Chafik AKMOUCHE
Université d'Angers – Février 2022
Algorithme génétique appliqué au problème OneMax

Résumé – Dans ce papier, nous avons implémenté un algorithme génétique avec plusieurs opérateurs de sélection, croisement, mutation, remplacement, etc. et nous l'avons appliqué au problème OneMax.

En testant les différents opérateurs, nous avons remarqué qu'il n'y a pas d'opérateur performant d'une manière absolue qu'un autre opérateur, mais tout dépend du problème traité, de sa taille mais surtout de l'instant où l'opérateur est utilisé. La problématique dressée est donc : quel opérateur et quand l'utiliser afin d'augmenter la fitness cumulée ? Et c'est ce que nous avons traité dans la deuxième partie de cet article, ensuite nous avons implémenté deux mécanismes d'apprentissage par renforcement *Probabilty Matching (PM)* et *Upper Confidence Bound (UCB)* et nous les avons injecté à l'algorithme génétique.

Enfin, nous avons présenté, analysé et discuté les résultats obtenus, et comparé aux résultats obtenus par l'algorithme génétique classique.

Mots clés : Algorithme évolutionnaire, Algorithme génétique, Problème OneMax, Problème du Bandit, PM, UCB.

Abstract – In this article, we implemented a genetic algorithm with several operators of selection, crossing, mutation, replacement... and we applied it to the OneMax problem.

By testing the different operators, we noticed that there is no absolute operator more efficient than another operator, but everything depends on the problem treated, its size but especially on the moment when the operator is used. The problem posed is therefore : which operator and when to use it to increase the cumulative fitness ? this is what we covered in the second part of this report, then we implemented two reinforcement learning mechanisms *Probabilty Matching (PM)* and *Upper Confidence Bound (UCB)* and injected them into the genetic algorithm.

Finally, we presented, analyzed and discussed the results obtained, and compared to the results obtained by the classical algorithm.

Keywords : Evolutionary algorithm, Genetic algorithm, OneMax problem, Bandit problem, PM, UCB.

1 Introduction

En l'absence d'une méthode de résolution exacte pour un problème, plusieurs méthodes de recherche de solutions approchées sont utilisées dont les algorithmes génétiques [8].

Dans la première partie de ce papier, nous allons parler brièvement des algorithmes génétiques et de leurs objectifs, ensuite nous allons appliquer cet algorithme à l'un des problèmes les plus connus dans ce domaine, à savoir le problème OneMax. Enfin, nous allons discuter les résultats obtenus en testant les différents opérateurs de mutation, croisement, etc.

Le nombre de paramètres liés à cet algorithme est très important (plusieurs opérateurs de mutation, croisement, sélection, remplacement...) et il n'y a pas d'opérateur meilleur que l'autre car l'efficacité de chaque opérateur dépend de plusieurs paramètres et c'est ce que nous allons voir dans les parties suivantes. Afin de résoudre ce problème et répondre à cette question : quel opérateur utiliser et quand l'utiliser ? Nous allons essayer de mettre en place d'un mécanisme permettant de choisir un opérateur o à l'instant t afin d'augmenter l'efficacité de l'algorithme génétique.

Dans la deuxième partie, nous allons donc discuter le fonctionnement de l'algorithme génétique implémenté dans la première partie, ensuite nous allons proposer des solutions qui permettent de choisir intelligemment l'opérateur à utiliser à un instant t de l'exécution de l'algorithme génétique. Enfin, nous allons discuter les nouveaux résultats obtenus et faire une comparaison avec les premiers résultats.

1.1 Rappels sur les algorithmes génétiques

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes. Leur but est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode de résolution exacte [2].

- Les algorithmes génétiques se basent au départ sur une population de solutions candidates qui va évoluer de génération en génération jusqu'à la génération qui contient les meilleures solutions [3].

- Chaque individu comprend des propriétés et il peut être sujet à des transformations génétiques (mutation, croisement...) [3].

- Chaque individu est évalué et cette valeur d'aptitude (fitness) est un critère pour sa survie d'une génération à une autre [3].

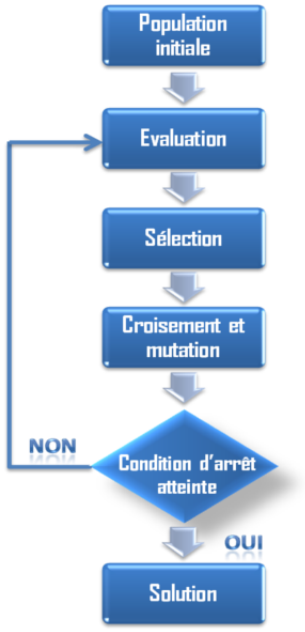

FIGURE 1 – Cycle de vie d'un algorithme génétique

1.2 Rappels sur le problème OneMax

Le problème OneMax est un problème simple qui consiste à maximiser le nombre de uns d'une séquence de bits [1].

Formellement, le problème OneMax peut être décrit comme la recherche d'une séquence (ou un vecteur) $\vec{x} = (x_1, x_2, ..., x_n)$, avec $x_i \in \{0, 1\}$, qui maximise la fonction suivante : $F(\vec{x}) = \sum_{i=1}^{n} x_i$

2 Application de l'algorithme génétique au problème OneMax

- Une population dans le cadre du problème OneMax est un ensemble fini d'individus ;
- Chaque individu est constitué de gênes (une séquence d'une taille finie de bits 0 ou 1) ;
- La population initiale est l'ensemble d'individus initiaux/première génération (n'ayant pas encore subit d'opérations de reproduction) ;
- La fitness d'un individu est la somme des bits ayant la valeur 1 ;
- La fitness d'une population est la somme des fitness des individus qui la composent.

À partir de la population initiale, l'algorithme génétique sélectionne un ou plusieurs individus et applique des opérations de reproduction (mutation, croisement...) sur ces derniers, ce qui va

générer de nouveaux enfants qui vont remplacer d'autres individus existants dans l'objectif de maximiser la fitness.

L'algorithme tourne en boucle jusqu'à ce qu'une condition d'arrêt soit atteinte.

3 Implémentation de l'algorithme génétique

Pour l'implémentation de l'algorithme génétique, nous avons utilisé le langage Java et gnuplot pour générer les graphes et les histogrammes.

