Les méthodes **"Élite"**, **"Roulette"**, et **"Tournoi"** sont des stratégies de **sélection** utilisées dans les **algorithmes génétiques**. Elles déterminent **quels individus** seront choisis pour se reproduire et transmettre leurs gènes à la génération suivante.

**1. Sélection par Élite (Elitism) :** On sélectionne directement **les meilleurs individus** et on les **copie** dans la nouvelle génération **sans modification**.Cela garantit que les meilleures solutions **ne sont pas perdues**.

**✅ Avantages :** Conserve les meilleures solutions à chaque génération. Accélère la convergence vers une bonne solution.

**❌ Inconvénients :**  Peut **réduire la diversité** et conduire à un minimum local.( Un **minimum local** est une solution d’un problème d’optimisation où la **valeur de la fonction objectif** (fitness) est **plus basse** que celle de ses voisins, mais **pas la meilleure possible** sur l’ensemble de l’espace de recherche. Dans un **problème de minimisation**, un minimum local est une solution qui est **meilleure que toutes les solutions proches**, mais **pas forcément la meilleure solution globale** (**minimum global**).

**Exemple**

Si on garde **les 2 meilleurs individus** :

* Génération actuelle :

scss

CopierModifier

Ind 1 (fitness 90)

Ind 2 (fitness 85)

Ind 3 (fitness 70)

Ind 4 (fitness 60)

* **Nouvelle génération** (avant croisement et mutation) :

scss

CopierModifier

Ind 1 (fitness 90) ✅ Conservé

Ind 2 (fitness 85) ✅ Conservé

**2. Sélection par Roulette (Roulette Wheel Selection)**

**🔹 Principe :**Chaque individu est sélectionné **proportionnellement à sa fitness**.Un individu avec une **grande fitness** a plus de chances d’être choisi.

**✅ Avantages :** Favorise les meilleures solutions, mais laisse une chance aux autres. Maintient une diversité génétique.

**❌ Inconvénients**

✖ Peut être lent à converger si les différences de fitness sont faibles.  
✖ Peut être biaisé si une solution est trop dominante.

**Exemple**

Si on a les individus suivants avec leurs fitness :

yaml

CopierModifier

Ind 1 : fitness = 10

Ind 2 : fitness = 30

Ind 3 : fitness = 60

Leur probabilité d’être sélectionné est :

matlab

CopierModifier

Ind 1 → 10%

Ind 2 → 30%

Ind 3 → 60%

On simule une **roue de la fortune**, et un individu est choisi aléatoirement en fonction de ces probabilités.

**3. Sélection par Tournoi (Tournament Selection)**

**🔹 Principe** On sélectionne **aléatoirement** un petit groupe d’individus (ex: 3). **Le meilleur** du groupe est choisi pour la reproduction.

**✅ Avantages**

✔ Simple et efficace.  
✔ Bon équilibre entre **exploitation (garder les meilleurs)** et **exploration (diversité)**.

**❌ Inconvénients**

✖ Le paramètre du **nombre d’individus dans le tournoi** doit être bien réglé.  
✖ Trop grand = moins de diversité, trop petit = sélection trop aléatoire.

**Exemple**

Si on sélectionne **3 individus au hasard** :

yaml

CopierModifier

Ind 1 : fitness 40

Ind 2 : fitness 80 ✅ (Gagnant)

Ind 3 : fitness 50

L’individu **2** est sélectionné car il a la meilleure fitness.

**4. Comparaison des méthodes**

| **Méthode** | **Rapidité** | **Diversité Génétique** | **Risque de Minimum Local** | **Robustesse** |
| --- | --- | --- | --- | --- |
| **Élite** | ⭐⭐⭐ | ❌ Faible | ⚠️ Risque élevé | ⭐⭐⭐ |
| **Roulette** | ⭐⭐ | ✅ Bonne | ⚠️ Peut être biaisé | ⭐⭐ |
| **Tournoi** | ⭐⭐⭐ | ✅ Bonne | ⚠️ Risque moyen | ⭐⭐⭐ |

**5. Quelle méthode choisir ?**

✔ **Sélection Élite** → si on veut **conserver les meilleures solutions** à chaque génération.  
✔ **Roulette** → si on veut un **équilibre entre hasard et performance**.  
✔ **Tournoi** → si on veut un **contrôle sur la pression sélective**.

👉 **En pratique**, on combine souvent **Élite + Tournoi ou Élite + Roulette** pour un bon équilibre !

