Tugas Besar 1 IF3070 Dasar Inteligensi Artifisial

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

Disusun oleh:

Samuel Chris Michael Bagasta S.	18223011
Stanislaus Ardy Bramantyo	18223057
Audy Alicia Renatha Tirayoh	18223097

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
DESKRIPSI PERSOALAN

Bin Packing Problem adalah permasalahan optimasi dimana sekumpulan barang dengan ukuran yang berbeda-beda harus ditempatkan ke dalam sejumlah kontainer yang sudah ditentukan. Setiap kontainer juga memiliki kapasitas yang seragam dan terbatas. Kondisi yang paling optimal adalah menggunakan jumlah kontainer sesedikit mungkin.

Setiap barang memiliki ID unik dan ukuran tertentu, sedangkan setiap kontainer memiliki kapasitas maksimum yang sudah ditentukan. Dalam konteks ini, sebuah solusi merepresentasikan alokasi setiap barang yang ada ke dalam kontainer yang tersedia. Pencarian solusi optimal sulit dilakukan dalam waktu singkat jika jumlah barang tergolong banyak. Oleh karena itu, tugas besar ini berfokus pada penerapan *Local Search Algorithm* untuk mencari solusi yang baik secara efisien.

Kondisi awal yang harus dipenuhi adalah penempatan setiap barang ke dalam kontainer, boleh secara acak atau menggunakan algoritma heuristik sederhana, seperti *Best Fit Algorithm*. Selanjutnya, *move* yang boleh dilakukan setiap iterasi, yaitu:

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

Untuk memenuhi kondisi yang paling optimal, dibutuhkan fungsi objektif untuk meminimalkan jumlah kontainer dengan memastikan tidak ada kontainer yang melebihi kapasitas. Untuk mencapai hal tersebut, fungsi objektif harus mencakup:

- Penalti Kapasitas Berlebih
  Setiap kontainer yang total ukuran barangnya melebihi kapasitas harus diberi penalti yang sangat besar. Ini untuk memastikan solusi akhir adalah solusi yang valid.
- Skor Berdasarkan Jumlah dan Kepadatan Kontainer
  Fungsi ini bertujuan untuk memprioritaskan *state* yang menggunakan lebih sedikit kontainer dan lebih padat.
- Seberapa berpengaruh setiap penalti atau *reward* terhadap *objective function* keseluruhan akan dibebaskan. Setiap konsiderasi besar pengaruh harus disertai alasan.

BAB II
PEMBAHASAN

2.1 Pemilihan *Objective Function*

Fungsi objektif merupakan komponen utama dalam pendekatan *local search,* karena menentukan bagaimana suatu solusi dievaluasi dan seberapa baik kualitasnya dibandingkan solusi-solusi lain. Dalam konteks tugas besar ini, yaitu *Bin Packing Problem*, fungsi objektifnya digunakan untuk menilai efisiensi peletakkan sekumpulan barang ke dalam sejumlah kontainer dengan kapasitas yang sama.

Tujuan utama dari *Bin Packing Problem* ini adalah untuk meminimalkan jumlah kontainer yang digunakan, namun tetap memastikan bahwa tidak ada kontainer yang melebihi kapasitas maksimum. Selain itu, fungsi objektif juga mendorong pencarian solusi yang padat dengan kontainer yang terisi mendekati kapasitasnya tanpa *overflow*.

Maka dari itu, fungsi objektif yang digunakan didefinisikan atas kombinasi dari tiga komponen utama, yaitu penalti jumlah kontainer, penalti kelebihan kapasitas, dan *reward* kepadatan kontainer. Rumusnya sebagai berikut:

$$f(S) = (\alpha \times K) + (\beta \times P_{overflow}) - (\gamma \times D_{density})$$

Keterangan:

K	: Jumlah kontainer yang digunakan dalam solusi S,
$P_{overflow}$	: **Penalti** total akibat *overflow*,
$D_{density}$	: ***Reward*** berdasarkan tingkat **kepadatan kontainer**,
α, β, γ	: **Bobot** pengatur kontribusi masing-masing **komponen**.

Penalti *overflow* diberikan agar algoritma menghindari solusi yang tidak valid. Nilai penalti dihitung dengan menjumlahkan kelebihan isi setiap kontainer:

$$P_{overflow} = \sum_{i=1}^{K} max(0,\ isi(C_i) - kapasitas)$$

Sementara itu, komponen kepadatan ($D_{density}$) memberikan insentif pada solusi yang memanfaatkan kapasitas kontainer dengan efisien. Nilainya dihitung berdasarkan rasio isi terhadap kapasitas maksimum:

$$D_{density} = \sum_{i=1}^{K} \left(\frac{isi(C_i)}{kapasitas}\right)^2$$

Semakin tinggi tingkat pengisian suatu kontainer, semakin besar reward-nya. Pendekatan ini mengikuti ide yang dikemukakan oleh Falkenauer (1996), yang menggunakan konsep *fill ratio* untuk menilai efisiensi pengepakan dalam algoritma genetika untuk *Bin Packing Problem*.

Bobot-bobot dalam implementasi ini ditentukan secara empiris, yaitu α=1.0, β=1000.0, dan γ=0.1. Nilai β yang besar memastikan bahwa algoritma memprioritaskan solusi yang tidak melanggar kapasitas, sedangkan γ memberikan pengaruh kecil sebagai dorongan tambahan agar hasil pengepakan lebih padat.

Pendekatan perancangan fungsi objektif seperti ini umum digunakan dalam penelitian-penelitian sebelumnya, misalnya oleh Lodi et al. (2002) dan Delorme et al. (2016), yang juga menambahkan penalti terhadap pelanggaran kapasitas dan reward untuk efisiensi ruang dalam model optimasi *Bin Packing*.

Dengan kombinasi ketiga komponen tersebut, fungsi objektif ini diharapkan mampu menyeimbangkan antara validitas solusi (tidak terjadi *overflow*) dan efisiensi penggunaan ruang (tingkat kepadatan tinggi serta jumlah kontainer yang sedikit). Fungsi ini kemudian digunakan secara konsisten pada semua algoritma yang diimplementasikan dalam tugas besar ini, yaitu *Hill-Climbing, Simulated Annealing*, dan *Genetic Algorithm*.

2.2 Penjelasan Implementasi Algoritma *Local Search*

2.2.1 *Steepest Ascent Hill-Climbing*

a. Deskripsi

Algoritma *Steepest Ascent Hill-Climbing* menempatkan setiap barang ke setiap kontainer secara acak, dengan memperhatikan kapasitas maksimum dari kontainer.

Setelah itu, algoritma akan membandingkan dua barang dari dua kontainer berbeda. Apabila kapasitas kontainer meningkat (efisiensi meningkat), terjadi pertukaran. Jika pada salah satu iterasi tidak terjadi pertukaran, algoritma berhenti bekerja, terjebak di *local maximum*.

b. *Source Code*

```
import random
import json
import sys
import os
import time
import copy
import matplotlib.pyplot as plt

script_dir = os.path.dirname(os.path.abspath(__file__))
if len(sys.argv) > 1:
    json_filename = sys.argv[1]
else:
    json_filename = 'case1.json'

json_path = os.path.join(script_dir, json_filename)

try:
    with open(json_path, 'r') as f:
        data = json.load(f)
    print(f"Using JSON file: {json_filename}")
except FileNotFoundError:
    print(f"Error: {json_filename} not found!")
    print("Usage: python3 HillClimbingSteepest.py [json_filename]")
    print("Example: python3 HillClimbingSteepest.py case2.json")
    sys.exit(1)

# variabel global
kapasitas_kontainer   = data['kapasitas_kontainer']  # kapasitas kontainer
barang                = data['barang'].copy()         # id barang
jumlah_barang         = len(barang)                   # jumlah barang
kontainer             = []                            # kontainer
kontainer_id          = 0                             # id kontainer

print(f"Loaded data from JSON file:")
print(f"Kapasitas kontainer: {kapasitas_kontainer} kg/m³")
```

```
print(f"Jumlah barang: {jumlah_barang}")
items_str = [f"{item['id']}({item['ukuran']})" for item in barang]
print(f"Barang: {items_str}")

kontainer.append([])
kontainer_space_left = kapasitas_kontainer

while True:
    if len(barang) == 0:
        break

    random_index   = random.randint(0, len(barang) - 1)
    random_barang  = barang[random_index]

    if random_barang['ukuran'] <= kontainer_space_left:
        kontainer[kontainer_id].append({
            'barang': random_barang
        })

        kontainer_space_left -= random_barang['ukuran']
        barang.pop(random_index)

    else:
        kontainer_id += 1
        kontainer.append([])
        kontainer_space_left = kapasitas_kontainer

print("\n" + "="*60)
print("SPAWN BARANG DALAM KONTAINER (STATE AWAL)")
print("="*60)

for idx, container in enumerate(kontainer):
    print(f"\nKontainer {idx + 1}:")
    total_ukuran = 0

    for item in container:
        barang_i_tempnfo = item['barang']
                        print(f"      -   ID:   {barang_i_tempnfo['id']},   Ukuran:
{barang_i_tempnfo['ukuran']} kg/m³")
        total_ukuran += barang_i_tempnfo['ukuran']

    sisa_kapasitas = kapasitas_kontainer - total_ukuran
    print(f"  Total Terisi: {total_ukuran}/{kapasitas_kontainer} kg/m³")
```

```
    print(f"  Sisa Kapasitas: {sisa_kapasitas} kg/m³")
    print(f"  Efisiensi: {(total_ukuran/kapasitas_kontainer)*100:.2f}%")

print("\n" + "="*60)
print(f"Total Kontainer Digunakan: {len(kontainer)}")
print("="*60)

def calculate_kontainer_total(kontainer):
    total = 0
    for arr_barang in kontainer:
        total += arr_barang['barang']['ukuran']
    return total

def calculate_unused(kontainer, kapasitas):
    total_unused = 0
    for container in kontainer:
        used          = calculate_kontainer_total(container)
        unused        = kapasitas - used
        total_unused += unused
    return total_unused

def calculate_waste_squared(kontainer_list, kapasitas):
    """Objective: jumlah kuadrat sisa kapasitas per kontainer (sensitif terhadap
redistribusi)."""
    total = 0
    for c in kontainer_list:
        used = calculate_kontainer_total(c)
        unused = kapasitas - used
        total += unused * unused
    return total

initial_state = copy.deepcopy(kontainer)

# inisialisasi variabel
obj_history = [calculate_waste_squared(kontainer, kapasitas_kontainer)]
total_attempts = 0         # jumlah percobaan swap (evaluasi)
total_accepted_swaps = 0   # jumlah swap yang diterima (perbaikan)
start_time = time.time()

initial_objective = obj_history[0]

current_iteration = 0
max_iterations    = 3
```

```
best_case_unused  = calculate_unused(kontainer, kapasitas_kontainer)
improvement_found = True

while improvement_found and current_iteration < max_iterations:
    improvement_found  = False

    for i in range(len(kontainer)):
        current_iteration = 0

        for j in range (i + 1, len(kontainer)):

            for index_i in range(len(kontainer[i])):
                for index_j in range(len(kontainer[j])):
                    current_total_i = calculate_kontainer_total(kontainer[i])

                                if current_total_i < kapasitas_kontainer and
current_iteration < max_iterations:
                        current_iteration += 1
                        total_attempts += 1
                        print(f"Iteration {current_iteration}")

                                                        current_total_j   =
calculate_kontainer_total(kontainer[j])

                        barang_i_temp   = kontainer[i][index_i]['barang']
                        barang_j_temp   = kontainer[j][index_j]['barang']

                                        new_total_i        = current_total_i  -
barang_i_temp['ukuran'] + barang_j_temp['ukuran']
                                        new_total_j        = current_total_j  -
barang_j_temp['ukuran'] + barang_i_temp['ukuran']

                        if new_total_i <= kapasitas_kontainer and new_total_j <=
kapasitas_kontainer:
                             # hitung objective sebelum dan setelah swap, terima
hanya jika after < before
                                before_obj = calculate_waste_squared(kontainer,
kapasitas_kontainer)

                            # lakukan swap sementara
                            kontainer[i][index_i]['barang'] = barang_j_temp
                            kontainer[j][index_j]['barang'] = barang_i_temp
```

```
                                after_obj = calculate_waste_squared(kontainer,
kapasitas_kontainer)

                            if after_obj < before_obj:
                                # terima swap secara permanen
                                improvement_found = True
                                total_accepted_swaps += 1
                                obj_history.append(after_obj)
                                        print(f"[!!!Swap Accepted!!!]: Swapped
{barang_i_temp['id']} ↔ {barang_j_temp['id']} (Obj {before_obj} → {after_obj})")
                                current_iteration = 0
                            else:
                                    # revert swap karena objektif global tidak
membaik
                                kontainer[i][index_i]['barang'] = barang_i_temp
                                kontainer[j][index_j]['barang'] = barang_j_temp
                        else:
                            break
                    else:
                        break

#hitung waktu
end_time = time.time()
duration_seconds = end_time - start_time

final_state = copy.deepcopy(kontainer)
final_objective    =    obj_history[-1]    if    len(obj_history)    >    0    else
calculate_waste_squared(final_state, kapasitas_kontainer)
iterations_until_stop = total_attempts  # jumlah percobaan evaluasi dilakukan
swaps_accepted = total_accepted_swaps

print("\n" + "="*60)
print("HASIL PENYIMPANAN BARANG DALAM KONTAINER (STATE AKHIR)")
print("="*60)

for idx, container in enumerate(final_state):
    print(f"\nKontainer {idx + 1}:")
    total_ukuran = 0

    for item in container:
        barang_i_tempnfo = item['barang']
                    print(f"     -   ID:   {barang_i_tempnfo['id']},   Ukuran:
{barang_i_tempnfo['ukuran']} kg/m³")
```

```
        total_ukuran += barang_i_tempnfo['ukuran']

    sisa_kapasitas = kapasitas_kontainer - total_ukuran
    print(f"  Total Terisi: {total_ukuran}/{kapasitas_kontainer} kg/m³")
    print(f"  Sisa Kapasitas: {sisa_kapasitas} kg/m³")
    print(f"  Efisiensi: {(total_ukuran/kapasitas_kontainer)*100:.2f}%")

print("\n" + "="*60)
print(f"Total Kontainer Digunakan: {len(final_state)}")
print("="*60)

# summary
print(f"Durasi proses pencarian: {duration_seconds:.4f} detik")
print(f"Jumlah percobaan swap (evaluasi): {iterations_until_stop}")
print(f"Jumlah swap diterima (perbaikan): {swaps_accepted}")
print(f"Nilai objective awal (sum unused^2): {initial_objective}")
print(f"Nilai objective akhir (sum unused^2): {final_objective}")
print(f"Jumlah langkah perbaikan yang tercatat (history length - 1):
{len(obj_history)-1}")

# plot obj value terhadap langkah swap/perbaikan
plt.figure(figsize=(8,4))
steps = list(range(len(obj_history)))
plt.plot(steps, obj_history, marker='o')
plt.title("Objective (sum of squared unused capacity) per improvement step")
plt.xlabel("Improvement step (0 = awal)")
plt.ylabel("Objective = sum(unused^2)")
plt.grid(True)

plt.tight_layout()
plt.show()
plt.close()

# visual obj func
def plot_state(kontainer_state, kapas, title):
    fig = plt.figure(figsize=(10,5))
    ax = fig.add_subplot(1,1,1)

    x_positions = range(len(kontainer_state))
    bar_width = 0.6

    for idx, container in enumerate(kontainer_state):
        bottoms = 0
```

```
        for seg in container:
            size = seg['barang']['ukuran']
            ax.bar(idx, size, bottom=bottoms, width=bar_width)
            if size >= max(3, kapas * 0.05):
                        ax.text(idx, bottoms + size/2, f"{seg['barang']['id']}
({size})", ha='center', va='center', fontsize=8)
            bottoms += size

           ax.hlines(kapas, idx - bar_width/2 - 0.2, idx + bar_width/2 + 0.2,
linestyles='dashed', linewidth=0.8)

        used = calculate_kontainer_total(container)
        if used < kapas:
            ax.bar(idx, kapas - used, bottom=used, width=bar_width, alpha=0.05)

            ax.text(idx, -kapas*0.05, f"{used}/{kapas}", ha='center', va='top',
fontsize=9)

    ax.set_xticks(list(x_positions))
    ax.set_xticklabels([f"K{idx+1}" for idx in x_positions])
    ax.set_ylim(-kapas*0.15, kapas*1.05)
    ax.set_ylabel("Ukuran (kg/m³)")
    ax.set_title(title)
    ax.grid(axis='y', linestyle=':', linewidth=0.6)
    plt.tight_layout()
    plt.show()
    plt.close()

plot_state(initial_state, kapasitas_kontainer, "State Awal: Penyebaran Barang
per Kontainer (Random Spawn)")
plot_state(final_state, kapasitas_kontainer, "State Akhir: Setelah Hill-Climbing
(Final)")
```

