LAPORAN TUGAS BESAR 1

IF3070 FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

Pencarian Solusi Pengepakan Barang (*Bin Packing Problem*) dengan Local Search

Disusun Oleh:

Valereo Jibril Al Buchori	18223030
Mahesa Satria Prayata	18223082
Jason Samuel	18223091

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

Daftar Isi

Daftar Isi 2
Daftar Tabel 3
Daftar Gambar 4
1. Deskripsi Persoalan 5
2. Pembahasan 5
2.1. Class 5
2.1.1. Class Item 5
2.1.2. Class Container 6
2.1.3. Class State 6
2.2. Objective Function 8
2.3. Implementasi Local Search 9
2.3.1. Implementasi Hill Climbing 9
2.3.1.1. Implementasi Steepest Ascent Hill Climbing 11
2.3.1.2. Implementasi Sideways Move Hill Climbing 12
2.3.1.3. Implementasi Stochastic Hill Climbing 13
2.3.1.4. Implementasi Random Restart Hill Climbing 14
2.3.1.5. 15
2.3.2. Implementasi Simulated Annealing 15
2.3.3. Implementasi Genetic Algorithm 18
2.4. Hasil Implementasi 22
2.4.1. Hasil Steepest Ascent Hill Climbing 23
2.4.2. Hasil Sideways Move Hill Climbing 29
2.4.3. Hasil Stochastic Hill Climbing 35
2.4.4. Hasil Random Restart Hill Climbing 41
2.4.5. Hasil Simulated Annealing 48
2.4.6. Hasil Genetic Algorithm 54
2.4.6.1. Variasi Populasi 58
2.4.6.2. Variasi Iterasi 74
2.4.6.3. Analisis 91
3. Kesimpulan dan Saran 92
Pembagian Tugas 95
Github Repository 96
Referensi 97

Daftar Tabel

Tabel 1.1 Dafter Variabel.....5
Tabel 2.1 Hasil Hill Climbing Steepest Ascent.....23
Tabel 2.2 Hasil Hill Climbing Sideways Move.....29
Tabel 2.3 Hasil Stochastic Hill Climbing.....35
Tabel 2.4 Hasil Random Restart Hill Climbing.....41
Tabel 2.5 Tabel gambar plotting Simulated Annealing.....48
Tabel 2.6 Tabel gambar plotting Genetic Algorithm dengan variasi populasi.....54
Tabel 2.7 Tabel gambar plotting Genetic Algorithm dengan variasi populasi.....56
Tabel 2.8 State awal dan akhir Genetic Algorithm variasi populasi penjalanan program pertama kali.....58
Tabel 2.9 State awal dan akhir Genetic Algorithm variasi populasi penjalanan program kedua kali.....63
Tabel 2.10 State awal dan akhir Genetic Algorithm variasi populasi penjalanan program ketiga kali.....69
Tabel 2.11 State awal dan akhir Genetic Algorithm variasi iterasi penjalanan program pertama kali.....74
Tabel 2.12 State awal dan akhir Genetic Algorithm variasi iterasi penjalanan program kedua kali.....80
Tabel 2.13 State awal dan akhir Genetic Algorithm variasi iterasi penjalanan program ketiga kali.....86

Daftar Gambar

Gambar 2.1 Plot Hasil Algoritma Steepest Ascent Hill Climbing.....28
Gambar 2.2 Plot Hasil Algoritma Sideways Hill Climbing.....34
Gambar 2.3 Plot Hasil Algoritma Stochastic Hill Climbing.....40
Gambar 2.4 Plot Hasil Algoritma Random Restart Hill Climbing.....47
Gambar 2.5 Plot objective function dengan iterasi - simulated annealing.....53
Gambar 2.6 Plot eET dengan iterasi - simulated annealing.....53
Gambar 2.7 Plot objective function dengan iterasi pada percobaan variasi populasi penjalanan program pertama kali.....54
Gambar 2.8 Plot objective function dengan iterasi pada percobaan variasi populasi penjalanan program kedua kali.....55
Gambar 2.9 Plot objective function dengan iterasi pada percobaan variasi populasi penjalanan program ketiga kali.....55
Gambar 2.10 Plot objective function dengan iterasi pada percobaan variasi iterasi penjalanan program pertama kali.....56
Gambar 2.11 Plot objective function dengan iterasi pada percobaan variasi iterasi penjalanan program kedua kali.....57
Gambar 2.12 Plot objective function dengan iterasi pada percobaan variasi iterasi penjalanan program ketiga kali.....57

1. Deskripsi Persoalan

Persoalan pada tugas besar ini adalah pencarian solusi pengepakan barang dengan menggunakan algoritma local search. Tujuan utama dari program ini adalah untuk menggunakan jumlah kontainer sesedikit mungkin.

Tabel 1.1 Dafter Variabel

Entitas	Atribut	Contoh	Keterangan
Barang	ID Barang	BRG001	Kode unik setiap barang
	Ukuran	25	Ukuran atau berat barang
Kontainer	Kapasitas	100	Setiap kontainer memiliki kapasitas yang sama dan seragam

Move tiap iterasi yang dibolehkan pada persoalan ini adalah:

- Memindahkan satu barang dari satu kontainer ke kontainer lain (yang sudah ada atau yang baru).
- Menukar dua barang dari dua kontainer yang berbeda.

2. Pembahasan

2.1. Class

2.1.1. Class *Item*

```
class Item:
    id: str
    size: int
```

Class Item merepresentasikan suatu barang yang terdiri dari atribut id (*string*) dan size (*int*)

2.1.2. Class *Container*

```
class Container:
    capacity: int
    item_list: list[Item]

    def __init__(self, capacity : int):
        self.capacity = capacity
        self.item_list = []

    def total_size(self):
        return sum(item['ukuran'] for item in self.item_list)
```

Class Container merepresentasikan suatu kontainer yang dapat diisi dari barang (*class item*). Class ini terdiri dari atribut capacity (*int*) dan item_list (*list[item]*) untuk menyimpan barang-barang yang ada di dalam kontainer. Class ini memiliki constructor dan method *total_size()* untuk mengambil total ukuran semua barang yang ada di dalam kontainer.

2.1.3. Class *State*

```
class State:
    list_container: list[Container]

    def __init__(self):
        self.list_container = []

    def count_penalty(self):
        penalty = 0
        count = 0
        for container in self.list_container:
            count += 1
            total_size = sum(item['ukuran'] for item in
container.item_list)
            if total_size > container.capacity:
                penalty += (total_size - container.capacity)*10000
            else :
                penalty += (container.capacity - total_size)
        penalty += count * 100
        return penalty

    def first_fit(self, filepath):
        script_dir = os.path.dirname(os.path.abspath(__file__))
        parent_dir = os.path.dirname(script_dir)
        problem_path = os.path.join(parent_dir, 'data',
f'{filepath}.json')
        list_items, capacity = load_problem(problem_path)
```

```
        self.list_container = []

        for item in list_items:
            item_placed = False
            for container in self.list_container:
                if container.total_size() + item['ukuran'] <=
capacity:
                    container.item_list.append(item)
                    item_placed = True
                    break
            if not item_placed:
                new_container = Container(capacity=capacity)
                new_container.item_list.append(item)
                self.list_container.append(new_container)

    def copy(self):
        new_state = State()
        for container in self.list_container:
            new_container = Container(capacity=container.capacity)
            new_container.item_list = container.item_list.copy()
            new_state.list_container.append(new_container)
        return new_state

    def generate_random_state(self, filepath):
        import random
        import os

        script_dir = os.path.dirname(os.path.abspath(__file__))
        parent_dir = os.path.dirname(script_dir)
        problem_path = os.path.join(parent_dir, 'data',
f'{filepath}.json')
        list_items, capacity = load_problem(problem_path)
        random.shuffle(list_items)
        num_containers = random.randint(1, len(list_items))

        self.list_container = [Container(capacity=capacity) for _ in
range(num_containers)]

        for item in list_items:
            random.choice(self.list_container).item_list.append(item)
        if random.random() < 0.2:
            extra_empty = random.randint(1, 3)
            for _ in range(extra_empty):

self.list_container.append(Container(capacity=capacity))
```

Class State mempunyai atribut list_container (*list[Container]*) yang menyimpan kondisi dari semua kontainer yang akan diproses. Class ini memiliki constructor dan beberapa method, *count_penalty()* untuk menghitung objective function, *first_fit()* sebagai cara pertama untuk inisialisasi state awal dengan memasukkan barang ke kontainer pertama yang muat, *copy()* untuk membuat state baru yang identik

dengan state saat menjalankan fungsi ini,, dan *generate_random_state()* sebagai cara kedua inisialisasi state awal dengan memasukkan barang ke kontainer benar-benar secara random (jadi bisa saja barang dimasukkan ke kontainer walaupun kontainer tersebut sudah penuh).

2.2. Objective Function

```
def count_penalty(self):
    penalty = 0
    count = 0
    for container in self.list_container:
        count += 1
        total_size = sum(item['ukuran'] for item in container.item_list)
        if total_size > container.capacity:
            penalty += (total_size - container.capacity) * 50
        else:
            penalty += (container.capacity - total_size) * 0.5
    penalty += count * 100
    return penalty
```

Objective function dipilih untuk mengukur kualitas dari hasil packaging barang ke kontainer. Semakin kecil poin maka semakin bagus. Hal-hal yang menjadi pertimbangan adalah sebagai berikut:

- Kapasitas Berlebih (*Over-Capacity*), memiliki penalti yang sangat besar yaitu 50 poin per ukuran yang lebih dari kapasitas kontainer.
- Jumlah Kontainer, sesuai dengan tujuan utama program ini jadi per kontainer akan menambahkan 100 poin.
- Kepadatan Kontainer, sisa dari kapasitas kontainer yang tidak terisi akan menambahkan 0.5 poin per ukuran. Jadi parameter ini tidak terlalu signifikan tetapi tetap dipertimbangkan.

2.3. Implementasi Local Search

2.3.1. Implementasi Hill Climbing

Algoritma *Hill Climbing* adalah metode pencarian lokal yang secara iteratif bergerak menuju solusi yang lebih baik dalam ruang pencarian. Pada setiap langkah, algoritma ini memeriksa tetangga-tetangga dari solusi saat ini dan memilih tetangga yang menghasilkan peningkatan atau penurunan nilai penalti paling besar. Proses ini berlanjut hingga tidak ada langkah yang dapat meningkatkan solusi lebih lanjut. Implementasi di bawah ini menyediakan kerangka dasar untuk algoritma *Hill Climbing* yang akan digunakan dan disesuaikan lebih lanjut dalam sub-bagian berikutnya untuk variasi spesifik

```
class HillClimbing:
    def __init__(self, problem_file, algorithm_type="steepest"):
        self.problem_file = problem_file
        self.algorithm_type = algorithm_type
        self.iterations = 0
        self.sideways_moves = 0
        self.restarts = 0

        script_dir = os.path.dirname(os.path.abspath(__file__))
        parent_dir = os.path.dirname(script_dir)
        project_root = os.path.dirname(parent_dir)
        self.problem_path = os.path.join(project_root, 'data',
f'{self.problem_file}.json')
        _, self.capacity = load_problem(self.problem_path)

        self.original_state = State()
        self.original_state.generate_random_state(problem_file)
        self.current_state = self.original_state.copy()
        self.best_state = self.current_state.copy()

        self.values = []

    def generate_successors(self, state):
        successors = []

        for i, container_from in enumerate(state.list_container):
            for item in container_from.item_list:
                for j, _ in enumerate(state.list_container):
                    if i != j:
                        new_state = state.copy()

new_state.list_container[i].item_list.remove(item)
```

```
new_state.list_container[j].item_list.append(item)
                        new_state.list_container = [c for c in
new_state.list_container if c.item_list]
                        successors.append(new_state)

                new_state = state.copy()
                new_state.list_container[i].item_list.remove(item)
                new_container = Container(self.capacity)
                new_container.item_list.append(item)
                new_state.list_container.append(new_container)
                new_state.list_container = [c for c in
new_state.list_container if c.item_list]
                successors.append(new_state)

        for i, container1 in enumerate(state.list_container):
            for j, container2 in enumerate(state.list_container):
                if i < j:
                    for item1 in container1.item_list:
                        for item2 in container2.item_list:
                            new_state = state.copy()

new_state.list_container[i].item_list.remove(item1)

new_state.list_container[j].item_list.remove(item2)


new_state.list_container[i].item_list.append(item2)

new_state.list_container[j].item_list.append(item1)
                            successors.append(new_state)
        return successors
```

Kelas HillClimbing berfungsi untuk mengimplementasikan algoritma *Hill Climbing*. Metode __init__ adalah konstruktor kelas yang menginisialisasi parameter seperti file masalah, jenis algoritma, dan variabel pelacak iterasi. Ini juga memuat data masalah (barang dan kapasitas kontainer) dan membuat *state* awal secara acak (original_state), yang kemudian disalin menjadi current_state (state yang sedang dievaluasi) dan best_state (state terbaik yang ditemukan). Metode generate_successors menghasilkan semua *state* tetangga yang mungkin dari *state* saat ini. Ini dilakukan dengan dua jenis operasi:

- Memindahkan satu barang: Memindahkan satu barang dari satu kontainer ke kontainer lain yang sudah ada, atau ke kontainer baru.
- Menukar dua barang: Menukar posisi dua barang yang berada di dua kontainer berbeda.

Metode ini memastikan bahwa setiap kontainer kosong yang terbentuk setelah pemindahan barang akan dihapus, dan semua *state* tetangga yang dihasilkan ditambahkan ke daftar successors.

2.3.1.1. Implementasi Steepest Ascent Hill Climbing

Algoritma ini mencari solusi terbaik dengan selalu memilih langkah yang menghasilkan peningkatan (penurunan nilai penalti) paling besar dari semua kemungkinan langkah yang tersedia. Proses ini berlanjut hingga tidak ada langkah yang dapat meningkatkan solusi lebih lanjut. Dalam kode yang diberikan, hcSteepest mengulang hingga max_iterations atau sampai tidak ada penerus yang lebih baik ditemukan. best_successor dipilih berdasarkan nilai penalti terendah, dan current_state diperbarui jika ditemukan solusi yang lebih baik.

```
 def hcSteepest(self, max_iterations=1000, save_plot=True):
        self.iterations = 0
        self.values = [self.current_state.count_penalty()]
        print(f"Initial Penalty:
{self.current_state.count_penalty()}")
        while self.iterations < max_iterations:
            self.iterations += 1
            successors =
self.generate_successors(self.current_state)
            if not successors:
                break
            best_successor = min(successors, key=lambda state:
state.count_penalty())
            if best_successor.count_penalty() <
self.current_state.count_penalty():
                self.current_state = best_successor
```

```
self.values.append(self.current_state.count_penalty())
                if self.current_state.count_penalty() <
self.best_state.count_penalty():
                    self.best_state = self.current_state.copy()
            else:
                break
        fig = self.plot_progress("Steepest Ascent Hill Climbing
Progress", save=save_plot)
        return self.current_state, fig
```

2.3.1.2. Implementasi Sideways Move Hill Climbing

Algoritma ini mirip dengan *Steepest Ascent Hill Climbing*, tetapi memiliki kemampuan untuk melakukan "gerakan menyamping" (*sideways moves*). Ini berarti algoritma dapat menerima solusi yang memiliki nilai penalti yang sama dengan solusi saat ini, dengan tujuan untuk keluar dari *local optimal* dan mencari solusi yang lebih baik di area lain. Dalam kode hcSideways, terdapat max_sideways yang membatasi jumlah gerakan menyamping yang diizinkan. Jika best_successor memiliki penalti yang sama dengan current_state, sideways_moves akan bertambah.

```
   def hcSideways(self, max_sideways=10, max_iterations=1000,
save_plot=True):
        self.iterations = 0
        self.sideways_moves = 0
        self.values = [self.current_state.count_penalty()]
        print(f"Initial Penalty:
{self.current_state.count_penalty()}")
        while self.iterations < max_iterations and
self.sideways_moves < max_sideways:
            self.iterations += 1
            successors =
self.generate_successors(self.current_state)
            if not successors:
                break
            best_successor = min(successors, key=lambda state:
state.count_penalty())
            if best_successor.count_penalty() <
self.current_state.count_penalty():
                self.current_state = best_successor
                self.sideways_moves = 0

self.values.append(self.current_state.count_penalty())
                if self.current_state.count_penalty() <
```

```
self.best_state.count_penalty():
                    self.best_state = self.current_state.copy()
            elif best_successor.count_penalty() ==
self.current_state.count_penalty():
                self.current_state = best_successor
                self.sideways_moves += 1

self.values.append(self.current_state.count_penalty())
            else:
                break
        fig = self.plot_progress("Sideways Hill Climbing
Progress", save=save_plot)
        return self.current_state, fig
```