Paramètres de l'algorithme : Taille de la population, taille des individus, nombre max de générations, nombre d'exécutions (afin d'avoir une moyenne sur plusieurs lancements de l'algorithme).

Population initiale : Ensemble d'individus dont tous les gênes sont initialisés à 0.

Sélection : Représente le choix des individus les mieux adaptés pour les différentes opérations d'évolution [5].

Opérateurs de sélection implémentés : Sélection d'un individu au hasard, 2 individus au hasard, meilleur individu, 2 meilleurs individus et les 2 meilleurs individus sur 5.

Croisement : Consiste à mélanger les gênes des individus choisis afin de reproduire leurs particularités [5].

Opérateurs de croisement implémentés :

- Croisement uniforme : L'enfant issu de ce croisement prend des gênes des 2 parents.
- Croisement simple : Selon une probabilité, l'enfant prend les gênes du parent 1 ou du parent 2.
- Croisement mono-point : Création de 2 nouveaux enfants à partir de 2 parents sélectionnés, tel que chaque enfant prend une partie des gênes du parent 1 et l'autre partie du parent 2.

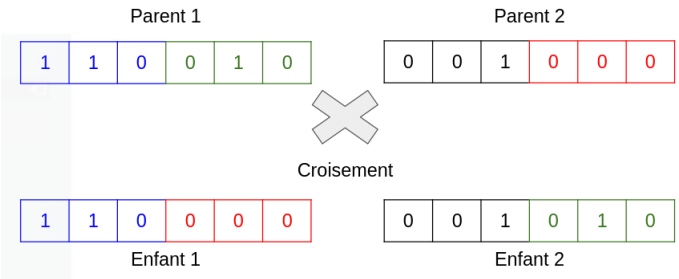

FIGURE 2 – Croisement mono-point

Mutation : Consiste à altérer un gène selon un facteur de mutation (probabilité). La mutation k-flip consiste à altérer k gênes d'un individu selon une probabilité de mutation [5].

Opérateurs de mutation implémentés : Mutation 1 flip, bit flip, 3 flip et 5 flip.

Remplacement : Représente le choix des individus à remplacer par les enfants issus des différentes opérations d'évolution.

Opérateurs implémentés : Remplacement du plus mauvais individu, des 2 plus mauvais individus et des 2 meilleurs individus

Fitness : $F(I) = \sum_{i=1}^{n} x_i$ tel que I : Individu ; x : gêne

Conditions d'arrêt : Nombre max de générations atteint, population parfaite atteinte (tous les gênes de tous les individus sont à 1).

4 Fonctionnement général de l'algorithme génétique

La figure 3 illustre le fonctionnement général de l'algorithme génétique implémenté ainsi que les différents opérateurs mis en place.

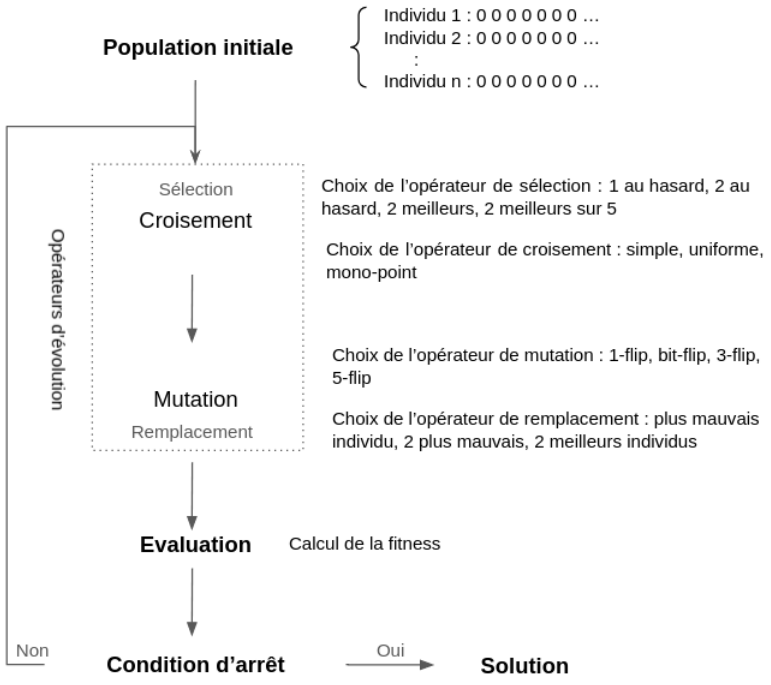

FIGURE 3 – Fonctionnement général de l'algorithme génétique implémenté

5 Résultats

Vu le nombre important de paramètres de l'algorithme (sélection, croisement, mutation, remplacement), nous ne pouvons pas étudier tous les résultats obtenus des différents paramètres. Dans cette étude, nous allons nous focaliser beaucoup plus sur les résultats liés à la mutation.

Paramètre	Valeur
Taille de la population	1000
Taille de l'individu	1000
Max génération	500
Nombre d'exécution	20
Opérateur de sélection	2 meilleurs individus
Opérateur de croisement	Croisement mono-point
Opérateur de remplacement	2 plus mauvais individus
Probabilité de mutation	0.25

TABLE 1 – Paramètres de l'algorithme génétique

5.1 Fitness en fonction des opérateurs de mutation

La figure 4 montre l'évolution de la Fitness moyenne obtenue sur 20 exécutions différentes en fonction des opérateurs de mutation (1-flip, bit-flip...).