c. Penjelasan *Source Code*

Kode di atas dapat dipecah menjadi empat (4) bagian yang memiliki tujuannya masing-masing. Pertama, pembacaan JSON *file* sebagai *input* dari *user* untuk memulai algoritma tersebut.

➢ Inisialisasi Awal

```
import random
```

```
import json
import sys
import os
import time
import copy
import matplotlib.pyplot as plt

script_dir = os.path.dirname(os.path.abspath(__file__))
if len(sys.argv) > 1:
    json_filename = sys.argv[1]
else:
    json_filename = 'case1.json'

json_path = os.path.join(script_dir, json_filename)

try:
    with open(json_path, 'r') as f:
        data = json.load(f)
    print(f"Using JSON file: {json_filename}")
except FileNotFoundError:
    print(f"Error: {json_filename} not found!")
    print("Usage: python3 HillClimbingSteepest.py [json_filename]")
    print("Example: python3 HillClimbingSteepest.py case2.json")
    sys.exit(1)

# variabel global
kapasitas_kontainer   = data['kapasitas_kontainer']
barang                = data['barang'].copy()
jumlah_barang         = len(barang)
kontainer             = []
kontainer_id          = 0
```

Bagian ini memuat *test case* dalam *file* JSON untuk diselesaikan dengan algoritma optimasi yang dibuat. Selain itu, terdapat inisialisasi struktur kontainer untuk menempatkan barang pertama kali secara acak. Variabel yang terdapat pada bagian ini, antara lain:

1. `kapasitas_kontainer`: kapasitas maksimum dalam satu kontainer
2. `barang`: barang-barang yang akan ditempatkan ke kontainer, disimpan dalam bentuk *list*
3. `kontainer`: tempat barang ditempatkan, disimpan dalam sebuah *dictionary* berbentuk {‘barang’: {___}}

4. `kontainer_id`: indeks kontainer yang ada atau statusnya aktif

➢ Penempatan Awal Barang (*Initial Solution*)

```
kontainer.append([])
kontainer_space_left = kapasitas_kontainer

while True:
    if len(barang) == 0:
        break

    random_index   = random.randint(0, len(barang) - 1)
    random_barang  = barang[random_index]

    if random_barang['ukuran'] <= kontainer_space_left:
        kontainer[kontainer_id].append({
            'barang': random_barang
        })

        kontainer_space_left -= random_barang['ukuran']
        barang.pop(random_index)

    else:
        kontainer_id += 1
        kontainer.append([])
        kontainer_space_left = kapasitas_kontainer
```

Pada bagian ini, setiap barang akan ditempatkan ke kontainer secara acak. Apabila kapasitas suatu kontainer sudah penuh, algoritma akan membuat kontainer baru. Beberapa variabel yang digunakan, antara lain:

1. `kontainer_space_left`: ruang tersisa di dalam kontainer yang ada
2. `random_index`: pemilihan barang secara acak
3. `random_barang`: pemilihan barang secara acak

➢ Evaluasi (*Helper Function)*

```
def calculate_kontainer_total(kontainer):
    total = 0
    for arr_barang in kontainer:
        total += arr_barang['barang']['ukuran']
    return total
```

```
def calculate_unused(kontainer, kapasitas):
    total_unused = 0
    for container in kontainer:
        used          = calculate_kontainer_total(container)
        unused        = kapasitas - used
        total_unused += unused
    return total_unused

def calculate_waste_squared(kontainer_list, kapasitas):
    """Objective: jumlah kuadrat sisa kapasitas per kontainer (sensitif terhadap
redistribusi)."""
    total = 0
    for c in kontainer_list:
        used = calculate_kontainer_total(c)
        unused = kapasitas - used
        total += unused * unused
    return total
```

Pada bagian ini, terdapat dua buah fungsi bantuan untuk melakukan evaluasi dalam optimasi algoritma, yaitu:

1. `calculate_kontainer_total(kontainer)`: ruang yang terpakai dalam satu kontainer
2. `calculate_unused(kontainer, kapasitas)`: ruang tersisa di seluruh kontainer, digunakan untuk menilai solusi yang dihasilkan. Semakin kecil nilainya, solusi semakin baik
3. `calculate_waste_squared(kontainer_list, kapasitas)`: untuk proses visualisasi *objective function*

➢ Loop Utama

```
current_iteration = 0
max_iterations    = 3
best_case_unused  = calculate_unused(kontainer, kapasitas_kontainer)
improvement_found = True
while improvement_found and current_iteration < max_iterations:
    improvement_found  = False

    for i in range(len(kontainer)):
        current_iteration = 0 # Menghitung iteration sesuai dengan constraint
```

```
yang berlaku

        for j in range (i + 1, len(kontainer)):
                        for index_i in range(len(kontainer[i])):
                for index_j in range(len(kontainer[j])):
                    current_total_i = calculate_kontainer_total(kontainer[i]) #
Hanya berfokus pada peningkatan value dari barang I.

                    if current_total_i < kapasitas_kontainer and
current_iteration < max_iterations:
                        current_iteration += 1
                        print(f"Iteration {current_iteration}")

                        current_total_j =
calculate_kontainer_total(kontainer[j])

                        barang_i_temp   = kontainer[i][index_i]['barang']
                        barang_j_temp   = kontainer[j][index_j]['barang']

                        new_total_i     = current_total_i -
barang_i_temp['ukuran'] + barang_j_temp['ukuran']
                        new_total_j     = current_total_j -
barang_j_temp['ukuran'] + barang_i_temp['ukuran']

                        if new_total_i <= kapasitas_kontainer and new_total_j <=
kapasitas_kontainer:

                            if (kapasitas_kontainer - new_total_i) <
(kapasitas_kontainer - current_total_i):
                                kontainer[i][index_i]['barang'] = barang_j_temp
                                kontainer[j][index_j]['barang'] = barang_i_temp
                                improvement_found = True
                                print(f"[!!!Swap!!!]: Swapped
{barang_i_temp['id']} ↔ {barang_j_temp['id']} (Waste reduced by {new_total_i -
current_total_i})")
                                current_iteration = 0

                            else: # Local Maxima break
                                break

                        else: # Overflow break
                            break
```

```
                    else: # 100% Capacity & Max Iterations break
                        break
```

Terdapat beberapa variabel yang digunakan, antara lain:

1. `current_iteration`: membatasi jumlah pengecekan per *i*, nilainya akan di-*set* menjadi 0 setiap melakukan *swap*
2. `max_iterations`: batas maksimum per *i*
3. `improvement_found`: *check flag* untuk menentukan apakah ada *swap* yang dilakukan atau tidak
4. `barang_i_temp`, `barang_j_temp`: barang yang akan ditukar
5. `new_total_i`, `new_total_j`: perhitungan total ukuran kontainer secara simulasi setelah terjadi *swap*. Digunakan untuk evaluasi dan pengecekan *overflow*

Pada bagian ini, *loop* utama akan terus berjalan selama `improvement_found` masih bernilai *true* dan `current_iteration` lebih kecil dari `max_iterations`. Algoritma akan menentukan setiap pasangan yang akan dilakukan *swap*, memastikan semua kombinasi ada. Setelah itu, simulasi perhitungan total pasangan kontainer sebelum dan sesudah terjadi *swap* dilakukan. Apabila total di dalam kontainer lebih besar dan tidak terjadi *overflow*, proses *swap* akan dilakukan.

Jika tidak terjadi *swap*, algoritma akan mencoba pasangan barang lainnya. Terminasi dilakukan apabila kondisi berikut terpenuhi:

1. `improvement_found` bernilai *false* → solusi terjebak di *local maximum*
2. `current_iteration` mencapai `max_iterations`

2.2.2 *Stochastic Hill-Climbing*

a. Deskripsi

Algoritma *Stochastic Hill-Climbing* menempatkan setiap barang secara acak ke setiap kontainer. Setelah itu, algoritma menentukan daftar semua kemungkinan pertukaran antar barang dari dua kontainer berbeda. Urutan pertukaran barang akan di acak dan diiterasi kan satu per satu. Hal ini membuat cakupan pencarian algoritma lebih luas sehingga membantu untuk keluar dari *local maximum*. Apabila sebuah iterasi meningkatkan efisiensi, pertukaran dilakukan. Apabila tidak, lanjut ke pertukaran yang berikutnya. Algoritma akan melakukan hal ini hingga tidak ada lagi peningkatan efisiensi atau batas iterasi telah tercapai.

b. *Source Code*

```
import random
import json
import sys

import os
script_dir = os.path.dirname(os.path.abspath(__file__))

if len(sys.argv) > 1:
    json_filename = sys.argv[1]
else:
    json_filename = 'case1.json'

json_path = os.path.join(script_dir, json_filename)

try:
    with open(json_path, 'r') as f:
        data = json.load(f)
    print(f"Using JSON file: {json_filename}")
except FileNotFoundError:
    print(f"Error: {json_filename} not found!")
    print("Usage: python3 HillClimbingStochastic.py [json_filename]")
    print("Example: python3 HillClimbingStochastic.py case2.json")
    sys.exit(1)

# Variable Library
kapasitas_kontainer   = data['kapasitas_kontainer']  # kapasitas kontainer
barang                = data['barang'].copy()         # ID Barang
jumlah_barang         = len(barang)                   # Jumlah Barang
kontainer             = []                            # Kontainer
kontainer_id          = 0                             # ID Kontainer

print(f"Loaded data from JSON file:")
print(f"Kapasitas kontainer: {kapasitas_kontainer} kg/m³")
print(f"Jumlah barang: {jumlah_barang}")
print(f"Barang: {[f"{item['id']}({item['ukuran']})" for item in barang]}")

kontainer.append([])
kontainer_space_left = kapasitas_kontainer

while True:
    if len(barang) == 0:
```

```
        break

    random_index   = random.randint(0, len(barang) - 1)
    random_barang  = barang[random_index]

    if random_barang['ukuran'] <= kontainer_space_left:
        kontainer[kontainer_id].append({
            'barang': random_barang
        })

        kontainer_space_left -= random_barang['ukuran']
        barang.pop(random_index)

    else:
        kontainer_id += 1
        kontainer.append([])
        kontainer_space_left = kapasitas_kontainer

print("\n" + "="*60)
print("SPAWN BARANG DALAM KONTAINER")
print("="*60)

for idx, container in enumerate(kontainer):
    print(f"\nKontainer {idx + 1}:")
    total_ukuran = 0

    for item in container:
        barang_i_tempnfo = item['barang']
                        print(f"     -   ID:   {barang_i_tempnfo['id']},   Ukuran:
{barang_i_tempnfo['ukuran']} kg/m³")
        total_ukuran += barang_i_tempnfo['ukuran']

    sisa_kapasitas = kapasitas_kontainer - total_ukuran
    print(f"  Total Terisi: {total_ukuran}/{kapasitas_kontainer} kg/m³")
    print(f"  Sisa Kapasitas: {sisa_kapasitas} kg/m³")
    print(f"  Efisiensi: {(total_ukuran/kapasitas_kontainer)*100:.2f}%")

print("\n" + "="*60)
print(f"Total Kontainer Digunakan: {len(kontainer)}")
print("="*60)

# Helper function untuk Hill Climbing Algorithm
def calculate_kontainer_total(kontainer):
```

```
    total = 0
    for arr_barang in kontainer:
        total += arr_barang['barang']['ukuran']
    return total

def calculate_unused(kontainer, kapasitas):
    total_unused = 0
    for container in kontainer:
        used          = calculate_kontainer_total(container)
        unused        = kapasitas - used
        total_unused += unused
    return total_unused

current_iteration = 0
max_iterations    = 10
improvement_found   = True

while improvement_found and current_iteration < max_iterations:
    improvement_found  = False

    possible_swaps = []
    for i in range(len(kontainer)):
        for j in range (i + 1, len(kontainer)):
            for index_i in range(len(kontainer[i])):
                for index_j in range(len(kontainer[j])):
                    possible_swaps.append((i, j, index_i, index_j))

    random.shuffle(possible_swaps) # pemilihan neighbor random
    current_iteration = 0

    for current_swap in possible_swaps:
        if current_iteration >= max_iterations:
            break

        i, j, index_i, index_j = current_swap

        current_total_i = calculate_kontainer_total(kontainer[i])

        if current_total_i < kapasitas_kontainer:
            current_iteration += 1
            print(f"Iteration {current_iteration}")

            current_total_j = calculate_kontainer_total(kontainer[j])
```

```
            barang_i_temp   = kontainer[i][index_i]['barang']
            barang_j_temp   = kontainer[j][index_j]['barang']

                new_total_i       = current_total_i - barang_i_temp['ukuran'] +
barang_j_temp['ukuran']
                new_total_j       = current_total_j - barang_j_temp['ukuran'] +
barang_i_temp['ukuran']

                    if new_total_i <= kapasitas_kontainer and new_total_j <=
kapasitas_kontainer: # Overflow countermeassure
                 if (kapasitas_kontainer - new_total_i) < (kapasitas_kontainer -
current_total_i): # Change Here untuk Sideways Modifcation, Current Algorithm is
Steepest (Neighbor [>]BETTER[>] Current)
                    kontainer[i][index_i]['barang'] = barang_j_temp
                    kontainer[j][index_j]['barang'] = barang_i_temp
                    improvement_found = True
                          print(f"[!!!Swap!!!]: Swapped {barang_i_temp['id']} ↔
{barang_j_temp['id']} (Waste reduced by {new_total_i - current_total_i})")
                    current_iteration = 0

                else: # local maxima break
                    continue

            else: # overflow break
                continue

        else: # full break
            continue

print("\n" + "="*60)
print("HASIL PENYIMPANAN BARANG DALAM KONTAINER")
print("="*60)

for idx, container in enumerate(kontainer):
    print(f"\nKontainer {idx + 1}:")
    total_ukuran = 0

    for item in container:
        barang_i_tempnfo = item['barang']
                    print(f"     -   ID:   {barang_i_tempnfo['id']},   Ukuran:
{barang_i_tempnfo['ukuran']} kg/m³")
        total_ukuran += barang_i_tempnfo['ukuran']
```

```
    sisa_kapasitas = kapasitas_kontainer - total_ukuran
    print(f"  Total Terisi: {total_ukuran}/{kapasitas_kontainer} kg/m³")
    print(f"  Sisa Kapasitas: {sisa_kapasitas} kg/m³")
    print(f"  Efisiensi: {(total_ukuran/kapasitas_kontainer)*100:.2f}%")

print("\n" + "="*60)
print(f"Total Kontainer Digunakan: {len(kontainer)}")
print("="*60)
```

