

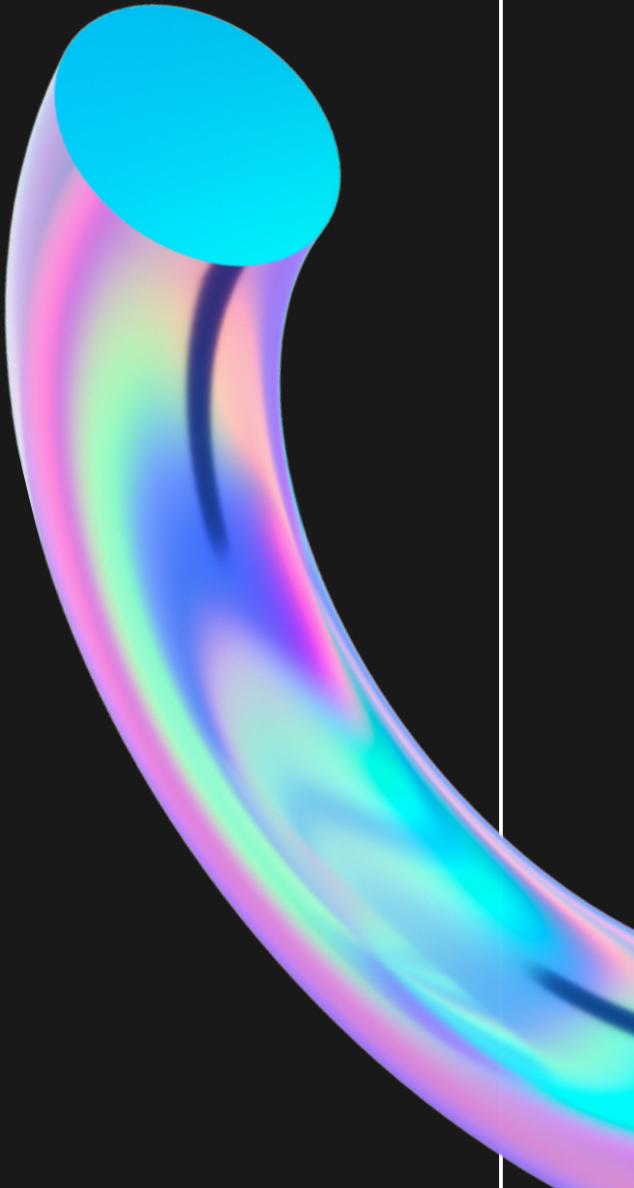
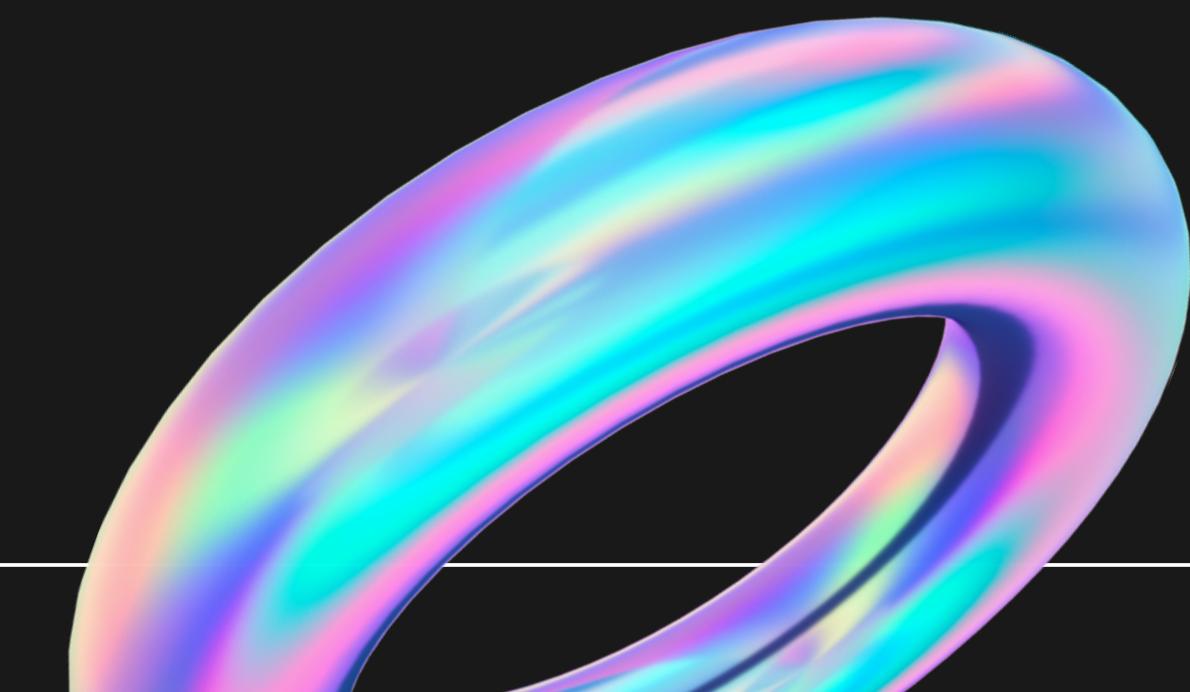
KECERDASAN BUATAN

