# Algorithmes Génétiques

Omar Arharbi

Janvier 2025

## Table des matières

# 1 Introduction

## 1.1 Algorithmes évolutionnaires

Les algorithmes évolutionnaires représentent une approche bio-inspirée de l'optimisation. Ils constituent une méthode d'optimisation innovante, puisant son inspiration dans les principes de l'évolution biologique. Cette approche vise à résoudre des problèmes complexes en simulant le processus d'évolution naturelle.

### 1.1.1 Processus itératif

L'algorithme fonctionne de manière itérative, suivant ces étapes clés :
— Initialisation : Création d'une population initiale diverse.
— Évaluation : Mesure de la qualité (fitness) de chaque individu.
— Sélection : Choix des individus les plus prometteurs.
— Reproduction : Création de nouveaux individus par croisement.
— Mutation : Introduction de variations aléatoires.
— Remplacement : Intégration des nouveaux individus dans la population.

### 1.1.2 Opérateurs clés

— Sélection : Identifie les meilleurs individus, souvent par des méthodes comme la
— roulette ou le tournoi.
— Croisement : Combine les caractéristiques de deux parents pour créer des descendants.
— Mutation : Apporte des modifications aléatoires pour maintenir la diversité.

### 1.1.3 Objectif et arrêt

L'algorithme vise à maximiser une fonction objective, représentant la qualité des solutions. Il s'arrête lorsqu'un critère prédéfini est atteint, comme un nombre maximal d'itérations ou l'obtention d'une solution satisfaisante.

### 1.1.4 Équilibre crucial

Le succès de l'algorithme repose sur un équilibre délicat entre l'exploitation des meilleures solutions trouvées et l'exploration de nouvelles possibilités, guidé par le choix judicieux des opérateurs.

## 1.2 Le problème One-Max

Le problème One-Max est un exemple classique utilisé pour illustrer le fonctionnement des algorithmes génétiques. Il se caractérise par sa simplicité et son efficacité pédagogique [3].

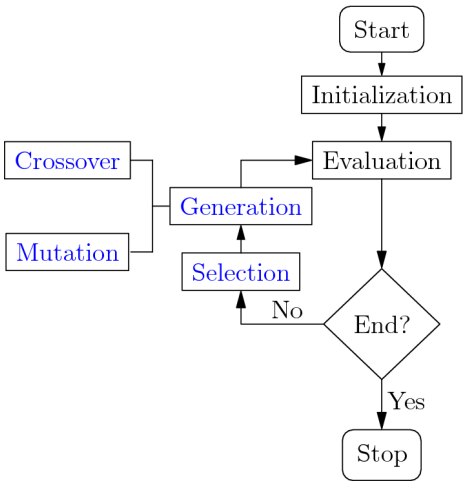



FIGURE 1 – Genetic algorithm process

### 1.2.1 Définition du problème

Soit une liste binaire $L$ de longueur $N$, où chaque élément $l_i \in \{0, 1\}$.
La fonction de fitness $f(L)$ est définie comme :

$$f(L) = \sum_{i=1}^{N} l_i$$

### 1.2.2 Objectif

L'objectif est de maximiser la valeur de fitness, c'est-à-dire trouver la liste $L$ qui produit la plus grande somme possible.

### 1.2.3 Exemple

Pour $N = 10$, considérons la liste suivante générée aléatoirement :

$$L = [0, 1, 0, 0, 1, 1, 1, 0, 0, 0]$$

Sa valeur de fitness est :

$$f(L) = 0 + 1 + 0 + 0 + 1 + 1 + 1 + 0 + 0 + 0 = 4$$

### 1.2.4 Solution optimale

La solution optimale pour $N = 10$ est :

$$L_{opt} = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

avec une fitness maximale :

$$f(L_{opt}) = 10$$

#### 1.2.5 Rôle de l'algorithme génétique

Un algorithme génétique efficace devrait converger vers cette solution optimale, produisant des individus (listes) composés uniquement de 1, atteignant ainsi la fitness maximale égale à $N$.

# 2 Présentation des trois méthodes étudiées

Afin de résoudre le problème du One-Max, nous avons implémenté différents algorithmes évolutionnaires.

## 2.1 Algorithme génétique steady state

L'algorithme steady state est conçu pour choisir un nombre spécifique d'individus à chaque génération, généralement deux, mais ce nombre peut être étendu. L'initialisation de la population peut se faire de manière fixe ou aléatoire.

### 2.1.1 Méthodes de sélection

Pour sélectionner N individus, plusieurs approches sont possibles :
- **Sélection par roulette :** La probabilité de sélection est proportionnelle à la valeur de fitness.
- **Sélection par tournoi :** Des tournois de taille k sont organisés, les vainqueurs étant sélectionnés.
- **Sélection par classement :** Les N individus ayant les meilleures performances sont retenus.
- **Sélection aléatoire :** N individus sont choisis au hasard dans la population.

### 2.1.2 Techniques de croisement

Les individus sélectionnés subissent un croisement selon diverses méthodes :
- **Croisement mono-point :** Un point d'échange est déterminé aléatoirement.
- **Croisement bi-point :** Deux points d'échange sont choisis au hasard.
- **Croisement uniforme :** Chaque caractéristique est héritée indépendamment d'un parent ou de l'autre.

### 2.1.3 Opérations de mutation

Après le croisement, chaque individu a une probabilité P de subir une mutation :
- **Mutation bit-flip :** Chaque bit peut être modifié avec une probabilité P.
- **Mutation one-flip :** Un seul bit est modifié aléatoirement.
- **Mutation k-flip :** k bits sont sélectionnés et modifiés au hasard.

### 2.1.4 Renouvellement de la population

L'intégration des nouveaux individus dans la population existante peut se baser sur différents critères, tels que l'ancienneté ou la performance des individus.

### 2.1.5 Critères d'arrêt

L'algorithme se termine selon des conditions prédéfinies, comme un nombre maximal de générations ou l'atteinte d'une solution satisfaisante.

### 2.1.6 Pseudo code

**Algorithm 1** Steady-State Genetic Algorithm

**Input:** *population_size*, *max_gen*, *p_mutation*, *p_crossover*
**Output:** Best individual found
Initialize population $P$ with *population_size* random individuals
Evaluate fitness of each individual in $P$
**for** $gen = 1$ to *max_gen* **do**
    Select two parents $p_1, p_2$ using a selection methods
    **if** $rand() < p\_crossover$ **then**
        Apply crossover to produce offspring $c_1, c_2$
    **end if**
    **if** $rand() < p\_mutation$ **then**
        Apply mutation on $c_1$ and $c_2$
    **end if**
    Evaluate fitness of $c_1, c_2$
    Replace the two worst individuals in $P$ with $c_1$ and $c_2$
**end for**
**return** best individual in $P$

## 2.2 Algorithme à estimation de distribution

### 2.2.1 Définition

L'algorithme à estimation de distribution vise à estimer la distribution de probabilité des solutions dans l'espace de recherche.

### 2.2.2 Exemple

Voici un exemple détaillé pour N = 4 et K = 3 :

1. Vecteur de distribution initiale : [0.5, 0.3, 0.7, 0.4]
2. Population initiale générée :
   — Individu 1 : [1, 0, 1, 0] → Fitness = 2
   — Individu 2 : [1, 1, 1, 0] → Fitness = 3
   — Individu 3 : [0, 0, 1, 1] → Fitness = 2
   — Individu 4 : [1, 0, 1, 1] → Fitness = 3
   — Individu 5 : [1, 1, 1, 1] → Fitness = 4

3. Sélection des K = 3 meilleurs individus :
   1, 1, 1, 1 → Fitness = 4
   1, 1, 1, 0 → Fitness = 3
   1, 0, 1, 1 → Fitness = 3
4. Calcul du nouveau vecteur de distribution :
   — Position 1 : $(1 + 1 + 1)/3 = 1.0$
   — Position 2 : $(1 + 1 + 0)/3 = 0.66$
   — Position 3 : $(1 + 1 + 1)/3 = 1.0$
   — Position 4 : $(1 + 0 + 1)/3 = 0.66$
5. Nouveau vecteur de distribution : [1.0, 0.66, 1.0, 0.66]
6. L'algorithme continue avec ce nouveau vecteur pour générer la prochaine population, et ainsi de suite jusqu'à atteindre le critère d'arrêt.

On peut observer que le vecteur de probabilité évolue pour favoriser les bits qui apparaissent fréquemment dans les meilleures solutions.

