

# **Tugas Besar 1 IF3070 Dasar Inteligensi Artifisial**

## **Pencarian Solusi Diagonal Magic Cube dengan Local Search**



**Disusun oleh  
Kelompok 3:**

Raizan Iqbal Resi (18222068)

Favian Izza Diasputra (18222070)

Ahmad Habibie Marjan (18222082)

Muhammad Raihan Ariffianto (18222092)

# Daftar Isi

<b>Daftar Isi.....</b>	<b>2</b>
<b>BAB I.....</b>	<b>3</b>
<b>Deskripsi Persoalan.....</b>	<b>3</b>
1.1 Latar Belakang.....	3
1.2 Rumusan Masalah.....	3
1.3 Tujuan.....	4
1.4 Batasan Masalah.....	4
<b>BAB II.....</b>	<b>6</b>
<b>Pembahasan.....</b>	<b>6</b>
2.1 Pemilihan Objective Function.....	6
2.1.1 Perumusan Objective Function.....	6
2.1.2 Alasan Pemilihan Objective Function.....	8
2.2 Penjelasan Implementasi Algoritma Local Search.....	9
2.2.1 Steepest Ascent Hill-Climbing.....	9
2.2.2 Hill-Climbing with Sideways Move.....	12
2.2.3 Random Restart Hill-Climbing.....	21
2.2.4 Stochastic Hill-Climbing.....	24
2.2.5 Simulated Annealing.....	26
2.2.6 Genetic Algorithm.....	30
2.3 Hasil Eksperimen.....	35
2.3.1 Steepest Ascent Hill-Climbing.....	35
2.3.2 Hill-Climbing with Sideways Move.....	36
2.3.3 Random Restart Hill-Climbing.....	37
2.3.4 Stochastic Hill-Climbing.....	38
2.3.5 Simulated Annealing.....	38
2.3.6 Genetic Algorithm.....	39
2.4 Hasil Analisis.....	41
<b>BAB III.....</b>	<b>43</b>
<b>Kesimpulan dan Saran.....</b>	<b>43</b>
<b>BAB IV.....</b>	<b>44</b>
<b>Pembagian Tugas.....</b>	<b>44</b>
<b>BAB V.....</b>	<b>45</b>
<b>Referensi.....</b>	<b>45</b>

# **BAB I**

## **Deskripsi Persoalan**

### **1.1 Latar Belakang**

Diagonal Magic Cube merupakan sebuah permasalahan yang menarik perhatian pada bidang kombinatorial dan matematika yang tersusun dari angka-angka 1 sampai  $n^3$  tanpa pengulangan dalam kubus berukuran  $n \times n \times n$  dengan jumlah setiap baris, kolom, tiang, diagonal bidang, dan diagonal ruangnya harus menghasilkan nilai yang sama yang disebut magic number. Nilai magic number ini berdasar pada ukuran kubus tersebut. Persoalan ini memiliki kompleksitas ruang pencarian yang sangat besar seiring dengan bertambahnya ukuran kubus.

Implementasi solusi untuk permasalahan Diagonal Magic Cube harus menggunakan penggunaan pencarian solusi yang efisien dengan kompleksitas ruang pencarian yang mungkin sangat besar. Salah satu pendekatan yang relevan adalah algoritma local search yang dapat digunakan untuk menyelesaikan permasalahan ini dengan mencari solusi yang lebih baik secara bertahap. Algoritma local search ini tidak mencari jalur menuju solusi, melainkan langsung mencari solusi dalam ruang keadaan yang lengkap. Variasi metode-metode local search yang umum digunakan yaitu Steepest Ascent Hill-Climbing, Hill-Climbing with Sideways Move, Random Restart Hill-Climbing, Stochastic Hill-Climbing, Simulated Annealing, dan Genetic Algorithm dengan kelebihan dan kekurangannya masing-masing.

### **1.2 Rumusan Masalah**

Penyelesaian permasalahan Diagonal Magic Cube  $5 \times 5 \times 5$  dapat dilakukan dengan menggunakan berbagai variasi algoritma local search. Permasalahan yang akan diselesaikan dapat dirumuskan sebagai berikut:

- 1.2.1 Seberapa dekat tiap-tiap algoritma bisa mendekati global optima dan mengapa hasilnya demikian?
- 1.2.2 Bagaimana perbandingan hasil pencarian tiap-tiap algoritma dengan algoritma local search yang lain?

- 1.2.3 Bagaimana perbandingan durasi proses pencarian tiap algoritma relatif terhadap algoritma lainnya?
- 1.2.4 Seberapa konsisten hasil akhir yang didapatkan dari tiap-tiap eksperimen yang dilakukan?
- 1.2.5 Bagaimana pengaruh banyak iterasi dan jumlah populasi terhadap hasil akhir pencarian pada Genetic Algorithm?

### **1.3 Tujuan**

Pencarian solusi ini memberikan wawasan mengenai implementasi algoritma local search untuk mencari solusi masalah Diagonal Magic Cube. Secara spesifik, tujuan yang ingin dicapai pada pencarian ini adalah sebagai berikut:

- 1.3.1 Menentukan seberapa dekat tiap algoritma local search dapat mendekati solusi optimal global dan memahami faktor-faktor yang mempengaruhi hasil tersebut.
- 1.3.2 Membandingkan hasil akhir pencarian dari setiap algoritma local search untuk mengidentifikasi keunggulan atau kelemahan spesifik dari masing-masing metode.
- 1.3.3 Menilai durasi proses pencarian tiap algoritma relatif terhadap algoritma lainnya untuk memahami efisiensi waktu masing-masing dalam mencapai hasil optimal.
- 1.3.4 Menguji konsistensi hasil akhir yang diperoleh dari setiap eksperimen untuk setiap algoritma guna mengetahui reliabilitas algoritma dalam kondisi pencarian yang berbeda.
- 1.3.5 Menganalisis pengaruh jumlah iterasi dan jumlah populasi terhadap hasil akhir pencarian pada Genetic Algorithm untuk mengoptimalkan parameter dan memastikan pencapaian hasil yang baik dalam waktu yang efisien.

### **1.4 Batasan Masalah**

Untuk fokus menyelesaikan masalah, batasan masalah pada persoalan Diagonal Magic Cube adalah sebagai berikut:

- 1.4.1 Program yang dibuat harus bisa memvisualisasikan state awal kubus, state akhir kubus, dan juga hasil eksperimennya.

- 1.4.2 Cara visualisasi dibebaskan selama seluruh angka pada kubus dan juga hasil eksperimen terlihat dengan jelas.
- 1.4.3 Dibebaskan menggunakan bahasa pemrograman apapun.
- 1.4.4 Diperbolehkan untuk menggunakan heuristik buatan sendiri atau dari referensi lain untuk optimasi pencarian solusi, asalkan masih dalam lingkup local search.

# BAB II

## Pembahasan

### 2.1 Pemilihan Objective Function

Pemilihan objective function yang efektif sangat penting untuk mengarahkan algoritma pencarian solusi menuju solusi penyelesaian persoalan Diagonal Magic Cube berukuran  $5 \times 5 \times 5$  yang optimal. Pada dasarnya, objective function ini harus mencerminkan seberapa dekat solusi saat ini dengan kondisi ideal, di mana setiap baris, kolom, tiang, diagonal bidang dan diagonal ruang memiliki jumlah yang sama dengan magic number yang dihitung berdasarkan ukuran kubus.

Magic number  $M$  untuk kubus berukuran  $5 \times 5 \times 5$  dapat dihitung menggunakan rumus:

$$M = \frac{1}{2} (n \cdot (n^3 + 1))$$

Untuk  $n = 5$ , nilai magic number yang didapatkan adalah:

$$M = \frac{1}{2} (5 \cdot (5^3 + 1)) = 315$$

Dengan demikian, setiap baris, kolom, tiang, diagonal bidang, dan diagonal ruang pada kubus harus mempunyai jumlah elemen yang sama dengan 315 agar kubus tersebut memenuhi syarat sebagai Diagonal Magic Cube.

#### 2.1.1 Perumusan Objective Function

Objective function dalam penyelesaian masalah Diagonal Magic Cube berukuran  $5 \times 5 \times 5$  ini dirancang untuk mengukur selisih antara total angka pada setiap baris, kolom, tiang, diagonal bidang, dan diagonal ruang pada magic number  $M$ . Adapun rumus untuk objective function yang digunakan adalah sebagai berikut:

##### 2.1.1.1 Perhitungan Deviasi dari Magic Number

Hitung selisih (deviasi) antara total angka pada baris, kolom, tiang, diagonal bidang, dan diagonal ruang pada kubus dengan magic number  $M$ . Semakin besar selisihnya, semakin buruk solusinya. Oleh karena itu, perhitungan total deviasi dari magic number dapat dirumuskan sebagai:

$$f(state) = \sum_{i=1}^{25} |S_{baris}(i) - M| + \sum_{i=1}^{25} |S_{kolom}(i) - M| + \sum_{i=1}^{25} |S_{tiang}(i) - M| \\ + \sum_{i=1}^{25} |S_{diagonal\ bidang}(i) - M| + \sum_{i=1}^{25} |S_{diagonal\ ruang}(i) - M|$$

Dengan:

- $S_{baris}(i)$ ,  $S_{kolom}(i)$ ,  $S_{tiang}(i)$ ,  $S_{diagonal\ bidang}(i)$ , dan  $S_{diagonal\ ruang}(i)$  adalah jumlah elemen pada baris, kolom, tiang, diagonal bidang, dan diagonal ruang ke-i.
- $M$  adalah nilai magic number (315 untuk kubus 5x5x5).

#### 2.1.1.2 Penambahan Skor Berdasarkan Kriteria Tercapai

Jika sebuah baris, kolom, tiang, diagonal bidang atau diagonal ruang memenuhi magic number, maka diberikan penambahan skor. Total maksimal skor yang bisa dicapai jika seluruh kondisi terpenuhi adalah sebagai berikut:

##### 2.1.1.2.1 Baris

25 baris, dengan masing-masing memberikan skor +1 jika jumlah angkanya tepat 315.

##### 2.1.1.2.2 Kolom

25 kolom, dengan masing-masing memberikan skor +1 jika jumlahnya sama dengan 315.

##### 2.1.1.2.3 Tiang

25 tiang, dengan masing-masing memberikan skor +1 jika jumlahnya sama dengan 315.

##### 2.1.1.2.4 Diagonal bidang

30 diagonal bidang, dengan masing-masing memberikan skor +1 jika jumlahnya sama dengan 315.

##### 2.1.1.2.5 Diagonal ruang

4 diagonal ruang, dengan masing-masing memberikan skor +1 jika jumlahnya sama dengan 315.

Dengan demikian, total penambahan skor maksimum adalah:

$$\text{Skor}_{\text{maksimum}} = 25 \text{ (baris)} + 25 \text{ (kolom)} + 25 \text{ (tiang)} + 30 \text{ (diagonal bidang)} + 4 \text{ (diagonal ruang)} = 109$$

#### 2.1.1.3 Heuristic Cost untuk Evaluasi State

Setiap *state* akan dimulai dengan *initial state cost* sebesar -109, dan *state* ini akan mendekati global optimum saat nilai objective function mendekati 0. Setiap kali ada kesesuaian antara jumlah elemen pada baris, kolom, tiang, diagonal bidang, atau diagonal ruang dengan magic number, skor akan ditambahkan, dan *state* tersebut akan menjadi lebih baik.

#### 2.1.1.4 Penalti untuk Pengulangan Angka

Penalti pun akan diberikan jika terjadi pengulangan angka di dalam kubus. Penalti ini penting untuk memastikan semua angka dari 1 hingga 125 muncul tepat satu kali. Setiap pengulangan atau angka yang hilang akan mengurangi skor atau menambah penalti dalam objective function.

### 2.1.2 Alasan Pemilihan Objective Function

Dalam perumusan objective function ini, beberapa pertimbangan penting menjadi dasar pemilihan:

#### 2.1.2.1 Mengukur Kedekatan dengan Solusi Optimal

Fungsi ini mengukur secara langsung jarak antara solusi saat ini dengan kondisi ideal. Setiap deviasi dihitung dan diakumulasikan untuk memberikan indikasi seberapa optimal solusi tersebut.