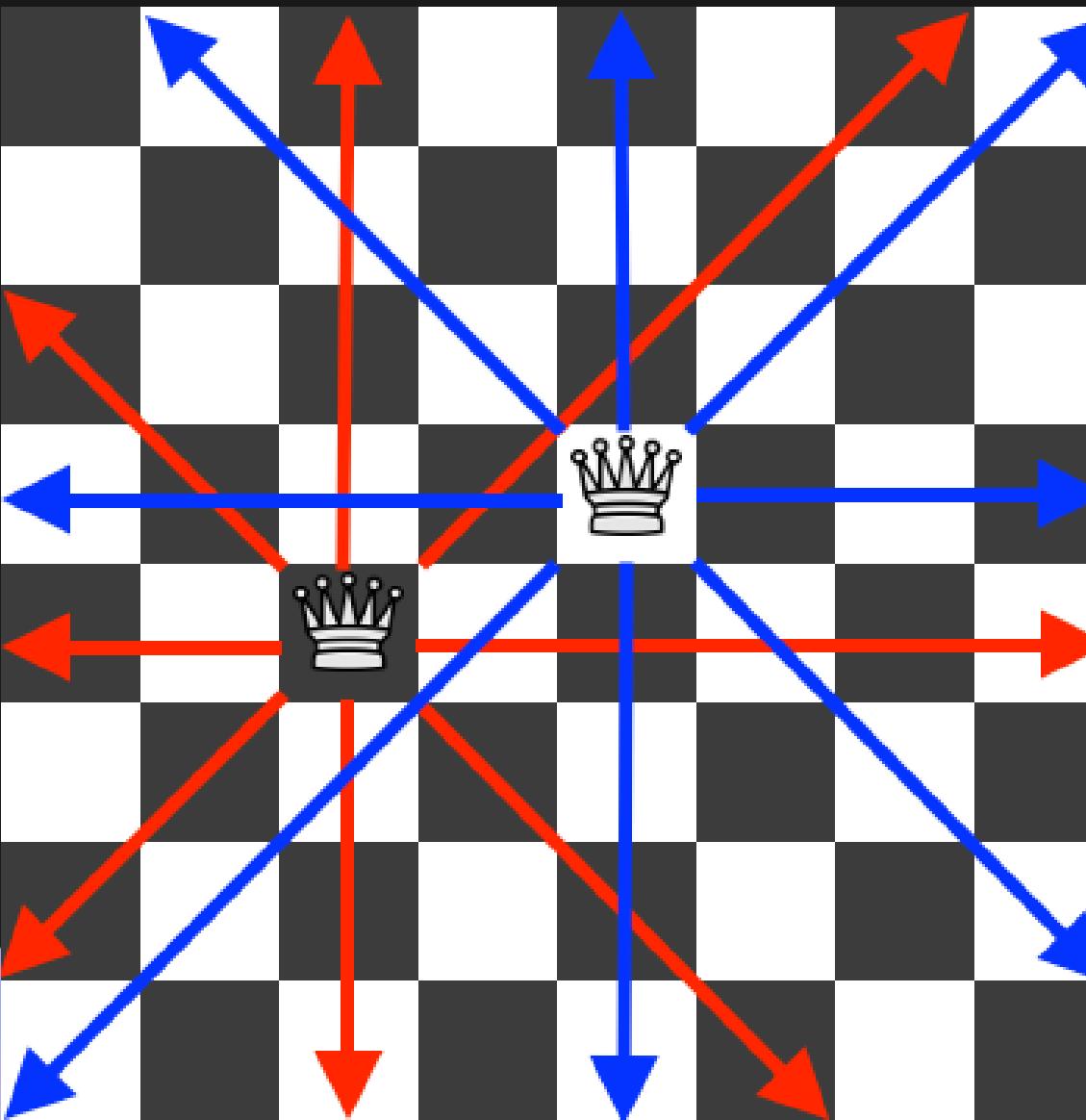
8-QUEENS USING GENETIC ALGORITHM

Timothy Hosia (5025211098)

