Présentation des algorithmes génétiques et du contrôle dynamique des opérateurs

Alexandre ROBIN
Université d'Angers, Master 2 Recherche Intelligence Décisionnelle

***Résumé*—Durant ces dernières années, de nombreux travaux dans divers domaines ont pris de l'importance concernant les algorithmes génétiques. Ceux-ci peuvent se révéler très utiles pour des problèmes NP-Complet à condition de choisir les bons paramètres et opérateurs permettant aussi bien l'exploration que l'intensification. Nous allons voir dans cet article les mécanismes de base de ces algorithmes puis le contrôle dynamique des différents opérateurs intervenants. Différentes méthodes de contrôle seront testées pour un problème simple qui est celui du One-Max, puis nous adapterons notre algorithme à un problème concret tel que celui du Voyageur de Commerce (TSP).**

I. Introduction

Les algorithmes génétiques (**AG**s) sont des algorithmes reproduisant le comportement naturel d'évolution. En effet, ceux-ci sont fondés sur des mécanismes de base présents dans la nature tels que la sélection naturelle ainsi que des changements génétiques. Le principe est simple, une population représente les solutions potentielles que nous pourrions avoir, un individu est une des ces solutions et ces derniers vont évoluer au cours du temps pour apporter de nouvelles solutions. Les individus seront évalués en fonction du problème donné par ce qui est appelé la fitness et qui correspond à sa performance sur ce problème.

Différentes étapes sont nécessaires au bon fonctionnement d'un AG. Tout d'abord, une étape d'initialisation de la population est nécessaire soit de manière aléatoire soit grâce à un précédent algorithme de recherche locale, par exemple, pour nous permettre de commencer avec des résultats déjà plus satisfaisants. Une première phase de sélection permet de choisir différents individus dans la population courante sur lesquels seront réalisées les opérations suivantes. Vient l'étape du croisement où deux parents préalablement sélectionnés laisseront place à deux nouveaux enfants. Enfin, des mutations génétiques seront réalisées sur les enfants avant que ceux-ci soient insérés à la population. Ce cycle est ensuite répété jusqu'à obtenir une solution satisfaisante.

Les AGs sont utilisés dans un grand nombres d'applications dès lors qu'une méthode de résolution exacte est impossible à appliquer à un problème. En effet, la simplicité de mise en application et leurs mécanismes les rendent adaptables à différents types de problèmes pour lesquels aucune heuristique n'est disponible.

Une des caractéristique des AGs est le nombre d'opérateurs applicables possibles. Le choix des meilleurs opérateurs pour la résolution d'un problème devient donc difficile et ce choix influence énormément la qualité de l'AG. Une méthode consiste donc à faire en sorte que notre algorithme puisse apprendre de lui-même le meilleur opérateur à appliquer en fonction de la population et du problème.

Nous allons dans cet article voir tout d'abord une présentation générale des AGs avec le codage des individus ainsi que des différents opérateurs de sélection, croisement, mutation et d'insertion pour ensuite voir le contrôle de ces opérateurs grâce à un apprentissage automatisé. Par la suite, nous mettrons en place ces algorithmes pour deux problèmes, celui du One-Max qui est un problème relativement simple qui consiste à obtenir une chaîne de bits à 1, puis le Voyageur de Commerce aussi appelé TSP.

II. Présentation des algorithmes génétiques

Un algorithme génétique est défini selon différents points :

- Un individu qui représente une solution potentielle au problème donné ;
- Une population qui est l'ensemble des individus s'inscrivant dans l'espace de recherche ;
- Une fonction de fitness qui nous permet pour chaque individu d'évaluer sa performance par rapport au problème ;
- Des opérateurs nous permettant de faire évoluer nos individus dans la population.

A. *Individu et Population*

Dans un AG, chaque problème aura ses individus souvent différents de ceux d'un autre problème. En effet, le type de codage des individus doit être adapté pour permettre une cohésion totale avec le problème et pouvoir donc trouver une solution à celui-ci. Deux cas bien connus sont le codage binaire où un individu va se révéler être une suite de bits telle que pour une longueur d'individu n, alors nous avons un individu

$$I = \{I_1, ..., I_i, ..., I_n\}, \forall i \in [1, n], I_i \in \{0, 1\}.$$

Dans le cas du codage réel, nous avons de même

$$I = \{I_1, ..., I_i, ..., I_n\}, \forall i \in [1, n], I_i \in \mathbb{R}.$$

En fonction du problème, chaque gêne de l'individu pourra être unique dans le cas d'un codage réel pour le problème du voyageur de commerce, par exemple, ou plusieurs fois identique comme avec la coloration de graphe. Une des premières étapes cruciales d'un AG est donc de choisir le codage adapté au problème avant de permettre une mise en œuvre simple et efficace du problème.

Une population se révèle être constituée de différents individus qui évolueront entre eux. Cette population sera de taille fixe du début à la fin de l'AG. Plusieurs initialisations des individus d'une population sont possibles : une recherche locale réalisée au préalable pour insérer des individus "bons" dès le départ, un tirage aléatoire des valeurs des individus...