c. Penjelasan *Source Code*

➢ Inisialisasi Awal

```
import random
import json
import sys

import os
script_dir = os.path.dirname(os.path.abspath(__file__))

if len(sys.argv) > 1:
    json_filename = sys.argv[1]
else:
    json_filename = 'case1.json'

json_path = os.path.join(script_dir, json_filename)

try:
    with open(json_path, 'r') as f:
        data = json.load(f)
    print(f"Using JSON file: {json_filename}")
except FileNotFoundError:
    print(f"Error: {json_filename} not found!")
    print("Usage: python3 HillClimbingStochastic.py [json_filename]")
    print("Example: python3 HillClimbingStochastic.py case2.json")
    sys.exit(1)
```

Pada bagian ini, *dataset* dalam bentuk JSON akan dibaca oleh program. Beberapa variabel yang digunakan, antara lain:

1. `kapasitas_kontainer`: kapasitas masing-masing kontainer, nilainya sama
2. `barang`: setiap barang disimpan dalam sebuah *dictionary* dengan format {‘id’, ‘ukuran’}

3. `jumlah_barang`: banyaknya barang, dalam bentuk *integer*

➢ Penempatan Awal Barang (*Initial Solution*)

```
kapasitas_kontainer   = data['kapasitas_kontainer']  barang                =
data['barang'].copy()
jumlah_barang         = len(barang)
kontainer             = []
kontainer_id          = 0

print(f"Loaded data from JSON file:")
print(f"Kapasitas kontainer: {kapasitas_kontainer} kg/m³")
print(f"Jumlah barang: {jumlah_barang}")
print(f"Barang: {[f"{item['id']}({item['ukuran']})" for item in barang]}")

kontainer.append([])
kontainer_space_left = kapasitas_kontainer

while True:
    if len(barang) == 0:
        break
    random_index   = random.randint(0, len(barang) - 1)
    random_barang  = barang[random_index]

    if random_barang['ukuran'] <= kontainer_space_left:
        kontainer[kontainer_id].append({
            'barang': random_barang
        })

        kontainer_space_left -= random_barang['ukuran']
        barang.pop(random_index)
    else:
        kontainer_id += 1
        kontainer.append([])
        kontainer_space_left = kapasitas_kontainer
```

Pada bagian ini, solusi awal dihasilkan dengan memasukkan setiap barang ke dalam kontainer secara acak. Apabila kontainer *overflow*, akan dibuat kontainer baru. Variabel yang digunakan, antara lain:

1. `kontainer`: tempat menyimpan barang yang ada
2. `kontainer_id`: kode unik untuk setiap kontainer
3. `kontainer_space_left`: ruang tersisa dalam kontainer

➢ Evaluasi (*Helper Function*)

```
def calculate_kontainer_total(kontainer):
    total = 0
    for arr_barang in kontainer:
        total += arr_barang['barang']['ukuran']
    return total

def calculate_unused(kontainer, kapasitas):
    total_unused = 0
    for container in kontainer:
        used          = calculate_kontainer_total(container)
        unused        = kapasitas - used
        total_unused += unused
    return total_unused
```

Terdapat dua fungsi bantuan yang digunakan dalam program, yaitu:

1. `calculate_kontainer_total(kontainer)`: menghitung total ukuran barang pada suatu kontainer
2. `calculate_unused(kontainer, kapasitas)`: menghitung total ruang tersisadi seluruh kontainer. Digunakan untuk evaluasi menyeluruh, semakin kecil nilainya, semakin baik

➢ Inisialisasi Loop Utama

```
current_iteration = 0
max_iterations    = 4
improvement_found   = True

while improvement_found and current_iteration < max_iterations:
    improvement_found  = False
```

Terdapat beberapa variabel yang digunakan, yaitu:

1. `current_iteration`: menandakan perhitungan percobaan. Setiap kali terjadi *swap*, nilai ini akan di reset ke 0
2. `max_iterations`: batas maksimum percobaan dalam satu kali eksplorasi
3. `improvement_found`: *checkpoint* untuk menandakan apakah ada *swap* yang diterima atau tidak

Loop akan terus berjalan selama ada perbaikan di *swap* sebelumnya dan `current_iteration` belum mencapai `max_iterations`. Di awal setiap iterasi *loop*, `improvement_found` di-*set* ulang ke *false* agar bisa mendeteksi apakah iterasi ini menjadi lebih baik. *Loop* pencarian akan berhenti karena tidak ada solusi yang lebih baik atau mencapai batas iterasi.

➢ Generate Neighbor

```
possible_swaps = []
 for i in range(len(kontainer)):
     for j in range (i + 1, len(kontainer)):
         for index_i in range(len(kontainer[i])):
             for index_j in range(len(kontainer[j])):
                 possible_swaps.append((i, j, index_i, index_j))

 random.shuffle(possible_swaps) # Untuk pemilihan neighbor random
 current_iteration = 0
```

Pada bagian ini, program akan menghasilkan semua pasangan yang menjadi kandidat *swap*. Kandidat diambil dari dua buah kontainer berbeda. Setiap pengujian, urutannya akan diacak. Setelah itu, *`current_iteration`* di-*set* menjadi 0 untuk memulai perhitungan.

➢ Loop Utama

```
for current_swap in possible_swaps:
     if current_iteration >= max_iterations:
         break

     i, j, index_i, index_j = current_swap

     current_total_i = calculate_kontainer_total(kontainer[i])

     if current_total_i < kapasitas_kontainer:
         current_iteration += 1
         print(f"Iteration {current_iteration}")

         current_total_j = calculate_kontainer_total(kontainer[j])
```

```
            barang_i_temp   = kontainer[i][index_i]['barang']
            barang_j_temp   = kontainer[j][index_j]['barang']

            new_total_i     = current_total_i - barang_i_temp['ukuran']
+ barang_j_temp['ukuran']
            new_total_j     = current_total_j - barang_j_temp['ukuran']
+ barang_i_temp['ukuran']

            if new_total_i <= kapasitas_kontainer and new_total_j <=
kapasitas_kontainer: # Overflow countermeassure

                if (kapasitas_kontainer - new_total_i) <
(kapasitas_kontainer - current_total_i): # Change Here untuk Sideways
Modifcation, Current Algorithm is Steepest (Neighbor [>]BETTER[>]
Current)
                    kontainer[i][index_i]['barang'] = barang_j_temp
                    kontainer[j][index_j]['barang'] = barang_i_temp
                    improvement_found = True
                    print(f"[!!!Swap!!!]: Swapped {barang_i_temp['id']}
↔ {barang_j_temp['id']} (Waste reduced by {new_total_i -
current_total_i})")
                    current_iteration = 0

                else: # Local Maxima break
                    break

            else: # Overflow break
                break

        else: # 100% Capacity break
            break
```

Berikut ini adalah proses yang terjaid di dalam *loop*, antara lain:

1. Mengambil kandidat dari daftar acak untuk dilakukan *swap*. Jumlah percobaan akan dibatasi oleh `max_iterations`

2. Perhitungan *help function* `calculate_kontainer_total(kontainer[i])` untuk variabel `current_total_i`. Variabel ini berisi informasi total isi barang di dalam kontainer-*i*
3. Jika kapasitas kontainer sudah penuh, tidak perlu melakukan *swap*. Program akan melakukan *break*
4. Simulasi *swap* barang akan dilakukan untuk menentukan apakah *swap* yang dilakukan memberikan solusi yang lebih baik atau buruk
5. Selanjutnya, program melakukan pengecekan *overflow*. Jika kapasitas kontainer sudah penuh, program akan keluar dari *loop* ini
6. Jika ruang tersisa lebih kecil dari pada sebelumnya, *swap* akan dilakukan. Program melakukan pengecekan dengan variabel `unused_after` dan `unused_before`
7. Setiap kali *swap* dilakukan, terdapat beberapa variabel yang diubah, yaitu:
    a. `improvement_found`: nilainya berubah menjadi true agar loop tetap berjalan
    b. `current_iteration`: nilainya berubah menjadi 0 untuk terus mencari solusi yang lebih baik, selama belum mencapai max_iterations
8. Jika *swap* tidak memberikan solusi yang lebih baik, program akan melakukan *break*

2.2.3 *Simulated Annealing*

a. Deskripsi

Algoritma *Simulated Annealing* menempatkan setiap barang ke dalam kontainer secara acak, ini akan menjadi *initial state*. Selanjutnya, algoritma akan mencari semua kemungkinan pasangan barang untuk di-*swap* antar dua kontainer berbeda. Iterasi akan dilakukan ke setiap pasangan *swap*, tetapi urutannya diacak terlebih dahulu. Algoritma akan melakukan iterasi ke semua pasangan *swap* selama *temperature* masih tinggi.

Fungsi objektif dalam algoritma akan membandingkan ruang tersisa di kontainer pertama. Jika ruang tersisa lebih baik dari *current state*, algoritma akan melakukan *swap*. Jika tidak, algoritma akan melakukan perhitungan probabilitas yang melibatkan variabel *temperature*. Selama *temperature* cenderung tinggi, algoritma cenderung lebih mudah menerima *neighbour* dengan kondisi yang lebih buruk. Seiring menurunnya nilai

temperature, probabilitas semakin mendekati nol dan algoritma hanya mempertahankan *neighbour* yang meningkatkan efisiensi *packing* barang.

