

REPORT FINAL PROJECT ETS LOCAL SEARCH ALGORITHM

KELOMPOK 7

Hill-Climbing & Genetic Algorithm — 8-Queens



Anggota :

Akmal Ariq Romadhon	(5025211188)
Sandyatama Fransisna Nugraha	(5025211196)
Sony Hermawan	(5025211226)
Rafi Aliefian Putra Ramadhani	(5025211234)

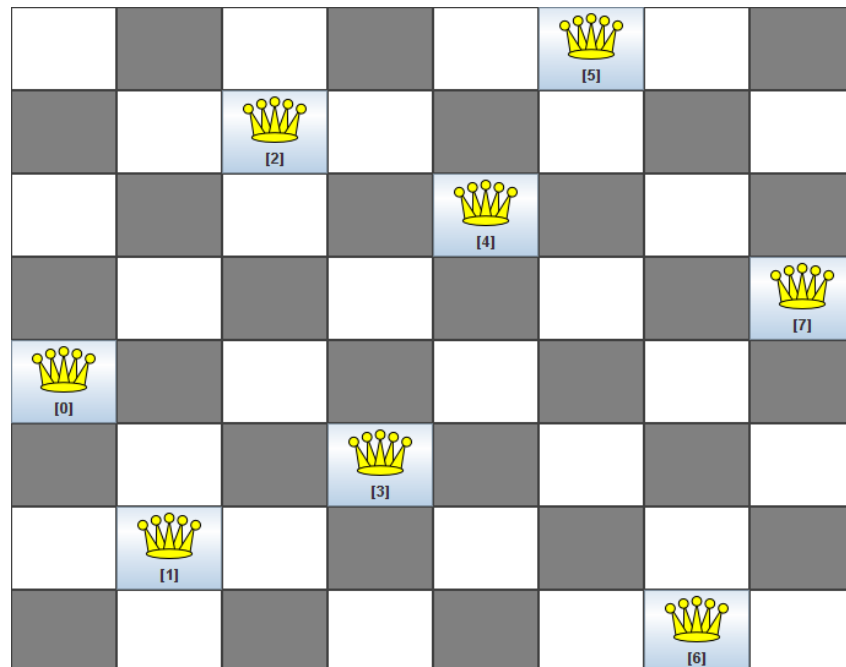
Mata Kuliah Kecerdasan Buatan (D)
FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

2023

- **Topik Permasalahan**

8-Queens Implementation

- **Gambaran Permasalahan**



- **Deskripsi dan Tujuan Permasalahan**

8-queens merupakan sebuah permasalahan dalam matematika dan ilmu komputer yang sering digunakan sebagai kasus uji dalam penyelesaian masalah optimasi kombinatorial dan algoritma pemecahan masalah. Tujuan dari permasalahan ini adalah menempatkan 8 ratu pada papan catur 8x8 sehingga **tidak ada dua ratu yang saling menyerang**, baik secara horizontal, vertikal, maupun diagonal.

Dalam papan catur 8x8 terdapat 64 kotak, dan terdapat 8 baris dan 8 kolom. Setiap ratu harus ditempatkan pada satu baris dan satu kolom yang berbeda. Jika terdapat dua atau lebih ratu pada baris atau kolom yang sama, maka ratu-ratu tersebut saling menyerang secara horizontal dan vertikal. Sedangkan jika terdapat dua atau lebih ratu pada diagonal yang sama, maka ratu-ratu tersebut saling menyerang secara diagonal.

- **Solusi Permasalahan (Algoritma *Hill-Climbing*)**

- **Penjelasan dan langkah-langkah:**

Untuk menyelesaikan permasalahan 8-queens, tentu kita dapat menggunakan berbagai teknik dan algoritma. Salah satu teknik yang kami gunakan yaitu teknik **hill-climbing**. Teknik hill-climbing adalah salah satu algoritma pencarian heuristik yang digunakan dalam menyelesaikan masalah optimasi. Algoritma ini mengambil keputusan berdasarkan langkah yang optimal pada saat itu, tanpa mempertimbangkan konsekuensi jangka panjang. Pada awalnya, algoritma hill-climbing memulai dengan menempatkan ratu secara acak pada papan catur dan menghitung jumlah konflik (atau serangan) yang terjadi. Kemudian, algoritma mencari tetangga-tetangga dari konfigurasi saat ini dengan mengubah posisi satu atau dua ratu pada papan catur. Setiap tetangga dihitung jumlah konfliknya dan dipilih yang memiliki jumlah konflik terendah.

