Algorithmes adaptatifs pour l'optimisation combinatoire

Thomas Landais

Résumé

Ce rapport présente la mise en oeuvre des méthodes suivantes :
— Roulette adaptative
— UCB (Upper Confidence Bound)
— Algorithme génétique simple
— Algorithme basé sur estimation de distribution (EAD)
— Modèle en îles

Le problèmes traité est le One Max.

1 Introduction

Dans ce rapport, nous allons analyser plusieurs méthodes d'aide à la décision appliquées à des problèmes d'optimisation combinatoire. Le domaine de l'optimisation combinatoire consiste à résoudre des problèmes dans lesquels l'ensemble des solutions réalisables est discret et souvent de cardinalité exponentielle, rendant leur énumération exhaustive impossible. L'objectif est de trouver, parmi cet ensemble, la solution optimale ou une bonne approximation, selon une fonction objectif définie. Ces problèmes se retrouvent dans divers domaines tels que la logistique, la planification, l'algorithmique ou encore la recherche opérationnelle, et nécessitent l'utilisation de techniques heuristiques ou métaheuristiques pour parvenir à une prise de décision efficace. Dans ce contexte, l'étude des différentes méthodes d'aide à la décision permet de mieux comprendre les compromis entre exploration et exploitation et d'améliorer la qualité des solutions obtenues.

2 Description des algorithmes et méthodes utilisés

2.1 Steady-State

Cet algorithme s'inspire des mécanismes de l'évolution naturelle (sélection, croisement et mutation) pour faire évoluer une population de solutions. À chaque génération, les individus sont évalués selon leur fitness et les meilleurs sont sélectionnés pour produire la génération suivante. L'algorithme itère ce processus jusqu'à l'atteinte d'un critère d'arrêt (nombre d'itérations maximum ou convergence de la population).

2.2 Roulette Adaptative

Dans cet algorithme, chaque opérateur i se voit attribuer une probabilité d'application $\pi_i(t)$ à l'itération t. Afin de récompenser les opérateurs qui améliorent la fitness, on met à jour cette probabilité en fonction de leur performance $u_i(t+1)$ via la formule :

$$\pi_i(t+1) = p_{\min} + (1 - N \cdot p_{\min}) \frac{u_i(t+1)\,\pi_i(t)}{\sum_{k=1}^{N} u_k(t+1)\,\pi_k(t)},$$

où $p_{\min}$ représente une probabilité minimale garantie pour chaque opérateur et N le nombre total d'opérateurs. Ainsi, la part $1-N\cdot p_{\min}$ est répartie proportionnellement au produit $u_i(t+1)\,\pi_i(t)$: les opérateurs qui génèrent une forte amélioration voient leur probabilité augmenter, tandis que ceux avec de faibles performances voient leur probabilité diminuer, assurant ainsi un bon compromis entre exploration et exploitation.

2.3 UCB (Upper Confidence Bound)

La stratégie UCB, issue de la théorie des bandits manchots, vise à équilibrer l'exploitation des solutions connues et l'exploration de nouvelles régions de l'espace de recherche. Pour chaque solution, UCB combine la valeur moyenne observée et un terme de confiance qui incite à explorer les solutions moins visitées, permettant ainsi une approche dynamique et adaptative dans la sélection des candidats. La politique UCB (Upper Confidence Bound) choisit l'opérateur qui maximise le critère :

$$\mathrm{UCB1}(o_i, t) = u_i(t) + \sqrt{\frac{2\log\left(\sum_{k=1}^{n} nb_k(t)\right)}{nb_i(t)}},$$

où $u_i(t)$ est le gain moyen de l'opérateur o_i à l'itération t et $nb_i(t)$ le nombre de fois où il a été appliqué. Le premier terme favorise l'exploitation des opérateurs performants, tandis que le second encourage l'exploration des moins utilisés.

2.4 Algorithme basé sur Estimation de Distribution (EAD)

Les EDA explorent l'espace de solutions en construisant un modèle probabiliste $M(g)$ à partir d'une population de solutions prometteuses $S(g)$. Par exemple, dans le cas d'un problème binaire, on peut estimer la distribution comme

$$P(x) = \prod_{i=1}^{n} p_i(x_i),$$

où $p_i(x_i)$ représente la probabilité d'obtenir la valeur x_i à la position i. Ce modèle est ensuite échantillonné pour générer de nouvelles solutions $O(g)$ qui sont intégrées dans la population pour la génération suivante. Le processus itératif se poursuit jusqu'à la convergence ou la satisfaction d'un critère d'arrêt.

2.5 Modèle en îles

Le modèle en îles répartit la population sur plusieurs sous-populations (îles), chacune utilisant un opérateur de mutation spécifique (par exemple, inversion d'un,

trois ou cinq bits, ou bit-flip). À chaque génération, une migration est effectuée entre les îles selon une matrice de migration, dont les probabilités sont ajustées en fonction des gains en fitness observés chez les individus migrants. Les individus qui améliorent leur fitness sont récompensés, favorisant ainsi la propagation de solutions prometteuses. Enfin, les statistiques (meilleure fitness, nombre d'évaluations et taille de population par île) sont suivies et moyennées sur plusieurs exécutions pour évaluer la performance globale.

3 Analyses expérimentales

3.1 Présentation du problème traité

Le problème One Max consiste à maximiser le nombre de bits à 1 dans une chaîne binaire. L'objectif est de trouver, pour une chaîne de longueur donnée, la configuration qui contient le plus grand nombre de 1 possibles. Ce problème, simple mais représentatif, sert de banc d'essai pour évaluer et comparer différentes approches d'optimisation.

3.2 Description des conditions expérimentales

Pour les expérimentations, la longueur de la chaîne a été fixée à 1000 bits, avec une population de 20 individus. Chaque expérimentation a été répétée 10 fois afin de moyenner les résultats et ainsi atténuer le caractère stochastique des courbes de convergence.

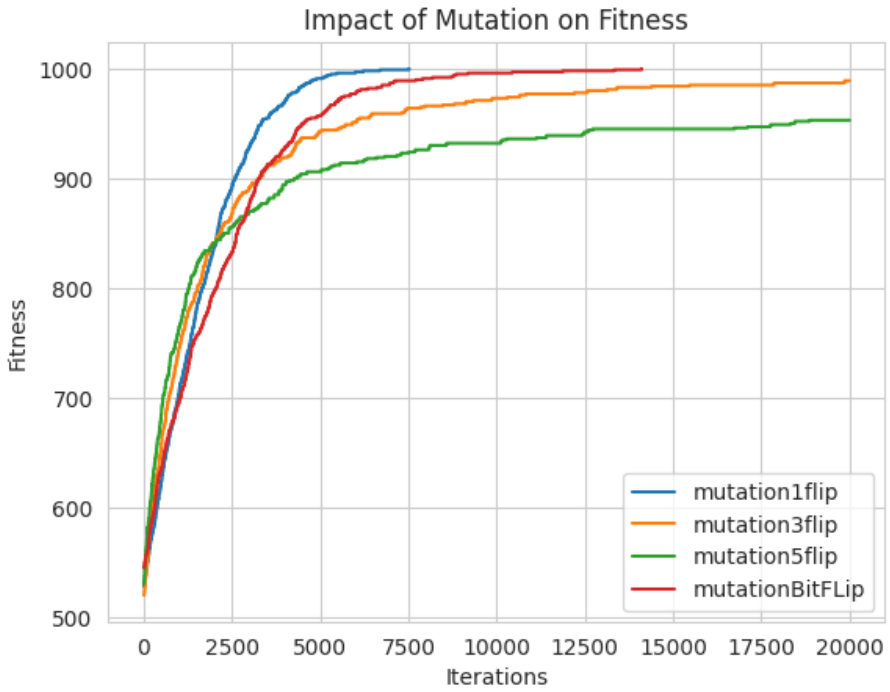
3.3 Analyse des méthodes adaptatives utilisées

Pour chaque méthode, une analyse des résultats à été réalisée.