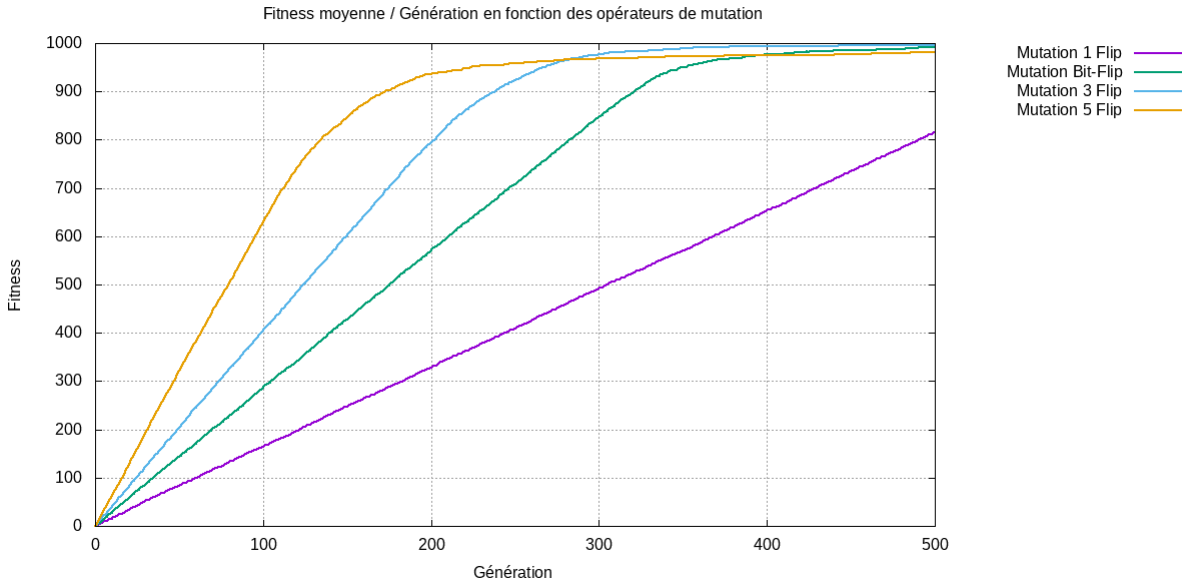

FIGURE 4 – Évolution de la Fitness moyenne en fonction des opérateurs de mutation

Discussion : La figure (4) illustre l'évolution de la fitness en testant les différents opérateurs de mutation. On remarque que l'opérateur 1-flip améliore légèrement la fitness mais d'une manière

continue et arrive après la condition d'arrêt à une fitness faible par rapport aux autres opérateurs.

L'opérateur 5-flip augmente considérablement la fitness et plus rapidement que les tous les autres opérateurs mais se stabilise après un certain nombre de génération (200 générations) où il devient moins améliorant (voir pas du tout améliorant), car à cet instant de l'exécution, la plupart des gênes des individus sont à 1 et une mutation 5-flip va donc diminuer la fitness au lieu de l'augmenter.

3-flip augment aussi la fitness un peu moins que 5-flip mais arrive à le dépasser après 300 générations. Bit-flip augment un peu moins la fitness en fonction de génération par rapport à 5-flip et 3-flip mais arrive à la fin de l'exécution à une fitness plus élevée que ces deux derniers.

Ce qu'on peut tirer comme conclusion à partir de la figure (4), c'est que au début de l'exécution de l'algorithme, tous les gêne des individus sont à *0*, dans ce cas, l'opérateur de mutation de plus performant est bien 5 flip car il mute 5 gênes à la fois ce qui augmente considérablement la fitness. Mais au bout de quelques générations, on se retrouve avec des individus qui ont des gênes majoritairement à *1*, donc une mutation 5 flip va diminuer la fitmness au lieu de l'augmenter et c'est ce qu'on remarque sur la figure, dans ce cas une mutation k-flip où $k < 5$ améliore mieux que 5 flip.

5.2 Fitness en fonction des opérateurs du croisement

La figure (5) montre l'évolution de la Fitness moyenne obtenue sur 20 exécutions différentes en fonction des opérateurs du croisement (croisement simple, uniforme et mono-point).

Pour cette expérimentation, nous avons utilisé les paramètres (1), avec Bit-flip comme opérateur de mutation.

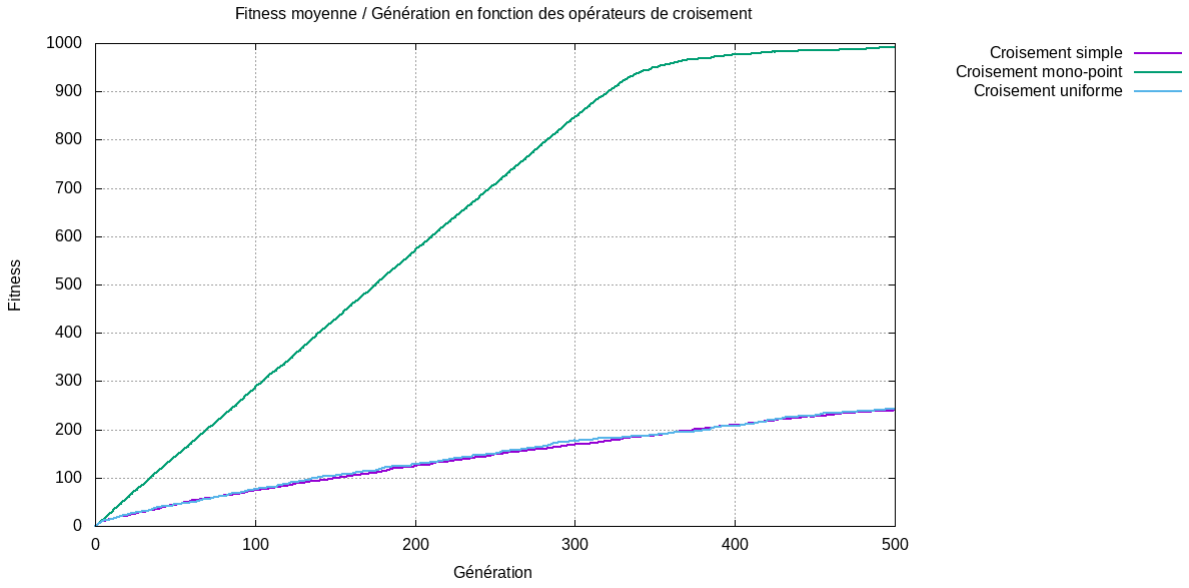

FIGURE 5 – Évolution de la Fitness moyenne en fonction des opérateurs du croisement

Discussion : On remarque que le croisement mono-point et plus performant que les autres opérateurs, car dans cette expérimentation l'algorithme sélectionne les 2 meilleurs individus de la population et donc le croisement mono-point (voir 3) de ces 2 parents va engendrer des enfants meilleurs (dans la plupart des cas) que les enfants issus du croisement simple (voir 3) ou uniforme (voir 3) ce qui explique cette différence.