B. Fonction de fitness

La fitness d'un individu correspond à son efficacité face au problème et exprimé en valeur positive réelle. Elle sera calculée grâce à la fonction de fitness d'un individu qui sera adapté à chaque problème. Pour le problème du One-Max, elle se révèle être relativement simple à calculer :

$$\sum_{i=1}^{n} I_i, I_i \in \{0, 1\}.$$

Dans le problème du voyageur de commerce que nous développerons en exemple, la fonction de fitness sera la somme de toutes les distances entre les villes reliées deux à deux. Pour cela, un parcours de la matrice des distances sera nécessaire contrairement au One-Max qui lui n'est basé que sur la composition de l'individu.

En revanche, pour certains problèmes, cette fonction peut-être plus complexe et peut même intégrer des pénalités. Dans un exemple tel que la coloration de graphe, deux choix sont possibles. Soit un graphe doit toujours être valide et deux sommets adjacents doivent donc disposer de couleurs différentes, soit un graphe peut être invalide mais une fonction de pénalité est donc ajoutée à la fonction de fitness pour permettre une valorisation des graphes valides tout en essayant de minimiser le nombre de couleurs aussi appelé nombre chromatique.

C. Opérateurs

Après la phase d'initialisation, un AG va reposer sur différentes phases qui seront répétées jusqu'au critère d'arrêt choisi :

- Sélection
- Croisement
- Mutation
- Insertion

*1) **Sélection**:* Différents opérateurs de sélection existent tel que le fait de choisir les k individus ayant la meilleure fitness. L'inconvénient de cette méthode est une convergence trop rapide vers un individu qui n'est pas forcément l'optimum global et cela peut engendrer une perte de diversité dans la population. Une alternative possible à ce problème en considérant toujours la performance des individus et la sélection dite par tournoi. Celle-ci tire au hasard deux individus dans la population et les compare entre eux, le meilleur est choisi pour accéder à la phase de croisement. Une sélection complètement aléatoire pourrait elle aussi être choisie. Cette procédure de sélection peut-être réalisée autant de fois que l'on souhaite sélectionner des individus. La sélection reste très similaire d'un problème à un autre du fait de l'utilisation d'une fonction de fitness pour n'importe quel AG. Pour le moment aucun des individus de la population n'a évolué. En effet, seuls le croisement et la mutation selon une probabilité de s'effectuer p peuvent induire des changements dans nos individus. Plus cette probabilité sera élevée et plus le croisement ou la mutation auront une chance d'être réalisés.

*2) **Croisement**:* L'opérateur de croisement lui va permettre de créer deux nouveaux individus souvent appelés enfants à partir de deux individus sélectionnés au préalable, les parents. De même que pour la sélection, beaucoup de croisements peuvent être réalisés, nous verrons ici seulement les croisements utilisés pour le codage binaire et utilisés pour le One-Max et nous reparlerons de ceux définis pour le codage réel dans notre application au TSP. Étant représenté par une chaîne de bits, le croisement est un algorithme simple. Une valeur aléatoire est choisie entre 1 et $n-1$ pour couper notre suite de bits de nos deux parents. Les enfants seront ensuite construits en intervertissant entre le premier et le deuxième parent et réciproquement pour obtenir deux nouveaux enfants.
Une autre méthode appelée uniforme consiste à choisir pour chaque bit de chaque enfant le bit correspondant d'un des deux parents avec une probabilité de 50%. Cette étape de croisement nécessaire au bon fonctionnement de l'AG ne pourra en revanche nous permettre d'atteindre certaines solutions. Prenons l'exemple du One-Max où le but est d'avoir une série de bits entièrement à 1. Le croisement ne pourra pas permettre de mettre un bit à 1 si aucun individu de la population ne possède ce même bit à 1. C'est pour cela que des opérateurs de mutations sont nécessaires.

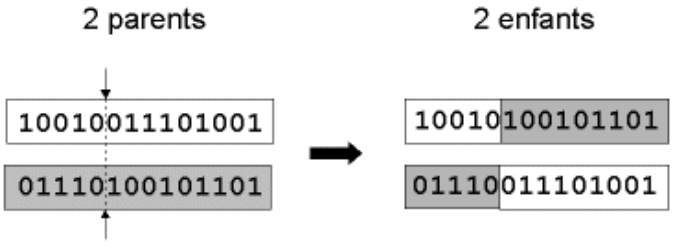

FIGURE 1. Croisement un point en codage binaire

*3) **Mutation**:* Comme nous venons de le voir, le rôle des opérateurs de mutation est essentiel pour rechercher dans tout l'espace et atteindre des individus que nous n'aurions jamais connus avec le croisement, et reposent pourtant sur un procédé simple. Pour un codage binaire, une inversion de k bits peut-être réalisée sur un individu ou une inversion de probabilité $1/n$ sur chaque bit. Lors d'un codage réel, un tirage aléatoire d'une nouvelle valeur peut-être effectué dans les points de l'espace mais si chaque gène doit être différent alors des complications peuvent intervenir et nous verrons comment les gérer dans notre exemple du TSP.