Yohanes Teguh (5025211179)

Shazia Naveeda (5025211203)





8-QUEENS

Masalah 8 Ratu adalah masalah klasik dalam ilmu komputer di mana tujuannya adalah untuk menempatkan delapan ratu di papan catur sedemikian rupa sehingga tidak ada dua ratu yang saling menyerang

GENETIC ALGORITHM

Genetics Algorithm adalah jenis algoritma optimisasi yang terinspirasi dari proses seleksi alam. Dalam algoritma genetika, populasi solusi kandidat dikembangkan dari waktu ke waktu untuk menemukan solusi terbaik untuk suatu masalah. Setiap solusi kandidat direpresentasikan sebagai kromosom, yang merupakan rangkaian gen. Dalam kasus Masalah 8-Ratu, kita dapat merepresentasikan setiap kandidat solusi sebagai array dengan panjang 8, di mana setiap elemen larik mewakili posisi baris ratu dalam kolom yang bersesuaian.

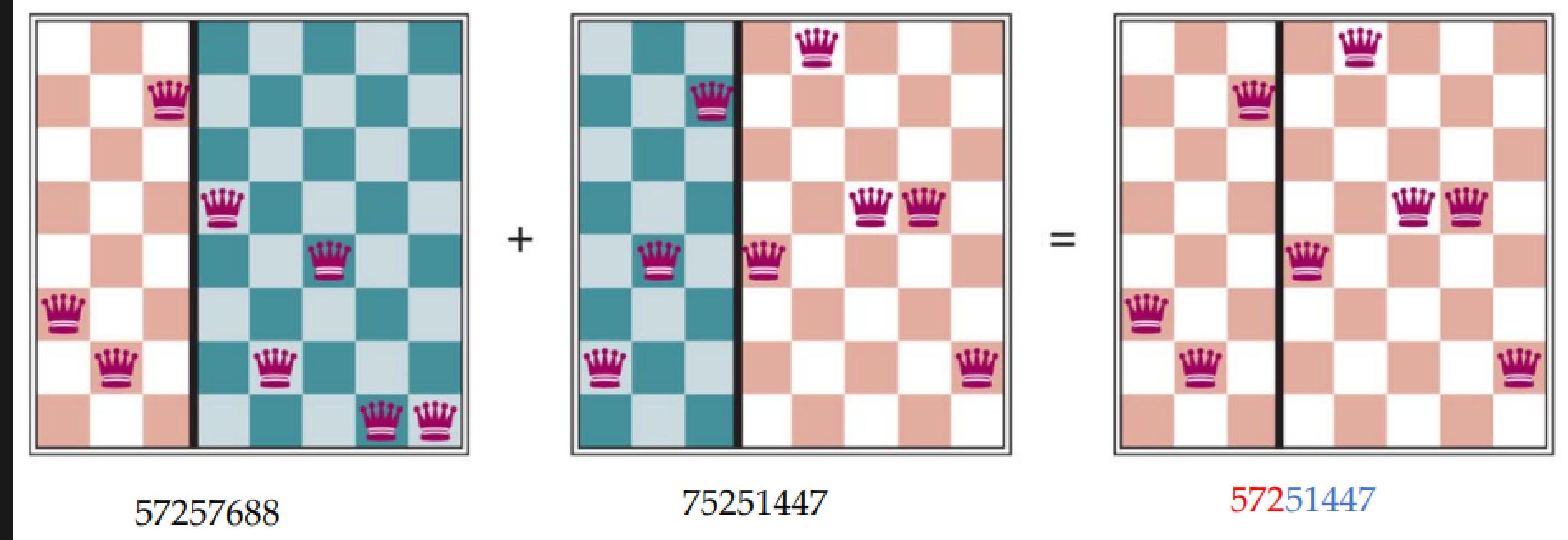
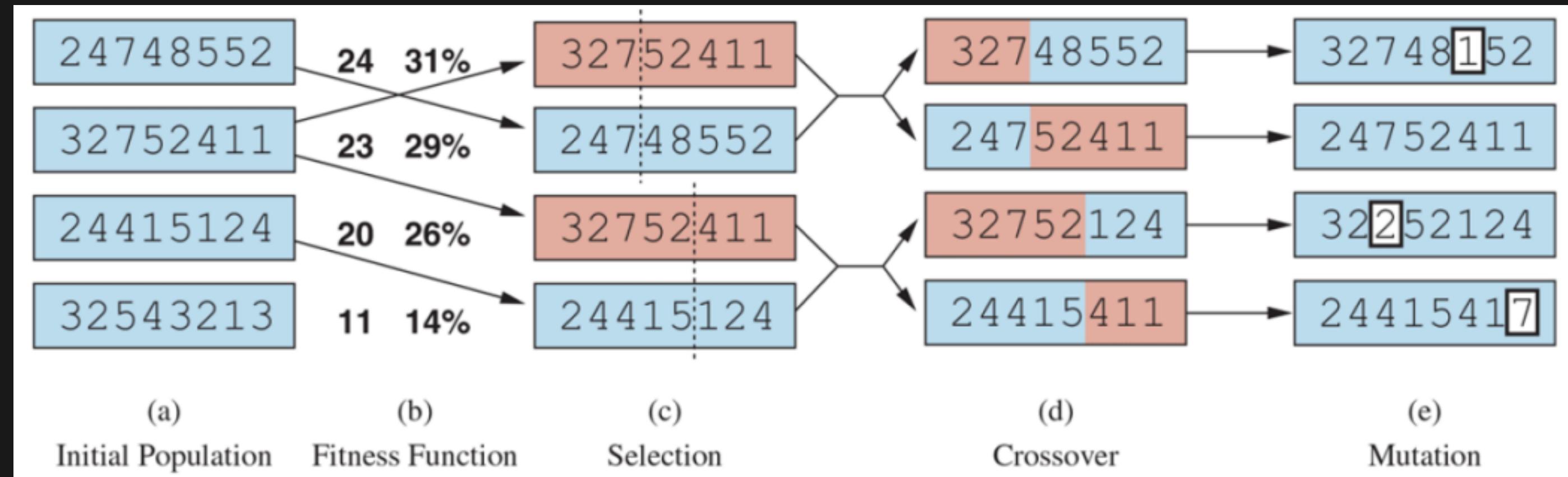
Genetic Algorithm merupakan salah satu evolutionary algorithms yang menggunakan teknik-teknik yang terinspirasi oleh evolutionary biology