5.3 Autres testes

Plusieurs autres testes peuvent être effectués sur l'algorithme génétique afin de bien comprendre son fonctionnement et ses comportements : évolution de la fitness en fonction de la taille de la population et/ou individu, évolution de la fitness en fonction de la probabilité de mutation, en fonction des opérateurs de sélection et remplacement, etc.

Les figures 6 et 7 montrent l'évolution de la fitness/génération en fonction des différents opérateurs de sélection et de remplacement. Sur ces deux figures, on remarque que presque dans tous les cas la sélection des 2 meilleurs individus et le remplacement des 2 plus mauvais est meilleur que les autres opérateurs contrairement au opérateurs de mutation qui dépendent d'une manière directe de la génération.

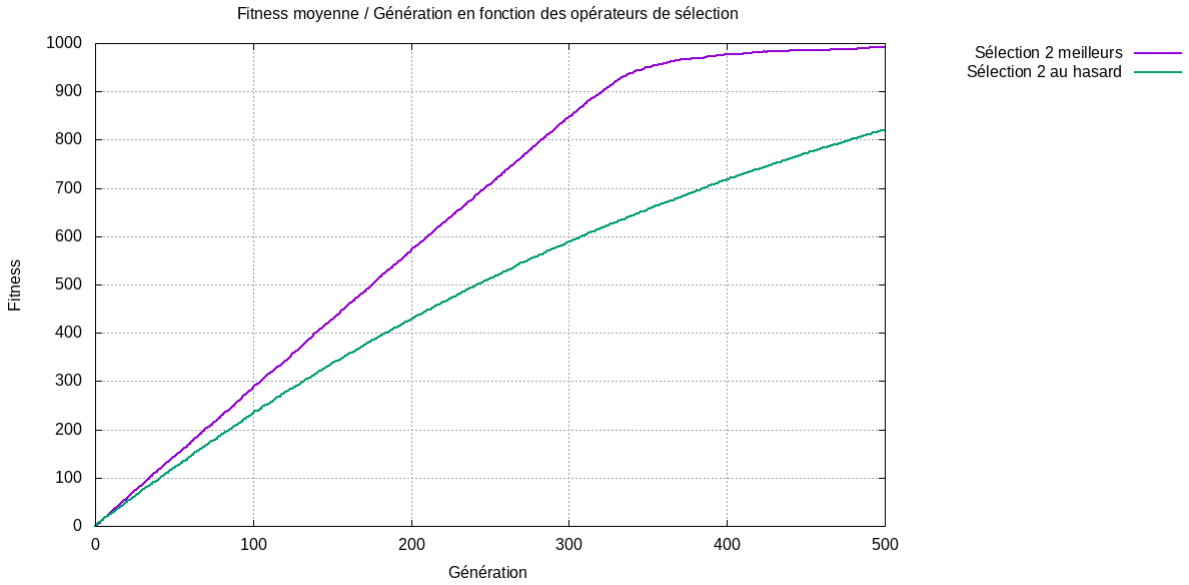

FIGURE 6 – Évolution de la Fitness moyenne en fonction des opérateurs de sélection

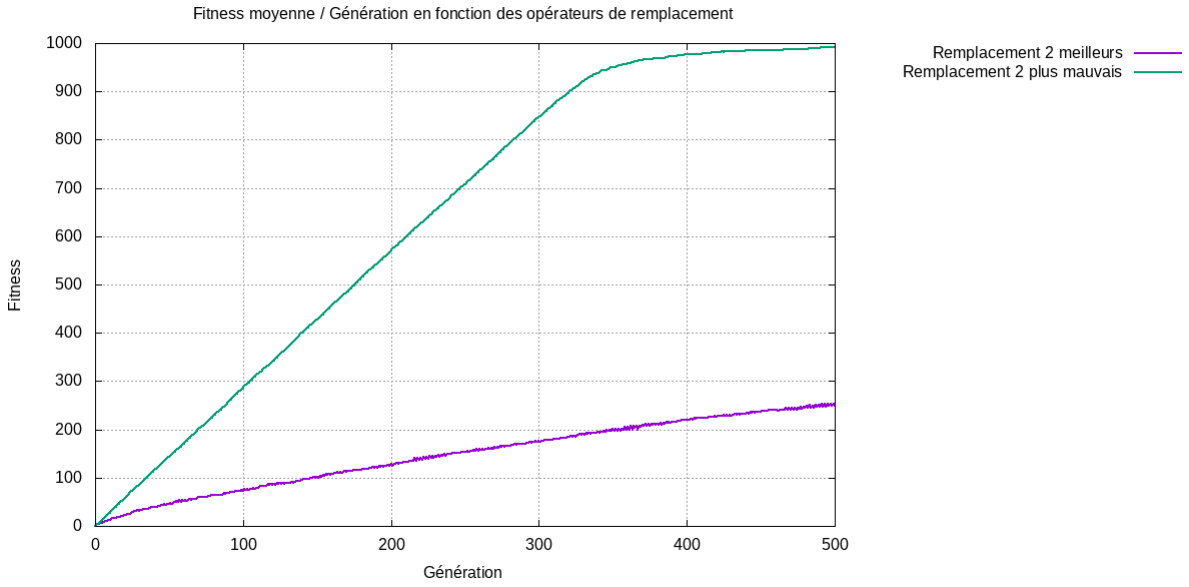

FIGURE 7 – Évolution de la Fitness moyenne en fonction des opérateurs de remplacement

6 Problématique du choix des opérateurs

Le choix de l'opérateur à utiliser comme nous l'avons vu dans la section précédente affecte d'une manière directe la Fitness ou l'efficacité de l'algorithme génétique d'une manière générale.

Dans cette étude, nous allons nous intéresser aux opérateurs de mutation. Nous avons vu dans la partie précédente que l'efficacité des opérateurs dépende directement de la taille du problème et de l'instant lorsque l'opérateur est utilisé. Nous avons vu qu'au début de l'exécution (0 - 200 génération) l'opérateur 5 Flip était plus améliorant que les autres opérateurs, mais à partir de la génération 200 nous avons remarqué l'inverse.

Une amélioration qui peut augmenter considérablement la Fitness serait donc de mettre en place un mécanisme qui permet de choisir intelligemment à un instant t l'opérateur o à utiliser.

Il existe plusieurs méthodes qui permettent à partir d'un certain nombre de données en entrée de décider de quelle action effectuer à un instant t afin de maximiser le gain.