*4) **Insertion**:* L'insertion quant à elle s'inspire de la sélection. En effet nous pouvons retrouver une insertion en fonction de la fitness des individus présents dans la population et remplacer les plus mauvais. Des insertions basées sur l'âge des individus sont aussi trouvées dans la littérature.

De nombreux opérateurs existent et peuvent être imaginés en fonction des problèmes mais comment déterminer quels sont les opérateurs les plus performants en fonction des instances du problème. Sur un AG "basique", un opérateur de chaque type sera sélectionné au lancement et ceux-ci seront utilisés durant tout le long de l'algorithme. L'AG doit donc être lancé un grand nombre de fois pour pouvoir déterminer le meilleur opérateur.
Un autre processus que nous allons voir ici consiste à contrôler dynamiquement et donc tout au long des différentes générations de notre population les opérateurs utilisés.

III. Contrôle dynamique des opérateurs

Le contrôle dynamique se réalise grâce à la sélection d'un opérateur à un moment t dans un ensemble d'opérateurs implémentés. Ces opérateurs seront sélectionnés grâces à différentes méthodes que nous verrons dans les sous parties suivantes.
Avant cela, les différents algorithmes ont besoin de différents paramètres afin de pouvoir permettre de sélectionner l'opérateur permettant la meilleure amélioration de fitness d'un individu. Une récompense est donc par la suite attribuée à chaque opérateur choisi.
Par la suite, nous considérerons un ensemble d'opérateurs $O = \{o_1, ..., o_n\}$, avec pour chaque opérateur i une récompense associée $r(o_i, t)$ à une itération t.

A. *Principe de récompense (reward) et utilité*

La récompense, aussi appelée le gain d'un opérateur, se trouve être souvent spécifique à un problème donné. En effet, celle-ci va se révéler ici être directement associée à la fitness d'un individu. Ici nous chercherons à obtenir une récompense normalisée et nous obtenons :

$$r(o_i, t) = f(t-1) - f(t)/F_{max}$$

avec $f(t)$ la fitness du meilleur individu de la sous-population sélectionnée au préalable par l'opérateur de sélection à l'itération t et F_{max} la meilleure récompense obtenue depuis le début de l'AG. Ces différentes récompenses seront accumulées au fil des itérations lorsque l'opérateur aura été sélectionné de telle sorte que nous disposons pour chaque opérateur $i \in n$ d'une moyenne de ses récompenses à une itération T de la forme

$$m_i = \frac{1}{T} * \sum_{t=1}^{T} r(o_i, t), r(o_i, t) \in [0, 1].$$

Pour un opérateur i non-sélectionné, sa récompense sera $r(o_i, t) = 0$.

Maintenant que nous avons donné un sens au principe de récompense, voyons le calcul de l'utilité à proprement parler de notre opérateur. En effet, cette récompense ne sera pas directement utilisée dans nos algorithmes de décision mais permettra de calculer ce qui est appelé utilité dans la littérature. Pour cette dernière, un coefficient α sera nécessaire pour permettre de réaliser une balance entre l'utilité immédiate $u_i(t)$ à l'itération t pour un opérateur i, et celle de l'itération précédente $u_i(t-1)$. Nous obtenons alors grâce à notre moyenne des récompenses m_i vu précédemment :

$$u_i(t) = (1 - \alpha) * u_i(t-1) + \alpha * m_i$$

Chaque opérateur verra son utilité recalculée à chaque itération même si celui-ci n'a pas été choisi à cette itération t.

B. *Mécanismes de sélection d'opérateurs*

Maintenant que nous avons pour chaque opérateur $o_i \in O = \{o_1, ..., o_n\}$ une récompense ainsi qu'une utilité, nous allons voir comment ces valeurs peuvent nous permettre de sélectionner un opérateur. Différentes méthodes existent et nous en présenterons ici trois. Deux d'elles, la roulette proportionnelle ainsi que l'adaptive pursuit calculent une probabilité p_i d'être choisi pour chaque opérateur $o_i, \forall i \in \{1, ..., n\}$. Celle dite de l'UCB pour Upper Confidence Bound utilise directement l'utilité calculée et se base comme son nom l'indique sur la confiance de la limite supérieure que nous verrons en dernier point.

*1) **Sélection par Adaptive Roulette Wheel (Roulette Proportionnelle Adaptative)**:* Une roulette proportionnelle fixe, que nous avons décidé de ne pas implémenter ici, serait de choisir une probabilité fixe pour chaque opérateur, qui ne varierai donc pas tout au long de l'AG. Ces valeurs pourraient être déterminées après avoir lancé un grand nombre de fois l'AG en testant les différents opérateurs de O. Un des problèmes majeurs de ce genre de choix est qu'un opérateur est considéré comme ayant une même "utilité" tout au long de l'AG alors que celle-ci peut être excellente au début pour à la fin tendre vers 0. C'est donc là que nous pouvons commencer à prendre en compte une version dite adaptative

de la roulette proportionnelle.