Jika ada tetangga yang memiliki jumlah konflik lebih sedikit daripada konfigurasi saat ini, algoritma akan beralih ke tetangga tersebut dan mengulang proses pencarian tetangga hingga solusi optimal ditemukan. Jika tidak ada tetangga yang memiliki jumlah konflik lebih sedikit, maka algoritma dianggap mencapai puncak lokal dan mengulang proses pencarian dari konfigurasi awal yang berbeda.

Proses pencarian diulang hingga solusi optimal ditemukan atau mencapai batas iterasi yang ditentukan. Algoritma hill-climbing pada kasus 8-queens akan mengembalikan konfigurasi papan catur yang menempatkan 8 ratu tanpa saling menyerang dan merupakan solusi optimal untuk masalah tersebut.

- **Langkah-langkah dalam bentuk poin:**

1. Mulai dengan menempatkan 8 queen secara acak pada papan catur yang berukuran 8x8.
2. Evaluasi solusi awal dengan menghitung jumlah konflik antar queen pada papan catur.
3. Cek apakah jumlah konflik sudah sama dengan 0. Jika sudah, kembalikan solusi saat ini.
4. Dapatkan semua solusi tetangga dengan melakukan perubahan satu posisi pada satu queen.
5. Evaluasi setiap solusi tetangga dan pilih solusi tetangga dengan jumlah konflik terkecil.
6. Cek apakah jumlah konflik solusi tetangga terkecil lebih besar atau sama dengan jumlah konflik solusi saat ini. Jika iya, kembalikan solusi saat ini karena sudah mencapai titik lokal optimal. Jika tidak, pindah ke solusi tetangga terbaik.

7. Ulangi langkah 3 sampai 6 hingga tidak ada lagi solusi tetangga dengan jumlah konflik lebih kecil.

- **Implementasi Algoritma Permasalahan**

- **Bahasa Pemrograman:**

- Python (.py)

- **Source Code:**

```
import random
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def generate_state():
    """Generate a random initial state"""

    # [[0]](https://www.geeksforgeeks.org/n-queen-problem-local-search-using-hill-climbing-with-random-neighbour/)
    state = [-1] * 8
    for i in range(8):
        while True:
            j = random.randint(0, 7)
            if j not in state[:i]:
                state[i] = j
                break
    return state

def calculate_fitness(state):
    """Calculate the fitness score of a state"""

    # [[0]](https://www.geeksforgeeks.org/n-queen-problem-local-search-using-hill-climbing-with-random-neighbour/)
    conflicts = 0
    for i in range(8):
        for j in range(i+1, 8):
            if state[i] == state[j] or abs(state[i] - state[j]) == j - i:
                conflicts += 1
    return 28 - conflicts

def get_neighbors(state):
    """Generate all possible next states"""

    # [[0]](https://www.geeksforgeeks.org/n-queen-problem-local-search-using-hill-climbing-with-random-neighbour/)
    neighbors = []
    for i in range(8):
        for j in range(8):
            if j != state[i]:
                neighbor = state[:]
                neighbor[i] = j
                neighbors.append(neighbor)
    return neighbors
```

```

def hill_climbing(allow_sideways=True, max_sideways=100,
max_restarts=20):
    """Run the hill climbing algorithm"""

    # [[0]](https://www.geeksforgeeks.org/n-queen-problem-local-search-using-hill-climbing-with-random-neighbour/)
    best_solution = None
    best_fitness = -1
    restarts = 0

    while restarts < max_restarts:
        current_state = generate_state()
        current_fitness = calculate_fitness(current_state)
        sideways_moves = 0

        while True:
            neighbors = get_neighbors(current_state)
            if not neighbors:
                break

            neighbor_fitnesses = [calculate_fitness(neighbor) for
neighbor in neighbors]
            best_neighbor = neighbors[np.argmax(
neighbor_fitnesses)]
            best_local_fitness = max(neighbor_fitnesses)

            if best_local_fitness < current_fitness:
                break
            elif best_local_fitness == current_fitness:
                if not allow_sideways or sideways_moves >=
max_sideways:
                    break
                sideways_moves += 1
            else:
                sideways_moves = 0

            current_state = best_neighbor
            current_fitness = best_local_fitness

            if current_fitness > best_fitness:
                best_solution = current_state
                best_fitness = current_fitness

            if best_fitness == 28:
                break

            restarts += 1

    return best_solution