Définition du problème : Nous avons :

- 4 opérateurs de mutation : 1 Flip, Bit-Flip, 3 Flip, 5 Flip.

- Un nombre max de générations.

- Une fonction Fitness (fitness cumulée qu'on cherche à maximiser).

Objectif : À chaque itération (génération), l'algorithme doit pouvoir choisir le meilleur opérateur de mutation qui permet de maximiser la fitness cumulée.

7 Algorithme du Bandit

Problème du bandit : Afin de simplifier le problème du bandit, nous allons prendre le cas le plus simple. On considère une machines à sous à 2 bras, notés A et B. L'action du bras A (resp. B) donne lieu à un gain qui n'est pas connu a priori, et le joueur cherche à actionner le bras le plus favorable [4]. L'idée consiste à choisir un bras au hasard, mais en modifiant au cours du temps la loi de probabilité du choix du bras, i.e. Si à l'étape n, le bras actionné avait une probabilité p_n d'être choisi, s'il produit un gain, on lui affecte à l'étape suivante une probabilité plus importante $p_{n+1} \geq p_n$.

Algorithme du bandit : L'algorithme du bandit tient son nom du problème expliqué dans le paragraphe précédent où le joueur cherche à maximiser son gain. L'algorithme du bandit est un modèle d'apprentissage par renforcement qui consiste à un instant t de choisir l'action a dont il a appris qu'elle récompense beaucoup [4].

Dans notre cas, les k-bras de la machine à sous (multi-armed bandit) sont les différents opérateurs de mutation et à chaque génération l'algorithme choisit un opérateur et ajuste les probabilités du choix des opérateurs en fonction des gains.

Récompense : La récompense est la valorisation immédiate d'une action dans un état donné [6] : $R_t = Fitness_t - Fitness_{t-1} + d$
tel que d est la valeur de décalage qui permet de ne pas avoir une récompense négative.

Il existe plusieurs approches (politiques) pour résoudre ce problème : $\varepsilon-$Greedy, Probability Matching, Upper-Confidence Bound (UCB), Lower Upper Confidence Bound (LUCB), etc.

Dans notre cas, nous avons implémenté les deux algorithmes "Sélection par roulette proportionnelle" et "UCB".

7.1 Roulette proportionnelle pour le choix d'opérateur

Au lancement de l'algorithme, tous les opérateurs ont une probabilité P_{init} d'être sélectionné, tel que $P_{init} = \frac{1}{nb\_operateurs}$.

Dans notre cas, nous avons 4 opérateurs, donc la probabilité initiale de chaque opérateur $P_{init} = \frac{1}{4} = 0.25$.

L'algorithme choisit donc aléatoirement un opérateur, calcule l'amélioration/récompense immédiate et le gain, ensuite ajuste les probabilités des opérateurs suivant cette formule :

$\pi_i^{t+1} = P_{min} + (1 - N.P_{min}) \frac{u_i^{(t+1)}}{\sum_{k=1}^{N} u_k^{(t+1)}}$, tel que :

π_i^t : probabilité d'appliquer l'opérateur i à l'itération t.

P_{min} : pour assurer une probabilité de sélection non nulle pour tous les opérateurs.

$0 \leq p_{min} \leq \frac{1}{nb\_operateur}$.

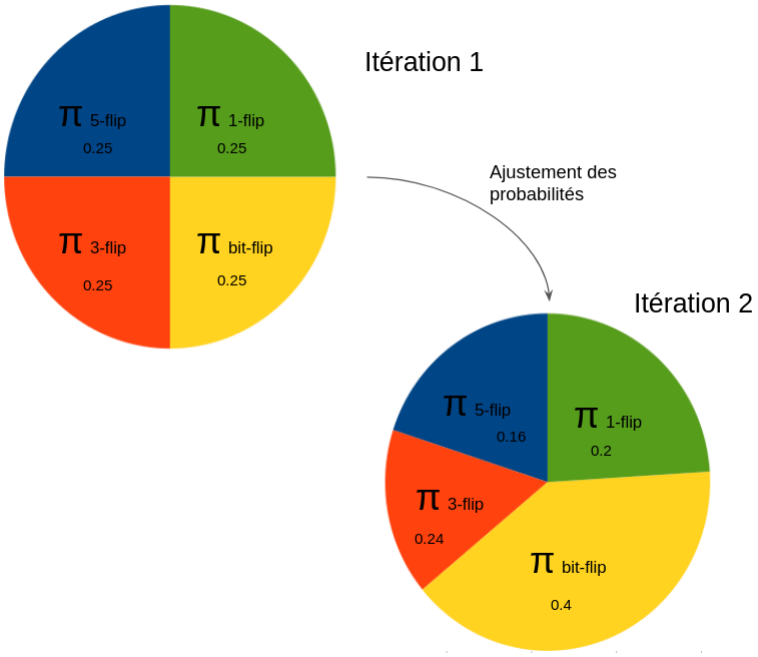

FIGURE 8 – Sélection des opérateurs par roulette proportionnelle - Probability Matching

7.2 Upper Confidence Bound (UCB)

Avec l'algorithme UCB, l'utilisateur calcule la moyenne empirique de la récompense [7] pour chacun des opérateurs :

$X_j = \frac{1}{T_j} \sum_{i=1}^{t} r_i x_{aj=i}$.

Tel que :
t : désigne le nombre d'essais réalisés par l'utilisateur.
T_j : le nombre d'essais sur l'opérateur j.
r_i : désigne la récompense obtenue lors de l'essai i.
x : désigne la fonction indicatrice qui indique que l'opérateur j a été choisi pour l'essai i.

Pour calculer l'index, on ajoute un biais qui permet à l'algorithme d'explorer les différents opérateurs :
$B_j = X_j + A_j$, tel que : $A_j = \sqrt{\frac{2log(t)}{T_j}}$

8 Analyse et discussion des résultats obtenus

Dans cette partie, nous allons présenter les différents résultats obtenus en utilisant les deux algorithmes présentés dans la section précédente, et enfin nous analysons et comparons ces résultats avec les résultats obtenus dans la section 5.

