

Tugas 3

8-Queen *using* Local Search

Kecerdasan Buatan F - Kelompok Cucur Adabi

Anggota Kelompok Cucur Adabi



Rizky Alfiyah Rahma
502521208



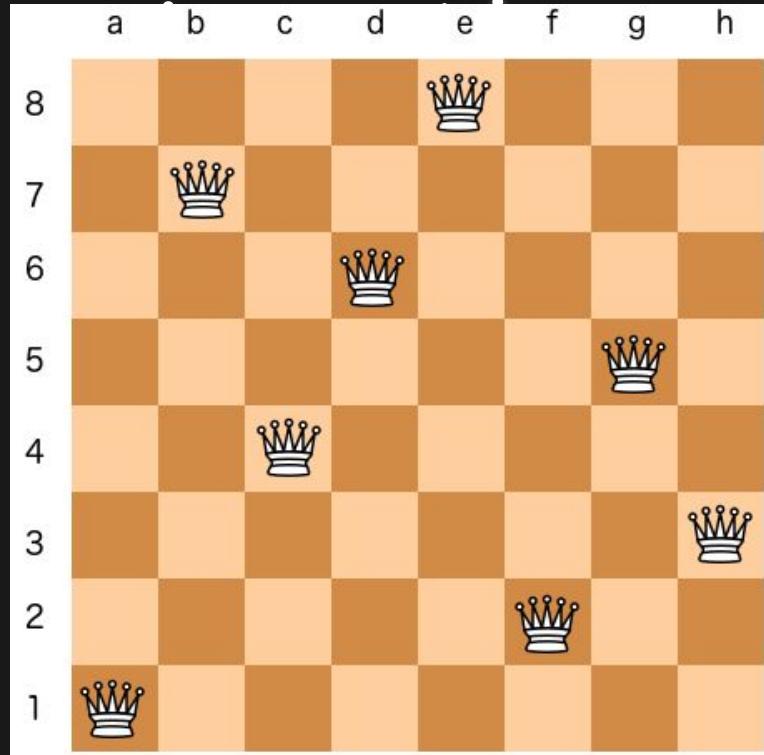
Salsabila Fatma Ayqa
5025211057



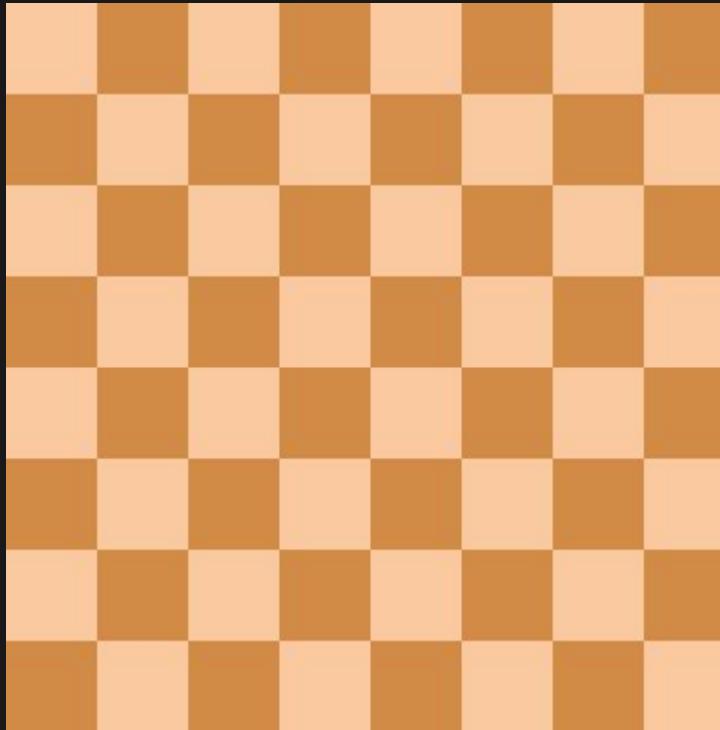
Tsabita Putri Ramadhany
5025211130

8-Queen *Puzzle*

- 8 Queens Puzzle merupakan sebuah problem di mana 8 queens yang diletakkan di papan catur tidak mengancam satu sama lain , baik itu secara vertikal, horizontal, dan diagonal



8-Queen Illustration



01

Local Search Hill
Climbing



Hill Climbing

Pendekatan pertama yang akan kami lakukan adalah **Local**

- **Search** menggunakan **Algoritma Hill Climbing**. Hill Climbing Algorithm merupakan sebuah Heuristic Search yang digunakan untuk mengoptimasi sebuah masalah secara sistematis di lingkup Kecerdasan Buatan.

