

Tugas Besar 1 IF3070 Dasar Inteligensi Artifisial
Pencarian Solusi Pengepakan Barang
(*Bin Packing Problem*)



Kelompok 5:
18223078 Vincentia Belinda Sumartoyo
18223100 Indiana Aulia Ayundazulfa
18223106 Nurul Na'im Natifah

Dosen Pengampu :
Dr. Nur Ulfa Maulidevi, ST, M.Sc.

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB 1 DESKRIPSI persoalan.....	3
BAB 2 PEMBAHASAN.....	5
2.1. Objective Function.....	5
2.2. Implementasi Algoritma Local Search.....	6
a. Hill-Climbing Steepest Ascent.....	9
b. Hill-Climbing with Sideways Move.....	13
c. Random Restart Hill-Climbing.....	18
d. Stochastic Hill-Climbing.....	25
e. Simulated Annealing.....	29
f. Genetic Algorithm.....	37
BAB 3 HASIL DAN ANALISIS.....	46
a. Steepest Ascent Hill-Climbing.....	46
b. Hill-Climbing with Sideways Move.....	49
c. Random Restart Hill-Climbing.....	52
d. Stochastic Hill-Climbing.....	58
e. Simulated Annealing.....	62
f. Genetic Algorithm.....	68
g. Analisis.....	82
BAB 4 KESIMPULAN DAN SARAN.....	84
4.1. Kesimpulan.....	84
4.2. Saran.....	84
PEMBAGIAN TUGAS.....	85
REFERENSI.....	86

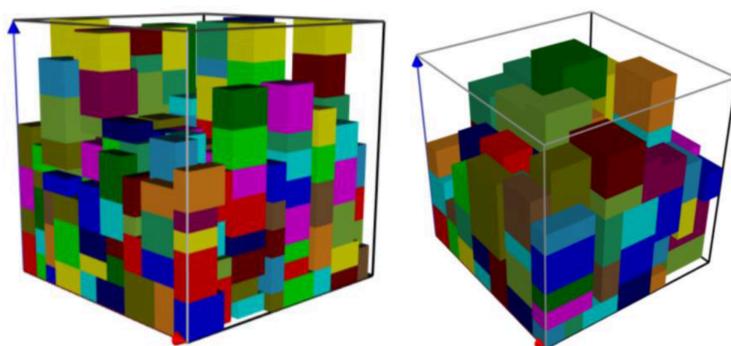
BAB 1

DESKRIPSI PERSOALAN

Masalah *Bin Packing* adalah masalah optimasi kombinatorial klasik di bidang ilmu komputer dan riset operasi. Dalam permasalahan ini, sejumlah *item* (barang) dengan berbagai ukuran harus ditempatkan ke dalam sejumlah *bin* (kontainer atau wadah) yang masing-masing memiliki kapasitas tetap, dengan tujuan meminimalkan jumlah *bin* yang digunakan. Sebagai contoh, jika setiap *bin* memiliki kapasitas satuan, penugasan *item* ke *bin* perlu dicari sedemikian sehingga jumlah *bin* yang terpakai minimal. Setiap barang memiliki ukuran tertentu (misalnya berat atau volume), dan tidak boleh melebihi kapasitas satu kontainer secara keseluruhan. Misalnya, jika kapasitas kontainer ditetapkan 100 satuan, maka gabungan ukuran barang dalam setiap kontainer tidak boleh lebih dari 100. Persyaratan penempatan ini menyiratkan bahwa setiap barang ditempatkan secara utuh ke dalam salah satu kontainer (tidak ada pemecahan barang) dan setiap kontainer hanya boleh menampung barang sampai batas kapasitasnya.

Batasan utamanya adalah setiap *bin* tidak boleh melebihi kapasitasnya dan setiap *item* harus diletakkan utuh (tidak dibagi). Dari sisi komputasional, *bin packing* termasuk masalah NP-hard dimana keputusan apakah semua *item* dapat diisi dalam dua *bin* yang bersifat NP-complete. Dengan kata lain, mencari solusi optimal sulit dicapai secara pasti pada skala besar, sehingga sering digunakan pendekatan heuristik atau metaheuristik.

Tujuan optimasi dalam masalah ini secara formal dinyatakan sebagai minimisasi jumlah kontainer yang terpakai. Dengan kata lain, solusi yang diinginkan adalah konfigurasi penempatan yang valid (semua barang teralokasi tanpa melebihi kapasitas) dengan jumlah kontainer sesedikit mungkin. Spesifikasi tugas bahkan mengingatkan bahwa setiap kontainer yang total ukuran barangnya melebihi kapasitas harus dikenai penalti besar agar solusi akhir bersifat valid. Oleh karena itu, dalam solusi yang sah, total ukuran barang dalam setiap kontainer tidak boleh melampaui kapasitas yang ditetapkan. Penempatan yang padat dan efisien (kepadatan muatan tinggi) secara langsung berkaitan dengan penggunaan kontainer yang minimum.



Gambar di atas mengilustrasikan konsep *bin packing* dimana *item* atau barang dengan ukuran beragam ditempatkan ke dalam beberapa kotak (*bin*) berbeda sehingga jumlah kotak yang terpakai seminimal mungkin. Sebagai contoh skematis, satu kotak bisa digambarkan menampung beberapa *item* hingga kapasitasnya terisi, sedangkan kotak lain menampung sisa *item*. Optimasi dalam konteks *bin packing* berarti menata pemilahan dan penempatan *item* sedemikian rupa agar penggunaan wadah menjadi efisien, dengan memperhatikan batasan kapasitas masing-masing wadah. Pendekatan optimal umumnya sulit dihitung langsung karena kompleksitasnya, sehingga laporan ini mempertimbangkan metode pencarian lokal untuk menemukan solusi mendekati optimum dalam waktu wajar.

BAB 2

PEMBAHASAN

2.1. Objective Function

Fungsi objektif pada masalah *bin packing* dirancang untuk menghasilkan satu nilai numerik yang mencerminkan kualitas keseluruhan solusi. Solusi ini sendiri direpresentasikan sebagai *list of lists* (atau *list of integers*), di mana setiap *list* mewakili satu kontainer berisi sejumlah barang. Dalam konteks ini, komponen pertama dan paling penting dari fungsi objektif adalah jumlah total kontainer yang digunakan. Karena tujuan utama adalah meminimalkan jumlah kontainer, fungsi objektif akan memberi skor lebih rendah (lebih baik) pada solusi yang menggunakan lebih sedikit kontainer. Setiap kontainer yang terpakai akan menambah nilai (bobot) pada fungsi objektif.

Komponen kedua adalah penalti. Dalam fungsi objektif ini, setiap kali suatu kontainer dipenuhi melebihi kapasitas maksimalnya, sistem memberikan penalti besar (misalnya 10.000 per kejadian). *Overflow* terdeteksi dengan memeriksa apabila jumlah ukuran barang dalam satu kontainer lebih besar dari kapasitas yang ditetapkan. Ketika terdeteksi *overflow*, skor fungsi objektif ditambah dengan nilai penalti tersebut. Penalti sebesar 10.000 dirancang sangat tinggi agar solusi yang menimbulkan *overflow* menjadi sangat tidak diinginkan, bahkan satu kali *overflow* dapat membuat skor akhir jauh lebih buruk dibandingkan menggunakan kontainer tambahan. Dengan begitu, setiap solusi yang valid (tanpa *overflow*) akan selalu diutamakan, dan *overflow* hanya boleh dipakai sebagai upaya terburuk karena mengakibatkan tingkatan skor yang masif.

Komponen ketiga adalah skor untuk kepadatan kontainer, yang memberikan penghargaan (*reward*) pada solusi dengan pengisian ruang yang tinggi. Kontainer yang terisi rapat (dekat dengan 100%) dianggap lebih baik. Salah satu cara untuk menghitung ini adalah dengan meminimalkan ruang kosong total atau memaksimalkan pemanfaatan kapasitas seluruh kontainer. Dengan demikian, solusi yang meningkatkan kepadatan setiap kontainer akan menurunkan nilai fungsi objektif secara keseluruhan. Pendekatan ini sejalan dengan praktik memaksimalkan *packing density*, di mana peningkatan utilisasi lebih diprioritaskan dibanding pengurangan satu kontainer ekstra. Untuk menghasilkan nilai numerik akhir tunggal, ketiga komponen di atas digabungkan (biasanya dalam bentuk kombinasi perkalian, penjumlahan *weighted* atau akumulasi skor/penalti).

Dalam praktiknya, fungsi semacam ini dirancang agar semakin kecil hasilnya untuk solusi yang diinginkan: jumlah kontainer lebih sedikit, tidak ada *overflow*, dan kepadatan tinggi. Sebagai ilustrasi ringkas, yaitu solusi yang menggunakan sedikit kontainer tanpa melanggar kapasitas dan dengan kepadatan kontainer tinggi akan menghasilkan nilai objektif terendah. Dalam rangkaian optimasi, fungsi objektif yang

terdefinisi demikian memastikan solusi yang dipilih secara otomatis menyeimbangkan ketiga aspek penting tersebut. Prioritas utama pada minimisasi jumlah kontainer, penalti kuat untuk overflow, dan penghargaan pada pengisian kontainer yang rapat.

2.2. Implementasi Algoritma Local Search

utils.py

```
import json

# parsing json

def load_data(json_file):
    with open(json_file, 'r') as f:
        file_data = json.load(f)

    kapasitas = file_data['kapasitas_kontainer']
    list_barang_awal = file_data['barang']

    data_barang = {brg['id']: brg['ukuran'] for brg in
list_barang_awal}

    return kapasitas, data_barang, list_barang_awal
```

File utils.py berfungsi untuk membaca file JSON yang berisi informasi kapasitas kontainer dan daftar barang. Fungsi ini mengekstrak data menjadi tiga komponen, yaitu kapasitas (kapasitas maksimum tiap kontainer), data_barang (*dictionary* dengan pasangan id dan ukuran barang), dan list_barang_awal (list berisi detail barang dalam urutan aslinya). Nilai-nilai ini dikembalikan ke program utama untuk digunakan dalam proses inisialisasi dan evaluasi solusi.

hc_utils.py

```
import copy
import random
import matplotlib.pyplot as plt

def get_total_ukuran(kontainer, data_barang):
    total = 0
    for id_barang in kontainer:
        total += data_barang[id_barang]
    return total

def inisialisasi_first_fit(list_barang_awal, kapasitas,
data_barang):
    state_awal = []

    for barang in list_barang_awal:
```

```

        id_barang_baru = barang['id']
        ukuran_barang_baru = barang['ukuran']

        barang_sudah_ditempatkan = False

        for kontainer in state_awal:
            total_ukuran_saat_ini =
get_total_ukuran(kontainer, data_barang)

            if total_ukuran_saat_ini + ukuran_barang_baru <=
kapasitas:
                kontainer.append(id_barang_baru)
                barang_sudah_ditempatkan = True
                break

            if not barang_sudah_ditempatkan:
                kontainer_baru = [id_barang_baru]
                state_awal.append(kontainer_baru)

        return state_awal

def inisialisasi_random(list_barang_awal, kapasitas,
data_barang):
    shuffled_list = copy.deepcopy(list_barang_awal)
    random.shuffle(shuffled_list)
    return inisialisasi_first_fit(shuffled_list, kapasitas,
data_barang)

def calculate_objective(state, kapasitas, data_barang):
    PENALTI_OVERLOAD = 10000
    FAKTOR_BOBOT_KONTAINER = 100

    jumlah_kontainer = len(state)
    skor_total = jumlah_kontainer * FAKTOR_BOBOT_KONTAINER

    total_reward_kepadatan = 0

    for kontainer in state:
        if not kontainer:
            continue

        total_ukuran_kontainer = get_total_ukuran(kontainer,
data_barang)

        if total_ukuran_kontainer > kapasitas:
            skor_total += PENALTI_OVERLOAD
        else:
            kepadatan = total_ukuran_kontainer / kapasitas
            total_reward_kepadatan += (kepadatan ** 2)

    skor_total -= total_reward_kepadatan

```

```

        return skor_total

def clean_empty_containers(state):
    return [kontainer for kontainer in state if kontainer]

def get_neighbors(current_state):
    numContainers = len(current_state)

    for i in range(numContainers):
        if not current_state[i]:
            continue

        for j in range(len(current_state[i])):
            for k in range(numContainers):
                if i == k:
                    continue

                neighbor_state = copy.deepcopy(current_state)
                barang_yg_dipindah = neighbor_state[i].pop(j)
                neighbor_state[k].append(barang_yg_dipindah)

                yield clean_empty_containers(neighbor_state)

                neighbor_state = copy.deepcopy(current_state)
                barang_yg_dipindah = neighbor_state[i].pop(j)
                neighbor_state.append([barang_yg_dipindah])

                yield clean_empty_containers(neighbor_state)

    for i in range(numContainers):
        for j in range(len(current_state[i])):
            for k in range(i + 1, numContainers):
                for l in range(len(current_state[k])):

                    neighbor_state =
copy.deepcopy(current_state)

                    barang1 = neighbor_state[i][j]
                    barang2 = neighbor_state[k][l]
                    neighbor_state[i][j] = barang2
                    neighbor_state[k][l] = barang1

                    yield neighbor_state

def print_state_terminal(state, kapasitas, data_barang):
    print("-" * 40)

    sorted_state = sorted(state, key=lambda k:
get_total_ukuran(k, data_barang), reverse=True)

```

```

for i, kontainer in enumerate(sorted_state):
    total_usage = get_total_ukuran(kontainer, data_barang)
    items_str = ", ".join(kontainer)
    warning = ""
    if total_usage > kapasitas:
        warning = " <-- OVERLOAD!"

    print(f"Kontainer {i+1:2d} (Total:
{total_usage:3d}/{kapasitas}) {warning}")
    print(f"  [{items_str}]")

print("-" * 40)
print(f"Total Kontainer Digunakan: {len(state)}")

```

File hc_utils.py berisi kumpulan fungsi pendukung yang digunakan oleh berbagai varian algoritma *hill-climbing* seperti steepest, sideways move, stochastic, dan random restart. Fungsinya meliputi pembuatan solusi awal dengan metode *first fit* (inisialisasi_first_fit()) atau acak (inisialisasi_random()), perhitungan nilai objective function (calculate_objective()), serta pembangkitan tetangga solusi (get_neighbors()) dengan cara memindahkan atau menukar barang antar kontainer.

Selain itu, file ini juga menyediakan fungsi lain seperti get_total_ukuran() untuk menghitung total isi kontainer, clean_empty_containers() untuk menghapus kontainer kosong, dan print_state_terminal() untuk menampilkan konfigurasi kontainer ke terminal. Dengan adanya file ini, setiap varian *hill-climbing* dapat menggunakan logika evaluasi dan eksplorasi solusi yang sama tanpa perlu menulis ulang kode dasarnya.

a. Hill-Climbing Steepest Ascent

Berikut merupakan salah satu cara untuk mendapatkan state value dari fungsi objektif untuk permasalahan *Bin Packing*.

Nama File	
steepest_ascent.py	Implementasi dari algoritma Hill-Climbing Steepest Ascent
Variabel	
file_data	Menyimpan nama file JSON yang berisi data barang dan kapasitas kontainer
kapasitas	Kapasitas maksimum tiap kontainer
data_barang	Dictionary berisi ID barang dan ukurannya

list_barang_awal	Daftar semua barang yang akan dikemas
initial_state	Solusi awal hasil metode <i>first fit</i> , digunakan sebagai <i>starting point</i> algoritma
initial_score	Nilai <i>objective function</i> dari solusi awal
current_state	Solusi (<i>state</i>) yang sedang dievaluasi pada iterasi tertentu
current_score	Nilai <i>objective function</i> dari current_state.
best_state	Solusi terbaik yang ditemukan dari semua tetangga pada satu iterasi
best_score	Nilai <i>objective function</i> terbaik di antara semua tetangga pada satu iterasi
skor_per_iterasi	List untuk menyimpan nilai <i>objective function</i> di setiap iterasi
iterasi	Menghitung jumlah iterasi yang dilakukan algoritma
start_time, end_time	Menyimpan waktu mulai dan waktu selesai proses pencarian solusi

Fungsi dan Prosedur

utils.load_data(file_data)	Membaca file JSON berisi data kapasitas dan daftar barang, lalu mengembalikannya dalam format Python (kapasitas, data barang, list barang)
hc_utils.inisialisasi_first_fit(list_barang, kapasitas, data_barang)	Membuat solusi awal menggunakan pendekatan <i>first fit</i> dimana tiap barang dimasukkan ke kontainer pertama yang masih cukup kapasitasnya
hc_utils.calculate_objective(state, kapasitas, data_barang)	Menghitung nilai <i>objective function</i> dari suatu state berdasarkan jumlah kontainer, penalti <i>overflow</i> , dan <i>reward</i> kepadatan
hc_utils.print_state_terminal(state, kapasitas, data_barang)	Menampilkan susunan barang di tiap kontainer dalam format teks di terminal
hc_utils.get_neighbors(current_state)	Menghasilkan semua <i>neighbor</i> (tetangga) dari solusi saat ini dengan melakukan

	perpindahan barang ke kontainer lain secara sistematis
while True	<i>Loop</i> utama algoritma <i>hill climbing</i> yang berjalan sampai tidak ada solusi lebih baik (<i>local optimum</i>)
if best_score < current_score:	Mengecek apakah tetangga terbaik memiliki nilai <i>objective function</i> lebih baik dari solusi saat ini. Jika ya, pindah ke tetangga tersebut
plt.plot(skor_per_iterasi)	Membuat plot grafik perubahan nilai <i>objective function</i> terhadap jumlah iterasi
plt.savefig('steepest_ascent_plot.png')	Menyimpan grafik hasil dalam bentuk gambar

Berikut merupakan *source code* implementasi Hill-Climbing Steepest Ascent.

```

import utils
import hc_utils
import time
import copy
import matplotlib.pyplot as plt

file_data = "data_bin_packing.json"

print(f"Loading data dari {file_data}...")
kapasitas, data_barang, list_barang_awal =
utils.load_data(file_data)

print("Inisialisasi state awal (First Fit)...")
initial_state =
hc_utils.inisialisasi_first_fit(list_barang_awal, kapasitas,
data_barang)
initial_score = hc_utils.calculate_objective(initial_state,
kapasitas, data_barang)

print("\nState Awal:")
hc_utils.print_state_terminal(initial_state, kapasitas,
data_barang)
print(f"Nilai Objective Function: {initial_score:.4f}")

print("\nMulai hill climbing (Steepest Ascent)...")


current_state = copy.deepcopy(initial_state)
current_score = initial_score
skor_per_iterasi = [current_score]
iterasi = 0

```

```

start_time = time.time()

while True:
    iterasi += 1
    print(f"\nIterasi ke-{iterasi}")

    best_state = None
    best_score = float('inf')

    for neighbor in hc_utils.get_neighbors(current_state):
        score = hc_utils.calculate_objective(neighbor,
                                              kapasitas, data_barang)
        if score < best_score:
            best_score = score
            best_state = neighbor

    if not best_state:
        print("Tidak ada tetangga yang bisa digenerate.")
        break

    if best_score < current_score:
        current_state = best_state
        current_score = best_score
        skor_per_iterasi.append(current_score)
        print(f"Skor membaik: {current_score:.4f}")
        ({len(current_state)} kontainer)")
    else:
        print("Tidak ada solusi lebih baik, terjebak di local optimum.")
        skor_per_iterasi.append(current_score)
        break

end_time = time.time()

print("\nHasil Akhir:")
print(f"Durasi: {end_time - start_time:.4f} detik")
print(f"Banyak iterasi: {iterasi}")
print(f"State awal ({len(initial_state)} kontainer), nilai objective function = {initial_score:.4f}")
print(f"State akhir ({len(current_state)} kontainer), nilai objective function = {current_score:.4f}\n")

print(f"State akhir:")
hc_utils.print_state_terminal(current_state, kapasitas,
                               data_barang)

# Plot
plt.figure(figsize=(10, 6))
plt.plot(skor_per_iterasi)
plt.title('Steepest Ascent Hill Climbing: Nilai Objective Function vs Iterasi')

```

```

plt.xlabel('Iterasi')
plt.ylabel('Nilai Objective Function')
plt.grid(True)
plt.savefig('steepest_ascent_plot.png')
print("\nPlot disimpan sebagai steepest_ascent_plot.png")
plt.show()

```

Algoritma *steepest hill climbing* bekerja dengan pendekatan iteratif untuk mencari solusi terbaik dengan cara memilih tetangga (*neighbor*) yang memiliki nilai *objective function* paling baik di setiap langkah. Proses diawali dengan memuat data dari file data_bin_packing.json menggunakan fungsi `utils.load_data()`, kemudian membentuk solusi awal melalui metode *first fit* dari `hc_utils.inisialisasi_first_fit()`. Solusi awal ini akan menjadi *current state* yang akan dievaluasi.

Selanjutnya, algoritma akan menghasilkan semua *neighbor* dari *current state* dengan memanfaatkan fungsi `hc_utils.get_neighbors()`, yaitu dengan mencoba memindahkan atau menukar posisi barang antar kontainer. Setiap *neighbor* kemudian dievaluasi menggunakan `hc_utils.calculate_objective()` untuk menghitung nilai *objective function*-nya. Dari seluruh *neighbor* yang dihasilkan, algoritma memilih satu dengan skor terbaik (terkecil), karena tujuannya adalah meminimalkan nilai fungsi objektif yang mencakup penalti *overflow*, jumlah kontainer, dan *reward* kepadatan.

Jika solusi terbaik yang ditemukan memiliki nilai *objective function* yang lebih baik dibandingkan *current state*, maka algoritma akan berpindah ke *neighbor* tersebut dan melanjutkan proses pencarian. Namun, jika tidak ada *neighbor* yang memberikan hasil lebih baik, maka proses berhenti karena algoritma telah mencapai *local optimum*. Nilai terbaik di setiap iterasi disimpan dalam list `skor_per_iterasi` untuk kemudian divisualisasikan dalam grafik menggunakan `matplotlib`. Grafik tersebut menunjukkan bagaimana nilai *objective function* menurun seiring bertambahnya iterasi hingga algoritma berhenti pada solusi optimal lokal.

b. Hill-Climbing with Sideways Move

Berikut merupakan salah satu cara untuk mendapatkan state value dari fungsi objektif untuk permasalahan *Bin Packing*.

Nama File	
sideways_move.py	Implementasi dari algoritma Hill-Climbing with Sideways Move
Variabel	

file_data	Menyimpan nama file JSON yang berisi data barang dan kapasitas kontainer
MAX_SIDEWAYS_MOVE	Batas maksimum jumlah <i>sideways move</i> yang diizinkan sebelum algoritma berhenti
kapasitas	Kapasitas maksimum tiap kontainer
data_barang	Dictionary berisi ID barang dan ukurannya
list_barang_awal	Daftar semua barang yang akan dikemas
initial_state	State awal hasil inisialisasi menggunakan metode <i>first fit</i>
initial_score	Nilai <i>objective function</i> dari solusi awal
current_state	Solusi (<i>state</i>) yang sedang dievaluasi pada iterasi tertentu
current_score	Nilai <i>objective function</i> dari current_state.
best_state	Menyimpan tetangga terbaik yang ditemukan pada iterasi tertentu
best_score	Nilai <i>objective function</i> dari tetangga terbaik
skor_per_iterasi	Menyimpan perkembangan nilai <i>objective function</i> setiap iterasi
iterasi	Menghitung jumlah iterasi yang dilakukan algoritma
sideways_moves_count	Menghitung jumlah <i>sideways move</i> berturut-turut yang sudah dilakukan
start_time, end_time	Waktu mulai dan selesai eksekusi algoritma (untuk menghitung durasi)
Fungsi dan Prosedur	
utils.load_data(file_data)	Membaca file JSON berisi data kapasitas dan daftar barang, lalu mengembalikannya dalam format Python (kapasitas, data barang, list barang)
hc_utils.inisialisasi_first_fit()	Membuat <i>state</i> awal dengan menempatkan barang ke kontainer pertama yang masih muat

hc_utils.calculate_objective()	Menghitung nilai <i>objective function</i> dari sebuah <i>state</i> dengan mempertimbangkan jumlah kontainer, penalti <i>overload</i> , dan reward kepadatan
hc_utils.get_neighbors()	Menghasilkan semua kemungkinan state tetangga dari <i>current_state</i> dengan cara memindahkan atau menukar barang antar kontainer
hc_utils.print_state_terminal()	Menampilkan konfigurasi kontainer dan isinya di terminal untuk keperluan <i>debugging</i> atau analisis
while True	Melakukan proses iteratif pencarian solusi terbaik berdasarkan prinsip <i>hill climbing</i>
if best_score < current_score	Jika tetangga lebih baik, algoritma berpindah ke tetangga tersebut dan mengatur ulang <i>sideways count</i>
elif best_score == current_score	Jika nilai sama, maka dianggap <i>sideways move</i> , boleh dilakukan hingga mencapai MAX_SIDeways_MOVES
else	Jika tidak ada solusi yang lebih baik, algoritma berhenti karena mencapai <i>local optimum</i>
matplotlib (plot hasil)	Menampilkan grafik hubungan antara nilai <i>objective function</i> dan jumlah iterasi untuk memvisualisasikan konvergensi algoritma

Berikut merupakan *source code* implementasi Hill-Climbing with Sideways Move.

```

import utils
import hc_utils
import time
import copy
import matplotlib.pyplot as plt

file_data = "data_bin_packing.json"
MAX_SIDWAYS_MOVES = 50

print(f"Loading data dari {file_data}...")
kapasitas, data_barang, list_barang_awal =
utils.load_data(file_data)

```

```

print("Inisialisasi state awal (First Fit)...")
initial_state =
hc_utils.inisialisasi_first_fit(list_barang_awal, kapasitas,
data_barang)
initial_score = hc_utils.calculate_objective(initial_state,
kapasitas, data_barang)

print("\nState Awal:")
hc_utils.print_state_terminal(initial_state, kapasitas,
data_barang)
print(f"Nilai Objective Function: {initial_score:.4f}")

print("\nMulai hill climbing (Sideways Move)...")


current_state = copy.deepcopy(initial_state)
current_score = initial_score
skor_per_iterasi = [current_score]
iterasi = 0
start_time = time.time()

sideways_moves_count = 0

while True:
    iterasi += 1
    print(f"\nIterasi ke-{iterasi} (Sideways count:
{sideways_moves_count}/{MAX_SIDEWAYS_MOVES})")

    best_state = None
    best_score = float('inf')

    for neighbor in hc_utils.get_neighbors(current_state):
        score = hc_utils.calculate_objective(neighbor,
kapasitas, data_barang)
        if score < best_score:
            best_score = score
            best_state = neighbor

    if not best_state:
        print("Tidak ada tetangga yang bisa digenerate.")
        break

    if best_score < current_score:
        current_state = best_state
        current_score = best_score
        skor_per_iterasi.append(current_score)

        sideways_moves_count = 0

        print(f"Skor membaik: {current_score:.4f}
({len(current_state)} kontainer)")

```

```

        elif best_score == current_score:
            if sideways_moves_count >= MAX_SIDEWAYS_MOVES:
                print(f"Batas sideways move ({MAX_SIDEWAYS_MOVES}) tercapai. Berhenti di dataran.")
                skor_per_iterasi.append(current_score)
                break
            else:
                current_state = best_state
                current_score = best_score
                skor_per_iterasi.append(current_score)

                sideways_moves_count += 1

                print(f"Melakukan SIDEWAYS MOVE (Count: {sideways_moves_count}/{MAX_SIDEWAYS_MOVES}). Skor tetap.")

            else:
                print("Tidak ada solusi lebih baik, terjebak di local optimum.")
                skor_per_iterasi.append(current_score)
                break

        end_time = time.time()

        print("\nHasil akhir:")
        print(f"Durasi: {end_time - start_time:.4f} detik")
        print(f"Banyak iterasi: {iterasi}")
        print(f"State awal ({len(initial_state)} kontainer), nilai objective function = {initial_score:.4f}")
        print(f"State akhir ({len(current_state)} kontainer), nilai objective function = {current_score:.4f}\n")

        print("State Akhir:")
        hc_utils.print_state_terminal(current_state, kapasitas, data_barang)

    # Plot
    plt.figure(figsize=(10, 6))
    plt.plot(skor_per_iterasi)
    plt.title('Hill Climbing with Sideways Move: Nilai Objective Function vs Iterasi')
    plt.xlabel('Iterasi')
    plt.ylabel('Nilai Objective Function')
    plt.grid(True)
    plt.savefig('sideways_move_plot.png')
    print("\nPlot disimpan sebagai sideways_move_plot.png")
    plt.show()

```

Algoritma *hill-climbing with sideways move* merupakan pengembangan dari metode *Steepest Ascent Hill Climbing* yang mengizinkan perpindahan ke solusi dengan nilai *objective function* yang sama, bukan hanya yang lebih baik. Tujuannya adalah untuk menghindari kondisi *plateau* (dataran) pada ruang pencarian, di mana banyak solusi memiliki nilai yang sama sehingga algoritma konvensional akan langsung berhenti.