Une nouvelle probabilité sera donc calculée pour tous les opérateurs une fois celui-ci appliqué, et donc sa nouvelle récompense déterminée. Dans le but de permettre une sélection à ceux possédants une moyenne de leurs récompenses faible, un paramètre p_{min} compris entre $[0, \frac{1}{n}]$ sera introduit pour assurer une probabilité minimale à chaque opérateur et donc un minimum d'exploration. Les probabilités calculées des opérateurs pour l'itération $t+1$ étant donné les nouvelles utilités calculées à l'itération t après avoir appliqué un des opérateurs sera donc de la forme :

$$p_i(t+1) = p_{min} + (1 - n * p_{min}) * \frac{u_i(t)}{\sum_{k=1}^{n} u_k(t)}, \forall u_i, i \in \{1, ..., n\}$$

Le i-ème opérateur est maintenant choisi proportionnellement à sa probabilité. Un "mauvais" opérateur devrait donc être sélectionné avec une probabilité p_{min} alors que le meilleur devrait l'être avec une probabilité $p_{max} = 1 - (n-1) * p_{min}$. En réalité, nous verrons en pratique que chaque opérateur ayant un minimum de pertinence dans l'amélioration d'un individu continuera d'être sélectionné aussi ce qui diminuera considérablement les performances de cette méthode. Nous pouvons donc voir qu'ici tous les opérateurs se verront dotés de la même fonction de calcul de probabilités ce qui va nous amener à notre seconde méthode où l'opérateur le plus performant se verra attribuer une fonction à part.

*2) **Sélection par Adaptive Pursuit**:* En effet, ici l'opérateur ayant la meilleure utilité que nous noterons i^* sera distingué des autres pour être mis en valeur. Nous aurons donc pour une itération t :

$$i^* = \arg\max_{i=1}^{n} u_i(t)$$

$$\begin{cases} p_{i^*}(t+1) = p_{i^*}(t) + \beta * (p_{max} - p_{i^*}(t)) \\ p_i(t+1) = p_i(t) + \beta * (p_{min} - p_i(t)) \end{cases}$$

avec un nouveau paramètre β permettant d'ajuster le poids de cette stratégie du "winner-take-all".

L'adaptive pursuit ne prend donc jamais en compte la valeur réelle de l'utilité des opérateurs mais se contente de savoir laquelle est la meilleure contrairement à la roulette proportionnelle adaptative. En revanche, les deux méthodes sont contrôlées par un paramètre p_{min} permettant l'exploration des différents opérateurs o_i de O ainsi que α dans le calcul de l'utilité qui permet de faire une balance entre passé et présent. L'adaptive pursuit possède en plus un taux d'apprentissage permettant de valoriser le meilleur opérateur que nous venons de voir appelé β.

*3) **Sélection par UCB (Upper Confidence Bound)**:* L'UCB, contrairement aux précédentes techniques de sélection, ne va pas calculer différentes probabilités. En effet, cette méthode va sélectionner l'opérateur possédant la meilleure limite supérieure de l'intervalle de confiance et celui qui maximise l'utilité tout au long des itérations. Nous allons donc retrouver ici l'utilité d'un opérateur $u_i(t)$ à une itération t ainsi qu'un intervalle de confiance qui dépend du nombre de fois $n_i(t)$ où l'opérateur o_i à été choisi. L'opérateur choisi sera donc celui retourné par :

$$\arg\max_{i=1}^{n} \left(u_i(t) + \sqrt{\frac{2 * \log \sum_{k=1}^{n} n_k(t)}{n_i(t)}} \right)$$

Nous constatons ici que le membre gauche de notre formule permet de favoriser le "meilleur" opérateur depuis le début et donc l'exploitation contrairement à l'exploration réalisé par le terme de droite qui favorise les opérateurs les moins essayés. Nous verrons qu'en pratique, après avoir itéré plusieurs fois et choisi semblablement toujours le même opérateur, l'UCB à tendance à avoir besoin de beaucoup de temps pour réaliser qu'un nouvel opérateur se révèle être meilleur.

C. *Fenêtre glissante*

Dans le but de permettre une meilleure exploration des différents opérateurs, une fenêtre dite glissante peut-être introduite dans notre AG. En effet, lorsqu'un opérateur se révèle être le meilleur durant un nombre d'itérations important, son utilité est donc fortement augmentée ce qui le favorise pour les prochaines itérations. En revanche, pour certains problèmes, le meilleur opérateur théorique changera au fur et à mesure des itérations.