### 2.2.3 Pseudo code

**Algorithm 1** Algorithme à Estimation de Distribution (EDA)

```
Initialiser: Vecteur de probabilité V de taille N
Initialiser: Nombre de meilleurs individus K
Initialiser: Taille de population P
Initialiser: Nombre maximum de générations G
for g ← 1 to G do
    // Génerer la population a partir du vecteur V
    for chaque individu i dans population do
        for chaque position j dans individu do
            if random() < V[j] then
                i[j] ← 1
            else
                i[j] ← 0
            end if
        end for
    end for
    Évaluer(Population)        ▷ Évaluer chaque individu de la population
    Sélectionner(Population, K)        ▷ Sélectionner les K meilleurs
    //Mise a jour de la distribution (Méttre a jour V)
    for chaque position j dans V do
        somme ← somme des bits à la position j des K meilleurs
        V[j] ← somme/K
        if V[j] = 0 then
            V[j] ← V[j] + 0.01                    ▷ Éviter convergence à 0
        end if
    end for
end for
return meilleure solution trouvée
```

## 2.3 Algorithme compact

### 2.3.1 Définition

L'algorithme génétique compact (cGA) est une variante minimaliste des algorithmes d'estimation de distribution. Il opère sur un vecteur de probabilité $V$ de taille $N$ initialisé uniformément à 0.5, représentant la probabilité de chaque bit d'être à 1. À chaque génération, seuls deux individus sont générés à partir de ce vecteur, puis comparés pour mettre à jour les probabilités.

La mise à jour du vecteur se fait en comparant le meilleur individu (winner) et le moins bon (loser). Pour chaque position $i$ :

— Si les bits sont identiques, $V[i]$ reste inchangé
— Si $winner[i] = 1$ et $loser[i] = 0 : V[i] \leftarrow V[i] + \frac{1}{N}$
— Si $winner[i] = 0$ et $loser[i] = 1 : V[i] \leftarrow V[i] - \frac{1}{N}$

### 2.3.2 Exemple

Pour $N = 6$ :

1. Vecteur initial : $V = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]$
2. Population générée :
   — Individu 1 : [1, 1, 0, 0, 1, 1] → Fitness = 4
   — Individu 2 : [0, 1, 0, 1, 0, 0] → Fitness = 2
3. Comparaison bit à bit (Ind1 = winner, Ind2 = loser) :
   — Position 1 : 1 vs 0 → $V[1] = 0.5 + \frac{1}{6}$
   — Position 2 : 1 vs 1 → $V[2]$ inchangé
   — Position 3 : 0 vs 0 → $V[3]$ inchangé
   — Position 4 : 0 vs 1 → $V[4] = 0.5 - \frac{1}{6}$
   — Position 5 : 1 vs 0 → $V[5] = 0.5 + \frac{1}{6}$
   — Position 6 : 1 vs 0 → $V[6] = 0.5 + \frac{1}{6}$
4. Nouveau vecteur : $V = [0.67, 0.5, 0.5, 0.33, 0.67, 0.67]$

Ce processus est répété jusqu'à convergence ou jusqu'à ce qu'un critère d'arrêt soit atteint.

### 2.3.3 Diagramme et Pseudo code

**Algorithm 1** Algorithme Génétique Compact (cGA)

**Require:** $N$ (longueur des individus), $G$ (nombre maximal de générations), $\alpha$ (taux d'apprentissage), $P_{\min}$ (probabilité minimale)
**Ensure:** Vecteur de probabilité $P$ optimisé

**Initialisation :**
$P \leftarrow (0.5, 0.5, \ldots, 0.5)$ (vecteur de probabilités de taille $N$)
**for** $t = 1$ **à** $G$ **do**
  **Génération de la population** :
    Générer deux individus $I_1$ et $I_2$ en échantillonnant $P$
  **Évaluation de la fitness** :
    Calculer $f(I_1)$ et $f(I_2)$
  **Sélection** :
  **if** $f(I_1) \geq f(I_2)$ **then**
    winner $\leftarrow I_1$, loser $\leftarrow I_2$
  **else**
    winner $\leftarrow I_2$, loser $\leftarrow I_1$
  **end if**
  **Mise à jour du vecteur de probabilités** :
  **for** $j = 1$ **à** $N$ **do**
    **if** winner$[j] \neq$ loser$[j]$ **then**
      **if** winner$[j] = 1$ **then**
        $P[j] \leftarrow \min(1, P[j] + \frac{\alpha}{N})$
      **else**
        $P[j] \leftarrow \max(P_{\min}, P[j] - \frac{\alpha}{N})$
      **end if**
    **end if**
  **end for**
**end for**
**return** $P$

## 2.4 Comparaison/Différences/ressemblances entre les approches

| Critère | Steady-State GA | EDA | cGA |
|---|---|---|---|
| **Population** | Explicite et évolutive | Explicite, remplacée progressivement | Implicite (vecteur de probabilité) |
| **Sélection** | Tournoi, élitisme | Modèle statistique | Duel entre individus |
| **Variation** | Croisement + mutation | Échantillonnage d'une distribution | Mise à jour du vecteur de probabilité |
| **Mémoire utilisée** | Élevée (garde une population) | Moyenne | Très faible |
| **Convergence** | Progressive | Rapide si bonne estimation | Peut osciller, dépend du taux d'apprentissage |

Ces trois algorithmes partagent l'objectif d'optimisation évolutionnaire mais

diffèrent dans leur approche de gestion de la population. L'algorithme génétique steady-state est le plus "classique" : il maintient une population constante et ne remplace que quelques individus à chaque génération via les opérateurs traditionnels (croisement et mutation). L'algorithme génétique compact (cGA) représente l'autre extrême : il simule une population via un simple vecteur de probabilités, sans maintenir de population réelle, ce qui le rend très économe en mémoire. L'algorithme à estimation de distribution (EDA) se situe entre les deux : comme le cGA, il utilise un modèle probabiliste, mais il est plus sophistiqué car il capture les dépendances entre les variables et maintient une vraie population. Le steady-state modifie sa population progressivement, tandis que l'EDA génère une population entièrement nouvelle à chaque génération en échantillonnant son modèle probabiliste. Le cGA peut être vu comme un cas particulier d'EDA avec une population minimale et un modèle probabiliste très simple.

# 3 Analyse des Algorithmes Génétiques

Pour évaluer l'efficacité et le comportement des algorithmes génétiques, nous adoptons une approche méthodique consistant à modifier un seul paramètre à la fois. Cette méthode nous permet d'isoler l'impact spécifique de chaque variable sur les performances de l'algorithme. Afin d'établir une base de comparaison cohérente, nous avons défini un ensemble de paramètres par défaut :

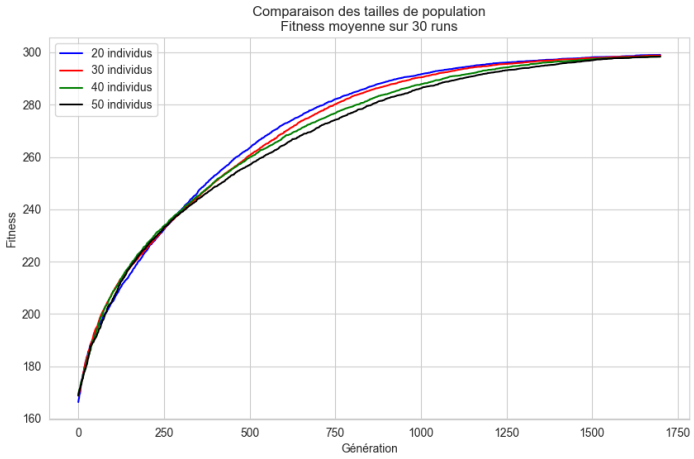
- **Structure des individus :** Chaque individu est représenté par une chaîne de 300 bits.
- **Taille de la population :** L'algorithme opère sur un ensemble de 20 individus.
- **Opérations génétiques :**
  - Le croisement est systématiquement appliqué (probabilité de 100%).
  - La mutation est également appliquée à chaque génération (probabilité de 100%).
- **Robustesse statistique :** Chaque configuration est exécutée 30 fois pour assurer la fiabilité des résultats.
- **Génération initiale :** La population de départ est créée de manière aléatoire.
- **Sélection parentale :** À chaque itération, deux individus sont choisis pour la reproduction.
- **Opérateurs génétiques par défaut :**
  - *Sélection :* Tournoi impliquant 3 individus.
  - *Croisement :* Méthode uniforme.
  - *Mutation :* Technique du "bit-flip".
- **Stratégie de remplacement :** Les nouveaux individus remplacent les moins performants de la génération précédente, favorisant ainsi le renouvellement basé sur la qualité plutôt que sur l'âge.

Ces paramètres constituent notre configuration de référence. Dans notre analyse, nous ferons varier un seul de ces paramètres à la fois, tout en maintenant

les autres constants. Cette approche nous permettra d'évaluer précisément l'influence de chaque facteur sur les performances globales de l'algorithme génétique.