 **Élite** : Sélection **directe** des meilleurs.

 **Roulette** : Sélection **aléatoire pondérée** par la fitness.

 **Tournoi** : Sélection par **compétition locale** entre petits groupes.

Ce code implémente un **algorithme génétique** pour résoudre le **problème du sac à dos** (Knapsack Problem), un problème d'optimisation combinatoire où on doit **maximiser la valeur des objets choisis** sans dépasser une **capacité de poids maximale**.

## ****🔍 1. Explication du code****

Le programme suit les étapes classiques d’un algorithme génétique :

1. **Initialisation** de la population
2. **Évaluation de la fitness** de chaque individu
3. **Sélection des meilleurs individus**
4. **Croisement** pour générer de nouveaux individus
5. **Mutation** pour maintenir la diversité
6. **Répétition du processus sur plusieurs générations**
7. **Affichage de la meilleure solution** trouvée

## ****📌 2. Détaillons chaque partie du code****

### ****🔹 Paramètres du problème****

python

CopierModifier

values = [60, 100, 120, 30, 50, 70, 90, 40, 80, 110, 55, 95, 85, 45, 65] # Valeurs des objets

weights = [10, 20, 30, 5, 10, 15, 25, 8, 18, 28, 12, 22, 17, 9, 13] # Poids des objets

capacity = 100 # Poids maximal du sac

num\_items = len(values) # Nombre total d'objets

* **Chaque objet a une valeur et un poids.**
* **Le sac ne peut pas dépasser un poids total de 100.**

### ****🔹 Paramètres de l'algorithme génétique****

python

CopierModifier

POP\_SIZE = 100 # Taille de la population

GENERATIONS = 10 # Nombre de générations

MUTATION\_RATE = 0.05 # Probabilité de mutation

SELECTION\_METHOD = "tournament" # Méthode de sélection (modifiable)

* **100 individus** sont générés au départ.
* L’évolution dure **10 générations**.
* **5% de mutation** pour éviter la stagnation.
* **Méthode de sélection** modifiable par l’utilisateur.

### ****🔹 Génération d’un individu aléatoire****

python

CopierModifier

def create\_individual():

return np.random.randint(0, 2, size=num\_items)

* Chaque individu est représenté par **un tableau de 0 et 1**.
  + 1 → L'objet est sélectionné.
  + 0 → L'objet n'est pas sélectionné.
* Ex: [1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1]

### ****🔹 Fonction de Fitness****

python

CopierModifier

def fitness(individual):

total\_weight = sum(w for i, w in enumerate(weights) if individual[i])

total\_value = sum(v for i, v in enumerate(values) if individual[i])

return total\_value if total\_weight <= capacity else 0

* **On calcule le poids et la valeur totale** des objets sélectionnés.
* Si le poids dépasse capacity, la **fitness devient 0** (solution invalide).

**🔹 Sélection des parents** La **sélection des parents** est une étape clé d’un **algorithme génétique** où on **choisit les individus** qui vont **se reproduire** pour créer la **prochaine génération**.

💡 **Objectif** :

* Favoriser les **meilleures solutions** (fitness élevée)
* Maintenir une **diversité génétique** pour éviter la stagnation
* Simuler l'**évolution naturelle**

python

CopierModifier

def selection(population):

if SELECTION\_METHOD == "elite":

return sorted(population, key=fitness, reverse=True) # Garde les meilleurs

#### **1. Sélection par ÉLITE**

* Trie la population par **fitness décroissante** et prend les **meilleurs individus**.

python

CopierModifier

elif SELECTION\_METHOD == "roulette":

total\_fit = sum(fitness(ind) for ind in population)

if total\_fit == 0:

return random.choices(population, k=POP\_SIZE)

probs = [fitness(ind) / total\_fit for ind in population]

return random.choices(population, weights=probs, k=POP\_SIZE)

#### **2. Sélection par ROULETTE**

* Un individu avec une **fitness élevée** a **plus de chances d’être choisi**.
* Simule une **roue de la fortune** où la taille de la section correspond à la fitness.

python

CopierModifier

elif SELECTION\_METHOD == "tournament":

selected = []

for \_ in range(POP\_SIZE):

competitors = random.sample(population, k=5)

winner = max(competitors, key=fitness)

selected.append(winner)

return selected

#### **3. Sélection par TOURNOI**

* Sélectionne **5 individus au hasard**, et **prend le meilleur** comme parent.
* Assure une bonne diversité génétique tout en favorisant les meilleurs.