Avec par exemple le problème du One-Max que nous allons voir par la suite et 3 opérateurs de mutation, l'inversion 5-bits, 3-bits et 1-bit, le premier à sélectionner serait le 5-bits étant donné que c'est celui-ci qui nous permettra au début de changer le plus de bits possible. Cependant, après un certain nombre d'itérations, et un individu où le nombre de bit à 1 est supérieur au nombre de bits à 0, nous devrions choisir une inversion 3-bits puis une inversion 1-bit au moment où très peu de bits restent à inverser.
En pratique, nous nous apercevrons que le 5-bits est le premier opérateur choisi mais que dû à sa forte augmentation de fitness des individus, et donc de la moyenne de ses récompenses importantes depuis le début de l'AG comparée aux autres opérateurs, ces derniers pourront donc difficilement augmenter à leur tour leurs moyennes pour être choisis par la suite.

Un nouveau paramètre entre en jeu : une taille de fenêtre qui se retrouvera fixe tout au long de l'AG. Ces fenêtres w_i vont stocker les différentes valeurs des récompenses obtenues pour chaque opérateur $r(o_i, t)$. Pour une taille de fenêtre maximum $w_{sizeMax} = 100$, nous aurons donc les 100 dernières récompenses de chaque opérateur de la forme

$$w_i = [r(o_i, t_1), ..., r(o_i, t_{100})], \forall i \in \{1, ..., n\}$$

.

Étant donné que toutes les valeurs ne sont plus prises en compte dans le calcul de la moyenne des récompenses, nous ne pénaliserons plus un opérateur o_i lorsque celui-ci ne sera pas sélectionné et ne lui attribuerons donc plus de valeur de récompense à ces itérations. Les fenêtres vont donc augmenter jusqu'à leur taille maximale à une différente vitesse en fonction de la sélection des opérateurs. Lorsqu'elles se retrouvent remplies, le processus de remplissage recommence en ne remplaçant à chaque fois que la plus vieille valeur de récompense.

De ce fait, le calcul de la moyenne pour chaque opérateur o_i va devenir

$$m_i = \frac{1}{w_{i_{size}}} * \sum_{j=1}^{w_{i_{size}}} w_{i_j}, w_{i_j} \in [0, 1]$$

Avec l'exemple du One-Max pris précédemment, si notre inversion 5-bits se retrouve ne plus être le meilleur opérateur, alors ses récompenses vont tendre vers 0. Avec la fenêtre glissante et notre nouveau calcul de moyenne des récompenses m_i, le calcul de l'utilité va donc être plus révélateur du fait qu'il sera basé sur un calcul au plus proche de la solution actuelle.

Nous allons donc maintenant voir comment ces différents algorithmes participent à la recherche d'une solution la plus optimale possible dans une population.

IV. Expérimentations

A. *Problème du One-Max*

Comme nous avons pu le voir précédemment, le problème du One-Max est relativement simple. Ce problème est codé par des individus sous forme de bits. La taille d'un individu sera le nombre de bits dont il possède et lors de l'initialisation, tous les bits de chaque individu sera fixé à 0. Le but sera de parvenir à un individu possédant tous ses bits à 1. Lorsqu'un individu aura sa fitness égale à sa taille alors nous saurons que notre individu est composé seulement de 1.

Notre algorithme possédera quatre opérateurs de mutation vu précédemment, une inversion de 1, 3 ou 5 bits que nous appellerons k-flips avec $k \in \{1, 3, 5\}$ ainsi que l'inversion dite bit-flips où chaque bit est inversé avec une probabilité de $1/n$ avec n la taille de l'individu. De même, nous possédons 3 opérateurs de croisement qui représentent le découpage des parents en 1 point, 2 points et le croisement uniforme.

Nous pouvons avoir une première intuition consistant à se dire qu'étant donné que nos individus ne sont composés que de 0 à l'initialisation de la population, alors nous devons le plus rapidement possible changer des gènes à 1 pour permettre une augmentation de la fitness. L'opérateur de mutation 5-flips est donc idéal au début de l'algorithme.

Prenons l'exemple d'une population composée de 30 individus qui possèdent 1000 bits à inverser. La fitness maximum est donc égale à 1000.
Comparons l'évolution de la population au cours des générations en utilisant une mutation 5-flips ou 1-flip.

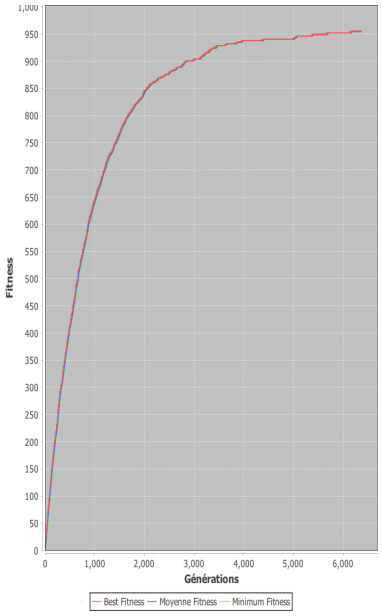

FIGURE 2. Évolution de la fitness des individus en fonction des générations avec utilisation du 5-flips