Diberikan sebuah input yang besar dengan heuristic function yang bagus, program ini akan berjalan untuk mencari solusi yang terbaik

- **Local Maximum**

State yang lebih baik dari state sekitar, namun bukan yang terbaik

- **Global Maximum**

State terbaik yang memiliki objective value tertinggi

- **Current State**

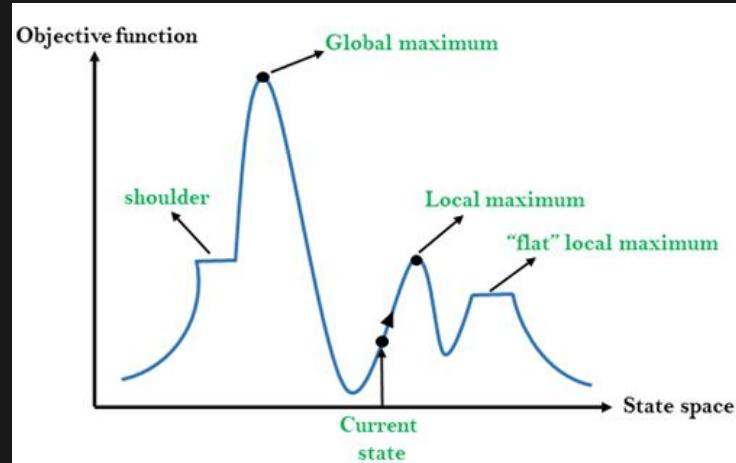
State dalam diagram dimana agent saat ini

- **Flat Local Maximum**

State dimana state sekitar memiliki nilai yang sama

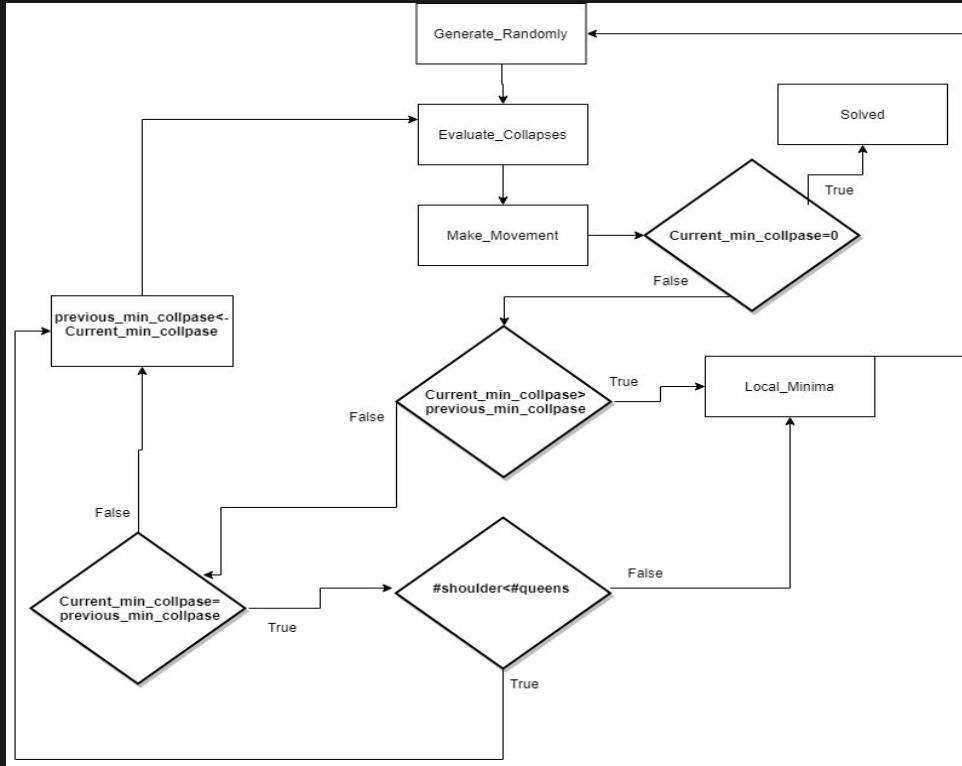
- **Shoulder**

space dengan area datar yang membuat algoritma pencarian tidak punya arah



Pada kasus 8-Queen Puzzle, pergerakan Queen diasumsikan sama halnya dengan mereka dapat berjalan hanya dengan kolom yang dimilikinya demi membatasi pergerakan. Oleh karena itu, perhitungan collapse sederhana dan lebih struktural