2.3.1.3. Implementasi Stochastic Hill Climbing

Berbeda dengan dua algoritma sebelumnya yang deterministik, *Stochastic Hill Climbing* memilih langkah yang lebih baik secara acak dari semua langkah yang menghasilkan peningkatan. Ini berarti tidak selalu memilih langkah terbaik, tetapi salah satu dari langkah-langkah yang lebih baik. Hal ini membantu algoritma menghindari *local optima* dengan menjelajahi lebih banyak ruang pencarian. Dalam kode hcStochastic, algoritma mengidentifikasi semua improving (penerus dengan penalti lebih rendah) dan kemudian memilih chosen secara random.choice dari daftar tersebut.

```
   def hcStochastic(self, max_iterations=1000, save_plot=True):
        self.iterations = 0
        self.values = [self.current_state.count_penalty()]
        print(f"Initial Penalty:
{self.current_state.count_penalty()}")
        for _ in range(max_iterations):
            self.iterations += 1
            successors =
self.generate_successors(self.current_state)
            if not successors:
                break
            current_penalty =
self.current_state.count_penalty()
            improving = [s for s in successors if
s.count_penalty() < current_penalty]
            if not improving:
                break
            chosen = random.choice(improving)
```

```
            self.current_state = chosen
            new_penalty = self.current_state.count_penalty()
            self.values.append(new_penalty)
            if new_penalty < self.best_state.count_penalty():
                self.best_state = self.current_state.copy()

        fig = self.plot_progress("Stochastic Hill Climbing
Progress", save=save_plot)
        return self.current_state, fig
```

2.3.1.4. Implementasi Random Restart Hill Climbing

Algoritma ini mengatasi masalah *local optima* dengan menjalankan algoritma *Hill Climbing* (dalam kasus ini, *Steepest Ascent Hill Climbing*) berkali-kali dari titik awal yang berbeda secara acak. Solusi terbaik dari semua restart kemudian dipilih sebagai solusi akhir. Ini meningkatkan peluang menemukan solusi global atau yang mendekati global. Kode hcRandomRestart menjalankan hcSteepest sebanyak max_restarts kali, melacak best_overall_penalty dan best_overall_state dari semua percobaan.

```
   def hcRandomRestart(self, max_restarts=10,
max_iterations_per_restart=100, save_plot=True):
        self.restarts = 0
        best_overall_state = None
        best_overall_penalty = float('inf')
        self.iterations_per_restart = []
        per_restart_final_penalties = []

        for _ in range(max_restarts):
            self.restarts += 1
            temp_hc = HillClimbing(self.problem_file,
'steepest')
            _final_state, _ =
temp_hc.hcSteepest(max_iterations=max_iterations_per_restart,
save_plot=False)

self.iterations_per_restart.append(temp_hc.iterations)

            final_penalty = _final_state.count_penalty()
            print(f"Final Penalty: {final_penalty}")
            per_restart_final_penalties.append(final_penalty)
```

```
            if final_penalty < best_overall_penalty:
                best_overall_penalty = final_penalty
                best_overall_state = _final_state.copy()

        self.current_state = best_overall_state
        self.best_state = best_overall_state.copy()
        self.values = per_restart_final_penalties
        self.iterations = sum(self.iterations_per_restart)

        print(f"Total Restarts: {self.restarts}, Iterations per
Restart: {self.iterations_per_restart}")

        fig = self.plot_progress("Random Restart Final Penalty
per Restart", save=save_plot)
        return self.current_state, fig
```

2.3.2. Implementasi Simulated Annealing

```
def cool_down(current_temp: float, cooling_rate: float) -> float:
    """Menurunkan suhu berdasarkan cooling rate"""
    return current_temp * cooling_rate


def generate_neighbor(state: State) -> State:
    """Menghasilkan tetangga dari state saat ini"""
    new_state = state.copy()

    if not new_state.list_container:
        return new_state

    non_empty = [c for c in new_state.list_container if c.item_list]

    if not non_empty:
        return new_state

    move_type = random.choice(['move', 'swap']) if len(non_empty) >= 2
else 'move'

    if move_type == 'move':
        container_from = random.choice(non_empty)
        item_to_move = random.choice(container_from.item_list)

        try:
            container_from.item_list.remove(item_to_move)
        except ValueError:
            return new_state

        if random.random() < 0.12:
            target_container =
Container(new_state.list_container[0].capacity)
            new_state.list_container.append(target_container)
        else:
            target_container = random.choice(new_state.list_container)

        target_container.item_list.append(item_to_move)
```

```
        new_state.list_container = [c for c in
new_state.list_container if c.item_list]

    else:  # swap
        c1, c2 = random.sample(non_empty, 2)
        i1 = random.choice(c1.item_list)
        i2 = random.choice(c2.item_list)

        try:
            idx1 = c1.item_list.index(i1)
            idx2 = c2.item_list.index(i2)
            c1.item_list[idx1], c2.item_list[idx2] =
c2.item_list[idx2], c1.item_list[idx1]
        except ValueError:
            pass

    return new_state


def simulated_annealing(file_path: str, initial_temp: float,
cooling_rate: float):
    """Menjalankan algoritma Simulated Annealing"""
    result = {
        'algorithm': 'Simulated Annealing',
        'initial_score': None,
        'final_score': None,
        'duration': 0.0,
        'iterations': 0,
        'stuck_iterations': 0,
        'objective_history': [],
        'temperature_history': [],
        'acceptance_prob_history': []
    }

    # Inisialisasi state awal
    initial_state = State()
    initial_state.generate_random_state(file_path)

    current_state = initial_state
    current_score = current_state.count_penalty()
    result['initial_score'] = current_score

    best_state = current_state.copy()
    best_score = current_score

    temperature = initial_temp
    stuck_count = 0
    iteration = 0

    print(f"Memulai SA. Skor Awal: {current_score:.2f}, Suhu Awal:
{temperature:.2f}")

    start_time = time.time()

    # Loop utama SA
    while temperature > 1e-3:
        iteration += 1

        # Generate neighbor
        neighbor_state = generate_neighbor(current_state)
```

```
        neighbor_score = neighbor_state.count_penalty()

        delta_score = neighbor_score - current_score
        acceptance_prob = 0.0

        # Evaluasi penerimaan neighbor
        if delta_score < 0:
            # Neighbor lebih baik, terima
            current_state = neighbor_state
            current_score = neighbor_score
            acceptance_prob = 1.0
            stuck_count = 0
        else:
            # Neighbor lebih buruk, terima dengan probabilitas
tertentu
            acceptance_prob = math.exp(-delta_score / temperature)
            if random.random() < acceptance_prob:
                current_state = neighbor_state
                current_score = neighbor_score
                stuck_count = 0
            else:
                stuck_count += 1

        # Update best state
        if current_score < best_score:
            best_state = current_state.copy()
            best_score = current_score

        # Simpan history
        result['objective_history'].append(best_score)
        result['temperature_history'].append(temperature)
        result['acceptance_prob_history'].append(acceptance_prob)

        # Turunkan suhu
        temperature = cool_down(temperature, cooling_rate)

        # Print progress
        if iteration % 1000 == 0:
            print(f"Iter: {iteration:5d} | Suhu: {temperature:8.4f} |
"
                  f"Current: {current_score:7.2f} | Best:
{best_score:7.2f} | "
                  f"AccProb: {acceptance_prob:.4f}")

    result['stuck_iterations'] = stuck_count
    result['duration'] = time.time() - start_time
    result['final_score'] = best_score
    result['iterations'] = iteration

    print(f"\nPencarian Selesai. Suhu terlalu rendah.")
    print(f"Skor terbaik: {best_score:.2f} setelah {iteration}
iterasi.")
    print(f"Durasi: {result['duration']:.4f} detik")

    return best_state, result


class SimulatedAnnealing:
    @staticmethod
    def print_solution(state: State, title: str):
```

```
        """Mencetak detail solusi"""
        print(f"\n--- {title} ---")
        print(f"Total Kontainer Digunakan:
{len(state.list_container)}")
        print(f"Nilai Objektif (Penalti): {state.count_penalty()}")
        print("-" * 20)
        for i, container in enumerate(state.list_container):
            total_size = container.total_size()
            capacity = container.capacity
            print(f"Kontainer {i + 1} (Total:
{total_size}/{capacity}):")
            if total_size > capacity:
                print(f"  !!! OVER CAPACITY sebanyak {total_size -
capacity} !!!")
            for item in container.item_list:
                print(f"  {item['id']} ({item['ukuran']})")
```

Algoritma Simulated Annealing (SA) berfokus pada pencarian solusi terbaik melalui proses penurunan suhu secara bertahap untuk meniru pendinginan logam. Proses diawali dengan pembangkitan solusi awal secara acak menggunakan fungsi generate_random_state(). Pada setiap iterasi, algoritma menghasilkan solusi tetangga menggunakan fungsi _neighbor(), yang melakukan pemindahan atau pertukaran item antar kontainer. Nilai objektif solusi baru dihitung menggunakan count_penalty(), dan perbedaannya dibandingkan dengan solusi saat ini (ΔE). Jika solusi baru lebih baik, maka langsung diterima; jika lebih buruk, tetap dapat diterima dengan probabilitas tertentu yang bergantung pada suhu dan nilai ΔE. Suhu kemudian diturunkan secara bertahap menggunakan cooling_rate, dan solusi terbaik yang ditemukan selama proses disimpan. Seluruh proses iterasi dijalankan di dalam fungsi run(), hingga mencapai batas iterasi maksimum atau suhu mendekati nol.

2.3.3. Implementasi Genetic Algorithm

```
class GeneticAlgorithm:
    def __init__(self, problem_file, population_size, mutation_rate,
crossover_rate, generations):
        self.problem_file = problem_file
        self.population_size = population_size
        self.mutation_rate = mutation_rate
```

```
        self.crossover_rate = crossover_rate
        self.generations = generations

        script_dir = os.path.dirname(os.path.abspath(__file__))
        parent_dir = os.path.dirname(script_dir)
        project_root = os.path.dirname(parent_dir)
        self.problem_path = os.path.join(project_root, 'data',
f'{self.problem_file}.json')
        self.all_items, self.capacity =
load_problem(self.problem_path)
        self.population = []
        self.history = {'best_objective': []}

    def initialize_population(self):
        self.population = []
        for _ in range(self.population_size):
            state = State()
            state.generate_random_state(self.problem_file)
            self.population.append(state)

    def selection(self, k=3):
        tournament_contenders =
random.sample(self.population_with_penalty, k)
        tournament_contenders.sort(key=lambda x: x[1])
        return tournament_contenders[0][0]

    def crossover(self, parent1, parent2):
        child = State()
        p1_containers = parent1.list_container
        p2_containers = parent2.list_container
        cut1 = random.randint(0, len(p1_containers))
        cut2 = random.randint(0, len(p2_containers))

        child_containers = []
        for c in p1_containers[:cut1]:
            new_c = Container(c.capacity)
            new_c.item_list = c.item_list.copy()
            child_containers.append(new_c)

        for c in p2_containers[cut2:]:
            new_c = Container(c.capacity)
            new_c.item_list = c.item_list.copy()
            child_containers.append(new_c)

        child.list_container = child_containers
        self.repair(child)
        return child

    def repair(self, state):
        items_in_state = set()

        for container in state.list_container:
            for item in container.item_list[:]:
                if item['id'] in items_in_state:
                    container.item_list.remove(item)
                else:
                    items_in_state.add(item['id'])

        all_item_ids = set(item['id'] for item in self.all_items)
        missing_item_ids = all_item_ids - items_in_state
```

```
        if missing_item_ids:
            for item_id in missing_item_ids:
                item_to_add = next(item for item in self.all_items if
item['id'] == item_id)
                placed = False
                for container in state.list_container:
                    if container.total_size() + item_to_add['ukuran']
<= container.capacity:
                        container.item_list.append(item_to_add)
                        placed = True
                        break
                if not placed:
                    new_container = Container(self.capacity)
                    new_container.item_list.append(item_to_add)
                    state.list_container.append(new_container)

        state.list_container = [c for c in state.list_container if
c.item_list]

    def mutate(self, state):
        if random.random() > self.mutation_rate:
            return

        move_type = random.randint(1, 2)

        if not state.list_container:
            return

        if move_type == 1 or len(state.list_container) < 2:
            container_from = random.choice([c for c in
state.list_container if c.item_list] or state.list_container)
            if not container_from.item_list:
                return

            item_to_move = random.choice(container_from.item_list)
            container_from.item_list.remove(item_to_move)

            if random.random() < 0.1:
                target_container = Container(self.capacity)
                state.list_container.append(target_container)
            else:
                target_container = random.choice(state.list_container)
            target_container.item_list.append(item_to_move)

            if not container_from.item_list:
                state.list_container.remove(container_from)
        else:
            non_empty_containers = [c for c in state.list_container if
c.item_list]
            if len(non_empty_containers) < 2:
                return
            c1, c2 = random.sample(non_empty_containers, 2)
            item1 = random.choice(c1.item_list)
            item2 = random.choice(c2.item_list)
            c1.item_list.remove(item1)
            c2.item_list.remove(item2)
            c1.item_list.append(item2)
            c2.item_list.append(item1)
```

```
    def run(self):
        start_time = time.time()
        self.initialize_population()
        initial_best_state = min(self.population, key=lambda s:
s.count_penalty()).copy()
        best_state_overall = initial_best_state
        best_penalty_overall = best_state_overall.count_penalty()

        for gen in range(self.generations):
            self.population_with_penalty = []
            for state in self.population:
                penalty = state.count_penalty()
                self.population_with_penalty.append((state, penalty))
                if penalty < best_penalty_overall:
                    best_penalty_overall = penalty
                    best_state_overall = state.copy()

            best_objective = best_state_overall.count_penalty()
            self.history['best_objective'].append(best_objective)
            new_population = []
            new_population.append(best_state_overall.copy())

            while len(new_population) < self.population_size:
                parent1 = self.selection()
                parent2 = self.selection()
                if random.random() < self.crossover_rate:
                    child = self.crossover(parent1, parent2)
                else:
                    child = parent1.copy()
                self.mutate(child)
                new_population.append(child)
            self.population = new_population

        end_time = time.time()
        duration = end_time - start_time
        return initial_best_state, best_state_overall, self.history,
duration
```

Algoritma Genetic Algorithm berfokus pada pencarian solusi terbaik melalui proses evolusi populasi secara bertahap. Proses diawali dengan pembangkitan populasi awal menggunakan fungsi *initialize_population()*, yang menghasilkan sejumlah individu berupa jadwal berbeda. Setiap individu dievaluasi menggunakan fungsi objektif untuk menentukan kualitasnya. Pada setiap generasi, dua individu dipilih dari populasi menggunakan metode seleksi, seperti turnamen, melalui fungsi *selection()*. Kedua individu tersebut dikombinasikan menggunakan fungsi *crossover()* untuk menghasilkan anak yang mewarisi karakteristik dari keduanya. Anak kemudian

dimodifikasi secara acak dengan probabilitas tertentu menggunakan fungsi *mutate()*. Setelah itu, nilai objektif anak dihitung kembali menggunakan *count_penalty()*, dan seluruh anak yang dihasilkan akan membentuk populasi baru. Individu dengan nilai objektif terbaik dari populasi baru akan dibandingkan dengan solusi terbaik sebelumnya, jika lebih baik, maka akan disimpan sebagai solusi terbaik sementara. Proses iterasi berlanjut hingga jumlah generasi yang ditentukan tercapai (semua ini dijalankan dalam fungsi *run()*).

2.4. Hasil Implementasi

Untuk implementasi ini kami akan menggunakan file *problem1.json* untuk seluruh algoritma. Kami juga menggunakan *generate random state* untuk inisiasi dibanding algoritma heuristik seperti *First Fit* untuk lebih mendemonstrasikan bahwa sejelek-jeleknya state awal, algoritma ini dapat menemukan solusi yang mendekati optima.

Problem1.json

```
{
  "kapasitas_kontainer": 150,
  "barang": [
    { "id": "BRG001", "ukuran": 50 },
    { "id": "BRG002", "ukuran": 60 },
    { "id": "BRG003", "ukuran": 40 },
    { "id": "BRG004", "ukuran": 70 },
    { "id": "BRG005", "ukuran": 35 },
    { "id": "BRG006", "ukuran": 55 },
    { "id": "BRG007", "ukuran": 45 },
    { "id": "BRG008", "ukuran": 30 },
    { "id": "BRG009", "ukuran": 65 },
    { "id": "BRG010", "ukuran": 25 },
    { "id": "BRG011", "ukuran": 80 },
    { "id": "BRG012", "ukuran": 20 },
    { "id": "BRG013", "ukuran": 75 },
    { "id": "BRG014", "ukuran": 42 },
    { "id": "BRG015", "ukuran": 58 }
  ]
}
```

2.4.1. Hasil Steepest Ascent Hill Climbing

Eksperimen ini bertujuan untuk mengevaluasi kinerja algoritma *Steepest Ascent Hill Climbing* dalam menyelesaikan *Bin Packing Problem*. Algoritma dijalankan sebanyak 3 kali, dengan setiap percobaan mencari solusi optimal dari *state* awal yang berbeda.