3.3.1 Steady-State

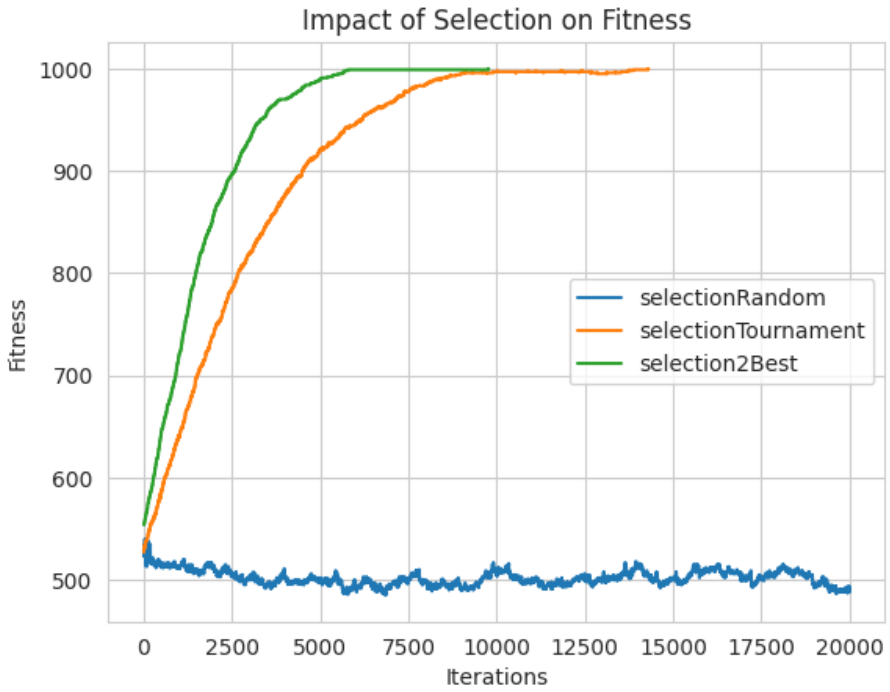
: Pour cette approche, les paramètres suivant on était analysés

- **L'opérateur de mutation**
  - **1flip** : Inverse un seul bit aléatoire, favorisant une exploration locale précise et une convergence rapide.
  - **3flip** : Inverse trois bits aléatoires, permettant une exploration plus large mais ralentissant la convergence.
  - **5flip** : Inverse cinq bits aléatoires, favorisant une exploration agressive mais limitant la précision.
  - **bitflip** : Inverse chaque bit avec une probabilité 1/TAILLE_CHAÎNE, combinant exploration globale et précision locale.



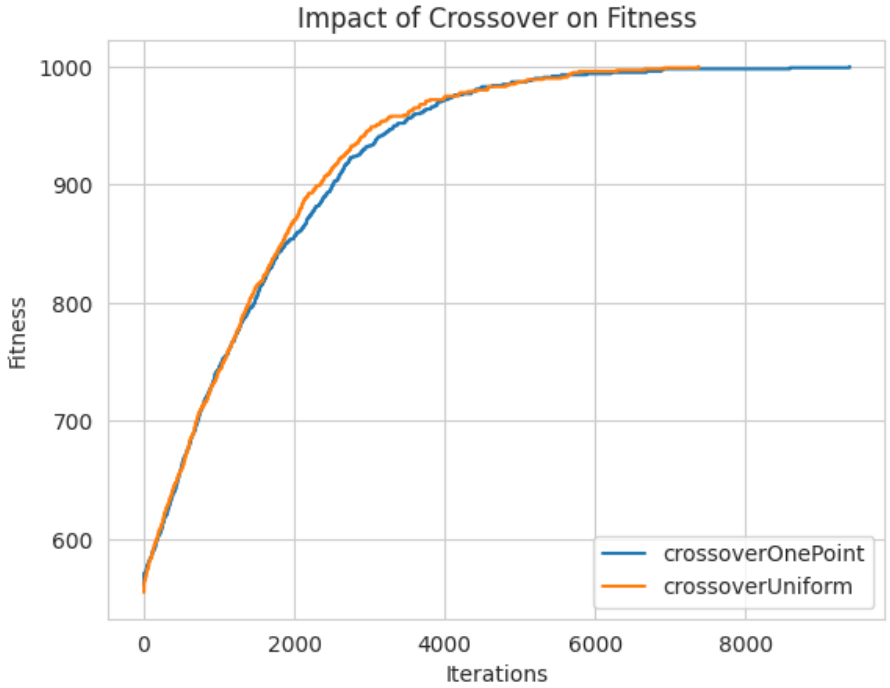
Les courbes montrent que mutation1flip est le plus performant, atteignant rapidement la solution optimale, suivi par mutationBitFlip, qui converge également mais plus lentement. mutation3flip et mutation5flip affichent initialement de meilleures performances en raison de leur capacité à modifier plusieurs bits simultanément, accélérant ainsi l'exploration en début de recherche. Cependant, cette agressivité nuit à leur précision, les empêchant de converger vers la solution optimale.

- **La sélection des individus à faire muter**
  - **selectionRandom** : Sélectionne les individus à muter de manière aléatoire, sans privilégier les individus prometteurs.
  - **selectionTournament** : Organise des tournois entre individus pour sélectionner les meilleurs, offrant un bon équilibre entre exploration et exploitation.
  - **selection2Best** : Sélectionne directement les deux meilleurs individus pour maximiser les chances de convergence rapide.



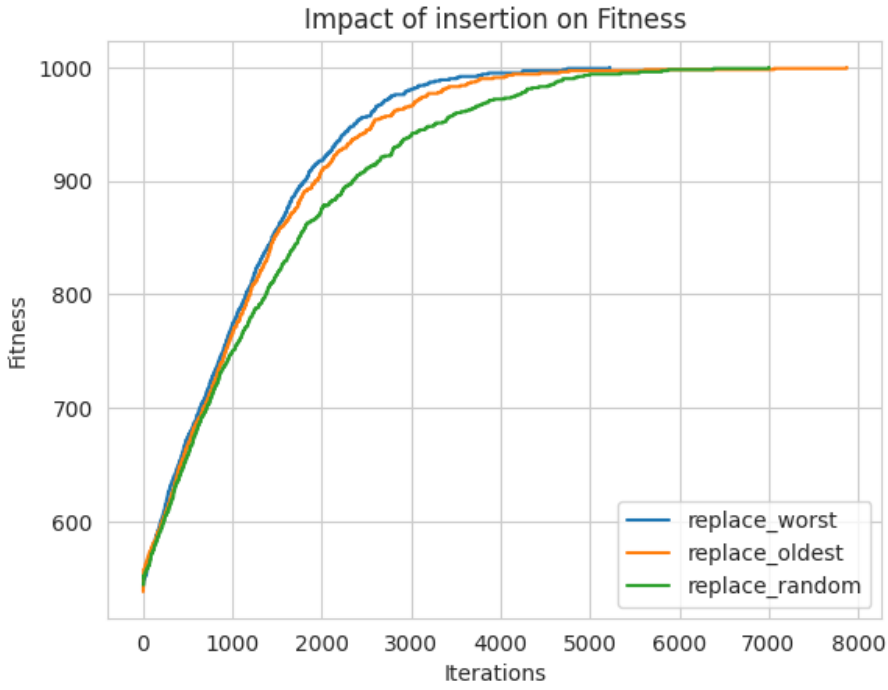
Les courbes montrent que la méthode `selection2Best` est la plus performante, convergeant rapidement vers la solution optimale. Elle est suivie par `selectionTournament`, qui atteint également la solution optimale mais avec une convergence plus lente. En revanche, `selectionRandom` affiche des performances nettement inférieures, incapable de progresser au-delà de la fitness initiale, car elle ne privilégie pas les individus prometteurs. Ces résultats soulignent l'importance d'une stratégie de sélection favorisant les meilleurs individus pour accélérer la convergence.

— **Le type de crossover effectué**
  — **crossoverOnePoint** : Effectue un croisement en un point, échangeant les segments après un point aléatoire entre deux individus.
  — **crossoverUniform** : Effectue un croisement uniforme, échangeant les bits entre deux individus avec une probabilité fixe.



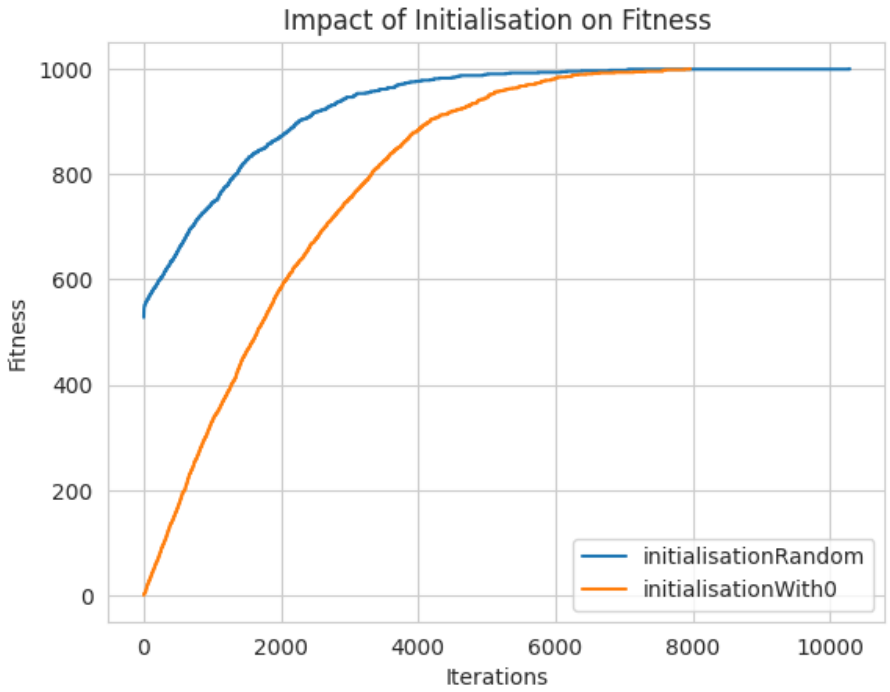
Les courbes montrent que les performances des deux méthodes de crossover, `crossoverOnePoint` et `crossoverUniform`, sont similaires. Les deux approches convergent vers la solution optimale avec des courbes presque identiques, ce qui suggère que le type de croisement effectué a un impact limité sur la performance pour ce problème particulier.