b. *Source Code*

```
import random
import math
import json
import sys
import os
import matplotlib.pyplot as plt
import numpy as np
import time

import os
script_dir = os.path.dirname(os.path.abspath(__file__))

if len(sys.argv) > 1:
    json_filename = sys.argv[1]
else:
    json_filename = 'case6.json'

json_path = os.path.join(script_dir, json_filename)

try:
    with open(json_path, 'r') as f:
        data = json.load(f)
    print(f"Using JSON file: {json_filename}")
except FileNotFoundError:
    print(f"Error: {json_filename} not found!")
    print("Usage: python3 SimulatedAnnealing.py [json_filename]")
    print("Example: python3 SimulatedAnnealing.py case2.json")
    sys.exit(1)

# Variable Library
kapasitas_kontainer   = data['kapasitas_kontainer']   # Kapasitas Kontainer
barang                = data['barang'].copy()         # ID Barang
jumlah_barang         = len(barang)                   # Jumlah Barang
kontainer             = []                            # Kontainer
kontainer_id          = 0                             # ID Kontainer

print(f"Loaded data from JSON file:")
print(f"Kapasitas kontainer: {kapasitas_kontainer} kg/m³")
```

```
print(f"Jumlah barang: {jumlah_barang}")
print(f"Barang: {[f"{item['id']}({item['ukuran']})" for item in barang]}")

kontainer.append([])
kontainer_space_left = kapasitas_kontainer

while True:
    if len(barang) == 0:
        break

    random_index   = random.randint(0, len(barang) - 1)
    random_barang  = barang[random_index]

    if random_barang['ukuran'] <= kontainer_space_left:
        kontainer[kontainer_id].append({
            'barang': random_barang
        })

        kontainer_space_left -= random_barang['ukuran']
        barang.pop(random_index)

    else:
        kontainer_id += 1
        kontainer.append([])
        kontainer_space_left = kapasitas_kontainer

print("\n" + "="*60)
print("SPAWN BARANG DALAM KONTAINER")
print("="*60)

for idx, container in enumerate(kontainer):
    print(f"\nKontainer {idx + 1}:")
    total_ukuran = 0

    for item in container:
        barang_i_tempnfo = item['barang']
                        print(f"     -   ID:   {barang_i_tempnfo['id']},   Ukuran:
{barang_i_tempnfo['ukuran']} kg/m³")
        total_ukuran += barang_i_tempnfo['ukuran']

    sisa_kapasitas = kapasitas_kontainer - total_ukuran
    print(f"  Total Terisi: {total_ukuran}/{kapasitas_kontainer} kg/m³")
    print(f"  Sisa Kapasitas: {sisa_kapasitas} kg/m³")
```

```
    print(f"  Efisiensi: {(total_ukuran/kapasitas_kontainer)*100:.2f}%")

print("\n" + "="*60)
print(f"Total Kontainer Digunakan: {len(kontainer)}")
print("="*60)

def calculate_kontainer_total(kontainer):
    total = 0
    for arr_barang in kontainer:
        total += arr_barang['barang']['ukuran']
    return total

def calculate_unused(kontainer, kapasitas):
    total_unused = 0
    for container in kontainer:
        used          = calculate_kontainer_total(container)
        unused        = kapasitas - used
        total_unused += unused
    return total_unused

def calculate_objective_function(kontainer, kapasitas):
    P_OVERFLOW = 1000
    P_BINS     = 1.0
    P_DENSITY  = 0.1

    K = len(kontainer)

    total_overflow          = 0
    sum_squared_fill_ratios = 0

    for container in kontainer:
        total_size = calculate_kontainer_total(container)

        if total_size > kapasitas:
            overflow        = total_size - kapasitas
            total_overflow += overflow

        if len(container) > 0:
            fill_ratio               = total_size / kapasitas
            sum_squared_fill_ratios += fill_ratio ** 2

       cost  =  (P_OVERFLOW  *  total_overflow)  +  (P_BINS  *  K)  -  (P_DENSITY  *
sum_squared_fill_ratios)
```

```
    return cost, K, total_overflow, sum_squared_fill_ratios

def calculate_initial_temperature(kontainer, kapasitas):

    max_delta_E = 0

    for i in range(len(kontainer) - 1):
        j = i + 1

        for index_i in range(len(kontainer[i])):
            for index_j in range(len(kontainer[j])):
                current_total_i = calculate_kontainer_total(kontainer[i])
                current_total_j = calculate_kontainer_total(kontainer[j])

                barang_i_temp = kontainer[i][index_i]['barang']
                barang_j_temp = kontainer[j][index_j]['barang']

                    new_total_i = current_total_i - barang_i_temp['ukuran'] +
barang_j_temp['ukuran']
                    new_total_j = current_total_j - barang_j_temp['ukuran'] +
barang_i_temp['ukuran']

                current_unused_i = kapasitas - current_total_i
                new_unused_i = kapasitas - new_total_i

                delta_E = abs(new_unused_i - current_unused_i)

                if delta_E > max_delta_E:
                    max_delta_E = delta_E

    return max_delta_E

acceptance_probability  = []
iterations              = []
history_POF             = []

start_time        = time.time()
temperature                         =    calculate_initial_temperature(kontainer,
kapasitas_kontainer)
alpha             = 0.985
min_temperature   = 0.01
max_iterations    = 1000
```

```
current_iteration = 0

# var untuk local optimum
local_stucked       = 0
sideways_counter    = 0
SIDEWAYS_THRESHOLD  = 5
last_changed_POF    = float('-inf')

print(f"\nInitial Temperature (T₀): {temperature:.2f}")
print(f"Alpha (cooling rate): {alpha}")
print(f"Min Temperature: {min_temperature}")
print(f"Max Iterations: {max_iterations}")
print("="*60)

while temperature > min_temperature and current_iteration < max_iterations:
    # Objective Function Data Logging
          current_POF,   _,   _,   _   =   calculate_objective_function(kontainer,
kapasitas_kontainer)
    history_POF.append(current_POF)

    if last_changed_POF != current_POF:
        last_changed_POF = current_POF
        sideways_counter = 0

    else:
        sideways_counter += 1

    if sideways_counter >= SIDEWAYS_THRESHOLD:
        local_stucked += 1
        sideways_counter = 0

    possible_swaps = []
    for i in range(len(kontainer)):
        for j in range(i + 1, len(kontainer)):
            for index_i in range(len(kontainer[i])):
                for index_j in range(len(kontainer[j])):
                    possible_swaps.append((i, j, index_i, index_j))

    if not possible_swaps:
        print("No possible swaps available. Stopping.")
        break

    i, j, index_i, index_j = random.choice(possible_swaps)
```

```
    current_total_i = calculate_kontainer_total(kontainer[i])
    current_total_j = calculate_kontainer_total(kontainer[j])

    barang_i_temp = kontainer[i][index_i]['barang']
    barang_j_temp = kontainer[j][index_j]['barang']

            new_total_i   =   current_total_i   -   barang_i_temp['ukuran']   +
barang_j_temp['ukuran']
            new_total_j   =   current_total_j   -   barang_j_temp['ukuran']   +
barang_i_temp['ukuran']

            if   new_total_i   <=   kapasitas_kontainer   and   new_total_j   <=
kapasitas_kontainer:
        current_unused = (kapasitas_kontainer - current_total_i)
        new_unused = (kapasitas_kontainer - new_total_i)

        delta_E = new_unused - current_unused

        if delta_E >= 0:
            accept = True
            probability = 1.0

        else:
            probability = math.exp(delta_E / temperature)
            accept      = random.random() < probability

        iterations.append(current_iteration)
        acceptance_probability.append(probability)

        if accept:
            kontainer[i][index_i]['barang'] = barang_j_temp
            kontainer[j][index_j]['barang'] = barang_i_temp

            if delta_E >= 0:
                                    print(f"Iter  {current_iteration+1}:  [ACCEPT]
{barang_i_temp['id']}    ⇔    {barang_j_temp['id']}    |    ΔE={delta_E:.2f}    |
T={temperature:.2f}")
            else:
                                print(f"Iter  {current_iteration+1}:  [PROB-ACCEPT]
{barang_i_temp['id']}    ⇔    {barang_j_temp['id']}    |    ΔE={delta_E:.2f}    |
P={probability:.4f} | T={temperature:.2f}")
        else:
```

```
            print(f"Iter {current_iteration+1}: [REJECT] {barang_i_temp['id']} ↔
{barang_j_temp['id']}    |    ΔE={delta_E:.2f}    |    P={probability:.4f}    |
T={temperature:.2f}")

    temperature         *= alpha
    current_iteration   += 1

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

print("\n" + "="*60)
print("HASIL PENYIMPANAN BARANG DALAM KONTAINER")
print("="*60)

for idx, container in enumerate(kontainer):
    print(f"\nKontainer {idx + 1}:")
    total_ukuran = 0

    for item in container:
        barang_i_tempnfo = item['barang']
                    print(f"     -   ID:   {barang_i_tempnfo['id']},   Ukuran:
{barang_i_tempnfo['ukuran']} kg/m³")
        total_ukuran += barang_i_tempnfo['ukuran']

    sisa_kapasitas = kapasitas_kontainer - total_ukuran
    print(f"  Total Terisi: {total_ukuran}/{kapasitas_kontainer} kg/m³")
    print(f"  Sisa Kapasitas: {sisa_kapasitas} kg/m³")
    print(f"  Efisiensi: {(total_ukuran/kapasitas_kontainer)*100:.2f}%")

print("\n" + "="*60)
print(f"Total Kontainer Digunakan: {len(kontainer)}")
print("="*60)

print(f"\nSimulated Annealing completed after {current_iteration} iterations.")
print(f"Final Temperature: {temperature:.2f}")
print(f"Execution Time: {execution_time:.4f} seconds")

print(f"Current case of Simulated Annealing made {local_stucked} times")

# plot 1
plt.ioff()

plt.figure(figsize=(10, 5))
```

```
plt.scatter(iterations, acceptance_probability, c='lightblue', s=20, alpha=0.6, label='Acceptance Probability')

if len(iterations) > 1:
    z = np.polyfit(iterations, acceptance_probability, 3)
    p = np.poly1d(z)
    plt.plot(iterations, p(iterations), 'b-', linewidth=2, label='Trend Line')

plt.xlabel('Iteration', fontsize=12)
plt.ylabel('Probability', fontsize=12)
plt.title('Acceptance Probability over Iterations (ΔE < 0)', fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3)
plt.legend()
plt.ylim(-0.02, 1.02)
plt.tight_layout()
plt.show()

# plot 2
plt.figure(figsize=(10, 5))
plt.plot(range(len(history_POF)), history_POF, 'r-', linewidth=2, label='Objective Function Value')
plt.xlabel('Iteration', fontsize=12)
plt.ylabel('Objective Function Value', fontsize=12)
plt.title('Objective Function (POF) over Iterations', fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```

c. Penjelasan *Source Code*

➢ Inisialisasi Awal

```
import random
import math
import json
import sys

random.seed(42)

import os
```

```
script_dir = os.path.dirname(os.path.abspath(__file__))

if len(sys.argv) > 1:
    json_filename = sys.argv[1]
else:
    json_filename = 'case1.json'  # Default

json_path = os.path.join(script_dir, json_filename)

try:
    with open(json_path, 'r') as f:
        data = json.load(f)
    print(f"Using JSON file: {json_filename}")
except FileNotFoundError:
    print(f"Error: {json_filename} not found!")
    print("Usage: python3 SimulatedAnnealing.py [json_filename]")
    print("Example: python3 SimulatedAnnealing.py case2.json")
    sys.exit(1)

kapasitas_kontainer   = data['kapasitas_kontainer']
barang                = data['barang'].copy()
jumlah_barang         = len(barang)
kontainer             = []
kontainer_id          = 0
```

Pada bagian ini, program menerima *dataset* dalam bentuk JSON. Fitur *random* juga menggunakan *seed* agar hasil acak selalu sama dan hasil eksperimen bisa dibandingkan. Terdapat beberapa variabel, antara lain:

1. `kapasitas_kontainer`: kapasitas maksimum tiap kontainer
2. `barang`: setiap barang direpresentasikan dalam bentuk *dictionary* dengan format {'id': ..., 'ukuran': ...}
3. `jumlah_barang`: banyaknya barang
4. `kontainer`: tempat penyimpanan barang
5.

➢ Mencari Ukuran Barang Terbesar dan Terkecil

```
min_ukuran = min(item['ukuran'] for item in barang)
max_ukuran = max(item['ukuran'] for item in barang)
```

Pada bagian ini, program mencari barang dengan ukuran terbesar dan terkecil. Tujuannya adalah menghitung *temperature* awal.

➢ Initial State

```
kontainer.append([])
kontainer_space_left = kapasitas_kontainer

while True:
    if len(barang) == 0:
        break

    random_index   = random.randint(0, len(barang) - 1)
    random_barang  = barang[random_index]

    if random_barang['ukuran'] <= kontainer_space_left:
        kontainer[kontainer_id].append({
            'barang': random_barang
        })

        kontainer_space_left -= random_barang['ukuran']
        barang.pop(random_index)

    else:
        kontainer_id += 1
        kontainer.append([])
        kontainer_space_left = kapasitas_kontainer
```

Pada bagian ini, setiap barang akan dimasukkan ke dalam kontainer secara acak. Jika kontainer penuh, akan dibuat kontainer baru. Hasil akhir dari proses ini adalah sebuah solusi awal yang akan dioptimasi oleh algoritma *simulated annealing*. Terdapat beberapa variabel yang digunakan, antara lain:

1. `kontainer`: *list* barang yang terdapat di setiap kontainer
2. `kontainer_id`: indeks setiap kontainer
3. `kontainer_space_left`: ruang tersisa di dalam kontainer

➢ Evaluasi (*Helper Function*)

```
def calculate_kontainer_total(kontainer):
    total = 0
    for arr_barang in kontainer:
        total += arr_barang['barang']['ukuran']
```

```
    return total

def calculate_unused(kontainer, kapasitas):
    total_unused = 0
    for container in kontainer:
        used          = calculate_kontainer_total(container)
        unused        = kapasitas - used
        total_unused += unused
    return total_unused
```

Terdapat dua fungsi bantuan yang digunakan dalam program, yaitu:

1. `calculate_kontainer_total(kontainer)`: menghitung total ukuran barang pada suatu kontainer
2. `calculate_unused(kontainer, kapasitas)`: menghitung total ruang tersisadi seluruh kontainer. Digunakan untuk evaluasi menyeluruh, semakin kecil nilainya, semakin baik

➢ Inisialisasi Parameter

```
temperature       = max_ukuran - min_ukuran
alpha             = 0.925
min_temperature   = 0.1
best_case_unused  = calculate_unused(kontainer, kapasitas_kontainer)
improvement_found = True
```

Berikut penjelasan setiap variabel:

1. temperature: selisih antara nilai max_ukuran dan min_ukuran
2. alpha: mengendalikan laju penurunan *temperature*
3. min_temperature: batas minimum *temperature*. Apabila sudah mencapai nilai ini, pencarian akan berhenti
4. best_case_unused: ruang tersisa dari solusi awal, digunakan sebagai acuan
5. improvement_found: *checkflag* untuk mengendalikan *loop* utama

➢ Loop Utama

```
while improvement_found and temperature > min_temperature:
    improvement_found  = False
```

Pada bagian ini, iterasi akan terus dilakukan selama *temperature* masih lebih besar dari `min_temperature` dan masih ada solusi yang lebih baik. Terminasi akan dilakukan pada kondisi berikut:

1. *Temperature* sangat kecil
2. Tidak ada *swap* yang diterima lagi, solusi sudah sampai di *global maximum* atau terjebak di *local maximum*

➢ Loop *Swap*

```
for i in range(len(kontainer)):
        for j in range (i + 1, len(kontainer)):

            for index_i in range(len(kontainer[i])):
                for index_j in range(len(kontainer[j])):
                    current_total_i = calculate_kontainer_total(kontainer[i])
```

Pada bagian ini, pencarian tetangga atau *neighbor* dilakukan dengan mencoba semua kemungkinan pasangan antar barang dari kontainer berbeda.

➢ Simulasi Efek *Swap*

```
if current_total_i < kapasitas_kontainer and temperature > min_temperature:
current_total_j = calculate_kontainer_total(kontainer[j])

                        barang_i_temp   = kontainer[i][index_i]['barang']
                        barang_j_temp   = kontainer[j][index_j]['barang']

                        new_total_i     = current_total_i -
barang_i_temp['ukuran'] + barang_j_temp['ukuran']
                        new_total_j     = current_total_j -
barang_j_temp['ukuran'] + barang_i_temp['ukuran']
```

Pada bagian ini, program melakukan simulasi dari proses *swap*. Terdapat dua variabel penting yang digunakan, antara lain:

1. `current_total_i`, `current_total_j`: ruang kontainer yang terisi sebelum *swap*
2. `new_total_i`, `new_total_j`: ruang kontainer yang terisi setelah *swap*

➢ Fungsi Evaluasi & Probabilitas Penerimaan Solusi Buruk

```
current_unused  = (kapasitas_kontainer - current_total_i)
                        new_unused      = (kapasitas_kontainer - new_total_i)
```

```
                        delta_E         = new_unused - current_unused
                        probability     = math.exp(delta_E/temperature)

                        print(f"Delta E: {delta_E}")
                        print(f"Temperature: {temperature:,.3f}")
                        print(f"Probability: {probability}")
                        temperature *= alpha
```

Pada bagian ini, algoritma melakukan perhitungan ΔE. Rumus yang digunakan adalah:

$$\Delta E = \texttt{new\_unused} - \texttt{current\_unused}$$

Terdapat dua kondisi yang mungkin, antara lain:

1. ΔE < 0 → *swap* membuat ruang kosong berkurang → lebih efisien
2. ΔE > 0 → *swap* membuat ruang kosong bertambah → lebih buruk

Untuk solusi yang buruk, algoritma masih bisa menerima dengan perhitungan probabilitas tertentu. Rumus yang digunakan adalah *Boltzmann Formula*:

$$P = e^{-\Delta E/T}$$

Seiring berjalannya algoritma, *temperature* akan berkurang secara geometrik. Rumus yang digunakan adalah:

$$T_j = T_i \times \alpha$$

➢ Keputusan Penerimaan *Swap*