Tabel 2.1 Hasil Hill Climbing Steepest Ascent

Hill Climbing Steepest Ascent
Eksperimen 1-3

`=== HILL CLIMBING EXPERIMENTS - STEEPEST ===`
 	
`--- Running Experiment 1/3 for STEEPEST ---`
 	
`--- Initial State - Eksperimen 1 ---`
`Total Kontainer Digunakan: 11`
`Nilai Objektif (Penalti): 8771.5`
`Jumlah Iterasi: 0`
`Jumlah Sideways Moves: 0`
`Jumlah Restarts: 0`
`--------------------`
`Kontainer 1 (Total: 30/150):`
`  BRG008 (30)`
`Kontainer 2 (Total: 205/150):`
`  !!! OVER CAPACITY by 55 !!!`
`  BRG013 (75)`
`  BRG002 (60)`
`  BRG004 (70)`
`Kontainer 3 (Total: 0/150):`
`Kontainer 4 (Total: 0/150):`
`Kontainer 5 (Total: 95/150):`
`  BRG006 (55)`
`  BRG003 (40)`
`Kontainer 6 (Total: 42/150):`
`  BRG014 (42)`
`Kontainer 7 (Total: 0/150):`
`Kontainer 8 (Total: 20/150):`
`  BRG012 (20)`
`Kontainer 9 (Total: 120/150):`
`  BRG010 (25)`
`  BRG007 (45)`
`  BRG001 (50)`
`Kontainer 10 (Total: 238/150):`
`  !!! OVER CAPACITY by 88 !!!`
`  BRG011 (80)`
`  BRG005 (35)`

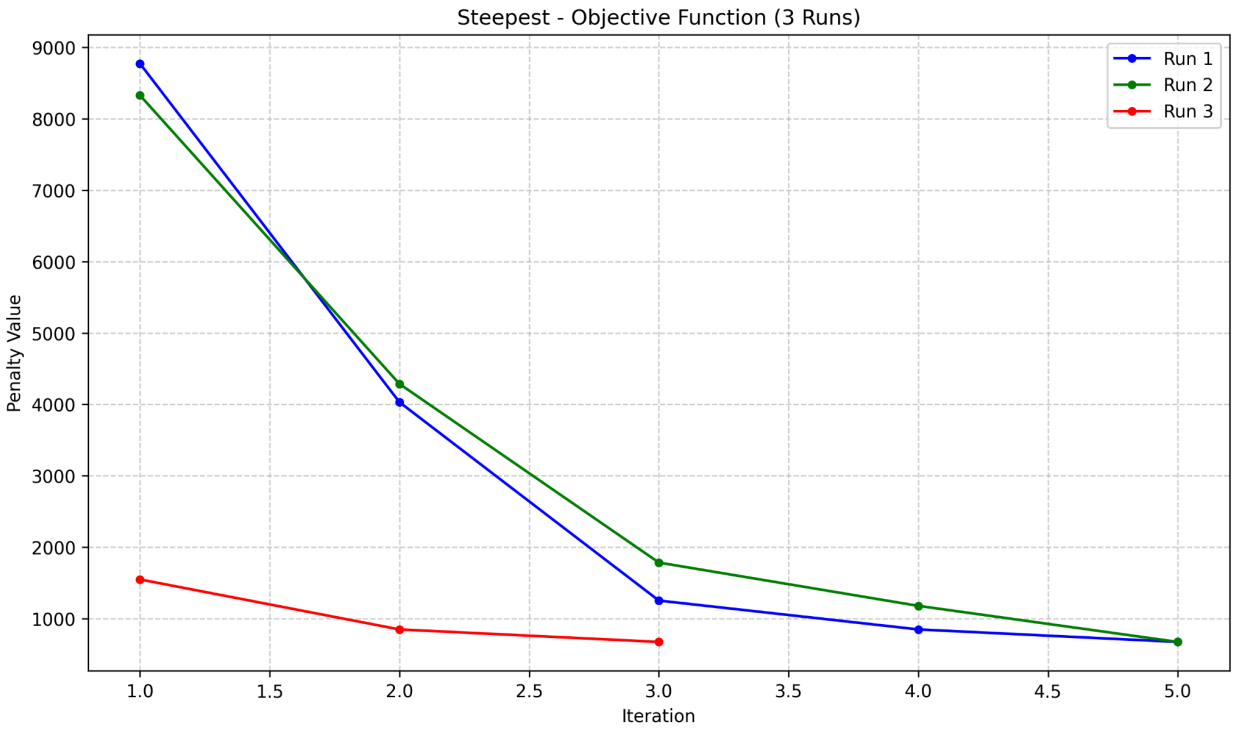
```
  BRG009 (65)
  BRG015 (58)
Kontainer 11 (Total: 0/150):
Initial Penalty: 8771.5

--- Final State - Eksperimen 1 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 5
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 145/150):
  BRG008 (30)
  BRG011 (80)
  BRG005 (35)
Kontainer 2 (Total: 150/150):
  BRG002 (60)
  BRG004 (70)
  BRG012 (20)
Kontainer 3 (Total: 95/150):
  BRG006 (55)
  BRG003 (40)
Kontainer 4 (Total: 117/150):
  BRG014 (42)
  BRG013 (75)
Kontainer 5 (Total: 120/150):
  BRG010 (25)
  BRG007 (45)
  BRG001 (50)
Kontainer 6 (Total: 123/150):
  BRG009 (65)
  BRG015 (58)
Durasi: 0.1551 detik

--- Running Experiment 2/3 for STEEPEST ---

--- Initial State - Eksperimen 2 ---
Total Kontainer Digunakan: 5
Nilai Objektif (Penalti): 8327.5
Jumlah Iterasi: 0
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 45/150):
  BRG007 (45)
Kontainer 2 (Total: 283/150):
  !!! OVER CAPACITY by 133 !!!
  BRG015 (58)
  BRG006 (55)
  BRG011 (80)
  BRG008 (30)
  BRG002 (60)
Kontainer 3 (Total: 160/150):
```

```
  !!! OVER CAPACITY by 10 !!!
  BRG012 (20)
  BRG013 (75)
  BRG009 (65)
Kontainer 4 (Total: 162/150):
  !!! OVER CAPACITY by 12 !!!
  BRG004 (70)
  BRG014 (42)
  BRG001 (50)
Kontainer 5 (Total: 100/150):
  BRG003 (40)
  BRG010 (25)
  BRG005 (35)
Initial Penalty: 8327.5

--- Final State - Eksperimen 2 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 5
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 145/150):
  BRG007 (45)
  BRG011 (80)
  BRG012 (20)
Kontainer 2 (Total: 145/150):
  BRG006 (55)
  BRG008 (30)
  BRG002 (60)
Kontainer 3 (Total: 140/150):
  BRG013 (75)
  BRG009 (65)
Kontainer 4 (Total: 92/150):
  BRG014 (42)
  BRG001 (50)
Kontainer 5 (Total: 100/150):
  BRG003 (40)
  BRG010 (25)
  BRG005 (35)
Kontainer 6 (Total: 128/150):
  BRG015 (58)
  BRG004 (70)
Durasi: 0.1215 detik

--- Running Experiment 3/3 for STEEPEST ---

--- Initial State - Eksperimen 3 ---
Total Kontainer Digunakan: 11
Nilai Objektif (Penalti): 1550.0
Jumlah Iterasi: 0
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
```

```
Kontainer 1 (Total: 0/150):
Kontainer 2 (Total: 45/150):
  BRG012 (20)
  BRG010 (25)
Kontainer 3 (Total: 125/150):
  BRG013 (75)
  BRG001 (50)
Kontainer 4 (Total: 95/150):
  BRG006 (55)
  BRG003 (40)
Kontainer 5 (Total: 115/150):
  BRG004 (70)
  BRG007 (45)
Kontainer 6 (Total: 0/150):
Kontainer 7 (Total: 88/150):
  BRG008 (30)
  BRG015 (58)
Kontainer 8 (Total: 137/150):
  BRG002 (60)
  BRG005 (35)
  BRG014 (42)
Kontainer 9 (Total: 145/150):
  BRG011 (80)
  BRG009 (65)
Kontainer 10 (Total: 0/150):
Kontainer 11 (Total: 0/150):
Initial Penalty: 1550.0

--- Final State - Eksperimen 3 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 3
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 145/150):
  BRG013 (75)
  BRG001 (50)
  BRG012 (20)
Kontainer 2 (Total: 120/150):
  BRG006 (55)
  BRG003 (40)
  BRG010 (25)
Kontainer 3 (Total: 115/150):
  BRG004 (70)
  BRG007 (45)
Kontainer 4 (Total: 88/150):
  BRG008 (30)
  BRG015 (58)
Kontainer 5 (Total: 137/150):
  BRG002 (60)
  BRG005 (35)
  BRG014 (42)
Kontainer 6 (Total: 145/150):
```

```
  BRG011 (80)
  BRG009 (65)
Durasi: 0.1186 detik
```

- **Eksperimen 1:**
    - Durasi Proses Pencarian: 0.1551 detik
    - Nilai Objective Function Awal: 8771.5
    - Nilai Objective Function Akhir: 675.0
    - Jumlah Iterasi: 5
    - Total Kontainer Digunakan: 6
- **Eksperimen 2:**
    - Durasi Proses Pencarian: 0.1215 detik
    - Nilai Objective Function Awal: 8327.5
    - Nilai Objective Function Akhir: 675.0
    - Jumlah Iterasi: 5
    - Total Kontainer Digunakan: 6
- **Eksperimen 3:**
    - Durasi Proses Pencarian: 0.1186 detik
    - Nilai Objective Function Awal: 1550.0
    - Nilai Objective Function Akhir: 675.0
    - Jumlah Iterasi: 3
    - Total Kontainer Digunakan: 6



Gambar 2.1 Plot Hasil Algoritma Steepest Ascent Hill Climbing

Plot yang dihasilkan untuk *Steepest Ascent Hill Climbing* menunjukkan bagaimana nilai fungsi objektif (penalti) menurun secara monoton seiring dengan bertambahnya iterasi. Hal ini mengindikasikan bahwa algoritma secara konsisten bergerak menuju *state* yang lebih baik.

Dari ketiga eksperimen, *Steepest Ascent Hill Climbing* berhasil secara signifikan mengurangi nilai penalti dari *state* awal yang tinggi menjadi nilai yang jauh lebih rendah, yaitu 675.0, dengan 6 kontainer di semua percobaan. Ini menunjukkan efisiensi algoritma dalam menemukan *local optima* yang konsisten. Durasi proses pencarian juga sangat cepat. Pada *state* awal Eksperimen 1 dan 2, terdapat kontainer yang *over capacity*, namun pada *state* akhir, semua kontainer telah diatur dengan kapasitas yang sesuai.

2.4.2. Hasil Sideways Move Hill Climbing

Eksperimen ini menguji algoritma *Sideways Move Hill Climbing*, yang memungkinkan pergerakan ke *state* dengan nilai penalti yang sama untuk mencoba keluar dari *local optima*. Algoritma dijalankan sebanyak 3 kali.

Tabel 2.2 Hasil Hill Climbing Sideways Move

Hill Climbing Sideways Move
Eksperimen 1-3

`=== HILL CLIMBING EXPERIMENTS - SIDEWAYS ===`
 	
`--- Running Experiment 1/3 for SIDEWAYS ---`
 	
`--- Initial State - Eksperimen 1 ---`
`Total Kontainer Digunakan: 7`
`Nilai Objektif (Penalti): 2213.5`
`Jumlah Iterasi: 0`
`Jumlah Sideways Moves: 0`
`Jumlah Restarts: 0`
`--------------------`
`Kontainer 1 (Total: 172/150):`
`  !!! OVER CAPACITY by 22 !!!`
`  BRG014 (42)`
`  BRG002 (60)`
`  BRG004 (70)`
`Kontainer 2 (Total: 118/150):`
`  BRG015 (58)`
`  BRG010 (25)`
`  BRG005 (35)`
`Kontainer 3 (Total: 155/150):`
`  !!! OVER CAPACITY by 5 !!!`
`  BRG007 (45)`
`  BRG008 (30)`
`  BRG011 (80)`
`Kontainer 4 (Total: 55/150):`
`  BRG006 (55)`
`Kontainer 5 (Total: 50/150):`
`  BRG001 (50)`
`Kontainer 6 (Total: 60/150):`
`  BRG003 (40)`
`  BRG012 (20)`
`Kontainer 7 (Total: 140/150):`
`  BRG009 (65)`
`  BRG013 (75)`
`Initial Penalty: 2213.5`

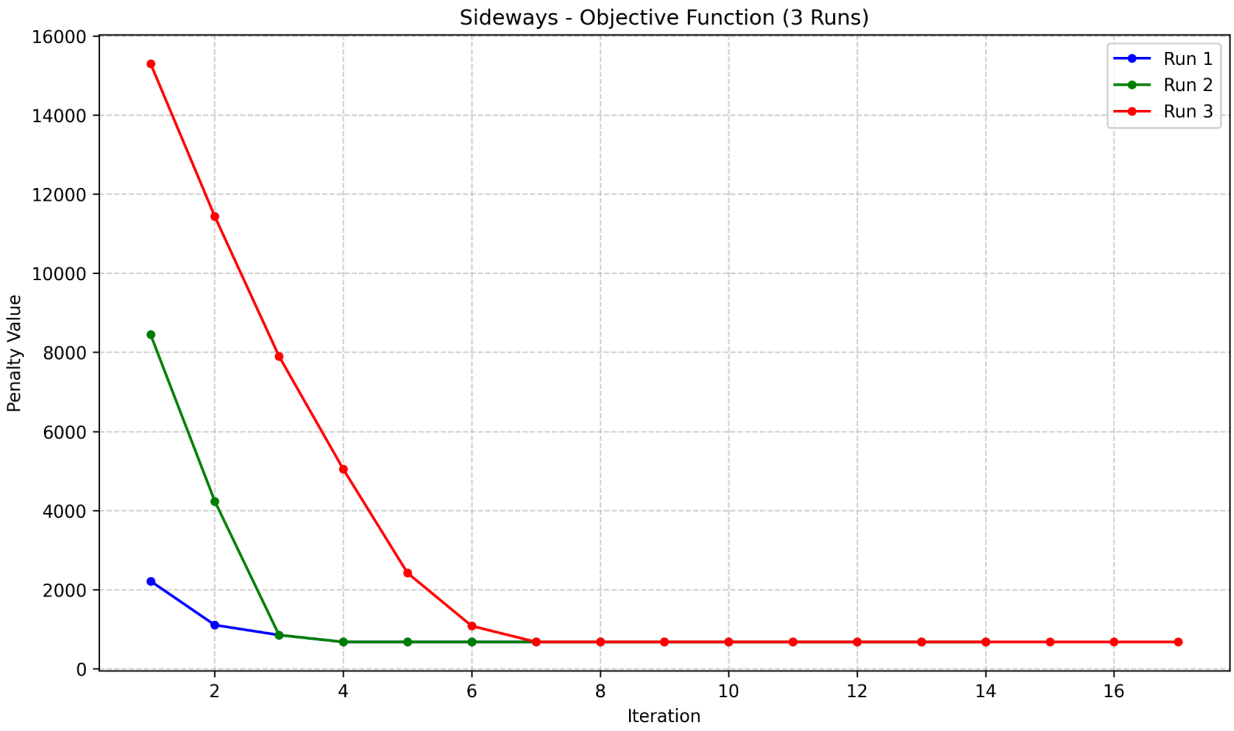
```
--- Final State - Eksperimen 1 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 13
Jumlah Sideways Moves: 10
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 87/150):
  BRG014 (42)
  BRG007 (45)
Kontainer 2 (Total: 145/150):
  BRG003 (40)
  BRG012 (20)
  BRG001 (50)
  BRG005 (35)
Kontainer 3 (Total: 140/150):
  BRG009 (65)
  BRG013 (75)
Kontainer 4 (Total: 128/150):
  BRG004 (70)
  BRG015 (58)
Kontainer 5 (Total: 115/150):
  BRG002 (60)
  BRG008 (30)
  BRG010 (25)
Kontainer 6 (Total: 135/150):
  BRG011 (80)
  BRG006 (55)
Durasi: 0.1237 detik

--- Running Experiment 2/3 for SIDEWAYS ---

--- Initial State - Eksperimen 2 ---
Total Kontainer Digunakan: 8
Nilai Objektif (Penalti): 8448.5
Jumlah Iterasi: 0
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 25/150):
  BRG010 (25)
Kontainer 2 (Total: 60/150):
  BRG002 (60)
Kontainer 3 (Total: 30/150):
  BRG008 (30)
Kontainer 4 (Total: 55/150):
  BRG006 (55)
Kontainer 5 (Total: 148/150):
  BRG003 (40)
  BRG015 (58)
  BRG001 (50)
Kontainer 6 (Total: 0/150):
Kontainer 7 (Total: 297/150):
```

```
  !!! OVER CAPACITY by 147 !!!
  BRG014 (42)
  BRG013 (75)
  BRG012 (20)
  BRG007 (45)
  BRG011 (80)
  BRG005 (35)
Kontainer 8 (Total: 135/150):
  BRG004 (70)
  BRG009 (65)
Initial Penalty: 8448.5

--- Final State - Eksperimen 2 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 13
Jumlah Sideways Moves: 10
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 98/150):
  BRG003 (40)
  BRG015 (58)
Kontainer 2 (Total: 105/150):
  BRG001 (50)
  BRG010 (25)
  BRG008 (30)
Kontainer 3 (Total: 142/150):
  BRG014 (42)
  BRG012 (20)
  BRG007 (45)
  BRG005 (35)
Kontainer 4 (Total: 135/150):
  BRG004 (70)
  BRG009 (65)
Kontainer 5 (Total: 140/150):
  BRG011 (80)
  BRG002 (60)
Kontainer 6 (Total: 130/150):
  BRG013 (75)
  BRG006 (55)
Durasi: 0.1220 detik

--- Running Experiment 3/3 for SIDEWAYS ---

--- Initial State - Eksperimen 3 ---
Total Kontainer Digunakan: 3
Nilai Objektif (Penalti): 15300
Jumlah Iterasi: 0
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 288/150):
  !!! OVER CAPACITY by 138 !!!
  BRG004 (70)
```

```
  BRG007 (45)
  BRG006 (55)
  BRG002 (60)
  BRG015 (58)
Kontainer 2 (Total: 282/150):
  !!! OVER CAPACITY by 132 !!!
  BRG011 (80)
  BRG014 (42)
  BRG005 (35)
  BRG013 (75)
  BRG001 (50)
Kontainer 3 (Total: 180/150):
  !!! OVER CAPACITY by 30 !!!
  BRG003 (40)
  BRG008 (30)
  BRG012 (20)
  BRG010 (25)
  BRG009 (65)
Initial Penalty: 15300

--- Final State - Eksperimen 3 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 16
Jumlah Sideways Moves: 10
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 70/150):
  BRG004 (70)
Kontainer 2 (Total: 135/150):
  BRG002 (60)
  BRG013 (75)
Kontainer 3 (Total: 140/150):
  BRG003 (40)
  BRG007 (45)
  BRG006 (55)
Kontainer 4 (Total: 135/150):
  BRG015 (58)
  BRG014 (42)
  BRG005 (35)
Kontainer 5 (Total: 125/150):
  BRG001 (50)
  BRG008 (30)
  BRG012 (20)
  BRG010 (25)
Kontainer 6 (Total: 145/150):
  BRG009 (65)
  BRG011 (80)
Durasi: 0.1252 detik
```

