

# 8 Queens using Genetic Algorithm

Kelompok 8

Anggota Kelompok :

Abdurrahman Farimza	5025201125
Kartika Diva Asmara Gita	5025211039
Hana Maheswari	5025211182

# Genetic Algorithm



Algoritma Genetik adalah algoritma optimasi yang menggunakan konsep evolusi dengan seleksi alam untuk mendapatkan solusi optimal dengan memilih solusi terbaik atau terkuat di samping kejadian mutasi yang langka dan acak.

# algoritma

- 
1. Initialization: Membuat populasi awal dari solusi kandidat, seringkali secara acak.
  2. Evaluation: Menilai kebugaran atau kualitas setiap individu dalam populasi berdasarkan fungsi kebugaran yang telah ditentukan sebelumnya.
  3. Selection: Memilih individu yang paling cocok dari populasi untuk menjadi orang tua dari generasi berikutnya.
  4. Reproduction: Menghasilkan solusi kandidat baru melalui operator genetik seperti mutasi dan persilangan.
  5. Termination: Menentukan kapan harus menghentikan algoritme berdasarkan kriteria penghentian yang telah ditentukan sebelumnya, seperti mencapai ambang kebugaran tertentu atau menjalankan sejumlah generasi tertentu.

```
● ● ●  
1 import random  
2 # Constants  
3 BOARD_SIZE = 8  
4 POPULATION_SIZE = 100  
5 MUTATION_RATE = 0.1  
6 GENERATIONS = 1000  
7  
8 def create_population(size):  
9     population = []  
10    for i in range(size):  
11        board = []  
12        for j in range(BOARD_SIZE):  
13            board.append(random.randint(0, BOARD_SIZE-1))  
14        population.append(board)  
15    return population
```

Fungsi `create_population` untuk membuat daftar kosong atau populasi untuk menampung konfigurasi generated board yang kemudian memasuki loop untuk menghasilkan papan acak pada setiap iterasi.



```
1 def calculate_fitness(board):
2     conflicts = 0
3
4     for i in range(BOARD_SIZE):
5         for j in range(i+1, BOARD_SIZE):
6             if board[i] == board[j]:
7                 conflicts += 1
8             if abs(board[i] - board[j]) == abs(i - j):
9                 conflicts += 1
10
11    return 1 / (conflicts + 1)
```

Fungsi `calculate_fitness` untuk menghitung kecocokan setiap individu dalam populasi. Fungsi ini memberikan skor untuk setiap board berdasarkanberapa banyak pasangan ratu yang saling menyerang. Skor didefinisikan sebagai  $1 / (\text{conflicts} + 1)$ , di mana konflik adalah jumlah total konflik antara pasangan ratu di papan.



```
1 def select_parents(population):  
2     fitnesses = [calculate_fitness(board) for board in population]  
3     total_fitness = sum(fitnesses)  
4     probabilities = [fitness / total_fitness for fitness in fitnesses]  
5     parents = random.choices(population, weights=probabilities, k=2)  
6     return parents
```

Fungsi `select_parents` untuk memilih dua parent boards dari populasi saat ini untuk membuat child board.

- `fitnesses` : berisi skor kecocokan untuk setiap board dalam populasi berdasarkan berapa banyak ratu saling menyerang.
- `total_fitness` : jumlah keseluruhan skor fitness.
- `probabilities` : kemungkinan terpilihnya board sebagai parent dengan membagi skor kebugaran dengan totalnya.
- `random.choices` : terdapat parameter `k = 2` untuk memilih 2 parents dan parameter `weight` untuk membobotkan pemilihan berdasarkan probabilitas.



```
1 def create_child(parents):  
2     split_point = random.randint(1, BOARD_SIZE-2)  
3     child = parents[0][:split_point] + parents[1][split_point:]  
4     return child
```

Fungsi `create_child` untuk menghasilkan individu baru / child dengan menggabungkan gen dari 2 parents-nya. Pada fungsi ini, bagian pertama child board diambil dari parent pertama dan bagian kedua child board diambil dari parent kedua.



```
1 def mutate(board):
2     if random.random() < MUTATION_RATE:
3         gene_to_change = random.randint(0, BOARD_SIZE-1)
4         new_gene = random.randint(0, BOARD_SIZE-1)
5         board[gene_to_change] = new_gene
```