Proses dimulai dengan memuat data dari file `data_bin_packing.json`, lalu membuat *initial state* menggunakan fungsi `hc_utils.inisialisasi_first_fit()`. Setelah nilai awal dihitung menggunakan `hc_utils.calculate_objective()`, algoritma mulai melakukan iterasi pencarian. Pada setiap iterasi, semua *neighbor* dari *current state* dihasilkan melalui `hc_utils.get_neighbors()`, kemudian dipilih yang memiliki nilai *objective function* paling baik. Jika nilai terbaik lebih kecil dari keadaan saat ini, algoritma berpindah ke *neighbor* tersebut dan menghitung ulang nilai objektifnya.

Namun, jika nilai terbaik sama dengan *current score*, algoritma melakukan *sideways move*, yaitu berpindah ke solusi dengan nilai yang sama, selama belum melebihi batas maksimum `MAX_SIDEWAYS_MOVES`. Variabel `sideways_moves_count` digunakan untuk melacak berapa kali langkah *sideways* telah dilakukan. Jika batasnya tercapai, algoritma berhenti karena dianggap terjebak di dataran luas tanpa perbaikan. Sebaliknya, jika ditemukan solusi yang lebih buruk, proses langsung berhenti karena mencapai *local optimum*. Nilai *objective function* pada setiap iterasi disimpan di `skor_per_iterasi` dan divisualisasikan dengan grafik menggunakan `matplotlib`, yang menunjukkan dinamika perbaikan solusi hingga berhenti pada titik optimum lokal.

c. Random Restart Hill-Climbing

Berikut merupakan salah satu cara untuk mendapatkan state value dari fungsi objektif untuk permasalahan *Bin Packing*.

Nama File	
random_restart.py	Implementasi dari algoritma Random Restart Hill-Climbing
Variabel	
file_data	Menyimpan nama file JSON yang berisi data barang dan kapasitas kontainer
MAX_RESTARTS	Jumlah maksimum <i>restart</i> yang diizinkan dalam algoritma
kapasitas	Kapasitas maksimum tiap kontainer

data_barang	Menyimpan pasangan ID dan ukuran masing-masing barang
list_barang_awal	Daftar semua barang yang akan dikemas
restart_counter	Menghitung berapa kali proses <i>restart</i> sudah dilakukan
current_state	<i>State</i> aktif yang sedang dievaluasi di dalam satu siklus <i>hill climbing</i>
current_score	Nilai <i>objective function</i> dari current_state.
best_overall_state	Menyimpan solusi terbaik dari semua percobaan (<i>restart</i>)
best_overall_score	Nilai <i>objective function</i> terbaik dari seluruh <i>restart</i>
skor_per_iterasi	Menyimpan perubahan skor selama iterasi dan <i>restart</i> berlangsung
iterasi	Menghitung total jumlah iterasi dari seluruh proses <i>restart</i>
start_time, end_time	Waktu mulai dan selesai eksekusi algoritma (untuk menghitung durasi)
Fungsi dan Prosedur	
utils.load_data()	Membaca file JSON dan mengembalikan kapasitas kontainer, data barang, serta daftar barang awal.
hc_utils.inisialisasi_random()	Membuat state awal secara acak dengan metode <i>first fit</i> untuk memulai setiap <i>restart</i>
hc_utils.calculate_objective()	Menghitung nilai <i>objective function</i> dari sebuah <i>state</i> dengan mempertimbangkan jumlah kontainer, penalti <i>overload</i> , dan reward kepadatan
hc_utils.get_neighbors()	Menghasilkan semua tetangga potensial dari <i>state</i> saat ini untuk eksplorasi lokal
hc_utils.print_state_terminal()	Menampilkan hasil konfigurasi kontainer dalam bentuk yang mudah dibaca di terminal
for restart in	Mengulang proses <i>hill climbing</i> dengan <i>state</i>

range(MAX_RESTARTS)	awal baru yang diacak setiap kali
Kondisi perbandingan hasil	Setelah setiap <i>restart</i> selesai, nilai solusi dibandingkan dengan <code>best_overall_score</code> . Jika lebih baik, disimpan sebagai solusi terbaik global
matplotlib (plot hasil)	Menampilkan grafik hubungan antara nilai <i>objective function</i> dan jumlah iterasi untuk memvisualisasikan konvergensi algoritma

Berikut merupakan *source code* implementasi Random Restart Hill-Climbing.

```

import utils
import hc_utils
import time
import copy
import matplotlib.pyplot as plt

file_data = "data_bin_packing.json"
MAX_RESTARTS = 10

print(f"Akan menjalankan pencarian sebanyak {MAX_RESTARTS} kali restart.")

kapasitas, data_barang, list_barang_awal =
utils.load_data(file_data)

def run_steepest_ascent_inner_loop(start_state, kapasitas,
data_barang):
    current_state = copy.deepcopy(start_state)
    current_score =
hc_utils.calculate_objective(current_state, kapasitas,
data_barang)

    skor_history = [current_score]
    iterasi = 0
    berhenti = False

    while not berhenti:
        iterasi += 1

        best_neighbor_state = None
        best_neighbor_score = float('inf')

        neighbor_generator =
hc_utils.get_neighbors(current_state)
        for neighbor in neighbor_generator:
            neighbor_score =

```

```

hc_utils.calculate_objective(neighbor, kapasitas, data_barang)
    if neighbor_score < best_neighbor_score:
        best_neighbor_score = neighbor_score
        best_neighbor_state = neighbor

    if best_neighbor_state is None:
        berhenti = True
    elif best_neighbor_score < current_score:
        current_state = best_neighbor_state
        current_score = best_neighbor_score
        skor_history.append(current_score)
    else:
        skor_history.append(current_score)
        berhenti = True

    return current_state, current_score, skor_history, iterasi

start_time = time.time()

global_best_score = float('inf')
global_best_final_state = None
global_best_skor_history = []

first_run_initial_state = None
first_run_initial_score = 0

list_iterasi_per_restart = []
skor_history_gabungan = []

for r in range(MAX_RESTARTS):
    print(f"\nRestart ke-{r+1}/{MAX_RESTARTS}")

    initial_state_random =
hc_utils.inisialisasi_random(list_barang_awal, kapasitas,
data_barang)
    initial_score_random =
hc_utils.calculate_objective(initial_state_random, kapasitas,
data_barang)
    print(f"State awal acak: {len(initial_state_random)}\nkontainer, Skor: {initial_score_random:.2f}")

    if r == 0:
        first_run_initial_state =
copy.deepcopy(initial_state_random)
        first_run_initial_score = initial_score_random

    final_state, final_score, skor_history, total_iterasi = \
        run_steepest_ascent_inner_loop(initial_state_random,
kapasitas, data_barang)

    print(f"Selesai di iterasi ke-{total_iterasi}. Skor lokal"

```

```

optimum: {final_score:.2f}")

    list_iterasi_per_restart.append(total_iterasi)
    skor_history_gabungan.extend(skor_history)

    if final_score < global_best_score:
        print(f"DITEMUKAN SOLUSI BARU TERBAIK! (Skor: {final_score:.2f})")
        global_best_score = final_score
        global_best_final_state = final_state
        global_best_skor_history = skor_history

end_time = time.time()

print("Hasil akhir:")
print(f"Durasi Total: {end_time - start_time:.4f} detik")
print(f"Total Restart Dilakukan: {MAX_RESTARTS}")
print(f"Rata-rata Iterasi per Restart: {sum(list_iterasi_per_restart) / len(list_iterasi_per_restart):.2f}")

print("\nState Awal:")
hc_utils.print_state_terminal(first_run_initial_state, kapasitas, data_barang)
print(f"Nilai Objective Function: {first_run_initial_score:.4f}")

print("\nState Akhir:")
hc_utils.print_state_terminal(global_best_final_state, kapasitas, data_barang)
print(f"Skor Akhir Terbaik: {global_best_score}")

# Plot
print("\nMembuat plot skor (dari run terbaik)...")
try:
    plt.figure(figsize=(10, 6))
    plt.plot(global_best_skor_history)
    plt.title('Random Restart: Nilai Objective Function vs Iterasi (Run Terbaik)')
    plt.xlabel('Iterasi (dalam 1 run)')
    plt.ylabel('Nilai Objective Function')
    plt.grid(True)

    nama_file_plot = 'random_restart_plot_skor.png'
    plt.savefig(nama_file_plot)
    print(f"Plot skor berhasil disimpan sebagai: {nama_file_plot}")
    plt.show()

except Exception as e:
    print(f"Gagal membuat plot skor: {e}")

```

```

print("\nMembuat plot analisis (iterasi per restart)...")
try:
    plt.figure(figsize=(10, 6))
    restart_labels = [f"R{i+1}" for i in range(MAX_RESTARTS)]
    plt.bar(restart_labels, list_iterasi_per_restart,
color='coral')
    plt.title('Analisis Random Restart: Banyak Iterasi per
Restart')
    plt.xlabel('Nomor Restart')
    plt.ylabel('Banyak Iterasi (sampai lokal optimum)')
    plt.bar_label(plt.gca().containers[0])

    nama_file_plot = 'random_restart_plot_analisis.png'
    plt.savefig(nama_file_plot)
    print(f"Plot analisis berhasil disimpan sebagai:
{nama_file_plot}")
    plt.show()

except Exception as e:
    print(f"Gagal membuat plot analisis: {e}")

print("\nMembuat plot gabungan...")
try:
    plt.figure(figsize=(15, 6))
    plt.plot(skor_history_gabungan, linestyle='--',
linewidth=0.8)
    plt.title('Analisis Random Restart: Plot Gabungan')
    plt.xlabel('Total Iterasi Gabungan (dari semua restart)')
    plt.ylabel('Nilai Objective Function')
    plt.grid(True, linestyle=':', alpha=0.7)

    nama_file_plot = 'random_restart_plot_gabungan.png'
    plt.savefig(nama_file_plot)
    print(f"Plot gabungan berhasil disimpan sebagai:
{nama_file_plot}")
    plt.show()

except Exception as e:
    print(f"Gagal membuat plot gabungan: {e}")

print("\nMembuat plot konvergensi (Best vs Iterasi)...")
try:
    best_so_far_list = []
    current_best = float('inf')

    for score in skor_history_gabungan:
        if score < current_best:
            current_best = score
    best_so_far_list.append(current_best)

```

```

plt.figure(figsize=(15, 6))
plt.plot(best_so_far_list, linestyle='-', linewidth=1.5,
color='blue',
label='Best Objective Function Value',
drawstyle='steps-post')

plt.title('Random Restart: Best Score vs Total Iterasi
(Konvergensi)')
plt.xlabel('Total Iterasi Gabungan (dari semua restart)')
plt.ylabel('Best Objective Function Value')
plt.grid(True)
plt.legend()

nama_file_plot = 'random_restart_plot_konvergensi.png'
plt.savefig(nama_file_plot)
print(f"Plot berhasil disimpan sebagai: {nama_file_plot}")
plt.show()

except Exception as e:
    print(f"Gagal membuat plot 'konvergensi': {e}")

```

Algoritma *random restart hill-climbing* merupakan pengembangan dari metode *steepest hill-climbing* yang bertujuan untuk mengatasi permasalahan terjebak pada *local optimum*. Prinsip utamanya adalah menjalankan algoritma *hill-climbing* berulang kali dengan titik awal yang berbeda secara acak (*random initial states*), sehingga peluang untuk menemukan solusi global yang lebih baik meningkat.

Proses diawali dengan memuat data dari file `data_bin_packing.json` menggunakan `utils.load_data()`, kemudian algoritma melakukan beberapa kali *restart* sebanyak `MAX_RESTARTS`. Pada setiap *restart*, solusi awal dihasilkan secara acak menggunakan fungsi `hc_utils.inisialisasi_random()`, lalu dievaluasi dengan `hc_utils.calculate_objective()` untuk mendapatkan nilai awal fungsi objektif. Setelah itu, fungsi `run_steepest_ascent_inner_loop()` dijalankan untuk melakukan proses *steepest hill-climbing* dari titik awal tersebut hingga mencapai *local optimum*.

Di dalam setiap *inner loop*, algoritma mencari tetangga terbaik dari *current state* dengan `hc_utils.get_neighbors()`, memilih solusi dengan skor terendah, dan memperbarui keadaan jika terjadi perbaikan nilai *objective function*. Ketika tidak ada lagi tetangga yang lebih baik, proses *hill-climbing* pada *restart* tersebut berhenti, dan hasilnya dibandingkan dengan solusi terbaik global. Jika skor dari hasil *restart* saat ini lebih baik, maka nilai dan *state* tersebut disimpan sebagai solusi terbaik keseluruhan (`global_best_score` dan `global_best_final_state`).

Setelah seluruh *restart* selesai, algoritma menampilkan hasil akhir yang mencakup durasi total, rata-rata iterasi per *restart*, serta menampilkan *state* awal dan akhir terbaik.

Proses juga menghasilkan beberapa visualisasi, seperti plot perkembangan skor per iterasi, analisis jumlah iterasi tiap *restart*, serta grafik konvergensi nilai terbaik secara keseluruhan. Dengan pendekatan ini, algoritma *random restart hill-climbing* mampu memperluas pencarian solusi dan meningkatkan peluang untuk mendekati *global optimum* dibandingkan *hill-climbing* biasa yang cenderung berhenti terlalu dini di *local optimum*.

d. Stochastic Hill-Climbing

Berikut merupakan salah satu cara untuk mendapatkan state value dari fungsi objektif untuk permasalahan *Bin Packing*.

Nama File	
stochastic.py	Implementasi dari algoritma Stochastic Hill-Climbing
Variabel	
file_data	Menyimpan nama file JSON yang berisi data barang dan kapasitas kontainer
kapasitas	Kapasitas maksimum tiap kontainer
data_barang	Menyimpan pasangan ID dan ukuran masing-masing barang
list_barang_awal	Daftar semua barang yang akan dikemas
initial_state	State awal hasil inisialisasi menggunakan metode <i>first fit</i>
initial_score	Nilai <i>objective function</i> dari solusi awal
current_state	Solusi (<i>state</i>) yang sedang dievaluasi pada iterasi tertentu
current_score	Nilai <i>objective function</i> dari current_state.
skor_per_iterasi	Menyimpan riwayat perkembangan nilai <i>objective function</i> setiap iterasi
iterasi	Menghitung jumlah iterasi yang telah dilakukan selama proses algoritma berjalan
better_neighbors	Menyimpan semua tetangga yang memiliki nilai <i>objective function</i> lebih baik dari state saat ini

neighbor_generator	Menghasilkan semua kandidat tetangga dari <code>current_state</code> menggunakan fungsi <code>get_neighbors()</code>
selected_neighbor	Salah satu tetangga yang dipilih secara acak dari daftar <code>better_neighbors</code>
selected_score	Nilai <i>objective function</i> dari <code>selected_neighbor</code>
start_time, end_time	Waktu mulai dan selesai eksekusi algoritma (untuk menghitung durasi)
Fungsi dan Prosedur	
<code>utils.load_data()</code>	Membaca file JSON dan mengembalikan kapasitas, data barang, serta daftar barang awal
<code>hc_utils.inisialisasi_first_fit()</code>	Menginisialisasi solusi awal menggunakan pendekatan <i>first fit</i> , menempatkan barang ke kontainer pertama yang masih muat
<code>hc_utils.calculate_objective()</code>	Menghitung nilai <i>objective function</i> yang mencakup penalti <i>overload</i> , jumlah kontainer, dan reward kepadatan
<code>hc_utils.get_neighbors()</code>	Menghasilkan semua <i>state</i> tetangga yang dapat terbentuk dengan memindahkan atau menukar barang antar kontainer
<code>hc_utils.print_state_terminal()</code>	Menampilkan konfigurasi kontainer dan barang di terminal dengan penanda jika terjadi <i>overload</i>
<code>while True</code>	Melakukan proses iterasi untuk mencari solusi yang lebih baik hingga tidak ada lagi tetangga dengan skor lebih rendah
<code>random.choice</code> (pemilihan tetangga acak)	Dari kumpulan tetangga yang lebih baik, satu dipilih secara acak untuk menggantikan <code>current_state</code>
<code>if not better_neighbors</code> (kondisi berhenti)	Jika tidak ditemukan tetangga yang lebih baik, algoritma berhenti karena mencapai <i>local optimum</i> .
<code>matplotlib</code>	Menampilkan grafik hubungan antara nilai

objective function dan jumlah iterasi untuk memvisualisasikan konvergensi algoritma

Berikut merupakan *source code* implementasi Stochastic Hill-Climbing.

```
import utils
import hc_utils
import time
import copy
import random
import matplotlib.pyplot as plt

file_data = "data_bin_packing.json"

print(f"Loading data dari {file_data}...")
kapasitas, data_barang, list_barang_awal =
utils.load_data(file_data)

print("Inisialisasi state awal (First Fit)...")
initial_state =
hc_utils.inisialisasi_first_fit(list_barang_awal, kapasitas,
data_barang)
initial_score = hc_utils.calculate_objective(initial_state,
kapasitas, data_barang)

print("\nState Awal:")
hc_utils.print_state_terminal(initial_state, kapasitas,
data_barang)
print(f"Nilai Objective Function: {initial_score:.4f}")

print("\nMulai hill climbing (Stochastic)...")


current_state = copy.deepcopy(initial_state)
current_score = initial_score
skor_per_iterasi = [current_score]
iterasi = 0
start_time = time.time()

while True:
    iterasi += 1
    print(f"\nIterasi ke-{iterasi}")

    better_neighbors = []

    neighbor_generator = hc_utils.get_neighbors(current_state)

    for neighbor in neighbor_generator:
        neighbor_score =
hc_utils.calculate_objective(neighbor, kapasitas, data_barang)
```

```

        if neighbor_score < current_score:
            better_neighbors.append((neighbor,
neighbor_score))

        if not better_neighbors:
            print("Tidak ada solusi lebih baik, terjebak di local
optimum.")
            skor_per_iterasi.append(current_score)
            break
        else:
            selected_neighbor, selected_score =
random.choice(better_neighbors)

            current_state = selected_neighbor
            current_score = selected_score

            skor_per_iterasi.append(current_score)

            print(f"Ditemukan {len(better_neighbors)} tetangga
lebih baik. Memilih 1 secara acak.")
            print(f"Skor membaik: {current_score:.4f}
({len(current_state)} kontainer)")

end_time = time.time()

print("\nHasil akhir:")
print(f"Durasi: {end_time - start_time:.4f} detik")
print(f"Banyak iterasi: {iterasi}")
print(f"State awal ({len(initial_state)} kontainer), nilai
objective function = {initial_score:.4f}")
print(f"State akhir ({len(current_state)} kontainer), nilai
objective function = {current_score:.4f}\n")

print("State Akhir:")
hc_utils.print_state_terminal(current_state, kapasitas,
data_barang)

# Plot
plt.figure(figsize=(10, 6))
plt.plot(skor_per_iterasi)
plt.title('Stochastic Hill Climbing: Nilai Objective Function
vs Iterasi')
plt.xlabel('Iterasi')
plt.ylabel('Nilai Objective Function')
plt.grid(True)
plt.savefig('stochastic_hc_plot.png')
print(f"\nPlot disimpan sebagai stochastic_hc_plot.png")
plt.show()

```

Algoritma *stochastic hill-climbing* merupakan varian dari *hill-climbing* yang memperkenalkan unsur acak (*stochastic element*) dalam pemilihan solusi tetangga. Berbeda dengan *steepest hill-climbing* yang selalu memilih tetangga terbaik, algoritma ini memilih satu tetangga secara acak dari sekumpulan tetangga yang memiliki nilai *objective function* lebih baik dibandingkan solusi saat ini. Tujuannya adalah untuk menghindari pola pencarian yang deterministik dan membantu algoritma keluar dari jebakan *local optimum* kecil yang mungkin terjadi di awal pencarian.

Proses dimulai dengan memuat data dari `data_bin_packing.json` menggunakan `utils.load_data()`, kemudian membentuk *initial state* dengan metode *first fit* melalui `hc_utils.inisialisasi_first_fit()`. Nilai fungsi objektif awal dihitung menggunakan `hc_utils.calculate_objective()`. Selanjutnya, algoritma menghasilkan semua *neighbor* dari *current state* menggunakan `hc_utils.get_neighbors()`, lalu mengevaluasi tiap tetangga. Tetangga yang memiliki nilai fungsi objektif lebih baik dari keadaan sekarang dimasukkan ke dalam daftar `better_neighbors`.

Jika tidak ada tetangga yang lebih baik, algoritma berhenti karena telah mencapai *local optimum*. Namun, jika terdapat beberapa tetangga yang lebih baik, satu di antaranya dipilih secara acak menggunakan `random.choice()`, lalu dijadikan sebagai *current state* yang baru. Proses ini diulang hingga tidak ditemukan perbaikan. Nilai *objective function* dari setiap iterasi disimpan di `skor_per_iterasi` dan divisualisasikan menggunakan *matplotlib* untuk menunjukkan dinamika perbaikan solusi.

e. Simulated Annealing

Berikut merupakan salah satu cara untuk mendapatkan state value dari fungsi objektif untuk permasalahan *Bin Packing*.

Nama File	
simulated_annealing.py	Implementasi dari algoritma Simulated Annealing
Variabel	
CONTAINER_CAPACITY	Kapasitas maksimum tiap kontainer (bin). Dimuat dari file JSON.
ITEM_DATA	Data pasangan ID dan ukuran masing-masing barang yang akan dikemas. Dimuat dari file JSON.
OVERFLOW_PENALTY	Nilai penalti yang sangat besar jika total ukuran barang melebihi

	CONTAINER_CAPACITY. Digunakan untuk menolak solusi yang tidak valid
DENSITY_ALPHA	Bobot yang diberikan pada sisa kapasitas kontainer.
BIN_COUNT_FACTOR	Bobot yang diberikan pada jumlah kontainer yang digunakan.
TEMP_START	Suhu awal Simulated Annealing yang menentukan energi eksplorasi di awal proses.
COOLING_RATE	Laju pendinginan suhu yang menentukan seberapa cepat eksplorasi menurun menjadi eksloitasi.
MAX_ITER	Jumlah maksimum iterasi yang akan dijalankan algoritma sebelum berhenti.
state_initial; state_current; state_next; state_best	Solusi (state) awal; sedang dievaluasi; tetangga; dan terbaik yang pernah ditemukan
f_current; f_next; f_best	Nilai objective function dari state
delta_E	Perbedaan skor objective function neighbor dan current
temp_current	Suhu pada iterasi saat ini, berkurang di setiap iterasi
prob_acceptance	Probabilitas penerimaan langkah buruk, dihitung dengan $e^{\frac{\Delta E}{T}}$
accepted_worse_moves	Jumlah langkah yang memiliki skor lebih buruk ($\Delta E > 0$) namun diterima secara probabilistik
history	Daftar (list) yang menyimpan riwayat skor dan suhu di setiap iterasi untuk keperluan plotting
Fungsi dan Prosedur	
first_fit()	Heuristik initial state; menempatkan barang ke kontainer pertama yang muat
objective_function(state)	Menghitung nilai fungsi objektif berdasarkan

	jumlah kontainer, kepadatan (alpha), dan penalti overload
generate_neighbor(state)	Menghasilkan state tetangga dari state saat ini melalui move atau swap
simulated_annealing(state_initial, temp_start, cooling_rate, max_iter)	Fungsi utama yang mengelola proses pendinginan (Suhu T) dan probabilitas penerimaan move buruk (ΔE)
print_state(state, state_name)	Mencetak alokasi barang per kontainer, total ukuran, dan skor fungsi objektif
plot_sa(history_data, run_id)	Membuat dua grafik (objective function terhadap iterasi dan $e^{\frac{\Delta E}{T}}$ terhadap iterasi) untuk analisis
if __name__ == "__main__"	Kode utama yang dieksekusi saat file dijalankan. Bertugas untuk: (1) Memuat data via utils.load_data, (2) Mengatur parameter Simulated Annealing, (3) Membuat state awal, (4) Mencatat waktu dan menjalankan algoritma, (5) Mencetak hasil (State Awal, Akhir, Durasi), (6) Membuat plot matplotlib.