#### 2.1.2.2 Mendorong Kesesuaian Diagonal Bidang dan Diagonal Ruang

Karena diagonal bidang dan diagonal ruang merupakan aspek penting dari Diagonal Magic Cube, memberikan skor tambahan pada setiap diagonal yang sesuai dengan magic number membantu algoritma pencarian untuk memprioritaskan bagian-bagian ini dalam pencarian suatu solusi.

#### 2.1.2.3 Penalti untuk Pengulangan

Penalti untuk pengulangan angka akan mencegah solusi yang valid secara numerik namun tidak valid secara struktural.



Dengan objective function ini, algoritma pencarian dapat diarahkan secara efektif untuk mendekati solusi optimal melalui proses iteratif.

## **2.2 Penjelasan Implementasi Algoritma Local Search**

Algoritma local search adalah metode untuk menemukan solusi dalam ruang pencarian yang besar. Algoritma ini tidak menyimpan jejak lintasan apa pun yang terambil dan hanya berfokus pada suatu state. Tujuannya adalah untuk menemukan solusi optimal atau mendekati optimal dengan memperbaiki solusi iteratif yang sudah ada.

Local search sangat efektif untuk masalah optimasi, terutama bila ruang pencarian sangat besar sehingga tidak praktis untuk melakukan complete search. Algoritma ini didasarkan pada heuristic function atau objective function untuk mengevaluasi kualitas solusi. Jika solusi yang ditemukan tidak optimal, algoritma akan terus bergerak menuju solusi yang lebih baik melalui perbaikan bertahap atau perubahan acak. Variasi dari algoritma local search adalah sebagai berikut:

### **2.2.1 Steepest Ascent Hill-Climbing**

Steepest ascent hill-climbing merupakan varian dari algoritma local search yang digunakan untuk menyelesaikan masalah yang secara berulang-ulang bergerak ke neighbouring state terbaik. Dalam Steepest Ascent, algoritma ini bergerak ke neighbour yang memiliki peningkatan yang paling bagus. Proses ini berlanjut sampai tidak ada lagi peningkatan yang terjadi (menyampai local maxima).

Kode di bawah ini mengimplementasikan algoritma steepest ascent hill-climbing untuk mengoptimalkan penyelesaian Diagonal Magic Cube. Diawali dengan mengevaluasi susunan awal dengan menghitung initial cost. Lalu, mencari semua kemungkinan konfigurasi pada kubus untuk mencari kemungkinan penukaran terbaik pada elemen yang mungkin merendahkan cost. Setiap kemungkinan penukaran dicek dan jika ditemukan konfigurasi baru dengan cost yang lebih rendah, konfigurasinya diganti. Proses diulang sampai mencapai solusi sempurna ( $\text{cost} = 0$ ) atau sampai mencapai jumlah iterasi maksimum. Jika tidak ada konfigurasi yang lebih baik, algoritma berhenti, dan berarti telah mencapai local minima. Hasil akhir yang dikeluarkan dari kode ini yaitu cost terendah yang ditemukan, konfigurasi terbaik yang ditemukan, jumlah iterasi, dan langkah-langkah yang diambil.

```

def steepest_ascent_hill_climbing(cube):
    max_iterations = 1000 # Define the maximum number of
iterations allowed
    iteration = 0
    current_cost = cube.calculate_cost() # Initial cost
    obj_values = [] # Collect the objective values for each
iteration
    steps = [] # Collect step data

    best_cube = cube.cube.copy() # Keep a copy of the
initial cube

    # Iterate until there are no conflicts (cost == 0) or max
iterations are reached
    while current_cost > 0 and iteration < max_iterations:
        obj_values.append(current_cost) # Record the current
cost

        print(f"Iteration {iteration}: {current_cost} cost")

        best_cost = current_cost # Start with the current
cost
        best_swap = None # Track the best swap

        # Explore all possible pairs of positions (i.e.,
every possible neighbour)
        for pos1, pos2 in
itertools.combinations(np.ndindex(cube.cube.shape), 2):
            new_cube = cube.cube.copy()

            # Swap the two elements
            new_cube[pos1], new_cube[pos2] = new_cube[pos2],
new_cube[pos1]

            # Set the cube to the new state and calculate the
cost

            cube.cube = new_cube
            new_cost = cube.calculate_cost()

            # If the new configuration has a lower cost,

```

```

update the best cube
    if new_cost < best_cost:
        best_cost = new_cost
        best_cube = new_cube.copy()
        best_swap = (pos1, pos2) # Track the swap

    # If no better configuration was found, stop the
algorithm (local minimum)
    if best_cost == current_cost:
        print("No better neighbours found, stopping.")
        return best_cost, best_cube, iteration, steps

    # Update the cube with the best found neighbour
configuration
    cube.cube = best_cube
    current_cost = best_cost

    # Print details of the best swap and record the step
    if best_swap:
        pos1, pos2 = best_swap
        step_info = {
            "index1": pos1[0] * cube.size**2 + pos1[1] *
cube.size + pos1[2],
            "index2": pos2[0] * cube.size**2 + pos2[1] *
cube.size + pos2[2],
            "cost": current_cost
        }
        steps.append(step_info)
        print(f"Step {iteration + 1}: {step_info}")
        print(f"Swapped positions {pos1} and {pos2},
resulting in new cost: {current_cost}")
        print(f"Elements swapped: {cube.cube[pos1]},
{cube.cube[pos2]}")
    else:
        steps.append({"index1": 0, "index2": 0, "cost":
current_cost})

    iteration += 1
    return current_cost, best_cube, iteration, steps

```

### 2.2.2 Hill-Climbing with Sideways Move

Hill-Climbing with Sideways Move merupakan varian dari algoritma local search yang digunakan untuk menyelesaikan masalah yang secara berulang-ulang bergerak ke neighbouring state terbaik atau sama dengan state saat ini. Dalam Sideways Move, algoritma ini bergerak ke neighbour yang memiliki peningkatan yang lebih bagus atau sama dengan saat ini. Fleksibilitas ini dapat membantu algoritma bergerak di daerah datar (plateau) yang tidak terdapat pengurangan cost yang nampak. Proses ini berlanjut sampai tidak ada lagi peningkatan yang terjadi (menyampai local maxima).

Kode di bawah ini mengimplementasikan algoritma hill-climbing with sideways move untuk penyelesaian Diagonal Magic Cube yang ditingkatkan untuk menangani kekompleksan. Diawali dengan menghitung initial cost. Lalu, secara berulang berusaha meningkatkan konfigurasi dengan mengecek kemungkinan penukaran. Daftar tabu digunakan agar tidak mengulangi langkah-langkah sebelumnya. Algoritma ini fokus pada area dalam kubus yang bernilai deviasi tinggi (jauh dari nilai ideal) dan memilih penukaran elemen secara efektif untuk memperbaiki area tersebut. Algoritma ini juga melacak efektivitas berbagai jenis penukaran dan secara berkala menyesuaikan strateginya berdasarkan pengamatan ini. Selain itu, parameter mirip Temperatur digunakan untuk menerima langkah yang tidak memperbaiki keadaan dengan probabilitas tertentu, meniru perilaku simulated annealing untuk memungkinkan eksplorasi keadaan yang kurang optimal dalam rangka pencarian yang lebih luas. Proses ini berlangsung hingga solusi mencapai keadaan biaya nol, jumlah gerakan menyamping yang diperbolehkan habis, atau jumlah iterasi maksimum tercapai. Kode ini mencatat perkembangan biaya dan langkah-langkah penting untuk analisis dan visualisasi.

```
def hill_climbing_with_sideways_move(cube,
max_sideways_moves, max_iterations, tabu_list_size=50):
    """
        Enhanced version of hill climbing with sideways moves
        that uses:
        - Tabu list to prevent cycling
        - Adaptive neighbourhood selection
        - Line sum analysis for informed swaps
        - Temperature-like parameter for dynamic acceptance

        Returns:
        - final_cost: The final cost after the algorithm
    """
```

```

terminates.
    - final_cube: The state of the cube when the algorithm
ends.
    - iteration: The number of iterations performed.
    - cost_progress: List of objective values over
iterations.
    - sideways_moves: Total number of sideways moves made.
    """
    iteration = 0
    sideways_moves = 0
    current_cost = cube.calculate_actual_cost()
    cost_progress = []
    steps = [] # Collect step data

    # Initialize tabu list to prevent revisiting recent
states
    tabu_list = []

    # Track the effectiveness of different types of swaps
    swap_effectiveness = defaultdict(lambda: {'attempts': 0,
'improvements': 0})

    # Temperature-like parameter for accepting worse moves
occasionally
    initial_temperature = 10.0

    while current_cost > 0 and iteration < max_iterations:
        print(f"Iteration {iteration}: {current_cost} cost,
Sideways moves: {sideways_moves}")

        # Calculate current line sums
        line_sums = calculate_line_sums(cube)

        # Find problematic lines (those far from magic
number)
        problem_areas = identify_problem_areas(line_sums)

        # Generate candidate swaps with preference for
problematic areas

```

```

        candidate_swaps =
generate_intelligent_swaps(problem_areas, cube.size,
tabu_list)

        # Track if we found any improvement
        found_improvement = False
        temperature = initial_temperature * (1 - iteration /
max_iterations)

        for swap_type, pos1, pos2 in candidate_swaps:
            if (pos1, pos2) in tabu_list:
                continue

            new_cube = cube.cube.copy()
            new_cube[pos1], new_cube[pos2] = new_cube[pos2],
new_cube[pos1]

            cube.cube = new_cube
            new_cost = cube.calculate_actual_cost()

            # Update swap effectiveness statistics
            swap_effectiveness[swap_type]['attempts'] += 1

            # Accept if better or with probability based on
temperature
            if new_cost < current_cost:
                best_cube = new_cube.copy()
                current_cost = new_cost
                found_improvement = True
                swap_effectiveness[swap_type]['improvements']
+= 1

                update_tabu_list(tabu_list, (pos1, pos2),
tabu_list_size)

            # Add step tracking here
            step_info = {
                "index1": pos1[0] * cube.size**2 +
pos1[1] * cube.size + pos1[2],
                "index2": pos2[0] * cube.size**2 +

```

```

pos2[1] * cube.size + pos2[2],
        "cost": current_cost
    }
    steps.append(step_info)
    print(f"Step {iteration + 1}: {step_info}")
    break

    elif new_cost == current_cost and sideways_moves
< max_sideways_moves:
        acceptance_prob = np.exp(-0.1 / temperature)
        if random.random() < acceptance_prob:
            best_cube = new_cube.copy()
            current_cost = new_cost
            sideways_moves += 1
            update_tabu_list(tabu_list, (pos1, pos2),
tabu_list_size)

            # Add step tracking here
            step_info = {
                "index1": pos1[0] * cube.size**2 +
pos1[1] * cube.size + pos1[2],
                "index2": pos2[0] * cube.size**2 +
pos2[1] * cube.size + pos2[2],
                "cost": current_cost
            }
            steps.append(step_info)
            print(f"Step {iteration + 1}:
{step_info}")

            break

            # Restore original cube for next attempt
            cube.cube = new_cube.copy()

        if not found_improvement and sideways_moves >=
max_sideways_moves:
            print("No improvement found and sideways moves
exhausted.")
            break

```

```

        cost_progress.append(current_cost)
        iteration += 1

        # Periodically adjust strategy based on effectiveness
        if iteration % 50 == 0:
            adjust_strategy(swap_effectiveness)

        # Return structured data matching the expected output
        return current_cost, cube.cube, iteration, steps

def calculate_line_sums(cube):
    """Calculate all line sums in the cube and their
    deviations from magic number."""
    line_sums = {
        'rows': [],
        'columns': [],
        'pillars': [],
        'diagonals': []
    }

    # Calculate row sums
    for i in range(cube.size):
        for j in range(cube.size):
            line_sums['rows'].append({
                'sum': cube.cube[i, j, :].sum(),
                'position': (i, j, ':'),
                'deviation': abs(cube.cube[i, j, :].sum() -
cube.magic_number)
            })

            # Calculate column sums
            line_sums['columns'].append({
                'sum': cube.cube[i, :, j].sum(),
                'position': (i, ':', j),
                'deviation': abs(cube.cube[i, :, j].sum() -
cube.magic_number)
            })

            # Calculate pillar sums