Paramètres de l'algorithme :

Paramètre	Valeur
Taille de la population	1000
Taille de l'individu	1000
Max génération	500
Nombre d'exécution	10
Opérateur de sélection	2 meilleurs individus
Opérateur de croisement	Croisement mono-point
Opérateur de remplacement	2 plus mauvais individus
Probabilité de mutation	0.25

TABLE 2 – Paramètres de l'algorithme utilisés pour les tests

Évolution de la fitness avec l'algorithme PM : Sur la figure 9 on remarque le gain apporté par l'algorithme PM en terme de fitness par rapport à l'algorithme génétique classique (mutation bit-flip). En fait, l'AG classique avec l'opérateur bit-flip améliore la fitness d'une manière continue mais légèrement jusqu'à ce que la condition d'arrêt *max génération* soit atteinte. Par contre l'algorithme PM commence d'abord par attribuer des probabilités égales aux différents opérateur ensuite ajuste ces probabilités en fonction du gain de chacun des opérateurs. l'opérateur qui récompense beaucoup a une probabilité plus importante que les autres opérateurs d'être choisi.

Au lancement de l'algorithme (avec une population initialisée à 0), la mutation 5-flip est l'opérateur qui améliore le plus la fitness, ce qui justifie le nombre d'utilisation de ce dernier plus que les autres opérateur (voir la figure 10) et ce qui explique également l'augmentation considérable de la fitness dans l'intervalle $[0, 250]$ générations. Ensuite (> 250 générations), l'opérateur 5-flip n'améliore pas la fitness, donc l'algorithme PM a choisi les autres opérateurs bit-flip, 3-flip et 1-flip ce qui explique l'évolution légère la fitness à la fin de l'exécution.

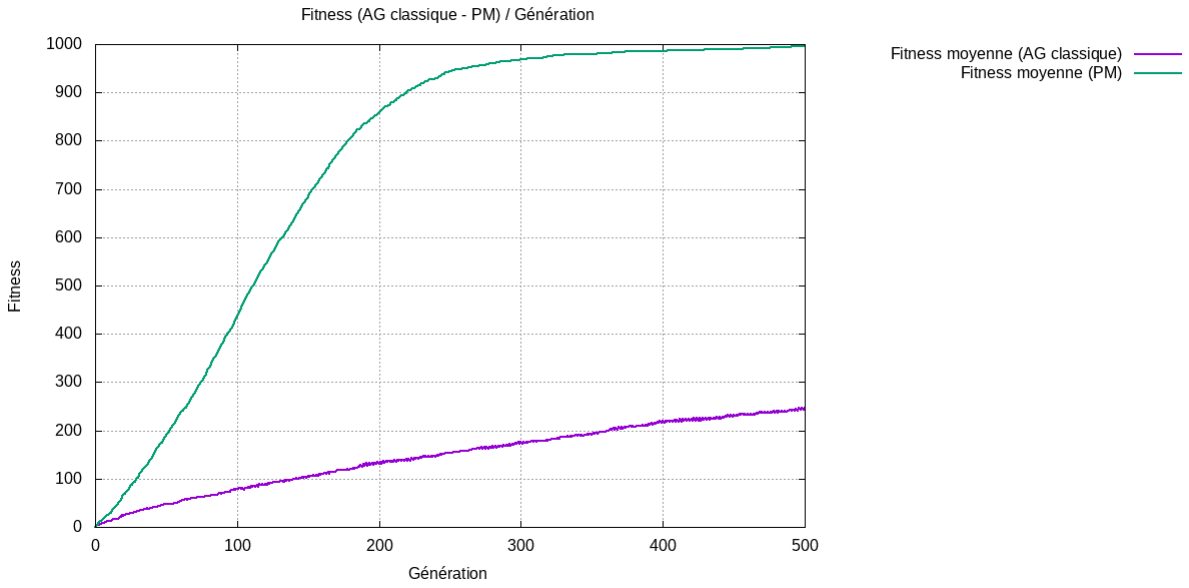

FIGURE 9 – Fitness moyenne de l'algorithme génétique classique et PM (Probability Matching)

Sur la figure 10 et 11, on peut remarquer ce qu'on a expliqué dans le paragraphe précédent. On remarque que l'opérateur 5-flip est beaucoup plus utilisé par rapport aux autres opérateurs car il améliore beaucoup plus la fitness au début l'exécution, ensuite l'opérateur 3 flip, 1 flip et enfin 2 fois avec une légère différence.

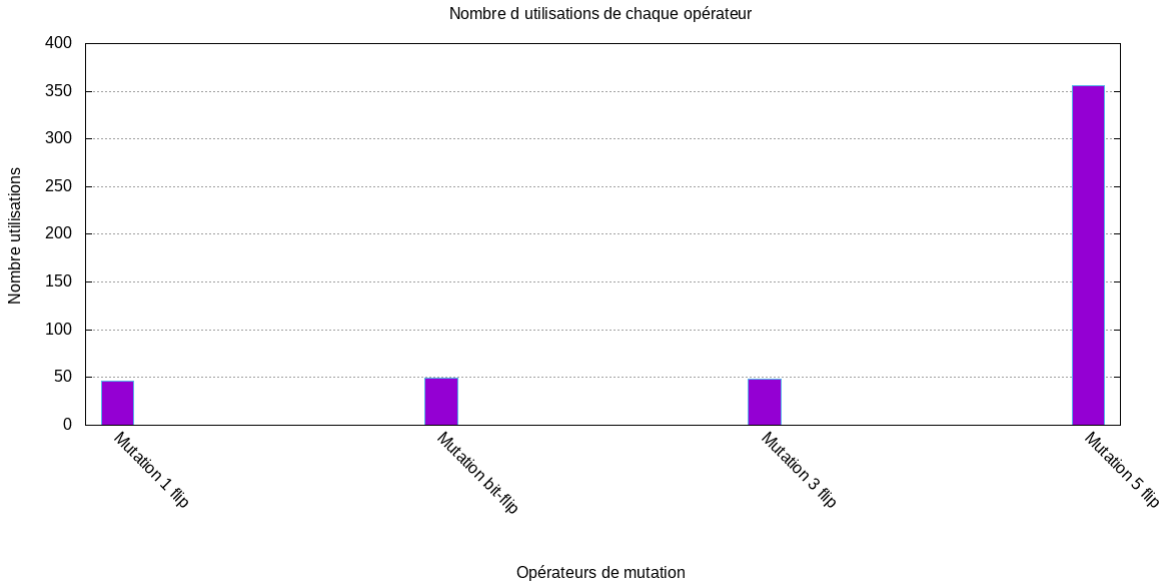

FIGURE 10 – Moyenne d'utilisation de chaque opérateur de mutation - PM (Probability Matching)