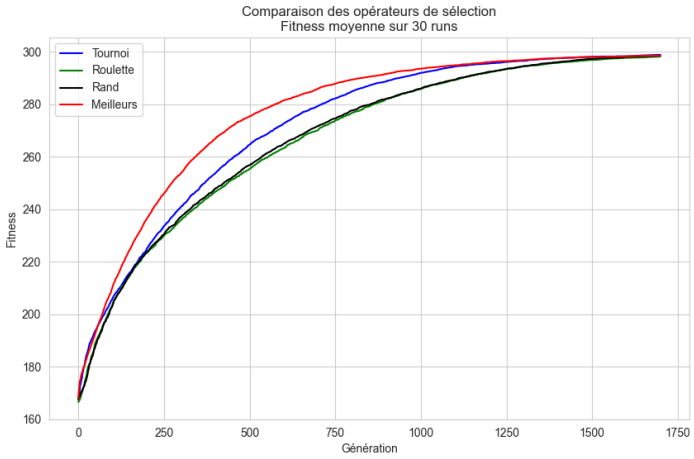
## 3.1 Algorithme steady state

### 3.1.1 la taille de la population




L'analyse des résultats révèle une corrélation inverse entre la taille de la population et la vitesse de convergence vers une solution optimale. Les populations de taille réduite tendent à atteindre plus rapidement un point de convergence. Ce phénomène s'explique par le processus de sélection : dans un groupe restreint, la probabilité de choisir les individus les plus performants augmente significativement. Par conséquent, les caractéristiques génétiques favorables se propagent plus efficacement aux générations subséquentes.

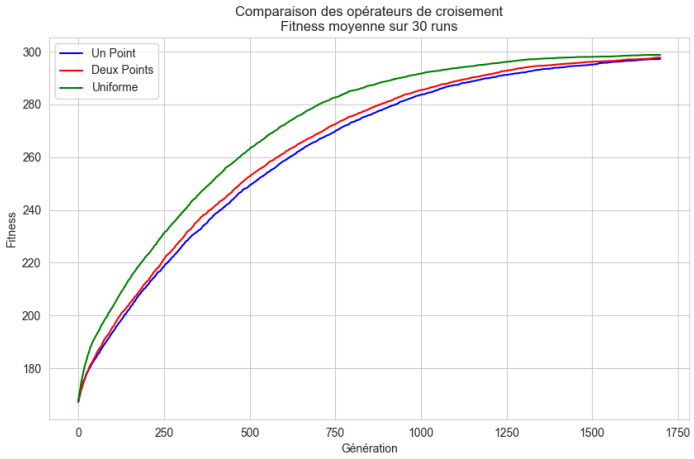
### 3.1.2 Méthodes de séléction




— **Sélection des Meilleurs** montre la convergence la plus rapide, ce qui est logique car elle sélectionne systématiquement les meilleurs individus. Cependant, cette approche pourrait limiter la diversité génétique.
— **Sélection par Tournoi** offre une performance intermédiaire avec une convergence régulière. Elle maintient un bon équilibre entre exploitation des bonnes solutions et exploration.
— **Sélection par Roulette** et **Sélection Aléatoire** montrent des performances similaires et plus lentes, mais finissent par atteindre des résultats comparables aux autres méthodes après 1500 générations.

**Conclusion** Si l'objectif est d'obtenir rapidement de bons résultats, la sélection des meilleurs est préférable. Cependant, la sélection par tournoi offre un meilleur compromis entre vitesse de convergence et maintien de la diversité. Les méthodes de roulette et aléatoire, bien que plus lentes, peuvent être utiles dans des problèmes où l'exploration de l'espace de recherche est cruciale.

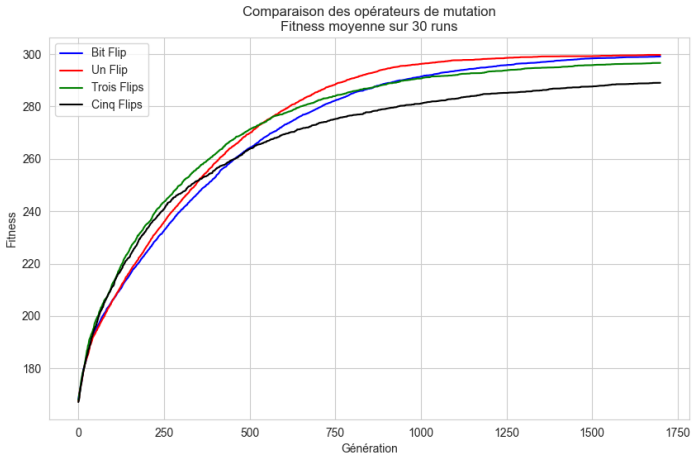
### 3.1.3 Méthodes de croisement




— **Croisement Uniforme** montre la convergence la plus rapide. Cette performance supérieur peut s'expliquer par sa capacité à échanger des bits de manière plus uniforme sur l'ensemble de la chaîne.
— **Croisement Deux Points** offre une performance intermédiaire, légèrement meilleure que le croisement un point mais moins efficace que le croisement uniforme.
— **Croisement Un Point** présente la convergence la plus lente, bien qu'il finisse par atteindre des résultats similaires aux autres méthodes après 1500 générations.

**Conclusion** Le croisement uniforme apparaît comme la méthode la plus efficace pour ce problème, probablement parce qu'il permet une meilleure exploitation des bonnes solutions en offrant plus de possibilités de combinaisons entre les parents. Les croisements à un et deux points, bien que moins performants initialement, restent viables car ils atteignent finalement la même qualité de solution.

### 3.1.4 Méthodes de mutations




— **Un Flip** (mutation d'un seul bit) montre une convergence légèrement plus rapide jusqu'à la génération 750 et atteint la meilleure performance finale.
— **Bit Flip** (mutation probabiliste de chaque bit) présente une performance similaire à Un Flip, avec une convergence un peu plus lente initialement mais rattrapant son retard après 1250 générations.
— **Trois Flips** offre une performance intermédiaire, légèrement inférieure aux deux premières méthodes.
— **Cinq Flips** montre la performance la plus faible, avec un écart significatif par rapport aux autres méthodes qui se maintient jusqu'à la fin.

**Conclusion** On observe une corrélation claire entre le nombre de bits mutés et la performance : plus le nombre de bits mutés est important, moins la performance est bonne. Cela suggère que des mutations trop importantes peuvent être déstabilisantes pour le problème OneMax. Les méthodes Un Flip et Bit Flip, qui modifient moins de bits en moyenne, semblent offrir le meilleur compromis entre exploration et exploitation.

## 3.2 Algorithme à estimation de distribution

L'algorithme à estimation de distribution offre une flexibilité intéressante dans le paramétrage du processus évolutif. Un aspect clé de cette flexibilité réside dans la possibilité d'ajuster le nombre d'individus considérés comme "élites" ou "parents" à chaque génération. Ce paramètre joue un rôle crucial dans la dynamique de l'algorithme, influençant directement la rapidité avec laquelle il converge vers une solution optimale.

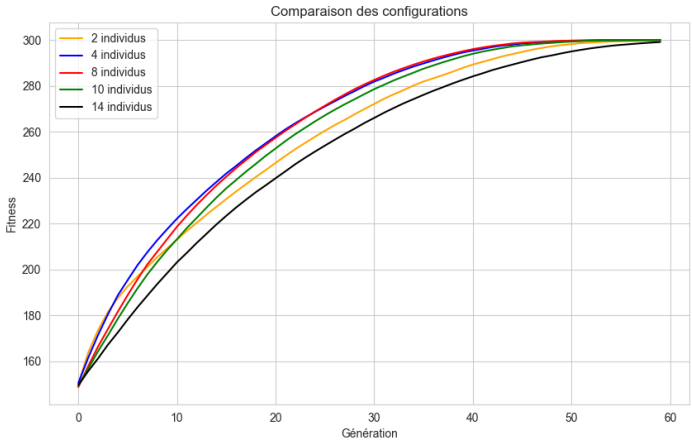
Le mécanisme central de cet algorithme repose sur la mise à jour du vecteur de probabilité. Cette mise à jour s'effectue en calculant la moyenne des carac-

téristiques (bits) des individus sélectionnés comme parents. Ainsi, le choix du nombre de parents impacte directement la façon dont l'information génétique est transmise et exploitée au fil des générations.

Un autre paramètre d'intérêt est la taille globale de la population. Il est pertinent d'examiner comment les variations de cette taille interagissent avec le nombre d'individus sélectionnés comme parents. Cette analyse permet de comprendre si une population plus large nécessite proportionnellement plus de parents pour maintenir une diversité génétique optimale, ou si un petit groupe d'élites suffit, quelle que soit la taille de la population.

Ces considérations ouvrent la voie à une exploration approfondie des dynamiques de l'algorithme, permettant d'optimiser son efficacité en fonction des spécificités du problème à résoudre.

### 3.2.1 Nombre de meilleures individus (K best)




Avec une taille de population = 30, l'analyse des performances de l'algorithme à estimation de distribution révèle des dynamiques intéressantes en fonction du nombre d'individus sélectionnés comme parents. Dans le contexte d'une population standard de 30 individus, on constate que la sélection d'un groupe de 4 à 8 parents semble offrir les meilleurs résultats.