```



[illegible]

```

reverse=True)

    problems.extend([x['position'] for x in
sorted_lines[:top_n]])

    return problems

def generate_intelligent_swaps(problem_areas, cube_size,
tabu_list):
    """Generate candidate swaps with focus on problem
areas."""
    candidates = []

    # Generate different types of swaps
    # 1. Within-line swaps for problematic lines
    for area in problem_areas:
        line_positions = get_line_positions(area, cube_size)
        for pos1, pos2 in
itertools.combinations(line_positions, 2):
            if (pos1, pos2) not in tabu_list:
                candidates.append(('within_line', pos1,
pos2))

    # 2. Cross-line swaps between problematic areas
    for area1, area2 in itertools.combinations(problem_areas,
2):
        pos1 = get_representative_position(area1, cube_size)
        pos2 = get_representative_position(area2, cube_size)
        if (pos1, pos2) not in tabu_list:
            candidates.append(('cross_line', pos1, pos2))

    # 3. Random swaps (for diversity)
    for _ in range(max(1, len(candidates) // 4)):
        pos1 = tuple(random.randrange(cube_size) for _ in
range(3))
        pos2 = tuple(random.randrange(cube_size) for _ in
range(3))
        if (pos1, pos2) not in tabu_list:
            candidates.append(('random', pos1, pos2))

```

```

        # Shuffle candidates to avoid getting stuck in patterns
        random.shuffle(candidates)
        return candidates

def get_line_positions(area, cube_size):
    """Get all positions in a given line."""
    positions = []
    if isinstance(area[0], str) and area[0] == ':': # Pillar
        i, j = area[1], area[2]
        positions.extend((x, i, j) for x in range(cube_size))
    elif isinstance(area[1], str) and area[1] == ':': #
Column
        i, j = area[0], area[2]
        positions.extend((i, x, j) for x in range(cube_size))
    elif isinstance(area[2], str) and area[2] == ':': # Row
        i, j = area[0], area[1]
        positions.extend((i, j, x) for x in range(cube_size))
    elif area[1] == 'diag': # Diagonal
        i = area[0]
        if area[2] == 'main':
            positions.extend((i, x, x) for x in
range(cube_size))
        else: # anti-diagonal
            positions.extend((i, x, cube_size-1-x) for x in
range(cube_size))
    return positions

def get_representative_position(area, cube_size):
    """Get a representative position from an area
specification."""
    if isinstance(area[0], str) and area[0] == ':': # Pillar
        return (random.randrange(cube_size), area[1],
area[2])
    elif isinstance(area[1], str) and area[1] == ':': #
Column
        return (area[0], random.randrange(cube_size),
area[2])
    elif isinstance(area[2], str) and area[2] == ':': # Row
        return (area[0], area[1],

```

```

random.randrange(cube_size))
    elif area[1] == 'diag': # Diagonal
        x = random.randrange(cube_size)
        if area[2] == 'main':
            return (area[0], x, x)
        else: # anti-diagonal
            return (area[0], x, cube_size-1-x)
    return area # Default case

def update_tabu_list(tabu_list, swap, max_size):
    """Update the tabu list with the new swap."""
    tabu_list.append(swap)
    if len(tabu_list) > max_size:
        tabu_list.pop(0)

def adjust_strategy(swap_effectiveness):
    """Adjust strategy based on the effectiveness of
    different swap types."""
    for swap_type, stats in swap_effectiveness.items():
        if stats['attempts'] > 0:
            effectiveness = stats['improvements'] /
stats['attempts']
            print(f"Swap type {swap_type} effectiveness:
{effectiveness:.2%}")

def plot_results(cost_progress):
    """Plot the cost progression with additional
    statistics."""
    plt.figure(figsize=(12, 6))
    plt.plot(cost_progress, label='Cost')
    plt.title("Cost Progression during Improved
Hill-Climbing")
    plt.xlabel("Iteration")
    plt.ylabel("Total Cost")
    plt.grid(True)

    # Add moving average
    window = min(50, len(cost_progress))
    if window > 0:

```

```
moving_avg = np.convolve(cost_progress,
                           np.ones(window)/window,
                           mode='valid')
plt.plot(range(window-1, len(cost_progress)),
         moving_avg,
         'r--',
         label='Moving Average')

plt.legend()
plt.show()
```

### 2.2.3 Random Restart Hill-Climbing

Random Restart Hill-Climbing adalah varian dari algoritma hill-climbing yang mencoba mengatasi kelemahan utama dari hill-climbing biasa, yaitu kecenderungan untuk terjebak pada local minima atau maxima. Dalam Random Restart Hill-Climbing, jika solusi berhenti pada titik yang tidak dapat diperbaiki lagi (local minima), algoritma akan melakukan "restart" dengan mengacak kembali susunan awal solusi, lalu memulai proses pencarian kembali. Algoritma ini mengulangi proses ini beberapa kali sampai batas maksimal restart atau sampai menemukan solusi optimal. Pendekatan ini meningkatkan peluang menemukan solusi global dibandingkan dengan hill-climbing biasa yang hanya melakukan satu kali pencarian.

Kode di bawah ini mengimplementasikan algoritma Random Restart Hill-Climbing untuk mengoptimalkan solusi Diagonal Magic Cube. Pertama-tama, algoritma menentukan solusi terbaik yang ditemukan sejauh ini (best overall) dengan nilai biaya (cost) tak terbatas. Untuk setiap restart, jika bukan restart pertama, kubus diacak ulang agar mulai dengan konfigurasi berbeda. Selanjutnya, algoritma melakukan hill-climbing pada konfigurasi ini dengan cara mengevaluasi cost awal, kemudian mencoba semua kombinasi elemen dalam kubus untuk menemukan penukaran elemen terbaik yang menurunkan cost. Jika penurunan cost ditemukan, konfigurasi kubus diperbarui ke keadaan terbaik ini.

Algoritma ini terus melakukan iterasi hingga mencapai cost nol, jumlah iterasi maksimum, atau jika tidak ada perbaikan lebih lanjut pada cost. Jika hasil dari restart ini lebih baik daripada hasil terbaik sebelumnya, maka hasil ini akan disimpan sebagai best overall, termasuk detail langkah-langkah yang diambil. Proses ini akan berhenti jika mencapai solusi sempurna (cost = 0) atau setelah jumlah restart maksimum. Hasil akhir dari algoritma ini mencakup cost terendah

yang ditemukan, konfigurasi terbaik yang ditemukan, jumlah langkah pada restart terbaik, daftar langkah-langkah yang dilakukan, dan jumlah iterasi untuk setiap restart.

```
def random_restart_hill_climbing(cube, max_restarts=10,
max_iterations_per_restart=1000):
    best_overall_cost = float('inf')
    best_overall_cube = cube.cube.copy()
    iterations_per_restart = [] # New array to track
iterations for each restart
    best_steps = [] # Modified to store steps with index1,
index2, cost format

    for restart in range(max_restarts):
        print(f"Restart {restart+1}/{max_restarts}")

        if restart > 0:
            numbers = list(range(1, cube.size**3 + 1))
            np.random.shuffle(numbers)
            cube.cube = np.array(numbers).reshape((cube.size,
cube.size, cube.size))

            current_cost = cube.calculate_actual_cost()
            steps = [] # Track steps for current restart
            iteration = 0

            while current_cost > 0 and iteration <
max_iterations_per_restart:
                print(f"Iteration {iteration}: {current_cost}
cost")

                best_cube = cube.cube.copy()
                best_cost = current_cost
                best_swap = None

                for pos1, pos2 in
itertools.combinations(np.ndindex(cube.cube.shape), 2):
                    new_cube = cube.cube.copy()
                    new_cube[pos1], new_cube[pos2] =
```

```

new_cube[pos2], new_cube[pos1]
    cube.cube = new_cube
    new_cost = cube.calculate_actual_cost()

    if new_cost < best_cost:
        best_cost = new_cost
        best_cube = new_cube.copy()
        best_swap = (pos1, pos2)

    if best_cost == current_cost:
        print("No better neighbors found, stopping
hill-climbing.")
        steps.append({
            'index1': 0,
            'index2': 0,
            'cost': current_cost
        })
        break

    if best_swap:
        pos1, pos2 = best_swap
        steps.append({
            'index1': np.ravel_multi_index(pos1,
cube.cube.shape),
            'index2': np.ravel_multi_index(pos2,
cube.cube.shape),
            'cost': best_cost
        })

        cube.cube = best_cube
        current_cost = best_cost
        iteration += 1

    iterations_per_restart.append(iteration) # Store
iterations for this restart

    if current_cost < best_overall_cost:
        best_overall_cost = current_cost
        best_overall_cube = cube.cube.copy()

```

```

        best_steps = steps # Store steps from best
restart

    if current_cost == 0:
        break

    return best_overall_cost, best_overall_cube,
len(best_steps), best_steps, iterations_per_restart

```

### 2.2.4 Stochastic Hill-Climbing

Stochastic hill-climbing adalah varian dari algoritma local search yang mencari solusi optimal dengan melakukan perubahan secara acak pada keadaan saat ini. Berbeda dengan Steepest Ascent yang mencoba setiap kemungkinan langkah untuk menemukan peningkatan terbaik, Stochastic Hill-Climbing memilih langkah secara acak. Jika langkah acak ini menghasilkan peningkatan, algoritma akan berpindah ke keadaan baru; jika tidak, langkah tersebut diabaikan dan algoritma kembali ke keadaan awal. Proses ini membantu algoritma menjelajahi ruang solusi secara luas, dengan harapan menemukan solusi optimal atau mendekati optimal.

Kode di bawah mengimplementasikan algoritma stochastic hill-climbing untuk optimasi Diagonal Magic Cube. Pertama-tama, algoritma menghitung cost awal untuk konfigurasi kubus saat ini. Kemudian, pada setiap iterasi, algoritma secara acak memilih dua posisi di kubus untuk ditukar. Setelah penukaran dilakukan, algoritma menghitung cost baru dari konfigurasi hasil pertukaran tersebut. Jika cost menurun, maka konfigurasi kubus disimpan sebagai keadaan baru, dan langkah tersebut dicatat dalam daftar langkah. Jika cost tidak membaik, kubus dikembalikan ke keadaan semula dan iterasi berlanjut dengan pertukaran acak lainnya.

Proses ini diulang hingga tercapai solusi sempurna (cost = 0) atau jumlah iterasi maksimum. Hasil akhir dari algoritma ini adalah cost terendah yang ditemukan, konfigurasi kubus terbaik yang ditemukan, jumlah iterasi yang dilakukan, serta langkah-langkah yang diambil untuk mencapai konfigurasi terbaik.

```

def stochastic_hill_climbing(cube, max_iterations=1000):
    iteration = 0
    current_cost = cube.calculate_actual_cost()

```



```

steps = [] # Track steps with index swaps and cost

while current_cost > 0 and iteration < max_iterations:
    print(f"Iteration {iteration}: {current_cost} cost")

    new_cube = cube.cube.copy()

    pos1 = tuple(random.randint(0, cube.size - 1) for _
in range(3))
    pos2 = tuple(random.randint(0, cube.size - 1) for _
in range(3))

    while pos1 == pos2:
        pos2 = tuple(random.randint(0, cube.size - 1) for
_ in range(3))

    new_cube[pos1], new_cube[pos2] = new_cube[pos2],
new_cube[pos1]

    cube.cube = new_cube
    new_cost = cube.calculate_actual_cost()

    if new_cost < current_cost:
        print(f"Accepted new configuration by swapping
{pos1} and {pos2}, new cost: {new_cost}")
        steps.append({'index1': pos1[0] * cube.size**2 +
pos1[1] * cube.size + pos1[2],
                        'index2': pos2[0] * cube.size**2 +
pos2[1] * cube.size + pos2[2],
                        'cost': new_cost})
        current_cost = new_cost
    else:
        cube.cube[pos1], cube.cube[pos2] =
cube.cube[pos2], cube.cube[pos1]
        steps.append({'index1': 0, 'index2': 0, 'cost':
current_cost})

    iteration += 1

```

```
return current_cost, cube.cube, iteration, steps
```

### 2.2.5 Simulated Annealing

Simulated annealing adalah algoritma optimasi yang terinspirasi oleh proses pendinginan logam, di mana logam dipanaskan hingga suhu tinggi lalu didinginkan secara perlahan untuk mencapai struktur yang lebih stabil. Dalam algoritma ini, pencarian solusi dilakukan dengan mengizinkan langkah-langkah yang mungkin tidak selalu menghasilkan perbaikan langsung, tetapi dapat membantu algoritma keluar dari local minima. Ini dilakukan dengan probabilitas penerimaan langkah buruk yang bergantung pada “temperatur” saat itu. Semakin tinggi temperatur, semakin besar kemungkinan menerima solusi yang lebih buruk, dan seiring waktu, temperatur menurun sehingga algoritma semakin selektif.