Fungsi pertama memeriksa apakah mutasi harus terjadi berdasarkan parameter MUTATION\_RATE. Jika angka acak yang dihasilkan oleh random.random() kurang dari tingkat mutasi, mutasi akan terjadi. Jika mutasi terjadi,

- indeks acak gene\_to\_change dipilih antara 0 dan BOARD\_SIZE-1
- nilai acak baru new\_gene dihasilkan antara 0 dan BOARD\_SIZE-1

Nilai gen pada indeks gene\_to\_change kemudian diubah menjadi nilai baru.



```
1 def evolve(population):
2     new_population = []
3     for i in range(POPULATION_SIZE):
4         parents = select_parents(population)
5         child = create_child(parents)
6         mutate(child)
7         new_population.append(child)
8     return new_population
```

fungsi evolve untuk membuat populasi baru dengan memilih parent board dari populasi saat ini. Selanjutnya, menerapkan operator crossover / create\_child untuk membuat child board baru, dan kemudian memperkenalkan / mutasi sejumlah kecil keacakan ke dalam child board.



```
1 def is_valid(board):
2     conflicts = 0
3     for i in range(BOARD_SIZE):
4         for j in range(i+1, BOARD_SIZE):
5             if board[i] == board[j]:
6                 conflicts += 1
7             if abs(board[i] - board[j]) == abs(i - j):
8                 conflicts += 1
9     return conflicts == 0
```

Fungsi `is_valid` untuk memeriksa konfigurasi board yang diberikan valid (tidak ada dua ratu yang saling menyerang). Pada fungsi ini menginisialisasi variabel `konflik`. Kemudian melakukan loop setiap pasangan ratu di papan dan memeriksa apakah mereka saling menyerang. Jika berada di baris atau kolom yang sama, mereka akan saling menyerang dan konflik akan meningkat.



```
1 def solve():
2     population = create_population(POPULATION_SIZE)
3     for generation in range(GENERATIONS):
4         population = evolve(population)
5         best_board = None
6         best_fitness = 0
7         for board in population:
8             fitness = calculate_fitness(board)
9             if fitness > best_fitness:
10                 best_board = board
11                 best_fitness = fitness
12         print(
13             f"Generation {generation+1}: Best solution
14             {best_board} with fitness {best_fitness}")
15         if is_valid(best_board):
16             return best_board
17
18 return None
```

Fungsi `solve` untuk menerapkan algoritma genetika pada 8 queens puzzle dan menemukan solusi yang valid. Fungsi ini menggunakan fungsi `evolve` untuk membuat konfigurasi board baru dan fungsi `calculate_fitness` untuk menilai skor setiap papan. Kemudian cetak board terbaik yang ditemukan pada setiap generasi dan kembali ke fungsi `is_valid` jika ada.

```
Generation 1: Best solution [4, 1, 7, 3, 0, 6, 2, 6] with fitness 0.2
Generation 2: Best solution [7, 5, 1, 6, 0, 0, 6, 4] with fitness 0.3333333333333333
Generation 3: Best solution [4, 0, 5, 3, 6, 6, 2, 1] with fitness 0.3333333333333333
Generation 4: Best solution [7, 5, 1, 6, 0, 0, 7, 4] with fitness 0.3333333333333333
Generation 5: Best solution [6, 2, 5, 3, 0, 7, 4, 6] with fitness 0.3333333333333333
Generation 6: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 7: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 8: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 9: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 10: Best solution [2, 7, 1, 6, 0, 0, 7, 4] with fitness 0.3333333333333333
Generation 11: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 12: Best solution [2, 5, 1, 6, 4, 0, 7, 4] with fitness 0.5
Generation 13: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 14: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 15: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 16: Best solution [2, 5, 1, 6, 3, 0, 7, 4] with fitness 0.5
Generation 17: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 18: Best solution [2, 5, 1, 6, 3, 0, 7, 4] with fitness 0.5
Generation 19: Best solution [2, 5, 1, 6, 0, 0, 7, 4] with fitness 0.5
Generation 20: Best solution [2, 5, 1, 6, 0, 3, 7, 4] with fitness 1.0
Found solution: [2, 5, 1, 6, 0, 3, 7, 4]
```



TERIMA KASIH