— **Le mode de remplacement des individus**
  — **replace_worst** : Remplace les pires individus de la population par les nouveaux individus.
  — **replace_oldest** : Remplace les individus les plus anciens dans la population.
  — **replace_random** : Remplace des individus choisis aléatoirement.



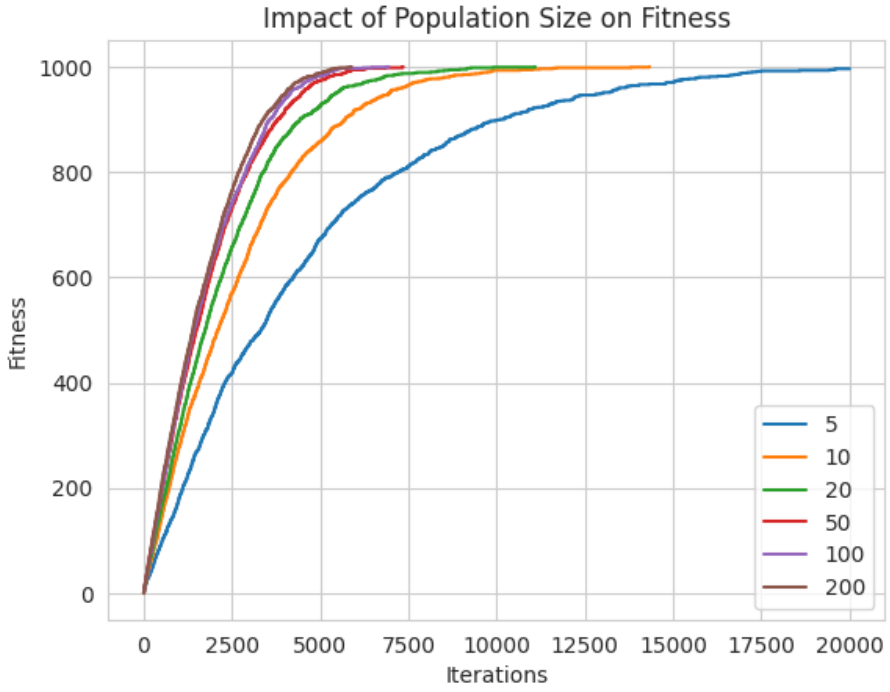
Les courbes montrent que la méthode `replace_worst` est la plus performante, convergeant plus rapidement vers la solution optimale. `replace_oldest` suit de près, avec une convergence légèrement plus lente. Enfin, `replace_random` affiche une performance inférieure, en raison du remplacement aléatoire qui ne prend pas en compte la qualité des individus. Ces résultats soulignent l'importance de stratégies de remplacement favorisant le maintien des meilleurs individus dans la population.

— **L'initialisation du problème**
  — **initialisationRandom** : Les individus sont initialisés aléatoirement, favorisant une exploration diversifiée dès le début.
  — **initialisationWith0** : Les individus sont initialisés avec des chaînes contenant principalement des zéros, favorisant une progression plus contrôlée.



Les courbes montrent que l'initialisation aléatoire (`initialisationRandom`) permet une convergence légèrement plus rapide que l'initialisation avec des zéros (`initialisationWith0`). Cependant, en raison de la nature logarithmique de la progression, les deux courbes finissent par se rejoindre approximativement au même moment. Cela souligne que, bien que le choix de l'initialisation puisse influencer les premières étapes de la recherche, son impact sur la convergence globale est limité.

— **La taille de la population**



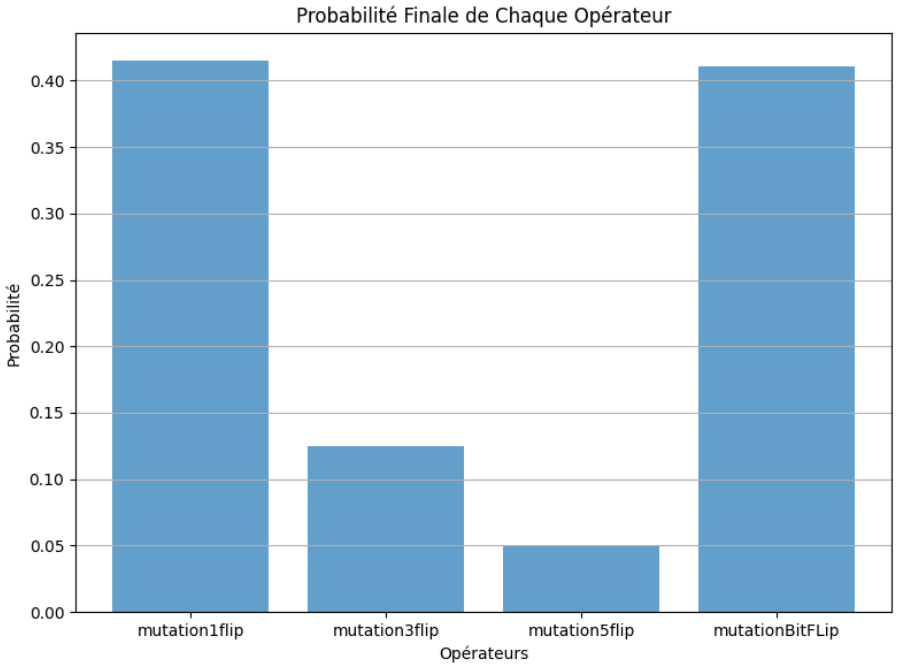
Les courbes montrent que la performance augmente significativement avec la taille de la population jusqu'à environ 20 individus. Au-delà, augmenter davantage la taille de la population n'apporte que des améliorations marginales tout en augmentant considérablement les temps de calcul. Cela suggère qu'une taille de population autour de 20 constitue un bon compromis entre

performance et efficacité computationnelle.

L'approche **Steady-State** converge vers la solution optimale même pour de grandes tailles de chaînes de bits, à condition que les paramètres tels que l'opérateur de mutation, la sélection, le mode de remplacement des individus, et la taille de la population soient bien ajustés. Ces résultats soulignent l'importance de choisir des configurations adaptées pour maximiser l'efficacité et minimiser les temps de calcul.