Kode di bawah mengimplementasikan simulated annealing untuk optimasi Diagonal Magic Cube. Algoritma dimulai dengan menghitung cost awal dari konfigurasi kubus dan menetapkan suhu awal. Pada setiap iterasi, algoritma secara acak memilih tetangga, yaitu dua posisi pada kubus untuk ditukar, atau terkadang melakukan penukaran beberapa elemen sekaligus (multi-swap) berdasarkan probabilitas tertentu. Jika perubahan ini menghasilkan cost yang lebih rendah, algoritma langsung menerima konfigurasi baru tersebut. Namun, jika cost meningkat, algoritma masih dapat menerima perubahan ini dengan probabilitas yang menurun sesuai dengan suhu.

Algoritma ini juga menerapkan jadwal penurunan suhu yang disesuaikan berdasarkan jumlah iterasi dan pola perbaikan yang terjadi, sehingga semakin mendekati solusi, suhu akan semakin rendah. Proses ini berlanjut hingga suhu mencapai nilai minimum yang telah ditentukan, atau cost mencapai nol (solusi sempurna), atau jumlah iterasi maksimum tercapai. Sepanjang iterasi, data seperti suhu, langkah yang diambil, dan kondisi stuck di local optima juga dicatat.

Hasil akhir dari algoritma ini adalah cost terbaik yang ditemukan, konfigurasi terbaik dari kubus, jumlah iterasi yang dilakukan, perubahan suhu sepanjang iterasi, langkah-langkah yang diambil, serta indikator jika algoritma sering terjebak di local minima.

```
def simulated_annealing(cube,  
                        initial_temperature=1000,  
                        min_temperature=0.01,  
                        max_iterations=5000,
```

```

        stage_iterations=1000,
        multi_swap_probability=0.3):

    current_temperature = initial_temperature
    current_cost = cube.calculate_actual_cost()
    best_cost = current_cost
    best_configuration = cube.cube.copy()

    steps = []
    temperatures = []
    consecutive_non_improvements = 0
    stuck_in_local_optima = 0 # Counter for tracking stuck
points

    def get_problem_specific_neighbor():
        pos1 = tuple(random.randint(0, cube.size - 1) for _
in range(3))
        pos2 = tuple(random.randint(0, cube.size - 1) for _
in range(3))
        while pos1 == pos2:
            pos2 = tuple(random.randint(0, cube.size - 1) for
_ in range(3))
        return pos1, pos2

    def perform_multi_swap():
        new_cube = cube.cube.copy()
        num_swaps = random.randint(2, 4)
        positions = [tuple(random.randint(0, cube.size - 1)
for _ in range(3)) for _ in range(num_swaps)]
        for i in range(len(positions) - 1):
            new_cube[positions[i]], new_cube[positions[i +
1]] = new_cube[positions[i + 1]], new_cube[positions[i]]
        return new_cube

    def calculate_acceptance_probability(cost_difference,
temperature):
        if cost_difference <= 0:
            return 1.0
        normalized_delta = cost_difference /

```

```

max(abs(current_cost), 1)
    normalized_delta = min(normalized_delta, 5.0)
    return math.exp(-normalized_delta / (temperature /
initial_temperature))

def adaptive_temperature_schedule(iteration):
    if consecutive_non_improvements > 5000:
        return initial_temperature * 0.5
    stage = iteration // stage_iterations
    cooling_rate = 0.999 if stage < max_iterations //
stage_iterations // 3 else 0.997 if stage < max_iterations //
stage_iterations * 2 // 3 else 0.995
    return current_temperature * cooling_rate

with tqdm(total=max_iterations) as pbar:
    for iteration in range(max_iterations):
        if current_temperature < min_temperature or
current_cost == 0:
            break

        pos1, pos2 = get_problem_specific_neighbor()
        if random.random() < multi_swap_probability:
            new_cube = perform_multi_swap()
        else:
            cube.cube[pos1], cube.cube[pos2] =
cube.cube[pos2], cube.cube[pos1]

        new_cost = cube.calculate_actual_cost()
        cost_difference = new_cost - current_cost

        acceptance_prob =
calculate_acceptance_probability(cost_difference,
current_temperature)

        if random.random() < acceptance_prob:
            current_cost = new_cost
            steps.append({
                'index1': pos1,
                'index2': pos2,

```

```

        'cost': current_cost,
        'exp_value': acceptance_prob
    })
    if cost_difference < 0:
        consecutive_non_improvements = 0
        if new_cost < best_cost:
            best_cost = new_cost
            best_configuration = cube.cube.copy()
        else:
            consecutive_non_improvements += 1
    else:
        cube.cube[pos1], cube.cube[pos2] =
cube.cube[pos2], cube.cube[pos1]
        steps.append({
            'index1': pos1,
            'index2': pos2,
            'cost': current_cost,
            'exp_value': acceptance_prob
        })
        consecutive_non_improvements += 1

    # Check for being stuck in local optima
    if consecutive_non_improvements > 1000:
        stuck_in_local_optima += 1
        consecutive_non_improvements = 0 # Reset
counter after counting as stuck

    current_temperature =
adaptive_temperature_schedule(iteration)
    temperatures.append(current_temperature)
    pbar.update(1)
    pbar.set_postfix({'Cost': current_cost, 'Temp':
f'{current_temperature:.2f}', 'Best': best_cost})

    cube.cube = best_configuration
    return best_cost, best_configuration, iteration,
temperatures, steps, stuck_in_local_optima

```

### 2.2.6 Genetic Algorithm

Genetic algorithm adalah metode optimasi berbasis populasi yang terinspirasi dari proses seleksi alam, di mana individu terbaik dipertahankan untuk menghasilkan generasi berikutnya yang lebih baik. Genetic algorithm bekerja dengan membentuk populasi awal secara acak, lalu menggunakan mekanisme seleksi, crossover, dan mutasi untuk menghasilkan populasi baru. Seleksi mempertahankan individu terbaik, crossover menggabungkan karakteristik dari dua individu, sementara mutasi memperkenalkan variasi tambahan. Proses ini berlanjut hingga tercapai generasi yang memiliki solusi optimal atau mendekati optimal.

Kode di bawah mengimplementasikan algoritma genetika untuk mengoptimalkan penyelesaian Diagonal Magic Cube. Pertama, algoritma membentuk populasi awal dengan beberapa individu acak. Setiap individu dinilai menggunakan fungsi fitness yang menghitung “deviasi” atau perbedaan dari konfigurasi ideal. Individu dengan deviasi lebih rendah memiliki fitness lebih tinggi dan lebih mungkin untuk dipilih dalam proses seleksi.

Selanjutnya, algoritma melakukan crossover pada pasangan individu terbaik untuk menghasilkan keturunan baru. Proses crossover di sini menggunakan metode ordered crossover, di mana bagian dari orang tua ditransfer langsung ke anak, lalu diisi dengan elemen-elemen yang tidak duplikat dari orang tua lainnya. Setelah itu, mutasi diterapkan dengan probabilitas tertentu, di mana dua elemen dari individu secara acak ditukar untuk menambah keragaman dalam populasi dan mencegah stagnasi pada solusi local minima. Algoritma ini juga menggunakan metode elitisme untuk mempertahankan individu terbaik dalam populasi dari generasi ke generasi.

Proses ini diulang hingga mencapai jumlah generasi maksimum atau solusi yang diinginkan dengan deviasi nol ditemukan. Hasil akhir dari algoritma ini mencakup deviasi terendah yang ditemukan, konfigurasi kubus terbaik yang ditemukan, dan generasi di mana solusi tersebut ditemukan.

```
# Genetic Algorithm Parameters
POPULATION_SIZE = 100
MUTATION_RATE = 0.1
CUBE_SIZE = 5
MAX_GENERATIONS = 1000

# Objective function to calculate deviation
```

```

def calculate_deviation(cube):
    target_sum = CUBE_SIZE * (CUBE_SIZE**3 + 1) // 2 # The
    target sum for rows, columns, and diagonals
    deviation = 0

    # Check rows, columns, and pillars
    for i in range(CUBE_SIZE):
        deviation += abs(target_sum - np.sum(cube[i, :, :]))
    # Row sum deviation
        deviation += abs(target_sum - np.sum(cube[:, i, :]))
    # Column sum deviation
        deviation += abs(target_sum - np.sum(cube[:, :, i]))
    # Pillar sum deviation

    # Check main space diagonals
    deviation += abs(target_sum - np.sum([cube[i, i, i] for i
in range(CUBE_SIZE)])) # Main diagonal 1
    deviation += abs(target_sum - np.sum([cube[i, i,
CUBE_SIZE - i - 1] for i in range(CUBE_SIZE)])) # Main
diagonal 2
    deviation += abs(target_sum - np.sum([cube[i, CUBE_SIZE -
i - 1, i] for i in range(CUBE_SIZE)])) # Main diagonal 3
    deviation += abs(target_sum - np.sum([cube[CUBE_SIZE - i
- 1, i, i] for i in range(CUBE_SIZE)])) # Main diagonal 4

    return deviation # Return the total deviation

# Initialising a random individual
def create_individual():
    # Create a 3D array with unique numbers shuffled randomly
    return np.random.permutation(range(1, CUBE_SIZE**3 +
1)).reshape((CUBE_SIZE, CUBE_SIZE, CUBE_SIZE))

# Fitness function
def fitness(individual):
    # Return negative deviation for maximisation (lower
deviation is better)
    return -calculate_deviation(individual)

```

```

# Ordered crossover function
def ordered_crossover(parent1, parent2):
    size = CUBE_SIZE**3 # Total number of elements in the
    cube
    start, end = sorted(random.sample(range(size), 2)) #
    Random crossover points

    # Initialise children with placeholders
    child1, child2 = np.empty(size, dtype=int),
    np.empty(size, dtype=int)
    child1.fill(-1)
    child2.fill(-1)

    # Copy crossover segment from parents to children
    child1[start:end] = parent1[start:end]
    child2[start:end] = parent2[start:end]

    # Fill the rest of the child with non-duplicate elements
    from the other parent
    def fill_child(child, parent):
        current_pos = end % size
        for num in parent:
            if num not in child:
                child[current_pos] = num
                current_pos = (current_pos + 1) % size
        return child

    child1 = fill_child(child1, parent2)
    child2 = fill_child(child2, parent1)

    # Reshape children into 3D arrays and return
    return child1.reshape((CUBE_SIZE, CUBE_SIZE, CUBE_SIZE)),
    child2.reshape((CUBE_SIZE, CUBE_SIZE, CUBE_SIZE))

# Mutation function with duplicate prevention and correction
def mutate(individual):
    if random.random() < MUTATION_RATE: # Check if mutation
    should occur
        size = CUBE_SIZE**3

```



```

        flat_individual = individual.flatten() # Flatten the
        3D array for easier manipulation

        # Swap two random elements
        idx1, idx2 = random.sample(range(size), 2)
        flat_individual[idx1], flat_individual[idx2] =
flat_individual[idx2], flat_individual[idx1]

        # Check and correct duplicates
        unique_values = set(flat_individual)
        full_range = set(range(1, CUBE_SIZE**3 + 1))
        missing_values = full_range - unique_values # Find
        values that should be in the cube but aren't

        while len(unique_values) < size: # Check if
        duplicates exist
            for i in range(size):
                if
list(flat_individual).count(flat_individual[i]) > 1: #
        Detect duplicates
                    flat_individual[i] = missing_values.pop()
# Replace duplicate with a unique value
                    unique_values = set(flat_individual) #
        Update unique values set

        # Reshape the flat array back into a 3D array
        individual = flat_individual.reshape((CUBE_SIZE,
CUBE_SIZE, CUBE_SIZE))
        return individual

# Genetic algorithm function
def genetic_algorithm(cube, population_size, generations,
mutation_rate, elitism):
    # Create an initial population of individuals
    population = [create_individual() for _ in
range(population_size)]
    best_cost = population[0].calculate_actual_cost()
    best_cube = population[0]