Hill Climbing Algorithm *Flowchart*



Implementasi

```
# Inisialisasi papan catur dan posisi awal queen
board_size = 8
current_state = start_state = [1, 3, 0, 6, 4, 7, 5, 2]

# Definisi fungsi untuk menghitung jumlah konflik pada papan catur
def count_conflicts(state):
    conflicts = 0
    for i in range(len(state)):
        for j in range(i+1, len(state)):
            if state[i] == state[j] or abs(state[i]-state[j]) == j-i:
                conflicts += 1
    return conflicts

# Definisi fungsi untuk mencari posisi queen yang dapat dipindahkan untuk meminimalkan konflik
def find_best_move(state):
    best_move = None
    min_conflicts = board_size
    for i in range(len(state)):
        for j in range(board_size):
            if j != state[i]:
                new_state = state.copy()
                new_state[i] = j
                new_conflicts = count_conflicts(new_state)
                if new_conflicts < min_conflicts:
                    best_move = (i, j)
                    min_conflicts = new_conflicts
    return best_move
```

```
# Definisi fungsi utama untuk menyelesaikan 8 queen tidak
# saling menyerang dengan hill climbing
def hill_climbing(state):
    while True:
        best_move = find_best_move(state)
        if best_move is None:
            break
        i, j = best_move
        state[i] = j
        if count_conflicts(state) == 0:
            return state
    return None
```

```
# Menyelesaikan 8 queen tidak saling menyerang dengan hill climbing
# dan mencetak hasilnya
print("Posisi awal:", start_state)
solution = hill_climbing(current_state)
if solution is not None:
    print("Solusi ditemukan:", solution)
else:
    print("Tidak ada solusi yang ditemukan.")
```

Hasil

Posisi awal: [1, 3, 0, 6, 4, 7, 5, 2]

Solusi ditemukan: [4, 2, 0, 6, 1, 7, 5, 3]