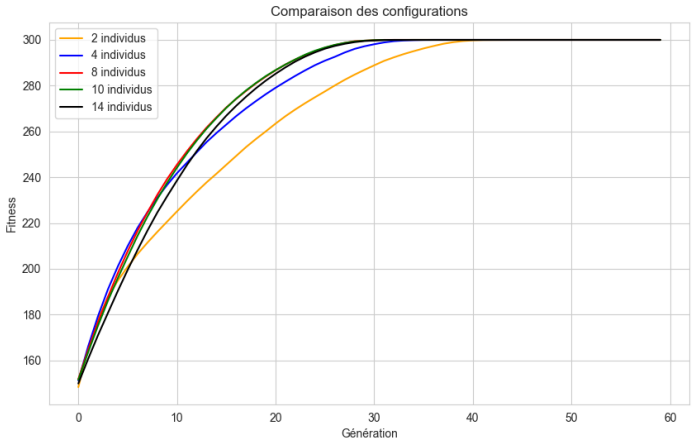
Lors des phases initiales de l'évolution, une sélection restreinte à quatre parents démontre une efficacité supérieur. Cette approche conservatrice minimise le risque d'inclure des individus sous-optimaux dans le pool génétique parental, assurant ainsi une convergence rapide vers des solutions prometteuses.

Cependant, à mesure que l'algorithme progresse, on observe une homogénéisation croissante de la population. Un nombre croissant d'individus atteint des niveaux de fitness proches du maximum observé. Dans ce contexte d'amélioration globale, l'élargissement du groupe parental à huit individus devient avantageux. Cette stratégie permet d'exploiter une plus grande diversité génétique tout en maintenant une qualité élevée des parents sélectionnés.

En revanche, l'extension du groupe parental à 14 individus s'avère contre-productive. Cette approche trop inclusive risque d'intégrer des individus de moindre qualité dans le processus de reproduction, diluant ainsi l'efficacité de la sélection et potentiellement ralentissant la convergence vers des solutions optimales.

### 3.2.2 Augmentation de taille de la population

**Taille = 90**




On remarque ici qu'ici la selection de 14 parents devient plus intéressantes.

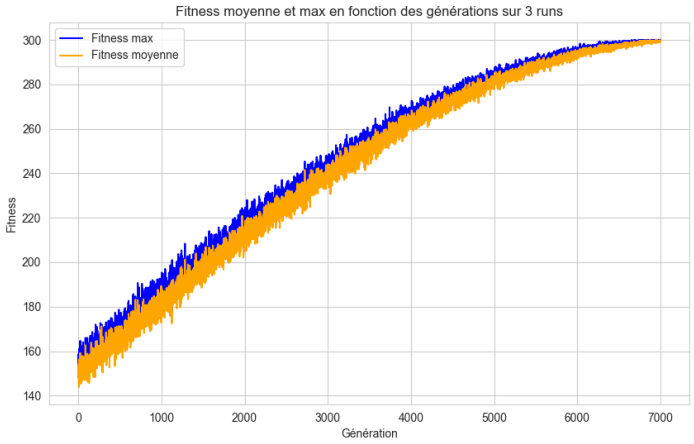
## 3.3 Algorithme génétique compact

Pour accélérer la convergence de l'algorithme, il est possible d'introduire un coefficient d'apprentissage $\alpha$. Par défaut, $\alpha = 1$, mais sa valeur peut être ajustée pour modifier la vitesse d'apprentissage. Avec ce coefficient, la formule de mise à jour devient :

$$\omega_i = \omega_i + \alpha \cdot \frac{1}{N} \cdot (winner_i - loser_i)$$

Cette formulation permet un contrôle plus fin de l'évolution du vecteur de probabilité, offrant la possibilité d'ajuster la balance entre exploration et exploitation dans l'espace de recherche, et bien sur en fonction du $\alpha$ en modifie le nombre maixmum de génération.

### 3.3.1 Taux d'apprentissage a 1 ($\alpha = 1$)

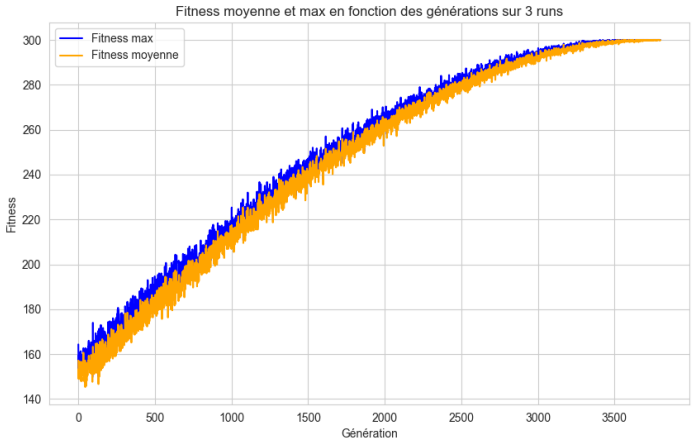



L'expérimentation montre l'évolution de la fitness sur 7000 générations avec deux métriques principales :

- **Convergence globale :** L'algorithme atteint une fitness proche de 300 (l'optimum) après environ 7000 générations, démontrant sa capacité à trouver la solution optimale, mais la progression reste lente.
- **Écart fitness max/moyenne :** L'écart entre la fitness maximale et moyenne reste relativement constant tout au long de l'évolution, indiquant une bonne stabilité de l'algorithme et une bonne diversité de population.
- **oscillations :** Les oscillations observées dans les courbes sont dues à la mise à jour progressive du vecteur de probabilité, qui détermine la génération des individus. À chaque génération, le vecteur est ajusté en fonction du gagnant et du perdant de la sélection, mais ces mises à jour ne sont pas continues ni uniformes, ce qui entraîne des fluctuations locales dans la fitness. De plus, ici on ne maintient pas une population explicite mais on ajuste uniquement des probabilités.

**Conclusion** L'algorithme montre une progression stable et constante vers l'optimum, caractéristique typique du cGA. La convergence est plus lente qu'un algorithme génétique classique et donc on a besoin d'un nombre de générations supérieur de ce qu'on a vu auparavant ce nombre qui peut diminuer en augementant le taux d'apprentissage $\alpha$.

### 3.3.2 Augmentation du taux d'apprentissage ($\alpha = 2$)




L'augmentation du taux d'apprentissage à 2 a permis une convergence plus rapide, réduisant le nombre de générations nécessaires pour atteindre la fitness maximale de 300 or ici on a eu besoin que de 3500 générations pour converger. Cependant, cette accélération entraîne des oscillations plus marquées en début de progression, dues aux mises à jour plus agressives du vecteur de probabilité. Globalement, le compromis entre vitesse de convergence et stabilité est bien visible, avec une convergence plus rapide mais des variations plus prononcées dans les premières phases de l'évolution.

# 4 Sélection automatique de l'opérateur de mutation

L'algorithme génétique steady state peut être optimisé en introduisant une sélection dynamique des opérateurs de mutation. Plutôt que d'utiliser exclusivement la mutation bit-flip par défaut, qui est sous-optimale dans les phases initiales de l'évolution, cette approche propose une adaptation en temps réel du choix de l'opérateur de mutation.

Cette amélioration permet à l'algorithme de sélectionner l'opérateur de mutation le plus approprié pour chaque individu, en tenant compte de son efficacité et de l'état actuel de la population. Pour mettre en œuvre cette stratégie adaptative, deux méthodes ont été développées : la roulette adaptative et l'algorithme UCB (Upper Confidence Bound).

Ces techniques d'adaptation dynamique visent à optimiser la performance de l'algorithme en ajustant continuellement la stratégie de mutation en fonction des résultats observés, permettant ainsi une exploration plus efficace de l'espace de recherche.

## 4.1 Roulette adaptative

### 4.1.1 Définition

La méthode de la roulette adaptative implémente une stratégie d'optimisation dynamique des opérateurs de mutation. Elle utilise un ensemble $\Omega = \{o_1, \ldots, o_n\}$, où chaque $o_i$ représente un opérateur de mutation distinct. À chaque opérateur est associée une probabilité de sélection, stockée dans un vecteur $\theta$.

L'efficacité $u_i^t$ d'un opérateur $o_i$ à l'instant $t$ est évaluée selon la formule :

$$u_i^t = (1 - \alpha)u_i^{t-1} + \alpha \cdot g(o_i, s_0, \ldots, s_{t-1}) \tag{1}$$

où :

— $\alpha$ est un facteur d'apprentissage

— $u_i^0 = 0$ (valeur initiale)

— $g(o_i, s_0, \ldots, s_{t-1}) = \max(0, nouvelle fitness - ancienne fitness)$ représente le gain de l'opérateur

La probabilité de sélection $\sigma_i^{t+1}$ d'un opérateur pour l'instant $t + 1$ est calculée comme suit :

$$\sigma_i^{t+1} = p_{min} + (1 - n \cdot p_{min}) \cdot \frac{u_i^{t+1}}{\sum_{k=1}^{n} u_k^{t+1}} \tag{2}$$

où $p_{min}$ est une probabilité minimale garantie pour chaque opérateur.