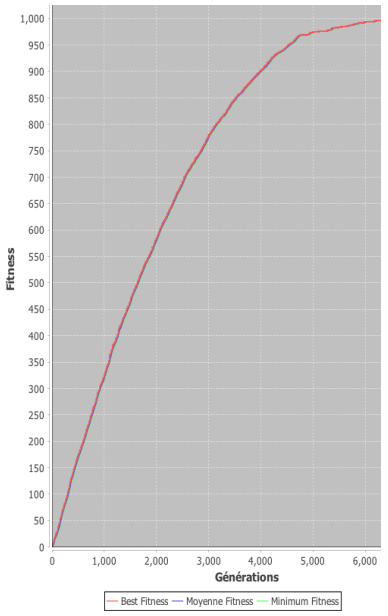

FIGURE 3. Évolution de la fitness des individus en fonction des générations avec utilisation du 1-flip

Nous pouvons voir sur la figure 1 qui utilise le 5-flips une fitness d'environ 850 à la génération 2000 alors que lorsque le 1-flip est utilisé, nous percevons une fitness de l'ordre de 580. En revanche, lorsque l'individu possède une fitness relativement élevée, nous voyons que le 5-flips devient difficilement utilisable du fait que certains bits à 1 seront mis à 0 contrairement au 1-flip qui lui essayera de transformer seulement un bit qui est encore à 0. Une seconde intuition serait donc de dire que le 5-flips doit être utilisé au début, puis le 3-flips pour enfin finir avec le 1-flip.
Nous comprenons donc ici l'utilité du contrôle dynamique des opérateurs et plus particulièrement pour les opérateurs de mutation.

Dans ce papier, nous ne développerons pas le travail de recherche et d'expérimentation effectué pour trouver les différentes valeurs des paramètres utilisés dans la sélection des opérateurs. En effet, ces valeurs doivent être étalonnées "manuellement" en testant un grand nombre de fois l'algorithme utilisé. Après cela, une valeur pratique pourra être ressorti pour être par la suite utilisée. Voici donc les valeurs qui seront utilisées :

α	0.3
p_{min}	0.1
β	0.3
$taille_{Fenetre}$	100

Nous allons maintenant avoir une vue comparative des différentes méthodes avec le résultats de celles-ci.

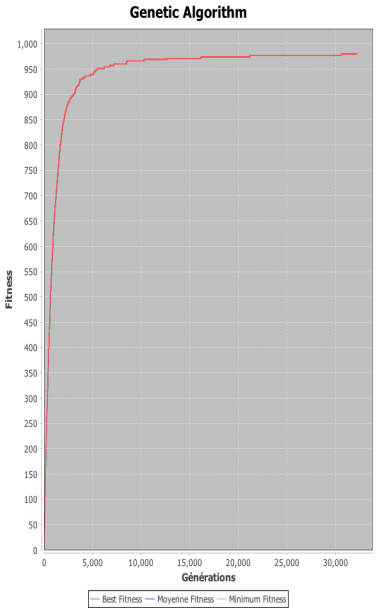

FIGURE 4. Évolution de la fitness des individus en fonction des générations avec utilisation du 5-flips

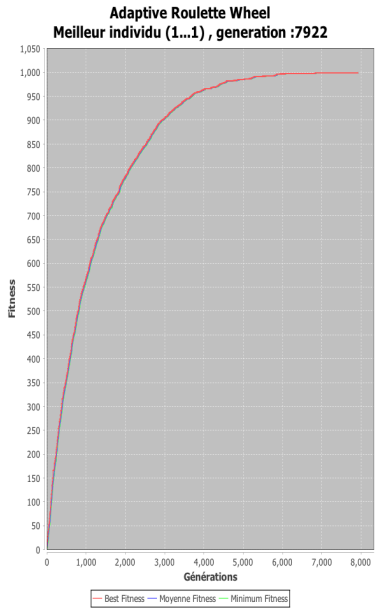

FIGURE 5. Évolution de la fitness des individus en fonction des générations avec l'Adaptive Roulette Wheel

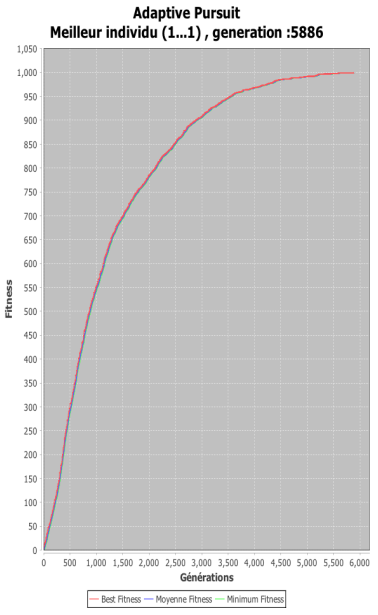

FIGURE 6. Évolution de la fitness des individus en fonction des générations avec l'Adaptive Pursuit