```



```
solution = hill_climbing()
print('Solution:')
for row, col in enumerate(solution):
    print(f'[{row}, {col}]')

# Display the final state using NumPy and Matplotlib
board = np.zeros((8, 8))
for i in range(8):
    board[solution[i], i] = 1
plt.figure(figsize=(8, 8))
sns.set(font_scale=1.5)
sns.heatmap(board, cmap='Purples', annot=True, cbar=False, square
=True, linewidths=.5, linecolor='black', xticklabels=['0', '1',
'2', '3', '4', '5', '6', '7'], yticklabels=['0', '1', '2', '3',
'4', '5', '6', '7'])
plt.xlabel('Column')
plt.ylabel('Row')
plt.show()
```

○ Hasil *running*:

Row	0	1	0	0	0	0	0	0
	1	0	0	0	0	1	0	0
	2	0	0	0	0	0	0	1
	3	0	0	1	0	0	0	0
	4	0	0	0	0	0	1	0
	5	0	0	0	1	0	0	0
	6	0	1	0	0	0	0	0
	7	0	0	0	0	1	0	0
		Column						

```
PS D:\Vscode> & C:/Python311/python.exe "d:/Vscode/Semester 4/Kabe/hillClimbing3.py"
Solution:
[0, 0]
[1, 6]
[2, 3]
[3, 5]
[4, 7]
[5, 1]
[6, 4]
[7, 2]
○ PS D:\Vscode> □
```

○ **Penjelasan Code**

Kode tersebut merupakan implementasi dari algoritma Hill Climbing untuk menyelesaikan masalah 8 Queen. Pada kode tersebut, terdapat beberapa fungsi yang digunakan untuk menyelesaikan masalah 8 Queen. Fungsi pertama yaitu `generate_state()` berfungsi untuk menghasilkan sebuah state acak (posisi awal ratu) pada papan catur. Fungsi ini akan menghasilkan sebuah list berisi angka antara 0-7 yang merepresentasikan posisi kolom dari masing-masing ratu pada setiap barisnya.

Fungsi kedua yaitu `calculate_fitness(state)` berfungsi untuk menghitung fitness score dari suatu state. Fitness score dihitung dengan cara menghitung jumlah konflik antar ratu pada suatu state. Konflik terjadi jika ada dua atau lebih ratu yang berada pada baris, kolom, atau diagonal yang sama. Jumlah konflik yang ditemukan akan dikurangi dari total kemungkinan konflik antara 8 ratu. Kesimpulannya ialah semakin sedikit konflik, maka semakin tinggi fitness score-nya.

Fungsi ketiga yaitu `get_neighbors(state)` berfungsi untuk menghasilkan semua kemungkinan state yang dapat dihasilkan dari state yang diberikan dengan cara memindahkan satu ratu dari satu kolom ke kolom yang lain. Fungsi ini menghasilkan sebuah list yang berisi semua kemungkinan state.

Fungsi keempat yaitu `hill_climbing(allow_sideways=True, max_sideways=100, max_restarts=20)` merupakan inti dari algoritma Hill Climbing. Fungsi ini akan menghasilkan solusi akhir dari masalah 8 Queen dengan menggunakan algoritma Hill Climbing. Fungsi ini memiliki beberapa parameter seperti `allow_sideways`, `max_sideways`, dan `max_restarts`. Parameter ini digunakan untuk mengontrol kondisi seperti apakah solusi yang memiliki fitness score sama dengan solusi saat ini boleh dipilih (`allow_sideways`), berapa kali boleh melakukan gerakan horizontal (`max_sideways`), dan berapa kali restart algoritma jika belum ditemukan solusi (`max_restarts`).