### 4.1.2 pseudo code

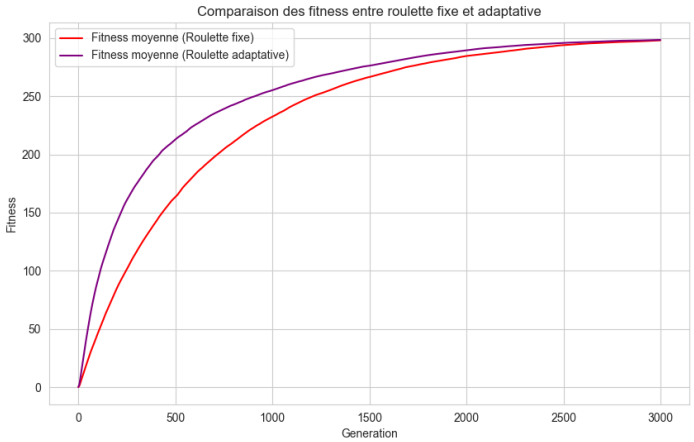
**Algorithm 1** Algorithme Génétique avec Roulette Adaptative

**Entrées:**
$n_{pop}$ : taille de la population
$n_{gen}$ : nombre de générations
$\Omega = \{\omega_1, ..., \omega_k\}$ : ensemble des opérateurs de mutation
$\alpha$ : facteur d'apprentissage
$p_{min}$ : probabilité minimale pour chaque opérateur
**Initialisation:**
$P_0 \leftarrow$ InitialiserPopulation($n_{pop}$)
$\theta \leftarrow [\frac{1}{|\Omega|}, ..., \frac{1}{|\Omega|}]$ ▷ Distribution initiale uniforme
$U \leftarrow [0, ..., 0]$ ▷ Vecteur d'utilités
$G \leftarrow [[], ..., []]$ ▷ Historique des gains
**for** $t \leftarrow 1$ to $n_{gen}$ **do**
    $f_{old} \leftarrow$ MoyenneFitness($P_{t-1}$)
    **Sélection et Mutation:**
    $parents \leftarrow$ SelectionTournoi($P_{t-1}$, 2)
    $\omega_i \leftarrow$ SelectionOperateur($\Omega$, $\theta$)
    $offspring \leftarrow$ Cloner($parents$)
    **for** $ind \in offspring$ **do**
        $ind \leftarrow \omega_i(ind)$ ▷ Application de l'opérateur
    **end for**
    **Évaluation et Insertion:**
    EvaluerFitness($offspring$)
    $P_t \leftarrow$ InsertionMeilleurs($P_{t-1}$, $offspring$)
    $f_{new} \leftarrow$ MoyenneFitness($P_t$)
    **Mise à jour Roulette:**
    $gain \leftarrow \max(0, f_{new} - f_{old})$
    **for** $j \leftarrow 1$ to $|\Omega|$ **do**
        **if** $j = i$ **then**
            $G_j \leftarrow G_j \cup \{gain\}$
        **else**
            $G_j \leftarrow G_j \cup \{0\}$
        **end if**
        $U_j \leftarrow (1 - \alpha)U_j + \alpha \cdot G_j[-1]$ ▷ Mise à jour utilité
    **end for**
    **Mise à jour Probabilités:**
    **for** $j \leftarrow 1$ to $|\Omega|$ **do**
        $\theta_j \leftarrow p_{min} + (1 - |\Omega| p_{min}) \frac{U_j}{\sum_{k=1}^{|\Omega|} U_k}$
    **end for**
**end for**
**Retourner:** Meilleur individu de $P_{n_{gen}}$

FIGURE 2 – roulette Adaptative - pseudo algo

## 4.2 Analyse expérimentale

### 4.2.1 Comparaison de la roulette adaptative et la roulette fixe




La comparaison entre la roulette adaptative (en violet) et la roulette fixe utilisant uniquement le bit-flip (en rouge) montre des performances intéressantes sur le problème OneMax avec des chaînes de 300 bits. La roulette adaptative démontre une convergence plus rapide dans les premières générations (0-1000), obtenant une meilleure fitness moyenne grace a son utilisation du 3-flip ou 5-flip au départ . Cependant, les deux approches finissent par atteindre des performances similaires vers la fin de l'évolution (autour de la génération 3000), avec une fitness moyenne proche de 300 (l'optimum pour ce problème). Cela suggère que bien que le bit-flip soit l'opérateur le plus efficace, l'utilisation adaptative de multiples opérateurs offre un avantage en termes de vitesse de convergence initiale.

### 4.2.2 Opérateurs de mutations

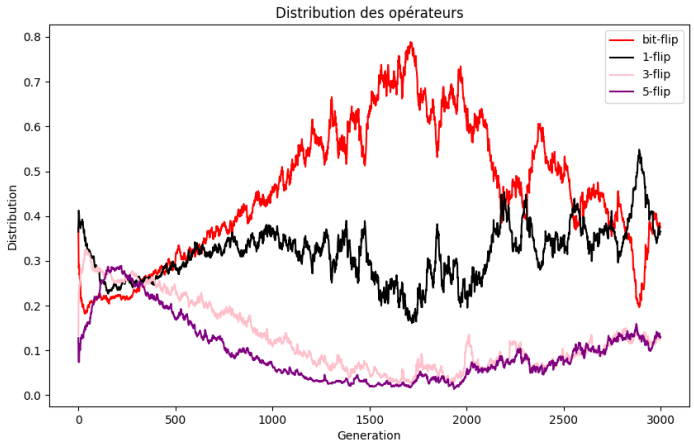



FIGURE 3 – alpha = 0.01, $p_{min} = 0.05$

On observe une convergence intéressante vers la fin où le 1-flip (ligne noire) devient plus compétitif avec le bit-flip, indiquant potentiellement une phase d'exploitation plus fine de l'espace de solutions. Les opérateurs 3-flip et 5-flip (lignes rose et violette) montrent une utilisation plus faible et décroissante au fil des générations, suggérant qu'ils sont moins adaptés pour ce problème spécifique.

# 5 UCB

L'algorithme Upper Confidence Bound (UCB) est une stratégie de sélection qui équilibre l'exploitation des options connues pour être bonnes et l'exploration d'options moins testées. Pour un opérateur $i$ au temps $t$, le score UCB est calculé comme suit :

$$UCB_i(t) = \bar{X}_i(t) + C\sqrt{\frac{\ln(t)}{n_i(t)}}$$

où :
- $\bar{X}_i(t)$ est la récompense moyenne de l'opérateur $i$ jusqu'au temps $t$
- $C$ est une constante positive qui contrôle le niveau d'exploration
- $t$ est le nombre total de sélections effectuées
- $n_i(t)$ est le nombre de fois que l'opérateur $i$ a été sélectionné, qui ne sont pas utilisés le même nombre de fois

## 5.1 Composants de la formule

- Terme d'exploitation : $\bar{X}_i(t)$
  - Représente la performance historique de l'opérateur
  - Favorise les opérateurs ayant donné de bons résultats
- Terme d'exploration : $C\sqrt{\frac{\ln(t)}{n_i(t)}}$
  - Augmente pour les opérateurs peu utilisés
  - Diminue avec le nombre d'utilisations de l'opérateur
  - Le paramètre $C$ ajuste l'importance de l'exploration

## 5.2 Pseudo code

**Algorithm 1** UCB pour la sélection d'opérateurs de mutation

**Require:** $C = 0.01$ (paramètre d'exploration)
**Require:** $operators = [bit\_flip, 1\_flip, 3\_flips, 5\_flips]$
**Require:** $window\_size = 10$ (taille de la fenêtre glissante pour les gains)
$prob\_dist \leftarrow [0.25, 0.25, 0.25, 0.25]$ {Distribution initiale équiprobable}
$gain \leftarrow [[0.25] * window\_size$ pour chaque opérateur] {Historique des gains}
$count\_use\_op \leftarrow$ [liste vide pour chaque opérateur] {Compteur d'utilisation}
$generation \leftarrow 0$
**while** $generation < MAX\_GENERATIONS$ **do**
  $old\_fitness \leftarrow$ fitness moyenne actuelle
  $operator \leftarrow$ randomChoise($operators$, weight = $prob\_dist$)
  $offspring \leftarrow$ selection(population)
  $operator(offspring)$
  $new\_fitness \leftarrow$ evalute()
  $current\_gain \leftarrow \max(0, new\_fitness - old\_fitness)$
  {Mise à jour de l'historique des gains}
  Retirer le plus ancien gain de $gain[operator]$
  Ajouter $current\_gain$ à $gain[operator]$
  {Mise à jour des compteurs d'utilisation}
  **for** chaque op dans operators **do**
    **if** $op = operator$ **then**
      $count\_use\_op[op]$ ++
    **end if**
  **end for**
  {Normalisation des gains}
  $gain\_norm \leftarrow$ normalise($gain$)
  {Calcul des moyennes}
  **for** chaque opérateur $i$ **do**
    $mean\_gains[i] \leftarrow \sum(gain\_norm[i])/window\_size$
  **end for**
  $mean\_gain\_all\_op \leftarrow \sum(mean\_gains)$
  {Mise à jour des probabilités selon UCB}
  **for** chaque opérateur $i$ **do**
    $sum\_use \leftarrow \max(1, \sum(count\_use\_op[i]))$
    $prob\_dist[i] \leftarrow \frac{mean\_gains[i] + C\sqrt{\frac{\log(generation)}{sum\_use}}}{mean\_gain\_all\_op}$
  **end for**
  $generation \leftarrow generation + 1$
**end while**