```

```

    for generation in range(generations):
        # Sort the population by fitness (descending order)
        population = sorted(population, key=lambda ind:
fitness(ind), reverse=True)

        # Create a new population with elitism (retain top
individuals)
        new_population = population[:10] if elitism else []

        # Generate new individuals through crossover and
mutation
        while len(new_population) < population_size:
            parent1, parent2 = random.sample(population[:50],
2) # Select parents from the top 50
            offspring1, offspring2 =
ordered_crossover(parent1.flatten(), parent2.flatten())
            offspring1 = mutate(offspring1) # Apply mutation
            offspring2 = mutate(offspring2)
            new_population.extend([offspring1, offspring2])

        # Update the population for the next generation
        population = new_population[:population_size]

        # Check for the best fitness in the current
generation
        for ind in population:
            cost = ind.calculate_actual_cost()
            if cost < best_cost:
                best_cost = cost
                best_cube = ind
            if cost == 0:
                return best_cost, best_cube, generation

    # Return the final results
    return best_cost, best_cube, generation

```

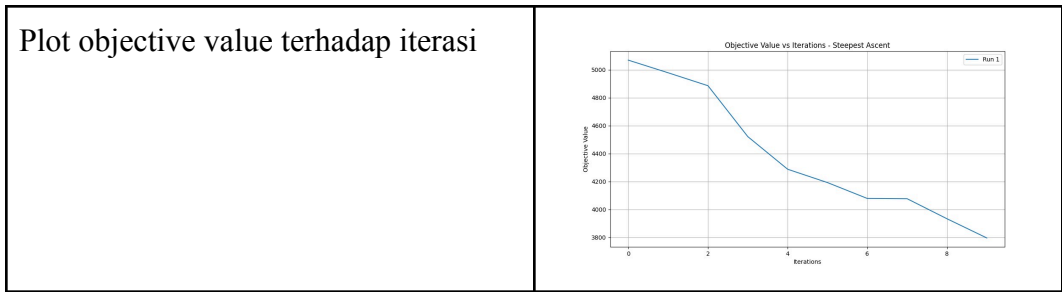
## 2.3 Hasil Eksperimen

Berikut merupakan hasil eksperimen pada percobaan-percobaan untuk setiap algoritma local search yang diimplementasikan sebelumnya:

### 2.3.1 Steepest Ascent Hill-Climbing

Percobaan 1

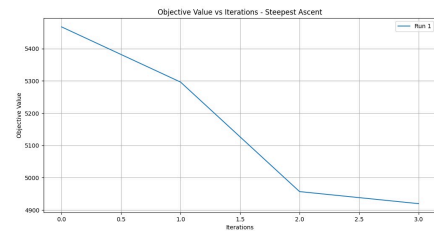
Objective Value dan Durasi	<code>initial_cost: 7000</code> <code>final_cost: 3000</code> <code>time: 32</code>
State Awal	State Akhir
<code>119, 13, 53, 121, 60, 52,</code> <code>27, 76, 57, 35, 79, 14, 105,</code> <code>4, 38, 69, 64, 50,</code> <code>49, 18, 40, 39, 84, 999,</code> <code>47, 58, 106, 51, 29, 115, 11,</code> <code>6, 72, 25, 74, 55, 68,</code> <code>23, 2, 104, 83, 87, 28,</code> <code>118, 102, 43, 117, 88, 12,</code> <code>48, 46, 41, 100, 32, 97,</code> <code>78, 95, 122, 70, 110,</code> <code>112, 20, 44, 81, 80, 19, 99,</code> <code>85, 42, 34, 16, 96, 86,</code> <code>92, 37, 89, 24, 108, 116,</code> <code>123, 101, 1, 114, 90, 103,</code> <code>33, 67, 65, 94, 45, 7,</code> <code>77, 62, 75, 111, 124, 73,</code> <code>30, 31, 120, 98, 59, 26, 107,</code> <code>109, 66, 91, 8, 113,</code> <code>56, 17, 125, 71, 22, 93,</code> <code>36, 21, 5, 63, 61, 9, 82, 10,</code> <code>54, 3</code>	<code>77, 62, 75, 111, 124,</code> <code>73, 30, 31, 120, 98, 59, 26,</code> <code>107, 109, 66, 91, 8,</code> <code>113, 56, 17, 125, 71,</code> <code>22, 93, 36, 21, 5, 63, 61, 9,</code> <code>82, 10, 54, 3, 49, 18,</code> <code>40, 39, 84, 15, 47, 58,</code> <code>106, 51, 29, 115, 11, 6, 72,</code> <code>25, 74, 55, 68, 23,</code> <code>2, 104, 83, 87, 28,</code> <code>118, 102, 43, 117, 88, 12,</code> <code>48, 46, 41, 100, 32, 97,</code> <code>119, 13, 53, 121, 60,</code> <code>52, 27, 76, 57, 35, 79, 14,</code> <code>105, 4, 38, 69, 64, 50,</code> <code>92, 37, 89, 24, 108,</code> <code>116, 123, 101, 1, 114, 90,</code> <code>103, 33, 67, 65, 94, 45,</code> <code>7, 78, 95, 122, 70,</code> <code>110, 112, 20, 44, 81, 80, 19,</code> <code>99, 85, 42, 34, 16, 96,</code> <code>86</code>



## Percobaan 2

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	

Plot objective value terhadap iterasi



Keterangan: typo seharusnya run 2

### Percobaan 3

Objective Value dan Durasi

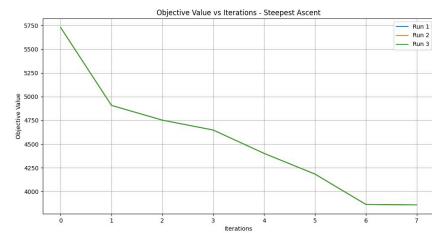
`initial_cost: 7000`

State Awal

State Akhir

```
119, 13, 53, 121, 60, 52,
27, 76, 57, 35, 79, 14, 105,
4, 38, 69, 64, 50,
49, 18, 40, 39, 84, 999,
47, 58, 106, 51, 29, 115, 11,
6, 72, 25, 74, 55, 68,
23, 2, 104, 83, 87, 28,
118, 102, 43, 117, 88, 12,
48, 46, 41, 100, 32, 97,
78, 95, 122, 70, 110,
112, 20, 44, 81, 80, 19, 99,
85, 42, 34, 16, 96, 86,
92, 37, 89, 24, 108, 116,
123, 101, 1, 114, 90, 103,
33, 67, 65, 94, 45, 7,
77, 62, 75, 111, 124, 73,
30, 31, 120, 98, 59, 26, 107,
109, 66, 91, 8, 113,
56, 17, 125, 71, 22, 93,
36, 21, 5, 63, 61, 9, 82, 10,
54, 3
```

Plot objective value terhadap iterasi



## 2.3.2 Hill-Climbing with Sideways Move

Percobaan 1

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	<pre> 7, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 40, 39, 84, 15, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3, 49, 18, 86 </pre>

Percobaan 2

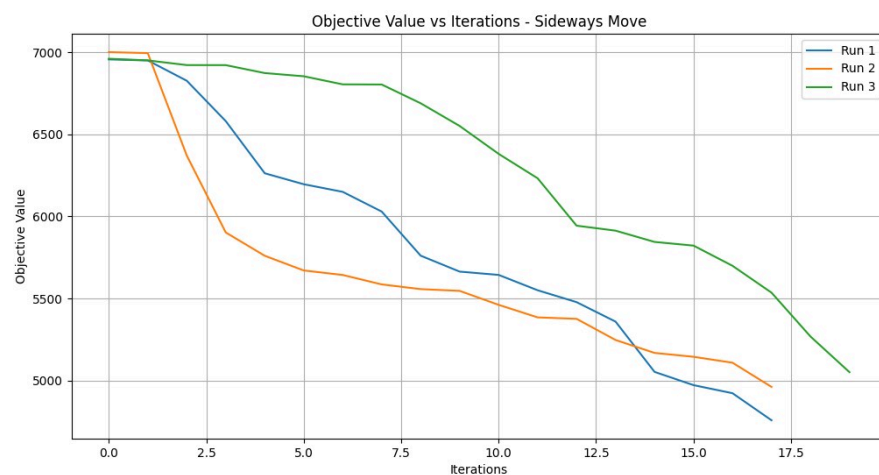
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir
<code>119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3</code>	

Percobaan 3

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir
<code>119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68,</code>	

<pre> 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	
---	--

Plot objective value terhadap iterasi pada ketiga percobaan



### 2.3.3 Random Restart Hill-Climbing

Percobaan 1: Jumlah restart 3

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir



<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	
---	--

## Percobaan 2: Jumlah restart 3

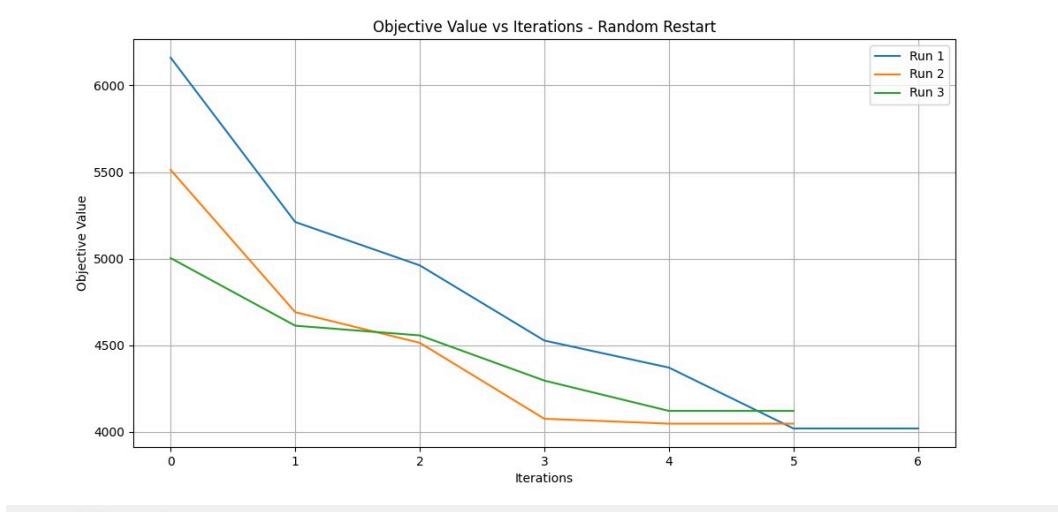
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, </pre>	

85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3	
--	--

### Percobaan 3: Jumlah restart 3

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir
119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3	

Plot objective value terhadap iterasi untuk ketiga percobaan



2.3.4 Stochastic Hill-Climbing

Percobaan 1

Objective Value dan Durasi	<div>initial_cost: 7000 final_cost: 2100 time: 23</div>
State Awal	State Akhir
<div>119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103,</div>	<div>7, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 40, 39, 84, 15, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14,</div>

33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3	105, 4, 38, 69, 64, 50, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3, 49, 18, 86
--	---

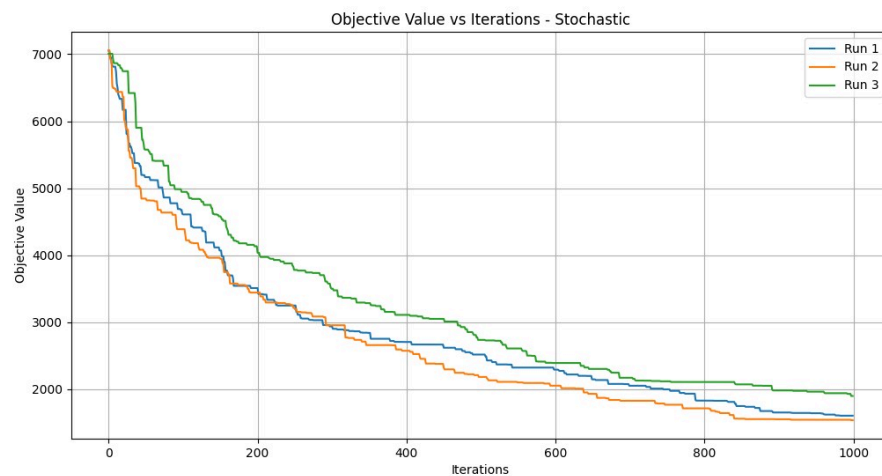
## Percobaan 2

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir
119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3	

## Percobaan 3

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	

Plot objective value terhadap iterasi untuk ketiga percobaan



### 2.3.5 Simulated Annealing

Keterangan: bila perlu

Percobaan 1: Jumlah iterasi ...