02

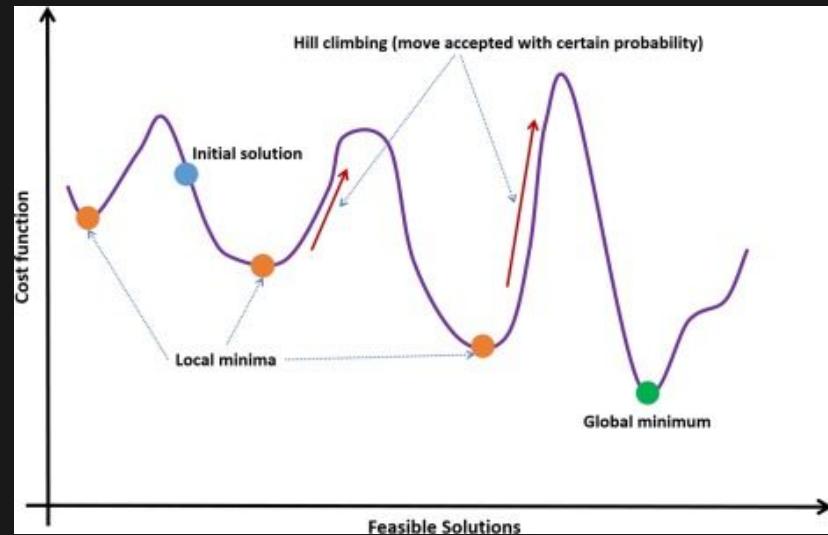
Local Search
**Simulated
Annealing**



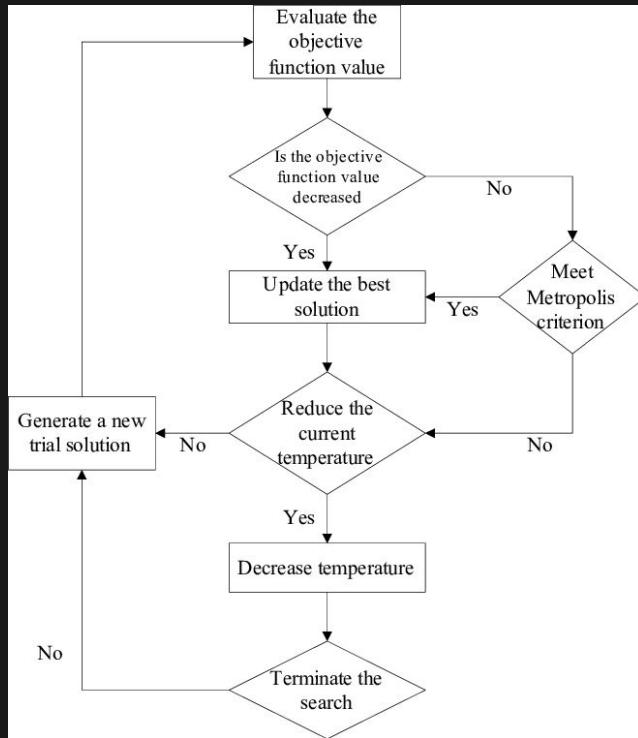
Simulated Annealing

Pendekatan kedua yang akan kami lakukan adalah **Local Search** menggunakan **Algoritma Simulated Annealing**. Simulated Annealing (SA) merupakan suatu pendekatan algoritma untuk memecahkan masalah optimasi kombinatorial. Algoritma SA mengeluarkan Local Minimum dengan menggunakan bilangan acak dalam pemilihan perpindahan.

Graph partition yang dimiliki oleh SA kurang lebih sama dengan Hill Climbing. Algoritma Hill Climbing menunjukkan grafik yang cenderung memuncak, sehingga yang dimiliki oleh SA adalah Local Minimum dan Global Minimum alih-alih Local Maximum dan Global Maximum



Simulated Annealing Flowchart



Implementasi

```
# fungsi untuk menghitung jumlah queen yang saling menyerang pada papan catur
def calculate_conflicts(state):
    conflicts = 0
    for i in range(len(state)):
        for j in range(i + 1, len(state)):
            if state[i] == state[j] or abs(state[i] - state[j]) == j - i:
                conflicts += 1
    return conflicts

# fungsi untuk menghasilkan kemungkinan solusi baru dengan mengganti
# posisi dua queen
def generate_neighbor(state):
    new_state = state.copy()
    i, j = random.sample(range(len(state)), 2)
    new_state[i], new_state[j] = new_state[j], new_state[i]
    return new_state

# fungsi untuk menentukan apakah solusi baru diterima atau tidak
def accept_solution(current_conflicts, new_conflicts, temperature):
    if new_conflicts < current_conflicts:
        return True
    delta = new_conflicts - current_conflicts
    probability = math.exp(-delta / temperature)
    return random.random() < probability
```

```
# inisialisasi posisi awal queen secara acak
start_state = [1, 3, 0, 6, 4, 7, 5, 2]
current_state = start_state

# konfigurasi parameter algoritma simulated annealing
initial_temperature = 1000
temperature_factor = 0.95
iterations_per_temperature = 100

# iterasi algoritma simulated annealing
temperature = initial_temperature
current_conflicts = calculate_conflicts(current_state)
while current_conflicts > 0 and temperature > 0.1:
    for i in range(iterations_per_temperature):
        new_state = generate_neighbor(current_state)
        new_conflicts = calculate_conflicts(new_state)
        if accept_solution(current_conflicts, new_conflicts, temperature):
            current_state = new_state
            current_conflicts = new_conflicts
    temperature *= temperature_factor

# output hasil solusi yang ditemukan
if current_conflicts == 0:
    print("posisi awal:", start_state)
    print("Solusi ditemukan:", current_state)
else:
    print("posisi awal:", start_state)
    print("Tidak ada solusi yang ditemukan.")
```

Hasil

```
posisi awal: [1, 3, 0, 6, 4, 7, 5, 2]  
Solusi ditemukan: [1, 3, 5, 7, 2, 0, 6, 4]
```

Thank You!

Kecerdasan Buatan - Kelompok cucur Adabi

