TD3 : recuit simulé et algorithme génétique

Le but de ce TP est de résoudre le problème des N-Reines à l'aide du recuit simulé et de l'algorithme génétique. Le problème des N-Reines consiste à placer n reines sur un échiquier de taille (N x N) sans qu'elles s'attaquent : deux reines ne doivent pas se trouver sur la même ligne, sur la même colonne ou sur la même diagonale.

La figure 1 illustre une solution du problème avec 8 reines.

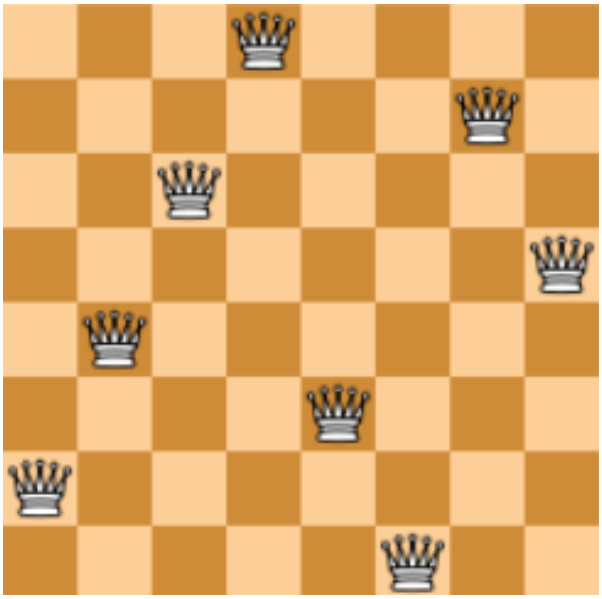

FIGURE 1 – solution du problème des 8-reines.

Résoudre le Problème des N Reines avec un Recuit Simulé

Un échiquier peut être représenté par un tableau à une dimension où chaque case contient l'indice de la ligne où se trouve la reine.

Donner la représentation sous forme de tableau de l'échiquier de la Figure 1.

La fonction suivante permet de générer un état d'échiquier avec des reines positionnées aléatoirement

```
import random
import math

def etatInitial(N):
        etat =[]
        for i in range(N) :
                etat.append(random.randint(0, N - 1))
        return etat
```

La fonction afficher permet d'afficher un échiquier à partir de sa représentation sous forme d'un tableau

```
def afficher(etat):
        N = len(etat)
        for i in range(N):
                for j in range(N):
                        if(etat[j]==i):
                                print ("|Q", end="")
                        else:
                                print("| ", end="")
                print("|")
```

La fonction évaluer permet de calculer le nombre de paires de reines qui s'attaquent d'un état donné (paires de reines sur la même ligne / colonne / diagonale).

```
def evaluer(etat):
        eval = 0
        N = len(etat)
        for i in range(N):
                for j in range(i+1, N):
                        if etat[i] == etat[j] or
                           abs(i - j) == abs(etat[i] - etat[j]):
                           eval += 1
        return eval
```

Générer un état aléatoire et afficher son évaluation.

Implémenter l'algorithme du recuit simulé pour résoudre le problème de N reines.

Exécutez votre algorithme du recuit simulé avec différentes valeurs de paramètres tels que la température initiale, le taux de refroidissement, la taille de l'échiquier. Analysez

comment les résultats varient en fonction des paramètres choisis.

Résoudre le Problème des N Reines avec un Algorithme Génétique

La fonction fitness(solution) permet d'évaluer la qualité d'un état/individu. Quel est le fitness d'une solution d'un problème N reines.

```
def fitness(etat):
        return 1/(1+ evaluer(etat))
```

La fonction `population_initiale` permet de générer une population d'individus. Comment sont générés ces individus ?

```
def population_initiale(taille_population, N):
        population = []
        for i in range(taille_population):
                population.append(etatInitial(N))
        return population
```

La fonction `selection(population)` permet de sélectionner un parent à partir d'une population d'individus. Générer une population initiale de 4 individus pour le problème de 4 reines. Sélectionner un individu avec la fonction `selection`. De quelle méthode de `selection` s'agit il ?

```
def selection(population):
        total =0
        for solution in population :
                total +=fitness(solution)
        r = random.uniform(0, total)
        cumulative_fitness = 0
        for solution in population:
                cumulative_fitness += fitness(solution)
                        if cumulative_fitness >= r:
                                return solution
```

Voici le code d'un algorithme génétique. Implémentez les fonction **croisement** et **mutation**.

Testez l'algorithme génétique avec différents paramètres :

- Comment les performances de l'algorithme varient-elles en fonction de la taille de l'échiquier (N) ?
- Comment les performances changent-elles avec différents paramètres de l'algorithme (taille de la population, taux de mutation, etc.) ?

```
def algorithme_génétique(N, taille_population, nombre_générations, taux_mutation):
        population = population_initiale(taille_population, N)

        for generation in range(nombre_générations):
                nouvelle_population = []

        for i in range(taille_population // 2):
                parent1 = selection(population)
                parent2 = selection(population)
                enfant = croisement(parent1, parent2)

        if random.random()<taux_mutation :
                enfant = mutation(enfant)

        nouvelle_population.extend([parent1, parent2, enfant])
                population = nouvelle_population

        solution = max(population, key=fitness)
        return solution
```