Objective Value dan Durasi	<code>initial_cost: 7000</code> <code>final_cost: 1220</code> <code>time: 156</code> <code>stuck_frequency: 12</code>
State Awal	State Akhir
<code>119, 13, 53, 121, 60, 52,</code> <code>27, 76, 57, 35, 79, 14, 105,</code> <code>4, 38, 69, 64, 50,</code> <code>49, 18, 40, 39, 84, 999,</code> <code>47, 58, 106, 51, 29, 115, 11,</code> <code>6, 72, 25, 74, 55, 68,</code> <code>23, 2, 104, 83, 87, 28,</code> <code>118, 102, 43, 117, 88, 12,</code> <code>48, 46, 41, 100, 32, 97,</code> <code>78, 95, 122, 70, 110,</code> <code>112, 20, 44, 81, 80, 19, 99,</code> <code>85, 42, 34, 16, 96, 86,</code> <code>92, 37, 89, 24, 108, 116,</code> <code>123, 101, 1, 114, 90, 103,</code> <code>33, 67, 65, 94, 45, 7,</code> <code>77, 62, 75, 111, 124, 73,</code> <code>30, 31, 120, 98, 59, 26, 107,</code> <code>109, 66, 91, 8, 113,</code> <code>56, 17, 125, 71, 22, 93,</code> <code>36, 21, 5, 63, 61, 9, 82, 10,</code> <code>54, 3</code>	<code>77, 62, 75, 111, 124,</code> <code>73, 30, 31, 120, 98, 59, 26,</code> <code>107, 109, 66, 91, 8, 7,</code> <code>78, 95, 122, 70, 110,</code> <code>112, 20, 44, 81, 80, 19, 99,</code> <code>85, 42, 34, 16, 96, 40,</code> <code>39, 84, 15, 47, 58,</code> <code>106, 51, 29, 115, 11, 6, 72,</code> <code>25, 74, 55, 68, 23, 2,</code> <code>104, 83, 87, 28, 118,</code> <code>102, 43, 117, 88, 12, 48, 46,</code> <code>41, 100, 32, 97, 119,</code> <code>13, 53, 121, 60, 52,</code> <code>27, 76, 57, 35, 79, 14, 105,</code> <code>4, 38, 69, 64, 50, 113,</code> <code>56, 17, 125, 71, 22,</code> <code>93, 36, 21, 5, 63, 61, 9, 82,</code> <code>10, 54, 3, 49, 18, 86,</code> <code>92, 37, 89, 24, 108,</code> <code>116, 123, 101, 1, 114, 90,</code> <code>103, 33, 67, 65, 94, 45</code>

Percobaan 2: Jumlah iterasi ...

Objective Value dan Durasi	<code>initial_cost: 7000</code>
----------------------------	---------------------------------

State Awal	State Akhir
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	

### Percobaan 3: Jumlah iterasi ...

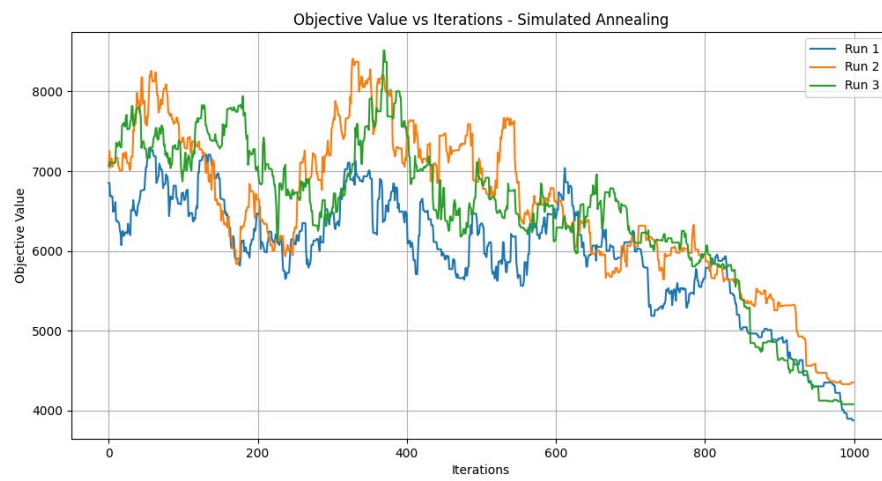
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, </pre>	

```

112, 20, 44, 81, 80, 19, 99,
85, 42, 34, 16, 96, 86,
    92, 37, 89, 24, 108, 116,
123, 101, 1, 114, 90, 103,
33, 67, 65, 94, 45, 7,
    77, 62, 75, 111, 124, 73,
30, 31, 120, 98, 59, 26, 107,
109, 66, 91, 8, 113,
    56, 17, 125, 71, 22, 93,
36, 21, 5, 63, 61, 9, 82, 10,
54, 3

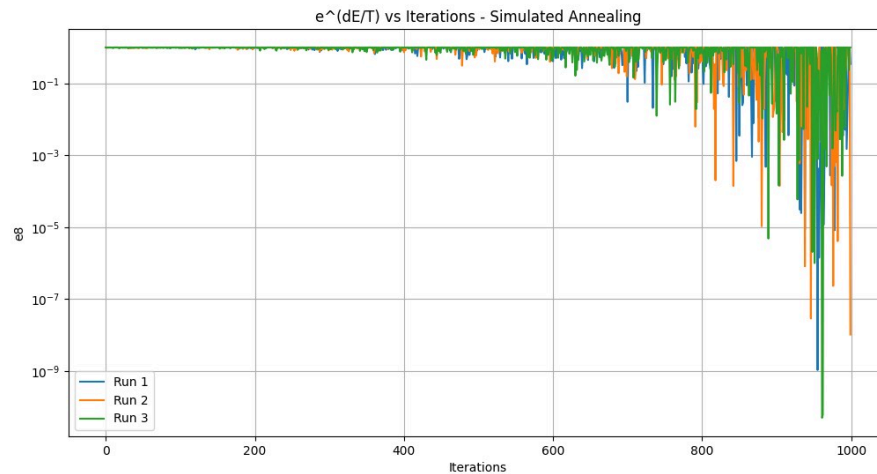
```

Plot objective value terhadap iterasi untuk ketiga percobaan



Plot  $e^{\Delta E/t}$  terhadap iterasi untuk ketiga percobaan





## 2.3.6 Genetic Algorithm

Keterangan: bila perlu

Percobaan 1: Iterasi = 3000 dan Jumlah Populasi = 5

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, </pre>	<pre> "[100, 102, 47, 106, 43, 76, 29, 24, 118, 38,\n 73, 75, 83, 65, 33, 97, 109, 53, 107, 57,\n 17, 95, 89, 18, 58, 48, 9, 80, 103, 117,\n 4, 72, 12, 125, 112, 101, 114, 91, 64, 104,\n 27, 90, 124, 59, 60, 68, 40, 85, 11, 99,\n 115, 62, 93, 67, 21, 42, 121, 6, 111, 61,\n 37, 81, 69, 1, 35, 98, 39, 116, 54, 96,\n 108, 8, 110, 41, 45, 26, 30, 36, 19, 16,\n 88, 34, 44, 25, 7, 86, 82, 79, 10, 77,\n 119, 70, 66, 49, 22, 71, 123, 14, 46, 51,\n 55, 31, 120, 94, 15, 56, 28, 122, 92, 113,\n 52, 63, 3, 20, 2, 13, 5, 23, 50, 32,\n 78, 105, 74, 84, 87\n ]", "[4, 69, 44, 24, 28, 43, 40, 65, 92, 99,\n 78, 21, 37, 94, 103, 18, 73, 121, 11, 104,\n 2, 125, 30, 1, 109, 122, 33, 15, 93, 6,\n 23, 31, 8, 49, </pre>

<p>30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3</p>	<p>74, 96, 41, 52, 61, 45,\n 80, 84, 120, 76, 79, 57, 123, 82, 68, 38,\n 107, 9, 64, 114, 110, 100, 55, 56, 81, 14,\n 67, 29, 36, 47, 66, 48, 60, 124, 3, 97,\n 91, 98, 111, 115, 118, 22, 117, 106, 34, 105,\n 53, 88, 62, 5, 46, 116, 72, 42, 95, 51,\n 108, 27, 86, 16, 85, 90, 17, 101, 20, 54,\n 70, 19, 26, 59, 83, 75, 77, 119, 25, 39,\n 13, 112, 7, 35, 50, 87, 113, 102, 32, 89,\n 58, 71, 12, 63, 10\n ]", "[89, 33, 6, 73, 72, 23, 15, 55, 14, 13,\n 16, 41, 25, 49, 118, 113, 57, 81, 103, 80,\n 56, 109, 71, 37, 62, 117, 79, 66, 115, 40,\n 4, 20, 63, 64, 70, 10, 91, 111, 50, 30,\n 29, 21, 96, 36, 7, 68, 92, 90, 24, 39,\n 69, 87, 101, 12, 54, 44, 52, 47, 38, 61,\n 31, 32, 93, 75, 105, 108, 125, 74, 124, 106,\n 45, 77, 42, 22, 2, 112, 58, 120, 100, 35,\n 28, 26, 60, 95, 65, 98, 8, 84, 59, 78,\n 3, 99, 46, 83, 48, 11, 121, 5, 1, 116,\n 102, 86, 9, 53, 88, 122, 76, 18, 123, 85,\n 107, 19, 97, 82, 43, 110, 104, 34, 51, 119,\n 94, 17, 114, 27, 67\n ]"</p>
	<p>State Akhir (Anak 2)</p>
	<p>"[100, 102, 47, 106, 43, 76, 29, 24, 118, 38,\n 73, 75, 83, 65, 33, 97, 109, 53, 107, 57,\n 17, 95, 89, 18, 58, 48, 9, 80, 103, 117,\n 4, 72, 12, 125, 112, 101, 114, 91, 64, 104,\n 27, 90, 124, 59, 60, 68, 40, 85, 11, 99,\n 115, 62, 93, 67, 21, 42, 121, 6, 111, 61,\n 37, 81, 69, 1, 35, 98, 39, 116, 54, 96,\n 108, 8, 110, 41, 45, 26, 30, 36, 19, 16,\n 88, 34, 44, 25, 7, 86, 82, 79, 10, 77,\n 119, 70, 66, 49, 22, 71, 123, 14, 46, 51,\n 55, 31, 120, 94, 15, 56, 28, 122, 92, 113,\n 52, 63, 3, 20, 2, 13, 5, 23, 50, 32,\n 78, 105, 74, 84, 87\n ]", "[4, 69, 44, 24, 28, 43, 40, 65, 92,</p>