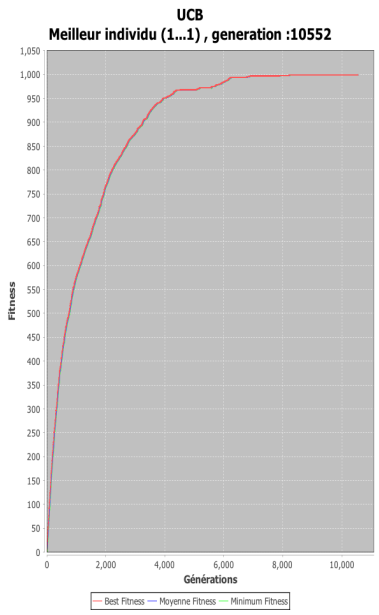

FIGURE 7. Évolution de la fitness des individus en fonction des générations avec l'UCB

Tout d'abord nous pouvons simplement constater qu'un algorithme génétique dit "basique" (Figure 4) peut se révéler efficace mais est dans l'impossibilité d'atteindre la valeur maximale. En effet, dû aux nombres de bits changés avec le 5-flips, l'algorithme ne progresse que très peu à la fin.

Un grand nombre de tests devrait être donc effectué pour permettre de déterminer les meilleurs opérateurs à un moment donné t.

L'utilisation de l'Adaptive Roulette Wheel nous permet d'obtenir de bons résultats dû à une probabilité minimum p_{min} qui nous permet d'explorer tous les opérateurs tout au long de l'algorithme. Nous sommes donc sûr que nos meilleurs opérateurs seront choisis à certains moments.

En revanche, comme nous l'avions évoqué dans la partie théorique, l'UCB se révèle être plutôt mauvais sur des problèmes de ce genre. En effet, comme nous pouvons le voir Figure 9, cet algorithme étant basé partiellement sur le nombre de fois où un opérateur à été choisi, lorsque la fitness ne peut qu'évoluer très lentement, alors tous les opérateurs sont choisis au fur et à mesure de telle sorte qu'ils le soient tous autant de fois. Nous perdons donc l'information que nous avions obtenue auparavant et chaque opérateur se retrouve avec une probabilité quasi-équitable.

Ce phénomène ne se retrouve pas pour l'Adaptive Pursuit (Figure 8). En effet, cette méthode possède le même atout que l'Adaptive Roulette Wheel, p_{min} qui permet une fois de plus l'exploration de tous les opérateurs. De plus, dû à sa distinction du meilleur opérateur et donc à son exploration d'opérateurs supérieurs à l'Adaptive Roulette, l'Adaptive Pursuit se trouve être nettement plus performante. Une amélioration de 45% et 26% respectivement par rapport à l'UCB et l'Adaptive Roulette.

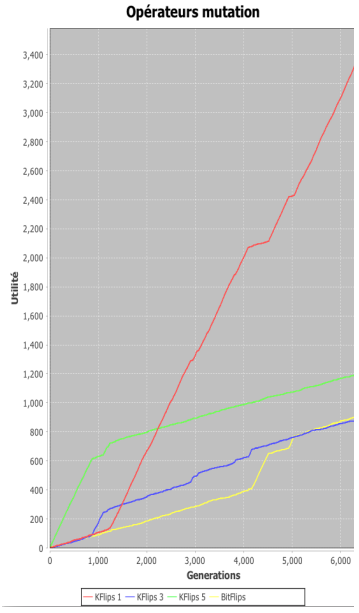

FIGURE 8. Nombre de fois où a été choisi un opérateur en fonction des générations avec l'Adaptive Pursuit

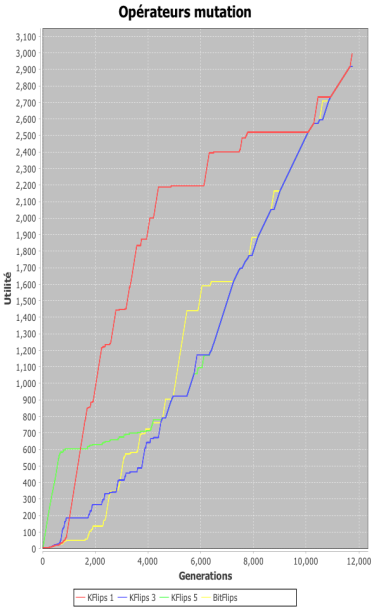

FIGURE 9. Nombre de fois où a été choisi un opérateur en fonction des générations avec l'UCB

B. *Problème du TSP*

Dans le problème du voyageur de commerce, TSP, nos individus seront représentés par une liste de ville ou l'ordre sera l'ordre dans lequel le "voyage" se déroule ainsi qu'une matrice de distance nous permettant de connaître l'éloignement de deux villes. Cette matrice va être utile dans le calcul de la fitness qui sera représenté par la somme des distances entre deux villes pour toutes les villes existantes et donc dans la liste.

Les opérateurs de sélection et d'insertion vont se retrouver être les mêmes que dans l'algorithme du One-Max mis à part que le meilleur individu aura maintenant la fitness la plus petite possible dû à la recherche d'un parcours le plus court possible, une minimisation.
En revanche, nous avons implémenté de tous nouveaux opérateurs pour la mutation et le croisement. Voyons quelques uns de ces opérateurs.