```
if delta_E > 0: # ΔE > 0 akan selalu diambil
                            kontainer[i][index_i]['barang'] = barang_j_temp
                            kontainer[j][index_j]['barang'] = barang_i_temp
                            improvement_found = True
                            print(f"[!!!Swap!!!]: Swapped {barang_i_temp['id']}
↔ {barang_j_temp['id']} (Waste reduced by {new_total_i - current_total_i})")

                        else: # ΔE < 0 akan melihat probability (P)
                            if random.random() < probability:
                                kontainer[i][index_i]['barang'] = barang_j_temp
                                kontainer[j][index_j]['barang'] = barang_i_temp
                                improvement_found = True
```

Pada bagian ini, algoritma menentukan *swap* mana yang dilakukan dan tidak dilakukan. Terdapat beberapa kondisi, antara lain:

1. ΔE < 0 → *swap* langsung diterima
2. ΔE > 0 → *swap* diterima dengan probabilitas *boltzmann formula*

Tujuan menerima solusi buruk adalah mencoba keluar dari *local maximum* Setiap kali *swap* diterima, variabel `improvement_found` berubah menjadi *true*.

2.2.4 *Genetic Algorithm*

a. Deskripsi

Algoritma *Genetic Algorithm(GA)* mengikuti proses evolusi biologis melalui seleksi, *crossover*, dan mutasi. Dalam konteks ini, setiap solusi direpresentasikan sebagai kromosom, di mana setiap gen menunjukkan kontainer tempat suatu barang ditempatkan. Dalam algoritma ini, individu terbaik dipilih menggunakan nilai *fitness,* gen digabungkan antar individu, dan dilakukan mutasi acak untuk menemukan solusi yang paling optimal. Proses ini dilakukan iterasi hingga mencapai konvergensi atau batas iterasi.

b. Source Code

```
import random
import math
import json
import sys
import os
import matplotlib.pyplot as plt
import numpy as np
import time

script_dir = os.path.dirname(os.path.abspath(__file__))

if len(sys.argv) > 1:
    json_filename = sys.argv[1]
else:
    json_filename = 'case6.json'

json_path = os.path.join(script_dir, json_filename)

try:
    with open(json_path, 'r') as f:
        data = json.load(f)
    print(f"Using JSON file: {json_filename}")
except FileNotFoundError:
    print(f"Error: {json_filename} not found!")
    sys.exit(1)
```

```
# Variable Library
kapasitas_kontainer    = data['kapasitas_kontainer']  # kapasitas kontainer dalam
kg/m³
barang                 = data['barang'].copy()         # id barang
jumlah_barang          = len(barang)                   # jumlah barang
kontainer                  =  []                                # kontainer untuk
menyimpan barang
kontainer_id           = 0                             # id kontainer

print(f"Loaded data from JSON file:")
print(f"Kapasitas kontainer: {kapasitas_kontainer} kg/m³")
print(f"Jumlah barang: {jumlah_barang}")
print(f"Barang: {[f"{item['id']}({item['ukuran']})" for item in barang]}")

barang_unrandomized = barang.copy()

# declare kontainer array
kontainer.append([])
kontainer_space_left = kapasitas_kontainer

# declare penyebaran kontainer secara random
while True:
    if len(barang) == 0:
        break

    random_index   = random.randint(0, len(barang) - 1)
    random_barang  = barang[random_index]

    if random_barang['ukuran'] <= kontainer_space_left:
        kontainer[kontainer_id].append({
            'barang': random_barang
        })

        kontainer_space_left -= random_barang['ukuran']
        barang.pop(random_index)

    else:
        kontainer_id += 1
        kontainer.append([])
        kontainer_space_left = kapasitas_kontainer

print("\n" + "="*60)
print("SPAWN BARANG DALAM KONTAINER")
```

```
print("="*60)

for idx, container in enumerate(kontainer):
    print(f"\nKontainer {idx + 1}:")
    total_ukuran = 0

    for item in container:
        barang_i_temp = item['barang']
        print(f"  - ID: {barang_i_temp['id']}, Ukuran: {barang_i_temp['ukuran']}
kg/m³")
        total_ukuran += barang_i_temp['ukuran']

    sisa_kapasitas = kapasitas_kontainer - total_ukuran
    print(f"  Total Terisi: {total_ukuran}/{kapasitas_kontainer} kg/m³")
    print(f"  Sisa Kapasitas: {sisa_kapasitas} kg/m³")
    print(f"  Efisiensi: {(total_ukuran/kapasitas_kontainer)*100:.2f}%")

print("\n" + "="*60)
print(f"Total Kontainer Digunakan: {len(kontainer)}")
print("="*60)

# variabel untuk GA
main_kromosom           = []
main_objective_function = 0   # objective function untuk kromosom utama

for item_unrandomized in barang_unrandomized:
    for idx_container, container in enumerate(kontainer):
        for barang_in_container in container:
            if barang_in_container['barang']['id'] == item_unrandomized['id']:
                main_kromosom.append(idx_container + 1)
                break

print("\n" + "="*60)
print("GENETIC SEQUENCE (KROMOSOM)")
print("="*60)

for i, item_unrandomized in enumerate(barang_unrandomized):
            print(f"{item_unrandomized['id']}   (item   {i+1})   ->   Kontainer
{main_kromosom[i]}")

print(f"\nKromosom: {main_kromosom}")
print("="*60)
```

```
def calculate_objective_function(kromosom, barang, kapasitas): # objective function untuk fitness test
    P_OVERFLOW = 1000
    P_BINS     = 1.0
    P_DENSITY  = 0.1

    if kromosom:
        K = max(kromosom)
    else:
        K = 0

    containers = [[] for _ in range (K)]

    for i, container_num in enumerate(kromosom):
        containers[container_num - 1].append(barang[i]['ukuran'])

    total_overflow          = 0
    sum_squared_fill_ratios = 0

    for container in containers:
        total_size = sum(container)

        if total_size > kapasitas:
            overflow        = total_size - kapasitas
            total_overflow += overflow

        if len(container) > 0:
            fill_ratio               = total_size / kapasitas
            sum_squared_fill_ratios += fill_ratio ** 2

       cost = (P_OVERFLOW * total_overflow) + (P_BINS * K) - (P_DENSITY * sum_squared_fill_ratios)

    return cost, K, total_overflow, sum_squared_fill_ratios


main_objective_function      =      calculate_objective_function(main_kromosom, barang_unrandomized, kapasitas_kontainer)

def parent_crossover(parent_a, parent_b):
    if len(parent_a) != len(parent_b):
        raise ValueError("Parents must have the same length for crossover.")
```

```
    n = len(parent_a)
    if n < 2:
        return parent_a.copy()

    point = random.randint(1, n - 1)
    child = parent_a[:point] + parent_b[point:]
    return child

def mutate_offspring(offspring, mutation_rate):
    if not offspring:
        return offspring

    mutated = offspring.copy()
    if mutated:
        K_max = max(mutated)
    else:
        K_max = 1

    for i in range(len(mutated)):
        if random.random() < mutation_rate:
            mutated[i] = random.randint(1, K_max + 1)

    return mutated

def repair_offspring(offspring, barang, kapasitas):
    if not offspring:
        return offspring

    K_max = max(offspring) if offspring else 1
    containers = [[] for _ in range(K_max)]

    for i, container_num in enumerate(offspring):
        if container_num <= K_max:
            containers[container_num - 1].append((i, barang[i]['ukuran']))

    repaired = offspring.copy()

    for container_idx, container in enumerate(containers):
        total_size = sum(item[1] for item in container)

        if total_size > kapasitas:
            items_sorted = sorted(container, key=lambda x: x[1], reverse=True)
```

```
            for item_idx, item_size in items_sorted:
                 current_total = sum(item[1] for item in container if item[0] !=
item_idx)

                if current_total <= kapasitas:
                    placed = False
                    for target_bin in range(len(containers)):
                        if target_bin != container_idx:
                                        target_total = sum(item[1] for item in
containers[target_bin])
                            if target_total + item_size <= kapasitas:
                                repaired[item_idx] = target_bin + 1
                                        containers[target_bin].append((item_idx,
item_size))
                                placed = True
                                break

                    if not placed:
                        K_max += 1
                        repaired[item_idx] = K_max
                        containers.append([(item_idx, item_size)])

                          container = [item for item in container if item[0] !=
item_idx]

    return repaired

# library untuk genetic algorithm
POPULATION_SIZE = 50
MAX_ITERATIONS  = 200
MUTATION_RATE   = 0.05
TOURNAMENT_SIZE = 5

population = []
population.append(main_kromosom.copy())

for _ in range(POPULATION_SIZE - 1):
    random_kromosom = []
    if barang_unrandomized:
        K_max = len(kontainer)
    else:
        K_max = 1
```

```
    for item in barang_unrandomized:
        random_kromosom.append(random.randint(1, K_max + 2))

    population.append(random_kromosom)

def tournament_selection(population, barang, kapasitas, tournament_size):
             tournament    =    random.sample(population,    min(tournament_size,
len(population)))

    best = tournament[0]
        best_fitness,  _,  _,  _  =  calculate_objective_function(best,  barang,
kapasitas)

    for individual in tournament[1:]:
           fitness, _, _, _ = calculate_objective_function(individual, barang,
kapasitas)
        if fitness < best_fitness:
            best = individual
            best_fitness = fitness

    return best.copy()

best_kromosom = None
best_fitness = float('inf')

iterations  = []
history_POF = []
improvement_count = 0

print("\n" + "="*60)
print("GENETIC ALGORITHM - STARTING")
print("="*60)

start_time = time.time()

for iteration in range(MAX_ITERATIONS):
    fitness_scores = []
    for individual in population:
               fitness,  _,  _,  _  =  calculate_objective_function(individual,
barang_unrandomized, kapasitas_kontainer)
        fitness_scores.append(fitness)

    min_fitness_idx = fitness_scores.index(min(fitness_scores))
```

```
    current_best_fitness = fitness_scores[min_fitness_idx]

    avg_fitness = sum(fitness_scores) / len(fitness_scores)
    worst_fitness = max(fitness_scores)

    if current_best_fitness < best_fitness:
        best_fitness = current_best_fitness
        best_kromosom = population[min_fitness_idx].copy()
        improvement_count += 1

                        best_cost,   best_K,   best_overflow,   best_density   =
calculate_objective_function(
            best_kromosom, barang_unrandomized, kapasitas_kontainer
        )

               print(f"[IMPROVEMENT  #{improvement_count}]  Iter  {iteration+1}:
fitness={best_fitness:.2f}, K={best_K}, overflow={best_overflow}")

    iterations.append(iteration + 1)
    history_POF.append(best_fitness)

    new_population = []

    new_population.append(best_kromosom.copy())

    while len(new_population) < POPULATION_SIZE:
               parent1  =  tournament_selection(population,  barang_unrandomized,
kapasitas_kontainer, TOURNAMENT_SIZE)
               parent2  =  tournament_selection(population,  barang_unrandomized,
kapasitas_kontainer, TOURNAMENT_SIZE)

        offspring = parent_crossover(parent1, parent2)
        offspring = mutate_offspring(offspring, MUTATION_RATE)
                 offspring  =  repair_offspring(offspring,  barang_unrandomized,
kapasitas_kontainer)

        new_population.append(offspring)

    population = new_population

end_time = time.time()
execution_time = end_time - start_time
```

```
print("\n" + "="*60)
print("GENETIC ALGORITHM - COMPLETED")
print(f"Execution Time: {execution_time:.4f} seconds ({execution_time*1000:.2f}
ms)")
print(f"Total Improvements: {improvement_count}")
print("="*60)

print("\n" + "="*60)
print("FINAL RESULT")
print("="*60)
print(f"Best Fitness: {best_fitness:.2f}")
print(f"Jumlah iterasi: {MAX_ITERATIONS}")
print(f"Jumlah populasi: {len(population)}")
print(f"Final Kromosom: {best_kromosom}")
print("="*60)

plt.ioff()

# plot 1: objective function over iterations
plt.figure(figsize=(10, 5))
plt.plot(iterations, history_POF, 'b-', linewidth=2, label='Best Fitness')
plt.xlabel('Iteration', fontsize=12)
plt.ylabel('Fitness (Lower is Better)', fontsize=12)
plt.title('Genetic Algorithm - Fitness Evolution', fontsize=14,
fontweight='bold')
plt.grid(True, alpha=0.3)
plt.legend()

plt.tight_layout()
plt.show()
```

c. Penjelasan *Source Code*

➢ Inisialisasi Awal

Pada bagian ini, program membaca file JSON untuk mendapatkan daftar barang dan kapasitas kontainer. Setiap barang akan ditempatkan secara acak ke dalam kontainer yang tersedia. Jika kontainer penuh, akan dibuat kontainer baru. Kemudian, hasil akhir dari proses ini membentuk solusi inisial yang akan diubah menjadi representasi kromosom untuk proses evolusi.

```
kapasitas_kontainer = data['kapasitas_kontainer']
barang = data['barang'].copy()
```

```
jumlah_barang = len(barang)
kontainer = []
kontainer_id = 0

kontainer.append([])
kontainer_space_left = kapasitas_kontainer

print(f"Loaded data from JSON file:")
print(f"Kapasitas kontainer: {kapasitas_kontainer} kg/m³")
print(f"Jumlah barang: {jumlah_barang}")
print(f"Barang: {[f"{item['id']}({item['ukuran']})" for item in barang]}")

barang_unrandomized = barang.copy()

while True:
    if len(barang) == 0:
        break
    random_index = random.randint(0, len(barang) - 1)
    random_barang = barang[random_index]

    if random_barang['ukuran'] <= kontainer_space_left:
        kontainer[kontainer_id].append({'barang': random_barang})
        kontainer_space_left -= random_barang['ukuran']
        barang.pop(random_index)
    else:
        kontainer_id += 1
        kontainer.append([])
        kontainer_space_left = kapasitas_kontainer
```

Terdapat beberapa variabel, antara lain:

1. `kontainer`: list yang menyimpan barang-barang pada setiap kontainer
2. `kontainer_id`: indeks dari kontainer yang sedang aktif
3. `kontainer_space_left`: ruang kosong yang tersisa dalam kontainer saat ini

➢ Representasi Kromosom

Setelah solusi awal terbentuk setiap barang dikonversi menjadi gen dalam kromosom (`main_kromosom`). Nilai gen menunjukkan indeks kontainer tempat barang ditempatkan.