FIGURE 4 – pseudo algo UCB

## 5.3 Application à la sélection d'opérateurs de mutation

Dans le contexte de la sélection d'opérateurs de mutation (bit-flip, 1-flip, 3-flips, 5-flips), l'algorithme UCB :

- Maintient les statistiques pour chaque opérateur :
  - Nombre d'utilisations ($n_i$)
  - Somme des améliorations obtenues ($sum_i$)
- Calcule le score UCB pour chaque opérateur à chaque itération
- Sélectionne l'opérateur avec le meilleur score UCB
- Met à jour les statistiques après chaque utilisation

## 5.4 Analyse expérimentale

### 5.4.1 Comparaison entre les différents opérateurs de mutation

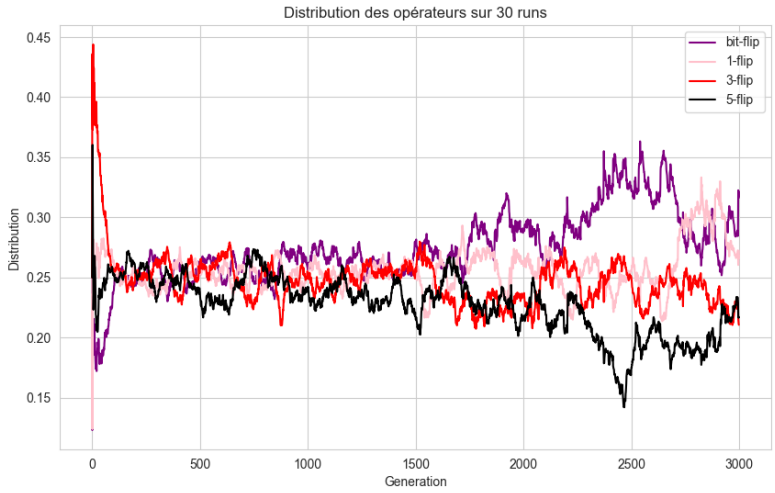



FIGURE 5 – alpha = 0.1

L'analyse du graphique généré par l'algorithme UCB ou le coefficient c est fixé à 0.01 et la fenêtre glissante est de taille 10 montre une évolution intéressante de la distribution des opérateurs de mutation sur 3000 générations et 30 runs. Contrairement à la roulette adaptative classique, UCB maintient une distribution plus équilibrée entre les opérateurs. Après une période initiale d'instabilité marquée par un pic pour le 1-flip vers la génération 0, les probabilités se stabilisent autour de 25-30% pour le bit-flip et légèrement en-dessous pour les autres opérateurs. Le bit-flip gagne progressivement en importance à partir de la génération 1500, puis il atteint des pics sans dominer aussi fortement que dans la roulette adaptative. Le 5-flip montre une légère tendance à la baisse vers la fin, tandis que le 1-flip et le 3-flip maintiennent des positions intermédiaires relativement stables. Cette distribution plus équilibrée suggère que UCB parvient à maintenir un meilleur équilibre entre exploration et exploitation, évitant la convergence forte vers un seul opérateur comme observé dans la roulette adaptative.

### 5.4.2 Opérateur inutile

L'opérateur identité est un opérateur de mutation qui ne modifie pas l'individu. Par exemple, pour un individu représenté par le tableau $[1, 0, 1, 0]$, l'application de l'opérateur identité donne le même individu :

$$I([1,0,1,0]) = [1,0,1,0]$$

Cet opérateur est essentiel pour tester l'efficacité des méthodes de sélection d'opérateurs, car il permet d'évaluer si celles-ci peuvent détecter et réduire la probabilité de sélection d'un opérateur qui n'apporte aucune amélioration.

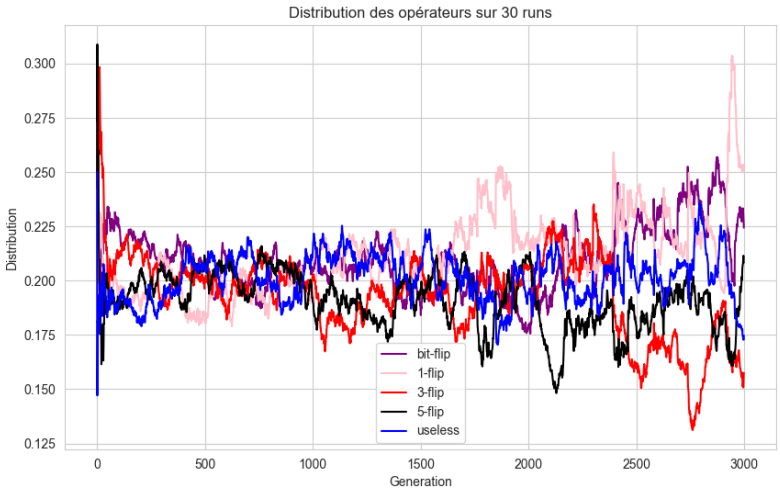



FIGURE 6 – alpha = 0.1

L'analyse de la distribution des opérateurs de mutation montre un comportement intéressant et logique de l'algorithme UCB dans les phases avancées de l'optimisation. Bien que l'opérateur useless ne contribue pas directement à l'amélioration de la solution, sa présence relativement stable dans la distribution peut s'expliquer par la nature du problème OneMax. En effet, lorsque la solution contient déjà une proportion élevée de 1's vers la fin de l'optimisation, les opérateurs plus agressifs comme 3-flip et 5-flip présentent un risque accru de dégrader la solution en transformant des 1's en 0's. Dans ce contexte, l'opérateur useless, qui préserve l'état actuel de la solution, peut être préférable à des opérateurs plus destructifs, illustrant ainsi la capacité de UCB à s'adapter à l'état de la recherche et à trouver un équilibre entre la préservation des bonnes solutions et la poursuite de l'amélioration.

# 6 Méthode des Îles Dynamiques

## 6.1 Introduction

La méthode des îles dynamiques est une approche évolutionnaire qui combine la recherche locale avec un mécanisme de migration adaptatif. Cette méthode utilise plusieurs populations isolées (îles) qui évoluent en parallèle avec différentes stratégies de mutation, permettant ainsi une exploration diversifiée de l'espace de recherche.

## 6.2 Principe de Base

L'algorithme des îles dynamiques repose sur deux concepts fondamentaux : (1) la diversification par îles spécialisées et (2) l'apprentissage des schémas de migration.

### 6.2.1 Structure Multi-îles

Chaque île maintient sa propre population d'individus et applique une stratégie de mutation spécifique :

— **One-flip**
— **Three-flips**
— **Five-flips**
— **Bit-flip**

Cette spécialisation permet d'explorer l'espace de recherche avec différentes intensités de perturbation.

## 6.3 Structure de l'Algorithme

### 6.3.1 Paramètres Clés

Les paramètres principaux qui influencent le comportement de l'algorithme sont :

— Facteur d'apprentissage
— Facteur d'exploration
— Taille de population par île
— Nombre d'îles

### 6.3.2 Mise à jour de la Matrice de Migration

La mise à jour des probabilités de migration est une composante essentielle de l'algorithme, s'appuyant sur une formule d'apprentissage par renforcement qui équilibre exploitation et exploration [1] :

$$V = (1 - \beta)(\alpha V + (1 - \alpha)D) + \beta N \tag{3}$$

Cette équation représente un processus d'apprentissage adaptatif où :

— $V$ : Matrice des probabilités de migration ($n \times n$ pour $n$ îles)

— $D$ : Matrice des récompenses calculée à partir des améliorations de fitness
— $N$ : Matrice stochastique de bruit uniformément distribuée
— $\alpha \in [0, 1]$ : Taux d'apprentissage contrôlant l'inertie du système
— $\beta \in [0, 1]$ : Taux d'exploration régulant l'influence du bruit

La formule se décompose en deux termes principaux :

1. $(1 - \beta)(\alpha V + (1 - \alpha)D)$ : Terme d'exploitation
   — $\alpha V$ maintient la mémoire des stratégies passées
   — $(1 - \alpha)D$ intègre les nouvelles informations de performance
2. $\beta N$ : Terme d'exploration pure

**Algorithm 1** Mise à jour des Probabilités de Migration

**Require:** $V$ : matrice de migration courante
**Require:** $D$ : matrice de récompenses
**Require:** $\alpha, \beta$ : paramètres d'apprentissage
**Ensure:** Nouvelle matrice de migration $V$

**Génération du bruit**
$N \leftarrow$ MatriceAléatoire$(n, n)$ ▷ Distribution uniforme
NormaliserLignes$(N)$ ▷ Somme = 1 pour chaque ligne
**Calcul du terme d'exploitation**
$E \leftarrow \alpha V + (1 - \alpha)D$
**Application de la formule complète**
$V_{temp} \leftarrow (1 - \beta)E + \beta N$
**Normalisation finale**

**for** chaque ligne $i$ de $V_{temp}$ **do**
    $V_{temp}[i] \leftarrow V_{temp}[i] / \sum V_{temp}[i]$
**end for**
$V \leftarrow V_{temp}$ **return** $V$

Cette mise à jour garantit :
— Le maintien de distributions de probabilités valides (somme = 1)
— Un équilibre dynamique entre exploitation et exploration
— Une adaptation progressive aux performances observées
— Une robustesse face aux fluctuations locales de performance

### 6.3.3 Mise à jour de la Matrice de Gain

Cette stratégie concentre les récompenses uniquement sur les meilleures migrations observées, favorisant ainsi l'exploitation des chemins de migration les plus efficaces [1].