Berikut merupakan *source code* implementasi Simulated Annealing

```

import math
import random
import time
import utils
import matplotlib.pyplot as plt

CONTAINER_CAPACITY, ITEM_DATA, _ =
utils.load_data('data_bin_packing.json')

# Parameter Objective Function
OVERFLOW_PENALTY = 10**4
DENSITY_ALPHA = 1.0
BIN_COUNT_FACTOR = 100

# Parameter Simulated Annealing
TEMP_START = 100.0
COOLING_RATE = 0.99999

```

```

MAX_ITER = 500000

class BinPackingSA:
    def __init__(self, containers):
        self.containers = containers
        self.item_data = DATA_BARANG

    def get_container_size(self, container_index):
        total = 0
        for item_id in self.containers[container_index]:
            total += self.item_data.get(item_id, 0)
        return total

    def __len__(self):
        return len(self.containers)

    def copy(self):
        return BinPackingSA([list(c) for c in self.containers])

    def first_fit(): # Metode heuristic first fit untuk penentuan
state awal
        items = list(DATA_BARANG.items())
        random.shuffle(items)
        containers = []

        for item_id, size in items:
            placed = False
            for c_idx in range(len(containers)):
                current_size = sum(DATA_BARANG[i] for i in
containers[c_idx])
                if current_size + size <= CONTAINER_CAPACITY:
                    containers[c_idx].append(item_id)
                    placed = True
                    break

            if not placed:
                containers.append([item_id])

        return BinPackingSA(containers)

    def objective_function(state: BinPackingSA):
        invalid_penalti = 0
        for i in range(len(state)):
            total_size = state.get_container_size(i)
            if total_size > CONTAINER_CAPACITY:
                invalid_penalti += (total_size -
CONTAINER_CAPACITY) * OVERFLOW_PENALTY

        skor1 = len(state) * BIN_COUNT_FACTOR

```

```

skor2 = 0

for i in range(len(state)):
    total_size = state.get_container_size(i)
    sisa_kapasitas = CONTAINER_CAPACITY - total_size
    skor2 += sisa_kapasitas * DENSITY_ALPHA

f_total = skor1 + skor2 + invalid_penalty
return f_total

def generate_neighbor(state: BinPackingSA):
    state_next = state.copy()
    if len(state_next) == 0:
        return state_next

    all_items = [(c_idx, item_id) for c_idx, c in
enumerate(state_next.containers) for item_id in c]

    if not all_items:
        return state_next

    if random.random() < 0.5:
        from_container_idx, item_id = random.choice(all_items)

    state_next.containers[from_container_idx].remove(item_id)

        if random.random() < 0.8 and len(state_next) > 0:
            to_container_idx =
random.randrange(len(state_next))
        else:
            state_next.containers.append([])
            to_container_idx = len(state_next) - 1

    state_next.containers[to_container_idx].append(item_id)

    else:
        if len(state_next) < 2 or len(all_items) < 2:
            return state_next

        item1_loc, item2_loc = random.sample(all_items, 2)
        idx1, item1 = item1_loc
        idx2, item2 = item2_loc

        if idx1 != idx2:
            state_next.containers[idx1].remove(item1)
            state_next.containers[idx2].remove(item2)
            state_next.containers[idx1].append(item2)
            state_next.containers[idx2].append(item1)

    state_next.containers = [c for c in state_next.containers]

```

```

if c]
    return state_next

def simulated_annealing(state_initial: BinPackingSA,
temp_start, cooling_rate, max_iter):
    state_current = state_initial.copy()
    temp_current = temp_start
    f_current = objective_function(state_current)

    state_best = state_current.copy()
    f_best = f_current

    history = []
    accepted_worse_moves = 0

    start_time = time.time()

    for iterasi in range(1, max_iter + 1):
        state_next = generate_neighbor(state_current)
        f_next = objective_function(state_next)
        delta_E = f_next - f_current

        prob_acceptance = 0.0
        is_accepted = False

        if delta_E < 0:
            is_accepted = True
            prob_acceptance = 1.0
        elif temp_current > 0:
            prob_acceptance = math.exp(-delta_E /
temp_current)
            if random.random() < prob_acceptance:
                is_accepted = True
                accepted_worse_moves += 1

        if is_accepted:
            state_current = state_next
            f_current = f_next

        if f_next < f_best:
            state_best = state_next.copy()
            f_best = f_next
        elif f_current < f_best:
            state_best = state_current.copy()
            f_best = f_current

        temp_current *= cooling_rate

        history.append({
            'iterasi': iterasi,
            'f_current': f_current,

```

```

        'f_best_so_far': f_best,
        'T': temp_current,
        'Prob_Acceptance': prob_acceptance
    })

end_time = time.time()
durasi = end_time - start_time

return state_best, f_best, durasi, history,
accepted_worse_moves

def print_state(state: BinPackingSA, state_name):
    print(f"\n--- {state_name} ({len(state)} Kontainer) ---")

    for i, container in enumerate(state.containers):
        total_size = state.get_container_size(i)

        # Menampilkan status validity jika over capacity
        status = "OK"
        if total_size > CONTAINER_CAPACITY:
            status = "OVERLOAD!"

        item_details = [f"{item_id}"
        ({state.item_data.get(item_id, '?')})" for item_id in
        container]
        print(f"Kontainer {i+1} [{status}]: Total =
{total_size}/{CONTAINER_CAPACITY}. Barang: {',
        '.join(item_details)}")

    print(f"Objective Function Score:
{objective_function(state):.4f}")

def plot_sa(history_data, run_id):
    iterasi = [d['iterasi'] for d in history_data]
    f_current = [d['f_current'] for d in history_data]
    f_best_so_far = [d['f_best_so_far'] for d in history_data]
    prob_acceptance = [d['Prob_Acceptance'] for d in
    history_data]
    temp_data = [d['T'] for d in history_data]

    fig, axes = plt.subplots(2, 1, figsize=(12, 10))
    fig.suptitle(f'Hasil Simulated Annealing (Run {run_id})',
    fontsize=16)

    # Plot 1: Objective Function vs. Iterasi
    axes[0].plot(iterasi, f_current, label='f(S) Current',
    alpha=0.6)
    axes[0].plot(iterasi, f_best_so_far, label='f(S) Terbaik',
    color='red', linewidth=2)
    axes[0].set_ylabel('Nilai Objective Function')
    axes[0].set_title('Plot Nilai Objective Function terhadap')

```

```

Iterasi')
    axes[0].legend()
    axes[0].grid(True, linestyle='--')

    # Plot 2: Probabilitas Penerimaan vs. Iterasi
    axes[1].plot(iterasi, prob_acceptance, label='Probabilitas
Penerimaan', alpha=0.5)
    axes[1].set_xlabel('Iterasi')
    axes[1].set_ylabel('Nilai ($e^{-\Delta E / T}$)')
    axes[1].set_title('Plot ($e^{-\Delta E / T}$) terhadap
Iterasi')

    ax2_twin = axes[1].twinx()
    ax2_twin.plot(iterasi, temp_data, label='Suhu (T)',
color='red', linestyle=':')
    ax2_twin.set_ylabel('Suhu (T)')
    ax2_twin.legend(loc='upper right')

    axes[1].grid(True, linestyle='--')
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

'''----- EKSEKUSI UTAMA -----'
program dijalankan tiga kali'''
if __name__ == "__main__":
    for run in range(1, 4):
        print(f"\nEKSPERIMEN RUN #{run}")

        state_initial = first_fit()
        print_state(state_initial, "State Awal")

        best_solution, final_f_best, duration, history_data,
stuck_counter = simulated_annealing(
            state_initial.copy(), TEMP_START, COOLING_RATE,
MAX_ITER
        )

        print("\n--- Ringkasan Hasil ---")
        print(f"Objective Function Awal:
{objective_function(state_initial):.4f}")
        print(f"Objective Function Akhir: {final_f_best:.4f}")
        print(f"Total Kontainer Digunakan:
{len(best_solution)}")
        print(f"Durasi Proses Pencarian: {duration:.4f}
detik")
        print(f"Total Move Buruk Diterima (Frekuensi Stuck):
{stuck_counter}")
        print(f"Rasio Penerimaan Move Buruk: {stuck_counter /
MAX_ITER:.4f}")

```

```
print_state(best_solution, "State Akhir (Terbaik)"  
plot_sa(history_data, run)
```

Alur kerja algoritma *simulated annealing* untuk memecahkan *bin packing problem* dimulai di blok eksekusi utama if `_name_ == "__main__"`. Pada fase inisialisasi, data barang dan kapasitas kontainer dimuat dari `utils.py`, dan parameter kritis SA seperti suhu awal (`TEMP_START`), laju pendinginan (`COOLING_RATE`), dan jumlah iterasi (`MAX_ITER`) didefinisikan secara global. Sebagai langkah awal, algoritma menghasilkan *state* awal (`state_initial`) menggunakan metode heuristik *first fit*, yang menyediakan solusi yang relatif baik untuk dimulai.

Proses optimasi utama dikendalikan oleh fungsi `simulated_annealing()`. Begitu fungsi ini dipanggil, ia mencatat waktu mulai dan menetapkan `TEMP_START` sebagai suhu kerja saat ini. Algoritma kemudian memasuki *loop* utama yang berulang sebanyak `MAX_ITER`. Di setiap iterasi, *state* saat ini (`state_current`) diubah sedikit oleh fungsi `generate_neighbor()`. Perubahan ini, yang meliputi *move* satu *item*, *swap* dua *item*, atau operator konsolidasi yang bertujuan mengosongkan kontainer, menghasilkan *neighbor state* (`state_next`). Kualitas dari kedua *state* ini diukur menggunakan fungsi objektif (`objective_function`), yang memberikan skor berdasarkan penalti *overflow*, bobot dominan jumlah kontainer, dan bobot sisa kapasitas kontainer yang lebih kecil.

Perbedaan skor (ΔE) antara `state_next` dan `state_current` menjadi dasar bagi keputusan penerimaan (Prinsip Metropolis). Jika `state_next` memiliki skor yang lebih baik ($\Delta E < 0$), solusi tersebut diterima. Namun, inti dari SA terletak pada kemampuannya untuk menerima langkah buruk ($\Delta E \geq 0$) secara probabilistik. Probabilitas penerimaan

langkah buruk ini dihitung berdasarkan rumus $e^{\frac{\Delta E}{T}}$. Jika angka acak yang dihasilkan lebih kecil dari probabilitas tersebut, langkah yang buruk akan diterima. Mekanisme ini krusial karena memungkinkan algoritma melompat keluar dari minimum lokal menuju area solusi baru. Setelah keputusan penerimaan dibuat, suhu saat ini secara wajib diturunkan dengan dikalikan `COOLING_RATE` (`temp_current *= cooling_rate`), secara bertahap mengurangi energi sistem dan mengalihkan fokus algoritma dari eksplorasi menuju eksloitasi. Di sepanjang proses ini, algoritma terus melacak dan menyimpan *state* terbaik (`state_best`) yang pernah ditemukan. Setelah *loop* iterasi selesai, solusi terbaik (`state_best`), durasi total, dan riwayat skor dikembalikan untuk dicetak dan diplot.

f. Genetic Algorithm

Berikut merupakan salah satu cara untuk mendapatkan state value dari fungsi objektif untuk permasalahan *Bin Packing*.

Nama File	
genetic_algorithm.py	Implementasi dari algoritma Genetic Algorithm
Variabel	
daftar_barang	Dictionary berisi pasangan ID dan ukuran setiap barang
id_barang	List yang berisi ID dari semua barang
kapasitas	Kapasitas maksimum setiap kontainer
ukuran_pop	Jumlah populasi yang digunakan dalam satu generasi
total_gen	Jumlah generasi (iterasi) yang dijalankan dalam proses evolusi
rasio_mutasi	Probabilitas terjadinya mutasi gen pada individu
populasi	List berisi kumpulan kromosom (solusi) pada setiap generasi.
riwayat_fitness	Menyimpan nilai fitness terbaik dan rata-rata tiap generasi untuk keperluan visualisasi
POP	Menentukan jumlah individu (kromosom) dalam setiap populasi. POP = 100 (ber variasi)
GEN	Menentukan jumlah total generasi (iterasi) yang akan dijalankan oleh algoritma. GEN = 1000 (ber variasi)
MUT	Menentukan probabilitas (rasio) mutasi. Ini adalah peluang (0.0-1.0) setiap "gen" (barang) dalam kromosom akan mengalami mutasi. MUT = 0.02 (ber variasi)
model	<i>Instance</i> atau objek dari kelas GenAlgo yang menjalankan algoritma
start, end, durasi	Variabel yang digunakan untuk menghitung durasi eksekusi algoritma
hasil, log_fit, awal	Variabel yang menyimpan nilai <i>return</i> dari

	model.jalankan()
Fungsi dan Prosedur	
init_engine	Inisialisasi objek Genetic Algorithm dan menyimpan parameter seperti data barang, kapasitas, ukuran populasi, dan rasio mutasi
buat_populasi_awal	Membuat populasi awal secara acak, di mana setiap individu merepresentasikan penempatan barang ke kontainer
hitung_fitness(kromosom)	Menghitung skor <i>fitness</i> dari individu berdasarkan jumlah kontainer, penalti kelebihan kapasitas, dan <i>reward</i> kepadatan.
pilih_parent()	Melakukan seleksi menggunakan <i>Tournament Selection</i> untuk memilih dua individu terbaik yang akan dikawinkan.
crossover(ortu1, ortu2)	Menggabungkan dua <i>parent</i> dengan <i>single-point crossover</i> untuk menghasilkan dua anak baru.
mutasi(kromosom)	Mengubah sebagian gen secara acak untuk menjaga keragaman solusi (<i>mutation process</i>).
jalankan()	Fungsi utama untuk menjalankan seluruh proses Genetic Algorithm, mulai dari inisialisasi, evaluasi, seleksi, <i>crossover</i> , mutasi, dan pencarian solusi terbaik.
tampilkan_solusi()	Menampilkan hasil akhir berupa pembagian barang ke kontainer dengan total kapasitas dan jumlah kontainer yang digunakan.
if __name__ == "__main__"	Kode utama yang dieksekusi saat file dijalankan yang bertujuan untuk memuat data, mengatur parameter, membuat <i>instance</i> , mencatat waktu, dan mencetak hasil

Berikut merupakan *source code* implementasi Genetic Algorithm:

```
import random
import matplotlib.pyplot as plt
import utils
import time
```

```

class GenAlgo:
    def init_engine(self, data_barang, list_id,
batas_kapasitas, jumlah_pop, total_gen, rasio_mutasi):
        # setup awal parameter
        self.data_barang = data_barang
        self.list_id = list_id
        self.batas_kapasitas = batas_kapasitas
        self.jumlah_pop = jumlah_pop
        self.total_gen = total_gen
        self.rasio_mutasi = rasio_mutasi
        self.jumlah_barang = len(list_id)
        self.populasi = []
        self.fitness_history = []

    def buat_populasi_awal(self):
        print("Membuat populasi awal...")
        for i in range(self.jumlah_pop):
            kromosom = []
            for j in range(self.jumlah_barang):
                # setiap barang ditempatkan di kontainer acak
                kromosom.append(random.randint(0,
self.jumlah_barang - 1))
            self.populasi.append(kromosom)

    def hitung_fitness(self, kromosom):
        kontainer = {}
        penalti_over = 0
        reward_padat = 0

        # masukkan barang ke dalam kontainer sesuai kromosom
        for i in range(self.jumlah_barang):
            id_brg = self.list_id[i]
            ukuran = self.data_barang[id_brg]
            no_kontainer = kromosom[i]

            if no_kontainer not in kontainer:
                kontainer[no_kontainer] = 0
            kontainer[no_kontainer] += ukuran

        # hitung penalti overflow dan reward kepadatan
        for key in kontainer:
            isi = kontainer[key]
            if isi > self.batas_kapasitas:
                penalti_over += isi - self.batas_kapasitas
            else:
                kepadatan = isi / self.batas_kapasitas
                reward_padat += kepadatan ** 2

        # hitung total fitness
        penalti = 10000

```

```

        faktor_kontainer = 100
        total_kontainer = len(kontainer)
        fitness = (penalty_over * penalty) + (total_kontainer
* faktor_kontainer) - reward_padat
        return fitness

    def pilih_parent(self):
        ukuran_turnamen = 5
        kandidat1 = random.sample(self.populasi,
ukuran_turnamen)
        kandidat2 = random.sample(self.populasi,
ukuran_turnamen)

        parent1 = min(kandidat1, key=lambda ind:
self.hitung_fitness(ind))
        parent2 = min(kandidat2, key=lambda ind:
self.hitung_fitness(ind))
        return parent1, parent2

    def crossover(self, parent1, parent2):
        titik = random.randint(1, self.jumlah_barang - 1)
        anak1 = parent1[:titik] + parent2[titik:]
        anak2 = parent2[:titik] + parent1[titik:]
        return anak1, anak2

    def mutasi(self, kromosom):
        hasil = list(kromosom)
        kontainer_unik = list(set(hasil))
        if len(kontainer_unik) == 0:
            return hasil

        # mutasi acak sebagian gen
        for i in range(self.jumlah_barang):
            if random.random() < self.rasio_mutasi:
                if random.random() < 0.5:
                    hasil[i] = random.choice(kontainer_unik)
                else:
                    hasil[i] = random.randint(0,
self.jumlah_barang - 1)
        return hasil

    def jalankan(self):
        self.buat_populasi_awal()
        individu_awal = None

        for gen in range(self.total_gen):
            semua_fitness = []
            total_fit = 0
            terbaik = float("inf")
            individu_terbaik = None

```

```

        for kromosom in self.populasi:
            nilai_fit = self.hitung_fitness(kromosom)
            semua_fitness.append((nilai_fit, kromosom))
            total_fit += nilai_fit
            if nilai_fit < terbaik:
                terbaik = nilai_fit
                individu_terbaik = kromosom

        if gen == 0:
            individu_awal = individu_terbaik

        rata_fit = total_fit / self.jumlah_pop
        self.fitness_history.append((terbaik, rata_fit))

        if gen % 10 == 0:
            print(f"Generasi {gen}: Fitness terbaik = {terbaik}, Rata-rata = {rata_fit}")

        populasi_baru = []
        while len(populasi_baru) < self.jumlah_pop:
            parent1, parent2 = self.pilih_parent()
            c1, c2 = self.crossover(parent1, parent2)
            c1 = self.mutasi(c1)
            c2 = self.mutasi(c2)
            populasi_baru.extend([c1, c2])

        self.populasi = populasi_baru[:self.jumlah_pop]

        hasil_akhir = min([(self.hitung_fitness(k), k) for k in self.populasi], key=lambda x: x[0])
        print(f"Fitness terbaik: {hasil_akhir[0]}")
        return hasil_akhir[1], self.fitness_history,
individu_awal

def tampilan_solusi(individu, data_barang, list_id,
kapasitas):
    wadah = {}
    for i in range(len(list_id)):
        idb = list_id[i]
        ukuran = data_barang[idb]
        bin_no = individu[i]
        if bin_no not in wadah:
            wadah[bin_no] = []
        wadah[bin_no].append((idb, ukuran))

    print(f"Total Kontainer: {len(wadah)}")
    for idx, key in enumerate(sorted(wadah.keys()), start=1):
        total = sum(x[1] for x in wadah[key])
        print(f"{idx}. Kontainer (Total: {total}/{kapasitas})")

```

```

        for brg in wadah[key]:
            print(f"    - {brg[0]} ({brg[1]})")

if __name__ == "__main__":
    file_data = "data_bin_packing.json"
    kapasitas, data_barang, _ = utils.load_data(file_data)
    list_id = list(data_barang.keys())

    POP = 100 # Populasi
    GEN = 50 # Generasi
    MUT = 0.02 # Probabilitas mutasi

    model = GenAlgo()
    model.init_engine(data_barang, list_id, kapasitas, POP,
    GEN, MUT)

    start = time.time()
    hasil, log_fit, awal = model.jalankan()
    end = time.time()

    print("\nGenerasi Awal")
    tampilan_solusi(awal, data_barang, list_id, kapasitas)

    print("\nGenerasi Akhir")
    tampilan_solusi(hasil, data_barang, list_id, kapasitas)

    durasi = end - start
    print(f"\nDurasi: {durasi:.2f} detik")

    # Plot
    log_fit_terbaik = [x[0] for x in log_fit]
    log_fit_rata = [x[1] for x in log_fit]
    fig, axes = plt.subplots(2, 1, figsize=(10, 8))

    axes[0].plot(log_fit_terbaik, label="Best Objective
Function Value", color='blue')
    axes[0].set_title("Best Objective Function Value vs
Iterations")
    axes[0].set_ylabel("Best Objective Value")
    axes[0].set_xlabel("Generation (Iteration)")

    skor_valid = [s for s in log_fit_terbaik if s < 10000]

    if skor_valid:
        min_skor = min(skor_valid)
        max_skor = max(skor_valid)
        axes[0].set_ylim(min_skor - 50, max_skor + 50)

    axes[0].legend()
    axes[0].grid(True)

```

```

        axes[1].plot(log_fit_rata, label="Average Objective
Function Value", color='orange')
        axes[1].set_title("Average Objective Function Value vs
Iterations")
        axes[1].set_xlabel("Generation (Iteration)")
        axes[1].set_ylabel("Average Objective Function Value")
        axes[1].legend()
        axes[1].grid(True)

plt.tight_layout()
plt.show()

```

Alur kerja *genetic algorithm* dimulai dari kode if `__name__ == "__main__"`. Pertama-tama, data barang akan dimuat dan kapasitas kontainer dari `utils.py`. Lalu, tiga parameter utama untuk eksperimen didefinisikan: POP (jumlah populasi), GEN (jumlah generasi atau iterasi), dan MUT (probabilitas mutasi). Nilai-nilai ini dapat diubah-ubah untuk menguji pengaruhnya terhadap hasil akhir. Setelah itu, sebuah objek dari kelas `GenAlgo` dibuat menggunakan `model = GenAlgo()`, dan semua parameter ini (termasuk data barang) disimpan ke dalam objek tersebut melalui `model.init_engine()`.

Setelah inisialisasi, `stopwatch` (`time.time()`) dinyalakan dan fungsi utama `model.jalankan()` dipanggil. Proses evolusi dimulai. Langkah pertama di dalam `jalankan()` adalah memanggil `buat_populasi_awal()`, yang akan menciptakan populasi awal (misalnya 100 individu) menggunakan inisialisasi acak (Random Initialization). Berbeda dengan heuristik pintar seperti *first fit*, metode ini sengaja menciptakan solusi awal yang bodoh dan sangat beragam. Setiap individu (kromosom) direpresentasikan sebagai *list of integers* (contoh: 0, 5, 2), di mana setiap barang (gen) diberi nomor kontainer secara acak. Pendekatan acak ini menghasilkan populasi awal dengan skor *fitness* yang sangat tinggi (buruk) karena banyaknya *overflow*, namun krusial untuk memaksa GA belajar dari nol dan berevolusi secara alami tanpa terjebak di *local optimum* yang mungkin ditemukan oleh *first fit*.

Selanjutnya, algoritma masuk ke *loop* utama yang berjalan sebanyak GEN (jumlah generasi). Di setiap generasi, langkah pertama adalah evaluasi. Setiap individu dalam populasi saat ini akan diukur kualitasnya menggunakan fungsi `hitung_fitness()` yang menerjemahkan kromosom menjadi skor numerik. Skor dihitung berdasarkan tiga komponen, yaitu penalti *overflow*, di mana setiap unit kapasitas yang berlebih dikalikan penalti besar (10000) lalu bobot jumlah kontainer, di mana total kontainer dikalikan bobot besar (100) agar faktor ini dominan, dan *reward* kepadatan, di mana kontainer yang padat (*valid*) akan mengurangi skor (menjadi lebih baik). Skor *fitness* terbaik dan rata-rata dari generasi ini kemudian disimpan ke `fitness_history` untuk dibuat plot.

Setelah evaluasi, langkah selanjutnya adalah reproduksi untuk menciptakan populasi_baru. Kode ini tidak menggunakan elitisme (tidak ada individu lama yang diselamatkan), sehingga 100% populasi baru diisi oleh anak. Proses ini berulang dimana pertama-tama, fungsi pilih_parent() dipanggil untuk memilih dua orang tua menggunakan *tournament selection* (mengadu 5 individu acak). Kedua *parent* ini kemudian dikawinkan menggunakan crossover() (Single-Point Crossover) untuk menghasilkan dua anak. Terakhir, kedua anak ini diberi gangguan acak melalui mutasi(). Fungsi mutasi() ini menggunakan *hybrid mutation* dimana ada 50% peluang gen akan dipindah ke kontainer yang sudah ada (untuk merapikan) dan 50% peluang dipindah ke kontainer acak baru (untuk menjelajah). Anak-anak baru ini dimasukkan ke populasi_baru hingga penuh, yang kemudian menggantikan populasi lama.

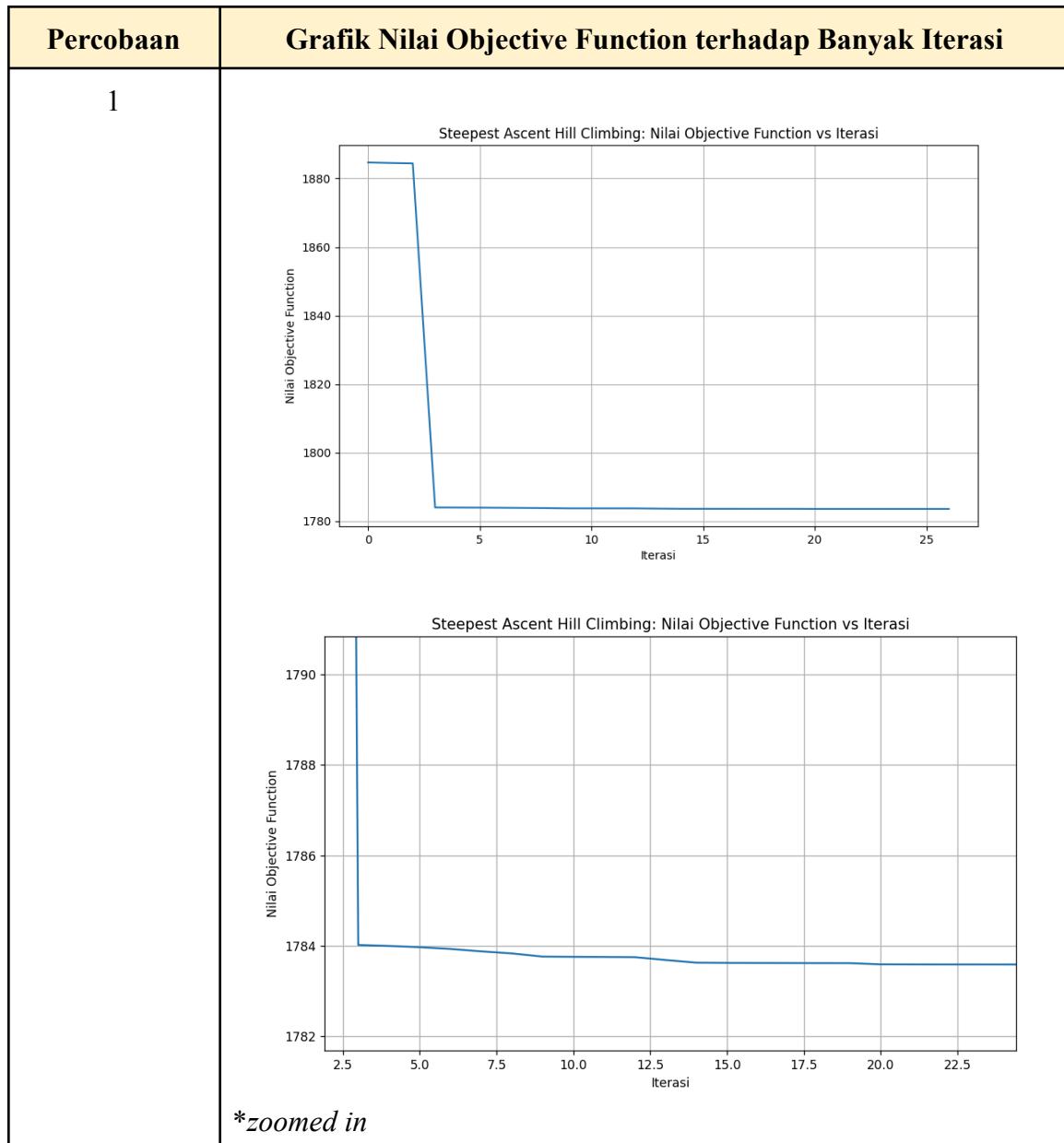
Setelah *loop* generasi selesai, jalankan() akan mencari individu terbaik dari generasi terakhir dan mengembalikannya. Setelah itu, *stopwatch* dihentikan, dan durasi total dihitung.

BAB 3

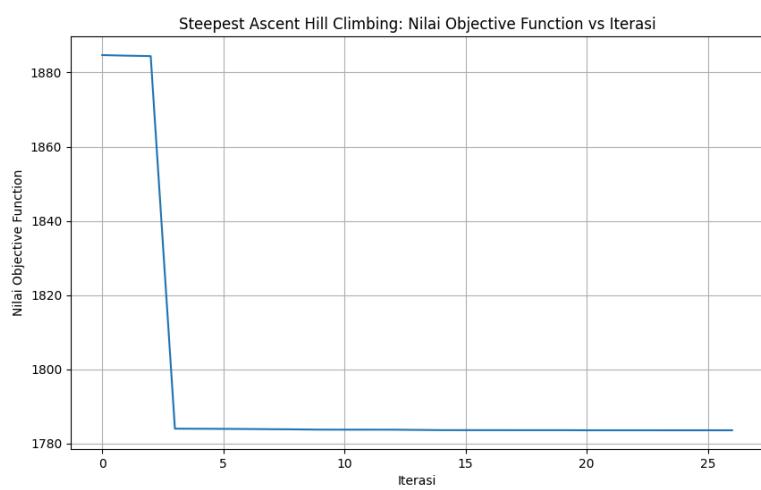
HASIL DAN ANALISIS

a. Steepest Ascent Hill-Climbing

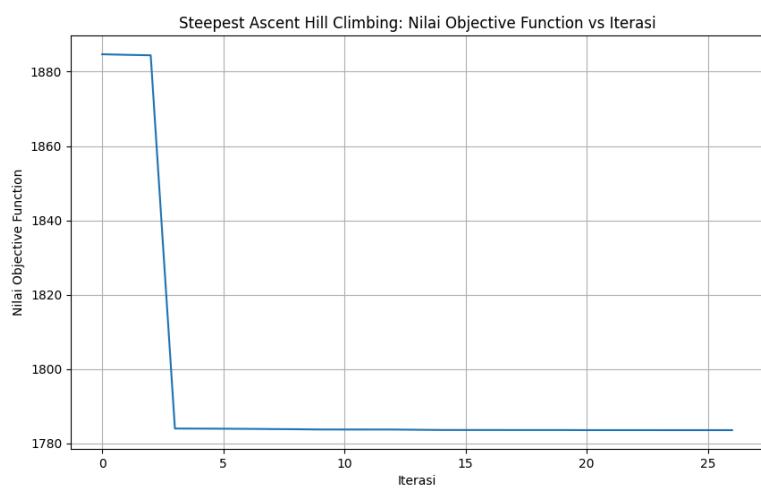
Diterapkan tiga percobaan yang masing-masing memiliki konfigurasi jumlah iterasi dan jumlah populasi maksimum yang berbeda untuk algoritma Steepest Ascent Hill-Climbing.