```
main_kromosom = []
for item_unrandomized in barang_unrandomized:
```

```
    for idx_container, container in enumerate(kontainer):
        for barang_in_container in container:
            if barang_in_container['barang']['id'] == item_unrandomized['id']:
                main_kromosom.append(idx_container + 1)
                break
```

Contoh hasil adalah `[1,2,1,3]` berarti barang 1 di kontainer-1, barang 2 di kontainer-2, dan seterusnya. Representasi ini mempermudah proses *crossover* dan mutasi karena solusi disimpan dalam bentuk numerik.

➢ Fungsi Evaluasi*(Objective Function)*

Bagian ini berfungsi untuk menghitung nilai *fitness* dari setiap kromosom. Nilai objektif dihitung berdasarkan penalti *overflow*, jumlah kontainer yang digunakan, serta *reward* berdasarkan tingkat kepadatan kontainer.

```
def calculate_objective_function(kromosom, barang, kapasitas):
    P_OVERFLOW = 1000
    P_BINS = 1.0
    P_DENSITY = 0.1
    ...
       cost = (P_OVERFLOW * total_overflow) + (P_BINS * K) - (P_DENSITY *
sum_squared_fill_ratios)
    return cost, K, total_overflow, sum_squared_fill_ratios
```

Terdapat tiga komponen utama dalam perhitungan:

1. *Overflow penalty:* penalti jika kontainer melebihi kapasitas
2. Jumlah kontainer(K): semakin sedikit jumlahnya, semakin baik
3. *Reward* kepadatan: semakin padat isi kontainer, semakin tinggi nilainya.

Nilai *fitness* terbaik ditandai dengan *total cost* yang paling kecil.

➢ Operator Genetik

Terdapat tiga operator utama yang digunakan untuk menghasilkan populasi baru:

1. *Crossover*

   Menggabungkan dua *parent* menjadi satu anak baru menggunakan titik potong acak.

```
def parent_crossover(parent_a, parent_b):
    point = random.randint(1, len(parent_a) - 1)
    child = parent_a[:point] + parent_b[point:]
```

```
    return child
```

2. *Mutation*

   Perubahan acak terhadap beberapa gen berdasarkan probabilitas tertentu.

```
def mutate_offspring(offspring, mutation_rate):
    for i in range(len(offspring)):
        if random.random() < mutation_rate:
            offspring[i] = random.randint(1, K_max + 1)
```

3. *Repair*

   Jika hasil *mutation* menyebabkan overflow, fungsi ini memperbaikinya dengan memindahkan barang ke kontainer lain.

```
def repair_offspring(offspring, barang, kapasitas):
    if total_size > kapasitas:
        items_sorted = sorted(container, key=lambda x: x[1], reverse=True)
```

➢ Seleksi dan Pembentukan Populasi

Tournament selection digunakan untuk seleksi, dimana beberapa individu dipilih secara acak dan individu dengan *fitness* terbaik akan digunakan sebagai *parent.* Proses ini dilakukan untuk memastikan individu yang lebih baik memiliki peluang lebih besar untuk berkembang ke generasi berikutnya

```
def tournament_selection(population, barang, kapasitas, tournament_size):
             tournament    =    random.sample(population,    min(tournament_size,
len(population)))
    best = tournament[0]
    for individual in tournament[1:]:
            fitness, _, _, _ = calculate_objective_function(individual, barang,
kapasitas)
        if fitness < best_fitness:
            best = individual
    return best.copy()
```

➢ *Loop* Utama

Algoritma melakukan iterasi sampai jumlah maksimum(`MAX_ITERATIONS = 200`). Setiap iterasi mencakup perhitungan *fitness* seluruh populasi, seleksi individu terbaik, serta pembentukan populasi baru melalui tiga operator utama.

```
for iteration in range(MAX_ITERATIONS):
    fitness_scores = []
    for individual in population:
                fitness,  _,  _,  _  =  calculate_objective_function(individual,
barang_unrandomized, kapasitas_kontainer)
        fitness_scores.append(fitness)

    min_fitness_idx = fitness_scores.index(min(fitness_scores))
    current_best_fitness = fitness_scores[min_fitness_idx]
```

➢ Visualisasi dan Hasil Akhir

Setelah semua iterasi selesai, program menampilkan hasil akhir berupa nilai *fitness* terbaik, jumlah kontainer, serta kromosom final. Terdapat juga grafik evolusi nilai *fitness* yang menunjukkan tren perbaikan selama proses evolusi.