1. Representasi genetic sebagai domain solusi
2. Fungsi fitness untuk mengevaluasi domain solusi

GENETIC ALGORITHM

Untuk menerapkan algoritma genetika dalam menyelesaikan 8-Queens Problem, kita dapat mengikuti langkah-langkah berikut:

1. Inisialisasi populasi solusi kandidat acak.
2. Evaluasi setiap solusi kandidat menggunakan fungsi fitness yang menghitung jumlah konflik antara ratu.
3. Pilih individu yang paling cocok dari populasi untuk menjadi parents dari generasi berikutnya.
4. Hasilkan populasi baru dengan menggabungkan kembali gen parents dan memperkenalkan mutasi acak.
5. Ulangi langkah 2-4 hingga solusi yang memuaskan ditemukan atau jumlah maksimum generasi tercapai



IMPLEMENTASI

INISIALISASI

```
import random

# Ukuran papan catur
board_size = 8

# Jumlah populasi
population_size = 50

# Jumlah generasi
max_generations = 1500

# Probabilitas mutasi
mutation_rate = 0.6

def initial_population(size):
    """
    Fungsi untuk menghasilkan populasi awal
    """
    population = []
    for i in range(size):
        chromosome = [random.randint(0, board_size-1) for j in range(board_size)]
        population.append(chromosome)
    return population
```

random library diimport untuk menghasilkan angka acak. board_size adalah ukuran papan catur yang akan dijadikan sebagai konstanta dengan nilai 8. population_size adalah jumlah populasi awal dengan nilai 50. max_generations adalah jumlah generasi yang akan dilakukan dengan nilai 1500. mutation_rate adalah probabilitas mutasi dengan nilai 0.6. Fungsi initial_population(size) akan menghasilkan populasi awal dengan ukuran size. Populasi awal tersebut akan dihasilkan dengan men-generate kromosom secara acak.

```
def fitness(chromosome):
    """
    Fungsi untuk menghitung nilai fitness
    """

    attacks = 0
    for i in range(board_size):
        for j in range(i+1, board_size):
            if chromosome[i] == chromosome[j]:
                attacks += 1
            elif abs(i-j) == abs(chromosome[i]-chromosome[j]):
                attacks += 1
    return board_size*(board_size-1)/2 - attacks
```