- **Eksperimen 1:**
    - Durasi Proses Pencarian: 0.1237 detik

    - Nilai Objective Function Awal: 2213.5
    - Nilai Objective Function Akhir: 675.0
    - Jumlah Iterasi: 13
    - Jumlah Sideways Moves: 10
    - Total Kontainer Digunakan: 6
- **Eksperimen 2:**
    - Durasi Proses Pencarian: 0.1220 detik
    - Nilai Objective Function Awal: 8448.5
    - Nilai Objective Function Akhir: 675.0
    - Jumlah Iterasi: 13
    - Jumlah Sideways Moves: 10
    - Total Kontainer Digunakan: 6
- **Eksperimen 3:**
    - Durasi Proses Pencarian: 0.1252 detik
    - Nilai Objective Function Awal: 15300
    - Nilai Objective Function Akhir: 675.0
    - Jumlah Iterasi: 16
    - Jumlah Sideways Moves: 10
    - Total Kontainer Digunakan: 6



Gambar 2.2 Plot Hasil Algoritma Sideways Hill Climbing

Plot yang dihasilkan untuk *Sideways Move Hill Climbing* menunjukkan penurunan nilai penalti yang serupa dengan *Steepest Ascent*, namun dengan potensi untuk melakukan *sideways moves* yang dapat dilihat dari jumlah *sideways moves* yang tercatat. Ini menunjukkan upaya algoritma untuk menjelajahi lebih banyak ruang pencarian.

Algoritma *Sideways Move Hill Climbing* secara konsisten mencapai nilai penalti akhir yang sama yaitu 675.0 dengan 6 kontainer di ketiga eksperimen. Ini menunjukkan bahwa kemampuan untuk melakukan *sideways moves* sangat efektif dalam membantu algoritma mencapai *local optima* yang optimal atau bahkan global optima. Meskipun jumlah iterasi sedikit lebih banyak dibandingkan *Steepest Ascent* dalam beberapa percobaan, durasi keseluruhan tetap sangat efisien. Pada *state* awal Eksperimen 1, 2, dan 3, terdapat kontainer yang *over*

capacity, namun pada *state* akhir, semua kontainer telah diatur dengan kapasitas yang sesuai.

2.4.3. Hasil Stochastic Hill Climbing

Eksperimen ini mengevaluasi kinerja algoritma *Stochastic Hill Climbing*, yang memilih langkah perbaikan secara acak dari semua langkah yang menghasilkan peningkatan. Pendekatan ini diharapkan dapat membantu menghindari *local optima.*

Tabel 2.3 Hasil Stochastic Hill Climbing

Stochastic Hill Climbing
Eksperimen 1-3

`=== HILL CLIMBING EXPERIMENTS - STOCHASTIC ===`
 	
`--- Running Experiment 1/3 for STOCHASTIC ---`
 	
`--- Initial State - Eksperimen 1 ---`
`Total Kontainer Digunakan: 9`
`Nilai Objektif (Penalti): 8522.5`
`Jumlah Iterasi: 0`
`Jumlah Sideways Moves: 0`
`Jumlah Restarts: 0`
`--------------------`
`Kontainer 1 (Total: 222/150):`
`  !!! OVER CAPACITY by 72 !!!`
`  BRG004 (70)`
`  BRG009 (65)`
`  BRG007 (45)`
`  BRG014 (42)`
`Kontainer 2 (Total: 110/150):`
`  BRG011 (80)`
`  BRG008 (30)`
`Kontainer 3 (Total: 20/150):`
`  BRG012 (20)`
`Kontainer 4 (Total: 223/150):`
`  !!! OVER CAPACITY by 73 !!!`
`  BRG001 (50)`
`  BRG006 (55)`
`  BRG015 (58)`
`  BRG002 (60)`
`Kontainer 5 (Total: 115/150):`
`  BRG013 (75)`
`  BRG003 (40)`

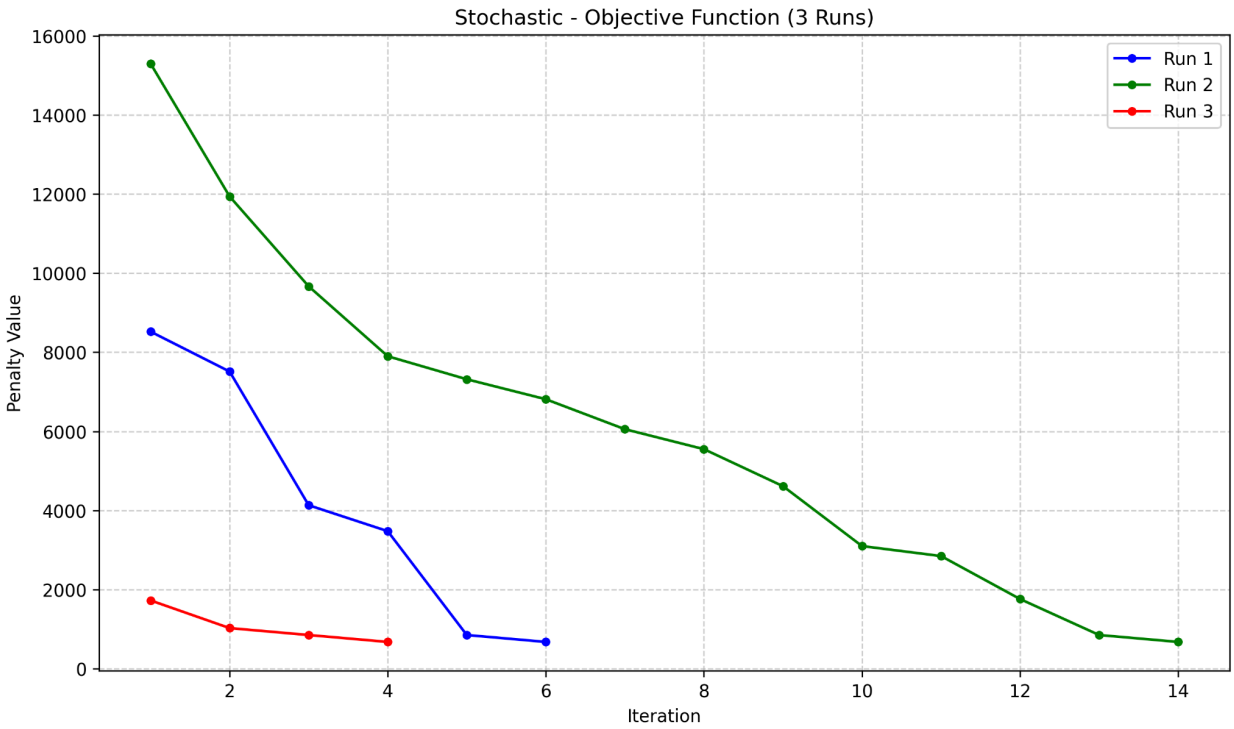
```
Kontainer 6 (Total: 35/150):
  BRG005 (35)
Kontainer 7 (Total: 25/150):
  BRG010 (25)
Kontainer 8 (Total: 0/150):
Kontainer 9 (Total: 0/150):
Initial Penalty: 8522.5

--- Final State - Eksperimen 1 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 6
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 137/150):
  BRG004 (70)
  BRG014 (42)
  BRG010 (25)
Kontainer 2 (Total: 110/150):
  BRG011 (80)
  BRG008 (30)
Kontainer 3 (Total: 135/150):
  BRG012 (20)
  BRG002 (60)
  BRG006 (55)
Kontainer 4 (Total: 143/150):
  BRG001 (50)
  BRG015 (58)
  BRG005 (35)
Kontainer 5 (Total: 115/150):
  BRG013 (75)
  BRG003 (40)
Kontainer 6 (Total: 110/150):
  BRG007 (45)
  BRG009 (65)
Durasi: 0.1175 detik

--- Running Experiment 2/3 for STOCHASTIC ---

--- Initial State - Eksperimen 2 ---
Total Kontainer Digunakan: 3
Nilai Objektif (Penalti): 15300
Jumlah Iterasi: 0
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 300/150):
  !!! OVER CAPACITY by 150 !!!
  BRG013 (75)
  BRG008 (30)
  BRG007 (45)
  BRG004 (70)
  BRG012 (20)
```

```
  BRG002 (60)
Kontainer 2 (Total: 172/150):
  !!! OVER CAPACITY by 22 !!!
  BRG006 (55)
  BRG003 (40)
  BRG005 (35)
  BRG014 (42)
Kontainer 3 (Total: 278/150):
  !!! OVER CAPACITY by 128 !!!
  BRG011 (80)
  BRG015 (58)
  BRG009 (65)
  BRG001 (50)
  BRG010 (25)
Initial Penalty: 15300

--- Final State - Eksperimen 2 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 14
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 150/150):
  BRG012 (20)
  BRG001 (50)
  BRG007 (45)
  BRG005 (35)
Kontainer 2 (Total: 137/150):
  BRG006 (55)
  BRG003 (40)
  BRG014 (42)
Kontainer 3 (Total: 138/150):
  BRG011 (80)
  BRG015 (58)
Kontainer 4 (Total: 130/150):
  BRG004 (70)
  BRG002 (60)
Kontainer 5 (Total: 130/150):
  BRG013 (75)
  BRG008 (30)
  BRG010 (25)
Kontainer 6 (Total: 65/150):
  BRG009 (65)
Durasi: 0.1253 detik

--- Running Experiment 3/3 for STOCHASTIC ---

--- Initial State - Eksperimen 3 ---
Total Kontainer Digunakan: 12
Nilai Objektif (Penalti): 1725.0
Jumlah Iterasi: 0
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
```

```
--------------------
Kontainer 1 (Total: 150/150):
  BRG006 (55)
  BRG001 (50)
  BRG007 (45)
Kontainer 2 (Total: 130/150):
  BRG004 (70)
  BRG002 (60)
Kontainer 3 (Total: 0/150):
Kontainer 4 (Total: 0/150):
Kontainer 5 (Total: 60/150):
  BRG010 (25)
  BRG005 (35)
Kontainer 6 (Total: 75/150):
  BRG013 (75)
Kontainer 7 (Total: 105/150):
  BRG003 (40)
  BRG009 (65)
Kontainer 8 (Total: 62/150):
  BRG012 (20)
  BRG014 (42)
Kontainer 9 (Total: 88/150):
  BRG015 (58)
  BRG008 (30)
Kontainer 10 (Total: 0/150):
Kontainer 11 (Total: 80/150):
  BRG011 (80)
Kontainer 12 (Total: 0/150):
Initial Penalty: 1725.0

--- Final State - Eksperimen 3 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 4
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 150/150):
  BRG006 (55)
  BRG001 (50)
  BRG007 (45)
Kontainer 2 (Total: 130/150):
  BRG004 (70)
  BRG002 (60)
Kontainer 3 (Total: 90/150):
  BRG010 (25)
  BRG005 (35)
  BRG008 (30)
Kontainer 4 (Total: 105/150):
  BRG003 (40)
  BRG009 (65)
Kontainer 5 (Total: 142/150):
  BRG012 (20)
  BRG014 (42)
```

```
  BRG011 (80)
Kontainer 6 (Total: 133/150):
  BRG015 (58)
  BRG013 (75)
Durasi: 0.1206 detik
```

- **Eksperimen 1:**
    - Durasi Proses Pencarian: 0.1175 detik
    - Nilai Objective Function Awal: 8522.5
    - Nilai Objective Function Akhir: 675.0
    - Jumlah Iterasi: 6
    - Total Kontainer Digunakan: 6
- **Eksperimen 2:**
    - Durasi Proses Pencarian: 0.1253 detik
    - Nilai Objective Function Awal: 15300
    - Nilai Objective Function Akhir: 675.0
    - Jumlah Iterasi: 14
    - Total Kontainer Digunakan: 6
- **Eksperimen 3:**
    - Durasi Proses Pencarian: 0.1206 detik
    - Nilai Objective Function Awal: 1725.0
    - Nilai Objective Function Akhir: 675.0
    - Jumlah Iterasi: 4
    - Total Kontainer Digunakan: 6



Gambar 2.3 Plot Hasil Algoritma Stochastic Hill Climbing

Plot yang dihasilkan untuk *Stochastic Hill Climbing* menunjukkan pola penurunan nilai penalti yang mungkin terlihat sedikit lebih fluktuatif dibandingkan *Steepest Ascent* karena pemilihan langkah acak. Namun, secara keseluruhan, tren penurunan tetap terlihat jelas.

Stochastic Hill Climbing berhasil mencapai nilai penalti akhir yang baik (675.0) dengan 6 kontainer di ketiga eksperimen. Durasi proses pencarian sangat cepat, menunjukkan efisiensi. Jumlah iterasi bervariasi, mencerminkan sifat acak dalam pemilihan langkah. Pada *state* awal Eksperimen 1 dan 2, terdapat kontainer yang *over capacity*, namun pada *state* akhir, semua kontainer telah diatur dengan kapasitas yang sesuai.

2.4.4. Hasil Random Restart Hill Climbing

Eksperimen ini menerapkan algoritma *Random Restart Hill Climbing*, yang menjalankan *Steepest Ascent Hill Climbing* dari beberapa titik awal acak untuk meningkatkan peluang menemukan solusi global. Algoritma dijalankan sebanyak 3 kali.

