croisement, un autre pour la mutation, la note moyenne de la génération satisfaisante représentant la moyenne des scores indiquant que la génération actuelle répond à notre problème. Enfin il pourra indiquer si l'algorithme devra exécuter une maximisation (augmentation des notes d'évaluations) ou une minimisation (atténuation des notes d'évaluations). Pour l'équation d'évaluation, l'utilisateur aura la possibilité de la saisir dans un champs texte ou encore d'afficher une calculatrice pour faciliter ainsi ça définition en cliquant sur une icône symbolisant cette dernière. Selon le type de gènes choisit, la calculatrice affichée aura plus ou moins différentes fonctions par exemple, les opérations logiques ne seront affichées que pour le type binaire. Une fois toutes ces données récupérées, elles seront passé en paramètres au constructeur de la classe EntreesSorties, qui hébergera ces informations au sein de ses attributs pendant toute l'exécution de l'algorithme génétique. Il aura ainsi configuré toutes les données nécessaires à l'exécution de l'algorithme génétique.

L'utilisateur a également la possibilité d'utiliser une configuration par défaut déjà implémenté, mais aussi de charger une configuration précédemment enregistrée en cliquant sur le bouton "Charger" et en saisissant dans le champ de texte correspondant le nom du fichier contenant les données. Le constructeur à un seul paramètre de la classe EntreesSorties sera alors appelé dans ce cas en lui passant le nom du fichier, puis se chargera d'initialiser tous les attributs de la classe nécessaire pour le bon déroulement de l'algorithme. De même l'utilisateur pourra sauvegarder la configuration actuelle pour une utilisation ultérieure ou pour un archivage en cliquant sur le bouton "Sauvegarder" suivis d'un nommage pour le fichier. Maintenant que l'utilisateur à modéliser son problème, il pourra demander le commencement de l'exécution de l'algorithme génétique sur ce dernier en cliquant sur le bouton "Lancer la simulation". La demande d'exécution de l'algorithme génétique sur le problème modélisé donnera accès à une nouvelle interface, qui hébergera à son centre un écran affichant dynamiquement la moyenne du score des générations accompagnées des meilleurs

individus de celles-ci.

L'utilisateur aura également la possibilité de mettre en pause l'exécution à tout moment. Par cela, il pourra ainsi reprendre l'exécution à son étape d'arrêt ou encore reconfigurer totalement ou partiellement la configuration actuelle puis relancer l'exécution de l'algorithme. Grace à un champs de texte présent sur l'interface, l'utilisateur pourra générer un fichier Latex en entrant le nom corresponds et en cliquant sur le bouton Générer. L'utilisateur aura la possibilité de quitter l'application à travers n'importe laquelle de ces deux interfaces en cliquant sur le bouton "Quitter". Il également la pour option de revenir à l'accueil de notre application en avec un bouton "Accueil" présent au sein des deux interfaces. Une icône disponible sur la première interface, permettra d'afficher un bloc note sur lequel l'utilisateur pourra saisir ses remarques et ses observations.



3.2.2 Problème des Huit Dames :

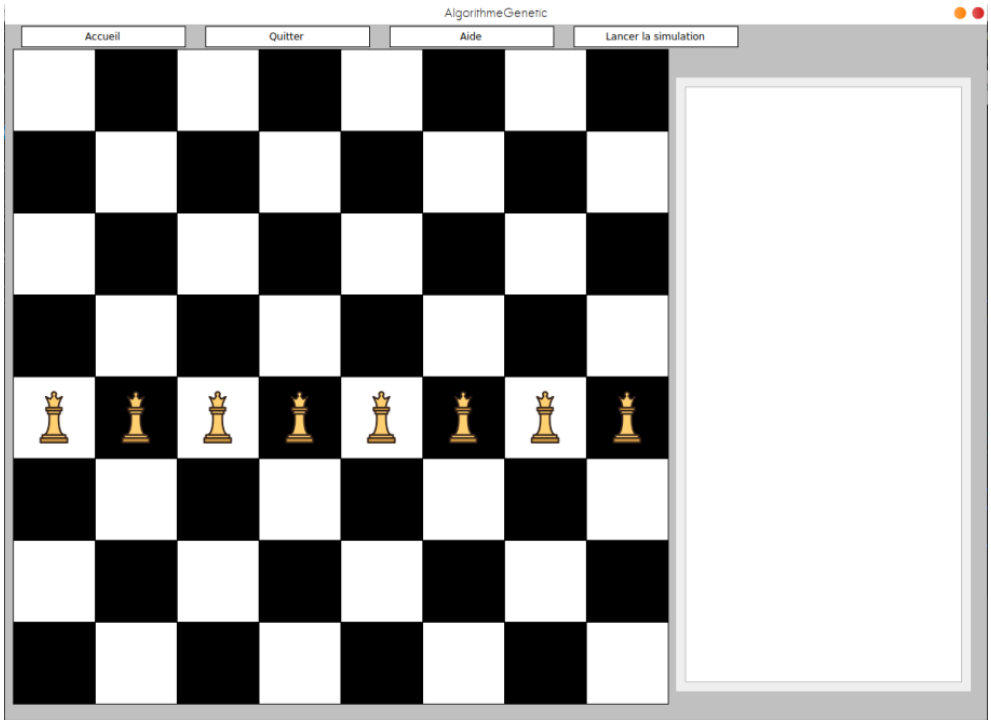
Pour illustrer l'utilisation de l'algorithme génétique, nous avons décidé d'implémenter un exemple. Notre choix s'est alors axé sur le problème des huit dames sur lequel nous avons testé la généricité de notre algorithme.