Fungsi terakhir yaitu `solution = hill_climbing()` berfungsi untuk mengeksekusi algoritma Hill Climbing untuk menyelesaikan masalah 8 Queen dan menghasilkan solusi akhirnya. Solusi akhir ini akan ditampilkan pada output dalam bentuk [baris, kolom] yang merepresentasikan posisi masing-masing ratu pada setiap barisnya. Selain itu, kode juga akan menampilkan papan catur dan meletakkan tanda pada posisi masing-masing ratu pada papan catur menggunakan heatmap pada Matplotlib.

- **Solusi Permasalahan (*Genetic Algorithm*)**

- **Penjelasan dan Langkah-Langkah**

Untuk menyelesaikan permasalahan 8-queens, tentu kita juga dapat menggunakan berbagai teknik dan algoritma yang lainnya. Salah satu teknik yang kami gunakan yaitu teknik **Genetic-Algorithm**. Dalam genetic algorithm, individu-individu dalam populasi direpresentasikan sebagai kumpulan genetik yang menggambarkan posisi ratu pada papan catur. Populasi awal dibentuk dengan cara menghasilkan beberapa individu secara acak atau menggunakan metode heuristik. Kemudian, setiap individu dievaluasi menggunakan fungsi fitness yang menghitung jumlah konflik antar ratu. Semakin sedikit konflik yang terjadi, semakin baik fitness nya. Proses seleksi dilakukan untuk memilih individu-individu dengan fitness yang lebih tinggi sebagai orang tua untuk reproduksi. Pasangan orang tua kemudian menjalani proses crossover, di mana gen-genenya ditukar untuk menghasilkan keturunan baru. Beberapa elemen dalam keturunan dapat dimutasi dengan probabilitas rendah untuk memperkenalkan variasi baru. Individu-individu baru ini kemudian menggantikan individu-individu yang lebih buruk dalam populasi sebelumnya.

Proses ini berulang dalam beberapa generasi hingga ditemukan solusi yang memenuhi kriteria, yaitu tidak ada konflik antar ratu pada papan catur, atau hingga mencapai batas generasi maksimum yang ditentukan sebelumnya. Dengan demikian, melalui iterasi langkah-langkah tersebut, genetic algorithm berusaha mencari solusi optimal untuk permasalahan 8-queens dengan memanfaatkan konsep seleksi alam yang **terinspirasi** dari **evolusi biologi**.

- **Langkah-langkah dalam bentuk poin:**

1. Inisialisasi: Bentuk populasi awal dengan menghasilkan individu-individu secara acak atau menggunakan metode heuristik.
2. Evaluasi Fitness: Hitung fitness untuk setiap individu dalam populasi berdasarkan jumlah konflik antar ratu. Semakin sedikit konflik, semakin tinggi fitnessnya.
3. Seleksi: Pilih individu-individu dengan fitness yang lebih tinggi sebagai orang tua untuk reproduksi. Metode seleksi yang umum digunakan adalah seleksi roulette wheel atau turnamen seleksi.
4. Crossover: Gabungkan gen-gennya dari pasangan orang tua yang dipilih pada langkah seleksi untuk menghasilkan keturunan baru. Misalnya, menggunakan metode one-point crossover, tentukan titik pemotongan genetik secara acak dan tukar gen-gennya antara orang tua.

5. Mutasi: Dalam beberapa keturunan yang dihasilkan, lakukan mutasi dengan probabilitas rendah untuk memperkenalkan variasi baru. Misalnya, ubah nilainya atau tukar dengan gen acak.
6. Penggantian Generasi: Gantikan individu-individu yang kurang baik dalam populasi dengan individu-individu baru yang dihasilkan dari langkah crossover dan mutasi.-
7. Kriteria Berhenti: Berhenti jika telah ditemukan solusi yang memenuhi kriteria, yaitu tidak ada konflik antar ratu, atau jika telah mencapai batas generasi maksimum yang ditentukan sebelumnya.
8. Ulangi Langkah 2-7: Iterasikan langkah-langkah 2 hingga 7 dalam beberapa generasi untuk mencari solusi yang optimal.