2



3



Initial State	Final State
<pre> State Awal: ----- Kontainer 1 (Total: 100/100) [BRG004, BRG008] Kontainer 2 (Total: 97/100) [BRG003, BRG012] Kontainer 3 (Total: 97/100) [BRG026, BRG028] Kontainer 4 (Total: 96/100) [BRG001, BRG002, BRG007] Kontainer 5 (Total: 96/100) [BRG005, BRG006] Kontainer 6 (Total: 95/100) [BRG013, BRG014] Kontainer 7 (Total: 95/100) [BRG019, BRG020, BRG029] Kontainer 8 (Total: 95/100) [BRG027, BRG033] Kontainer 9 (Total: 94/100) [BRG016, BRG017, BRG018, BRG024] Kontainer 10 (Total: 94/100) [BRG031, BRG032] Kontainer 11 (Total: 93/100) [BRG021, BRG022, BRG035] Kontainer 12 (Total: 92/100) [BRG009, BRG011] Kontainer 13 (Total: 92/100) [BRG023, BRG025] Kontainer 14 (Total: 90/100) [BRG010, BRG015] Kontainer 15 (Total: 85/100) [BRG034, BRG036] Kontainer 16 (Total: 78/100) [BRG030] Kontainer 17 (Total: 75/100) [BRG037, BRG038] Kontainer 18 (Total: 66/100) [BRG040] Kontainer 19 (Total: 64/100) [BRG039] Total Kontainer Digenakan: 19 Nilai Objective Function: 1884.6940 </pre>	<pre> State akhir: ----- Kontainer 1 (Total: 100/100) [BRG017, BRG025, BRG031] Kontainer 2 (Total: 100/100) [BRG003, BRG029] Kontainer 3 (Total: 100/100) [BRG004, BRG008] Kontainer 4 (Total: 100/100) [BRG010, BRG033] Kontainer 5 (Total: 100/100) [BRG039, BRG014] Kontainer 6 (Total: 100/100) [BRG021, BRG015, BRG011] Kontainer 7 (Total: 100/100) [BRG016, BRG035, BRG007] Kontainer 8 (Total: 100/100) [BRG023, BRG019] Kontainer 9 (Total: 100/100) [BRG026, BRG002] Kontainer 10 (Total: 100/100) [BRG027, BRG038] Kontainer 11 (Total: 100/100) [BRG040, BRG028] Kontainer 12 (Total: 99/100) [BRG005, BRG001, BRG018, BRG024] Kontainer 13 (Total: 96/100) [BRG022, BRG006] Kontainer 14 (Total: 96/100) [BRG037, BRG034] Kontainer 15 (Total: 95/100) [BRG030, BRG012] Kontainer 16 (Total: 90/100) [BRG036, BRG013] Kontainer 17 (Total: 89/100) [BRG009, BRG032] Kontainer 18 (Total: 29/100) [BRG020] Total Kontainer Digenakan: 18 </pre>

Parameter	Percobaan 1	Percobaan 2	Percobaan 3
Durasi (s)	0.8508	0.9631	0.8501
Jumlah Iterasi	26	26	26
Initial State	1884.6940	1884.6940	1884.6940
Final State	1783.5880	1783.5880	1783.5880

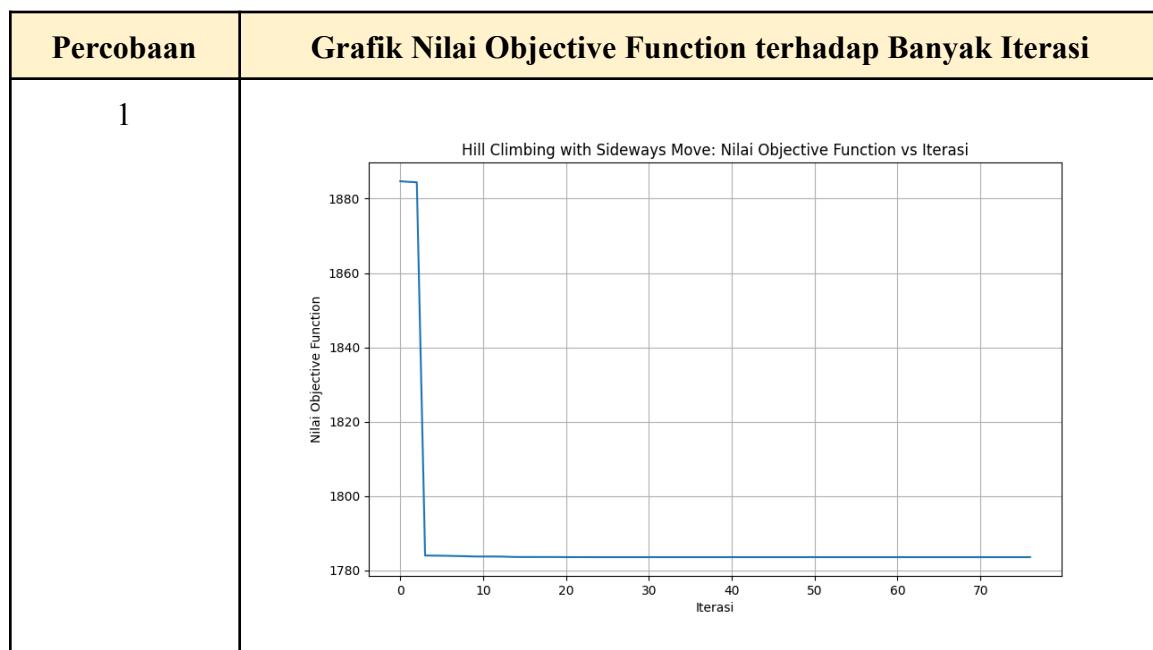
Algoritma *steepest ascent hill-climbing* menunjukkan konsistensi yang sangat tinggi dalam menemukan local optimum. Dari tiga kali percobaan yang semuanya dimulai dari kondisi awal dengan nilai 1884.6940, hasil akhirnya selalu identik dengan nilai fungsi objektif 1783.5880. Solusi ini merepresentasikan konfigurasi 18 kontainer dari total 19 pada kondisi awal. Konsistensi ini muncul karena sifat deterministik SAHC yang selalu menempuh jalur pencarian yang sama selama titik awalnya tidak berubah.

Namun, hasil yang sangat stabil ini sekaligus memperlihatkan kelemahan utama SAHC, yaitu kecenderungan untuk berhenti pada *local optimum*. Algoritma berhenti begitu tidak menemukan tetangga yang memberikan perbaikan nilai fungsi objektif, padahal masih ada solusi global yang lebih baik dengan 17 kontainer. Hal ini menunjukkan bahwa pendekatan *greedy* SAHC membuatnya hanya mampu mencari di sekitar solusi saat ini tanpa menjelajahi area lain dalam ruang pencarian.

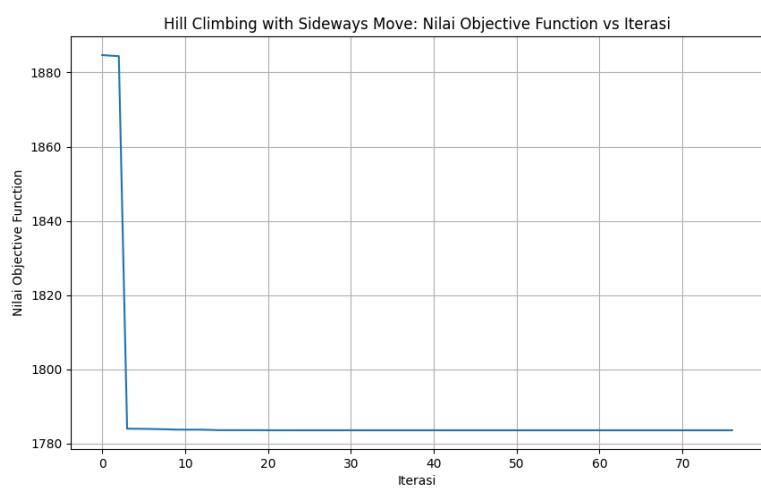
Dari segi efisiensi, SAHC menjadi yang paling cepat dengan rata-rata waktu kurang dari satu detik dan hanya memerlukan sekitar 26 iterasi untuk mencapai titik berhenti. Artinya, SAHC mampu memberikan hasil yang cepat tetapi kurang eksploratif. Jika dibandingkan dengan algoritma *local search* lain, SAHC adalah yang paling efisien secara komputasi namun paling mudah terjebak di *local optimum*, sehingga hasil akhirnya selalu sama dan tidak pernah mendekati *global optimum*.

b. Hill-Climbing with Sideways Move

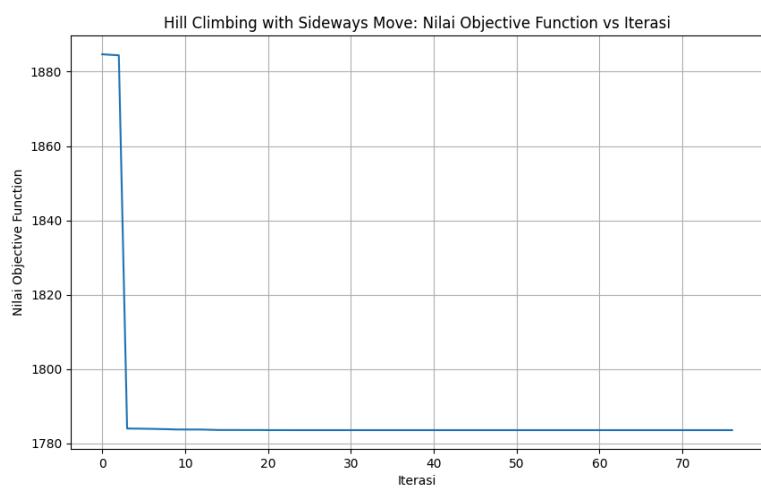
Diterapkan tiga percobaan yang masing-masing memiliki konfigurasi jumlah iterasi dan jumlah populasi maksimum yang berbeda untuk algoritma Hill-Climbing with Sideways Move.



2



3



Initial State	Final State
<pre> State Awal: ----- Kontainer 1 (Total: 100/100) [BRG004, BRG008] Kontainer 2 (Total: 97/100) [BRG003, BRG012] Kontainer 3 (Total: 97/100) [BRG026, BRG028] Kontainer 4 (Total: 96/100) [BRG001, BRG002, BRG007] Kontainer 5 (Total: 96/100) [BRG005, BRG006] Kontainer 6 (Total: 95/100) [BRG013, BRG014] Kontainer 7 (Total: 95/100) [BRG019, BRG020, BRG029] Kontainer 8 (Total: 95/100) [BRG027, BRG033] Kontainer 9 (Total: 94/100) [BRG016, BRG017, BRG018, BRG024] Kontainer 10 (Total: 94/100) [BRG031, BRG032] Kontainer 11 (Total: 93/100) [BRG021, BRG022, BRG035] Kontainer 12 (Total: 92/100) [BRG009, BRG011] Kontainer 13 (Total: 92/100) [BRG023, BRG025] Kontainer 14 (Total: 90/100) [BRG010, BRG015] Kontainer 15 (Total: 85/100) [BRG034, BRG036] Kontainer 16 (Total: 78/100) [BRG030] Kontainer 17 (Total: 75/100) [BRG037, BRG038] Kontainer 18 (Total: 66/100) [BRG040] Kontainer 19 (Total: 64/100) [BRG039] ----- Total Kontainer Digunakan: 19 Nilai Objective Function: 1884.6940 </pre>	<pre> State Akhir: ----- Kontainer 1 (Total: 100/100) [BRG017, BRG025, BRG031] Kontainer 2 (Total: 100/100) [BRG003, BRG029] Kontainer 3 (Total: 100/100) [BRG004, BRG008] Kontainer 4 (Total: 100/100) [BRG010, BRG033] Kontainer 5 (Total: 100/100) [BRG039, BRG014] Kontainer 6 (Total: 100/100) [BRG021, BRG015, BRG011] Kontainer 7 (Total: 100/100) [BRG016, BRG035, BRG007] Kontainer 8 (Total: 100/100) [BRG023, BRG019] Kontainer 9 (Total: 100/100) [BRG026, BRG002] Kontainer 10 (Total: 100/100) [BRG027, BRG038] Kontainer 11 (Total: 100/100) [BRG040, BRG028] Kontainer 12 (Total: 99/100) [BRG005, BRG001, BRG018, BRG024] Kontainer 13 (Total: 96/100) [BRG022, BRG006] Kontainer 14 (Total: 96/100) [BRG037, BRG034] Kontainer 15 (Total: 95/100) [BRG030, BRG012] Kontainer 16 (Total: 90/100) [BRG036, BRG013] Kontainer 17 (Total: 89/100) [BRG009, BRG032] Kontainer 18 (Total: 29/100) [BRG020] ----- Total Kontainer Digunakan: 18 </pre>

Parameter	Percobaan 1	Percobaan 2	Percobaan 3
Durasi (s)	2.5920	2.6613	2.4552
Jumlah Iterasi	76	76	76
Maximum sideways move	50	50	50
Initial State	1884.6940	1884.6940	1884.6940
Final State	1783.5880	1783.5880	1783.5880

Varian *hill-climbing* dengan mekanisme *sideways move* menambahkan kemampuan untuk berpindah ke solusi tetangga dengan nilai yang sama hingga batas 50 langkah. Tujuannya adalah untuk membantu algoritma keluar dari *plateau*, yaitu area

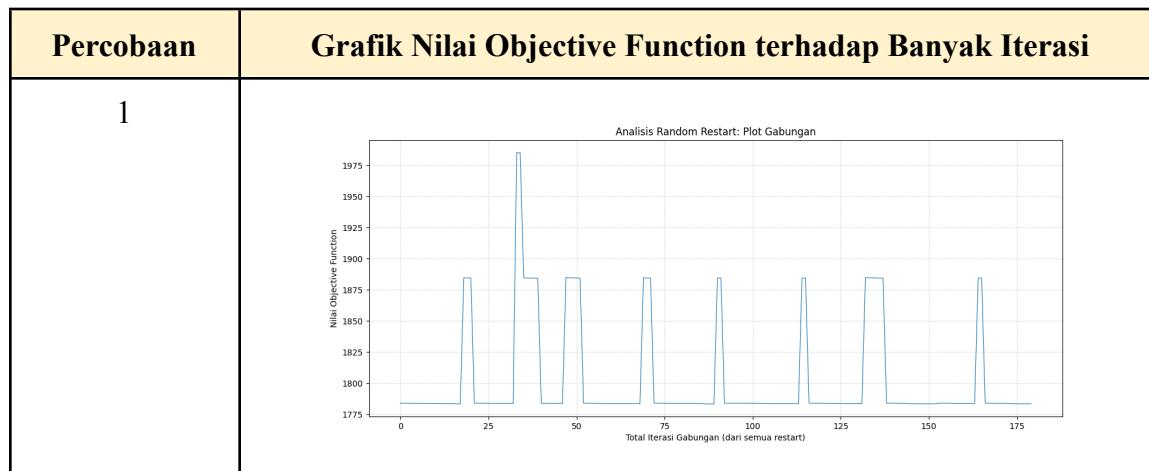
datar di mana banyak solusi memiliki nilai serupa. Akan tetapi, pada eksperimen ini, seluruh percobaan tetap berakhir pada hasil yang identik dengan SAHC murni, yaitu nilai akhir 1783.5880 dengan solusi 18 kontainer.

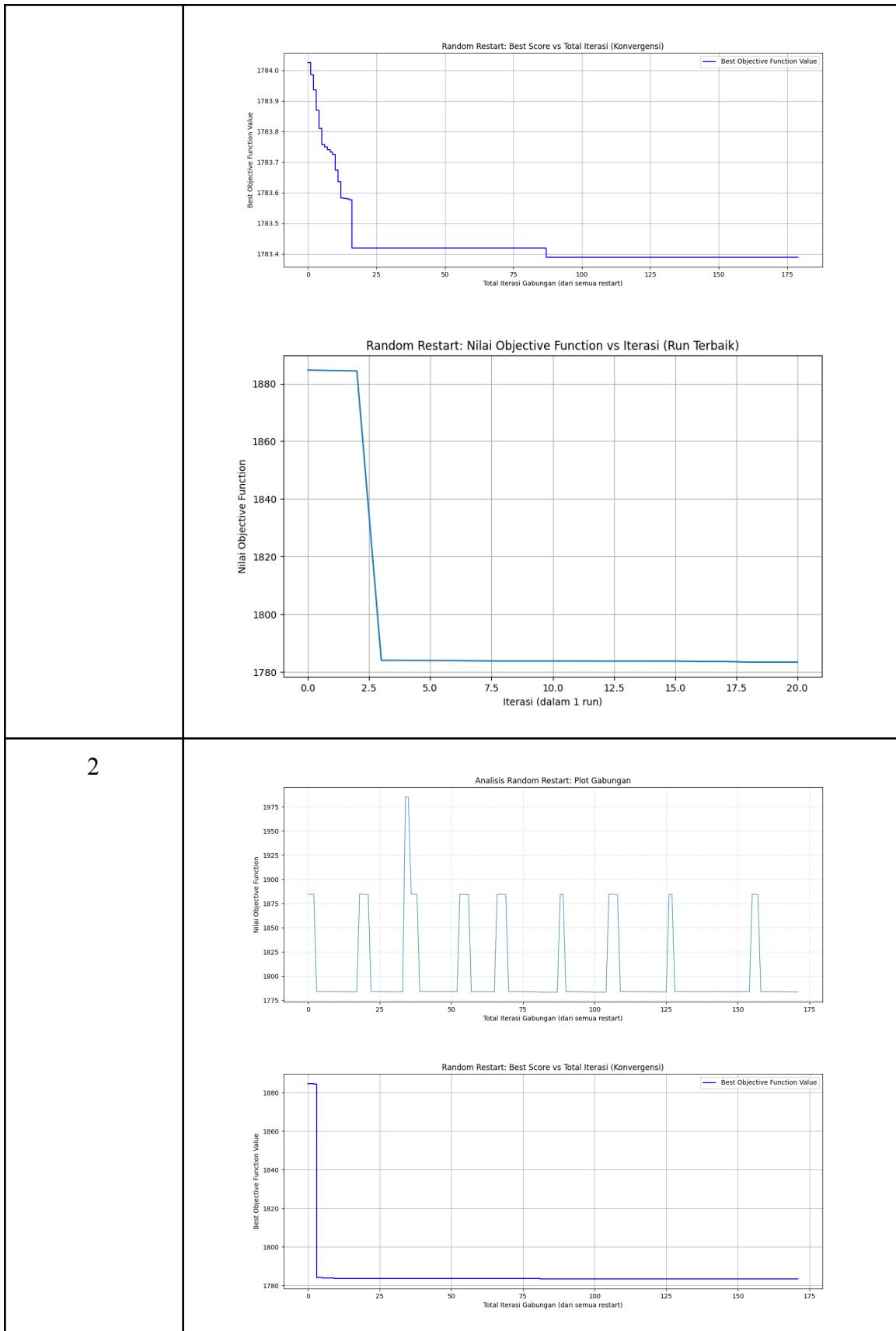
Konsistensi hasil ini menunjukkan bahwa solusi yang ditemukan bukan merupakan *plateau*, melainkan sebuah puncak curam yang tidak memiliki jalur lateral menuju solusi yang lebih baik. Karena itu, mekanisme *sideways move* tidak berpengaruh terhadap hasil akhir. Sebaliknya, mekanisme ini justru menambah waktu eksekusi secara signifikan. SAHC berhenti sekitar 0.8 hingga 0.9 detik, sedangkan varian ini membutuhkan rata-rata 2.5 detik akibat 50 percobaan *sideways* yang tidak membawa hasil.

Jika dibandingkan dengan algoritma lain, varian *sideways move* tidak memberikan peningkatan kualitas solusi tetapi memperlambat performa. Hal ini memperlihatkan bahwa efektivitas *sideways move* sangat tergantung pada bentuk *landscape* fungsi objektif. Ketika solusi berada di puncak yang curam, *sideways move* menjadi tidak berguna dan hanya menambah beban komputasi.

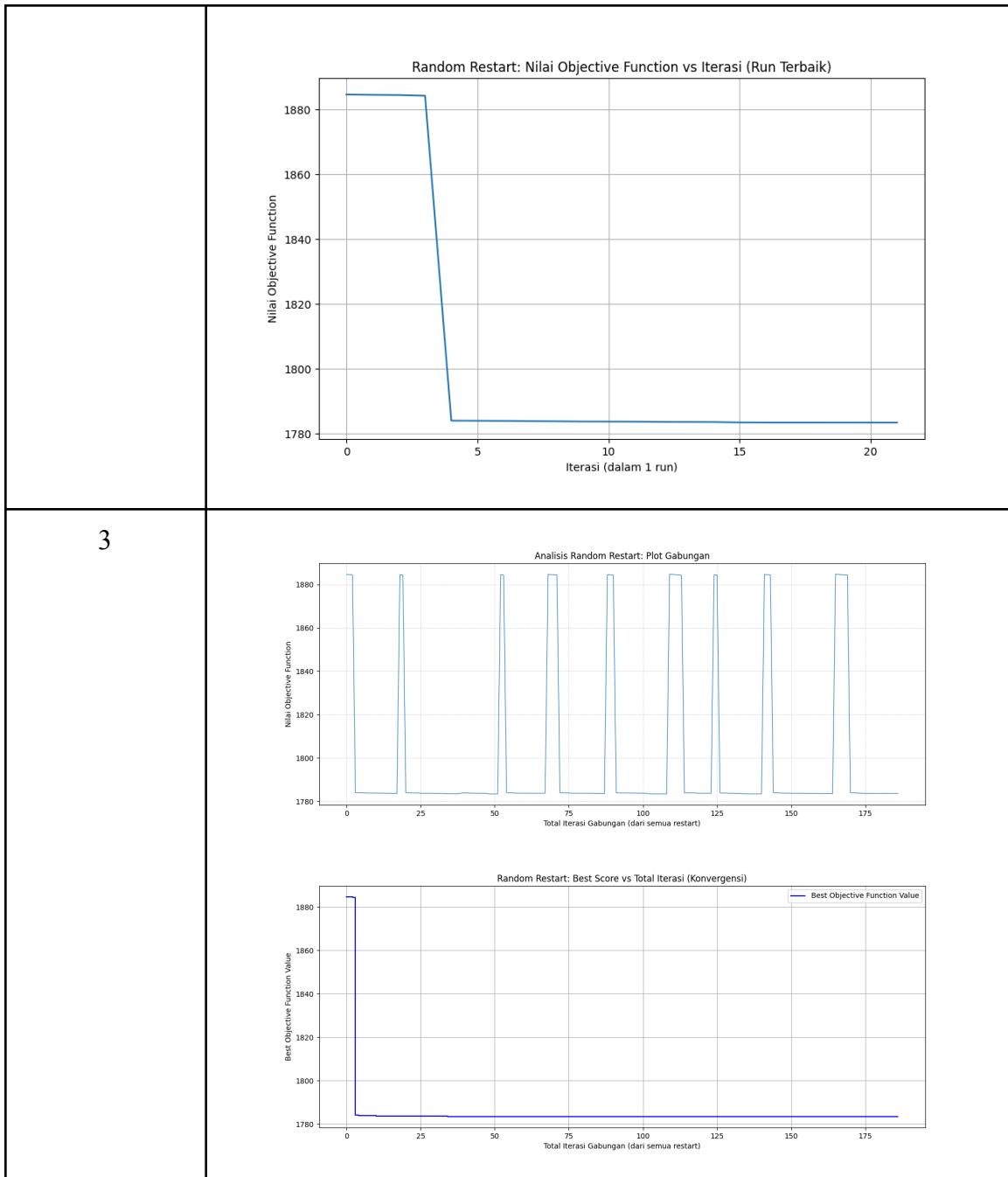
c. Random Restart Hill-Climbing

Diterapkan tiga percobaan yang masing-masing memiliki konfigurasi jumlah iterasi dan jumlah populasi maksimum yang berbeda untuk algoritma Random Restart Hill-Climbing.

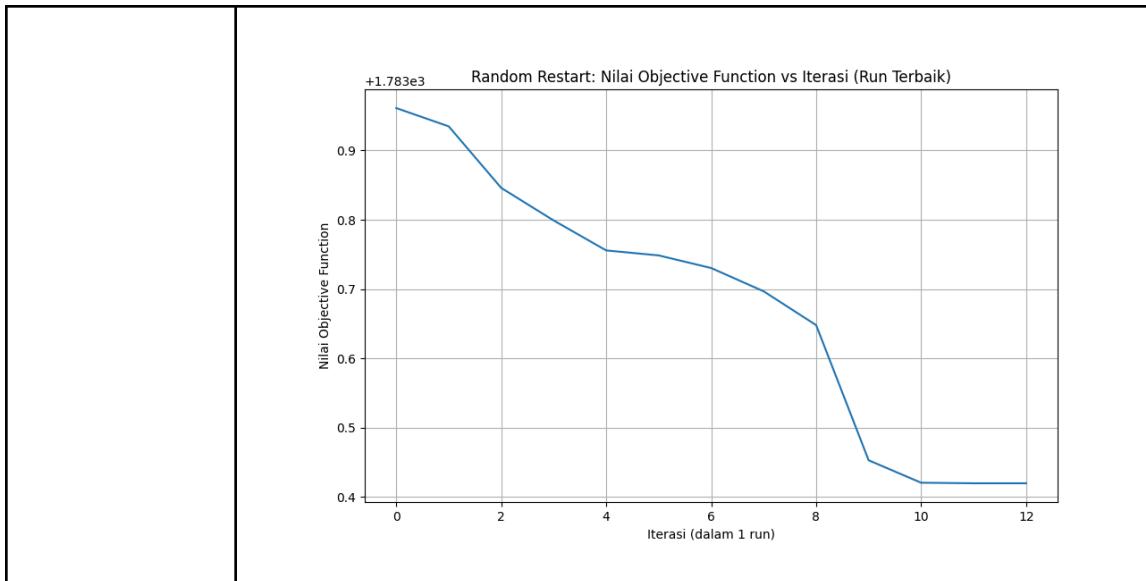




2



3



Percobaan	Initial State	Final State
1	<pre> State Awal: ----- Kontainer 1 (Total: 100/100) [BRG034, BRG019] Kontainer 2 (Total: 99/100) [BRG027, BRG012, BRG017] Kontainer 3 (Total: 98/100) [BRG009, BRG007, BRG029] Kontainer 4 (Total: 98/100) [BRG003, BRG015] Kontainer 5 (Total: 97/100) [BRG004, BRG001] Kontainer 6 (Total: 97/100) [BRG030, BRG024] Kontainer 7 (Total: 97/100) [BRG039, BRG038] Kontainer 8 (Total: 96/100) [BRG023, BRG037] Kontainer 9 (Total: 95/100) [BRG028, BRG005, BRG035] Kontainer 10 (Total: 95/100) [BRG016, BRG011] Kontainer 11 (Total: 95/100) [BRG013, BRG014] Kontainer 12 (Total: 95/100) [BRG040, BRG020] Kontainer 13 (Total: 91/100) [BRG002, BRG006] Kontainer 14 (Total: 91/100) [BRG021, BRG033, BRG036] Kontainer 15 (Total: 89/100) [BRG022, BRG032] Kontainer 16 (Total: 89/100) [BRG026, BRG008] Kontainer 17 (Total: 87/100) [BRG010, BRG018] Kontainer 18 (Total: 85/100) [BRG031, BRG025] ----- Total Kontainer Digunakan: 18 Nilai Objective Function: 1784.0250 </pre>	<pre> State Akhir: ----- Kontainer 1 (Total: 100/100) [BRG002, BRG016, BRG015] Kontainer 2 (Total: 100/100) [BRG039, BRG007] Kontainer 3 (Total: 100/100) [BRG004, BRG008] Kontainer 4 (Total: 100/100) [BRG003, BRG029] Kontainer 5 (Total: 100/100) [BRG011, BRG035, BRG036] Kontainer 6 (Total: 100/100) [BRG010, BRG033] Kontainer 7 (Total: 100/100) [BRG027, BRG038] Kontainer 8 (Total: 100/100) [BRG023, BRG020, BRG012] Kontainer 9 (Total: 100/100) [BRG034, BRG019] Kontainer 10 (Total: 100/100) [BRG040, BRG028] Kontainer 11 (Total: 99/100) [BRG037, BRG009, BRG017] Kontainer 12 (Total: 99/100) [BRG026, BRG014] Kontainer 13 (Total: 97/100) [BRG038, BRG024] Kontainer 14 (Total: 97/100) [BRG001, BRG005, BRG021] Kontainer 15 (Total: 97/100) [BRG013, BRG025] Kontainer 16 (Total: 96/100) [BRG006, BRG022] Kontainer 17 (Total: 94/100) [BRG031, BRG032] Kontainer 18 (Total: 15/100) [BRG018] Total Kontainer Digunakan: 18 Skor Akhir Terbaik: 1783.3894 </pre>