**Algorithm 2** Calcul des Récompenses de Migration

**for** chaque île source $s$ **do**
    Calculer l'amélioration moyenne $\Delta f_{s,d}$ pour chaque destination $d$
    $meilleure\_amlioration \leftarrow \max(\Delta f_{s,*})$
    $B_s \leftarrow \{d : \Delta f_{s,d} = meilleure\_amlioration\}$
    **if** $meilleure\_amlioration > 0$ **then**
        $D[s, d] \leftarrow 1/|B_s|$ pour tout $d \in B_s$
    **end if**
**end for**

## 6.4 Analyse

### 6.4.1 La progression de la fitness

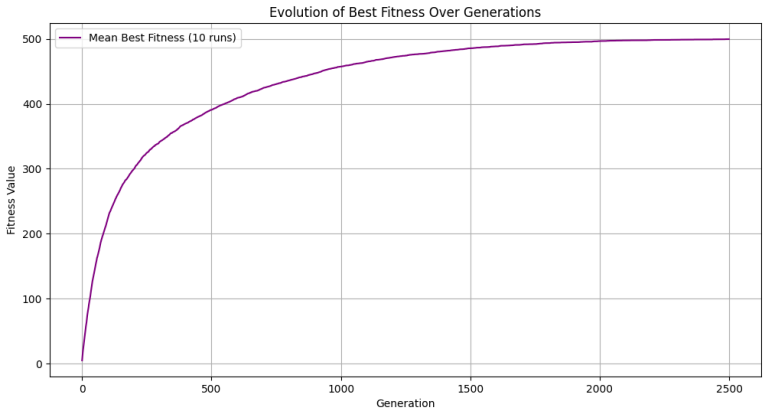



FIGURE 7 – fitness progression

Dans les 200 premières générations, on observe une convergence très rapide avec une augmentation exponentielle de la fitness, montrant l'efficacité initiale des mutations. De 200 à 1000 générations, la progression ralentit mais reste constante, suggérant une phase d'exploration plus fine de l'espace de recherche. De 1000 à 2000 générations, la courbe s'aplatit progressivement, indiquant que l'algorithme approche d'un optimum local ou global. À partir de 2000 générations, la stagnation de la courbe autour d'une fitness de 500 suggère que l'algorithme a atteint sa limite de convergence avec les paramètres actuels.

### 6.4.2 Dynamique des Populations

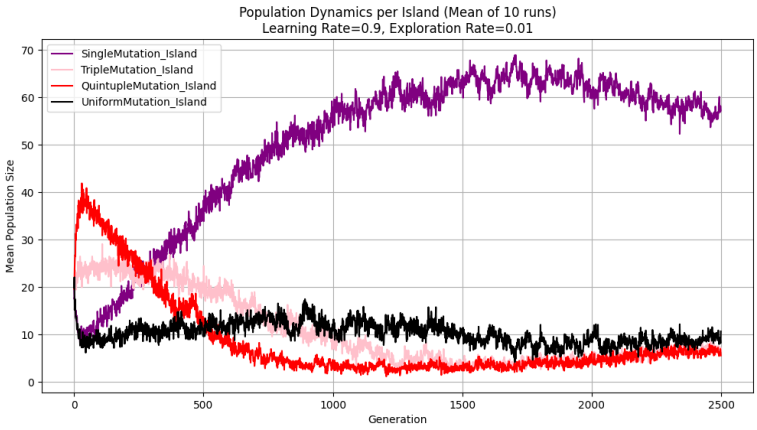



FIGURE 8 – Dynamique des Populations

L'île SingleMutation montre une croissance continue jusqu'à dominer la population totale (environ 60-70 individus), suggérant qu'elle trouve les meilleures solutions locales. QuintupleMutation et TripleMutation démarrent fort mais leur population décline rapidement, indiquant que leurs stratégies de mutation plus agressives sont moins efficaces sur le long terme. L'île UniformMutation maintient une population stable mais faible (environ 10 individus) tout au long de l'évolution, suggérant qu'elle joue un rôle de diversification plutôt que d'optimisation.

Ce graphique montre une forte similitude dans leurs résultats avec le graphique de [1] (figure 5), on a une domination claire du 1-flip/SingleMutation qui devient progressivement l'opérateur le plus performant au fil des générations. Les autres opérateurs suivent également les mêmes tendances dans les deux cas, notamment avec le 5-flip/QuintupleMutation qui est efficace au début mais décline rapidement, tandis que le 3-flip/TripleMutation et bit-flip/UniformMutation maintiennent des niveaux intermédiaires plus stables.

### 6.4.3 Analyse sur le facteur d'exploration

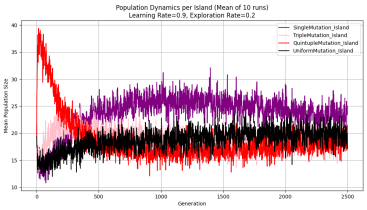



FIGURE 9 – Beta = 0.2

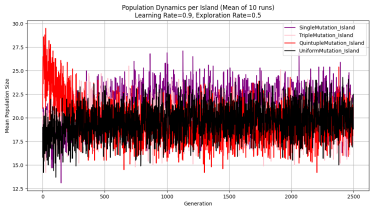



FIGURE 10 – Beta = 0.5

# 7 Autres problèmes binaires

## 7.1 Leading ones

Le problème Leading Ones est un problème d'optimisation binaire où l'objectif est de maximiser le nombre de 1's consécutifs depuis le début de la chaîne.

Autrement dit, pour une position $i$, la contribution à la fitness est 1 si et seulement si tous les bits de la position 1 jusqu'à la position $i$ sont égaux à 1. La fitness s'arrête de compter au premier 0 rencontré.

Par exemple, pour une chaîne $x$ de longueur $n = 8$ :

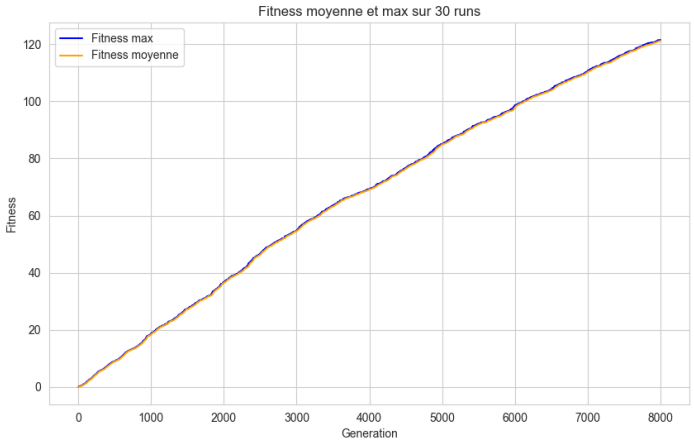
— $x = (1, 1, 1, 0, 1, 1, 1, 1) \rightarrow f_{LO}(x) = 3$

— $x = (1, 1, 1, 1, 1, 0, 0, 0) \rightarrow f_{LO}(x) = 5$

— $x = (0, 1, 1, 1, 1, 1, 1, 1) \rightarrow f_{LO}(x) = 0$

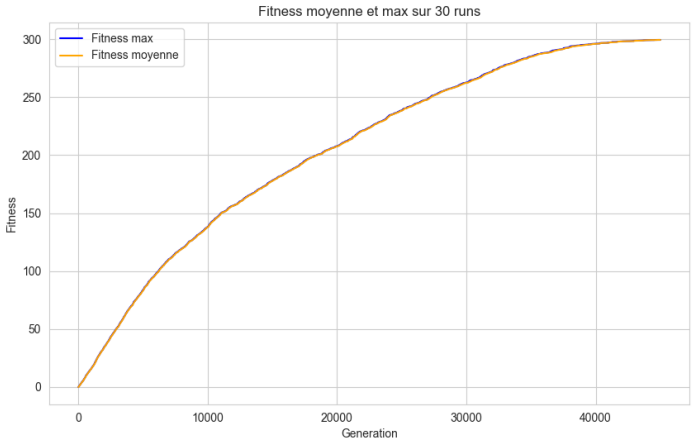
La valeur maximal est $n$, obtenue uniquement lorsque tous les bits sont à 1. Ce problème est considéré comme plus difficile que OneMax car les bits sont dépendants : la contribution d'un bit à la fitness dépend des valeurs de tous les bits qui le précèdent.