Le problème des huit dames, consiste en effet à trouver une combinaison de positions pour 8 reines d'un jeu d'échecs ne correspondant pas à la solution recherchée. Pour ce faire, la partie algorithmique a été modélisée par une classe comportant 1 constructeur et 3 méthodes. L'objectif de cette classe étant de proposer des méthodes de gestion d'échiquier et de mettre en forme le problème de manière à faciliter l'implémentation graphique.

Ainsi, la classe a pour objectif de renvoyer un tableau à deux dimensions echiquier[][] représentant l'échiquier. Ce tableau comporte un « 1 » si une dame est présente à la case de l'échiquier, 0 sinon.

Grâce à la méthode verification_solution on analyse l'échiquier afin de détecter si les dames présentent sur ce dernier se menacent entre elles. Une note est affectée à chaque reine (ici les individus) en fonction du nombre de menaces pesant sur elle. L'algorithme génétique utilisera par conséquent la minimisation afin de trouver une solution au problème. La méthode detection_position quant à elle permet de convertir les gênes de l'algorithme génétique, générées aléatoirement à la première itération, en position sur un échiquier. Elle initialise le tableau gene[][] représentant les positions des 8 dames sur l'échiquier. Enfin la méthode generation_echiquier initialise un tableau à 2 dimensions conformément à la description donnée plus haut.

Ainsi une fois que l'utilisateur aura cliquer sur le bouton "Lancer la simulation", l'exécution de notre algorithme génétique commencera et ce dernier procédera aux différentes opérations (sélection, croisement, mutation...) de génération en génération jusqu'à s'arrêter sur celle qui contient l'individu parfait. Autrement dit, la combinaison de gènes constituant huit positions pour les reines dans lesquelles elles ne se menaces pas mutuellement c'est à dire qu'elles ne se croisent pas. L'algorithme génétique affichera alors l'individu parfait sous forme graphique "huit dames de reines de couleur jaune posé sur un échiquier de couleur noir et blanc". L'utilisateur pourra répéter l'opération autant de fois qu'il le souhaite en cliquant sur le bouton "Lancer une nouvelle simulation" pour mettre ainsi en lumière, les différences présentes entres les exécutions, le numéro de la génération satisfaisante (l'affichage de l'individu parfait) tout en sachant que, d'une exécution à une autre les solutions peuvent être différentes. Il pourra aussi se diriger vers la page d'accueil en appuyant sur le bouton "Accueil" et pour exécuter d'autres fonctionnalité de notre application. L'utilisateur aura la possibilité de quitter à tout instant l'application en cliquant sur le bouton "Quitter".



3.2.3 Problème du voyageur de commerce :

Nous avons également utilisé l'algorithme génétique pour implémenter le problème du voyageur de commerce.

Le problème du « voyageur de commerce » a pour objectif de trouver le chemin avec le poids le plus faible passant par tous les sommets d'un graphe.