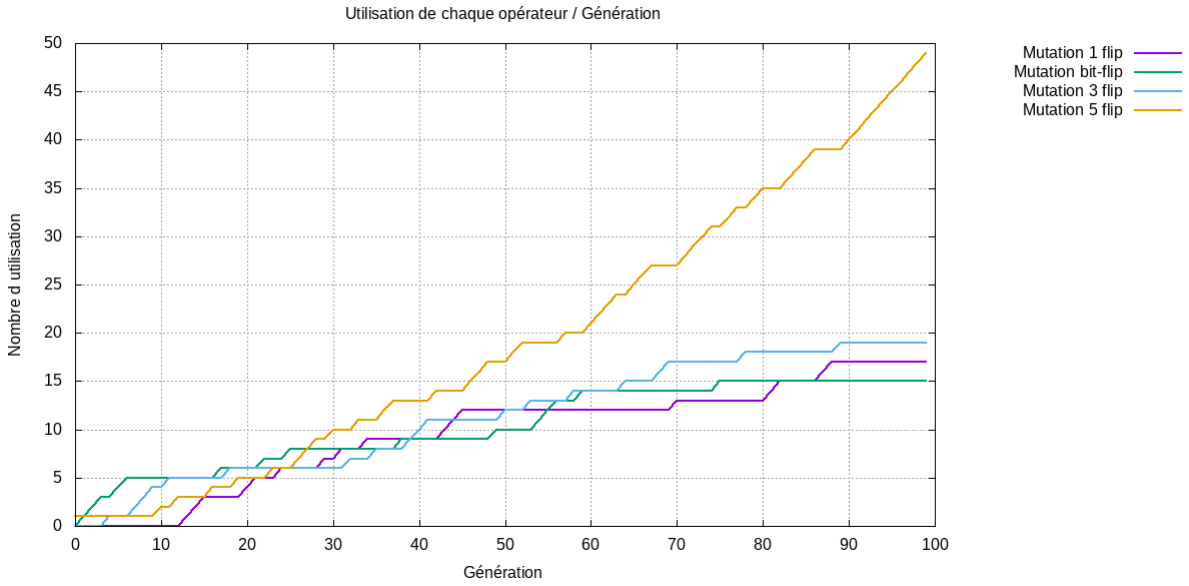

FIGURE 11 – Moyenne d'utilisation de chaque opérateur de mutation - PM (Probability Matching)

Évolution de la fitness avec l'algorithme UCB : Sur la figure 12, on remarque les fitness obtenues avec l'algorithme UCB dépasse largement les fitness de l'AG classique. UCB dans la phase d'exploration il choisi aléatoirement les opérateurs à utiliser ce qui explique l'augmentation moins considérable de la fitness dans l'intervalle $[0, 40]$ génération, ensuite l'algorithme choisi l'opérateur qui récompense beaucoup.

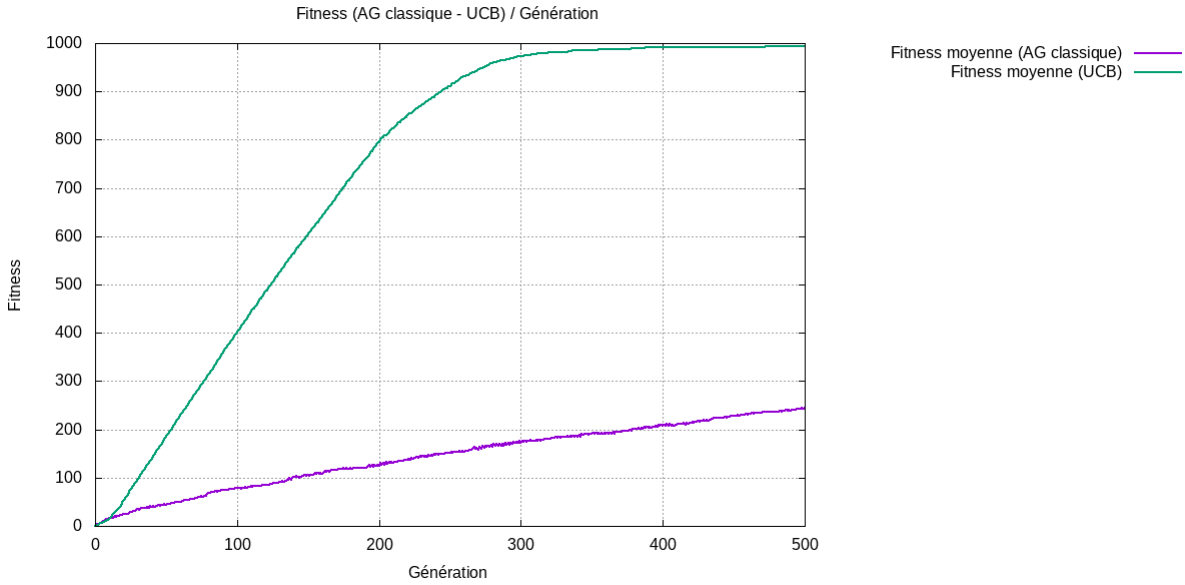

FIGURE 12 – Fitness moyenne de l'algorithme génétique classique et UCB

Sur l'histogramme 13, on remarque que UCB, contrairement à PM a utilisé beaucoup plus l'opérateur 3-flip. En fait, 3-flip et 5-flip sont en concurrence parce que les deux améliore considérablement la fitness.