### 7.1.1 UCB

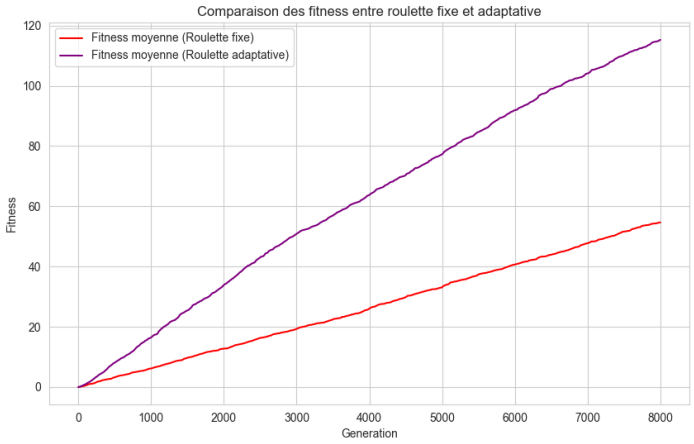



UCB montre une progression remarquablement linéaire pour le problème LeadingOnes, avec la fitness maximale et moyenne presque confondues, atteignant environ 120 1's consécutifs après 8000 générations. En comparaison avec la roulette adaptative qui atteignait une performance similaire, UCB semble maintenir une stabilité plus forte entre la fitness maximale et moyenne, suggérant une meilleure robustesse de l'approche. Cette similitude de performance entre UCB et la roulette adaptative, tous deux nettement supérieurs à la roulette fixe (qui n'atteignait que 55), confirme l'importance d'une sélection adaptative des opérateurs pour le problème LeadingOnes, bien que UCB semble offrir une progression plus stable.

Les résultats obtenus sur le problème LeadingOnes ne sont pas satisfaisants car atteindre seulement 120 1's consécutifs sur 300 possibles après 8000 générations représente une performance limitée (40% de l'optimum), avec une progression trop lente. Cette performance pourrait être améliorée en ajustant plusieurs paramètres comme la taille de la population, les paramètres de mutation, l'ajout d'opérateurs spécifiques à LeadingOnes ou l'incorporation d'un mécanisme de croisement, pour atteindre la solution optimal il faut encore augmenter le nombre de générations et donc avoir un temps d'éxécution plus important :




### 7.1.2 Roulette adaptative




L'analyse du graphique comparant les performances des roulettes fixe et adaptative sur le problème LeadingOnes montre une différence notable entre les deux approches. La roulette adaptative surpasse significativement la roulette fixe, atteignant une fitness d'environ 115 contre 55 pour la roulette fixe, sur une chaîne de longueur 300 et 8000 générations alors que pour le probléme oneMax on atteint la solution optimal (300) avec 3000 générations.

Cette supériorité de la roulette adaptative s'explique par sa capacité à mieux ajuster ses opérateurs de mutation face à la complexité du problème LeadingOnes, permettant une progression plus constante dans la construction et le maintien de longues séquences de 1's consécutifs, contrairement à la roulette fixe qui éprouve plus de difficultés avec son unique opérateur bit-flip, il faut

noter aussi que le temps de résolution est beaucoup plus lent que le problème One-Max classique.

## 7.2 One-max avec masque

Dans le cas du problème OneMax standard, l'objectif est de maximiser le nombre de 1's dans une chaîne binaire. L'introduction d'un masque permet de modifier cette fonction objectif en pondérant l'importance de chaque position. Formellement, si $x = (x_1, \ldots, x_n)$ est une solution et $m = (m_1, \ldots, m_n)$ est un masque binaire, alors la fonction objectif devient :

$$f(x) = \sum_{i=1}^{n} x_i \cdot m_i$$

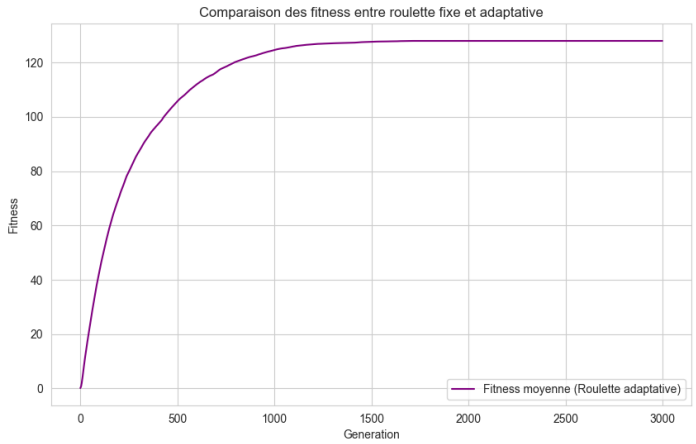
où $x_i, m_i \in \{0, 1\}$. Le masque permet ainsi de définir des "positions cibles" où la présence d'un 1 est souhaitée ($m_i = 1$) ou ignorée ($m_i = 0$). Cette modification transforme le problème OneMax classique en un problème plus général où certaines positions peuvent être plus importantes que d'autres dans l'évaluation de la qualité d'une solution.

Le masque $m$ peut être généré de plusieurs manières selon les objectifs d'analyse :

- **Génération aléatoire :** $m_i \sim Bernoulli(p)$, où $p$ est la probabilité d'avoir un 1
- **Motif régulier :** Par exemple, $m_i = i \bmod 2$ pour créer une alternance de 0 et 1
- **Blocs structurés :** Division du masque en sections avec différentes densités de 1's
- **Distribution biaisée :** Favoriser la présence de 1's ou de 0's selon une distribution de probabilité spécifique

Le choix de la méthode de génération du masque permet de créer différentes variantes du problème OneMax pour tester la robustesse et l'adaptabilité des algorithmes d'optimisation, ici on choisi la méthode aléatoire avec une application.

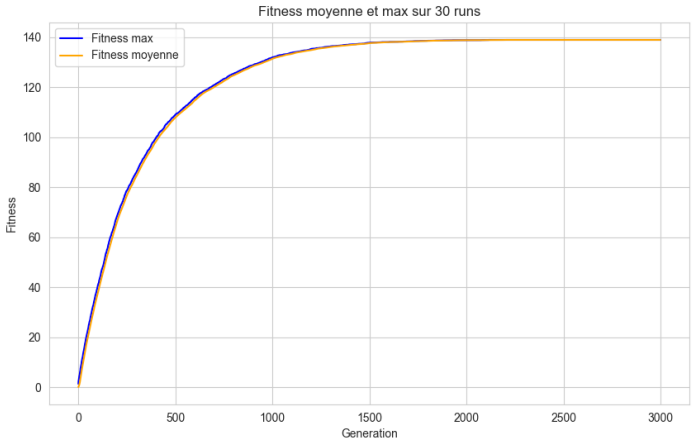
### 7.2.1 Roulette adaptative




On constate que l'algorithme utilisant la roulette adaptative ne réussit pas à atteindre l'objectif souhaité. Sa performance plafonne à une valeur de fitness de 120, ce qui équivaut approximativement à avoir un tiers des bits activés. Cette stagnation indique que l'algorithme n'arrive pas à améliorer significativement les solutions au fil des générations.

Par ailleurs, l'utilisation des différents opérateurs génétiques semble être désordonnée et inefficace. L'algorithme ne parvient pas à déterminer de manière cohérente quels opérateurs sont les plus performants pour résoudre le problème donné. Cette incapacité à identifier et à privilégier les opérateurs les plus efficaces contribue probablement à l'échec de l'algorithme à converger vers une solution optimale.

### 7.2.2 UCB




On remarque ici que le masque a eu le même effet sur l'UCB, dont la performance plafonne cette fois à une valeur de fitness de 140.

### 7.2.3 Conclusion

Le problème OneMax avec masque représente une variante intéressante qui permet de tester la robustesse des algorithmes,les résultats montrent que tant la roulette adaptative que l'UCB peinent à atteindre des performances optimales, plafonnant respectivement à des fitness de 120 et 140. on déduit que l'introduction du masque rend le problème significativement plus complexe que le OneMax classique.

# 8 Conclusion

Notre étude comparative des différentes stratégies adaptatives d'optimisation combinatoire, appliquées principalement au problème OneMax, révèle un panorama nuancé des performances. L'analyse démontre que l'efficacité de chaque méthode est intimement liée à sa configuration et au contexte d'application, soulignant l'importance d'une sélection réfléchie de l'approche en fonction des spécificités du problème à résoudre.

# 9 Références

[1] Caner Candan, Adrien Goeffon, Frédéric Lardeux, and Frédéric Saubion. A dynamic island model for adaptive operator selection.

[2] Caner Candan, Adrien Goeffon, Frédéric Lardeux, and Frédéric Saubion. Non stationary operator selection with island models.

[3] Adrien Goeffon, Frédéric Lardeux. Toward Autonomous Search with Island Models