### ****🔹 Croisement (Crossover)****

python

CopierModifier

def crossover(p1, p2):

point = random.randint(1, num\_items - 1)

return np.concatenate((p1[:point], p2[point:]))

* **Sélectionne un point de coupure** aléatoire.
* **Combine les gènes** des deux parents pour former un enfant.
* Exemple :
  + P1 = [1, 1, 0, | 0, 1, 0]
  + P2 = [0, 0, 1, | 1, 0, 1]
  + Child = [1, 1, 0, | 1, 0, 1]

### ****🔹 Mutation****

python

CopierModifier

def mutate(individual):

for i in range(num\_items):

if random.random() < MUTATION\_RATE:

individual[i] = 1 - individual[i]

return individual

* Chaque gène a **5% de chance d’être inversé** (0 ↔ 1).
* Permet d’éviter **la stagnation** dans un minimum local.

### ****🔹 Boucle principale****

python

CopierModifier

def run():

population = [create\_individual() for \_ in range(POP\_SIZE)]

best\_scores = []

* Génère **la population initiale**.

python

CopierModifier

for generation in range(GENERATIONS):

population = selection(population)

next\_gen = population[:5] # Conservation des meilleurs (ÉLITE)

* **Sélectionne** les parents et **garde les 5 meilleurs**.

python

CopierModifier

while len(next\_gen) < POP\_SIZE:

parents = random.choices(population, k=2)

child = crossover(parents[0], parents[1])

child = mutate(child)

next\_gen.append(child)

* **Remplit la nouvelle génération** par croisement et mutation.

python

CopierModifier

best = max(population, key=fitness)

best\_scores.append(fitness(best))

print(f"Génération {generation+1} - Score: {fitness(best)} - Sélection: {best}")

* Affiche **le meilleur score** de chaque génération.

### ****🔹 Affichage des résultats****

python

CopierModifier

plt.plot(range(1, GENERATIONS + 1), best\_scores, marker='o')

plt.title("Évolution du score maximum par génération")

plt.xlabel("Génération")

plt.ylabel("Score maximum")

plt.grid(True)

plt.show()

* **Trace un graphique** de l’évolution du score au fil des générations.

## ****🎯 3. Conclusion****

✅ Ce programme montre comment un **algorithme génétique** peut résoudre un **problème d'optimisation combinatoire**.  
✅ Il propose **trois méthodes de sélection** pour comparer leur efficacité.  
✅ Il équilibre **exploitation (garder les meilleurs)** et **exploration (mutation, diversité)**.

**1. Applications dans l'industrie**

**🔧 a. Optimisation de la production**

Problèmes résolus : Planification des tâches sur les machines Réduction des coûts et du temps de fabrication Réduction des déchets ou des pertes de matières premières

Exemples : Ordonnancement dans une usine (Job Shop Scheduling) Optimisation de la chaîne de production (ex : production d’ammoniac)

**🚛 b. Logistique et transport**

Problèmes résolus : Problème du voyageur de commerce (TSP) Optimisation de tournées de livraison  Répartition optimale de camions ou de conteneurs

Exemples : Amazon, DHL : optimiser les itinéraires de livraison Entreprises de fret : chargement de conteneurs (bin packing problem)

**🔌 c. Énergie et automatisation** Problèmes résolus :Réglage des paramètres d’un système automatiséOptimisation des réseaux électriques

Exemples :Réduction de la consommation d’énergie d’un four industrielRéglage automatique d’un thermostat ou d’un robot

**2. Applications dans la vie quotidienne et les sciences**

**🧠 a. Intelligence artificielle**

Évolution d’agents intelligents : Apprentissage automatique sans rétropropagation

Exemples : Agents de jeux (Snake, Flappy Bird, voitures autonomes…) Réseaux de neurones optimisés par AG

**📈 b. Finance**

Problèmes résolus : Optimisation de portefeuille Sélection d’actifs

Exemples : Trouver la meilleure répartition d’un investissement entre plusieurs actions pour maximiser le rendement et réduire le risque

**c. Bio-informatique et médecine**

Alignement de séquences ADN Optimisation de traitements (dosage, combinaison de médicaments) Planification de radiothérapie

**🎓 d. Éducation et emploi du temps**

Génération automatique d’emplois du temps optimisés Répartition équitable des ressources et des enseignants