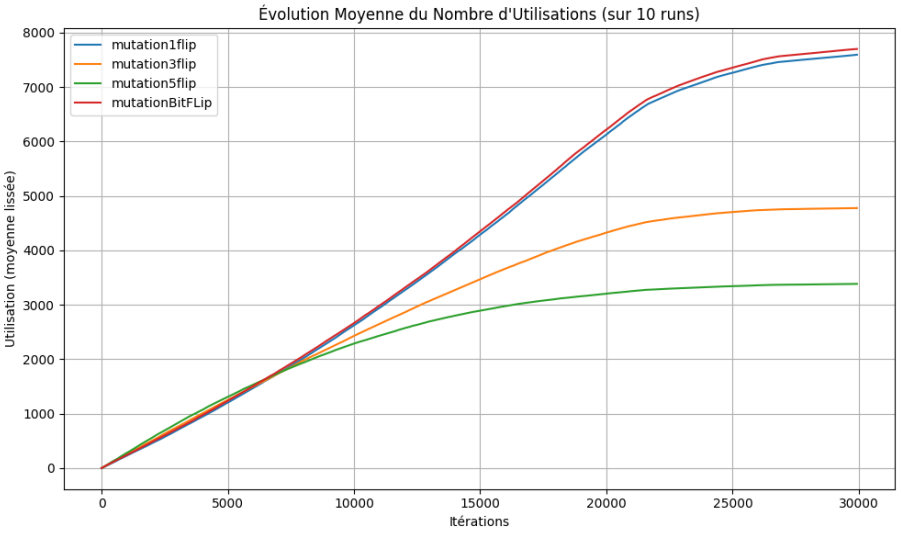
3.3.2 Roulette adaptative

— **Probabilité finale de chaque opérateur**



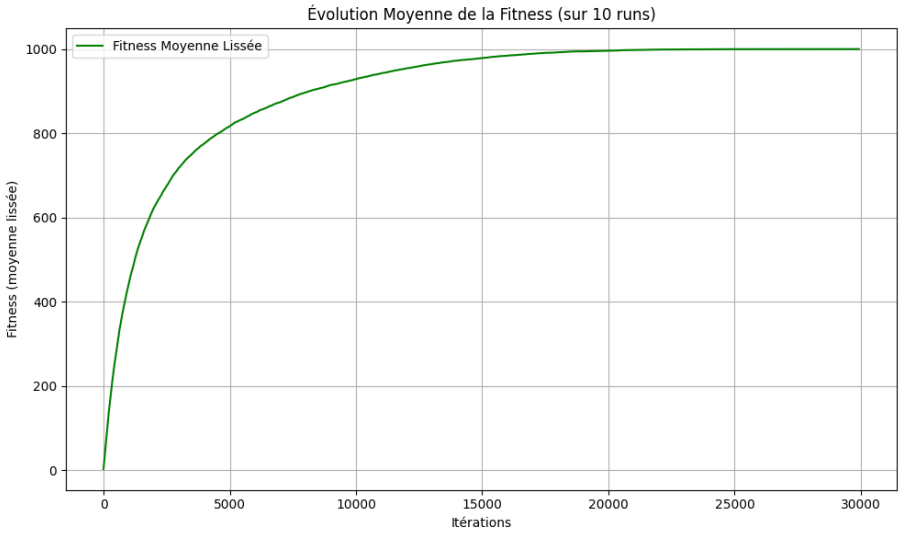
Cet histogramme représente les probabilités finales attribuées à chaque opérateur après l'exécution de la roulette adaptative. Les résultats montrent que les opérateurs `mutation1flip` et `mutationBitFlip` sont les plus favorisés, avec des probabilités finales supérieures à 0.4. Les opérateurs `mutation3flip` et `mutation5flip`, moins efficaces, ont vu leurs probabilités diminuer, illustrant l'adaptation dynamique en faveur des mutations performantes.

— **Évolution moyenne du nombre d'utilisations des opérateurs**



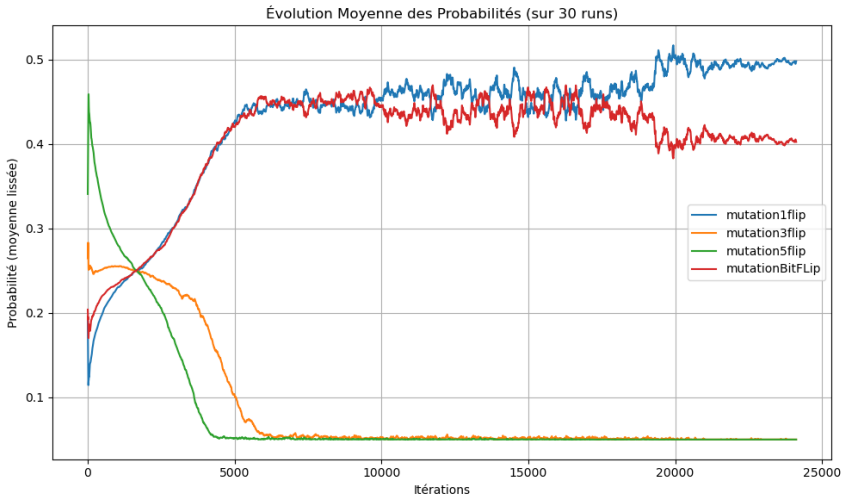
Ce graphique représente l'évolution moyenne du nombre d'utilisations pour chaque opérateur au fil des itérations. On observe que les opérateurs `mutation1flip` et `mutationBitFlip` ont été progressivement davantage utilisés, reflétant leur efficacité supérieure. En revanche, `mutation3flip` et `mutation5flip` ont été moins souvent sélectionnés, suivant leur probabilité décroissante.

— **Évolution moyenne de la fitness**



Ce graphique illustre l'évolution moyenne de la fitness au cours des itérations. La courbe montre une progression rapide vers l'optimum global, atteignant une fitness maximale de 1000. Cela souligne l'efficacité de la roulette adaptative pour sélectionner dynamiquement les opérateurs les plus performants, contribuant à une amélioration régulière des solutions.

— **Évolution moyenne des probabilités des opérateurs**



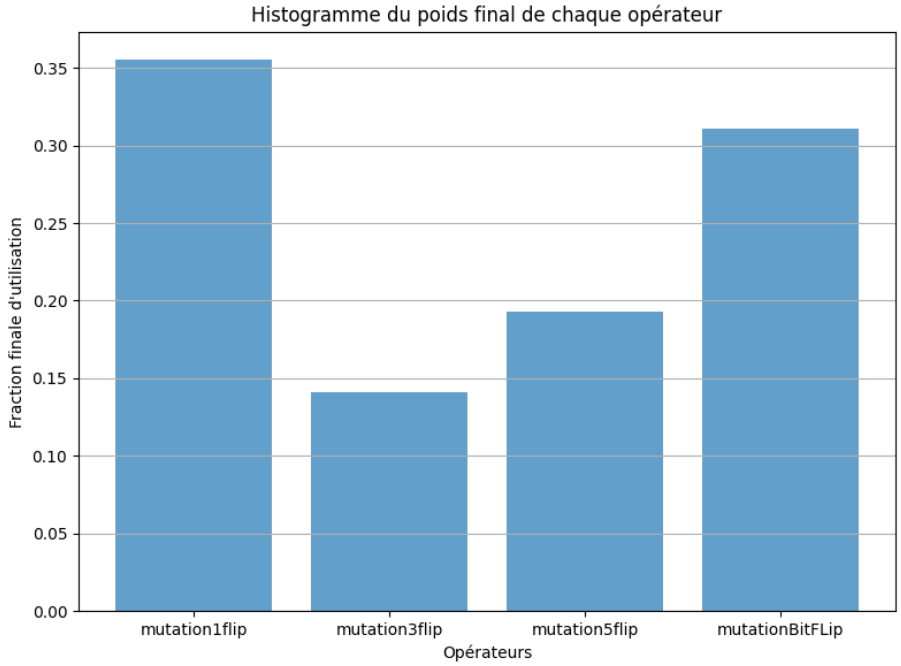
Ce graphique représente l'évolution des probabilités moyennes attribuées aux différents opérateurs au fil des itérations. On observe qu'au début, `mutation5flip` est privilégié car il permet d'augmenter rapidement la fitness lorsque la solution est éloignée de l'optimum. Ensuite, `mutation3flip` prend le relais avant que `mutation1flip` et `mutationBitFlip` deviennent dominants. Ce comportement adaptatif met en avant l'efficacité de la roulette pour privilégier successivement les opérateurs en fonction de leur pertinence à différents stades de la recherche.

La **roulette adaptative** s'avère être une approche robuste qui permet de s'adapter dynamiquement à l'évolution de l'exploration de l'espace de re-

cherche au cours du temps. En ajustant les probabilités en fonction de la performance des opérateurs, elle favorise une transition fluide entre exploration large et exploitation fine, contribuant ainsi à une convergence efficace vers l'optimum global.

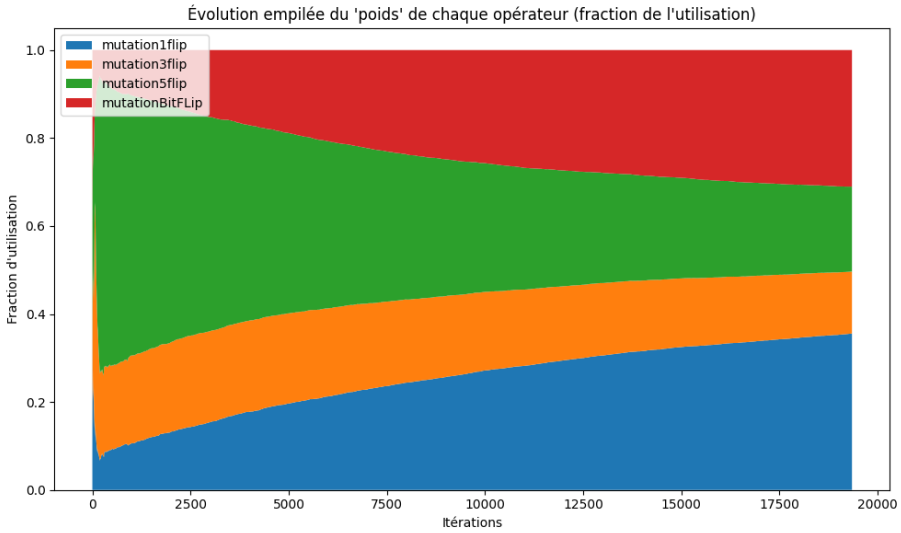
3.3.3 UCB (Upper Confidence Bound)

— **Histogramme du poids final de chaque opérateur**



Cet histogramme représente la fraction finale d'utilisation de chaque opérateur après l'exécution avec UCB. Les opérateurs `mutation1flip` et `mutationBitFlip` sont les plus utilisés, soulignant leur efficacité dans la recherche de l'optimum. Les opérateurs `mutation3flip` et `mutation5flip`, moins performants, ont une utilisation plus faible.