2	<p>State Awal:</p> <pre> Kontainer 1 (Total: 100/100) [BRG040, BRG017, BRG024] Kontainer 2 (Total: 100/100) [BRG019, BRG014, BRG015] Kontainer 3 (Total: 99/100) [BRG033, BRG005, BRG020] Kontainer 4 (Total: 98/100) [BRG027, BRG036] Kontainer 5 (Total: 98/100) [BRG013, BRG035, BRG029] Kontainer 6 (Total: 98/100) [BRG010, BRG088] Kontainer 7 (Total: 96/100) [BRG026, BRG038] Kontainer 8 (Total: 95/100) [BRG003, BRG018] Kontainer 9 (Total: 95/100) [BRG030, BRG012] Kontainer 10 (Total: 90/100) [BRG006, BRG007] Kontainer 11 (Total: 89/100) [BRG009, BRG031] Kontainer 12 (Total: 88/100) [BRG011, BRG025] Kontainer 13 (Total: 88/100) [BRG023, BRG028] Kontainer 14 (Total: 87/100) [BRG022, BRG016] Kontainer 15 (Total: 87/100) [BRG039, BRG001] Kontainer 16 (Total: 86/100) [BRG034, BRG021] Kontainer 17 (Total: 84/100) [BRG032, BRG002] Kontainer 18 (Total: 74/100) [BRG004] Kontainer 19 (Total: 42/100) [BRG037] </pre> <p>Total Kontainer Digunakan: 19 Nilai Objective Function: 1884.5782</p>	<p>State Akhir:</p> <pre> Kontainer 1 (Total: 100/100) [BRG015, BRG013, BRG001] Kontainer 2 (Total: 100/100) [BRG014, BRG039] Kontainer 3 (Total: 100/100) [BRG019, BRG006] Kontainer 4 (Total: 100/100) [BRG038, BRG036, BRG007] Kontainer 5 (Total: 100/100) [BRG012, BRG020, BRG023] Kontainer 6 (Total: 100/100) [BRG002, BRG026] Kontainer 7 (Total: 100/100) [BRG028, BRG040] Kontainer 8 (Total: 100/100) [BRG033, BRG010] Kontainer 9 (Total: 100/100) [BRG025, BRG005, BRG029] Kontainer 10 (Total: 100/100) [BRG008, BRG004] Kontainer 11 (Total: 99/100) [BRG003, BRG035] Kontainer 12 (Total: 99/100) [BRG021, BRG027] Kontainer 13 (Total: 99/100) [BRG016, BRG034] Kontainer 14 (Total: 99/100) [BRG022, BRG009, BRG017] Kontainer 15 (Total: 97/100) [BRG030, BRG024] Kontainer 16 (Total: 97/100) [BRG031, BRG011] Kontainer 17 (Total: 89/100) [BRG037, BRG032] Kontainer 18 (Total: 15/100) [BRG018] </pre> <p>Total Kontainer Digunakan: 18 Skor Akhir Terbaik: 1783.3832</p>
3	<p>State Awal:</p> <pre> Kontainer 1 (Total: 100/100) [BRG026, BRG029, BRG012] Kontainer 2 (Total: 99/100) [BRG038, BRG040] Kontainer 3 (Total: 99/100) [BRG003, BRG024] Kontainer 4 (Total: 98/100) [BRG025, BRG001, BRG002] Kontainer 5 (Total: 96/100) [BRG030, BRG015] Kontainer 6 (Total: 96/100) [BRG006, BRG037] Kontainer 7 (Total: 95/100) [BRG013, BRG014] Kontainer 8 (Total: 94/100) [BRG028, BRG036, BRG020] Kontainer 9 (Total: 93/100) [BRG027, BRG008] Kontainer 10 (Total: 91/100) [BRG010, BRG035] Kontainer 11 (Total: 90/100) [BRG034, BRG007] Kontainer 12 (Total: 89/100) [BRG009, BRG032] Kontainer 13 (Total: 89/100) [BRG004, BRG017] Kontainer 14 (Total: 89/100) [BRG031, BRG022] Kontainer 15 (Total: 88/100) [BRG016, BRG033, BRG018] Kontainer 16 (Total: 88/100) [BRG005, BRG019] Kontainer 17 (Total: 86/100) [BRG023, BRG021] Kontainer 18 (Total: 64/100) [BRG039] Kontainer 19 (Total: 50/100) [BRG011] </pre> <p>Total Kontainer Digunakan: 19 Nilai Objective Function: 1884.6228</p>	<p>State Akhir:</p> <pre> Kontainer 1 (Total: 100/100) [BRG028, BRG024, BRG032] Kontainer 2 (Total: 100/100) [BRG004, BRG008] Kontainer 3 (Total: 100/100) [BRG005, BRG025, BRG029] Kontainer 4 (Total: 100/100) [BRG019, BRG023] Kontainer 5 (Total: 100/100) [BRG034, BRG036, BRG018] Kontainer 6 (Total: 100/100) [BRG039, BRG014] Kontainer 7 (Total: 100/100) [BRG010, BRG033] Kontainer 8 (Total: 100/100) [BRG026, BRG002] Kontainer 9 (Total: 100/100) [BRG027, BRG038] Kontainer 10 (Total: 99/100) [BRG001, BRG031, BRG020] Kontainer 11 (Total: 99/100) [BRG003, BRG035] Kontainer 12 (Total: 99/100) [BRG009, BRG022, BRG017] Kontainer 13 (Total: 99/100) [BRG006, BRG016] Kontainer 14 (Total: 98/100) [BRG040, BRG021] Kontainer 15 (Total: 96/100) [BRG030, BRG015] Kontainer 16 (Total: 95/100) [BRG013, BRG007] Kontainer 17 (Total: 92/100) [BRG037, BRG011] Kontainer 18 (Total: 17/100) [BRG012] </pre> <p>Total Kontainer Digunakan: 18 Skor Akhir Terbaik: 1783.4198</p>

Parameter	Percobaan 1	Percobaan 2	Percobaan 3
Durasi (s)	5.2708	5.1122	5.5871
Jumlah Restart	10	10	10
Maximum Restart	10	10	10
Initial State	1784.0250	1884.5782	1884.6228
Final State	1783.3894	1783.3832	1783.4198

Percobaan	Banyak Iterasi Per Restart
1	Jumlah iterasi pada restart ke-1: 17 Jumlah iterasi pada restart ke-2: 14 Jumlah iterasi pada restart ke-3: 13 Jumlah iterasi pada restart ke-4: 21 Jumlah iterasi pada restart ke-5: 20 Jumlah iterasi pada restart ke-6: 23 Jumlah iterasi pada restart ke-7: 17 Jumlah iterasi pada restart ke-8: 20 Jumlah iterasi pada restart ke-9: 10 Jumlah iterasi pada restart ke-10: 15
2	Jumlah iterasi pada restart ke-1: 17 Jumlah iterasi pada restart ke-2: 15 Jumlah iterasi pada restart ke-3: 18 Jumlah iterasi pada restart ke-4: 12 Jumlah iterasi pada restart ke-5: 21 Jumlah iterasi pada restart ke-6: 16 Jumlah iterasi pada restart ke-7: 20 Jumlah iterasi pada restart ke-8: 15 Jumlah iterasi pada restart ke-9: 12 Jumlah iterasi pada restart ke-10: 16
3	Jumlah iterasi pada restart ke-1: 17 Jumlah iterasi pada restart ke-2: 20 Jumlah iterasi pada restart ke-3: 12 Jumlah iterasi pada restart ke-4: 15 Jumlah iterasi pada restart ke-5: 19 Jumlah iterasi pada restart ke-6: 20

	Jumlah iterasi pada restart ke-7: 14 Jumlah iterasi pada restart ke-8: 16 Jumlah iterasi pada restart ke-9: 23 Jumlah iterasi pada restart ke-10: 21
--	---

Random restart hill-climbing dirancang untuk mengatasi kelemahan utama SAHC, yaitu ketidakmampuannya keluar dari *local optimum*. Algoritma ini melakukan sepuluh kali pencarian ulang dari titik awal yang acak, lalu memilih hasil terbaik dari seluruh percobaan tersebut. Dari tiga kali eksekusi, algoritma menghasilkan nilai akhir yang berbeda-beda, yaitu 1783.3894, 1783.3832, dan 1783.4198. Perbedaan ini menunjukkan adanya variasi akibat sifat acak dari proses restart.

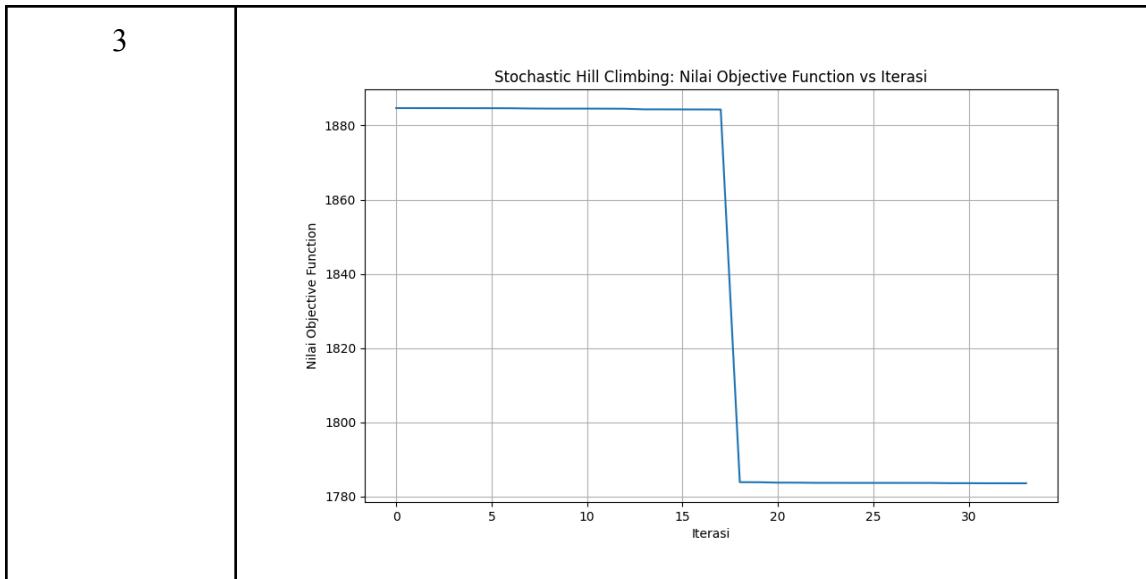
Meskipun seluruh percobaan tetap berakhir pada solusi 18 kontainer dan belum mencapai *global optimum* dengan 17 kontainer, hasil ini menunjukkan peningkatan kualitas dibanding SAHC murni. SAHC selalu berhenti pada 1783.5880, sementara *random restart* berhasil menemukan nilai yang sedikit lebih baik di sekitar 1783.38. Ini membuktikan bahwa dengan melakukan *restart* dari titik awal yang berbeda, algoritma memiliki peluang lebih besar untuk menemukan local optimum yang lebih baik.

Dari sisi waktu, *random restart* jauh lebih lambat dengan durasi sekitar lima hingga enam detik karena menjalankan sepuluh proses *hill-climbing* secara terpisah. Walau begitu, peningkatan hasil ini sebanding dengan biaya komputasi tambahan. Dibandingkan algoritma lain, *random restart* merupakan pendekatan paling eksploratif dan paling mendekati *global optimum*, meskipun tetap belum mampu menemukan solusi terbaik teoretis. Hal tersebut menjadi contoh kompromi antara kecepatan dan kualitas hasil pencarian.

d. Stochastic Hill-Climbing

Diterapkan tiga percobaan yang masing-masing memiliki konfigurasi jumlah iterasi dan jumlah populasi maksimum yang berbeda untuk algoritma Stochastic Hill-Climbing.

Percobaan	Grafik Nilai Objective Function terhadap Banyak Iterasi										
1	<p style="text-align: center;">Stochastic Hill Climbing: Nilai Objective Function vs Iterasi</p> <table border="1"> <caption>Data for Experiment 1</caption> <thead> <tr> <th>Iterasi</th> <th>Nilai Objective Function</th> </tr> </thead> <tbody> <tr><td>0</td><td>1885</td></tr> <tr><td>20</td><td>1885</td></tr> <tr><td>21</td><td>1785</td></tr> <tr><td>26</td><td>1785</td></tr> </tbody> </table>	Iterasi	Nilai Objective Function	0	1885	20	1885	21	1785	26	1785
Iterasi	Nilai Objective Function										
0	1885										
20	1885										
21	1785										
26	1785										
2	<p style="text-align: center;">Stochastic Hill Climbing: Nilai Objective Function vs Iterasi</p> <table border="1"> <caption>Data for Experiment 2</caption> <thead> <tr> <th>Iterasi</th> <th>Nilai Objective Function</th> </tr> </thead> <tbody> <tr><td>0</td><td>1885</td></tr> <tr><td>25</td><td>1885</td></tr> <tr><td>26</td><td>1785</td></tr> <tr><td>31</td><td>1785</td></tr> </tbody> </table>	Iterasi	Nilai Objective Function	0	1885	25	1885	26	1785	31	1785
Iterasi	Nilai Objective Function										
0	1885										
25	1885										
26	1785										
31	1785										



Percobaan	Initial State	Final State
1	<pre> State Awal: ----- Kontainer 1 (Total: 100/100) [BRG004, BRG008] Kontainer 2 (Total: 97/100) [BRG003, BRG012] Kontainer 3 (Total: 97/100) [BRG026, BRG028] Kontainer 4 (Total: 96/100) [BRG001, BRG002, BRG007] Kontainer 5 (Total: 96/100) [BRG005, BRG006] Kontainer 6 (Total: 95/100) [BRG013, BRG014] Kontainer 7 (Total: 95/100) [BRG019, BRG020, BRG029] Kontainer 8 (Total: 95/100) [BRG027, BRG033] Kontainer 9 (Total: 94/100) [BRG016, BRG017, BRG018, BRG024] Kontainer 10 (Total: 94/100) [BRG031, BRG032] Kontainer 11 (Total: 93/100) [BRG021, BRG022, BRG035] Kontainer 12 (Total: 92/100) [BRG009, BRG011] Kontainer 13 (Total: 92/100) [BRG023, BRG025] Kontainer 14 (Total: 90/100) [BRG010, BRG015] Kontainer 15 (Total: 85/100) [BRG034, BRG036] Kontainer 16 (Total: 78/100) [BRG030] Kontainer 17 (Total: 75/100) [BRG037, BRG038] Kontainer 18 (Total: 66/100) [BRG040] Kontainer 19 (Total: 64/100) [BRG039] ----- Total Kontainer Digunakan: 19 Nilai Objective Function: 1884.6940 </pre>	<pre> State Akhir: ----- Kontainer 1 (Total: 100/100) [BRG003, BRG029] Kontainer 2 (Total: 100/100) [BRG004, BRG008] Kontainer 3 (Total: 100/100) [BRG019, BRG006] Kontainer 4 (Total: 100/100) [BRG010, BRG033] Kontainer 5 (Total: 100/100) [BRG039, BRG014] Kontainer 6 (Total: 100/100) [BRG031, BRG017, BRG035, BRG024] Kontainer 7 (Total: 100/100) [BRG025, BRG032, BRG018] Kontainer 8 (Total: 100/100) [BRG028, BRG040] Kontainer 9 (Total: 100/100) [BRG026, BRG002] Kontainer 10 (Total: 100/100) [BRG011, BRG021, BRG015] Kontainer 11 (Total: 100/100) [BRG027, BRG038] Kontainer 12 (Total: 99/100) [BRG023, BRG016] Kontainer 13 (Total: 96/100) [BRG009, BRG036, BRG001] Kontainer 14 (Total: 96/100) [BRG034, BRG005] Kontainer 15 (Total: 95/100) [BRG012, BRG007, BRG022] Kontainer 16 (Total: 88/100) [BRG013, BRG020] Kontainer 17 (Total: 78/100) [BRG030] Kontainer 18 (Total: 42/100) [BRG037] ----- Total Kontainer Digunakan: 18 </pre>

2

State Awal:

```

Kontainer 1 (Total: 100/100)
[BRG004, BRG008]
Kontainer 2 (Total: 97/100)
[BRG003, BRG012]
Kontainer 3 (Total: 97/100)
[BRG026, BRG028]
Kontainer 4 (Total: 96/100)
[BRG001, BRG002, BRG007]
Kontainer 5 (Total: 96/100)
[BRG005, BRG006]
Kontainer 6 (Total: 95/100)
[BRG013, BRG014]
Kontainer 7 (Total: 95/100)
[BRG019, BRG020, BRG029]
Kontainer 8 (Total: 95/100)
[BRG027, BRG033]
Kontainer 9 (Total: 94/100)
[BRG016, BRG017, BRG018, BRG024]
Kontainer 10 (Total: 94/100)
[BRG031, BRG032]
Kontainer 11 (Total: 93/100)
[BRG021, BRG022, BRG035]
Kontainer 12 (Total: 92/100)
[BRG009, BRG011]
Kontainer 13 (Total: 92/100)
[BRG023, BRG025]
Kontainer 14 (Total: 90/100)
[BRG010, BRG015]
Kontainer 15 (Total: 85/100)
[BRG034, BRG036]
Kontainer 16 (Total: 78/100)
[BRG030]
Kontainer 17 (Total: 75/100)
[BRG037, BRG038]
Kontainer 18 (Total: 66/100)
[BRG040]
Kontainer 19 (Total: 64/100)
[BRG039]

```

Total Kontainer Digunakan: 19
Nilai Objective Function: 1884.6940

State Akhir:

```

Kontainer 1 (Total: 100/100)
[BRG017, BRG014, BRG028, BRG018]
Kontainer 2 (Total: 100/100)
[BRG004, BRG008]
Kontainer 3 (Total: 100/100)
[BRG013, BRG001, BRG015]
Kontainer 4 (Total: 100/100)
[BRG034, BRG019]
Kontainer 5 (Total: 100/100)
[BRG011, BRG036, BRG024]
Kontainer 6 (Total: 100/100)
[BRG020, BRG006, BRG012]
Kontainer 7 (Total: 100/100)
[BRG039, BRG007]
Kontainer 8 (Total: 100/100)
[BRG010, BRG033]
Kontainer 9 (Total: 100/100)
[BRG037, BRG025, BRG029]
Kontainer 10 (Total: 100/100)
[BRG027, BRG038]
Kontainer 11 (Total: 100/100)
[BRG026, BRG002]
Kontainer 12 (Total: 99/100)
[BRG003, BRG035]
Kontainer 13 (Total: 99/100)
[BRG016, BRG023]
Kontainer 14 (Total: 98/100)
[BRG040, BRG021]
Kontainer 15 (Total: 94/100)
[BRG031, BRG032]
Kontainer 16 (Total: 84/100)
[BRG005, BRG022]
Kontainer 17 (Total: 78/100)
[BRG030]
Kontainer 18 (Total: 42/100)
[BRG009]

```

Total Kontainer Digunakan: 18

3

State Awal:

```

Kontainer 1 (Total: 100/100)
[BRG004, BRG008]
Kontainer 2 (Total: 97/100)
[BRG003, BRG012]
Kontainer 3 (Total: 97/100)
[BRG026, BRG028]
Kontainer 4 (Total: 96/100)
[BRG001, BRG002, BRG007]
Kontainer 5 (Total: 96/100)
[BRG005, BRG006]
Kontainer 6 (Total: 95/100)
[BRG013, BRG014]
Kontainer 7 (Total: 95/100)
[BRG019, BRG020, BRG029]
Kontainer 8 (Total: 95/100)
[BRG027, BRG033]
Kontainer 9 (Total: 94/100)
[BRG016, BRG017, BRG018, BRG024]
Kontainer 10 (Total: 94/100)
[BRG031, BRG032]
Kontainer 11 (Total: 93/100)
[BRG021, BRG022, BRG035]
Kontainer 12 (Total: 92/100)
[BRG009, BRG011]
Kontainer 13 (Total: 92/100)
[BRG023, BRG025]
Kontainer 14 (Total: 90/100)
[BRG010, BRG015]
Kontainer 15 (Total: 85/100)
[BRG034, BRG036]
Kontainer 16 (Total: 78/100)
[BRG030]
Kontainer 17 (Total: 75/100)
[BRG037, BRG038]
Kontainer 18 (Total: 66/100)
[BRG040]
Kontainer 19 (Total: 64/100)
[BRG039]

```

Total Kontainer Digunakan: 19
Nilai Objective Function: 1884.6940

State Akhir:

```

Kontainer 1 (Total: 100/100)
[BRG004, BRG008]
Kontainer 2 (Total: 100/100)
[BRG019, BRG006]
Kontainer 3 (Total: 100/100)
[BRG016, BRG025, BRG012]
Kontainer 4 (Total: 100/100)
[BRG002, BRG026]
Kontainer 5 (Total: 100/100)
[BRG040, BRG028]
Kontainer 6 (Total: 100/100)
[BRG034, BRG033, BRG015]
Kontainer 7 (Total: 100/100)
[BRG039, BRG014]
Kontainer 8 (Total: 100/100)
[BRG029, BRG038, BRG031]
Kontainer 9 (Total: 100/100)
[BRG036, BRG023, BRG018]
Kontainer 10 (Total: 99/100)
[BRG037, BRG017, BRG005]
Kontainer 11 (Total: 99/100)
[BRG003, BRG024]
Kontainer 12 (Total: 99/100)
[BRG027, BRG021]
Kontainer 13 (Total: 97/100)
[BRG030, BRG035]
Kontainer 14 (Total: 97/100)
[BRG011, BRG032]
Kontainer 15 (Total: 95/100)
[BRG007, BRG013]
Kontainer 16 (Total: 95/100)
[BRG010, BRG001]
Kontainer 17 (Total: 84/100)
[BRG022, BRG009]
Kontainer 18 (Total: 29/100)
[BRG020]

```

Total Kontainer Digunakan: 18

Parameter	Percobaan 1	Percobaan 2	Percobaan 3
Durasi (s)	0.9847	1.1699	1.1650
Jumlah Iterasi	28	34	33
Initial State	1884.6940	1884.6940	1884.6940
Final State	1783.7150	1783.7054	1783.5832

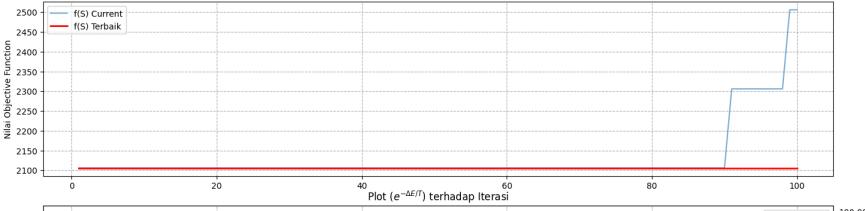
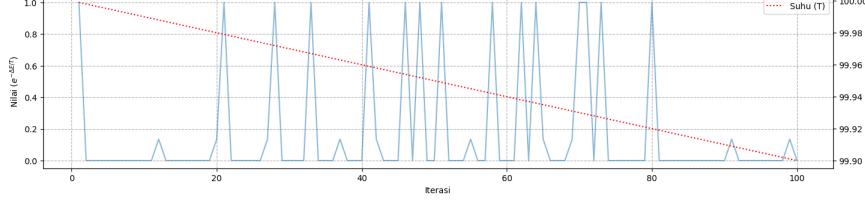
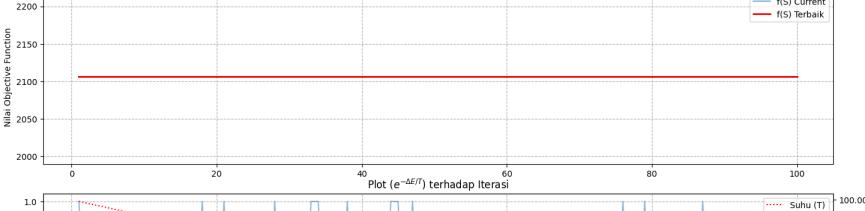
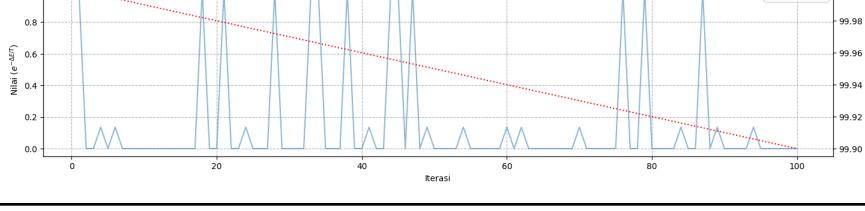
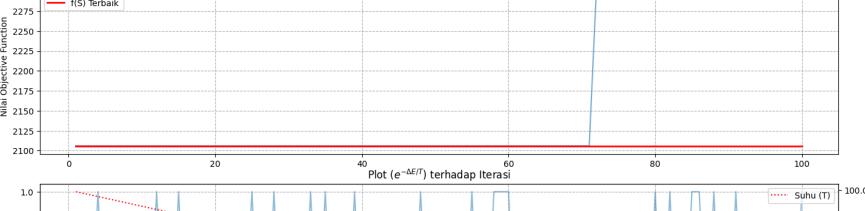
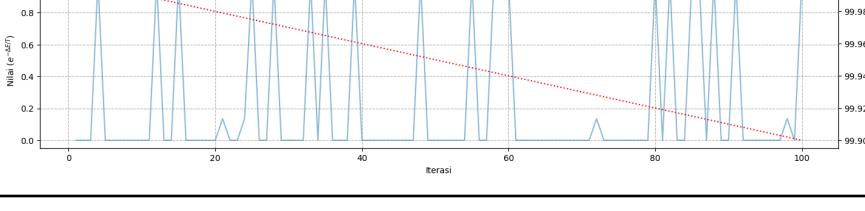
Stochastic hill-climbing menunjukkan sifat yang lebih acak dibanding ketiga algoritma sebelumnya. Dari tiga percobaan dengan kondisi awal yang sama, hasil akhirnya berbeda-beda dengan nilai 1783.7150, 1783.7054, dan 1783.5832. Perbedaan ini disebabkan oleh mekanisme pemilihan tetangga secara acak di setiap iterasi, sehingga setiap percobaan menempuh jalur pencarian yang berbeda.

Hasil eksperimen menunjukkan bahwa algoritma ini kadang mampu menemukan solusi yang sedikit lebih baik dari SAHC, seperti pada percobaan ketiga dengan nilai 1783.5832. Namun, pada percobaan lain, hasilnya justru lebih buruk. Dengan demikian, *stochastic hill-climbing* mampu memberikan variasi hasil yang lebih luas, tetapi tidak menjamin peningkatan kualitas secara konsisten. Sama seperti SAHC, algoritma ini tetap tidak bisa keluar dari local optimum karena tidak menerima langkah yang memperburuk nilai fungsi objektif.

Waktu eksekusi rata-ratanya sekitar satu detik, sedikit lebih lambat dari SAHC tetapi jauh lebih cepat dibanding *random restart*. Jika dibandingkan, *stochastic hill-climbing* menawarkan keseimbangan antara kecepatan dan keragaman hasil, tetapi masih kalah dalam hal kemampuan mendekati global optimum. Sifat acaknya membantu eksplorasi ruang solusi, namun tanpa mekanisme pelarian dari puncak local optimum, hasil terbaiknya tetap terbatas pada solusi 18 kontainer.

e. Simulated Annealing

Dilakukan tiga percobaan untuk algoritma Simulated Annealing dengan MAX_ITER = 100.