```
plt.plot(iterations, history_POF, 'b-', linewidth=2, label='Best Fitness')
plt.xlabel('Iteration')
plt.ylabel('Fitness (Lower is Better)')
plt.title('Genetic Algorithm - Fitness Evolution')
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()
```

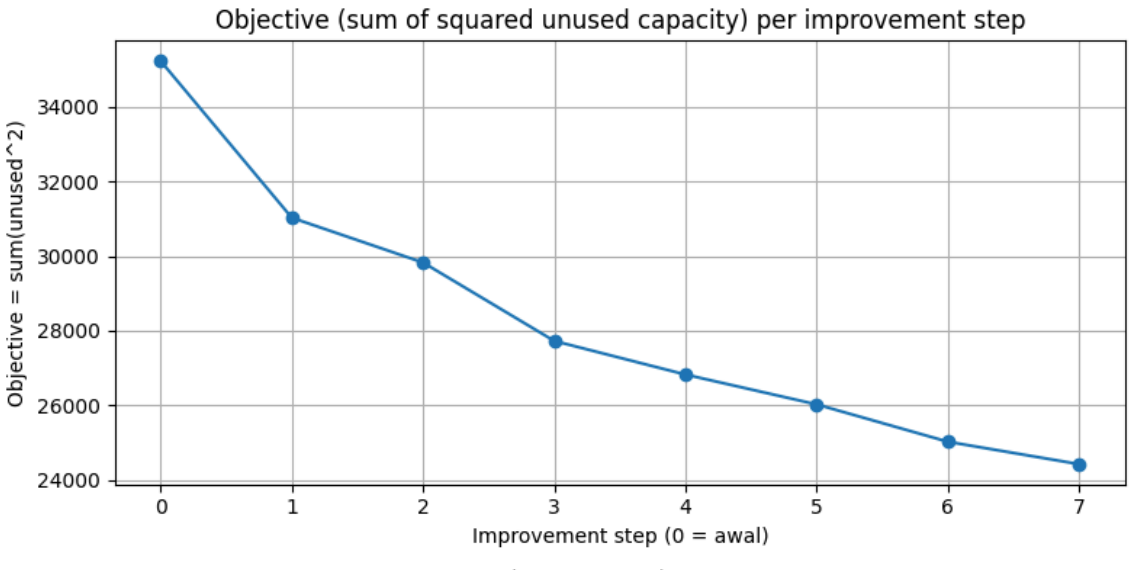
2.3 Hasil Eksperimen dan Analisis

2.3.1 *Steepest Ascent Hill-Climbing*

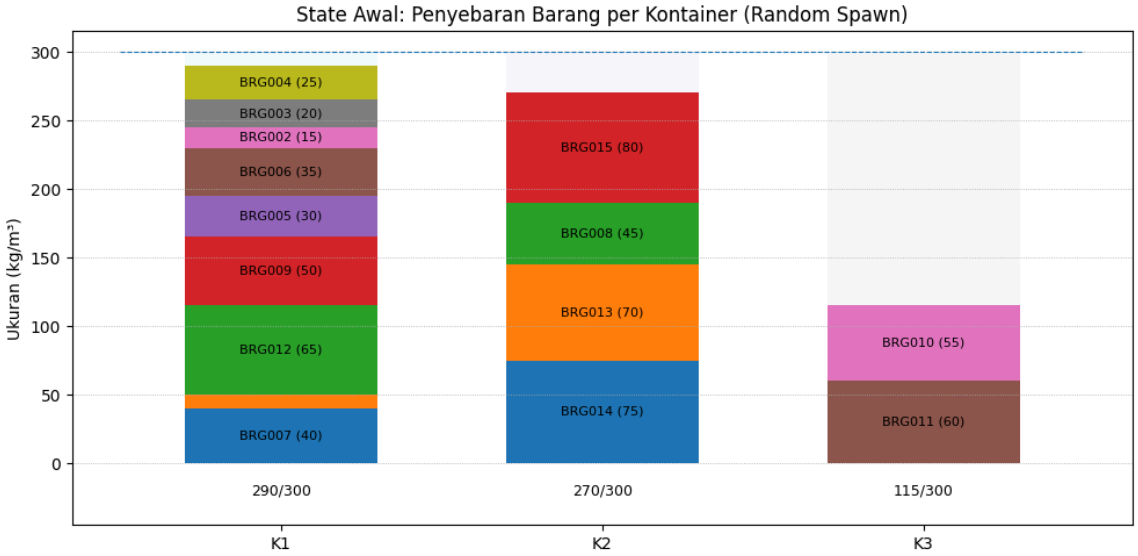
Tabel 1. Hasil Eksperimen *Steepest Ascent Hill-Climbing*

Run	*Final Objective Value*	*State* Awal	*State* Akhir	Durasi Pencarian	Iterasi hingga berhenti
1	15500 → 9550	Kontainer 1: ['BRG002', 'BRG001'] Kontainer 2: ['BRG006', 'BRG004', 'BRG005'] Kontainer 3: ['BRG010', 'BRG003'] Kontainer 4: ['BRG008'] Kontainer 5: ['BRG007']	Kontainer 1: ['BRG003', 'BRG004'] Kontainer 2: ['BRG009', 'BRG001', 'BRG008'] Kontainer 3: ['BRG006', 'BRG010'] Kontainer 4: ['BRG002'] Kontainer 5: ['BRG007']	0.0006 detik	7

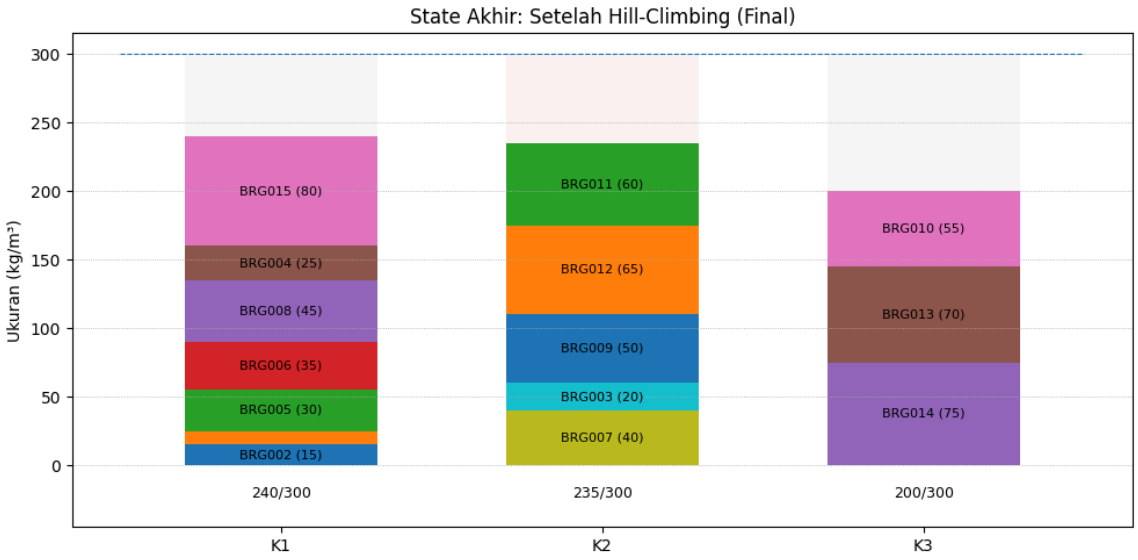
		Kontainer 6: ['BRG009']	Kontainer 6: ['BRG005']		
2	20066 → 14068	Kontainer 1: ['BRG002', 'BRG001'] Kontainer 2: ['BRG006', 'BRG004'] Kontainer 3: ['BRG010', 'BRG003'] Kontainer 4: ['BRG009', 'BRG005', 'BRG007'] Kontainer 5: ['BRG008']	Kontainer 1: ['BRG003', 'BRG009'] Kontainer 2: ['BRG001', 'BRG006'] Kontainer 3: ['BRG008', 'BRG004'] Kontainer 4: ['BRG010', 'BRG005', 'BRG007'] Kontainer 5: ['BRG002']	0.0008 detik	9
3	29475 → 16875	Kontainer 1: ['BRG011', 'BRG002', 'BRG001', 'BRG015', 'BRG007', 'BRG006', 'BRG008'] Kontainer 2: ['BRG004', 'BRG013', 'BRG014', 'BRG012', 'BRG003'] Kontainer 3: ['BRG009', 'BRG005', 'BRG010']	Kontainer 1: ['BRG007', 'BRG003', 'BRG001', 'BRG009', 'BRG004', 'BRG006', 'BRG008'] Kontainer 2: ['BRG005', 'BRG010', 'BRG011', 'BRG012', 'BRG002'] Kontainer 3: ['BRG015', 'BRG014', 'BRG013']	0.0012 detik	11



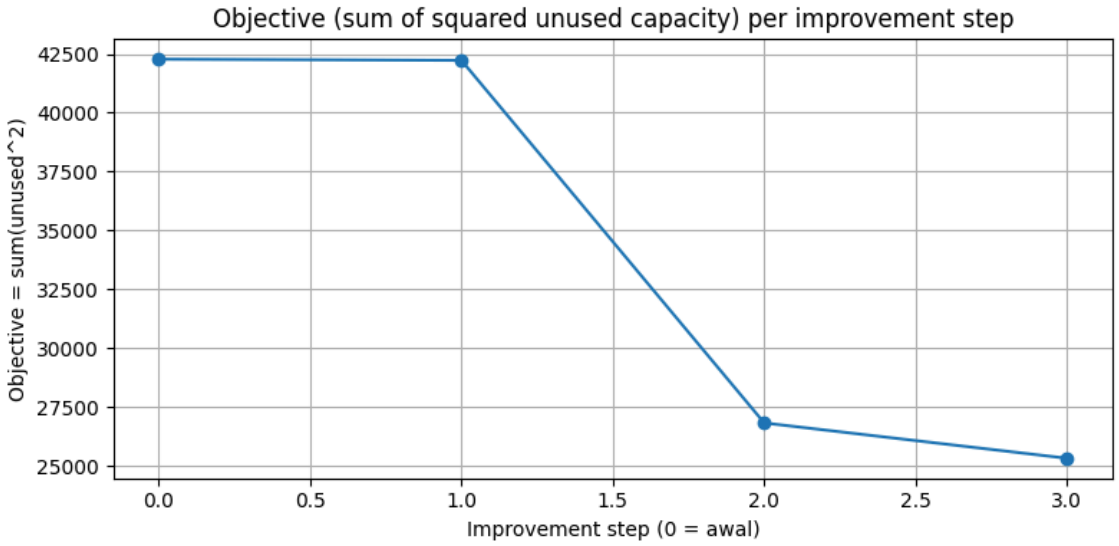
Gambar 1. *Objective Value Run* Pertama (Case4)



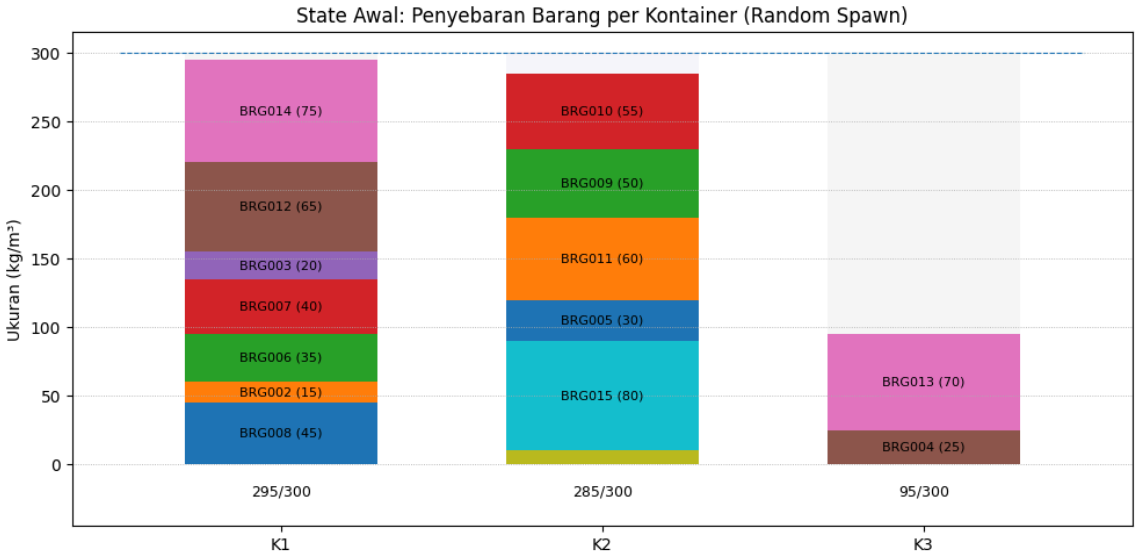
Gambar 2. *State* Awal *Run* Pertama (Case4)



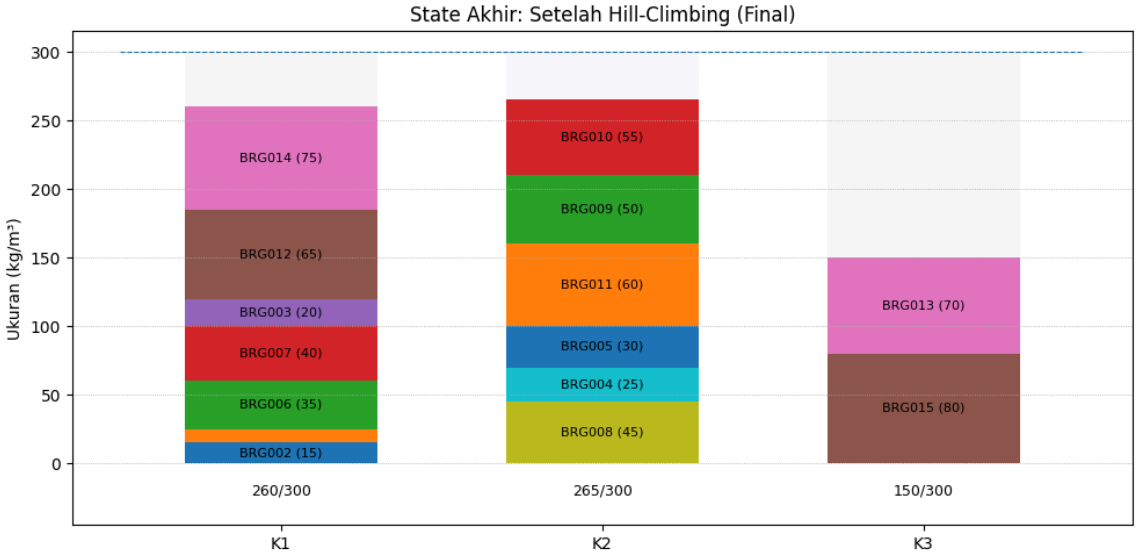
Gambar 3. State Akhir Run Pertama (Case4)



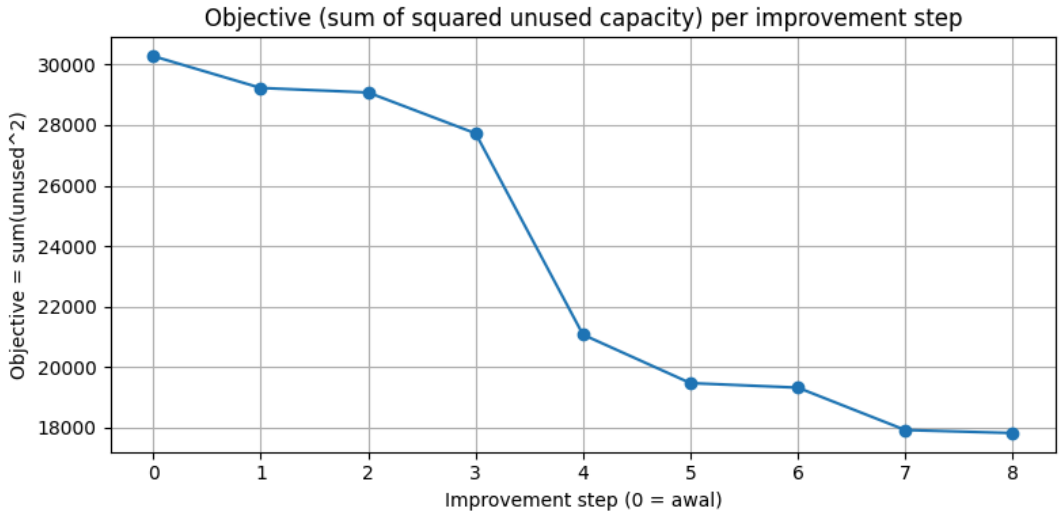
Gambar 4. Objective Value Run Kedua (Case4)



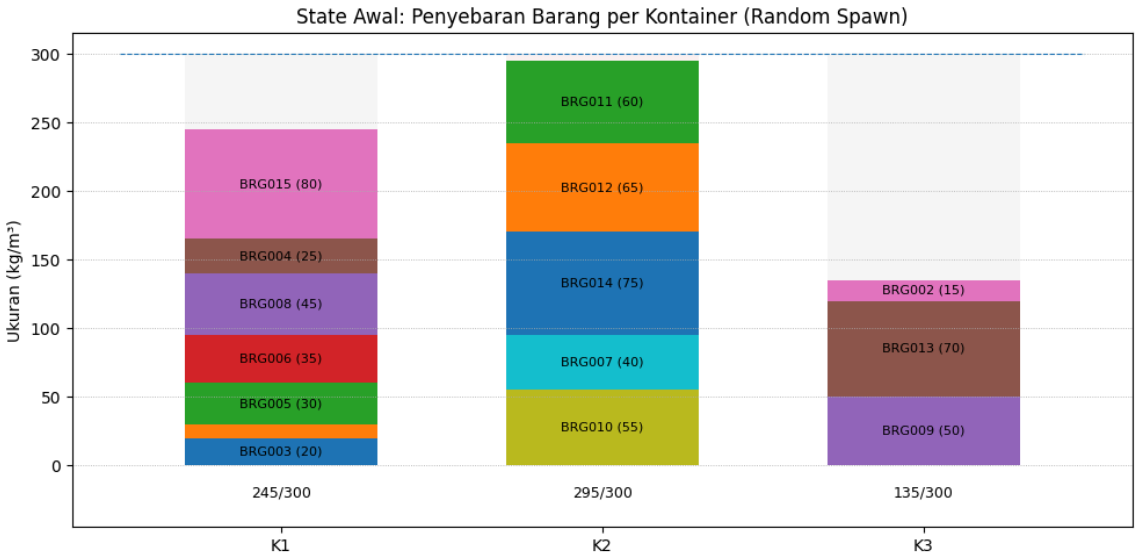