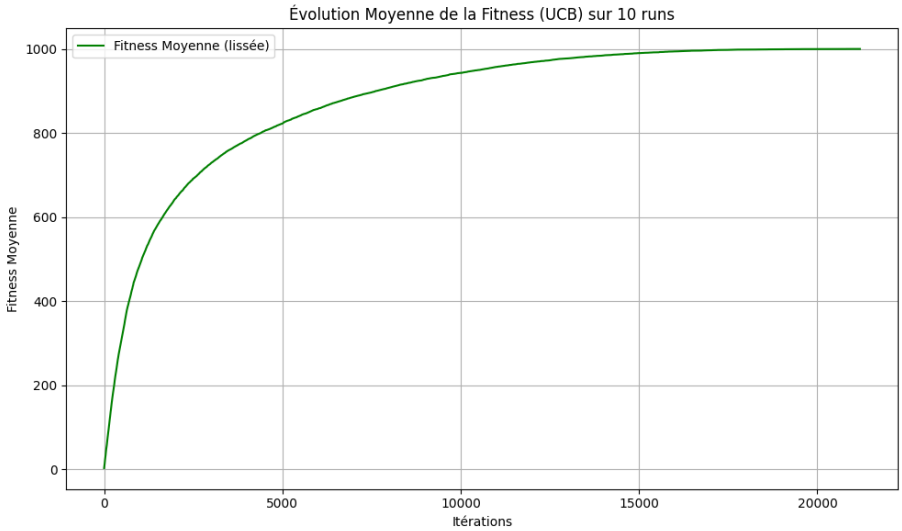
— **Évolution empilée des poids des opérateurs (fraction d'utilisation)**



Ce graphique montre comment les fractions d'utilisation des différents opérateurs évoluent au fil des itérations. Au départ, les opérateurs exploratoires comme `mutation5flip` et `mutation3flip` sont privilégiés pour

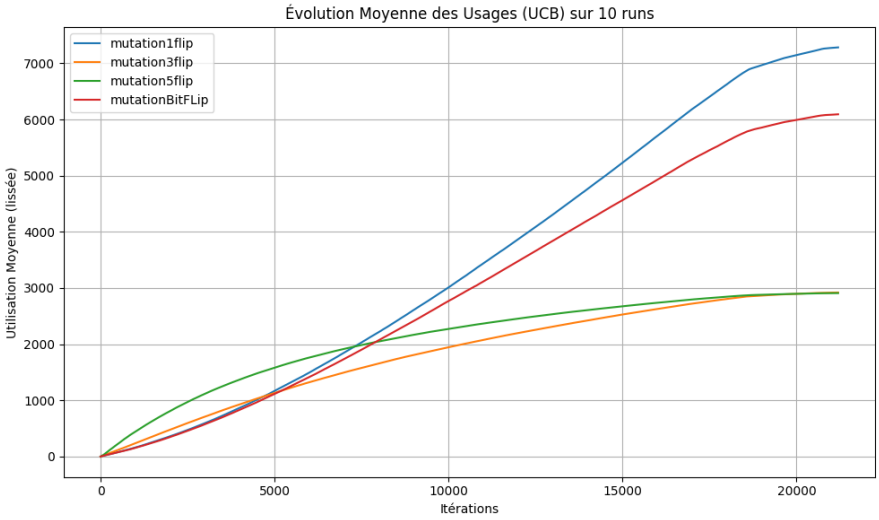
accélérer la progression. Progressivement, les opérateurs `mutation1flip` et `mutationBitFlip`, plus précis, dominent, reflétant l'adaptation de l'algorithme UCB pour équilibrer exploration et exploitation.

— **Évolution moyenne de la fitness**



Ce graphique illustre la progression moyenne de la fitness au fil des itérations. L'approche UCB permet une amélioration régulière, atteignant finalement l'optimum global. Cela démontre son efficacité pour guider dynamiquement la sélection des opérateurs.

— **Évolution moyenne des usages des opérateurs**

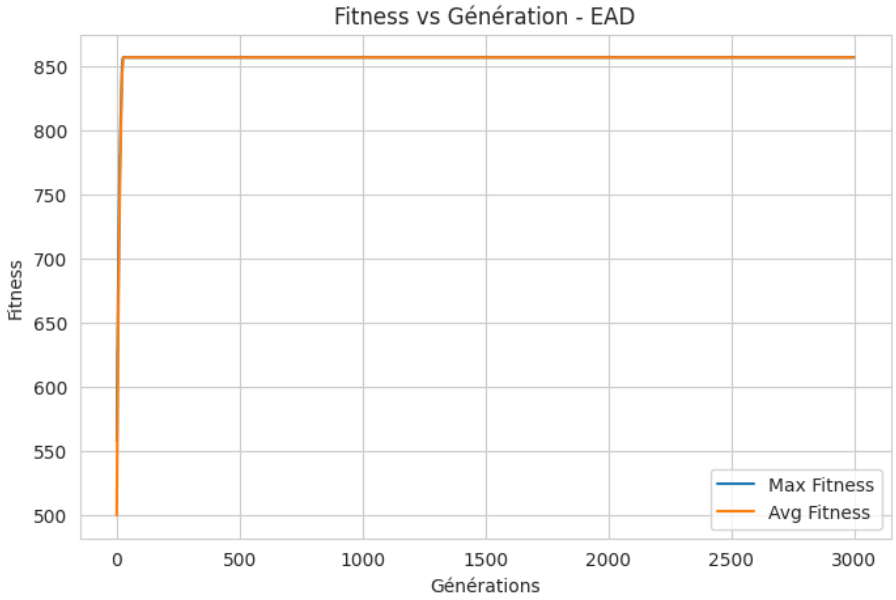


Ce graphique montre l'évolution du nombre d'utilisations des différents opérateurs. Les opérateurs `mutation1flip` et `mutationBitFlip` voient leur fréquence augmenter au fil du temps, en cohérence avec leur performance. Les opérateurs moins performants, comme `mutation3flip` et `mutation5flip`, sont progressivement délaissés.

En conclusion, l'approche UCB démontre sa robustesse en équilibrant efficacement exploration et exploitation pour sélectionner dynamiquement les opérateurs les plus performants, ce qui conduit à une convergence efficace vers l'optimum global.

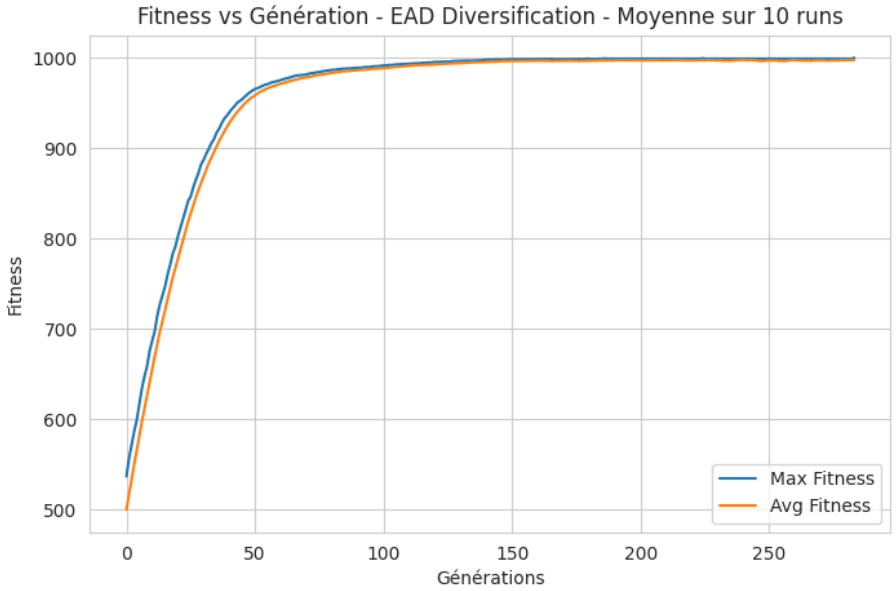
3.3.4 Algorithme basé sur Estimation de Distribution (EAD)

— **Fitness vs Génération sans Diversification**



Ce graphique illustre l'évolution de la fitness sans stratégie de diversification. On constate que l'algorithme se bloque autour d'une fitness de 850, et aucune amélioration n'est possible. Cela s'explique par le fait que certaines probabilités tombent à zéro et ne sont jamais réajustées, empêchant l'algorithme d'explorer de nouvelles solutions et d'atteindre l'optimum global.

— **Fitness vs Génération avec Diversification**



Ce graphique représente l'évolution de la fitness maximale et moyenne au fil des générations lorsque des stratégies de diversification sont appliquées. On observe une convergence rapide vers l'optimum global, avec la fitness moyenne et maximale atteignant 1000. Cela montre que la diversification permet d'éviter les blocages dans des solutions sous-optimales en maintenant la probabilité de générer des solutions variées.

L'ajout de stratégies de diversification dans l'EAD est essentiel pour maintenir l'exploration dans l'espace de recherche. Sans diversification, l'al-

gorithme peut se bloquer prématurément sur des solutions sous-optimales en raison de probabilités nulles non réajustées, limitant ainsi sa capacité à converger vers l'optimum global. Il est à noté que ce type de méthode est particulièrement efficace dans des problème d'optimisation combinatoire lorsque qu'il n'y a pas d'interaction entre les variables.