FUNGSI FITNESS

Fungsi `fitness(chromosome)` akan menghitung nilai fitness dari suatu kromosom. Untuk menghitung jumlah serangan, fungsi ini akan membandingkan setiap pasangan ratu dan menambahkan 1 pada variabel `attacks` jika terdapat serangan. Nilai fitness dihitung dengan mengurangi total serangan dari jumlah maksimal serangan yang mungkin

```
38 def selection(population):
39     """
40     Fungsi untuk memilih kandidat solusi terbaik
41     """
42     fitnesses = [fitness(chromosome) for chromosome in population]
43     total_fitness = sum(fitnesses)
44     probabilities = [fitness/total_fitness for fitness in fitnesses]
45     selected_index = random.choices(range(len(population)), weights=probabilities)[0]
46     return population[selected_index]
```

FUNGSI SELECTION

memilih kandidat solusi terbaik dari populasi saat ini berdasarkan nilai fitness dari setiap kromosom dalam populasi. Pertama, fungsi ini menghitung total fitness dari seluruh populasi. Kemudian, setiap kromosom diberikan probabilitas dipilih yang sebanding dengan nilai fitness-nya.

FUNGSI CROSSOVER

melakukan operasi crossover pada dua kromosom orang tua untuk menghasilkan dua kromosom anak. Pada implementasi ini, dilakukan single-point crossover, yaitu salah satu titik pada gen dipilih secara acak, dan bagian kiri dan kanan kromosom dipertukarkan antara dua orang tua untuk menghasilkan dua anak.

```
48 def crossover(parent1, parent2):  
49     """  
50     Fungsi untuk melakukan operasi crossover  
51     """  
52     crossover_point = random.randint(0, board_size-1)  
53     child1 = parent1[:crossover_point] + parent2[crossover_point:]  
54     child2 = parent2[:crossover_point] + parent1[crossover_point:]  
55     return child1, child2
```

FUNGSI MUTATION

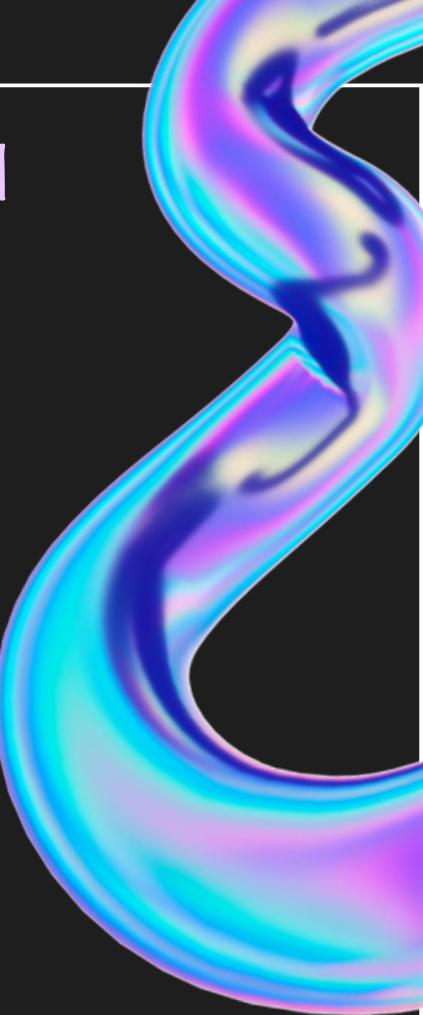
Fungsi mutation digunakan untuk melakukan operasi mutasi pada kromosom anak. Pada implementasi ini, dilakukan mutasi dengan probabilitas mutation_rate pada setiap gen. Jika memenuhi kondisi probabilitas mutasi, maka dilakukan pertukaran posisi dua gen secara acak pada kromosom anak.