Tabel 2.4 Hasil Random Restart Hill Climbing

Random Restart Hill Climbing
Eksperimen 1-3

`=== HILL CLIMBING EXPERIMENTS - RANDOM_RESTART ===`
 	
`--- Running Experiment 1/3 for RANDOM_RESTART ---`
 	
`--- Initial State - Eksperimen 1 ---`
`Total Kontainer Digunakan: 10`
`Nilai Objektif (Penalti): 2385.0`
`Jumlah Iterasi: 0`
`Jumlah Sideways Moves: 0`
`Jumlah Restarts: 0`
`--------------------`
`Kontainer 1 (Total: 155/150):`
`  !!! OVER CAPACITY by 5 !!!`
`  BRG014 (42)`
`  BRG015 (58)`
`  BRG006 (55)`
`Kontainer 2 (Total: 0/150):`
`Kontainer 3 (Total: 70/150):`
`  BRG004 (70)`
`Kontainer 4 (Total: 75/150):`
`  BRG013 (75)`
`Kontainer 5 (Total: 85/150):`
`  BRG001 (50)`
`  BRG005 (35)`
`Kontainer 6 (Total: 0/150):`
`Kontainer 7 (Total: 165/150):`
`  !!! OVER CAPACITY by 15 !!!`
`  BRG007 (45)`
`  BRG011 (80)`
`  BRG003 (40)`
`Kontainer 8 (Total: 25/150):`
`  BRG010 (25)`
`Kontainer 9 (Total: 95/150):`
`  BRG008 (30)`
`  BRG009 (65)`
`Kontainer 10 (Total: 80/150):`

```
  BRG002 (60)
  BRG012 (20)
Initial Penalty: 1530.0
Final Penalty: 850.0
Initial Penalty: 9085.0
Final Penalty: 675.0
Initial Penalty: 4290.0
Final Penalty: 850.0
Initial Penalty: 30100
Final Penalty: 675.0
Initial Penalty: 15300
Final Penalty: 675.0
Initial Penalty: 4482.5
Final Penalty: 675.0
Initial Penalty: 1725.0
Final Penalty: 675.0
Initial Penalty: 3088.5
Final Penalty: 675.0
Initial Penalty: 22700
Final Penalty: 675.0
Initial Penalty: 8075.0
Final Penalty: 675.0
Total Restarts: 10, Iterations per Restart: [3, 5, 5, 11, 8, 4,
3, 5, 8, 5]

--- Final State - Eksperimen 1 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 57
Jumlah Sideways Moves: 0
Jumlah Restarts: 10
--------------------
Kontainer 1 (Total: 142/150):
  BRG003 (40)
  BRG014 (42)
  BRG002 (60)
Kontainer 2 (Total: 115/150):
  BRG001 (50)
  BRG008 (30)
  BRG005 (35)
Kontainer 3 (Total: 145/150):
  BRG011 (80)
  BRG009 (65)
Kontainer 4 (Total: 145/150):
  BRG004 (70)
  BRG013 (75)
Kontainer 5 (Total: 148/150):
  BRG007 (45)
  BRG012 (20)
  BRG015 (58)
  BRG010 (25)
Kontainer 6 (Total: 55/150):
  BRG006 (55)
Durasi: 1.4389 detik
```

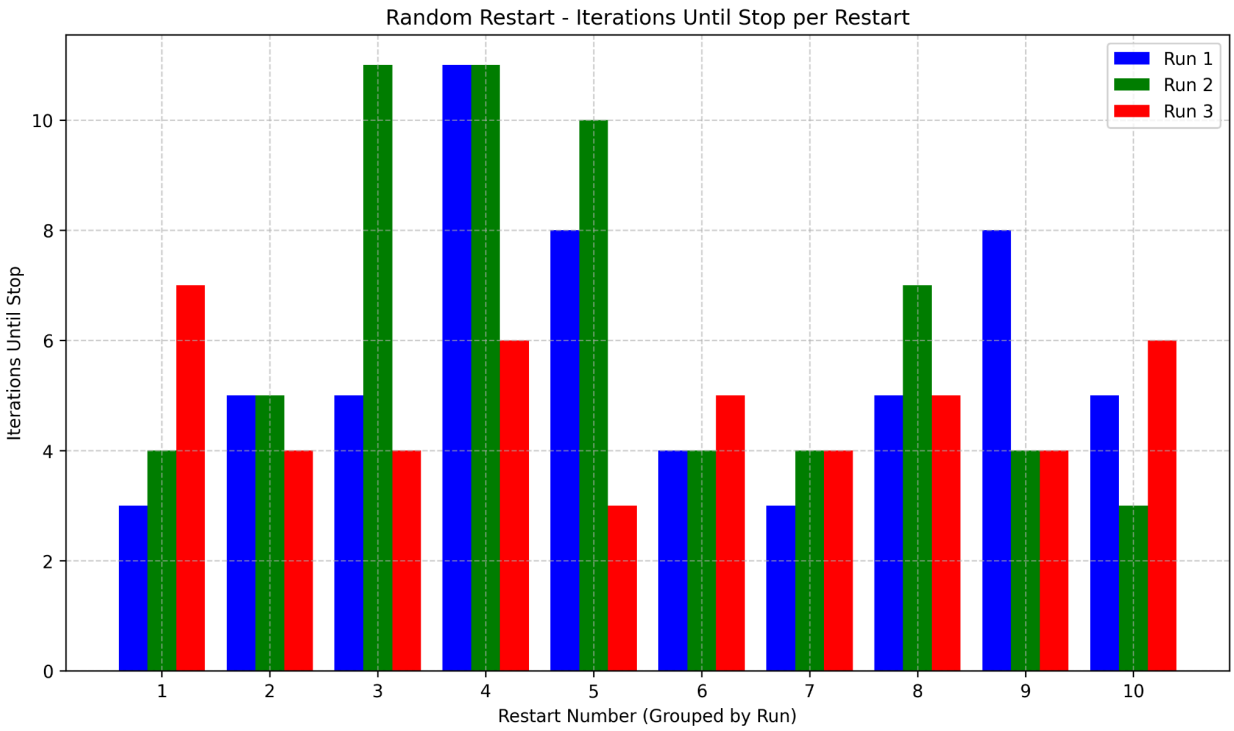
```
--- Running Experiment 2/3 for RANDOM_RESTART ---

--- Initial State - Eksperimen 2 ---
Total Kontainer Digunakan: 9
Nilai Objektif (Penalti): 4634.0
Jumlah Iterasi: 0
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 105/150):
  BRG002 (60)
  BRG007 (45)
Kontainer 2 (Total: 85/150):
  BRG008 (30)
  BRG006 (55)
Kontainer 3 (Total: 50/150):
  BRG001 (50)
Kontainer 4 (Total: 0/150):
Kontainer 5 (Total: 218/150):
  !!! OVER CAPACITY by 68 !!!
  BRG015 (58)
  BRG009 (65)
  BRG010 (25)
  BRG004 (70)
Kontainer 6 (Total: 120/150):
  BRG011 (80)
  BRG003 (40)
Kontainer 7 (Total: 35/150):
  BRG005 (35)
Kontainer 8 (Total: 42/150):
  BRG014 (42)
Kontainer 9 (Total: 95/150):
  BRG013 (75)
  BRG012 (20)
Initial Penalty: 10135.0
Final Penalty: 850.0
Initial Penalty: 1900.0
Final Penalty: 850.0
Initial Penalty: 30275.0
Final Penalty: 675.0
Initial Penalty: 30100
Final Penalty: 675.0
Initial Penalty: 22700
Final Penalty: 675.0
Initial Penalty: 1375.0
Final Penalty: 675.0
Initial Penalty: 1550.0
Final Penalty: 675.0
Initial Penalty: 15300
Final Penalty: 752.5
Initial Penalty: 5395.0
Final Penalty: 675.0
Initial Penalty: 8852.5
```

```
Final Penalty: 675.0
Total Restarts: 10, Iterations per Restart: [4, 5, 11, 11, 10,
4, 4, 7, 4, 3]

--- Final State - Eksperimen 2 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 63
Jumlah Sideways Moves: 0
Jumlah Restarts: 10
--------------------
Kontainer 1 (Total: 150/150):
  BRG012 (20)
  BRG003 (40)
  BRG008 (30)
  BRG005 (35)
  BRG010 (25)
Kontainer 2 (Total: 150/150):
  BRG011 (80)
  BRG004 (70)
Kontainer 3 (Total: 140/150):
  BRG013 (75)
  BRG009 (65)
Kontainer 4 (Total: 118/150):
  BRG002 (60)
  BRG015 (58)
Kontainer 5 (Total: 150/150):
  BRG006 (55)
  BRG001 (50)
  BRG007 (45)
Kontainer 6 (Total: 42/150):
  BRG014 (42)
Durasi: 1.2467 detik

--- Running Experiment 3/3 for RANDOM_RESTART ---

--- Initial State - Eksperimen 3 ---
Total Kontainer Digunakan: 9
Nilai Objektif (Penalti): 2462.5
Jumlah Iterasi: 0
Jumlah Sideways Moves: 0
Jumlah Restarts: 0
--------------------
Kontainer 1 (Total: 77/150):
  BRG014 (42)
  BRG005 (35)
Kontainer 2 (Total: 175/150):
  !!! OVER CAPACITY by 25 !!!
  BRG011 (80)
  BRG008 (30)
  BRG009 (65)
Kontainer 3 (Total: 125/150):
  BRG010 (25)
  BRG003 (40)
```

```
  BRG002 (60)
Kontainer 4 (Total: 133/150):
  BRG013 (75)
  BRG015 (58)
Kontainer 5 (Total: 115/150):
  BRG007 (45)
  BRG004 (70)
Kontainer 6 (Total: 20/150):
  BRG012 (20)
Kontainer 7 (Total: 55/150):
  BRG006 (55)
Kontainer 8 (Total: 50/150):
  BRG001 (50)
Kontainer 9 (Total: 0/150):
Initial Penalty: 16990.0
Final Penalty: 675.0
Initial Penalty: 2563.5
Final Penalty: 675.0
Initial Penalty: 2250.0
Final Penalty: 675.0
Initial Penalty: 2075.0
Final Penalty: 675.0
Initial Penalty: 4560.0
Final Penalty: 675.0
Initial Penalty: 1900.0
Final Penalty: 675.0
Initial Penalty: 3624.0
Final Penalty: 675.0
Initial Penalty: 5391.5
Final Penalty: 675.0
Initial Penalty: 3566.5
Final Penalty: 675.0
Initial Penalty: 14061.0
Final Penalty: 675.0
Total Restarts: 10, Iterations per Restart: [7, 4, 4, 6, 3, 5,
4, 5, 4, 6]

--- Final State - Eksperimen 3 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 675.0
Jumlah Iterasi: 48
Jumlah Sideways Moves: 0
Jumlah Restarts: 10
--------------------
Kontainer 1 (Total: 145/150):
  BRG003 (40)
  BRG005 (35)
  BRG007 (45)
  BRG010 (25)
Kontainer 2 (Total: 135/150):
  BRG006 (55)
  BRG012 (20)
  BRG002 (60)
Kontainer 3 (Total: 150/150):
```

```
  BRG015 (58)
  BRG014 (42)
  BRG001 (50)
Kontainer 4 (Total: 150/150):
  BRG011 (80)
  BRG004 (70)
Kontainer 5 (Total: 140/150):
  BRG013 (75)
  BRG009 (65)
Kontainer 6 (Total: 30/150):
  BRG008 (30)
Durasi: 1.3698 detik
```

- **Eksperimen 1:**
    - Durasi Proses Pencarian: 1.4389 detik
    - Nilai Objective Function Terbaik Akhir: 675.0
    - Jumlah Restarts: 10
    - Iterasi per Restart: [3, 5, 5, 11, 8, 4, 3, 5, 8, 5]
    - Total Kontainer Digunakan: 6
- **Eksperimen 2:**
    - Durasi Proses Pencarian: 1.2467 detik
    - Nilai Objective Function Terbaik Akhir: 675.0
    - Jumlah Restarts: 10
    - Iterasi per Restart: [4, 5, 11, 11, 10, 4, 4, 7, 4, 3]
    - Total Kontainer Digunakan: 6
- **Eksperimen 3:**
    - Durasi Proses Pencarian: 1.3698 detik
    - Nilai Objective Function Terbaik Akhir: 675.0
    - Jumlah Restarts: 10
    - Iterasi per Restart: [7, 4, 4, 6, 3, 5, 4, 5, 4, 6]
    - Total Kontainer Digunakan: 6



Gambar 2.4 Plot Hasil Algoritma Random Restart Hill Climbing

Plot yang dihasilkan untuk *Random Restart Hill Climbing* menampilkan diagram batang yang dikelompokkan. Pada sumbu X, setiap kelompok batang merepresentasikan nomor restart, dan di atasnya terdapat tiga batang, masing-masing mewakili jumlah iterasi yang dibutuhkan hingga berhenti untuk 'Run 1', 'Run 2', dan 'Run 3'. Visualisasi ini memungkinkan perbandingan langsung jumlah iterasi yang diperlukan untuk setiap restart di ketiga *run* eksperimen, menunjukkan konsistensi atau variasi dalam konvergensi algoritma.

Algoritma *Random Restart Hill Climbing* secara konsisten mencapai nilai penalti akhir terbaik yaitu 675.0 dengan 6 kontainer di ketiga eksperimen. Diagram batang yang dikelompokkan dengan jelas menunjukkan jumlah iterasi yang bervariasi untuk setiap restart di setiap *run*. Meskipun durasi total proses pencarian lebih lama

dibandingkan algoritma *Hill Climbing* tunggal, hasil ini menunjukkan peningkatan keandalan dalam menemukan solusi yang optimal. Variasi iterasi antar restart menunjukkan bahwa titik awal acak memang memengaruhi jalur pencarian, namun algoritma ini tetap mampu menemukan solusi optimal yang sama secara konsisten. Pada *state* awal Eksperimen 1, 2, dan 3, terdapat kontainer yang *over capacity*, namun pada *state* akhir, semua kontainer telah diatur dengan kapasitas yang sesuai.

2.4.5. Hasil Simulated Annealing

Tabel 2.5 Tabel gambar plotting Simulated Annealing

Simulated Annealing
Eksperimen 1
========== ========== EKSPERIMEN 1 dari 3 ========== ========== Eksperimen 1 - Masukkan suhu awal (T0): 1000 Eksperimen 1 - Masukkan cooling rate (0-1): 0.99 Memulai SA. Skor Awal: 2080458.00, Suhu Awal: 1000.00 Iter: 1000 | Suhu: 0.0432 | Current: 750.00 | Best: 750.00 | AccProb: 1.0000 Pencarian Selesai. Suhu terlalu rendah. Skor terbaik: 750.00 setelah 1375 iterasi. Durasi: 0.0156 detik

--- HASIL EKSPERIMEN 1 ---

Skor Awal: 2080458.00

Skor Akhir: 750.00

Durasi: 0.0156 detik

Iterasi: 1375

Jumlah Stuck: 1

--- Solusi Akhir Eksperimen 1 ---

Total Kontainer Digunakan: 6

Nilai Objektif (Penalti): 750

Kontainer 1 (Total: 90/150):

  BRG003 (40)

  BRG001 (50)

Kontainer 2 (Total: 150/150):

  BRG006 (55)

  BRG009 (65)

  BRG008 (30)

Kontainer 3 (Total: 137/150):

  BRG002 (60)

  BRG005 (35)

  BRG014 (42)

Kontainer 4 (Total: 145/150):

  BRG007 (45)

  BRG012 (20)

  BRG011 (80)

Kontainer 5 (Total: 128/150):

BRG015 (58)

BRG004 (70)

Kontainer 6 (Total: 100/150):

BRG013 (75)

BRG010 (25)

==========

==========

Eksperimen 2

==========
==========
EKSPERIMEN 2 dari 3
==========
==========
Eksperimen 2 - Masukkan suhu awal (T0): 1000
Eksperimen 2 - Masukkan cooling rate (0-1): 0.99
Memulai SA. Skor Awal: 352785.00, Suhu Awal: 1000.00
Iter: 1000 | Suhu: 0.0432 | Current: 750.00 | Best: 750.00 | AccProb: 0.0000

Pencarian Selesai. Suhu terlalu rendah.
Skor terbaik: 750.00 setelah 1375 iterasi.
Durasi: 0.0156 detik

--- HASIL EKSPERIMEN 2 ---
Skor Awal: 352785.00
Skor Akhir: 750.00
Durasi: 0.0156 detik
Iterasi: 1375
Jumlah Stuck: 2

--- Solusi Akhir Eksperimen 2 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 750

Kontainer 1 (Total: 145/150):
  BRG005 (35)
  BRG001 (50)
  BRG002 (60)
Kontainer 2 (Total: 115/150):
  BRG013 (75)
  BRG003 (40)
Kontainer 3 (Total: 135/150):
  BRG010 (25)
  BRG007 (45)
  BRG009 (65)
Kontainer 4 (Total: 138/150):
  BRG015 (58)
  BRG011 (80)
Kontainer 5 (Total: 132/150):
  BRG012 (20)
  BRG004 (70)
  BRG014 (42)
Kontainer 6 (Total: 85/150):
  BRG006 (55)
  BRG008 (30)

==========
==========

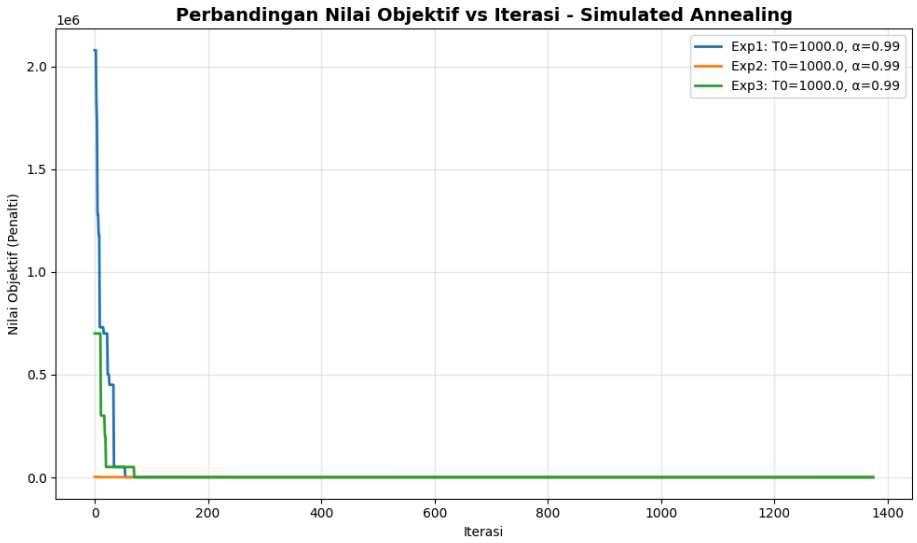
Eksperimen 3

==========
==========
EKSPERIMEN 3 dari 3
==========
==========
Eksperimen 3 - Masukkan suhu awal (T0): 1000
Eksperimen 3 - Masukkan cooling rate (0-1): 0.99
Memulai SA. Skor Awal: 703320.00, Suhu Awal: 1000.00
Iter: 1000 | Suhu: 0.0432 | Current: 750.00 | Best: 750.00 | AccProb: 0.0000