3.4 Modèle en îles

L'analyse des différents hyperparamètres (**alpha**, **beta**, **niveau de bruit**) révèle des tendances cohérentes dans l'évolution des tailles de population par île au cours des générations. Les résultats suivent toujours un même schéma :

- **Évolution des tailles de population par île :**
  - Initialement, la majorité des individus se trouve sur l'île associée à `mutation5flip`, favorisant une exploration large.
  - Progressivement, les individus migrent vers les îles `mutation3flip` et `mutation1flip`, en raison de l'adaptation dynamique qui favorise les solutions exploitables à mesure que l'algorithme converge.
  - L'île associée à `mutationBitFlip` maintient une population réduite mais stable, indiquant son rôle de précision locale à la fin de l'optimisation.
- **Comportement global des hyperparamètres :**
  - La valeur de **alpha** influence l'intensité des transitions entre les îles, tandis que **beta** contrôle la persistance des populations sur les îles initiales.
  - Le niveau de bruit module la stabilité des courbes, affectant la variabilité entre les exécutions.

Illustrations des résultats : Les graphiques ci-dessous montrent les courbes d'évolution des tailles de population moyenne sur 20 runs, pour différents paramètres (**alpha**, **beta**, **noise**). Chaque graphique illustre les mêmes dynamiques, adaptées aux valeurs des hyperparamètres.

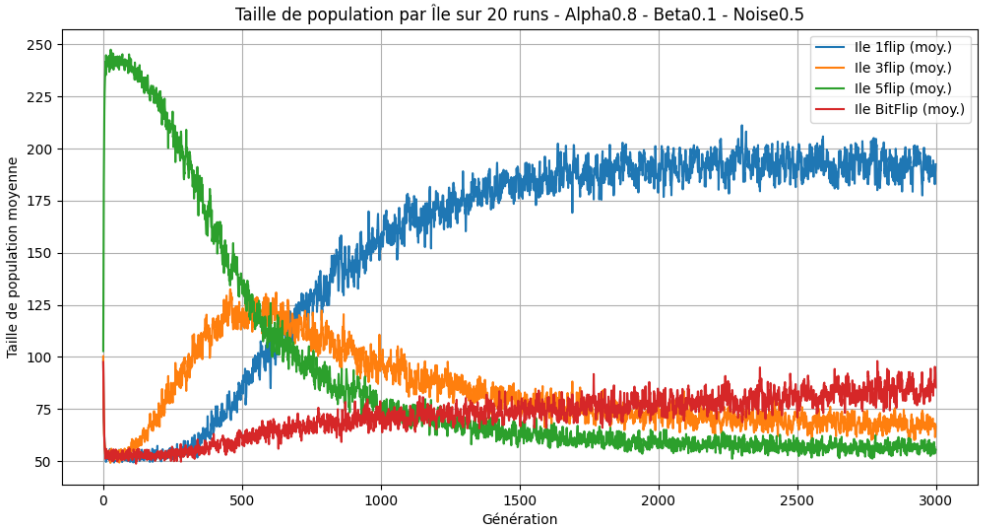


FIGURE 1 – Taille de population par île pour **noise = 0.5**.

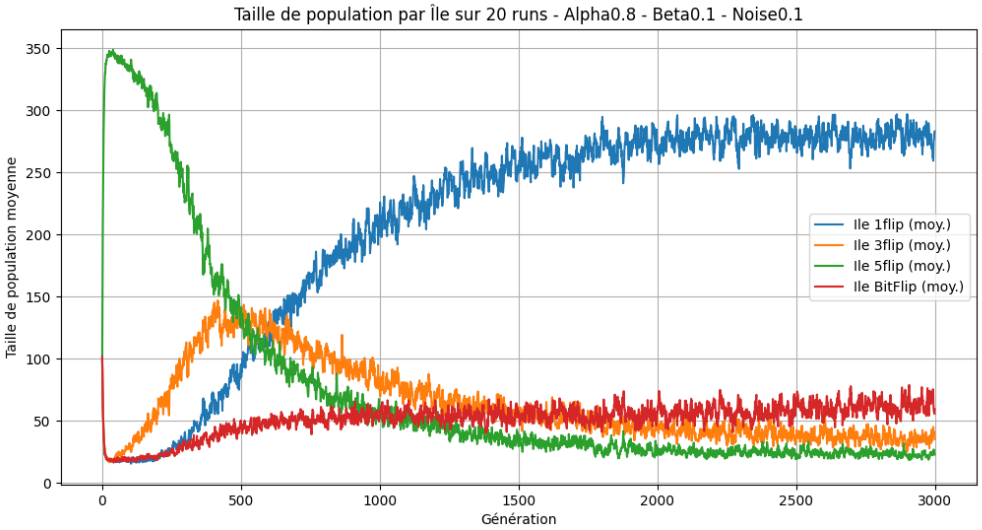


FIGURE 2 – Taille de population par île pour **noise = 0.1**.

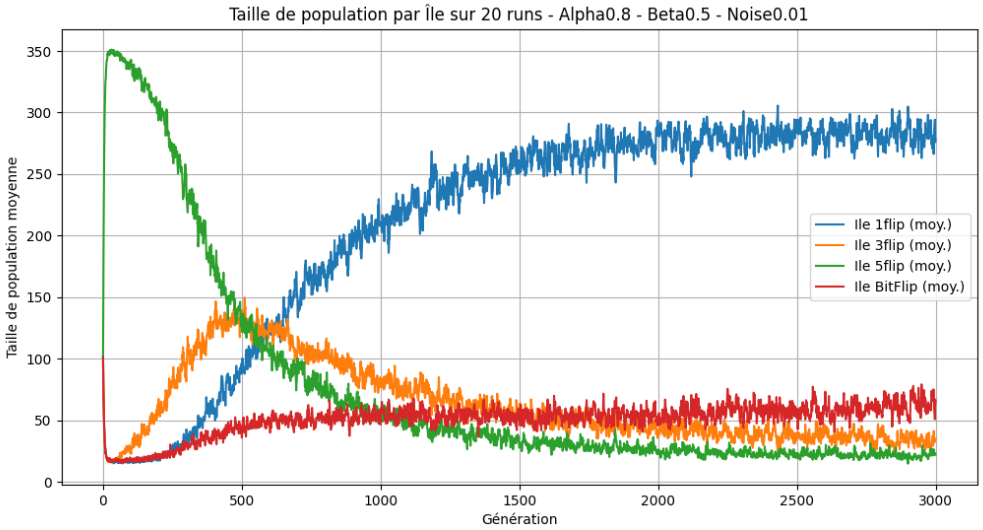


FIGURE 3 – Taille de population par île pour **alpha = 0.8**, **beta = 0.5**.

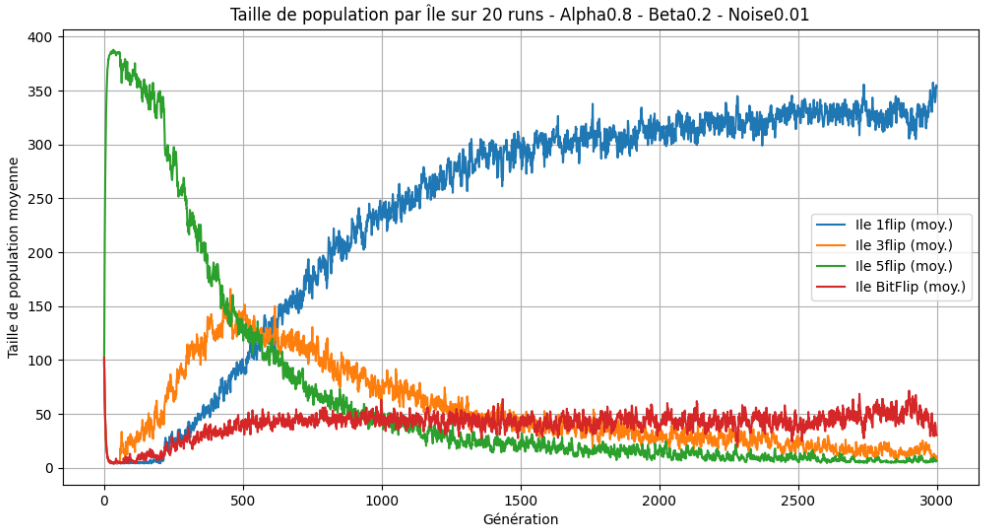


FIGURE 4 – Taille de population par île pour **alpha = 0.8**, **beta = 0.2**.