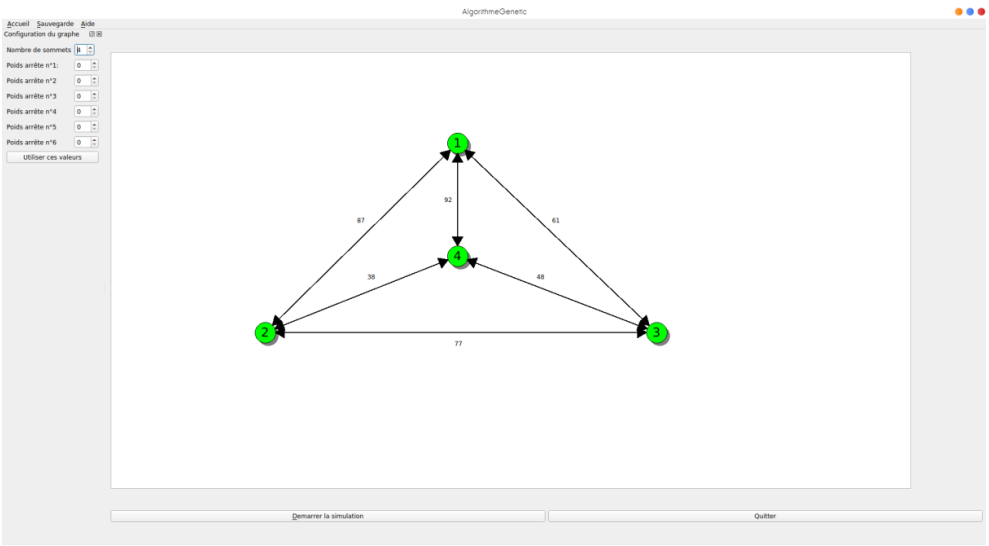
A ce niveau, l'utilisateur aura à choisir entre un graphe compris entre quatre et sept sommets. Ces derniers peuvent être généré aléatoirement à part le graphe à 4 sommets pour lequel l'utilisateur pourra saisir manuellement le poids des différentes arrêtes. Une fois le nombre de sommets (la taille du graphe) choisie par l'utilisateur, ce dernier devra cliquer sur le bouton "Lancer la simulation" pour qu'un graphe complet (pour un maximum de complexité) avec des poids aléatoires lui soit généré dans le cas où il aura saisi une configuration pour un graphe de taille 4, après quoi, le graphe correspondant lui sera conçu.

Une fois le graphe original afficher à l'utilisateur ce dernier devra cliquer sur le bouton "voyageur de commerce". Ainsi le plus court chemin sera calculé avec notre algorithme génétique et lui sera ensuite afficher. A ce stade l'utilisateur aura le choix de faire un deuxième calcule sur un graphe de la même taille, il n'aura dans ce cas qu'à cliquer sur le bouton "Lancer la simulation". Les poids

des arêtes étant générés aléatoirement, deux exécutions d'un même graphe successives attribueront des poids différents aux arrêtes.

Dans le cas où l'utilisateur souhaite configurer manuellement les poids des arêtes du graphe à 4 sommets, 6 champs correspondants aux arêtes lui seront afficher prêts à recevoir leurs poids respectifs.

L'utilisateur aura également le choix de changer le nombre de sommets constituant le graphe afin d'aboutir à la meilleure solution, en cliquant sur le bouton "Lancer la simulation", pour que le plus court chemin soit calculé puis l'affiché, en cliquant sur le bouton "voyageur de commerce". L'utilisateur pourra à tout moment quitter l'application en cliquant sur le bouton "Quitter". Un bouton donnant accès vers accueil de notre application est présent pour répondre à cette demande.



4 Description des points délicats de la programmation :

Durant la programmation, nous avons rencontré une difficulté liée à l'évaluation. En effet la priorité des différentes opérations nous a quelque peu donné du fil à retordre. Nous avons donc opté pour différentes fonctions dont on se sert comme des nœuds. Cela nous a donc permis une meilleure gestion de la priorité dans nos calculs.

D'autre part nous avons implémenté la sélection par roulette : celle-ci fonctionne mais nous avons remarqué que cette dernière n'était pas très efficace comparé aux deux autres méthodes de sélections.

Nous avons aussi rencontré un problème avec notre interface. Il y avait un décalage lors de l'affichage. Pour résoudre ce problème, nous avons donc utilisé un thread pour avoir un affichage immédiat.

5 Changements mineurs des spécifications :

Les spécifications ont été intégralement respectées au niveau de l'algorithme génétique (evaluation, EntreesSorties, individu, gene, operationsgenetiques).

Les modifications ont eu lieu plus au moins au niveau des 3 interfaces, pour la partie voyageur de commerce on à u besoin de créer de nouvelles classes pour affichage (une classe node représentant le nœud, une autre edge représentant les arêtes, graphWidgets conteneur de notre graphe et une classe interface regroupant tout les composants graphiques).