Pencarian Selesai. Suhu terlalu rendah.
Skor terbaik: 750.00 setelah 1375 iterasi.
Durasi: 0.0156 detik

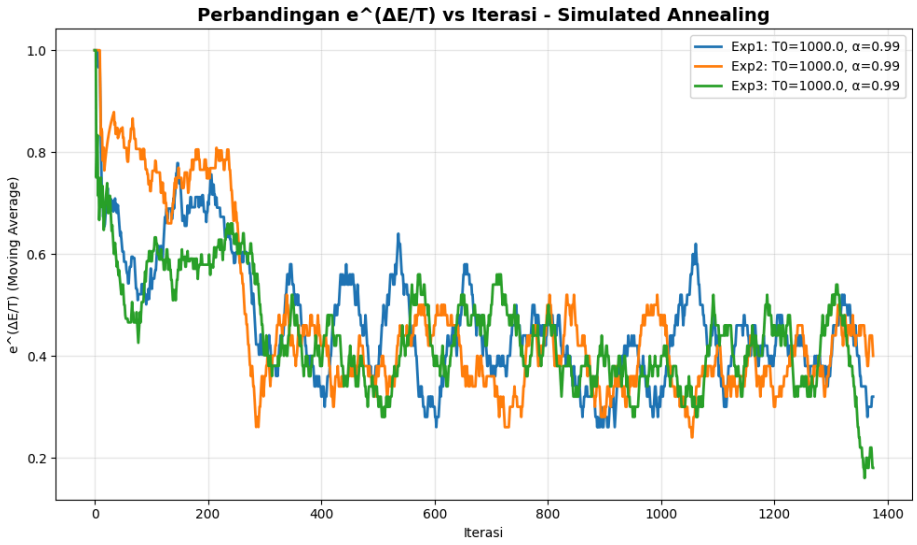
```
--- HASIL EKSPERIMEN 3 ---
Skor Awal: 703320.00
Skor Akhir: 750.00
Durasi: 0.0156 detik
Iterasi: 1375
Jumlah Stuck: 5

--- Solusi Akhir Eksperimen 3 ---
Total Kontainer Digunakan: 6
Nilai Objektif (Penalti): 750
--------------------
Kontainer 1 (Total: 90/150):
  BRG009 (65)
  BRG010 (25)
Kontainer 2 (Total: 147/150):
  BRG014 (42)
  BRG008 (30)
  BRG013 (75)
Kontainer 3 (Total: 133/150):
  BRG005 (35)
  BRG003 (40)
  BRG015 (58)
Kontainer 4 (Total: 125/150):
  BRG012 (20)
  BRG007 (45)
  BRG002 (60)
Kontainer 5 (Total: 105/150):
  BRG006 (55)
  BRG001 (50)
Kontainer 6 (Total: 150/150):
  BRG004 (70)
  BRG011 (80)

================================================
================================
```

Nilai Objektif vs iterasi



Gambar 2.5 Plot objective function dengan iterasi - simulated annealing

Nilai $e^{\frac{\Delta E}{T}}$ vs iterasi



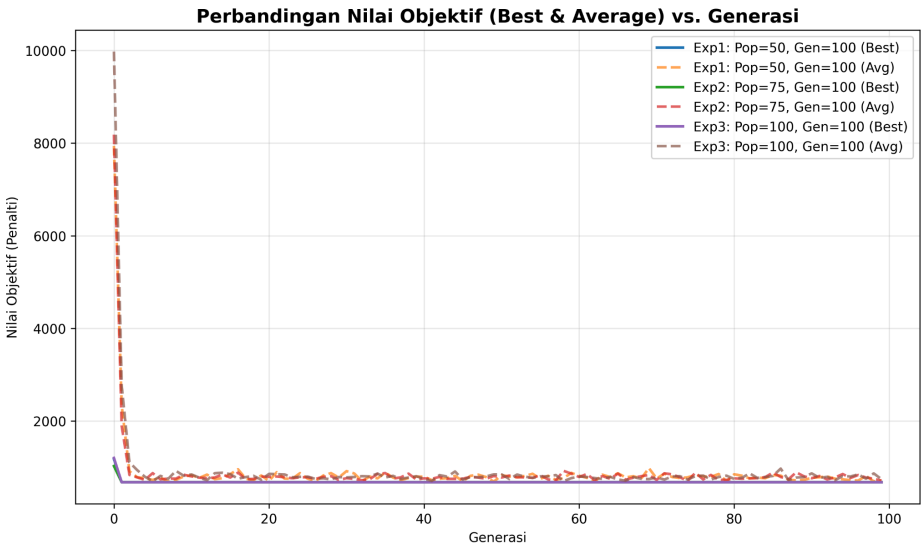
Gambar 2.6 Plot $e^{\frac{\Delta E}{T}}$ dengan iterasi - simulated annealing

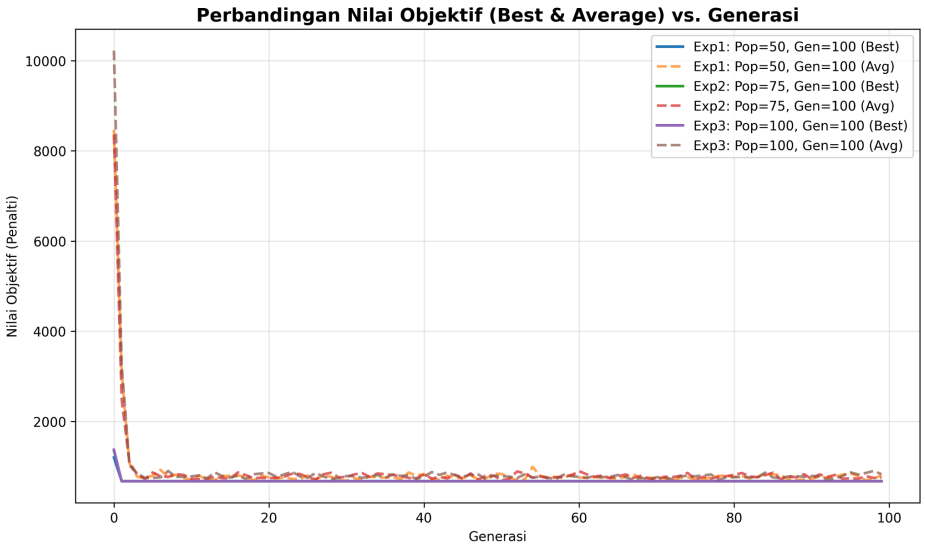
Plot nilai objektif vs iterasi menunjukkan bahwa ketiga eksperimen berhasil mencapai solusi maksimum, dimana nilai objektif adalah 0.

Plot $e^{\frac{\Delta E}{T}}$ menunjukkan fluktuasi. Fluktuasi naik turun akan terjadi ketika algoritma mengambil state yang lebih buruk. Selain itu fluktuasi naik turun juga bisa terjadi ketika sampai di local optima, dimana algoritma akan mulai lagi dari state random. Tapi grafik secara umum terus menurun, menunjukkan bahwa semakin banyaknya iterasi semakin mendekati solusi optimal.

2.4.6. Hasil Genetic Algorithm

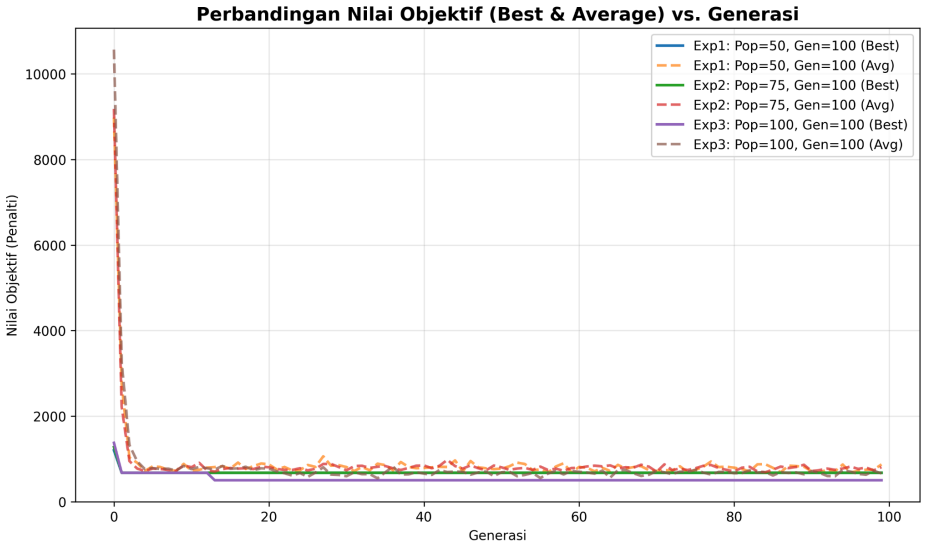
Tabel 2.6 Tabel gambar plotting Genetic Algorithm dengan variasi populasi

Variasi Populasi
PENJALANAN PROGRAM PERTAMA KALI
 **Gambar 2.7 Plot objective function dengan iterasi pada percobaan variasi populasi penjalanan program pertama kali**
PENJALANAN PROGRAM PERTAMA KALI



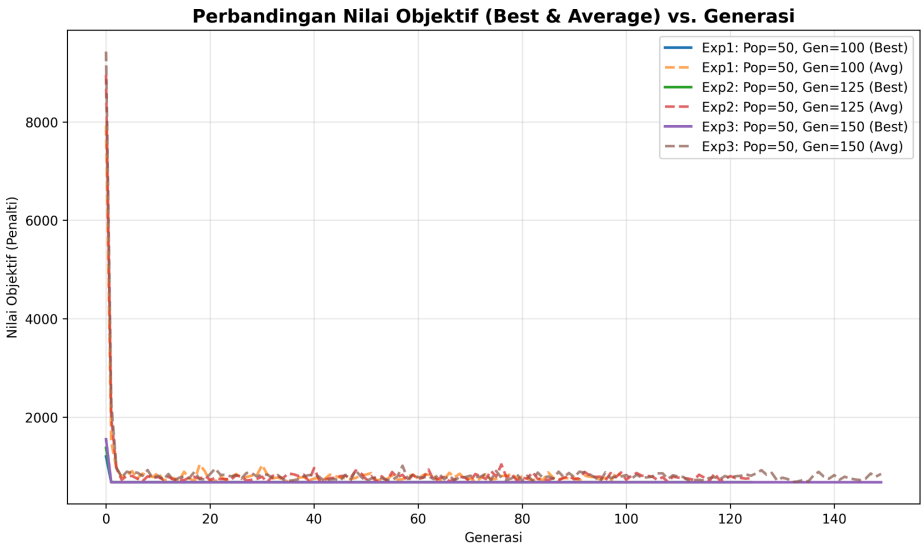
Gambar 2.8 Plot objective function dengan iterasi pada percobaan variasi populasi penjalanan program kedua kali

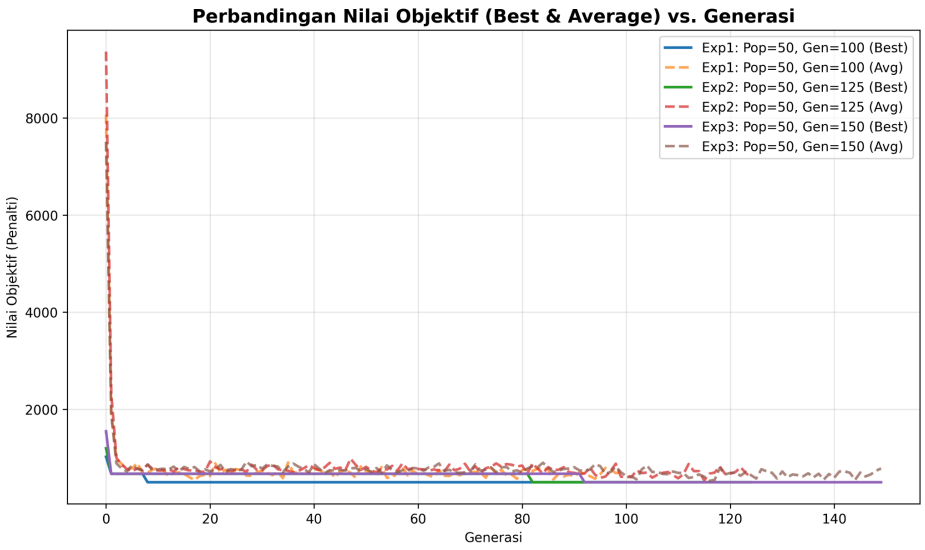
PENJALANAN PROGRAM PERTAMA KALI



Gambar 2.9 Plot objective function dengan iterasi pada percobaan variasi populasi penjalanan program ketiga kali

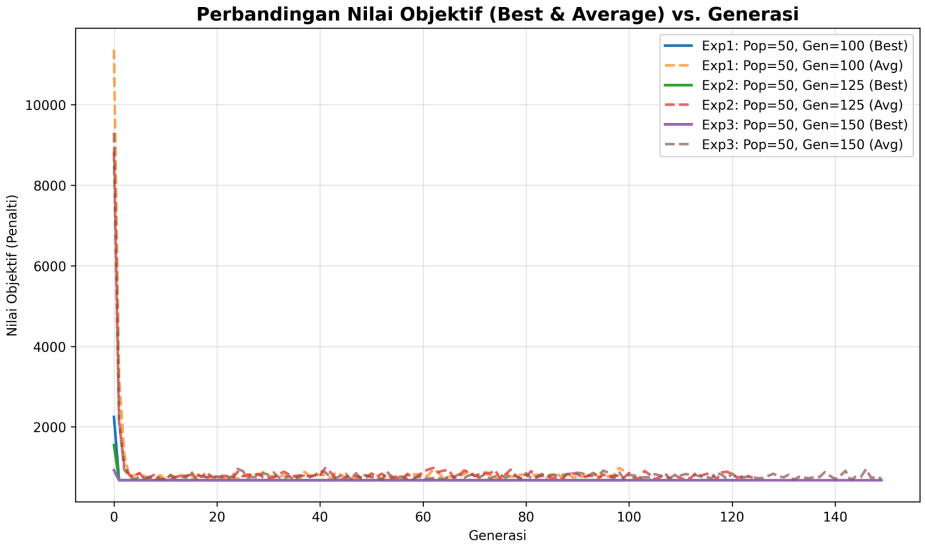
Tabel 2.7 Tabel gambar plotting Genetic Algorithm dengan variasi populasi

Variasi Iterasi
PENJALANAN PROGRAM PERTAMA KALI
 **Gambar 2.10 Plot objective function dengan iterasi pada percobaan variasi iterasi penjalanan program pertama kali**
PENJALANAN PROGRAM PERTAMA KALI



Gambar 2.11 Plot objective function dengan iterasi pada percobaan variasi iterasi penjalanan program kedua kali

PENJALANAN PROGRAM PERTAMA KALI



Gambar 2.12 Plot objective function dengan iterasi pada percobaan variasi iterasi penjalanan program ketiga kali

2.4.6.1. Variasi Populasi

Populasi: 50, 75, 100 dengan Iterasi: 100

Tabel 2.8 State awal dan akhir Genetic Algorithm variasi populasi penjalanan program pertama kali

PENJALANAN PROGRAM PERTAMA KALI
Eksperimen 1
Data Penting
Durasi Proses Pencarian: 0.1340 detik Nilai Objective Function Awal: 1176.5 Nilai Objective Function Akhir: 675.0 Nilai Objective Function Maximum: 30275.00 Rata-Rata Nilai Objective Function: 712.70 Total Kontainer Awal: 8 Total Kontainer Akhir: 6
State Awal
Kontainer 1 (Total: 153/150): !!! OVER CAPACITY by 3 !!! BRG015 (58) BRG007 (45) BRG008 (30) BRG012 (20) Kontainer 2 (Total: 65/150): BRG009 (65) Kontainer 3 (Total: 0/150): Kontainer 4 (Total: 117/150): BRG014 (42) BRG013 (75) Kontainer 5 (Total: 135/150): BRG006 (55) BRG011 (80)

Kontainer 6 (Total: 0/150): Kontainer 7 (Total: 135/150): BRG001 (50) BRG010 (25) BRG002 (60) Kontainer 8 (Total: 145/150): BRG005 (35) BRG003 (40) BRG004 (70)
State Akhir
Kontainer 1 (Total: 145/150): BRG006 (55) BRG003 (40) BRG012 (20) BRG008 (30) Kontainer 2 (Total: 147/150): BRG002 (60) BRG014 (42) BRG007 (45) Kontainer 3 (Total: 140/150): BRG013 (75) BRG009 (65) Kontainer 4 (Total: 138/150): BRG011 (80) BRG015 (58) Kontainer 5 (Total: 75/150): BRG001 (50) BRG010 (25) Kontainer 6 (Total: 105/150): BRG005 (35) BRG004 (70)
Eksperimen 2