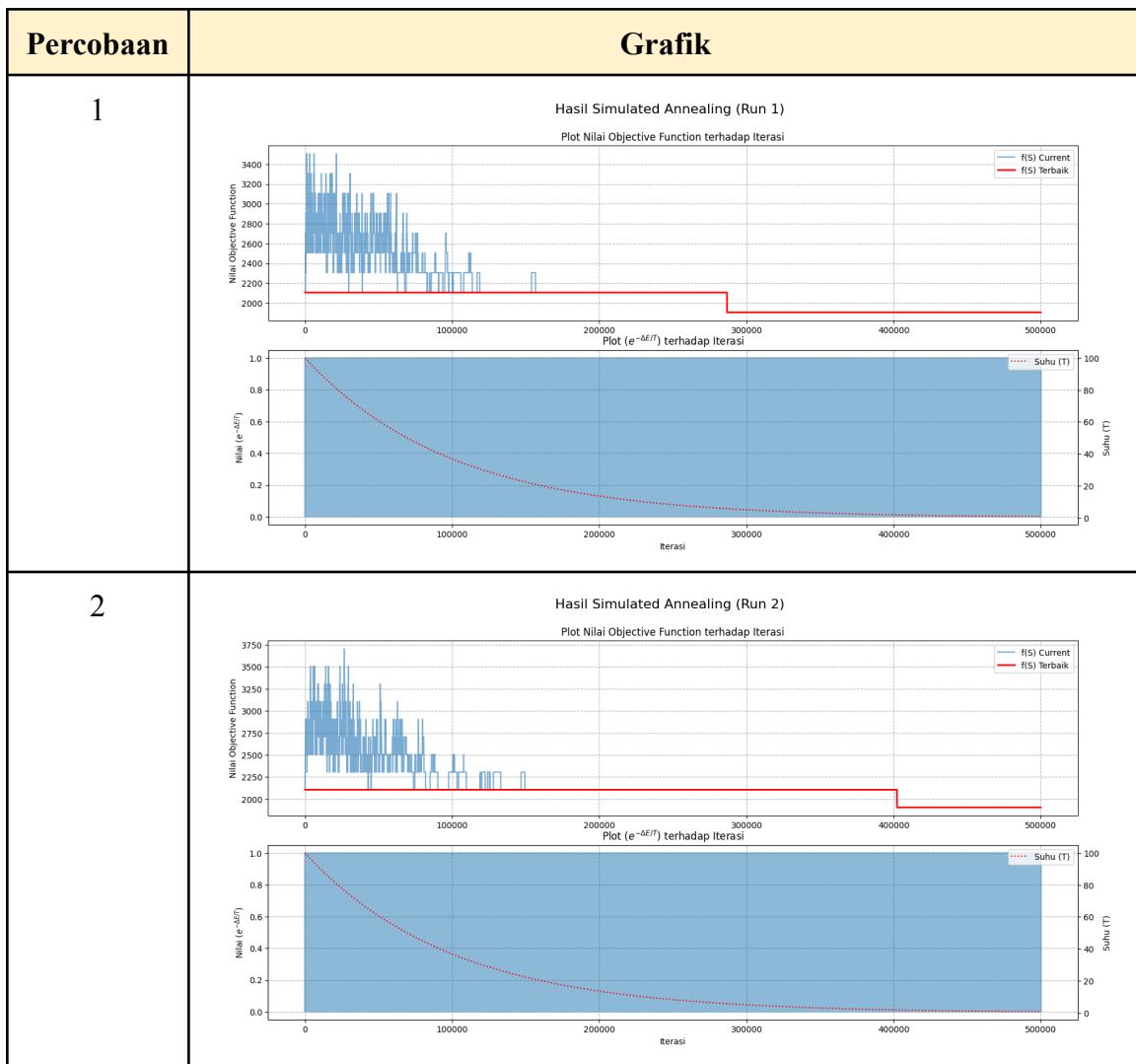
Percobaan	Grafik
1	<p style="text-align: center;">Hasil Simulated Annealing (Run 1)</p> <p style="text-align: center;">Plot Nilai Objective Function terhadap Iterasi</p>  <p style="text-align: center;">Plot ($e^{-\Delta E/T}$) terhadap Iterasi</p> 
2	<p style="text-align: center;">Hasil Simulated Annealing (Run 2)</p> <p style="text-align: center;">Plot Nilai Objective Function terhadap Iterasi</p>  <p style="text-align: center;">Plot ($e^{-\Delta E/T}$) terhadap Iterasi</p> 
3	<p style="text-align: center;">Hasil Simulated Annealing (Run 3)</p> <p style="text-align: center;">Plot Nilai Objective Function terhadap Iterasi</p>  <p style="text-align: center;">Plot ($e^{-\Delta E/T}$) terhadap Iterasi</p> 

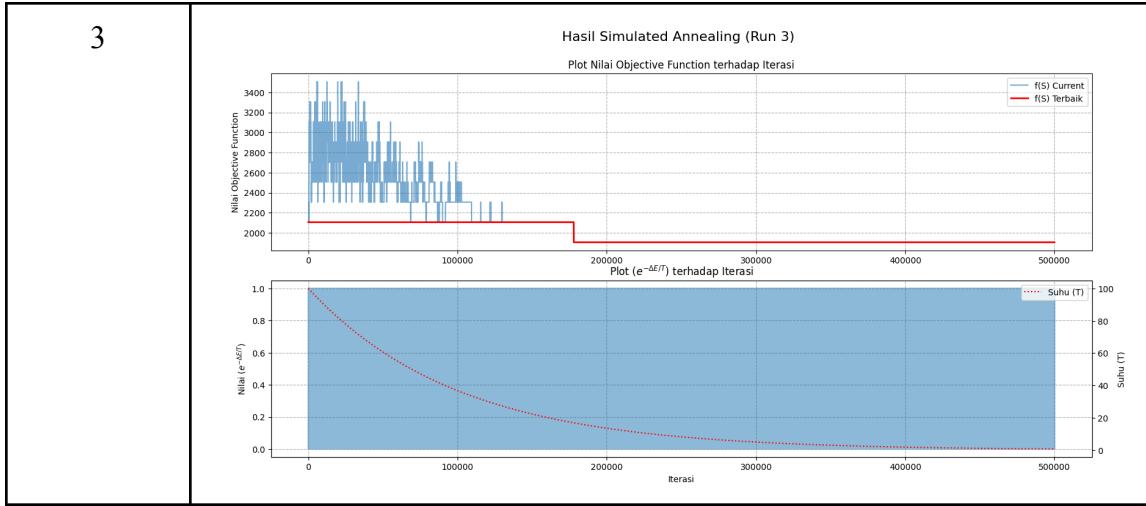
Berikut merupakan *state* awal dan *state* akhir dari masing-masing percobaan.

Percobaan	Initial State	Final State
1	<pre>--- State Awal (19 Kontainer) --- Kontainer 1 [OK]: Total = 99/100. Barang: BRG006 (54), BRG016 (45) Kontainer 2 [OK]: Total = 95/100. Barang: BRG022 (42), BRG040 (36), BRG012 (17) Kontainer 3 [OK]: Total = 98/100. Barang: BRG008 (26), BRG040 (34) Kontainer 4 [OK]: Total = 96/100. Barang: BRG030 (33), BRG005 (19) Kontainer 5 [OK]: Total = 87/100. Barang: BRG002 (37), BRG015 (59) Kontainer 6 [OK]: Total = 87/100. Barang: BRG027 (57), BRG029 (20) Kontainer 7 [OK]: Total = 95/100. Barang: BRG001 (80), BRG018 (15) Kontainer 8 [OK]: Total = 96/100. Barang: BRG037 (47), BRG021 (23), BRG008 (26) Kontainer 9 [OK]: Total = 88/100. Barang: BRG036 (31), BRG017 (15) Kontainer 10 [OK]: Total = 94/100. Barang: BRG040 (66), BRG033 (28) Kontainer 11 [OK]: Total = 91/100. Barang: BRG027 (67), BRG024 (19) Kontainer 12 [OK]: Total = 99/100. Barang: BRG038 (33), BRG031 (47), BRG035 (19) Kontainer 13 [OK]: Total = 88/100. Barang: BRG009 (42), BRG019 (46) Kontainer 14 [OK]: Total = 92/100. Barang: BRG026 (61), BRG020 (29) Kontainer 15 [OK]: Total = 78/100. Barang: BRG025 (38), BRG021 (32) Kontainer 16 [OK]: Total = 95/100. Barang: BRG014 (36), BRG013 (59) Kontainer 17 [OK]: Total = 74/100. Barang: BRG004 (74) Kontainer 18 [OK]: Total = 96/100. Barang: BRG023 (54), BRG037 (42) Kontainer 19 [OK]: Total = 54/100. Barang: BRG034 (54) Objective Function Score: 2106.0000</pre>	<pre>--- State Akhir (Terbaik) (19 Kontainer) --- Kontainer 1 [OK]: Total = 95/100. Barang: BRG003 (80), BRG019 (15) Kontainer 2 [OK]: Total = 95/100. Barang: BRG022 (42), BRG007 (36), BRG012 (17) Kontainer 3 [OK]: Total = 98/100. Barang: BRG039 (64), BRG028 (24) Kontainer 4 [OK]: Total = 96/100. Barang: BRG030 (78), BRG015 (18) Kontainer 5 [OK]: Total = 87/100. Barang: BRG002 (37), BRG011 (50) Kontainer 6 [OK]: Total = 87/100. Barang: BRG027 (67), BRG029 (20) Kontainer 7 [OK]: Total = 95/100. Barang: BRG003 (80), BRG019 (15) Kontainer 8 [OK]: Total = 96/100. Barang: BRG032 (47), BRG006 (23), BRG008 (26) Kontainer 9 [OK]: Total = 88/100. Barang: BRG045 (47), BRG016 (31), BRG017 (15) Kontainer 10 [OK]: Total = 97/100. Barang: BRG025 (38), BRG022 (42), BRG003 (66), BRG033 (28) Kontainer 11 [OK]: Total = 91/100. Barang: BRG031 (72), BRG024 (19) Kontainer 12 [OK]: Total = 99/100. Barang: BRG038 (33), BRG031 (47), BRG035 (19) Kontainer 13 [OK]: Total = 89/100. Barang: BRG009 (42), BRG019 (46) Kontainer 14 [OK]: Total = 92/100. Barang: BRG026 (63), BRG020 (29) Kontainer 15 [OK]: Total = 79/100. Barang: BRG025 (38), BRG021 (32) Kontainer 16 [OK]: Total = 95/100. Barang: BRG014 (36), BRG013 (59) Kontainer 17 [OK]: Total = 74/100. Barang: BRG004 (74) Kontainer 18 [OK]: Total = 96/100. Barang: BRG023 (54), BRG037 (42) Kontainer 19 [OK]: Total = 54/100. Barang: BRG034 (54) Objective Function Score: 2106.0000</pre>
2	<pre>--- State Awal (19 Kontainer) --- Kontainer 1 [OK]: Total = 93/100. Barang: BRG020 (29), BRG039 (64) Kontainer 2 [OK]: Total = 98/100. Barang: BRG037 (42), BRG035 (19), BRG002 (37) Kontainer 3 [OK]: Total = 99/100. Barang: BRG008 (26), BRG012 (17), BRG038 (33), BRG001 (23) Kontainer 4 [OK]: Total = 80/100. Barang: BRG037 (67), BRG027 (19) Kontainer 5 [OK]: Total = 91/100. Barang: BRG025 (38), BRG022 (42), BRG003 (66) Kontainer 6 [OK]: Total = 98/100. Barang: BRG033 (28), BRG025 (38), BRG021 (32) Kontainer 7 [OK]: Total = 95/100. Barang: BRG005 (42), BRG029 (20), BRG018 (15), BRG015 (18) Kontainer 8 [OK]: Total = 99/100. Barang: BRG026 (63), BRG007 (36) Kontainer 9 [OK]: Total = 95/100. Barang: BRG013 (59), BRG014 (36) Kontainer 10 [OK]: Total = 96/100. Barang: BRG002 (42), BRG023 (54) Kontainer 11 [OK]: Total = 90/100. Barang: BRG001 (54), BRG049 (42) Kontainer 12 [OK]: Total = 92/100. Barang: BRG016 (45), BRG032 (47) Kontainer 13 [OK]: Total = 72/100. Barang: BRG010 (72) Kontainer 14 [OK]: Total = 78/100. Barang: BRG030 (78) Kontainer 15 [OK]: Total = 81/100. Barang: BRG028 (34), BRG031 (47) Kontainer 16 [OK]: Total = 100/100. Barang: BRG019 (46), BRG034 (54) Kontainer 17 [OK]: Total = 85/100. Barang: BRG006 (54), BRG036 (31) Kontainer 18 [OK]: Total = 88/100. Barang: BRG004 (66), BRG006 (66) Kontainer 19 [OK]: Total = 74/100. Barang: BRG004 (74) Objective Function Score: 2106.0000</pre>	<pre>--- State Akhir (Terbaik) (19 Kontainer) --- Kontainer 1 [OK]: Total = 93/100. Barang: BRG020 (29), BRG039 (64) Kontainer 2 [OK]: Total = 98/100. Barang: BRG037 (42), BRG035 (19), BRG002 (37) Kontainer 3 [OK]: Total = 99/100. Barang: BRG008 (26), BRG012 (17), BRG038 (33), BRG001 (23) Kontainer 4 [OK]: Total = 80/100. Barang: BRG037 (67), BRG027 (19) Kontainer 5 [OK]: Total = 91/100. Barang: BRG025 (38), BRG022 (42), BRG003 (66) Kontainer 6 [OK]: Total = 98/100. Barang: BRG033 (28), BRG025 (38), BRG021 (32) Kontainer 7 [OK]: Total = 95/100. Barang: BRG005 (42), BRG029 (20), BRG018 (15), BRG015 (18) Kontainer 8 [OK]: Total = 99/100. Barang: BRG026 (63), BRG007 (36) Kontainer 9 [OK]: Total = 95/100. Barang: BRG013 (59), BRG014 (36) Kontainer 10 [OK]: Total = 96/100. Barang: BRG002 (42), BRG023 (54) Kontainer 11 [OK]: Total = 90/100. Barang: BRG001 (54), BRG049 (42) Kontainer 12 [OK]: Total = 92/100. Barang: BRG016 (45), BRG032 (47) Kontainer 13 [OK]: Total = 72/100. Barang: BRG010 (72) Kontainer 14 [OK]: Total = 78/100. Barang: BRG030 (78) Kontainer 15 [OK]: Total = 81/100. Barang: BRG028 (34), BRG031 (47) Kontainer 16 [OK]: Total = 100/100. Barang: BRG019 (46), BRG034 (54) Kontainer 17 [OK]: Total = 85/100. Barang: BRG006 (54), BRG036 (31) Kontainer 18 [OK]: Total = 66/100. Barang: BRG004 (66) Kontainer 19 [OK]: Total = 74/100. Barang: BRG004 (74) Objective Function Score: 2106.0000</pre>
3	<pre>--- State Awal (19 Kontainer) --- Kontainer 1 [OK]: Total = 100/100. Barang: BRG013 (59), BRG008 (26), BRG017 (15) Kontainer 2 [OK]: Total = 95/100. Barang: BRG026 (63), BRG021 (32) Kontainer 3 [OK]: Total = 97/100. Barang: BRG031 (47), BRG012 (17), BRG038 (33) Kontainer 4 [OK]: Total = 94/100. Barang: BRG040 (66), BRG033 (28) Kontainer 5 [OK]: Total = 91/100. Barang: BRG006 (54), BRG002 (37) Kontainer 6 [OK]: Total = 100/100. Barang: BRG025 (38), BRG022 (42), BRG029 (28) Kontainer 7 [OK]: Total = 88/100. Barang: BRG019 (46), BRG009 (42) Kontainer 8 [OK]: Total = 88/100. Barang: BRG036 (31), BRG005 (42), BRG018 (15) Kontainer 9 [OK]: Total = 82/100. Barang: BRG015 (64), BRG015 (18) Kontainer 10 [OK]: Total = 88/100. Barang: BRG023 (54), BRG028 (34) Kontainer 11 [OK]: Total = 97/100. Barang: BRG032 (47), BRG011 (50) Kontainer 12 [OK]: Total = 96/100. Barang: BRG034 (54), BRG001 (23), BRG035 (19) Kontainer 13 [OK]: Total = 91/100. Barang: BRG001 (23), BRG035 (19) Kontainer 14 [OK]: Total = 96/100. Barang: BRG010 (72), BRG024 (19) Kontainer 15 [OK]: Total = 72/100. Barang: BRG007 (36), BRG014 (36) Kontainer 16 [OK]: Total = 78/100. Barang: BRG030 (78) Kontainer 17 [OK]: Total = 87/100. Barang: BRG016 (45), BRG037 (42) Kontainer 18 [OK]: Total = 74/100. Barang: BRG004 (74) Kontainer 19 [OK]: Total = 80/100. Barang: BRG003 (80) Objective Function Score: 2106.0000</pre>	<pre>--- State Akhir (Terbaik) (19 Kontainer) --- Kontainer 1 [OK]: Total = 100/100. Barang: BRG013 (59), BRG008 (26), BRG017 (15) Kontainer 2 [OK]: Total = 95/100. Barang: BRG026 (63), BRG021 (32) Kontainer 3 [OK]: Total = 97/100. Barang: BRG031 (47), BRG012 (17), BRG038 (33) Kontainer 4 [OK]: Total = 94/100. Barang: BRG040 (66), BRG033 (28) Kontainer 5 [OK]: Total = 91/100. Barang: BRG000 (54), BRG002 (37) Kontainer 6 [OK]: Total = 100/100. Barang: BRG025 (38), BRG022 (42), BRG029 (28) Kontainer 7 [OK]: Total = 88/100. Barang: BRG019 (46), BRG009 (42) Kontainer 8 [OK]: Total = 88/100. Barang: BRG036 (31), BRG005 (42), BRG018 (15) Kontainer 9 [OK]: Total = 92/100. Barang: BRG039 (64), BRG015 (18) Kontainer 10 [OK]: Total = 88/100. Barang: BRG001 (23), BRG035 (19) Kontainer 11 [OK]: Total = 95/100. Barang: BRG001 (23), BRG035 (19) Kontainer 12 [OK]: Total = 96/100. Barang: BRG034 (54), BRG001 (23), BRG035 (19) Kontainer 13 [OK]: Total = 91/100. Barang: BRG010 (72), BRG024 (19) Kontainer 14 [OK]: Total = 96/100. Barang: BRG027 (67), BRG020 (29) Kontainer 15 [OK]: Total = 72/100. Barang: BRG007 (36), BRG014 (36) Kontainer 16 [OK]: Total = 78/100. Barang: BRG030 (78) Kontainer 17 [OK]: Total = 87/100. Barang: BRG016 (45), BRG037 (42) Kontainer 18 [OK]: Total = 74/100. Barang: BRG004 (74) Kontainer 19 [OK]: Total = 80/100. Barang: BRG003 (80) Objective Function Score: 2106.0000</pre>

Parameter	Percobaan 1	Percobaan 2	Percobaan 3
Durasi (s)	0.0030	0.0105	0.0084
Frekuensi Stuck	17	13	21
Rasio Penerimaan Move Buruk	0.17	0.13	0.21
Nilai Objective Function Awal	2106	2106	2106
Nilai Objective Function Akhir	2106	2106	2106

Melihat nilai *objective function* yang tidak berubah, dilakukan kembali tiga percobaan untuk algoritma Simulated Annealing dengan MAX_ITER = 500.000.





Berikut merupakan *state* awal dan *state* akhir dari masing-masing percobaan.

Percobaan	Initial State	Final State
1	<pre>-- State Awal (19 Kontainer) -- Kontainer 1 [OK]: Total = 89/100, Barang: BRG031 (47), BRG022 (42) Kontainer 2 [OK]: Total = 87/100, Barang: BRG014 (45), BRG000 (45) Kontainer 3 [OK]: Total = 109/100, Barang: BRG008 (74), BRG000 (26) Kontainer 4 [OK]: Total = 94/100, Barang: BRG040 (66), BRG033 (28) Kontainer 5 [OK]: Total = 100/100, Barang: BRG037 (42), BRG029 (20), BRG001 (23), BRG018 (15) Kontainer 6 [OK]: Total = 98/100, Barang: BRG034 (54), BRG020 (29), BRG017 (15) Kontainer 7 [OK]: Total = 86/100, Barang: BRG027 (67), BRG024 (19) Kontainer 8 [OK]: Total = 98/100, Barang: BRG002 (55), BRG003 (45), BRG035 (19) Kontainer 9 [OK]: Total = 91/100, Barang: BRG014 (51), BRG017 (24), BRG018 (23) Kontainer 10 [OK]: Total = 95/100, Barang: BRG026 (63), BRG021 (32) Kontainer 11 [OK]: Total = 90/100, Barang: BRG013 (59), BRG016 (31) Kontainer 12 [OK]: Total = 90/100, Barang: BRG015 (72), BRG015 (18) Kontainer 13 [OK]: Total = 100/100, Barang: BRG039 (64), BRG014 (36) Kontainer 14 [OK]: Total = 91/100, Barang: BRG025 (38), BRG011 (47), BRG018 (17), BRG007 (36) Kontainer 15 [OK]: Total = 96/100, Barang: BRG011 (50), BRG019 (46) Kontainer 16 [OK]: Total = 80/100, Barang: BRG003 (80) Kontainer 17 [OK]: Total = 97/100, Barang: BRG010 (78) Kontainer 18 [OK]: Total = 88/100, Barang: BRG003 (54), BRG028 (34) Kontainer 19 [OK]: Total = 47/100, Barang: BRG032 (47) Objective Function Score: 2106,0000</pre>	<pre>-- State Akhir (Terbaik) (18 Kontainer) -- Kontainer 1 [OK]: Total = 98/100, Barang: BRG029 (20), BRG016 (45), BRG036 (31) Kontainer 2 [OK]: Total = 85/100, Barang: BRG025 (62), BRG021 (32) Kontainer 3 [OK]: Total = 89/100, Barang: BRG004 (74), BRG018 (15) Kontainer 4 [OK]: Total = 93/100, Barang: BRG020 (29), BRG039 (64) Kontainer 5 [OK]: Total = 100/100, Barang: BRG033 (28), BRG010 (72) Kontainer 6 [OK]: Total = 86/100, Barang: BRG012 (17), BRG007 (36), BRG038 (33) Kontainer 7 [OK]: Total = 86/100, Barang: BRG002 (37), BRG011 (59) Kontainer 8 [OK]: Total = 93/100, Barang: BRG008 (26), BRG027 (67) Kontainer 9 [OK]: Total = 98/100, Barang: BRG015 (18), BRG022 (42), BRG025 (38) Kontainer 10 [OK]: Total = 100/100, Barang: BRG006 (54), BRG019 (46) Kontainer 11 [OK]: Total = 89/100, Barang: BRG009 (42), BRG032 (47) Kontainer 12 [OK]: Total = 95/100, Barang: BRG034 (54), BRG008 (42) Kontainer 13 [OK]: Total = 86/100, Barang: BRG014 (36), BRG011 (50) Kontainer 14 [OK]: Total = 100/100, Barang: BRG040 (66), BRG028 (34) Kontainer 15 [OK]: Total = 99/100, Barang: BRG003 (80), BRG035 (19) Kontainer 16 [OK]: Total = 95/100, Barang: BRG023 (54), BRG037 (42) Kontainer 17 [OK]: Total = 87/100, Barang: BRG001 (23), BRG001 (45), BRG024 (19) Kontainer 18 [OK]: Total = 98/100, Barang: BRG030 (78), BRG017 (19) Objective Function Score: 1900,0000</pre>
2	<pre>-- State Awal (18 Kontainer) -- Kontainer 1 [OK]: Total = 91/100, Barang: BRG029 (20), BRG016 (45), BRG008 (26) Kontainer 2 [OK]: Total = 93/100, Barang: BRG007 (36), BRG037 (42), BRG017 (15) Kontainer 3 [OK]: Total = 100/100, Barang: BRG025 (38), BRG011 (47), BRG018 (15) Kontainer 4 [OK]: Total = 100/100, Barang: BRG026 (63), BRG002 (37) Kontainer 5 [OK]: Total = 100/100, Barang: BRG003 (46), BRG019 (46) Kontainer 6 [OK]: Total = 95/100, Barang: BRG027 (67), BRG033 (28) Kontainer 7 [OK]: Total = 100/100, Barang: BRG014 (26), BRG012 (17) Kontainer 8 [OK]: Total = 90/100, Barang: BRG010 (72), BRG015 (18) Kontainer 9 [OK]: Total = 97/100, Barang: BRG040 (66), BRG036 (31) Kontainer 10 [OK]: Total = 80/100, Barang: BRG002 (47), BRG022 (42), BRG024 (19) Kontainer 11 [OK]: Total = 92/100, Barang: BRG001 (59), BRG038 (33) Kontainer 12 [OK]: Total = 97/100, Barang: BRG004 (74), BRG001 (23) Kontainer 13 [OK]: Total = 98/100, Barang: BRG020 (29), BRG022 (42), BRG024 (19) Kontainer 14 [OK]: Total = 96/100, Barang: BRG009 (42), BRG034 (54) Kontainer 15 [OK]: Total = 99/100, Barang: BRG003 (80), BRG035 (19) Kontainer 16 [OK]: Total = 98/100, Barang: BRG039 (64), BRG028 (34) Kontainer 17 [OK]: Total = 92/100, Barang: BRG005 (42), BRG011 (50) Kontainer 18 [OK]: Total = 78/100, Barang: BRG030 (78) Objective Function Score: 1906,0000</pre>	<pre>-- State Akhir (Terbaik) (18 Kontainer) -- Kontainer 1 [OK]: Total = 91/100, Barang: BRG029 (20), BRG016 (45), BRG008 (26) Kontainer 2 [OK]: Total = 93/100, Barang: BRG007 (36), BRG037 (42), BRG017 (15) Kontainer 3 [OK]: Total = 100/100, Barang: BRG025 (38), BRG031 (47), BRG018 (15) Kontainer 4 [OK]: Total = 100/100, Barang: BRG026 (61), BRG002 (37) Kontainer 5 [OK]: Total = 95/100, Barang: BRG027 (67), BRG033 (28) Kontainer 6 [OK]: Total = 100/100, Barang: BRG032 (54), BRG019 (46) Kontainer 7 [OK]: Total = 97/100, Barang: BRG010 (72), BRG014 (36), BRG012 (17) Kontainer 8 [OK]: Total = 90/100, Barang: BRG004 (66), BRG031 (31) Kontainer 9 [OK]: Total = 97/100, Barang: BRG006 (54), BRG021 (32) Kontainer 10 [OK]: Total = 92/100, Barang: BRG013 (59), BRG038 (33) Kontainer 11 [OK]: Total = 97/100, Barang: BRG004 (74), BRG001 (23) Kontainer 12 [OK]: Total = 90/100, Barang: BRG020 (29), BRG022 (42), BRG024 (19) Kontainer 13 [OK]: Total = 96/100, Barang: BRG009 (42), BRG034 (54) Kontainer 14 [OK]: Total = 99/100, Barang: BRG003 (80), BRG035 (19) Kontainer 15 [OK]: Total = 98/100, Barang: BRG039 (64), BRG028 (34) Kontainer 16 [OK]: Total = 92/100, Barang: BRG005 (42), BRG011 (50) Kontainer 17 [OK]: Total = 78/100, Barang: BRG030 (78) Objective Function Score: 1906,0000</pre>
3	<pre>-- State Awal (19 Kontainer) -- Kontainer 1 [OK]: Total = 98/100, Barang: BRG030 (78), BRG029 (20) Kontainer 2 [OK]: Total = 89/100, Barang: BRG004 (74), BRG017 (15) Kontainer 3 [OK]: Total = 90/100, Barang: BRG006 (54), BRG007 (36) Kontainer 4 [OK]: Total = 99/100, Barang: BRG003 (80), BRG024 (19) Kontainer 5 [OK]: Total = 99/100, Barang: BRG020 (29), BRG039 (64), BRG002 (37) Kontainer 6 [OK]: Total = 95/100, Barang: BRG010 (72), BRG001 (23) Kontainer 7 [OK]: Total = 89/100, Barang: BRG037 (42), BRG031 (47) Kontainer 8 [OK]: Total = 98/100, Barang: BRG027 (67), BRG036 (31) Kontainer 9 [OK]: Total = 98/100, Barang: BRG040 (66), BRG021 (32) Kontainer 10 [OK]: Total = 85/100, Barang: BRG009 (42), BRG018 (15), BRG033 (28) Kontainer 11 [OK]: Total = 95/100, Barang: BRG008 (26), BRG035 (19), BRG011 (50) Kontainer 12 [OK]: Total = 99/100, Barang: BRG002 (47), BRG028 (34), BRG015 (18) Kontainer 13 [OK]: Total = 99/100, Barang: BRG003 (54), BRG014 (36) Kontainer 14 [OK]: Total = 81/100, Barang: BRG039 (64), BRG012 (17) Kontainer 15 [OK]: Total = 88/100, Barang: BRG019 (46), BRG022 (42) Kontainer 16 [OK]: Total = 99/100, Barang: BRG016 (45), BRG034 (54) Kontainer 17 [OK]: Total = 97/100, Barang: BRG001 (59), BRG025 (38) Kontainer 18 [OK]: Total = 47/100, Barang: BRG009 (42) Kontainer 19 [OK]: Total = 63/100, Barang: BRG026 (63) Objective Function Score: 2106,0000</pre>	<pre>-- State Akhir (Terbaik) (19 Kontainer) -- Kontainer 1 [OK]: Total = 98/100, Barang: BRG030 (78), BRG029 (20) Kontainer 2 [OK]: Total = 89/100, Barang: BRG004 (74), BRG017 (15) Kontainer 3 [OK]: Total = 90/100, Barang: BRG006 (54), BRG007 (36) Kontainer 4 [OK]: Total = 99/100, Barang: BRG003 (80), BRG024 (19) Kontainer 5 [OK]: Total = 99/100, Barang: BRG020 (29), BRG038 (33), BRG002 (37) Kontainer 6 [OK]: Total = 95/100, Barang: BRG010 (72), BRG001 (23) Kontainer 7 [OK]: Total = 89/100, Barang: BRG037 (42), BRG031 (47) Kontainer 8 [OK]: Total = 98/100, Barang: BRG027 (67), BRG036 (31) Kontainer 9 [OK]: Total = 98/100, Barang: BRG040 (66), BRG021 (32) Kontainer 10 [OK]: Total = 85/100, Barang: BRG009 (42), BRG018 (15), BRG033 (28) Kontainer 11 [OK]: Total = 95/100, Barang: BRG008 (26), BRG035 (19), BRG011 (50) Kontainer 12 [OK]: Total = 99/100, Barang: BRG002 (47), BRG028 (34), BRG015 (18) Kontainer 13 [OK]: Total = 99/100, Barang: BRG003 (54), BRG014 (36) Kontainer 14 [OK]: Total = 81/100, Barang: BRG039 (64), BRG012 (17) Kontainer 15 [OK]: Total = 88/100, Barang: BRG019 (46), BRG022 (42) Kontainer 16 [OK]: Total = 99/100, Barang: BRG016 (45), BRG034 (54) Kontainer 17 [OK]: Total = 97/100, Barang: BRG001 (59), BRG025 (38) Kontainer 18 [OK]: Total = 47/100, Barang: BRG005 (42) Kontainer 19 [OK]: Total = 63/100, Barang: BRG026 (63) Objective Function Score: 2106,0000</pre>

Parameter	Percobaan 1	Percobaan 2	Percobaan 3
Durasi (s)	20.8143	16.2437	12.9654
Frekuensi Stuck	89528	86074	108546
Rasio Penerimaan Move Buruk	0.1791	0.1721	0.2171
Nilai Objective Function Awal	2106	1906	2106
Nilai Objective Function Akhir	1906	1906	2106

Eksperimen komparatif dilakukan untuk menguji dampak dari jumlah iterasi maksimum (MAX_ITER) terhadap kinerja algoritma Simulated Annealing (SA). Dua skenario diuji, yaitu batas iterasi rendah (MAX_ITER = 100) dan batas iterasi tinggi (MAX_ITER = 500.000).