Pour la classe modélisation de problèmes, de nouvelles classes ont été créer pour des fines extensions et de design (en ajoutant par exemple un clavier virtuel pour la saisie de l'équation ou bien un bloc note pour la prise de notes).

6 Comparaison entre l'estimation et l'implémentation :

Les modules	Entré sortie	Opération génétique	Initialisation	Test et évaluation	Interface graphique	Total
Nombre de ligne prévue	1000	1000	400	600	5000	8000
Nombre de ligne final	350	450	200	1200	4400	6600
Temps prévue	10h	10h	4h	6h	50h	80h
Temps final	10h	15h	2h	16h	60h	103h

En comparant les estimations du cahier des charges au coup final de l'implémentation, on remarque des écarts de prévisions pour certains modules.

En effet certains modules nous ont demander plus de temps par rapports a des problèmes rencontrés lors de l'implémentation. Nous avons aussi consacré du temps supplémentaire pour améliorer notre code et facilité les futures améliorations que l'utilisateur pourrai vouloir faire.

On se retrouve donc avec un nombre de lignes de code plus bas pour l'ensemble des modules et 25 % de moins par rapport au nombre de lignes total.

7 Conclusion technique du produit et amélioration :

Notre application a été codé de façon à avoir une indépendance entre les différents modules. Chaque module agis alors uniquement sur lui-même et peut donc être modifié sans compromettre le bon déroulement des autres. De ce fait chacun d'entre eux a un potentiel d'amélioration ou de modification suivant les différents besoins de l'utilisateur.

Coté programmation, l'individu possède un nombre de gène infini a contrario de l'évaluation. Celle-ci a donc la flexibilité de pouvoir être modifié permettant ainsi l'ajout de plus de gène. De plus elle est implémentée de manière à pouvoir ajouter facilement des opérations ainsi que des types différents.

La partie « voyageur de commerce » permet de créer des graphs complets allant de 4 à 7 sommets et peut facilement être changer pour permettre l'ajout de sommets supplémentaires si besoin est. Il en est de même pour « le problème des huit dames » dans lequel il est possible d'augmenter le nombre de dames si voulu.

En résumer, l'Indépendance entre les différents modules permet d'ajouter de nouveaux croisements, mutations et divers autres paramètres de l'algorithme génétique.

8 Conclusion sur l'organisation interne du projet :

Durant notre dernier semestre de Licence informatique à Versailles, nous avons eu le choix entre plusieurs projets dans le module IN608 dirigé par madame Leila KLOUL. Celui choisi par notre équipe consistait à réaliser une application mettant en œuvre le principe de la généricité dans les algorithmes génétiques.

De ce fait, notre groupe a établi différents plannings pour permettre à tout un chacun de travailler afin d'obtenir finalement le résultat escompté. Pendant les premières semaines, nous faisions deux réunions en présentiel à la fac pour nous mettre d'accord sur le travail accomplis durant la semaine, discuter des prochaines tâches et le reste du temps on restait en contact sur discord afin de se partager les différentes informations. Nous avions également un groupe de travail sur WhatsApp sur lequel nous fixions des rendez-vous pour se réunir. Par la suite, toute notre organisation a été modifié suite à l'épidémie du Covid19 survenue lors du deuxième mois. Ceci a grandement

compliqué la situation de tout un chacun car ils nous étaient plus difficile de communiquer et encore moins de se retrouver. Tout au long de cette période nous avions donc continuer à travailler en ligne en utilisant surtout WhatsApp et discord pour évoluer sur le travail à rendre.

Nous avons longuement débattu et sommes arrivé à ce logiciel offrant la possibilité à l'utilisateur de modéliser son problème ou de sélectionner un parmi deux exemples prédéfini. Le but principal de notre projet étant de montrer l'utilité des algorithmes génétiques d'une part pour la complexité et d'autre part pour le temps de résolution. Malgré la répartition des différentes tâches, nous avons pris du retard au niveau de certaines étapes notamment lors de la remise des spécifications et aussi de la programmation. Cependant le travail effectuer nous a permis de produire une application simple d'utilisation et efficace dans la résolution de problèmes mais aussi de rendre notre travail avant le temps imparti.

Nous sommes sortis grandis de cette expérience car nous avons gagné en compétences grâce à ce projet. Nous avons également appris en termes de gestion de projet et l'importance d'une bonne cohésion de groupe. Nous avons acquis des compétences techniques et organisationnelles qui nous permettront d'éviter à l'avenir de reproduire ce genre d'erreurs.