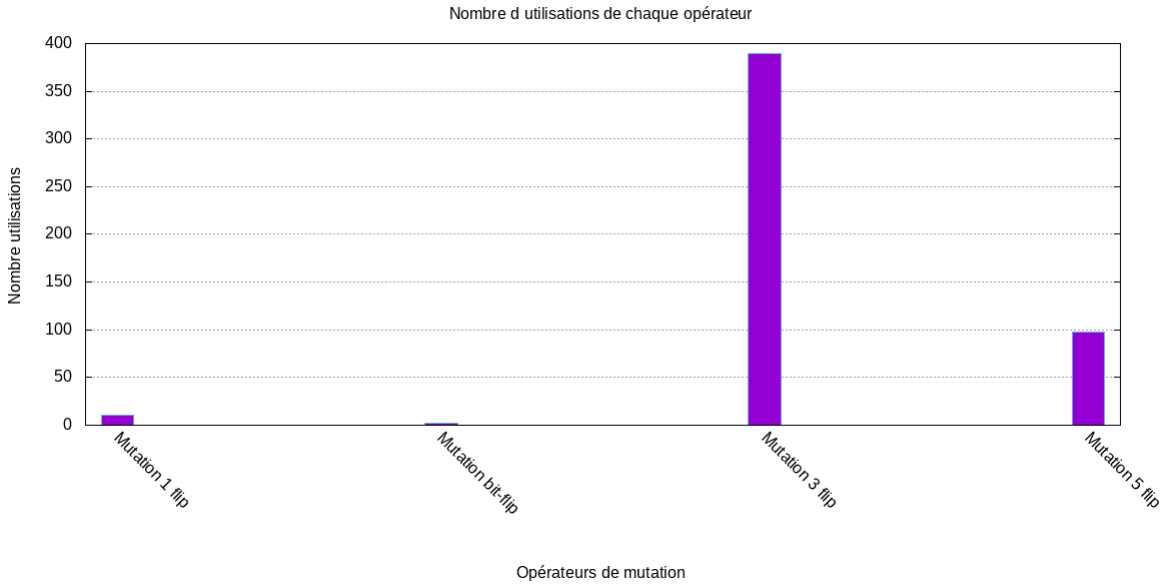

FIGURE 13 – Moyenne d'utilisation de chaque opérateur de mutation par UCB

La figure 14 est une comparaison entre UCB et PM. On remarque qu'il n'y a pas une grande différence entre les deux algorithmes en terme d'évolution de la fitness en fonction de générations.

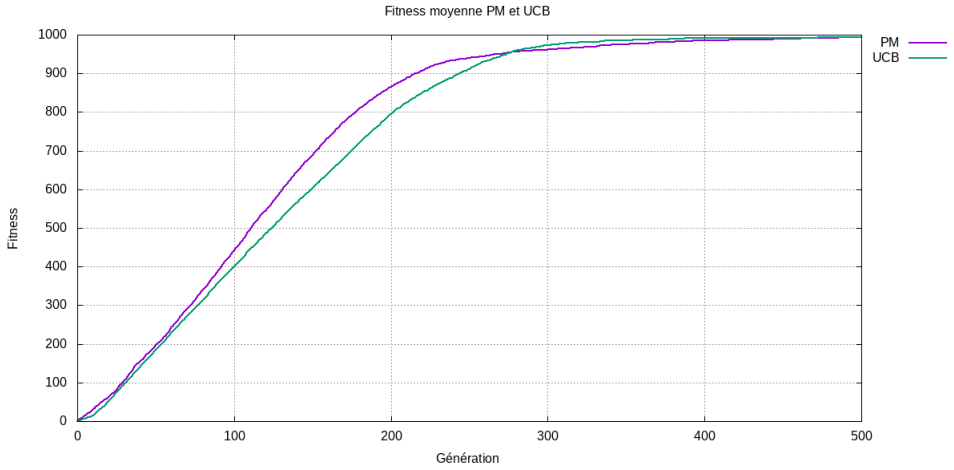

FIGURE 14 – Fitness moyenne PM et UCB

9 Conclusion

Dans ce papier, nous avons vu les algorithmes génétiques, le problème OneMax, ensuite on a implémenté un algorithme génétique en java et nous l'avons appliqué au problème OneMax.

Dans la première partie, nous avons présenté les résultats obtenus par l'AG classique et nous vu la difficulté du choix des opérateurs (de mutation, croisement...), car le gain de chaque opérateur dépend de l'état et de l'instant lorsqu'il est utilisé.

Dans la deuxième partie, nous avons traité le problème du choix des opérateur, et avons implémenté deux mécanismes d'apprentissage par renforcement PM et UCB qui permettent de choisir l'opérateur de mutation de plus performant en fonction de l'état de la population.

Enfin, nous avons présenté les résultats obtenus par ces deux derniers algorithmes UCB et PM, discuté et comparé ces résultats avec les résultats obtenus dans la première partie.

9.1 Perspectives d'amélioration

Nous avons vu que le choix d'opérateur dépend d'une manière directe de l'instant et de l'état actuel de la population. Dans ce travail nous avons implémenté des mécanisme qui permettent de choisir intelligemment les opérateurs de mutation, une amélioration serait donc d'étendre ce mécanisme pour la sélection des autres opérateurs (croisement, remplacement...). Une autre amélioration serait aussi d'implémenter d'autres mécanisme d'apprentissage par renforcement et les comparer à UCB et PM.

Références

[1] The OneMax Problem. https ://tracer.lcc.uma.es/problems/onemax/onemax.html, Consulté le 06 février 2022.

[2] Algorithme génétique - définition et explications. https ://www.techno-science.net/glossaire-definition/Algorithme-genetique.html, Consulté le 07 février 2022.

[3] Alban. Algorithme génétique. https ://ledatascientist.com/algorithme-genetique/, Consulté le 07 février 2022.

[4] Bandit manchot. *https : //fr.wikipedia.org/wiki/Bandit_manchot*, Consulté le 09 février 2022.

[5] Algorithme génétique - Fonctionnement. *https : //igm.univ−mlv.fr/ dr/XPOSE*2013/*tleroux _genetic_algorithm/fonctionnement.html*, Consulté le 07 février 2022.

[6] Reinforcement learning https ://blog.engineering.publicissapient.fr/2020/02/12/reinforcement-learning-introduction/, Consulté le 08 février 2022

[7] Introduction aux algorithmes de bandit *https : //perso.crans.org/besson/publis/notebooks/ Introduction_aux_algorithmes_de_bandit__comme_UCB*1*_et_Thompson_Sampling.html*, Consulté le 08 février 2022

[8] Jorge Maturana, Frédéric Lardeux, Frédéric Saubion. Génération et contrôle autonomes d'opérateurs pour les algorithmes évolutionnaires. Cinquièmes Journées Francophones de Programmation par Contraintes, Orléans, juin 2009, Jun 2009, France. pp.185-195. hal-00387811