Pada konfigurasi MAX_ITER = 100, algoritma terbukti gagal total, karena waktu eksekusi hanya beberapa milidetik dan hanya sekitar 20 langkah buruk yang diterima. Jumlah kontainer tetap stagnan pada 19 di semua percobaan, membuktikan bahwa tanpa waktu yang memadai, SA tidak dapat mengaktifkan mekanisme eksplorasinya dan langsung terkunci pada minimum lokal yang dihasilkan oleh solusi awal *first fit*.

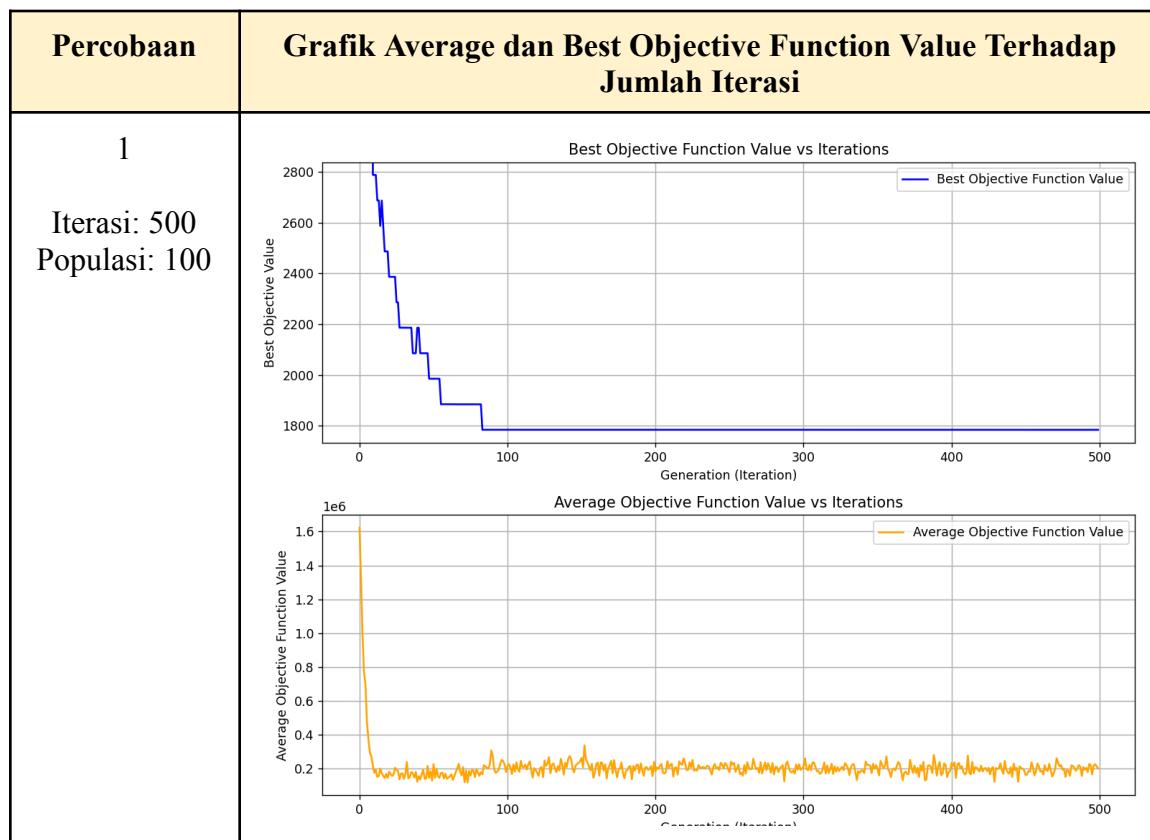
Sebaliknya, peningkatan drastis anggaran menjadi MAX_ITER = 500.000 memberikan hasil yang signifikan dan probabilistik. Dalam Run #1, algoritma berhasil mengurangi jumlah kontainer dari 19 menjadi 18, yang menghasilkan penurunan skor *objective function* sebesar 200 poin. Keberhasilan ini didukung oleh penerimaan lebih dari 86.000 langkah buruk, yang memungkinkan SA untuk melompati energi *barrier* dan melakukan reorganisasi barang yang kompleks. Meskipun demikian, hasil Run #3 menunjukkan bahwa keberhasilan ini bersifat probabilistik. Solusi awal yang terlampau kokoh tetap membuat algoritma gagal mencari perbaikan, meskipun dengan jumlah iterasi maksimum yang tinggi.

Secara keseluruhan, perbandingan ini mengonfirmasi bahwa jumlah iterasi maksimum yang tinggi sangat krusial dalam penerapan SA pada masalah Bin Packing, memungkinkan algoritma untuk memecahkan minimum lokal dan mencapai solusi yang mendekati optimal, meskipun sifat probabilistiknya tetap menghasilkan variasi antar percobaan.

Namun, hasil eksperimen menunjukkan konsistensi yang rendah, mencerminkan sifat inheren SA yang probabilistik. Sementara Run #1 berhasil mencapai 18 kontainer, Run #3 yang dimulai dari 19 kontainer gagal mencapai perbaikan dan skornya stagnan di 2106.0000. Inkonsistensi ini disebabkan oleh faktor-faktor acak seperti perbedaan susunan item pada solusi awal *first fit* di tiap *run*, serta jalur pencarian acak yang diambil oleh algoritma. Run #3 menyimpulkan bahwa susunan item awalnya menciptakan energi *barrier* yang terlalu tinggi, sehingga urutan langkah acak yang tepat untuk mencapai 18 kontainer tidak ditemukan dalam 500.000 iterasi yang ditentukan. Kegagalan ini menekankan bahwa keberhasilan SA pada waktu yang terbatas hanya bersifat potensial dan probabilistik, bukan deterministik.

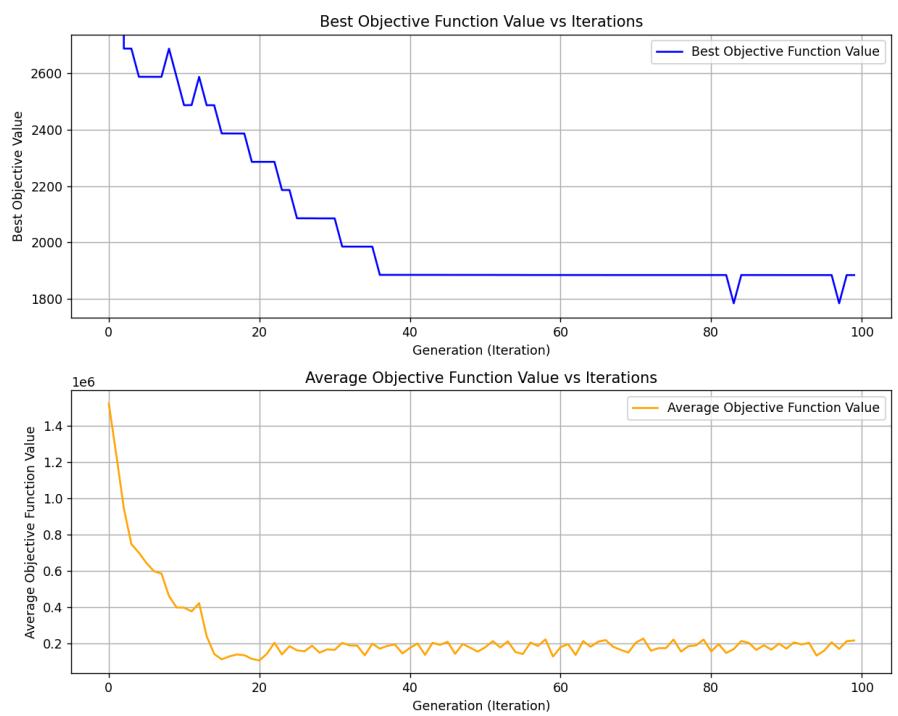
f. Genetic Algorithm

Diterapkan enam percobaan yang masing-masing memiliki konfigurasi jumlah iterasi dan jumlah populasi maksimum yang berbeda untuk algoritma Genetic Algorithm.



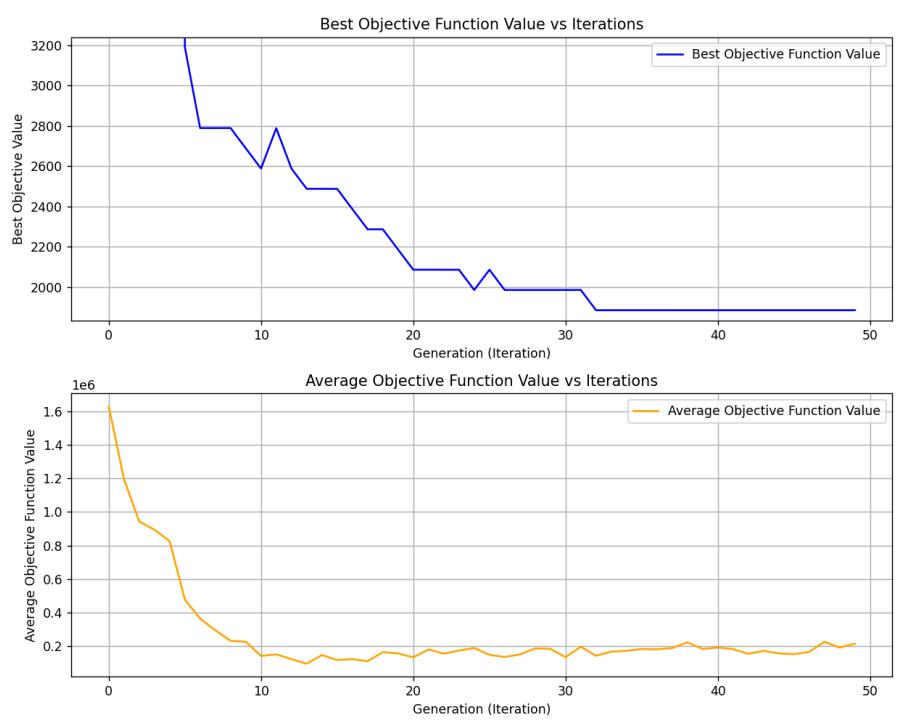
2

Iterasi: 100
Populasi: 100

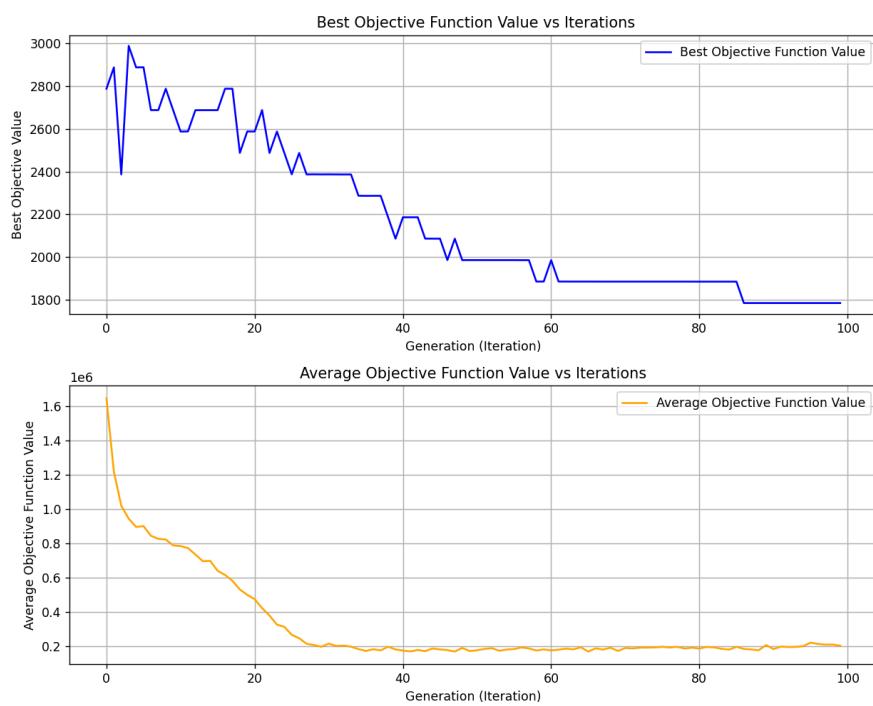


3

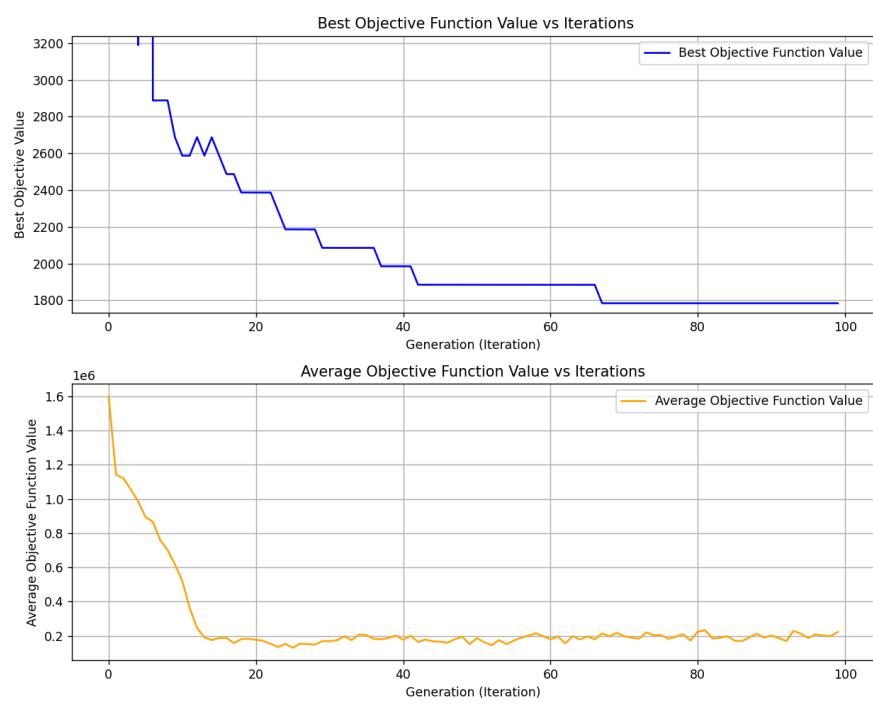
Iterasi: 50
Populasi: 100



4
Iterasi: 100
Populasi: 1000

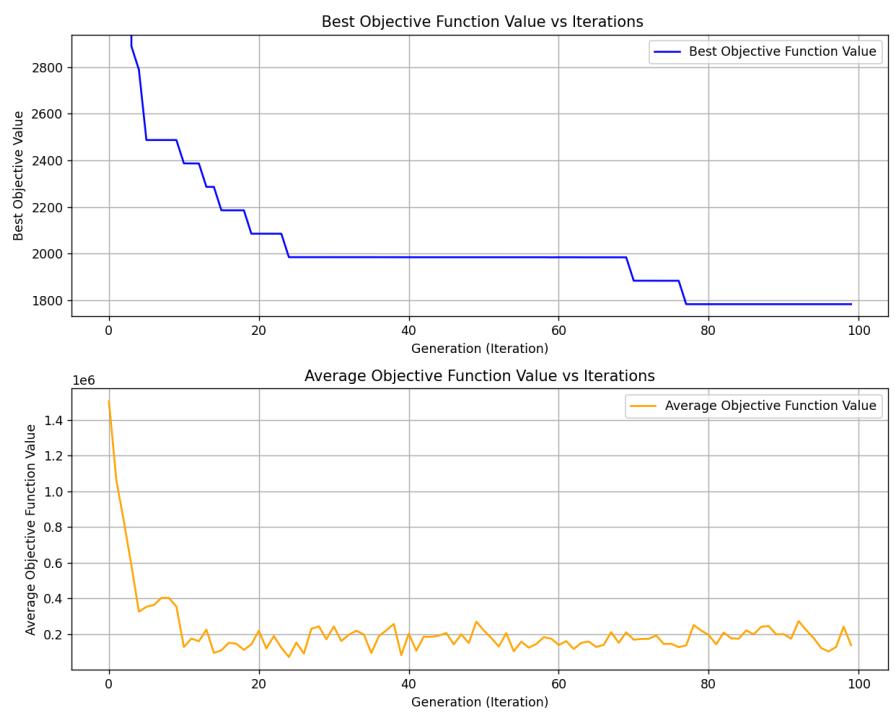


5
Iterasi: 100
Populasi: 300



6

Iterasi: 100
Populasi: 50



Berikut merupakan *state* awal dan *state* akhir dari masing-masing percobaan.

Percobaan	Initial State	Final State
-----------	---------------	-------------

1	<p>Generasi Awal</p> <p>Total Kontainer: 28</p> <ol style="list-style-type: none"> 1. Kontainer (Total: 45/100) <ul style="list-style-type: none"> - BRG016 (45) 2. Kontainer (Total: 91/100) <ul style="list-style-type: none"> - BRG002 (37) - BRG034 (54) 3. Kontainer (Total: 88/100) <ul style="list-style-type: none"> - BRG015 (18) - BRG029 (28) - BRG037 (42) 4. Kontainer (Total: 88/100) <ul style="list-style-type: none"> - BRG003 (88) 5. Kontainer (Total: 88/100) <ul style="list-style-type: none"> - BRG022 (42) - BRG025 (38) 6. Kontainer (Total: 78/100) <ul style="list-style-type: none"> - BRG005 (42) - BRG007 (36) 7. Kontainer (Total: 15/100) <ul style="list-style-type: none"> - BRG017 (15) 8. Kontainer (Total: 51/100) <ul style="list-style-type: none"> - BRG012 (17) - BRG018 (15) - BRG024 (19) 9. Kontainer (Total: 67/100) <ul style="list-style-type: none"> - BRG027 (67) 10. Kontainer (Total: 47/100) <ul style="list-style-type: none"> - BRG032 (47) 11. Kontainer (Total: 54/100) <ul style="list-style-type: none"> - BRG006 (54) 12. Kontainer (Total: 59/100) <ul style="list-style-type: none"> - BRG013 (59) 13. Kontainer (Total: 54/100) <ul style="list-style-type: none"> - BRG023 (54) 14. Kontainer (Total: 74/100) <ul style="list-style-type: none"> - BRG004 (74) 15. Kontainer (Total: 42/100) <ul style="list-style-type: none"> - BRG009 (42) 16. Kontainer (Total: 32/100) <ul style="list-style-type: none"> - BRG021 (32) 17. Kontainer (Total: 72/100) <ul style="list-style-type: none"> - BRG010 (72) 18. Kontainer (Total: 63/100) <ul style="list-style-type: none"> - BRG020 (29) - BRG028 (34) 19. Kontainer (Total: 33/100) <ul style="list-style-type: none"> - BRG038 (33) 20. Kontainer (Total: 63/100) <ul style="list-style-type: none"> - BRG026 (63) 21. Kontainer (Total: 36/100) <ul style="list-style-type: none"> - BRG014 (36) 22. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG008 (26) - BRG019 (46) - BRG033 (28) 23. Kontainer (Total: 114/100) <ul style="list-style-type: none"> - BRG011 (50) - BRG039 (64) 24. Kontainer (Total: 50/100) <ul style="list-style-type: none"> - BRG035 (19) - BRG036 (31) 25. Kontainer (Total: 78/100) <ul style="list-style-type: none"> - BRG030 (78) 26. Kontainer (Total: 66/100) <ul style="list-style-type: none"> - BRG040 (66) 27. Kontainer (Total: 47/100) <ul style="list-style-type: none"> - BRG031 (47) 28. Kontainer (Total: 23/100) <ul style="list-style-type: none"> - BRG001 (23) 	<p>Generasi Akhir</p> <p>Total Kontainer: 18</p> <ol style="list-style-type: none"> 1. Kontainer (Total: 96/100) <ul style="list-style-type: none"> - BRG006 (54) - BRG009 (42) 2. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG017 (15) - BRG025 (38) - BRG031 (47) 3. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG014 (36) - BRG039 (64) 4. Kontainer (Total: 92/100) <ul style="list-style-type: none"> - BRG005 (42) - BRG011 (50) 5. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG015 (18) - BRG024 (19) - BRG026 (63) 6. Kontainer (Total: 98/100) <ul style="list-style-type: none"> - BRG029 (20) - BRG030 (78) 7. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG002 (37) - BRG012 (17) - BRG019 (46) 8. Kontainer (Total: 95/100) <ul style="list-style-type: none"> - BRG007 (36) - BRG013 (59) 9. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG004 (74) - BRG008 (26) 10. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG018 (15) - BRG034 (54) - BRG036 (31) 11. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG001 (23) - BRG020 (29) - BRG032 (47) 12. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG027 (67) - BRG038 (33) 13. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG016 (45) - BRG023 (54) 14. Kontainer (Total: 84/100) <ul style="list-style-type: none"> - BRG022 (42) - BRG037 (42) 15. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG010 (72) - BRG033 (28) 16. Kontainer (Total: 32/100) <ul style="list-style-type: none"> - BRG021 (32) 17. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG028 (34) - BRG040 (66) 18. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG003 (80) - BRG035 (19)
---	--	--

Generasi Awal	Generasi Akhir
Total Kontainer: 32	Total Kontainer: 18
1. Kontainer (Total: 121/100)	1. Kontainer (Total: 78/100)
- BRG001 (23)	- BRG030 (78)
- BRG029 (20)	- BRG002 (37)
- BRG030 (78)	- BRG026 (63)
2. Kontainer (Total: 94/100)	3. Kontainer (Total: 96/100)
- BRG031 (47)	- BRG034 (54)
- BRG032 (47)	- BRG037 (42)
3. Kontainer (Total: 54/100)	4. Kontainer (Total: 98/100)
- BRG034 (54)	- BRG003 (80)
4. Kontainer (Total: 67/100)	- BRG015 (18)
- BRG027 (67)	5. Kontainer (Total: 99/100)
5. Kontainer (Total: 60/100)	- BRG021 (32)
- BRG016 (45)	- BRG027 (67)
- BRG018 (15)	6. Kontainer (Total: 59/100)
6. Kontainer (Total: 36/100)	- BRG013 (59)
- BRG007 (36)	7. Kontainer (Total: 98/100)
7. Kontainer (Total: 34/100)	- BRG017 (15)
- BRG028 (34)	- BRG024 (19)
8. Kontainer (Total: 72/100)	- BRG039 (64)
- BRG010 (72)	8. Kontainer (Total: 99/100)
9. Kontainer (Total: 80/100)	- BRG022 (42)
- BRG003 (80)	- BRG025 (38)
10. Kontainer (Total: 71/100)	- BRG035 (19)
- BRG020 (29)	9. Kontainer (Total: 100/100)
- BRG022 (42)	- BRG006 (54)
11. Kontainer (Total: 59/100)	- BRG012 (17)
- BRG013 (59)	- BRG020 (29)
12. Kontainer (Total: 19/100)	10. Kontainer (Total: 94/100)
- BRG024 (19)	- BRG004 (74)
13. Kontainer (Total: 64/100)	- BRG029 (20)
- BRG039 (64)	11. Kontainer (Total: 97/100)
14. Kontainer (Total: 17/100)	- BRG011 (50)
- BRG012 (17)	- BRG031 (47)
15. Kontainer (Total: 36/100)	12. Kontainer (Total: 100/100)
- BRG014 (36)	- BRG028 (34)
16. Kontainer (Total: 63/100)	- BRG040 (66)
- BRG026 (63)	13. Kontainer (Total: 97/100)
17. Kontainer (Total: 26/100)	- BRG014 (36)
- BRG008 (26)	- BRG033 (28)
18. Kontainer (Total: 74/100)	- BRG038 (33)
- BRG004 (74)	14. Kontainer (Total: 99/100)
19. Kontainer (Total: 31/100)	- BRG016 (45)
- BRG036 (31)	- BRG023 (54)
20. Kontainer (Total: 37/100)	15. Kontainer (Total: 96/100)
- BRG002 (37)	- BRG001 (23)
21. Kontainer (Total: 54/100)	- BRG005 (42)
- BRG023 (54)	- BRG036 (31)
22. Kontainer (Total: 33/100)	16. Kontainer (Total: 88/100)
- BRG038 (33)	- BRG009 (42)
23. Kontainer (Total: 70/100)	- BRG019 (46)
- BRG009 (42)	17. Kontainer (Total: 98/100)
- BRG033 (28)	- BRG007 (36)
24. Kontainer (Total: 54/100)	- BRG018 (15)
- BRG006 (54)	- BRG032 (47)
25. Kontainer (Total: 19/100)	18. Kontainer (Total: 98/100)
- BRG035 (19)	- BRG008 (26)
26. Kontainer (Total: 66/100)	- BRG010 (72)
- BRG040 (66)	
27. Kontainer (Total: 18/100)	
- BRG015 (18)	
28. Kontainer (Total: 32/100)	
- BRG021 (32)	

- | | | |
|--|---|--|
| | <p>29. Kontainer (Total: 57/100)
- BRG005 (42)
- BRG017 (15)</p> <p>30. Kontainer (Total: 88/100)
- BRG011 (50)
- BRG025 (38)</p> <p>31. Kontainer (Total: 46/100)
- BRG019 (46)</p> <p>32. Kontainer (Total: 42/100)
- BRG037 (42)</p> | |
|--|---|--|