Data Penting
Durasi Proses Pencarian: 0.1818 detik Nilai Objective Function Awal: 1025.0 Nilai Objective Function Akhir: 675.0 Nilai Objective Function Maximum: 30450.00 Rata-Rata Nilai Objective Function: 710.50 Total Kontainer Awal: 8 Total Kontainer Akhir: 6
State Awal
Kontainer 1 (Total: 133/150): BRG015 (58) BRG012 (20) BRG006 (55) Kontainer 2 (Total: 0/150): Kontainer 3 (Total: 125/150): BRG011 (80) BRG007 (45) Kontainer 4 (Total: 50/150): BRG001 (50) Kontainer 5 (Total: 130/150): BRG005 (35) BRG010 (25) BRG004 (70) Kontainer 6 (Total: 137/150): BRG009 (65) BRG014 (42) BRG008 (30) Kontainer 7 (Total: 135/150): BRG013 (75) BRG002 (60) Kontainer 8 (Total: 40/150): BRG003 (40)

State Akhir
Kontainer 1 (Total: 133/150): BRG015 (58) BRG012 (20) BRG006 (55) Kontainer 2 (Total: 127/150): BRG002 (60) BRG010 (25) BRG014 (42) Kontainer 3 (Total: 125/150): BRG011 (80) BRG007 (45) Kontainer 4 (Total: 150/150): BRG004 (70) BRG001 (50) BRG008 (30) Kontainer 5 (Total: 140/150): BRG009 (65) BRG005 (35) BRG003 (40) Kontainer 6 (Total: 75/150): BRG013 (75)
Eksperimen 3
Data Penting
Durasi Proses Pencarian: 0.2406 detik Nilai Objective Function Awal: 1200.0 Nilai Objective Function Akhir: 675.0 Nilai Objective Function Maximum: 30450.00 Rata-Rata Nilai Objective Function: 740.98 Total Kontainer Awal: 9 Total Kontainer Akhir: 6
State Awal

Kontainer 1 (Total: 30/150):
BRG008 (30)

Kontainer 2 (Total: 78/150):
BRG015 (58)
BRG012 (20)

Kontainer 3 (Total: 100/150):
BRG006 (55)
BRG007 (45)

Kontainer 4 (Total: 70/150):
BRG004 (70)

Kontainer 5 (Total: 150/150):
BRG009 (65)
BRG005 (35)
BRG001 (50)

Kontainer 6 (Total: 120/150):
BRG011 (80)
BRG003 (40)

Kontainer 7 (Total: 0/150):

Kontainer 8 (Total: 75/150):
BRG013 (75)

Kontainer 9 (Total: 127/150):
BRG010 (25)
BRG014 (42)
BRG002 (60)

State Akhir

Kontainer 1 (Total: 140/150):
BRG011 (80)
BRG002 (60)

Kontainer 2 (Total: 137/150):
BRG010 (25)
BRG014 (42)

BRG004 (70) Kontainer 3 (Total: 150/150): BRG009 (65) BRG001 (50) BRG005 (35) Kontainer 4 (Total: 135/150): BRG003 (40) BRG012 (20) BRG013 (75) Kontainer 5 (Total: 143/150): BRG015 (58) BRG006 (55) BRG008 (30) Kontainer 6 (Total: 45/150): BRG007 (45)

Tabel 2.9 State awal dan akhir Genetic Algorithm variasi populasi penjalanan program kedua kali

PENJALANAN PROGRAM KEDUA KALI
Eksperimen 1
Data Penting
Durasi Proses Pencarian: 0.1178 detik Nilai Objective Function Awal: 1203.5 Nilai Objective Function Akhir: 675.0 Nilai Objective Function Maximum: 30100.00 Rata-Rata Nilai Objective Function: 804.07 Total Kontainer Awal: 7 Total Kontainer Akhir: 6
State Awal
Kontainer 1 (Total: 157/150):

!!! OVER CAPACITY by 7 !!!
BRG014 (42)
BRG011 (80)
BRG005 (35)

Kontainer 2 (Total: 128/150):
BRG008 (30)
BRG015 (58)
BRG003 (40)

Kontainer 3 (Total: 90/150):
BRG012 (20)
BRG010 (25)
BRG007 (45)

Kontainer 4 (Total: 125/150):
BRG001 (50)
BRG013 (75)

Kontainer 5 (Total: 65/150):
BRG009 (65)

Kontainer 6 (Total: 55/150):
BRG006 (55)

Kontainer 7 (Total: 130/150):
BRG004 (70)
BRG002 (60)

State Akhir

Kontainer 1 (Total: 132/150):
BRG010 (25)
BRG014 (42)
BRG009 (65)

Kontainer 2 (Total: 145/150):
BRG004 (70)
BRG005 (35)
BRG003 (40)

Kontainer 3 (Total: 130/150):

BRG002 (60)
BRG001 (50)
BRG012 (20)

Kontainer 4 (Total: 150/150):
BRG008 (30)
BRG007 (45)
BRG013 (75)

Kontainer 5 (Total: 138/150):
BRG011 (80)
BRG015 (58)

Kontainer 6 (Total: 55/150):
BRG006 (55)

Eksperimen 2

Data Penting

Durasi Proses Pencarian: 0.1819 detik
Nilai Objective Function Awal: 1351.5
Nilai Objective Function Akhir: 675.0
Nilai Objective Function Maximum: 30100.00
Rata-Rata Nilai Objective Function: 753.81

Total Kontainer Awal: 9
Total Kontainer Akhir: 6

State Awal

Kontainer 1 (Total: 135/150):
BRG006 (55)
BRG011 (80)

Kontainer 2 (Total: 50/150):
BRG012 (20)
BRG008 (30)

Kontainer 3 (Total: 50/150):
BRG001 (50)

Kontainer 4 (Total: 25/150):
BRG010 (25)

Kontainer 5 (Total: 153/150):
!!! OVER CAPACITY by 3 !!!
BRG005 (35)
BRG002 (60)
BRG015 (58)

Kontainer 6 (Total: 107/150):
BRG009 (65)
BRG014 (42)

Kontainer 7 (Total: 145/150):
BRG013 (75)
BRG004 (70)

Kontainer 8 (Total: 45/150):
BRG007 (45)

Kontainer 9 (Total: 40/150):
BRG003 (40)

State Akhir

Kontainer 1 (Total: 140/150):
BRG003 (40)
BRG004 (70)
BRG008 (30)

Kontainer 2 (Total: 100/150):
BRG005 (35)
BRG009 (65)

Kontainer 3 (Total: 102/150):
BRG014 (42)
BRG002 (60)

Kontainer 4 (Total: 150/150):
BRG013 (75)
BRG001 (50)
BRG010 (25)

Kontainer 5 (Total: 135/150):
BRG006 (55)
BRG011 (80)

Kontainer 6 (Total: 123/150):
BRG007 (45)
BRG012 (20)
BRG015 (58)

Eksperimen 3

Data Penting

Durasi Proses Pencarian: 0.2443 detik
Nilai Objective Function Awal: 1375.0
Nilai Objective Function Akhir: 675.0
Nilai Objective Function Maximum: 30625.00
Rata-Rata Nilai Objective Function: 830.53

Total Kontainer Awal: 10
Total Kontainer Akhir: 5

State Awal

Kontainer 1 (Total: 25/150):
BRG010 (25)

Kontainer 2 (Total: 133/150):
BRG013 (75)
BRG015 (58)

Kontainer 3 (Total: 40/150):
BRG003 (40)

Kontainer 4 (Total: 0/150):

Kontainer 5 (Total: 85/150):
BRG001 (50)
BRG005 (35)

Kontainer 6 (Total: 70/150):
BRG004 (70)

Kontainer 7 (Total: 142/150):
BRG006 (55)
BRG014 (42)
BRG007 (45)

Kontainer 8 (Total: 20/150):
BRG012 (20)

Kontainer 9 (Total: 110/150):
BRG008 (30)
BRG011 (80)

Kontainer 10 (Total: 125/150):
BRG009 (65)
BRG002 (60)

State Akhir

Kontainer 1 (Total: 147/150):
BRG008 (30)
BRG010 (25)
BRG014 (42)
BRG001 (50)

Kontainer 2 (Total: 150/150):
BRG011 (80)
BRG004 (70)

Kontainer 3 (Total: 145/150):
BRG002 (60)
BRG009 (65)
BRG012 (20)

Kontainer 4 (Total: 150/150):
BRG005 (35)
BRG003 (40)
BRG013 (75)

Kontainer 5 (Total: 113/150):
BRG015 (58)
BRG006 (55)

Kontainer 6 (Total: 45/150): BRG007 (45)

Tabel 2.10 State awal dan akhir Genetic Algorithm variasi populasi penjalanan program ketiga kali

PENJALANAN PROGRAM KETIGA KALI
Eksperimen 1
Data Penting
Durasi Proses Pencarian: 0.1228 detik Nilai Objective Function Awal: 1200.0 Nilai Objective Function Akhir: 675.0 Nilai Objective Function Maximum: 30275.00 Rata-Rata Nilai Objective Function: 869.98 Total Kontainer Awal: 9 Total Kontainer Akhir: 6
State Awal
Kontainer 1 (Total: 142/150): BRG004 (70) BRG008 (30) BRG014 (42) Kontainer 2 (Total: 50/150): BRG001 (50) Kontainer 3 (Total: 0/150): Kontainer 4 (Total: 93/150): BRG015 (58) BRG005 (35) Kontainer 5 (Total: 85/150): BRG012 (20) BRG009 (65)

Kontainer 6 (Total: 140/150):
BRG011 (80)
BRG002 (60)

Kontainer 7 (Total: 40/150):
BRG003 (40)

Kontainer 8 (Total: 145/150):
BRG010 (25)
BRG013 (75)
BRG007 (45)

Kontainer 9 (Total: 55/150):
BRG006 (55)

State Akhir

Kontainer 1 (Total: 147/150):
BRG010 (25)
BRG002 (60)
BRG014 (42)
BRG012 (20)

Kontainer 2 (Total: 150/150):
BRG006 (55)
BRG009 (65)
BRG008 (30)

Kontainer 3 (Total: 120/150):
BRG004 (70)
BRG001 (50)

Kontainer 4 (Total: 150/150):
BRG003 (40)
BRG005 (35)
BRG013 (75)

Kontainer 5 (Total: 138/150):
BRG011 (80)
BRG015 (58)

Kontainer 6 (Total: 45/150): BRG007 (45)
Eksperimen 2
Data Penting
Durasi Proses Pencarian: 0.1782 detik Nilai Objective Function Awal: 1277.5 Nilai Objective Function Akhir: 675.0 Nilai Objective Function Maximum: 30275.00 Rata-Rata Nilai Objective Function: 806.02 Total Kontainer Awal: 8 Total Kontainer Akhir:
State Awal
Kontainer 1 (Total: 75/150): BRG013 (75) Kontainer 2 (Total: 155/150): !!! OVER CAPACITY by 5 !!! BRG004 (70) BRG005 (35) BRG001 (50) Kontainer 3 (Total: 0/150): Kontainer 4 (Total: 127/150): BRG010 (25) BRG014 (42) BRG002 (60) Kontainer 5 (Total: 40/150): BRG003 (40) Kontainer 6 (Total: 135/150): BRG011 (80) BRG006 (55) Kontainer 7 (Total: 78/150): BRG015 (58)

BRG012 (20) Kontainer 8 (Total: 140/150): BRG008 (30) BRG009 (65) BRG007 (45)
State Akhir
Kontainer 1 (Total: 150/150): BRG002 (60) BRG004 (70) BRG012 (20) Kontainer 2 (Total: 135/150): BRG013 (75) BRG005 (35) BRG010 (25) Kontainer 3 (Total: 122/150): BRG001 (50) BRG008 (30) BRG014 (42) Kontainer 4 (Total: 145/150): BRG009 (65) BRG011 (80) Kontainer 5 (Total: 143/150): BRG003 (40) BRG015 (58) BRG007 (45) Kontainer 6 (Total: 55/150): BRG006 (55)
Eksperimen 3
Data Penting
Durasi Proses Pencarian: 0.2348 detik Nilai Objective Function Awal: 1375.0 Nilai Objective Function Akhir: 500.0

Nilai Objective Function Maximum: 30625.00
Rata-Rata Nilai Objective Function: 660.08

Total Kontainer Awal: 10
Total Kontainer Akhir: 5

State Awal

Kontainer 1 (Total: 60/150):
- BRG002 (60)

Kontainer 2 (Total: 0/150):

Kontainer 3 (Total: 75/150):
- BRG001 (50)
- BRG010 (25)

Kontainer 4 (Total: 97/150):
- BRG014 (42)
- BRG006 (55)

Kontainer 5 (Total: 150/150):
- BRG008 (30)
- BRG003 (40)
- BRG011 (80)

Kontainer 6 (Total: 58/150):
- BRG015 (58)

Kontainer 7 (Total: 90/150):
- BRG012 (20)
- BRG004 (70)

Kontainer 8 (Total: 45/150):
- BRG007 (45)

Kontainer 9 (Total: 65/150):
- BRG009 (65)

Kontainer 10 (Total: 110/150):
- BRG005 (35)
- BRG013 (75)

State Akhir
Kontainer 1 (Total: 150/150): BRG002 (60) BRG010 (25) BRG009 (65) Kontainer 2 (Total: 150/150): BRG015 (58) BRG014 (42) BRG001 (50) Kontainer 3 (Total: 150/150): BRG006 (55) BRG003 (40) BRG005 (35) BRG012 (20) Kontainer 4 (Total: 150/150): BRG004 (70) BRG011 (80) Kontainer 5 (Total: 150/150): BRG013 (75) BRG008 (30) BRG007 (45)

2.4.6.2.Variasi Iterasi

Populasi: 50 dengan Iterasi: 100, 125, 150

Tabel 2.11 State awal dan akhir Genetic Algorithm variasi iterasi penjalanan program pertama kali

PENJALANAN PROGRAM PERTAMA KALI
Eksperimen 1
Data Penting

Durasi Proses Pencarian: 0.1220 detik
Nilai Objective Function Awal: 1200.0
Nilai Objective Function Akhir: 675.0
Nilai Objective Function Maximum: 30625.00
Rata-Rata Nilai Objective Function: 813.77

Total Kontainer Awal: 9
Total Kontainer Akhir: 6

State Awal

Kontainer 1 (Total: 123/150):
- BRG015 (58)
- BRG009 (65)

Kontainer 2 (Total: 115/150):
- BRG010 (25)
- BRG004 (70)
- BRG012 (20)

Kontainer 3 (Total: 75/150):
- BRG008 (30)
- BRG007 (45)

Kontainer 4 (Total: 115/150):
- BRG013 (75)
- BRG003 (40)

Kontainer 5 (Total: 145/150):
- BRG005 (35)
- BRG001 (50)
- BRG002 (60)

Kontainer 6 (Total: 122/150):
- BRG014 (42)
- BRG011 (80)

Kontainer 7 (Total: 55/150):
- BRG006 (55)

Kontainer 8 (Total: 0/150):

Kontainer 9 (Total: 0/150):
State Akhir
Kontainer 1 (Total: 137/150): BRG010 (25) BRG014 (42) BRG004 (70) Kontainer 2 (Total: 145/150): BRG005 (35) BRG001 (50) BRG003 (40) BRG012 (20) Kontainer 3 (Total: 150/150): BRG009 (65) BRG008 (30) BRG006 (55) Kontainer 4 (Total: 140/150): BRG002 (60) BRG011 (80) Kontainer 5 (Total: 133/150): BRG013 (75) BRG015 (58) Kontainer 6 (Total: 45/150): BRG007 (45)
Eksperimen 2
Data Penting
Durasi Proses Pencarian: 0.1588 detik Nilai Objective Function Awal: 1375.0 Nilai Objective Function Akhir: 675.0 Nilai Objective Function Maximum: 30100.00 Rata-Rata Nilai Objective Function: 760.17 Total Kontainer Awal: 10 Total Kontainer Akhir: 6

State Awal
Kontainer 1 (Total: 55/150): BRG006 (55) Kontainer 2 (Total: 100/150): BRG014 (42) BRG015 (58) Kontainer 3 (Total: 120/150): BRG003 (40) BRG012 (20) BRG002 (60) Kontainer 4 (Total: 150/150): BRG011 (80) BRG004 (70) Kontainer 5 (Total: 45/150): BRG007 (45) Kontainer 6 (Total: 110/150): BRG013 (75) BRG005 (35) Kontainer 7 (Total: 50/150): BRG001 (50) Kontainer 8 (Total: 120/150): BRG010 (25) BRG008 (30) BRG009 (65) Kontainer 9 (Total: 0/150): Kontainer 10 (Total: 0/150):
State Akhir
Kontainer 1 (Total: 147/150): BRG010 (25) BRG002 (60) BRG014 (42)

BRG012 (20)

Kontainer 2 (Total: 140/150):
BRG001 (50)
BRG006 (55)
BRG005 (35)

Kontainer 3 (Total: 135/150):
BRG009 (65)
BRG004 (70)

Kontainer 4 (Total: 150/150):
BRG011 (80)
BRG003 (40)
BRG008 (30)

Kontainer 5 (Total: 133/150):
BRG013 (75)
BRG015 (58)