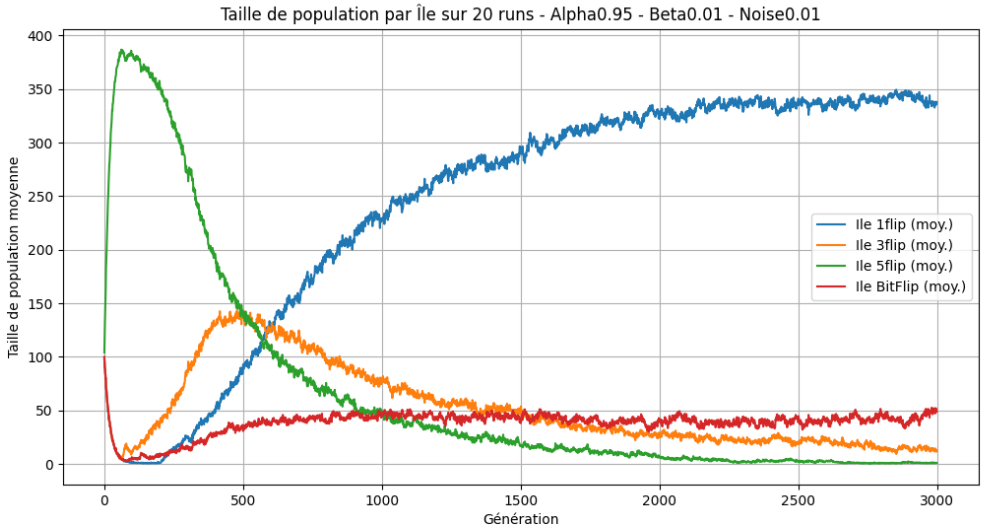


FIGURE 5 – Taille de population par île pour **alpha = 0.95**, **beta = 0.01**.

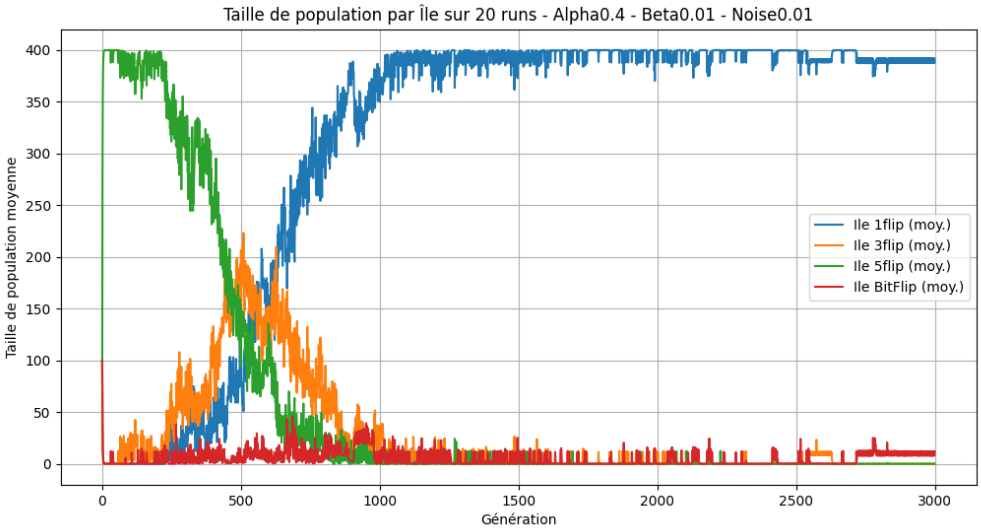


FIGURE 6 – Taille de population par île pour **alpha = 0.4**, **beta = 0.01**.

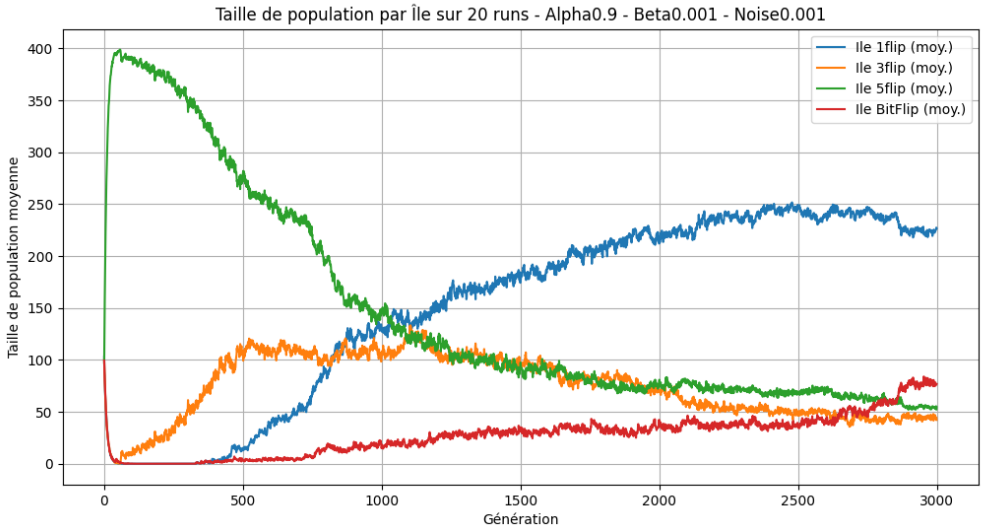


FIGURE 7 – Taille de population par île pour **alpha = 0.9**, **beta = 0.001**.

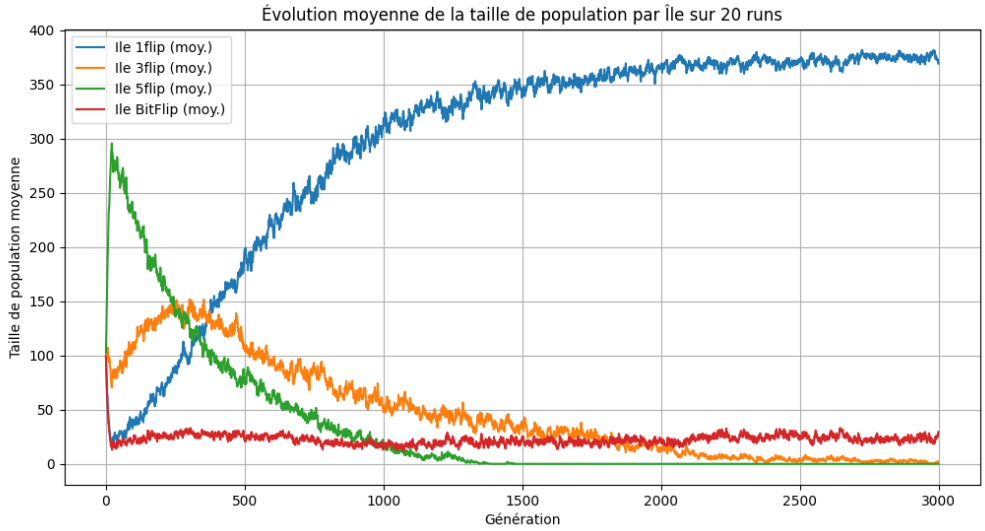


FIGURE 8 – Taille de population par île pour **alpha = 0.8**, **beta = 0.01**.

Synthèse : Les dynamiques observées mettent en évidence la capacité du modèle en îles à s'adapter aux stades successifs de la recherche. Les populations migrent naturellement vers les îles favorisant l'exploitation des solutions. Ces tendances sont robustes face aux variations des paramètres, confirmant la pertinence du modèle pour des problèmes combinatoires.