- **Implementasi Algoritma Permasalahan**

- **Bahasa Pemrograman:**

- Python (.py)

- **Source Code:**


```
import random
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

POPULATION_SIZE = 500
MUTATION_RATE = 0.1
GENERATIONS = 1000

def generate_state():
    state = list(range(8))
    random.shuffle(state)
    return state

def calculate_fitness(state):
    conflicts = 0
    for i in range(8):
        for j in range(i+1, 8):
            if state[i] == state[j] or abs(
state[i] - state[j]) == j - i:
                conflicts += 1
    return 1.0 / (1 + conflicts)
# Modified fitness function

def crossover(parent1, parent2):
    crossover_point = random.randint(0, 7)
    child1 = parent1[:crossover_point] +
parent2[crossover_point:]
    child2 = parent2[:crossover_point] +
parent1[crossover_point:]
    return child1, child2
```



```

def mutate(state):
    if random.random() < MUTATION_RATE:
        i, j = random.sample(range(8), 2)
        state[i], state[j] = state[j], state[i]
    return state

def select_parents(population):
    tournament_size = 5
    tournament = random.sample(population, tournament_size)
    return max(tournament, key=lambda state: calculate_fitness(state))

def generate_population():
    return [generate_state() for _ in range(POPULATION_SIZE)]

def evolve_population(population):
    new_population = []
    for _ in range(POPULATION_SIZE // 2):
        parent1 = select_parents(population)
        parent2 = select_parents(population)
        child1, child2 = crossover(parent1, parent2)
        child1 = mutate(child1)
        child2 = mutate(child2)
        new_population.extend([child1, child2])
    return new_population

def genetic_algorithm():
    population = generate_population()
    for i in range(GENERATIONS):
        population = evolve_population(population)
        best_solution = max(population, key=lambda state: calculate_fitness(state))
        if calculate_fitness(best_solution) == 1: # Modified stopping condition
            return best_solution
    return best_solution

```

```

solution = genetic_algorithm()
print('Solution:')
for row, col in enumerate(solution):
    print(f'[{row}, {col}]')

board = np.zeros((8, 8))
for i in range(8):
    board[solution[i], i] = 1
plt.figure(figsize=(8, 8))
sns.set(font_scale=1.5)
sns.heatmap(board, cmap='Reds', annot=True,
cbar=False, square=True, linewidths=.5,
linecolor='black', xticklabels=['0', '1', '2',
'3', '4', '5', '6', '7'], yticklabels=['0', '1',
'2', '3', '4', '5', '6', '7'])
plt.xlabel('Column')
plt.ylabel('Row')
plt.show()

```

○ Hasil Running:

0	0	0	0	1	0	0	0	
1	0	0	1	0	0	0	0	
2	1	0	0	0	0	0	0	
3	0	0	0	0	1	0	0	
4	0	0	0	0	0	0	1	
5	0	1	0	0	0	0	0	
6	0	0	0	1	0	0	0	
7	0	0	0	0	0	1	0	
	0	1	2	3	4	5	6	7

Column

```

PS D:\Vscode\Semester 4> & C:/Python311/python.exe "d:/Vscode/Semester 4/Kabe/geneticAlgorithm2.py"
Solution:
[0, 2]
[1, 5]
[2, 1]
[3, 6]
[4, 0]
[5, 3]
[6, 7]
[7, 4]
○ PS D:\Vscode\Semester 4>

```

- **Penjelasan Code:**

Dalam kode tersebut, diimplementasikan *genetic algorithm* untuk menyelesaikan masalah N-Queens. *genetic algorithm* sendiri adalah algoritma yang terinspirasi dari konsep seleksi alam dalam evolusi biologi untuk menyelesaikan masalah optimasi.

Pertama-tama, terdapat beberapa modul yang di- untuk mengolah data dan membuat visualisasi. Selanjutnya, terdapat deklarasi variabel, yaitu `POPULATION_SIZE`, `MUTATION_RATE`, dan `GENERATIONS`. `POPULATION_SIZE` adalah ukuran populasi yang akan dibuat, `MUTATION_RATE` adalah probabilitas mutasi, dan `GENERATIONS` adalah jumlah iterasi dalam evolusi populasi.