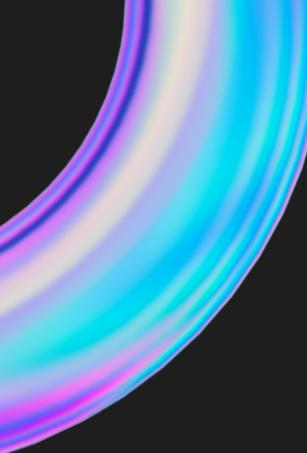
```
57 def mutation(chromosome):
58     """
59     Fungsi untuk melakukan mutasi
60     """
61     mutated_chromosome = chromosome[:]
62     if random.random() < mutation_rate:
63         index1, index2 = random.sample(range(board_size), 2)
64         mutated_chromosome[index1], mutated_chromosome[index2] = \
65             mutated_chromosome[index2], mutated_chromosome[index1]
66     return mutated_chromosome
```

```
68 def genetic_algorithm():
69     """
70     Fungsi untuk menjalankan algoritma genetik
71     """
72     # Inisialisasi populasi awal
73     population = initial_population(population_size)
74
75     # Iterasi sebanyak max_generations
76     for generation in range(max_generations):
77         # Seleksi orang tua
78         parent1 = selection(population)
79         parent2 = selection(population)
80         while parent2 == parent1:
81             parent2 = selection(population)
82
83         # Operasi crossover
84         child1, child2 = crossover(parent1, parent2)
85
86         # Mutasi
87         child1 = mutation(child1)
88         child2 = mutation(child2)
89
90         # Evaluasi fitness
91         children_fitness = [fitness(child1), fitness(child2)]
92
93         # Seleksi kelangsungan hidup
94         worst_index = population.index(min(population, key=fitness))
95         if children_fitness[0] > children_fitness[1]:
96             if children_fitness[0] > fitness(population[worst_index]):
97                 population[worst_index] = child1
98             else:
99                 if children_fitness[1] > fitness(population[worst_index]):
100                     population[worst_index] = child2
101
102         # Cek apakah sudah ditemukan solusi
103         best_index = population.index(max(population, key=fitness))
104         if fitness(population[best_index]) == board_size*(board_size-1)/2:
105             return population[best_index], generation
106
107     # Jika tidak ditemukan solusi
108     best_index = population.index(max(population, key=fitness))
109     return population[best_index], max_generations
```

VYING FOR ATTENTION

fungsi utama yang menjalankan algoritma genetik. Pertama-tama, populasi awal diinisialisasi dengan ukuran sebanyak population_size. Selanjutnya, dilakukan iterasi sebanyak max_generations dan dalam setiap iterasi, dilakukan seleksi orang tua, crossover, mutasi, evaluasi fitness, dan seleksi kelangsungan hidup. Jika ditemukan solusi pada suatu generasi, maka kromosom solusi dan generasi tersebut akan dikembalikan sebagai output. Jika setelah dilakukan iterasi sebanyak max_generations solusi tidak ditemukan, maka kromosom dengan nilai fitness terbaik dari seluruh populasi dan max_generations akan dikembalikan sebagai output.





PENGGUNAAN ALGORITMA GENETIK UNTUK 8-QUEENS PROBLEM

Pada implementasi algoritma genetik untuk 8-Queens problem ini, setiap kromosom pada populasi awal merepresentasikan posisi dari 8 ratu pada papan catur. Kemudian, fungsi fitness dihitung berdasarkan jumlah serangan antar ratu yang terjadi pada papan catur, dan nilai maksimum fitness yang mungkin adalah $(8*7)/2 = 28$, saat tidak ada serangan antar ratu pada papan catur.

Selama proses evolusi, orang tua dipilih dari populasi saat ini berdasarkan probabilitas dipilih yang sebanding dengan nilai fitness-nya. Kemudian, dilakukan operasi crossover dan mutasi pada kromosom anak. Setelah itu, dilakukan evaluasi fitness pada kromosom anak dan dipilih kromosom yang memiliki nilai fitness lebih baik untuk bertahan hidup di populasi. Jika dalam satu generasi ditemukan solusi, maka proses evolusi akan dihentikan dan solusi akan dicetak. Jika tidak ditemukan solusi setelah max_generations iterasi, maka kromosom dengan nilai fitness terbaik akan

OUTPUT

Solusi ditemukan pada generasi ke- 588

```
...Q....  
.....Q.  
..Q.....  
.....Q  
.Q.....  
....Q...  
Q.....  
....Q..
```

Solusi ditemukan pada generasi ke- 728

```
...Q....  
.Q.....  
.....Q.  
....Q...  
Q.....  
.....Q  
....Q..  
..Q.....
```

TERIMA KASIH!