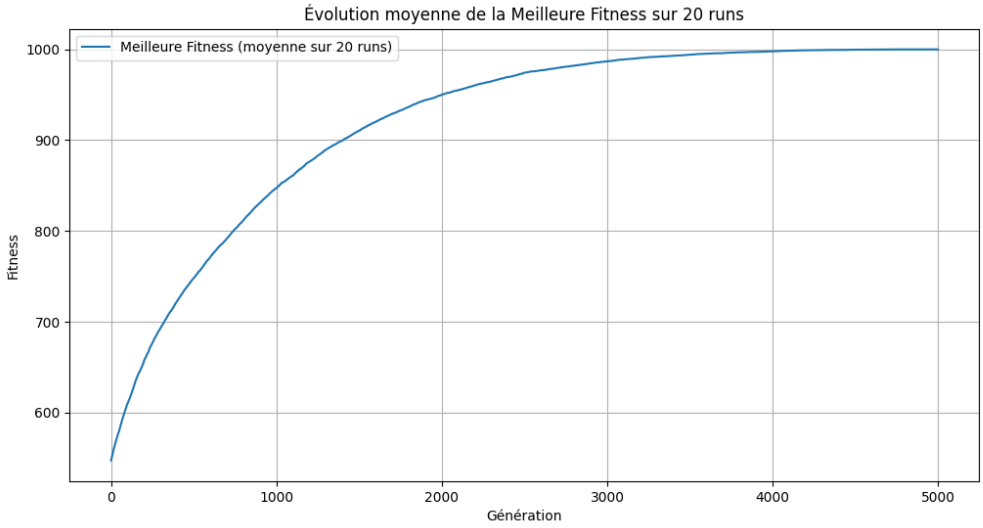


FIGURE 9 – Evolution de la fitness à travers les générations

3.5 Comparaison

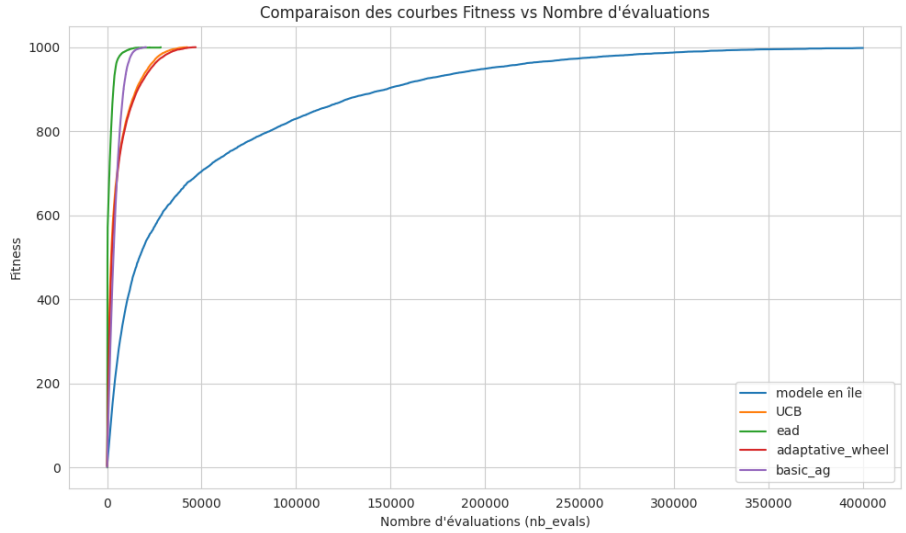


FIGURE 10 – Comparaison des courbes Fitness vs Nombre d'évaluations

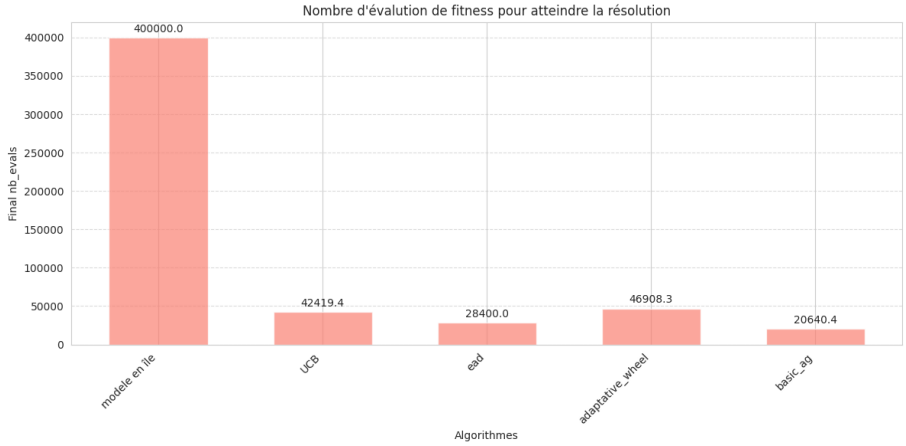


FIGURE 11 – Nombre d'appels à la fonction fitness pour atteindre la résolution

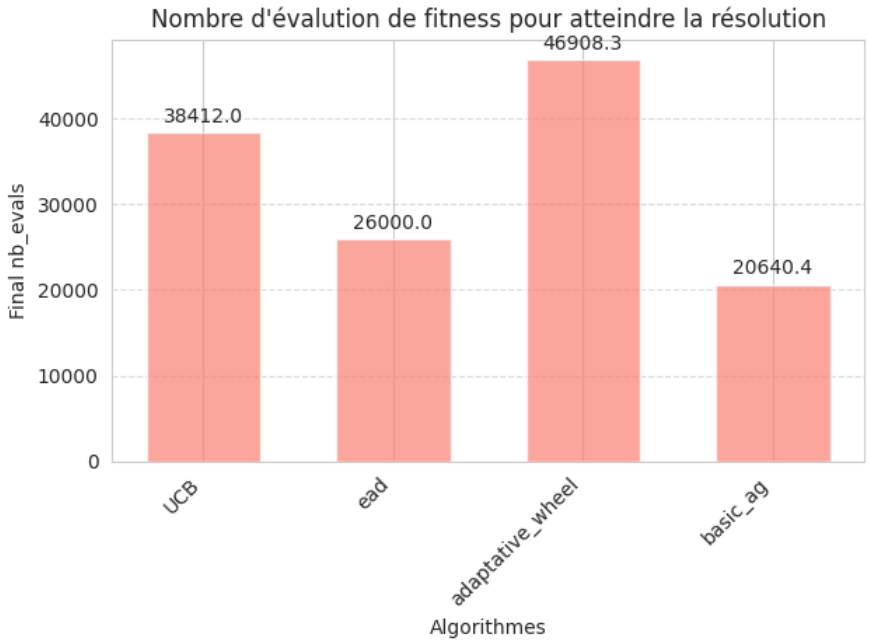


FIGURE 12 – Nombre d'appels à la fonction fitness sans le modèle en île

Algorithm	**Final Generation**	**Final nb_evals**
modele en île	4999	400000.0
UCB	24040	42419.4
ead	283	28400.0
Roulette adaptative	29926	46908.3
Steady-state	12410	20640.4

TABLE 1 – Comparaison des performances des algorithmes

Analyse : Le critère principal pour évaluer les algorithmes est le nombre d'appels à la fonction de fitness, car le nombre de générations n'est pas assez pertinent. En effet, certains algorithmes, comme **EAD**, effectuent peu de générations mais appellent la fonction de fitness pour tous les individus à chaque génération.

Les courbes 10 et 12 montrent que :

— Les algorithmes **EAD** et **Steady-State** sont les plus performants en termes de nombre d'appels à la fonction de fitness pour converger rapidement vers la solution optimale.

— Le **modèle en île** effectue un très grand nombre d'appels à la fonction de fitness en raison de sa stratégie de recherche locale systématique lors de chaque mutation. Cela entraîne un coût computationnel élevé.

Malgré ses coûts élevés, le **modèle en île** est solide et met en évidence l'importance d'utiliser différentes stratégies de mutation de manière adaptative pour optimiser la recherche au cours du temps.

4 Conclusion

Ce rapport met en lumière l'efficacité et les limitations de différentes approches adaptatives appliquées à l'optimisation combinatoire, en particulier

sur le problème One Max. Chaque méthode possède ses forces et ses faiblesses, et leur performance dépend fortement des paramètres choisis et des caractéristiques spécifiques du problème traité.

Les analyses montrent que des algorithmes comme **EAD** (Estimation de Distribution Algorithm) se démarquent par leur simplicité et leur robustesse, faisant d'eux une alternative solide pour des problèmes comme le One Max. Leur capacité à converger rapidement vers l'optimum global tout en limitant le nombre d'appels à la fonction de fitness les rend particulièrement adaptés aux scénarios où l'efficacité computationnelle est cruciale.

D'autres approches, comme le **modèle en île**, bien que plus coûteuses en termes d'évaluations de fitness, illustrent l'importance de stratégies diversifiées de mutation et d'exploration pour maintenir une bonne robustesse sur des problèmes plus complexes. Ce modèle met en avant la flexibilité d'une recherche locale systématique tout en soulignant les compromis entre précision et coût.

La performance des méthodes n'est pas absolue : elle dépend largement de leur ajustement au problème spécifique, des hyperparamètres choisis et de l'équilibre entre exploration et exploitation. Ainsi, il est essentiel de tester la robustesse de chaque modèle, d'explorer une variété de paramètres et de les ajuster selon les besoins du problème à résoudre.

En conclusion, bien que certaines approches comme l'**EAD** puissent s'avérer particulièrement efficaces pour des problèmes simples comme le One Max, chaque méthode possède un potentiel unique qui peut être exploité en fonction des objectifs spécifiques et des contraintes du problème. il est importance d'adopter une approche méthodique pour choisir et paramétrer les algorithmes d'optimisation combinatoire.

Références

[CGLS12] Caner Candan, Adrien Goeffon, Frédéric Lardeux, and Frédéric Saubion. A dynamic island model for adaptive operator selection. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, page 1253–1260, New York, NY, USA, 2012. Association for Computing Machinery.

[CGLS13] Caner Candan, Adrien Goëffon, Frédéric Lardeux, and Frédéric Saubion. Non stationary operator selection with island models. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, page 1509–1516, New York, NY, USA, 2013. Association for Computing Machinery.

[EMSS07] Aguston Eiben, Zbigniew Michalewicz, Marc Schoenauer, and Jim Smith. Parameter Control in Evolutionary Algorithms. In Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer Verlag, 2007. http ://www.springerlink.com/content/978-3-540-69431-1/.

[GL12] Adrien Goëffon and Frédéric Lardeux. Autonomous local search algorithms with island representation. In *Autonomous Local Search*

Algorithms with Island Representation, pages 390–395, 01 2012.

[WM97] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1) :67–82, 1997.

[EMSS07] [WM97] [GL12] [CGLS13] [CGLS12]