Selanjutnya, beberapa fungsi dibuat untuk membantu menjalankan *genetic algorithm*. Fungsi `generate_state` menghasilkan keadaan awal secara acak, sementara fungsi `calculate_fitness` menghitung nilai fitness dari sebuah keadaan. Fungsi `crossover` digunakan untuk melakukan operasi crossover antara dua *parents*, dan fungsi `mutate` digunakan untuk memperkenalkan variasi pada populasi.

Fungsi `select_parents` menggunakan *tournament selection* untuk memilih dua *parents* dari populasi. Pada metode *tournament selection*, sejumlah kecil keadaan diambil secara acak dan diadu untuk memilih keadaan terbaik. Setelah dipilih, fungsi `crossover` dan `mutate` diterapkan untuk menghasilkan *childs*.

Fungsi `generate_population` menghasilkan populasi awal secara acak dengan ukuran sebesar `POPULATION_SIZE`, sedangkan fungsi `evolve_population` memperbaharui populasi dengan menjalankan metode `select_parents`, `crossover`, dan `mutate` pada setiap iterasi.

Fungsi `genetic_algorithm` adalah inti dari *genetic algorithm*. Pada fungsi ini, populasi awal dihasilkan dengan memanggil fungsi `generate_population`, dan selanjutnya, fungsi `evolve_population` dijalankan sebanyak `GENERATIONS` kali atau sampai menemukan solusi optimal. Terakhir, solusi yang ditemukan ditampilkan dalam bentuk papan catur berukuran 8x8 menggunakan `matplotlib` dan `seaborn`, di mana setiap lokasi pada papan catur yang diisi oleh ratu ditandai dengan warna merah.

Dengan kode tersebut, masalah N-Queens dapat diselesaikan secara efisien dan efektif dengan bantuan *genetic algorithm*.

- **Kesimpulan:**

Berdasarkan penerapan *genetic algorithm* pada permasalahan 8-queens, dapat disimpulkan bahwa metode ini efektif dalam mencari solusi optimal dengan meminimalkan konflik antar ratu. Genetic algorithm memiliki kemampuan untuk mengeksplorasi ruang pencarian yang lebih luas dibandingkan dengan algoritma hill-climbing, yang memungkinkannya menemukan solusi yang lebih baik. Dalam perbandingan performa, genetic algorithm cenderung memberikan hasil yang lebih baik, terutama pada permasalahan dengan ruang pencarian yang kompleks dan memiliki banyak local optimum yang membingungkan.

Namun, genetic algorithm juga memiliki beberapa kelemahan. Salah satunya adalah waktu komputasi yang lebih lama dibandingkan dengan algoritma hill-climbing. Hal ini disebabkan oleh langkah-langkah seperti seleksi, crossover, dan mutasi yang harus diulang dalam beberapa generasi. Oleh karena itu, dalam kasus permasalahan 8-queens yang relatif kecil, genetic algorithm dapat memberikan solusi dalam waktu yang masuk akal. Namun, pada permasalahan dengan ukuran yang lebih besar, waktu yang dibutuhkan untuk menemukan solusi optimal mungkin akan meningkat secara signifikan.

Dalam hal akurasi, genetic algorithm memiliki potensi untuk mencapai solusi yang optimal. Namun, akurasi dari metode ini sangat tergantung pada inisialisasi populasi awal. Jika populasi awal tidak memiliki individu-individu yang cukup baik atau variasi genetik yang mencukupi, maka kemungkinan untuk mencapai solusi optimal akan berkurang. Oleh karena itu, pemilihan parameter dan teknik inisialisasi yang tepat sangat penting dalam meningkatkan akurasi genetic algorithm.

Secara keseluruhan, genetic algorithm adalah metode yang efektif dan dapat diandalkan dalam menyelesaikan permasalahan 8-queens. Meskipun memerlukan waktu komputasi yang lebih lama dan membutuhkan penyesuaian parameter yang cermat, genetic algorithm memiliki potensi untuk memberikan solusi optimal dengan kemampuan untuk mengeksplorasi ruang pencarian yang lebih luas. Dalam konteks permasalahan 8-queens, genetic algorithm dapat memberikan performa yang baik dengan tingkat akurasi yang memadai.