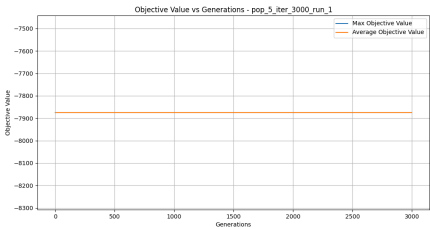
	<p>99,\n 78, 21, 37, 94, 103, 18, 73, 121, 11, 104,\n 2, 125, 30, 1, 109, 122, 33, 15, 93, 6,\n 23, 31, 8, 49, 74, 96, 41, 52, 61, 45,\n 80, 84, 120, 76, 79, 57, 123, 82, 68, 38,\n 107, 9, 64, 114, 110, 100, 55, 56, 81, 14,\n 67, 29, 36, 47, 66, 48, 60, 124, 3, 97,\n 91, 98, 111, 115, 118, 22, 117, 106, 34, 105,\n 53, 88, 62, 5, 46, 116, 72, 42, 95, 51,\n 108, 27, 86, 16, 85, 90, 17, 101, 20, 54,\n 70, 19, 26, 59, 83, 75, 77, 119, 25, 39,\n 13, 112, 7, 35, 50, 87, 113, 102, 32, 89,\n 58, 71, 12, 63, 10\n ]",</p> <p>"[89, 33, 6, 73, 72, 23, 15, 55, 14, 13,\n 16, 41, 25, 49, 118, 113, 57, 81, 103, 80,\n 56, 109, 71, 37, 62, 117, 79, 66, 115, 40,\n 4, 20, 63, 64, 70, 10, 91, 111, 50, 30,\n 29, 21, 96, 36, 7, 68, 92, 90, 24, 39,\n 69, 87, 101, 12, 54, 44, 52, 47, 38, 61,\n 31, 32, 93, 75, 105, 108, 125, 74, 124, 106,\n 45, 77, 42, 22, 2, 112, 58, 120, 100, 35,\n 28, 26, 60, 95, 65, 98, 8, 84, 59, 78,\n 3, 99, 46, 83, 48, 11, 121, 5, 1, 116,\n 102, 86, 9, 53, 88, 122, 76, 18, 123, 85,\n 107, 19, 97, 82, 43, 110, 104, 34, 51, 119,\n 94, 17, 114, 27, 67\n ]"</p>
	State Akhir (Anak 3)
	<p>"[100, 102, 47, 106, 43, 76, 29, 24, 118, 38,\n 73, 75, 83, 65, 33, 97, 109, 53, 107, 57,\n 17, 95, 89, 18, 58, 48, 9, 80, 103, 117,\n 4, 72, 12, 125, 112, 101, 114, 91, 64, 104,\n 27, 90, 124, 59, 60, 68, 40, 85, 11, 99,\n 115, 62, 93, 67, 21, 42, 121, 6, 111, 61,\n 37, 81, 69, 1, 35, 98, 39, 116, 54, 96,\n 108, 8, 110, 41, 45, 26, 30, 36, 19, 16,\n 88, 34, 44, 25, 7, 86, 82, 79, 10, 77,\n 119, 70, 66, 49, 22, 71, 123, 14, 46, 51,\n 55, 31, 120, 94, 15, 56, 28, 122, 92, 113,\n ]"</p>

	<p>52, 63, 3, 20, 2, 13, 5, 23, 50, 32,\n 78, 105, 74, 84, 87\n ]",</p> <p>"[4, 69, 44, 24, 28, 43, 40, 65, 92, 99,\n 78, 21, 37, 94, 103, 18, 73, 121, 11, 104,\n 2, 125, 30, 1, 109, 122, 33, 15, 93, 6,\n 23, 31, 8, 49, 74, 96, 41, 52, 61, 45,\n 80, 84, 120, 76, 79, 57, 123, 82, 68, 38,\n 107, 9, 64, 114, 110, 100, 55, 56, 81, 14,\n 67, 29, 36, 47, 66, 48, 60, 124, 3, 97,\n 91, 98, 111, 115, 118, 22, 117, 106, 34, 105,\n 53, 88, 62, 5, 46, 116, 72, 42, 95, 51,\n 108, 27, 86, 16, 85, 90, 17, 101, 20, 54,\n 70, 19, 26, 59, 83, 75, 77, 119, 25, 39,\n 13, 112, 7, 35, 50, 87, 113, 102, 32, 89,\n 58, 71, 12, 63, 10\n ]",</p> <p>"[89, 33, 6, 73, 72, 23, 15, 55, 14, 13,\n 16, 41, 25, 49, 118, 113, 57, 81, 103, 80,\n 56, 109, 71, 37, 62, 117, 79, 66, 115, 40,\n 4, 20, 63, 64, 70, 10, 91, 111, 50, 30,\n 29, 21, 96, 36, 7, 68, 92, 90, 24, 39,\n 69, 87, 101, 12, 54, 44, 52, 47, 38, 61,\n 31, 32, 93, 75, 105, 108, 125, 74, 124, 106,\n 45, 77, 42, 22, 2, 112, 58, 120, 100, 35,\n 28, 26, 60, 95, 65, 98, 8, 84, 59, 78,\n 3, 99, 46, 83, 48, 11, 121, 5, 1, 116,\n 102, 86, 9, 53, 88, 122, 76, 18, 123, 85,\n 107, 19, 97, 82, 43, 110, 104, 34, 51, 119,\n 94, 17, 114, 27, 67\n ]"</p>
	State Akhir (Anak 4)
	<p>"[100, 102, 47, 106, 43, 76, 29, 24, 118, 38,\n 73, 75, 83, 65, 33, 97, 109, 53, 107, 57,\n 17, 95, 89, 18, 58, 48, 9, 80, 103, 117,\n 4, 72, 12, 125, 112, 101, 114, 91, 64, 104,\n 27, 90, 124, 59, 60, 68, 40, 85, 11, 99,\n 115, 62, 93, 67, 21, 42, 121, 6, 111, 61,\n 37, 81, 69, 1, 35, 98, 39, 116, 54, 96,\n 108, 8, 110, 41, 45, 26, 30, 36, 19, 16,\n 88, 34, 44, 25,</p>

	<p>7, 86, 82, 79, 10, 77,\n 119, 70, 66, 49, 22, 71, 123, 14, 46, 51,\n 55, 31, 120, 94, 15, 56, 28, 122, 92, 113,\n 52, 63, 3, 20, 2, 13, 5, 23, 50, 32,\n 78, 105, 74, 84, 87\n ]",</p> <p>"[4, 69, 44, 24, 28, 43, 40, 65, 92, 99,\n 78, 21, 37, 94, 103, 18, 73, 121, 11, 104,\n 2, 125, 30, 1, 109, 122, 33, 15, 93, 6,\n 23, 31, 8, 49, 74, 96, 41, 52, 61, 45,\n 80, 84, 120, 76, 79, 57, 123, 82, 68, 38,\n 107, 9, 64, 114, 110, 100, 55, 56, 81, 14,\n 67, 29, 36, 47, 66, 48, 60, 124, 3, 97,\n 91, 98, 111, 115, 118, 22, 117, 106, 34, 105,\n 53, 88, 62, 5, 46, 116, 72, 42, 95, 51,\n 108, 27, 86, 16, 85, 90, 17, 101, 20, 54,\n 70, 19, 26, 59, 83, 75, 77, 119, 25, 39,\n 13, 112, 7, 35, 50, 87, 113, 102, 32, 89,\n 58, 71, 12, 63, 10\n ]",</p> <p>"[89, 33, 6, 73, 72, 23, 15, 55, 14, 13,\n 16, 41, 25, 49, 118, 113, 57, 81, 103, 80,\n 56, 109, 71, 37, 62, 117, 79, 66, 115, 40,\n 4, 20, 63, 64, 70, 10, 91, 111, 50, 30,\n 29, 21, 96, 36, 7, 68, 92, 90, 24, 39,\n 69, 87, 101, 12, 54, 44, 52, 47, 38, 61,\n 31, 32, 93, 75, 105, 108, 125, 74, 124, 106,\n 45, 77, 42, 22, 2, 112, 58, 120, 100, 35,\n 28, 26, 60, 95, 65, 98, 8, 84, 59, 78,\n 3, 99, 46, 83, 48, 11, 121, 5, 1, 116,\n 102, 86, 9, 53, 88, 122, 76, 18, 123, 85,\n 107, 19, 97, 82, 43, 110, 104, 34, 51, 119,\n 94, 17, 114, 27, 67\n ]"</p>
	State Akhir (Anak 5)
	<p>"[100, 102, 47, 106, 43, 76, 29, 24, 118, 38,\n 73, 75, 83, 65, 33, 97, 109, 53, 107, 57,\n 17, 95, 89, 18, 58, 48, 9, 80, 103, 117,\n 4, 72, 12, 125, 112, 101, 114, 91, 64, 104,\n 27, 90, 124, 59, 60, 68, 40, 85, 11, 99,\n 115, 62, 93, 67, 21, 42, 121, 6,</p>

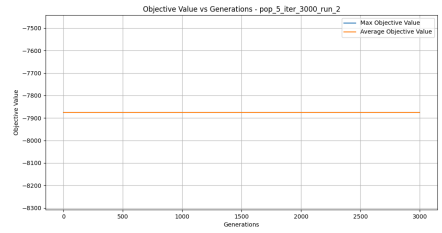
	<p>111, 61,\n 37, 81, 69, 1, 35, 98, 39,  116, 54, 96,\n 108, 8, 110, 41, 45,  26, 30, 36, 19, 16,\n 88, 34, 44, 25,  7, 86, 82, 79, 10, 77,\n 119, 70, 66,  49, 22, 71, 123, 14, 46, 51,\n 55, 31,  120, 94, 15, 56, 28, 122, 92, 113,\n  52, 63, 3, 20, 2, 13, 5, 23, 50, 32,\n  78, 105, 74, 84, 87\n ]",</p> <p>"[4, 69, 44, 24, 28, 43, 40, 65, 92,  99,\n 78, 21, 37, 94, 103, 18, 73,  121, 11, 104,\n 2, 125, 30, 1, 109,  122, 33, 15, 93, 6,\n 23, 31, 8, 49,  74, 96, 41, 52, 61, 45,\n 80, 84, 120,  76, 79, 57, 123, 82, 68, 38,\n 107, 9,  64, 114, 110, 100, 55, 56, 81, 14,\n  67, 29, 36, 47, 66, 48, 60, 124, 3, 97,\n  91, 98, 111, 115, 118, 22, 117, 106, 34,  105,\n 53, 88, 62, 5, 46, 116, 72, 42,  95, 51,\n 108, 27, 86, 16, 85, 90, 17,  101, 20, 54,\n 70, 19, 26, 59, 83, 75,  77, 119, 25, 39,\n 13, 112, 7, 35, 50,  87, 113, 102, 32, 89,\n 58, 71, 12,  63, 10\n ]",</p> <p>"[89, 33, 6, 73, 72, 23, 15, 55, 14,  13,\n 16, 41, 25, 49, 118, 113, 57,  81, 103, 80,\n 56, 109, 71, 37, 62,  117, 79, 66, 115, 40,\n 4, 20, 63, 64,  70, 10, 91, 111, 50, 30,\n 29, 21, 96,  36, 7, 68, 92, 90, 24, 39,\n 69, 87,  101, 12, 54, 44, 52, 47, 38, 61,\n 31,  32, 93, 75, 105, 108, 125, 74, 124,  106,\n 45, 77, 42, 22, 2, 112, 58,  120, 100, 35,\n 28, 26, 60, 95, 65,  98, 8, 84, 59, 78,\n 3, 99, 46, 83, 48,  11, 121, 5, 1, 116,\n 102, 86, 9, 53,  88, 122, 76, 18, 123, 85,\n 107, 19,  97, 82, 43, 110, 104, 34, 51, 119,\n  94, 17, 114, 27, 67\n ]"</p>
--	---

Plot Objective Value terhadap Iterasi



Percobaan 2: Iterasi = 3000 dan Jumlah Populasi = 5

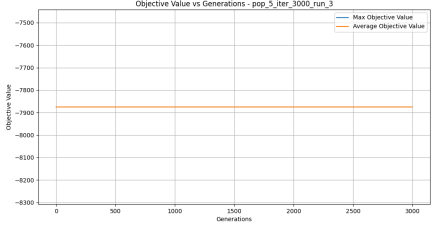
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	
	State Akhir (Anak 2)
	State Akhir (Anak 3)

	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 3: Iterasi = 3000 dan Jumlah Populasi = 5

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, </pre>	



56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3	
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 4: Iterasi = 30000 dan Jumlah Populasi = 5

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110,	

<pre> 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86,     92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7,     77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113,     56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 4: Iterasi = 3000... dan Jumlah Populasi = 7

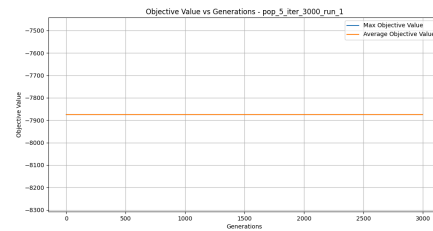
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, </pre>	

<div> 49, 18, 40, 39, 84, 999,  47, 58, 106, 51, 29, 115, 11,  6, 72, 25, 74, 55, 68,  23, 2, 104, 83, 87, 28,  118, 102, 43, 117, 88, 12,  48, 46, 41, 100, 32, 97,  78, 95, 122, 70, 110,  112, 20, 44, 81, 80, 19, 99,  85, 42, 34, 16, 96, 86,  92, 37, 89, 24, 108, 116,  123, 101, 1, 114, 90, 103,  33, 67, 65, 94, 45, 7,  77, 62, 75, 111, 124, 73,  30, 31, 120, 98, 59, 26, 107,  109, 66, 91, 8, 113,  56, 17, 125, 71, 22, 93,  36, 21, 5, 63, 61, 9, 82, 10,  54, 3 </div>	
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 5: Iterasi = 30000 dan Jumlah Populasi = 5