Kontainer 6 (Total: 45/150):
BRG007 (45)

Eksperimen 3

Data Penting

Durasi Proses Pencarian: 0.1978 detik
Nilai Objective Function Awal: 1550.0
Nilai Objective Function Akhir: 675.0
Nilai Objective Function Maximum: 30275.00
Rata-Rata Nilai Objective Function: 844.07

Total Kontainer Awal: 11
Total Kontainer Akhir: 6

State Awal

Kontainer 1 (Total: 30/150):
BRG008 (30)

Kontainer 2 (Total: 0/150):

Kontainer 3 (Total: 133/150): BRG013 (75) BRG015 (58) Kontainer 4 (Total: 65/150): BRG009 (65) Kontainer 5 (Total: 35/150): BRG005 (35) Kontainer 6 (Total: 82/150): BRG014 (42) BRG003 (40) Kontainer 7 (Total: 115/150): BRG006 (55) BRG002 (60) Kontainer 8 (Total: 0/150): Kontainer 9 (Total: 115/150): BRG007 (45) BRG004 (70) Kontainer 10 (Total: 105/150): BRG010 (25) BRG011 (80) Kontainer 11 (Total: 70/150): BRG001 (50) BRG012 (20)
State Akhir
Kontainer 1 (Total: 145/150): BRG009 (65) BRG002 (60) BRG012 (20) Kontainer 2 (Total: 147/150): BRG005 (35) BRG014 (42)

BRG004 (70) Kontainer 3 (Total: 125/150): BRG013 (75) BRG001 (50) Kontainer 4 (Total: 150/150): BRG011 (80) BRG003 (40) BRG008 (30) Kontainer 5 (Total: 138/150): BRG010 (25) BRG015 (58) BRG006 (55) Kontainer 6 (Total: 45/150): BRG007 (45)

Tabel 2.12 State awal dan akhir Genetic Algorithm variasi iterasi penjalanan program kedua kali

PENJALANAN PROGRAM KEDUA KALI
Eksperimen 1
Data Penting
Durasi Proses Pencarian: 0.1125 detik Nilai Objective Function Awal: 1025.0 Nilai Objective Function Akhir: 500.0 Nilai Objective Function Maximum: 30275.00 Rata-Rata Nilai Objective Function: 699.02 Total Kontainer Awal: 8 Total Kontainer Akhir: 6
State Awal
Kontainer 1 (Total: 138/150): BRG011 (80)

BRG015 (58) Kontainer 2 (Total: 150/150): BRG003 (40) BRG002 (60) BRG008 (30) BRG012 (20) Kontainer 3 (Total: 70/150): BRG004 (70) Kontainer 4 (Total: 80/150): BRG007 (45) BRG005 (35) Kontainer 5 (Total: 130/150): BRG013 (75) BRG006 (55) Kontainer 6 (Total: 50/150): BRG001 (50) Kontainer 7 (Total: 132/150): BRG010 (25) BRG009 (65) BRG014 (42) Kontainer 8 (Total: 0/150):
State Akhir
Kontainer 1 (Total: 150/150): BRG007 (45) BRG002 (60) BRG010 (25) BRG012 (20) Kontainer 2 (Total: 150/150): BRG014 (42) BRG001 (50) BRG015 (58)

Kontainer 3 (Total: 150/150): BRG006 (55) BRG009 (65) BRG008 (30) Kontainer 4 (Total: 150/150): BRG011 (80) BRG004 (70) Kontainer 5 (Total: 150/150): BRG013 (75) BRG005 (35) BRG003 (40)
Eksperimen 2
Data Penting
Durasi Proses Pencarian: 0.1570 detik Nilai Objective Function Awal: 1200.0 Nilai Objective Function Akhir: 500.0 Nilai Objective Function Maximum: 30100.00 Rata-Rata Nilai Objective Function: 591.49 Total Kontainer Awal: 9 Total Kontainer Akhir: 5
State Awal
Kontainer 1 (Total: 113/150): BRG015 (58) BRG006 (55) Kontainer 2 (Total: 100/150): BRG005 (35) BRG003 (40) BRG010 (25) Kontainer 3 (Total: 130/150): BRG002 (60) BRG004 (70) Kontainer 4 (Total: 75/150):

BRG008 (30)
BRG007 (45)

Kontainer 5 (Total: 145/150):
BRG012 (20)
BRG013 (75)
BRG001 (50)

Kontainer 6 (Total: 0/150):

Kontainer 7 (Total: 0/150):

Kontainer 8 (Total: 65/150):
BRG009 (65)

Kontainer 9 (Total: 122/150):
BRG014 (42)
BRG011 (80)

State Akhir

Kontainer 1 (Total: 150/150):
BRG013 (75)
BRG006 (55)
BRG012 (20)

Kontainer 2 (Total: 150/150):
BRG011 (80)
BRG003 (40)
BRG008 (30)

Kontainer 3 (Total: 150/150):
BRG009 (65)
BRG002 (60)
BRG010 (25)

Kontainer 4 (Total: 150/150):
BRG007 (45)
BRG004 (70)
BRG005 (35)

Kontainer 5 (Total: 150/150):

BRG001 (50)
BRG014 (42)
BRG015 (58)

Eksperimen 3

Data Penting

Durasi Proses Pencarian: 0.1911 detik
Nilai Objective Function Awal: 1550.0
Nilai Objective Function Akhir: 500.0
Nilai Objective Function Maximum: 30100.00
Rata-Rata Nilai Objective Function: 784.33

Total Kontainer Awal: 11
Total Kontainer Akhir: 5

State Awal

Kontainer 1 (Total: 40/150):
BRG003 (40)

Kontainer 2 (Total: 118/150):
BRG002 (60)
BRG015 (58)

Kontainer 3 (Total: 65/150):
BRG008 (30)
BRG005 (35)

Kontainer 4 (Total: 140/150):
BRG007 (45)
BRG004 (70)
BRG010 (25)

Kontainer 5 (Total: 120/150):
BRG009 (65)
BRG006 (55)

Kontainer 6 (Total: 100/150):
BRG011 (80)
BRG012 (20)

Kontainer 7 (Total: 75/150):
BRG013 (75)

Kontainer 8 (Total: 0/150):

Kontainer 9 (Total: 92/150):
BRG014 (42)
BRG001 (50)

Kontainer 10 (Total: 0/150):

Kontainer 11 (Total: 0/150):

State Akhir

Kontainer 1 (Total: 150/150):
BRG009 (65)
BRG002 (60)
BRG010 (25)

Kontainer 2 (Total: 150/150):
BRG013 (75)
BRG006 (55)
BRG012 (20)

Kontainer 3 (Total: 150/150):
BRG015 (58)
BRG014 (42)
BRG001 (50)

Kontainer 4 (Total: 150/150):
BRG005 (35)
BRG004 (70)
BRG007 (45)

Kontainer 5 (Total: 150/150):
BRG011 (80)
BRG003 (40)
BRG008 (30)

Tabel 2.13 State awal dan akhir Genetic Algorithm variasi iterasi penjalanan program ketiga kali

PENJALANAN PROGRAM KETIGA KALI
Eksperimen 1
Data Penting
Durasi Proses Pencarian: 0.1180 detik Nilai Objective Function Awal: 2250.0 Nilai Objective Function Akhir: 675.0 Nilai Objective Function Maximum: 30100.00 Rata-Rata Nilai Objective Function: 912.75 Total Kontainer Awal: 15 Total Kontainer Akhir: 6
State Awal
Kontainer 1 (Total: 115/150): BRG011 (80) BRG005 (35) Kontainer 2 (Total: 0/150): Kontainer 3 (Total: 65/150): BRG009 (65) Kontainer 4 (Total: 42/150): BRG014 (42) Kontainer 5 (Total: 0/150): Kontainer 6 (Total: 0/150): Kontainer 7 (Total: 115/150): BRG002 (60) BRG006 (55) Kontainer 8 (Total: 70/150): BRG004 (70)

Kontainer 9 (Total: 75/150):
BRG013 (75)

Kontainer 10 (Total: 0/150):

Kontainer 11 (Total: 83/150):
BRG010 (25)
BRG015 (58)

Kontainer 12 (Total: 50/150):
BRG001 (50)

Kontainer 13 (Total: 50/150):
BRG008 (30)
BRG012 (20)

Kontainer 14 (Total: 40/150):
BRG003 (40)

Kontainer 15 (Total: 45/150):
BRG007 (45)

State Akhir

Kontainer 1 (Total: 147/150):
BRG002 (60)
BRG010 (25)
BRG014 (42)
BRG012 (20)

Kontainer 2 (Total: 135/150):
BRG004 (70)
BRG009 (65)

Kontainer 3 (Total: 125/150):
BRG001 (50)
BRG005 (35)
BRG003 (40)

Kontainer 4 (Total: 138/150):
BRG011 (80)
BRG015 (58)

Kontainer 5 (Total: 130/150):
BRG013 (75)
BRG006 (55)

Kontainer 6 (Total: 75/150):
BRG008 (30)
BRG007 (45)

Eksperimen 2

Data Penting

Durasi Proses Pencarian: 0.1431 detik
Nilai Objective Function Awal: 1550.0
Nilai Objective Function Akhir: 675.0
Nilai Objective Function Maximum: 30625.00
Rata-Rata Nilai Objective Function: 739.90

Total Kontainer Awal: 11
Total Kontainer Akhir: 6

State Awal

Kontainer 1 (Total: 70/150):
BRG004 (70)

Kontainer 2 (Total: 60/150):
BRG002 (60)

Kontainer 3 (Total: 120/150):
BRG001 (50)
BRG007 (45)
BRG010 (25)

Kontainer 4 (Total: 138/150):
BRG011 (80)
BRG015 (58)

Kontainer 5 (Total: 107/150):
BRG009 (65)
BRG014 (42)

Kontainer 6 (Total: 30/150):
BRG008 (30)

Kontainer 7 (Total: 130/150):
BRG013 (75)
BRG006 (55)

Kontainer 8 (Total: 55/150):
BRG012 (20)
BRG005 (35)

Kontainer 9 (Total: 40/150):
BRG003 (40)

Kontainer 10 (Total: 0/150):

Kontainer 11 (Total: 0/150):

State Akhir

Kontainer 1 (Total: 147/150):
BRG002 (60)
BRG010 (25)
BRG014 (42)
BRG012 (20)

Kontainer 2 (Total: 145/150):
BRG008 (30)
BRG009 (65)
BRG001 (50)

Kontainer 3 (Total: 150/150):
BRG004 (70)
BRG005 (35)
BRG007 (45)

Kontainer 4 (Total: 120/150):
BRG003 (40)
BRG011 (80)

Kontainer 5 (Total: 133/150):
BRG013 (75)

BRG015 (58)

Kontainer 6 (Total: 55/150):
BRG006 (55)

Eksperimen 3

Data Penting

Durasi Proses Pencarian: 0.1720 detik
Nilai Objective Function Awal: 927.5
Nilai Objective Function Akhir: 675.0
Nilai Objective Function Maximum: 30100.00
Rata-Rata Nilai Objective Function: 719.70

Total Kontainer Awal: 6
Total Kontainer Akhir: 6

State Awal

Kontainer 1 (Total: 117/150):
BRG014 (42)
BRG013 (75)

Kontainer 2 (Total: 90/150):
BRG012 (20)
BRG010 (25)
BRG007 (45)

Kontainer 3 (Total: 125/150):
BRG004 (70)
BRG006 (55)

Kontainer 4 (Total: 145/150):
BRG009 (65)
BRG011 (80)

Kontainer 5 (Total: 118/150):
BRG015 (58)
BRG002 (60)

Kontainer 6 (Total: 155/150):
!!! OVER CAPACITY by 5 !!!

BRG003 (40) BRG008 (30) BRG005 (35) BRG001 (50)
State Akhir
Kontainer 1 (Total: 120/150): BRG003 (40) BRG002 (60) BRG012 (20) Kontainer 2 (Total: 150/150): BRG005 (35) BRG004 (70) BRG007 (45) Kontainer 3 (Total: 137/150): BRG014 (42) BRG008 (30) BRG009 (65) Kontainer 4 (Total: 130/150): BRG010 (25) BRG006 (55) BRG001 (50) Kontainer 5 (Total: 138/150): BRG011 (80) BRG015 (58) Kontainer 6 (Total: 75/150): BRG013 (75)

2.4.6.3. Analisis

Berdasarkan hasil eksperimen, Genetic Algorithm dalam sebagian variasi percobaan mampu mencapai global optima, terutama pada percobaan yang menggunakan iterasi yang lebih banyak. Walaupun pada eksperimen ini terlihat iterasi lebih berpengaruh dari populasi dalam mencapai global optima,

tetapi sebenarnya tidak seperti itu karena proses ini tidak konsisten karena banyak variabel dan proses yang random (mulai dari inisiasi state awal, proses crossover, dan proses mutate).

Durasi percobaan Genetic Algorithm cenderung lebih lambat dibandingkan algoritma local search lain karena perlu melakukan iterasi yang lebih banyak dan lebih kompleks. Hasil dari Genetic Algorithm juga lebih bervariasi karena banyaknya proses dan variabel pada prosesnya bersifat random, tetapi dengan iterasi dan populasi yang cukup banyak, hasilnya biasanya konsisten dan mendekati optimum. Jadi semakin banyak iterasi dan populasi, kualitas solusi dari Genetic Algorithm akan semakin bagus dengan trade-off membutuhkan waktu komputasi yang meningkat.

Dapat dilihat juga pada plotting bahwa saat iterasi rendah nilai average objective function dan best objective function masih sangat tinggi, dan setelah beberapa iterasi keduanya menjadi jauh lebih baik. Hal ini terjadi karena populasi yang diambil dari generasi sebelumnya dipilih berdasarkan objective function. Jadi hasil iterasi selanjutnya akan memiliki populasi yang memiliki objective function yang lebih rendah. Proses ini akan berlanjut sesuai dengan jumlah iterasi yang sudah ditetapkan.

3. Kesimpulan dan Saran

Algoritma yang digunakan pada eksperimen-eksperimen ini adalah local search, jadi solusi akhirnya belum tentu semua sampai global optima. Hal ini juga dipengaruhi

banyak parameter dan proses yang sifatnya random jadi berbeda setiap eksperimennya. Kemampuan algoritma untuk mencari solusi yang terbaik juga dipengaruhi oleh banyaknya jumlah iterasi dan populasi. Berikut adalah kelebihan dan kekurangan singkat dari masing-masing algoritma yang digunakan:

Hill Climbing Steepest Ascent memiliki kelebihan karena prosesnya yang sederhana sehingga membutuhkan durasi yang singkat karena algoritmanya hanya memilih tetangga dengan nilai terbaik. Namun, algoritma ini mudah terjebak pada local optima.

Hill Climbing Sideways Move memperbolehkan untuk berpindah ke neighbour dengan nilai penalti sama sebagai upaya keluar dari local optima, tetapi tetap tidak bisa memastikan menghindari terjebak dari local optima.

Hill Climbing Stochastic memiliki kelebihan dimana algoritmanya bisa menghindari local optima dengan cara pemilihan tetangga yang dilakukan secara acak sehingga eksplorasi lebih luas, tetapi kekurangannya pemilihan tidak selalu jelas sehingga hasil setiap eksperimen bisa sangat bervariasi.

Hill Climbing Random Restart memiliki kelebihan dari algoritma Hill Climbing lain karena eksplorasi yang dilakukan dimulai dari banyak titik awal acak dengan banyak iterasi sehingga peluang mencapai global optima lebih besar.

Simulated Annealing memiliki kelebihan bahwa dia bisa keluar dari local optima dengan menerima solusi yang lebih buruk di awal jalannya algoritma. Kekurangannya adalah perlunya pengaturan parameter yang tepat, jika tidak tepat maka hasil tidak stabil dan proses komputasi akan lambat.

Genetic Algorithm memiliki kelebihan dengan kemampuannya untuk menggabungkan proses seleksi, crossover, dan mutasi akan membuat ruang solusi yang lebih besar dibanding algoritma local search lain sehingga memiliki

kemungkinan yang paling besar untuk mencapai global optima dibandingkan dengan algoritma local search lain. Kekurangannya adalah butuhnya pengaturan parameter yang tepat dan perlu waktu komputasi yang banyak

Dalam program ini disarankan untuk melakukan eksperimen dengan problem yang lebih kompleks dengan banyak edge case. Selain itu, ada juga penggunaan lebih dari satu algoritma local search untuk mencapai global optima juga dapat dieksplorasi.

Pembagian Tugas

Anggota	NIM	Pembagian Tugas
Valereo Jibril Al Buchori	18223030	All Hill Climbing, Laporan
Mahesa Satria Prayata	18223082	Genetic Algorithm, Laporan
Jason Samuel	18223091	Simulated Annealing, Laporan

Github Repository

echaa0018/tubes1_ai

Referensi

- https://www.geeksforgeeks.org/artificial-intelligence/introduction-hill-climbing-artificial-intelligence/
- https://www.datacamp.com/tutorial/hill-climbing-algorithm-for-ai-in-python
- Artificial Intelligence A Modern Approach 4th - Stuart Russel