Pour les opérateurs de mutation, nous retrouvons deux types. Tout d'abord celui basé sur des sous-listes de la liste de villes telle que l'inversion d'une sous-liste où le mélange aléatoire dans une sous-liste. Une autre mutation consiste à déplacer une sous-liste et l'insérer juste après une ville choisie aléatoirement. D'un autre côté, certaines mutations sont basées sur l'échange de villes, deux villes sont tirées et celles-ci sont inversées.

De-même que pour les mutations, des croisements propres aux problèmes du TSP et bien connus de la littérature ont été implémentés. Par exemple, le croisement ordonné (Order Crossover) qui choisi une sous-liste du premier parent pour l'insérer au même niveau dans le premier enfant. Une sous-liste choisie au même endroit que dans le premier parent mais prise du second parent, et cette fois-ci, insérée au second enfant. Le parent n'ayant "injecté" aucune de ces propriétés à un enfant comble ensuite en commençant par le deuxième point de croisement et en omettant les éléments déjà présents. Un autre opérateur appelé Edge Recombination Crossover (ERX) utilise une liste "d'adjacences" pour construire des enfants qui héritent des informations des parents. Cette liste stocke toutes les relations entre nœuds contenus dans les deux parents, qu'il s'agisse d'arcs entrants ou sortants dans une ville. Donc, chaque ville aura au minimum deux relations d'adjacences.

Nous n'allons pas exposer ici les deux autres opérateurs de croisements utilisés appelés PMX pour Partially Mapped Crossover et CX, Cycle Crossover, mais nous devons garder en tête que lors de la construction d'un algorithme génétique pour un problème tel que le TSP, il faut, lors de croisements, être capable de délivrer des individus qui ne comportent qu'une seule fois chaque ville. De plus, les sous-listes utilisées lors de la mutation ou du croisement, sont fixées à une longueur de 2, 3 ou 5 villes.

Nous allons maintenant présenter quelques résultats d'expérimentations obtenus avec la sélection dites de l'Adaptive Pursuit ainsi que les valeurs minimales connues. Nous présentons ici seulement ces résultats car comme dans le One-Max, nous trouvons une disparité des valeurs obtenues avec chaque algorithme.

Instances	Nos meilleurs résultats	Meilleurs résultats connues
wi29	27734	27603
dj38	7044	6656
berlin52	7966	7542
qa194	12469	9352

V. Conclusion

Comme nous avons pu le constater dans le tableau précèdent, nous obtenons des résultats plutôt satisfaisants par rapport aux meilleurs résultats connus dans la littérature. En revanche, lorsque nous avons testé notre algorithme sur des instances possédant beaucoup de villes, nous nous sommes rapidement aperçu que notre conception ne le permettait pas. En effet, une itération prend beaucoup plus de temps et notre algorithme ne peut trouver efficacement une solution. Une parallélisation du code aurait pû être envisagée mais n'était pas le but de cette recherche.

De plus, une autre méthode de sélection d'opérateurs qui pourra être envisagée dans un autre papier est le modèle en île. Cela consiste à grouper un nombre d'individus en "île" sachant que chaque île représente un opérateur. Lorsqu'un individu est dans une île, le même opérateur sera appliqué. Cependant, sur un même principe de récompense, les individus vont se déplacer entre les îles, plus une île sera peuplée et meilleur sera donc son opérateur.

Enfin, les algorithmes génétiques ont déjà démontré leurs forces dans la littérature et nous venons de voir que ceux-ci peuvent être considérablement améliorés, grâce à l'apprentissage par renforcement de différents algorithmes que nous venons de voir, et cela peut permettre d'obtenir une meilleure solution ou au moins l'obtenir plus rapidement que pour les algorithmes "traditionnels".

Références

[1] Dr.Mrs.G.Nirmala and Mr.D.Ramprasad. Innovative pmx - cx- operators for ga to tsp. 2012.

[2] Audibert Jean-Yves, Munos Rémi, and Szepesvari Csaba. Use of variance estimation in the multi-armed bandit problem. 2008.

[3] Haj-Rachid Mais, Bloch Christelle, Ramdane-Cherif Wahiba, and Chatonnay Pascal. Différents opérateurs évolutionnaires de permutation : sélections, croisements et mutations, 2010.

[4] Veerapen Nadarajen, Maturana Jorge, and Saubion Frédéric. *Sélection adaptative d'opérateurs pour la recherche locale basée sur un compromis exploration-exploitation*. PhD thesis, 2013.

[5] Kumar Naveen, Karambir, and Kumar Rajiv. A comparative analysis of pmx, cx and ox crossover operators for solving travelling salesman problem. 2012.

[6] Kumar Rakesh, Gopal Girdhar, and Kumar Rajesh. Novel crossover operator for genetic algorithm for permutation problems. 2013.

[7] Fialho Álvaro, Da Costa Luis, Schoenauer Marc, and Sebag Michèle. Analyzing bandit-based adaptive operator selection mechanisms. 2010.