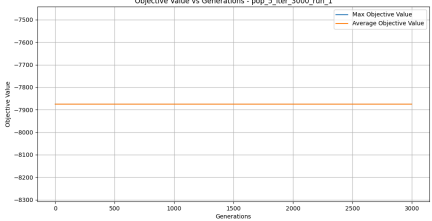
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
<code>119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3</code>	
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)

Plot Objective Value terhadap Iterasi



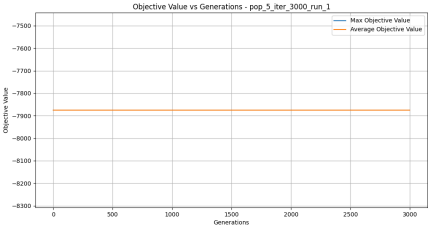
Percobaan 6: Iterasi = 30000 dan Jumlah Populasi = 5

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
<code>119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3</code>	
	State Akhir (Anak 2)
	State Akhir (Anak 3)

	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 7: Iterasi = 300000 dan Jumlah Populasi = 5

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
<code>119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7, 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3</code>	

	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 8: Iterasi = 300000 dan Jumlah Populasi = 5

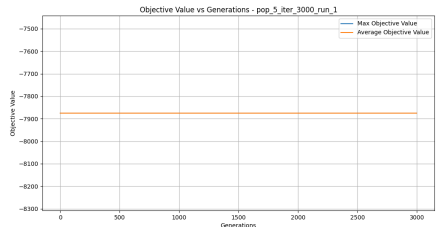
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
<div>119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, 48, 46, 41, 100, 32, 97, 78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86, 92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7,</div>	

<pre> 77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113, 56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 9: Iterasi = 300000 dan Jumlah Populasi = 5

Objective Value dan Durasi	<pre> initial_cost: 7000 </pre>
State Awal (Parent 1)	State Akhir (Anak 1)
<pre> 119, 13, 53, 121, 60, 52, 27, 76, 57, 35, 79, 14, 105, 4, 38, 69, 64, 50, 49, 18, 40, 39, 84, 999, 47, 58, 106, 51, 29, 115, 11, 6, 72, 25, 74, 55, 68, 23, 2, 104, 83, 87, 28, 118, 102, 43, 117, 88, 12, </pre>	



<pre> 48, 46, 41, 100, 32, 97,     78, 95, 122, 70, 110, 112, 20, 44, 81, 80, 19, 99, 85, 42, 34, 16, 96, 86,     92, 37, 89, 24, 108, 116, 123, 101, 1, 114, 90, 103, 33, 67, 65, 94, 45, 7,     77, 62, 75, 111, 124, 73, 30, 31, 120, 98, 59, 26, 107, 109, 66, 91, 8, 113,     56, 17, 125, 71, 22, 93, 36, 21, 5, 63, 61, 9, 82, 10, 54, 3 </pre>	
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 10: Iterasi = 3000 dan Jumlah Populasi = 7

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
<code>119, 13, 53, 121, 60, 52,</code>	

27, 76, 57, 35, 79, 14, 105,  
4, 38, 69, 64, 50,  
49, 18, 40, 39, 84, 999,  
47, 58, 106, 51, 29, 115, 11,  
6, 72, 25, 74, 55, 68,  
23, 2, 104, 83, 87, 28,  
118, 102, 43, 117, 88, 12,  
48, 46, 41, 100, 32, 97,  
78, 95, 122, 70, 110,  
112, 20, 44, 81, 80, 19, 99,  
85, 42, 34, 16, 96, 86,  
92, 37, 89, 24, 108, 116,  
123, 101, 1, 114, 90, 103,  
33, 67, 65, 94, 45, 7,  
77, 62, 75, 111, 124, 73,  
30, 31, 120, 98, 59, 26, 107,  
109, 66, 91, 8, 113,  
56, 17, 125, 71, 22, 93,  
36, 21, 5, 63, 61, 9, 82, 10,  
54, 3

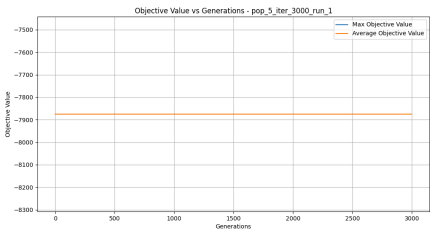
State Akhir (Anak 2)

State Akhir (Anak 3)

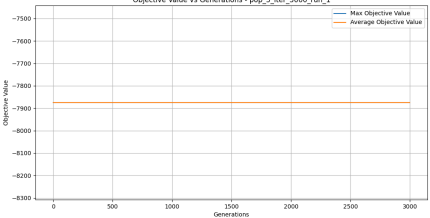
State Akhir (Anak 4)

State Akhir (Anak 5)

Plot Objective Value terhadap Iterasi

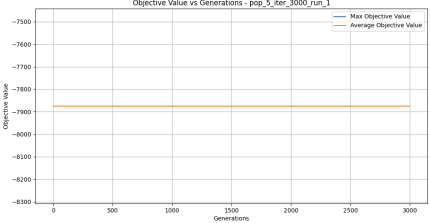


Percobaan 11: Iterasi = 3000 dan Jumlah Populasi = 7

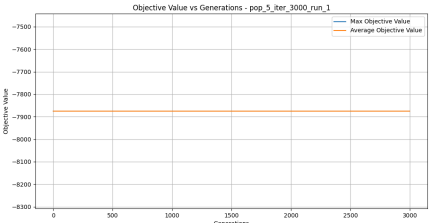
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 12: Iterasi = 3000 dan Jumlah Populasi = 7

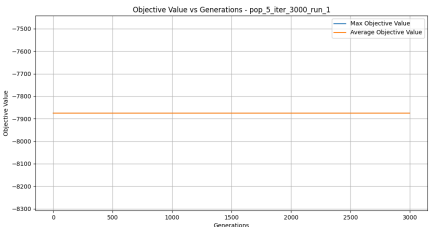
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)

	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 13: Iterasi = 30000 dan Jumlah Populasi = 7

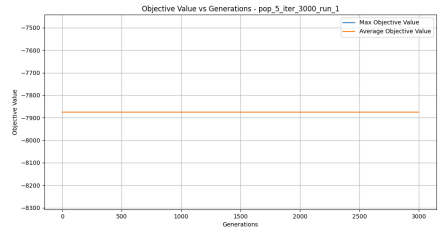
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 14: Iterasi = 30000 dan Jumlah Populasi = 7

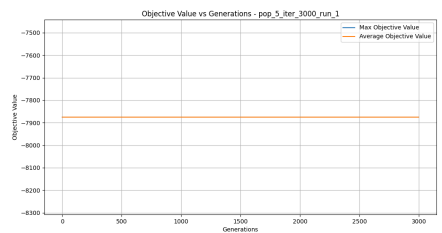
Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 15: Iterasi = 30000 dan Jumlah Populasi = 7

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)

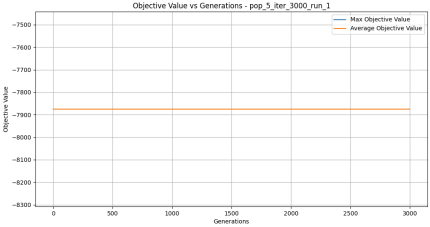
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 16: Iterasi = 3000 dan Jumlah Populasi = 10

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal (Parent 1)	State Akhir (Anak 1)
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

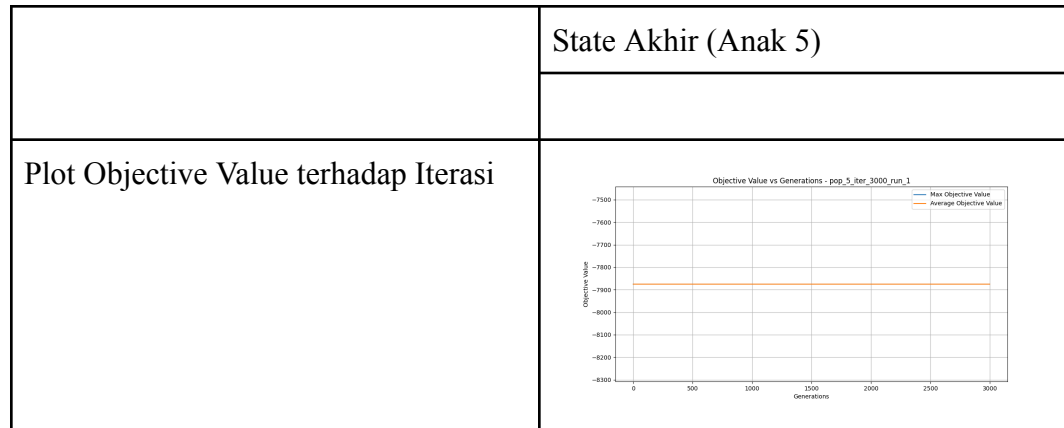
Percobaan 17: Iterasi = 30000 dan Jumlah Populasi = 10

Objective Value dan Durasi	<code>initial_cost: 7000</code>
----------------------------	---------------------------------

State Awal (Parent 1)	State Akhir (Anak 1)
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)
	State Akhir (Anak 5)
Plot Objective Value terhadap Iterasi	

Percobaan 18: Iterasi = 300000 dan Jumlah Populasi = 10

Objective Value dan Durasi	<code>initial_cost: 7000</code>
State Awal	State Akhir (Anak 1)
	State Akhir (Anak 2)
	State Akhir (Anak 3)
	State Akhir (Anak 4)



## 2.4 Hasil Analisis

Berikut merupakan tabel hasil analisis yang menyimpulkan hasil eksperimen dari tiap algoritma local search sebelumnya dengan Objective Value Terbaik yang diambil:

Algoritma	Objective Value Terbaik	Durasi Rata-Rata dari Durasi Eksperimen (detik)
Steepest Ascent Hill-Climbing	4708	29.4
Hill-Climbing with Sideways Move	4736	54.3
Random Restart Hill-Climbing	4019	105.4
Stochastic Hill-Climbing	1538.5	0.7
Simulated Annealing	4076	0.9
Genetic Algorithm	2058	60

Penjelasan lorem ipsum

Seberapa dekat tiap-tiap algoritma bisa mendekati global optima dan mengapa hasilnya demikian?

Bagaimana perbandingan hasil pencarian tiap-tiap algoritma dengan algoritma local search yang lain?



Bagaimana perbandingan durasi proses pencarian tiap algoritma relatif terhadap algoritma lainnya?

Seberapa konsisten hasil akhir yang didapatkan dari tiap-tiap eksperimen yang dilakukan?

Bagaimana pengaruh banyak iterasi dan jumlah populasi terhadap hasil akhir pencarian pada Genetic Algorithm?

# **BAB III**

## **Kesimpulan dan Saran**

Dari hasil percobaan yang kami lakukan, didapatkan bahwa percobaan dengan algoritma stochastic h

## BAB IV

# Pembagian Tugas

Berikut merupakan tabel dari pembagian tugas ini.

<b>Nama</b>	<b>NIM</b>	<b>Tugas</b>
Raizan Iqbal Resi	18222068	Membuat laporan Membuat code M
Favian Izza Diasputra	18222070	
Ahmad Habibie Marjan	18222082	
Muhammad Raihan Ariffianto	18222092	

# **BAB V**

## **Referensi**

- 5.1 Weisstein, Eric W. Magic Cube. diakses dari <https://mathworld.wolfram.com/MagicCube.html>. diakses 2 Oktober 2024.
- 5.2 Triumph, Walter. 2023. Perfect Magic Cubes. diakses dari <https://www.trump.de/magic-squares/magic-cubes/cubes-1.html>. diakses 2 Oktober 2024.
- 5.3 Russel, Stuart dan Norvig, Peter. 2021. Artificial Intelligence: A Modern Approach, 4th Global ed.
- 5.4 Leite, et. al. 2014. Distributed Constraint Optimization Problems: Review and perspectives. diakses dari <https://www.sciencedirect.com/science/article/abs/pii/S0957417414001158>. diakses 3 Oktober 2024.