29. Kontainer (Total: 57/100)
- BRG005 (42)
- BRG017 (15)

30. Kontainer (Total: 88/100)
- BRG011 (50)
- BRG025 (38)

31. Kontainer (Total: 46/100)
- BRG019 (46)

32. Kontainer (Total: 42/100)
- BRG037 (42)

3	<p>Generasi Awal</p> <p>Total Kontainer: 31</p> <ol style="list-style-type: none"> 1. Kontainer (Total: 19/100) <ul style="list-style-type: none"> - BRG035 (19) 2. Kontainer (Total: 19/100) <ul style="list-style-type: none"> - BRG024 (19) 3. Kontainer (Total: 38/100) <ul style="list-style-type: none"> - BRG025 (38) 4. Kontainer (Total: 47/100) <ul style="list-style-type: none"> - BRG032 (47) 5. Kontainer (Total: 64/100) <ul style="list-style-type: none"> - BRG039 (64) 6. Kontainer (Total: 120/100) <ul style="list-style-type: none"> - BRG019 (46) - BRG029 (20) - BRG034 (54) 7. Kontainer (Total: 102/100) <ul style="list-style-type: none"> - BRG014 (36) - BRG040 (66) 8. Kontainer (Total: 29/100) <ul style="list-style-type: none"> - BRG020 (29) 9. Kontainer (Total: 62/100) <ul style="list-style-type: none"> - BRG007 (36) - BRG008 (26) 10. Kontainer (Total: 96/100) <ul style="list-style-type: none"> - BRG026 (63) - BRG038 (33) 11. Kontainer (Total: 90/100) <ul style="list-style-type: none"> - BRG005 (42) - BRG012 (17) - BRG036 (31) 12. Kontainer (Total: 54/100) <ul style="list-style-type: none"> - BRG006 (54) 13. Kontainer (Total: 42/100) <ul style="list-style-type: none"> - BRG037 (42) 14. Kontainer (Total: 30/100) <ul style="list-style-type: none"> - BRG017 (15) - BRG018 (15) 15. Kontainer (Total: 103/100) <ul style="list-style-type: none"> - BRG001 (23) - BRG003 (80) 16. Kontainer (Total: 45/100) <ul style="list-style-type: none"> - BRG016 (45) 17. Kontainer (Total: 34/100) <ul style="list-style-type: none"> - BRG028 (34) 18. Kontainer (Total: 42/100) <ul style="list-style-type: none"> - BRG009 (42) 19. Kontainer (Total: 72/100) <ul style="list-style-type: none"> - BRG010 (72) 20. Kontainer (Total: 32/100) <ul style="list-style-type: none"> - BRG021 (32) 21. Kontainer (Total: 47/100) <ul style="list-style-type: none"> - BRG031 (47) 22. Kontainer (Total: 67/100) <ul style="list-style-type: none"> - BRG027 (67) 23. Kontainer (Total: 59/100) <ul style="list-style-type: none"> - BRG013 (59) 24. Kontainer (Total: 74/100) <ul style="list-style-type: none"> - BRG004 (74) 25. Kontainer (Total: 50/100) <ul style="list-style-type: none"> - BRG011 (50) 26. Kontainer (Total: 18/100) <ul style="list-style-type: none"> - BRG015 (18) 27. Kontainer (Total: 42/100) <ul style="list-style-type: none"> - BRG022 (42) 28. Kontainer (Total: 54/100) <ul style="list-style-type: none"> - BRG023 (54) 	<p>Generasi Akhir</p> <p>Total Kontainer: 20</p> <ol style="list-style-type: none"> 1. Kontainer (Total: 80/100) <ul style="list-style-type: none"> - BRG005 (42) - BRG025 (38) 2. Kontainer (Total: 98/100) <ul style="list-style-type: none"> - BRG028 (34) - BRG039 (64) 3. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG038 (33) - BRG040 (66) 4. Kontainer (Total: 95/100) <ul style="list-style-type: none"> - BRG016 (45) - BRG035 (19) - BRG036 (31) 5. Kontainer (Total: 90/100) <ul style="list-style-type: none"> - BRG007 (36) - BRG034 (54) 6. Kontainer (Total: 63/100) <ul style="list-style-type: none"> - BRG026 (63) 7. Kontainer (Total: 90/100) <ul style="list-style-type: none"> - BRG006 (54) - BRG014 (36) 8. Kontainer (Total: 91/100) <ul style="list-style-type: none"> - BRG020 (29) - BRG029 (20) - BRG037 (42) 9. Kontainer (Total: 97/100) <ul style="list-style-type: none"> - BRG003 (80) - BRG012 (17) 10. Kontainer (Total: 23/100) <ul style="list-style-type: none"> - BRG001 (23) 11. Kontainer (Total: 74/100) <ul style="list-style-type: none"> - BRG009 (42) - BRG021 (32) 12. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG010 (72) - BRG033 (28) 13. Kontainer (Total: 84/100) <ul style="list-style-type: none"> - BRG002 (37) - BRG031 (47) 14. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG015 (18) - BRG018 (15) - BRG027 (67) 15. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG008 (26) - BRG013 (59) - BRG017 (15) 16. Kontainer (Total: 74/100) <ul style="list-style-type: none"> - BRG004 (74) 17. Kontainer (Total: 92/100) <ul style="list-style-type: none"> - BRG011 (50) - BRG022 (42) 18. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG019 (46) - BRG023 (54) 19. Kontainer (Total: 47/100) <ul style="list-style-type: none"> - BRG032 (47) 20. Kontainer (Total: 97/100) <ul style="list-style-type: none"> - BRG024 (19) - BRG030 (78)
---	--	---

	<p>29. Kontainer (Total: 28/100) - BRG033 (28)</p> <p>30. Kontainer (Total: 78/100) - BRG030 (78)</p> <p>31. Kontainer (Total: 37/100) - BRG002 (37)</p>	
--	--	--

4	<p>Generasi Awal</p> <p>Total Kontainer: 32</p> <ol style="list-style-type: none"> 1. Kontainer (Total: 42/100) <ul style="list-style-type: none"> - BRG005 (42) 2. Kontainer (Total: 42/100) <ul style="list-style-type: none"> - BRG022 (42) 3. Kontainer (Total: 74/100) <ul style="list-style-type: none"> - BRG009 (42) - BRG021 (32) 4. Kontainer (Total: 28/100) <ul style="list-style-type: none"> - BRG033 (28) 5. Kontainer (Total: 89/100) <ul style="list-style-type: none"> - BRG004 (74) - BRG017 (15) 6. Kontainer (Total: 79/100) <ul style="list-style-type: none"> - BRG002 (37) - BRG037 (42) 7. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG016 (45) - BRG023 (54) 8. Kontainer (Total: 19/100) <ul style="list-style-type: none"> - BRG035 (19) 9. Kontainer (Total: 54/100) <ul style="list-style-type: none"> - BRG006 (54) 10. Kontainer (Total: 15/100) <ul style="list-style-type: none"> - BRG018 (15) 11. Kontainer (Total: 59/100) <ul style="list-style-type: none"> - BRG013 (59) 12. Kontainer (Total: 92/100) <ul style="list-style-type: none"> - BRG010 (72) - BRG029 (20) 13. Kontainer (Total: 66/100) <ul style="list-style-type: none"> - BRG040 (66) 14. Kontainer (Total: 33/100) <ul style="list-style-type: none"> - BRG038 (33) 15. Kontainer (Total: 31/100) <ul style="list-style-type: none"> - BRG036 (31) 16. Kontainer (Total: 74/100) <ul style="list-style-type: none"> - BRG007 (36) - BRG025 (38) 17. Kontainer (Total: 17/100) <ul style="list-style-type: none"> - BRG012 (17) 18. Kontainer (Total: 47/100) <ul style="list-style-type: none"> - BRG032 (47) 19. Kontainer (Total: 29/100) <ul style="list-style-type: none"> - BRG020 (29) 20. Kontainer (Total: 23/100) <ul style="list-style-type: none"> - BRG001 (23) 21. Kontainer (Total: 80/100) <ul style="list-style-type: none"> - BRG003 (80) 22. Kontainer (Total: 19/100) <ul style="list-style-type: none"> - BRG024 (19) 23. Kontainer (Total: 63/100) <ul style="list-style-type: none"> - BRG026 (63) 24. Kontainer (Total: 34/100) <ul style="list-style-type: none"> - BRG028 (34) 25. Kontainer (Total: 96/100) <ul style="list-style-type: none"> - BRG011 (50) - BRG019 (46) 26. Kontainer (Total: 78/100) <ul style="list-style-type: none"> - BRG030 (78) 	<p>Generasi Akhir</p> <p>Total Kontainer: 18</p> <ol style="list-style-type: none"> 1. Kontainer (Total: 89/100) <ul style="list-style-type: none"> - BRG009 (42) - BRG032 (47) 2. Kontainer (Total: 98/100) <ul style="list-style-type: none"> - BRG006 (54) - BRG018 (15) - BRG020 (29) 3. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG007 (36) - BRG039 (64) 4. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG005 (42) - BRG017 (15) - BRG037 (42) 5. Kontainer (Total: 54/100) <ul style="list-style-type: none"> - BRG034 (54) 6. Kontainer (Total: 97/100) <ul style="list-style-type: none"> - BRG003 (80) - BRG012 (17) 7. Kontainer (Total: 96/100) <ul style="list-style-type: none"> - BRG015 (18) - BRG030 (78) 8. Kontainer (Total: 92/100) <ul style="list-style-type: none"> - BRG008 (26) - BRG031 (47) - BRG035 (19) 9. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG019 (46) - BRG023 (54) 10. Kontainer (Total: 92/100) <ul style="list-style-type: none"> - BRG025 (38) - BRG028 (34) - BRG029 (20) 11. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG038 (33) - BRG040 (66) 12. Kontainer (Total: 96/100) <ul style="list-style-type: none"> - BRG002 (37) - BRG013 (59) 13. Kontainer (Total: 100/100) <ul style="list-style-type: none"> - BRG010 (72) - BRG033 (28) 14. Kontainer (Total: 93/100) <ul style="list-style-type: none"> - BRG004 (74) - BRG024 (19) 15. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG001 (23) - BRG016 (45) - BRG036 (31) 16. Kontainer (Total: 92/100) <ul style="list-style-type: none"> - BRG011 (50) - BRG022 (42) 17. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG014 (36) - BRG026 (63) 18. Kontainer (Total: 99/100) <ul style="list-style-type: none"> - BRG021 (32) - BRG027 (67)
---	---	--

- | | | |
|--|---|--|
| | <p>27. Kontainer (Total: 93/100)
- BRG008 (26)
- BRG027 (67)</p> <p>28. Kontainer (Total: 18/100)
- BRG015 (18)</p> <p>29. Kontainer (Total: 64/100)
- BRG039 (64)</p> <p>30. Kontainer (Total: 47/100)
- BRG031 (47)</p> <p>31. Kontainer (Total: 54/100)
- BRG034 (54)</p> <p>32. Kontainer (Total: 36/100)
- BRG014 (36)</p> | |
|--|---|--|

Generasi Awal	Generasi Akhir
Total Kontainer: 27	Total Kontainer: 18
1. Kontainer (Total: 54/100) - BRG034 (54)	1. Kontainer (Total: 99/100) - BRG005 (42)
2. Kontainer (Total: 54/100) - BRG006 (54)	- BRG024 (19)
3. Kontainer (Total: 99/100) - BRG021 (32)	- BRG025 (38)
- BRG027 (67)	2. Kontainer (Total: 100/100) - BRG011 (50)
4. Kontainer (Total: 23/100) - BRG001 (23)	- BRG012 (17)
5. Kontainer (Total: 83/100) - BRG011 (50)	- BRG038 (33)
- BRG038 (33)	3. Kontainer (Total: 99/100) - BRG007 (36)
6. Kontainer (Total: 99/100) - BRG008 (26)	- BRG026 (63)
- BRG016 (45)	4. Kontainer (Total: 96/100) - BRG002 (37)
- BRG033 (28)	- BRG013 (59)
7. Kontainer (Total: 66/100) - BRG040 (66)	5. Kontainer (Total: 54/100) - BRG006 (54)
8. Kontainer (Total: 36/100) - BRG014 (36)	6. Kontainer (Total: 99/100) - BRG016 (45)
9. Kontainer (Total: 42/100) - BRG037 (42)	- BRG023 (54)
10. Kontainer (Total: 72/100) - BRG010 (72)	7. Kontainer (Total: 89/100) - BRG009 (42)
11. Kontainer (Total: 66/100) - BRG002 (37)	- BRG032 (47)
- BRG020 (29)	8. Kontainer (Total: 100/100) - BRG014 (36)
12. Kontainer (Total: 95/100) - BRG007 (36)	- BRG017 (15)
- BRG013 (59)	- BRG018 (15)
13. Kontainer (Total: 62/100) - BRG022 (42)	- BRG028 (34)
- BRG029 (28)	9. Kontainer (Total: 96/100) - BRG020 (29)
14. Kontainer (Total: 57/100) - BRG024 (19)	- BRG027 (67)
- BRG025 (38)	10. Kontainer (Total: 98/100) - BRG029 (20)
15. Kontainer (Total: 42/100) - BRG009 (42)	- BRG030 (78)
16. Kontainer (Total: 63/100) - BRG026 (63)	11. Kontainer (Total: 89/100) - BRG031 (47)
17. Kontainer (Total: 46/100) - BRG019 (46)	- BRG037 (42)
18. Kontainer (Total: 62/100) - BRG017 (15)	12. Kontainer (Total: 98/100) - BRG003 (80)
- BRG032 (47)	- BRG015 (18)
19. Kontainer (Total: 34/100) - BRG028 (34)	13. Kontainer (Total: 100/100) - BRG010 (72)
20. Kontainer (Total: 15/100) - BRG018 (15)	- BRG033 (28)
21. Kontainer (Total: 88/100) - BRG003 (88)	14. Kontainer (Total: 100/100) - BRG019 (46)
22. Kontainer (Total: 96/100) - BRG015 (18)	- BRG034 (54)
- BRG031 (47)	15. Kontainer (Total: 93/100) - BRG004 (74)
- BRG036 (31)	- BRG035 (19)
23. Kontainer (Total: 36/100) - BRG012 (17)	16. Kontainer (Total: 96/100) - BRG021 (32)
- BRG035 (19)	- BRG039 (64)
24. Kontainer (Total: 78/100) - BRG030 (78)	17. Kontainer (Total: 89/100) - BRG001 (23)
25. Kontainer (Total: 54/100) - BRG023 (54)	- BRG040 (66)
26. Kontainer (Total: 64/100) - BRG039 (64)	18. Kontainer (Total: 99/100) - BRG008 (26)
27. Kontainer (Total: 116/100) - BRG004 (74)	- BRG022 (42)
- BRG005 (42)	- BRG036 (31)

Generasi Awal

Total Kontainer: 29

1. Kontainer (Total: 19/100)
 - BRG035 (19)
2. Kontainer (Total: 63/100)
 - BRG026 (63)
3. Kontainer (Total: 86/100)
 - BRG008 (26)
 - BRG016 (45)
 - BRG018 (15)
4. Kontainer (Total: 38/100)
 - BRG025 (38)
5. Kontainer (Total: 74/100)
 - BRG004 (74)
6. Kontainer (Total: 54/100)
 - BRG006 (54)
7. Kontainer (Total: 32/100)
 - BRG021 (32)
8. Kontainer (Total: 82/100)
 - BRG014 (36)
 - BRG019 (46)
9. Kontainer (Total: 65/100)
 - BRG001 (23)
 - BRG005 (42)
10. Kontainer (Total: 79/100)
 - BRG017 (15)
 - BRG039 (64)
11. Kontainer (Total: 42/100)
 - BRG009 (42)
12. Kontainer (Total: 66/100)
 - BRG040 (66)
13. Kontainer (Total: 42/100)
 - BRG022 (42)
14. Kontainer (Total: 72/100)
 - BRG010 (72)
15. Kontainer (Total: 59/100)
 - BRG013 (59)
16. Kontainer (Total: 42/100)
 - BRG037 (42)
17. Kontainer (Total: 34/100)
 - BRG028 (34)
18. Kontainer (Total: 117/100)
 - BRG002 (37)
 - BRG003 (80)
19. Kontainer (Total: 31/100)
 - BRG036 (31)
20. Kontainer (Total: 87/100)
 - BRG034 (54)
 - BRG038 (33)
21. Kontainer (Total: 28/100)
 - BRG033 (28)
22. Kontainer (Total: 19/100)
 - BRG024 (19)
23. Kontainer (Total: 50/100)
 - BRG011 (50)
24. Kontainer (Total: 95/100)
 - BRG012 (17)
 - BRG030 (78)
25. Kontainer (Total: 83/100)
 - BRG020 (29)
 - BRG023 (54)
26. Kontainer (Total: 47/100)
 - BRG032 (47)
27. Kontainer (Total: 20/100)
 - BRG029 (20)
28. Kontainer (Total: 18/100)
 - BRG015 (18)
29. Kontainer (Total: 150/100)
 - BRG007 (36)
 - BRG027 (67)
 - BRG031 (47)

Generasi Akhir

Total Kontainer: 18

1. Kontainer (Total: 97/100)
 - BRG036 (31)
 - BRG040 (66)
2. Kontainer (Total: 92/100)
 - BRG010 (72)
 - BRG029 (20)
3. Kontainer (Total: 99/100)
 - BRG016 (45)
 - BRG023 (54)
4. Kontainer (Total: 99/100)
 - BRG009 (42)
 - BRG017 (15)
 - BRG022 (42)
5. Kontainer (Total: 95/100)
 - BRG007 (36)
 - BRG013 (59)
6. Kontainer (Total: 95/100)
 - BRG003 (80)
 - BRG018 (15)
7. Kontainer (Total: 99/100)
 - BRG024 (19)
 - BRG025 (38)
 - BRG037 (42)
8. Kontainer (Total: 99/100)
 - BRG012 (17)
 - BRG026 (63)
 - BRG035 (19)
9. Kontainer (Total: 90/100)
 - BRG014 (36)
 - BRG034 (54)
10. Kontainer (Total: 100/100)
 - BRG004 (74)
 - BRG008 (26)
11. Kontainer (Total: 96/100)
 - BRG011 (50)
 - BRG019 (46)
12. Kontainer (Total: 93/100)
 - BRG020 (29)
 - BRG039 (64)
13. Kontainer (Total: 95/100)
 - BRG027 (67)
 - BRG033 (28)
14. Kontainer (Total: 78/100)
 - BRG030 (78)
15. Kontainer (Total: 94/100)
 - BRG031 (47)
 - BRG032 (47)
16. Kontainer (Total: 89/100)
 - BRG001 (23)
 - BRG021 (32)
 - BRG028 (34)
17. Kontainer (Total: 87/100)
 - BRG006 (54)
 - BRG038 (33)
18. Kontainer (Total: 97/100)
 - BRG002 (37)
 - BRG005 (42)
 - BRG015 (18)

Percobaan 1-6 menggunakan probabilitas mutasi 0.02.

Parameter	Percobaan 1	Percobaan 2	Percobaan 3	Percobaan 4	Percobaan 5	Percobaan 6
Durasi (s)	5.59	1.17	0.61	12.02	4.19	0.71
Banyak Populasi	100	100	100	1000	300	50
Jumlah Iterasi	500	100	50	100	100	100
Initial State	142789.61	213190.63	382991.54	122990.89	162689.1	672890.97
Final State	1783.62	1783.88	1884.59	1783.9	1783.86	1784.01

Probabilitas	Banyak Populasi	Jumlah Iterasi	Objective Function	State
0.05			2186.64	Initial State: 30 Final State: 22
1	50	100	822690.75	Initial State: 27 Final State: 25

Algoritma GA diimplementasikan secara konsisten untuk menemukan *local optimum* yang berkualitas tinggi. Berdasarkan total ukuran barang (1694) dan kapasitas kontainer (100), batas minimum teoritis (*global optimum*) adalah 17 kontainer. Dari 6 percobaan yang dilakukan, 5 di antaranya (percobaan 1, 2, 4, 5, 6) berhasil mencapai *final state* dengan skor *fitness* yang sangat mirip, berkisar antara 1783 dan 1784. Skor ini merepresentasikan solusi 18 kontainer dengan tingkat kepadatan yang sedikit berbeda-beda (dihitung dari rumus $(18 * 100) - reward$ kepadatan). Konsistensi ini menunjukkan bahwa algoritma berhasil konvergen pada solusi 18 kontainer dan tidak terjebak di solusi yang lebih buruk (seperti 20 atau 21). Tetapi, percobaan 3 (populasi 100, iterasi 50) yang menghasilkan *final state* 1884.59 merepresentasikan solusi 19 kontainer. Hal ini membuktikan bahwa parameter yang tidak memadai, khususnya jumlah iterasi yang terlalu sedikit (hanya 50), menyebabkan GA berhenti secara prematur sebelum sempat menemukan solusi 18 kontainer.

Jumlah iterasi & populasi juga berpengaruh. Hal ini ditunjukkan pada percobaan 3 (iterasi 50) dimana jumlah iterasi yang terlalu sedikit (0.61 detik) adalah jaminan kegagalan untuk menemukan solusi terbaik. Di sisi lain, percobaan 2 (100 iterasi) sudah cukup untuk mencapai 18 kontainer. Sedangkan, jumlah iterasi 500 (percobaan 1)

memberikan hasil *fitness* yang sedikit lebih baik (1783.62 dan 1783.86), yang berarti GA berhasil menemukan susunan 18 kontainer yang lebih padat. Namun, ada titik jenuh dimana percobaan 4 (populasi 1000) tidak menghasilkan solusi yang lebih baik dari percobaan 5, dan membutuhkan durasi 3x lebih lama (12.02 detik). Hal ini menunjukkan bahwa banyak populasi yang terlalu besar tidak efisien.

Begitupula dengan probabilitas mutasi yang memengaruhi hasil. Solusi 18 kontainer dicapai menggunakan probabilitas mutasi 0.02 dimana ini adalah *sweet spot* (titik ideal) yang menyeimbangkan dua tujuan. Ketika probabilitas ini dinaikkan sedikit saja menjadi 0.05, hasilnya langsung memburuk drastis menjadi 22 kontainer. Hal ini terjadi karena mutasi yang terlalu sering merusak kromosom anak sebelum gen-gen baik sempat stabil, sehingga proses belajar GA terganggu. Puncaknya, pada mutasi = 1 (100%), hasilnya sangat buruk dimana ini adalah skor *invalid* yang terkena penalti *overflow* besar, membuktikan bahwa mutasi 100% menghapus semua warisan genetik dan mengubah GA menjadi *random search* (pencarian acak) yang tidak efektif.

g. Analisis

Dari hasil implementasi berbagai algoritma, berikut adalah nilai *state* terbaik yang dicapai oleh masing-masing algoritma dalam eksperimen yang dilakukan.

Algoritma	State Value Terbaik	Durasi (s)
Hill-Climbing Steepest Ascent	1783.5880	0.8501
Hill-Climbing with Sideways Move	1783.5880	2.4552
Random Restart Hill-Climbing	1783.3832	5.1122
Stochastic Hill-Climbing	1783.5832	1.1650
Simulated Annealing	1906	16.2437
Genetic Algorithm	1783.6200	5.59

Dari hasil implementasi keenam algoritma, terlihat adanya *trade-off* yang jelas antara kecepatan eksekusi, konsistensi, dan kualitas solusi. Tidak ada satupun algoritma yang berhasil mencapai *global optimum* (17 kontainer). Namun, Random Restart Hill-Climbing (RRHC) menjadi algoritma yang paling unggul dalam hal kualitas solusi, berhasil mencapai skor *fitness* terendah (paling baik) yaitu 1783.3832. Genetic Algorithm (GA) dan Stochastic HC menyusul tipis di belakangnya dengan skor yang juga sangat

baik. Kelima algoritma ini (termasuk varian HC lainnya) secara konsisten berhasil menemukan *local optimum* 18 kontainer. Pengecualian utama adalah Simulated Annealing (SA), yang terbukti paling tidak konsisten, meskipun satu *run* berhasil mencapai 18 kontainer, *run* lainnya gagal dan terjebak di 19 kontainer (skor 2106).

Perbedaan kualitas ini berbanding terbalik dengan durasi. Algoritma tercepat yaitu Steepest Ascent Hill-Climbing (SAHC), hanya membutuhkan 0.85 detik karena sifatnya yang *greedy* dan langsung berhenti di *local optimum* pertama. Sebaliknya, algoritma yang paling eksploratif adalah yang paling lambat. Simulated Annealing (dengan 500.000 iterasi) adalah yang paling lambat (13-21 detik), diikuti oleh Random Restart HC (menjalankan 10 *run* HC, 5-6 detik) dan Genetic Algorithm (yang durasinya sangat bervariasi tergantung parameter, 0.61s hingga 12.02s).

Dalam hal konsistensi, SAHC dan HC *with sideways move* adalah yang paling konsisten (deterministik), di mana keduanya selalu menghasilkan skor 1783.5880 jika dimulai dari *initial state* yang sama. GA juga sangat konsisten, di mana 5 dari 6 *run* konvergen di 18 kontainer. HC *with sideways move* terbukti paling tidak efisien karena meskipun konsisten, algoritma ini 3x lebih lambat dari SAHC tanpa memberikan peningkatan kualitas solusi sama sekali.

BAB 4

KESIMPULAN DAN SARAN

4.1. Kesimpulan

Pada tugas ini, kelompok kami telah melakukan percobaan menggunakan berbagai algoritma *local search* untuk menyelesaikan masalah *Bin Packing Problem*. Algoritma yang diimplementasikan meliputi empat variasi *hill-climbing* (*Steepest Ascent*, *Sideways Move*, *Stochastic*, dan *Random Restart*), *Simulated Annealing*, dan *Genetic Algorithm*. Berdasarkan hasil eksperimen, algoritma yang paling mendekati *global optimum* (17 kontainer) adalah Random Restart Hill-Climbing (RRHC), yang berhasil mencapai skor *fitness* terendah yaitu 1783.3832 (merepresentasikan 18 kontainer yang sangat padat). Mekanisme *restart* pada RRHC terbukti efektif untuk menghindari dari *local optimum* yang dangkal dan menemukan solusi yang lebih baik.

Percobaan ini juga mengevaluasi konsistensi dari setiap algoritma. Algoritma deterministik seperti *Steepest Ascent HC* dan *HC with Sideways Move* terbukti sangat konsisten, selalu menghasilkan skor akhir identik 1783.5880 dari *initial state* yang sama. Sebaliknya, algoritma yang memiliki elemen acak (*randomness*), seperti GA, SA, Stochastic HC, dan RRHC menunjukkan hasil yang probabilistik dan bervariasi. Hal ini menunjukkan adanya *trade-off* antara konsistensi dan kemampuan eksplorasi.

Dengan demikian, solusi pencarian untuk masalah *Bin Packing Problem* telah berhasil ditemukan dan dianalisis menggunakan algoritma *Steepest Ascent Hill-Climbing*, *Hill-Climbing with Sideways Move*, *Random Restart Hill-Climbing*, *Stochastic Hill-Climbing*, *Simulated Annealing*, dan *Genetic Algorithm*.

4.2. Saran

- Melakukan *tuning* parameter lebih lanjut (seperti COOLING_RATE pada SA atau rasio_mutasi pada GA) untuk mengoptimalkan keseimbangan antara eksplorasi (mencari solusi baru) dan eksploitasi (merapikan solusi saat ini).
- Menerapkan konsep *concurrency* atau *multiprocessing*, terutama pada algoritma Random Restart HC (untuk menjalankan 10 *run* secara paralel) dan Genetic Algorithm (mengevaluasi *fitness* populasi secara paralel), untuk mengurangi durasi eksekusi total.
- Mengimplementasikan operator *move* atau *crossover* yang lebih kompleks yang dirancang khusus untuk masalah *packing*, yang mungkin memiliki peluang lebih besar untuk “melompat” dari *local optimum* 18 kontainer ke *global optimum* 17 kontainer.
- Mendalami pengetahuan terkait *library* visualisasi data seperti matplotlib agar dapat menyajikan beberapa grafik dalam satu kanvas untuk perbandingan yang lebih mudah.

PEMBAGIAN TUGAS

Nama	NIM	Pembagian Tugas
Vincentia Belinda Sumartoyo	18223078	Implementasi: - Genetic Algorithm Laporan: - Deskripsi persoalan - Pembahasan
Indiana Aulia Ayundazulfa	18223100	Implementasi: - Hill Climbing Laporan: - Pembahasan
Nurul Na'im Natifah	18223106	Implementasi: - Simulated Annealing Laporan: - Pembahasan

REFERENSI

1. Wikipedia. *Bin packing problem*. Wikipedia. Diakses dari: https://en.wikipedia.org/wiki/Bin_packing_problem
2. Kothari, S. (2023). *Local Search Algorithms in AI: A Comprehensive Guide*. Simplilearn. Diakses dari: <https://www.simplilearn.com/local-search-algorithms-in-ai-article>
3. Quora. (2023). *Why do algorithms become entrapped in local optima?* Diakses dari: <https://www.quora.com/Why-do-algorithms-become-entrapped-in-local-optima>
4. Quora. (2020). *What is a complete search algorithm?* Diakses dari: <http://www.quora.com/What-is-a-complete-search-algorithm>
5. Slide Kuliah IF3070 - Dasar Inteligensi Artifisial 2025. Program Studi Teknik Informatika, Institut Teknologi Bandung.
6. GeeksforGeeks. (2024). *Genetic Algorithms – Introduction to Evolutionary Computation*. Diakses dari: <https://www.geeksforgeeks.org/genetic-algorithms/>
7. Erdiwansyah, E. (2016). *Analisis Perbandingan Metode Local Search dan Population Based dalam Algoritma Berevolusi untuk Penyelesaian Travelling Salesman Problem (TSP)*. Jurnal Serambi Engineering. Diakses dari: <https://ojs.serambimekkah.ac.id/jse/article/view/307/287>
8. Scaler. *Types of Local Search Algorithm*. Scaler Topics. Diakses dari: <https://www.scaler.com/topics/artificial-intelligence-tutorial/types-of-local-search-algorithm>
9. Baeldung. *Simulated Annealing*. Baeldung | Computer Science. Diakses dari: <https://www.baeldung.com/cs/simulated-annealing>
10. Wikipedia. *Genetic Algorithm*. Wikipedia. Diakses dari: https://en.wikipedia.org/wiki/Genetic_algorithm
11. 3dBinPacking.com. *Bin Packing Optimization Strategies*. Blog 3dBinPacking. Diakses dari: <https://www.3dbinpaking.com/en/blog/bin-packing-optimization-strategies>
12. Rao, V. (2023). *Bin Packing Problem: A Brief Overview and a Quantum Solution*. Medium. Diakses dari: <https://venkyrao.medium.com/bin-packing-problem-a-brief-overview-and-a-quantum-solution-dd93600a876b>