Gambar 5. State Awal Run Kedua (Case4)



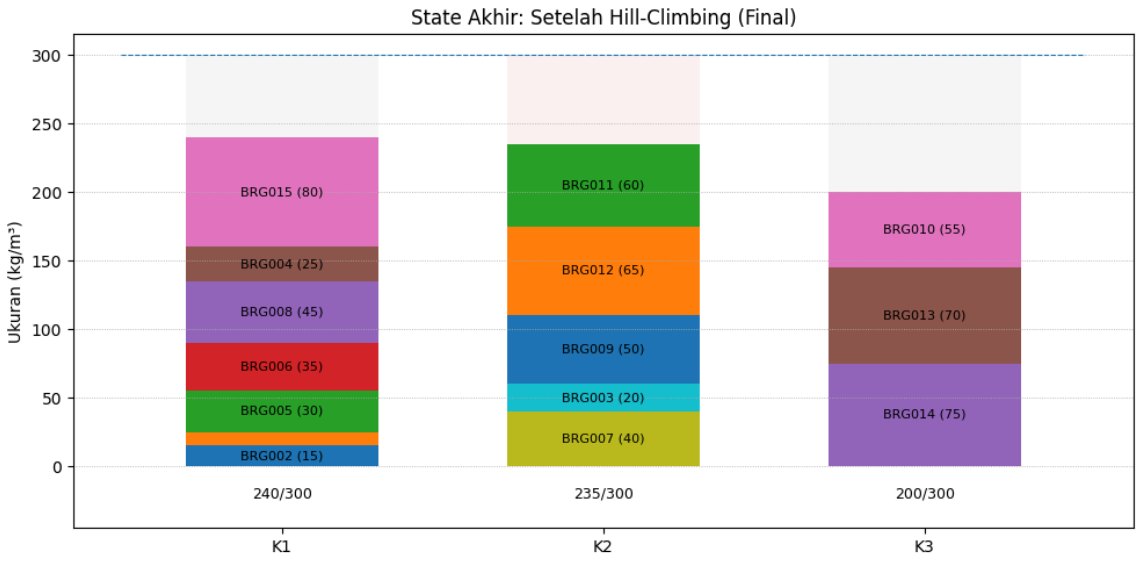
Gambar 6. *State* Akhir *Run* Kedua (Case4)



Gambar 7. Objective Value Run Ketiga (Case4)



Gambar 8. State Awal Run Ketiga (Case4)



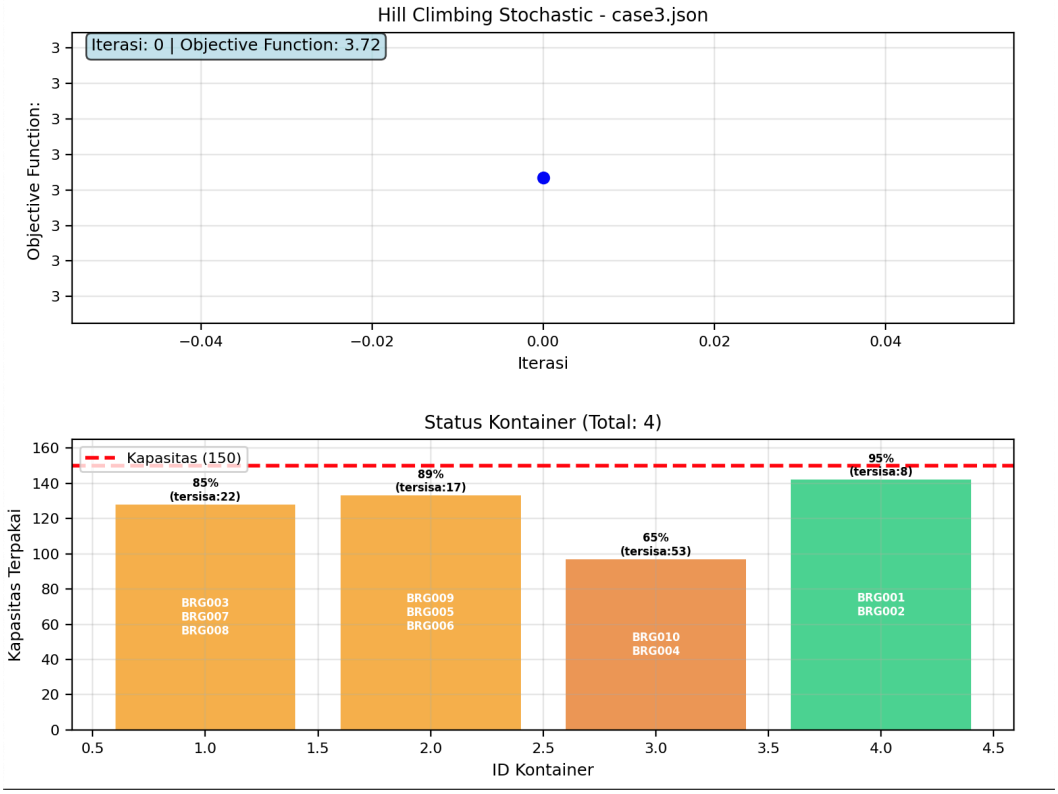
Gambar 9. State Akhir Run Ketiga (Case4)

2.3.2 *Stochastic Hill-Climbing*

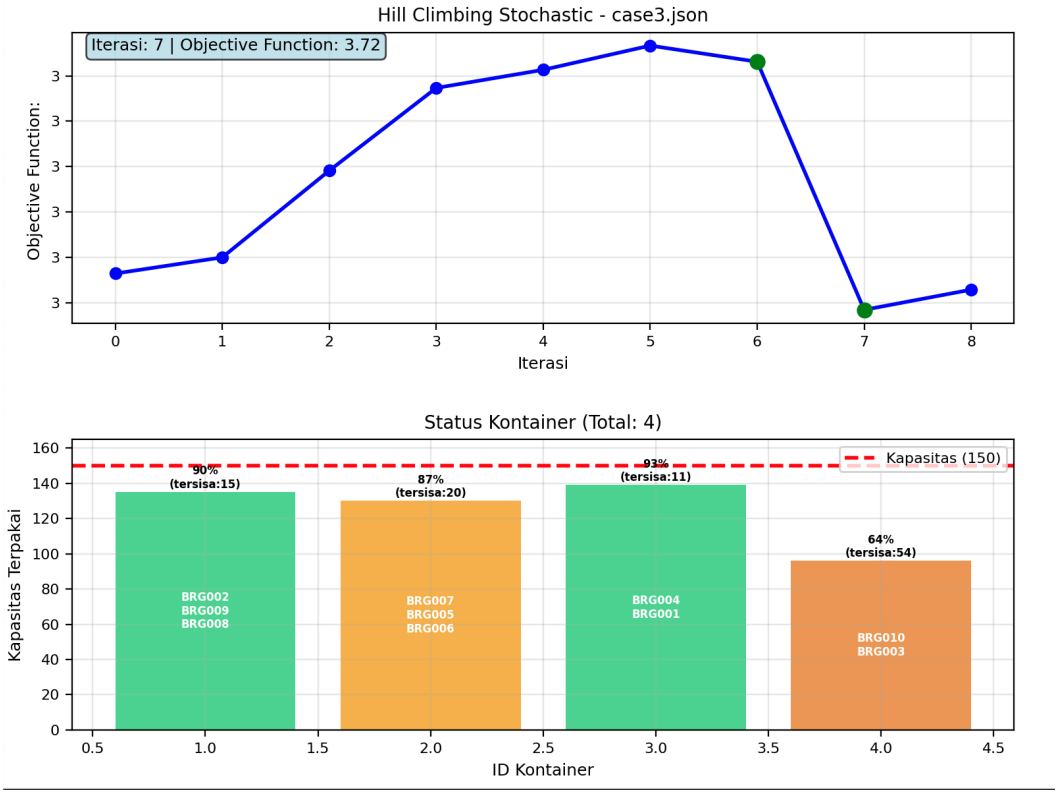
Tabel 2. Hasil Eksperimen *Stochastic Hill-Climbing*

Run	*Final Objective Value*	*State* Awal	*State* Akhir	Durasi Pencarian	Iterasi hingga berhenti
1	15500 → 9550	Kontainer 1: ['BRG002', 'BRG001'] Kontainer 2: ['BRG006', 'BRG004', 'BRG005'] Kontainer 3: ['BRG010', 'BRG003'] Kontainer 4: ['BRG008'] Kontainer 5: ['BRG007'] Kontainer 6: ['BRG009']	Kontainer 1: ['BRG003', 'BRG004'] Kontainer 2: ['BRG009', 'BRG001', 'BRG008'] Kontainer 3: ['BRG006', 'BRG010'] Kontainer 4: ['BRG002'] Kontainer 5: ['BRG007'] Kontainer 6: ['BRG005']	0.0006 detik	7
2	20066 → 14068	Kontainer 1: ['BRG002', 'BRG001'] Kontainer 2: ['BRG006', 'BRG004'] Kontainer 3: ['BRG010', 'BRG003'] Kontainer 4: ['BRG009', 'BRG005', 'BRG007'] Kontainer 5: ['BRG008']	Kontainer 1: ['BRG003', 'BRG009'] Kontainer 2: ['BRG001', 'BRG006'] Kontainer 3: ['BRG008', 'BRG004'] Kontainer 4: ['BRG010', 'BRG005', 'BRG007'] Kontainer 5: ['BRG002']	0.0008 detik	9
3	29475 → 16875	Kontainer 1: ['BRG011', 'BRG002', 'BRG001', 'BRG015', 'BRG007', 'BRG006', 'BRG008'] Kontainer 2: ['BRG004', 'BRG013', 'BRG014', 'BRG012', 'BRG003'] Kontainer 3: ['BRG009', 'BRG005',	Kontainer 1: ['BRG007', 'BRG003', 'BRG001', 'BRG009', 'BRG004', 'BRG006', 'BRG008'] Kontainer 2: ['BRG005', 'BRG010', 'BRG011', 'BRG012', 'BRG002'] Kontainer 3: ['BRG015', 'BRG014',	0.0012 detik	11

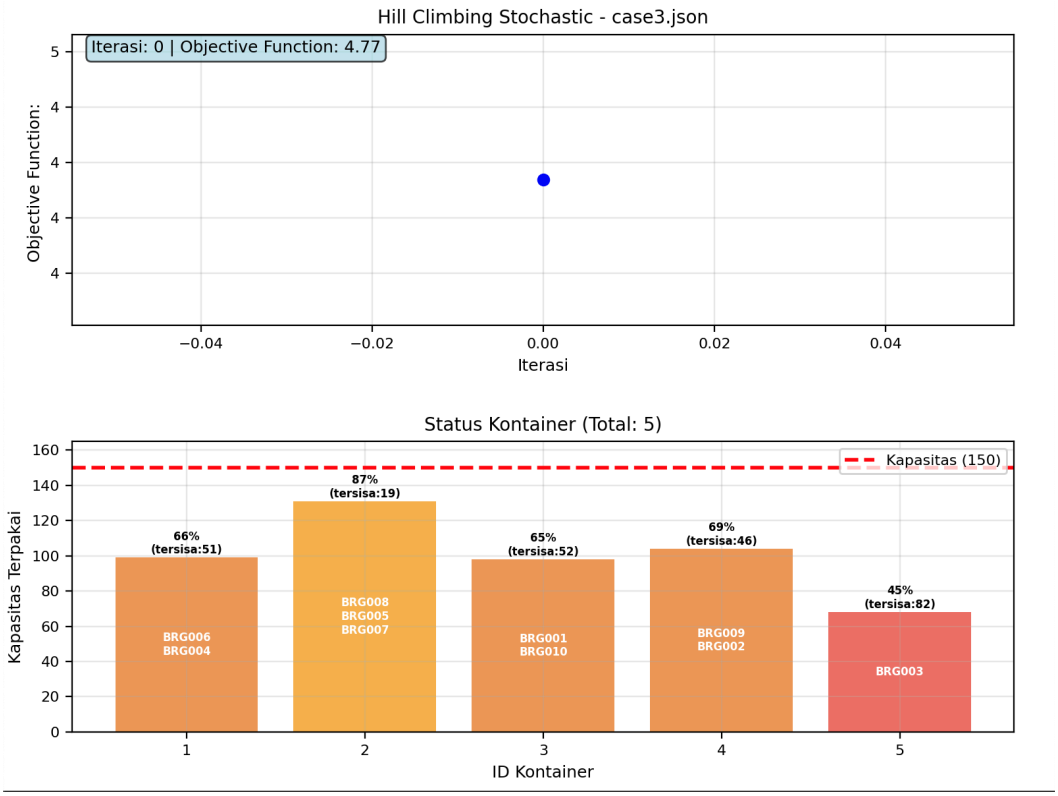
		'BRG010']	'BRG013']		



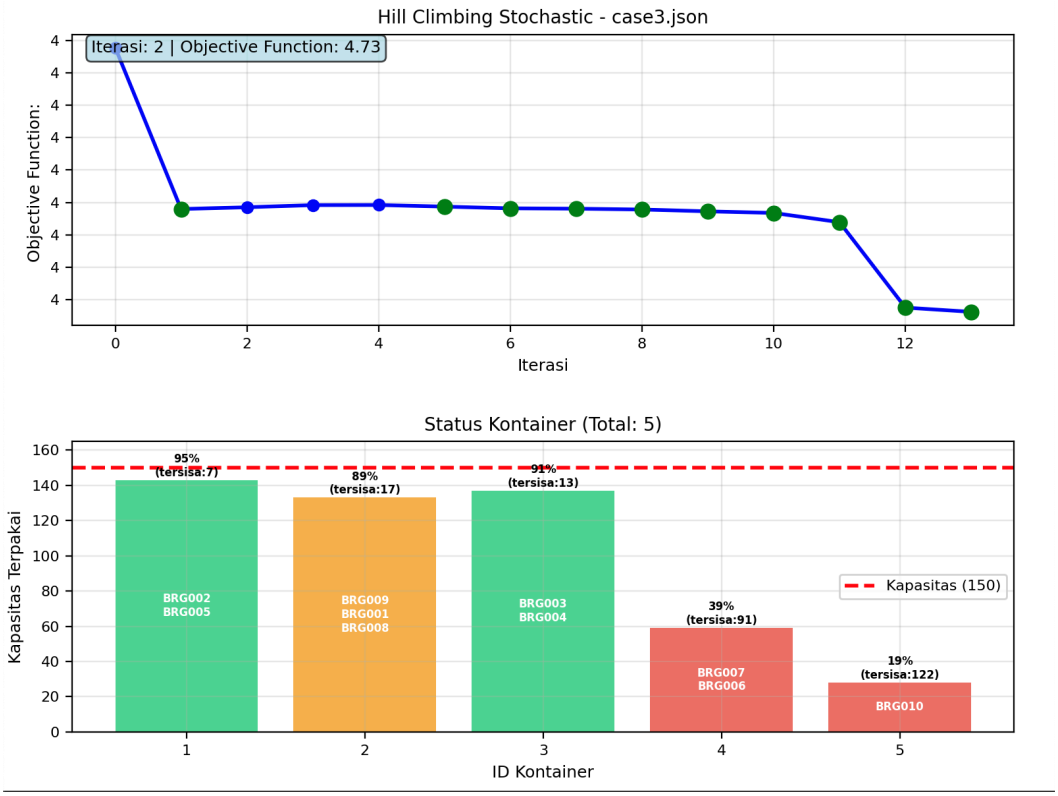
Gambar 10. *Objective Value* dan *State* Awal *Run* Pertama



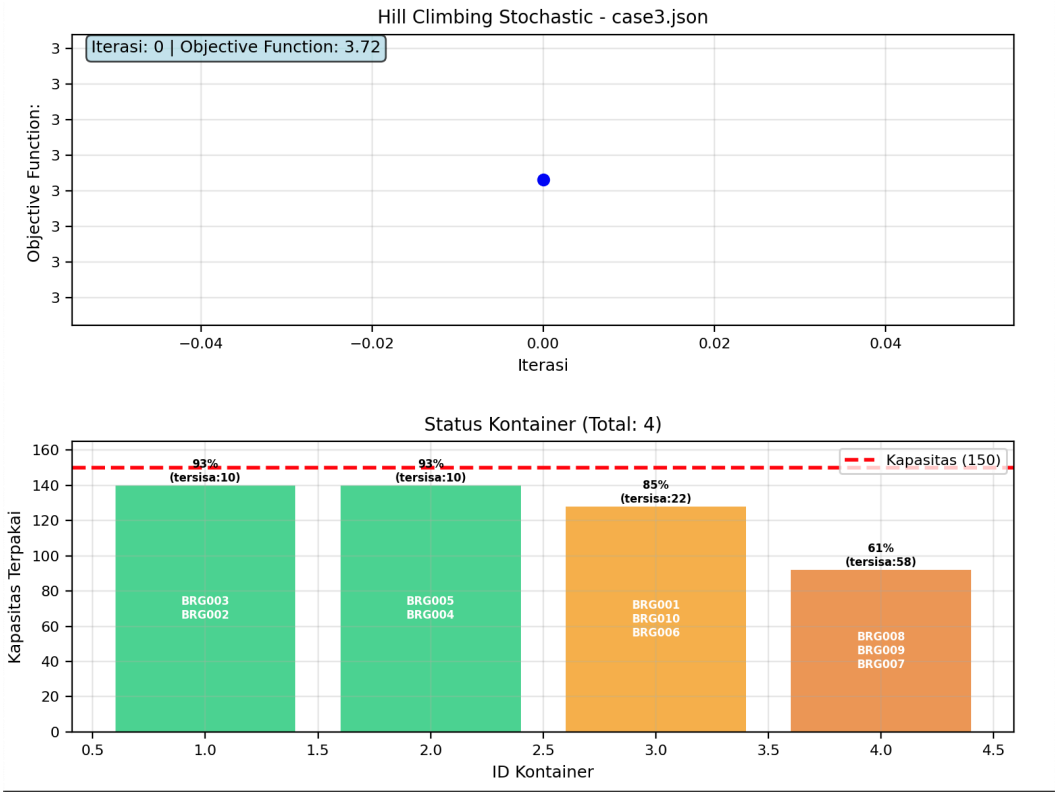
Gambar 11. *Objective Value* dan *State* Akhir *Run* Pertama



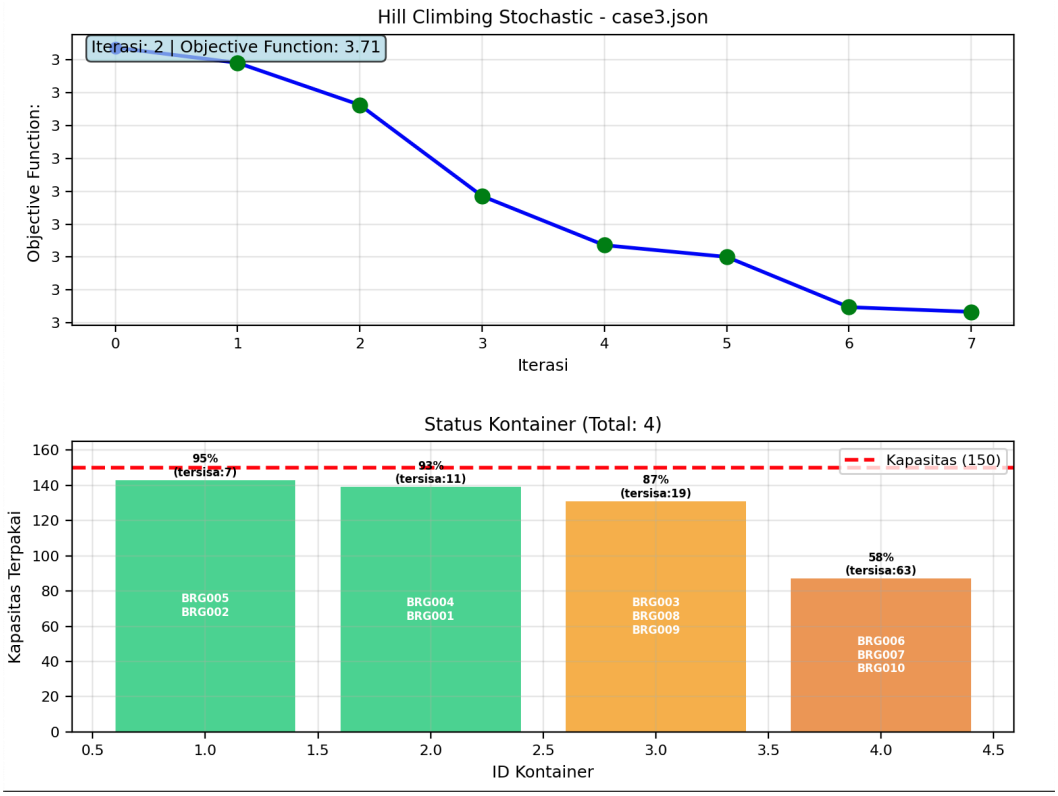
Gambar 12. Objective Value dan State Awal Run Kedua



Gambar 13. Objective Value dan State Akhir Run Kedua



Gambar 14. Objective Value dan State Awal Run Ketiga



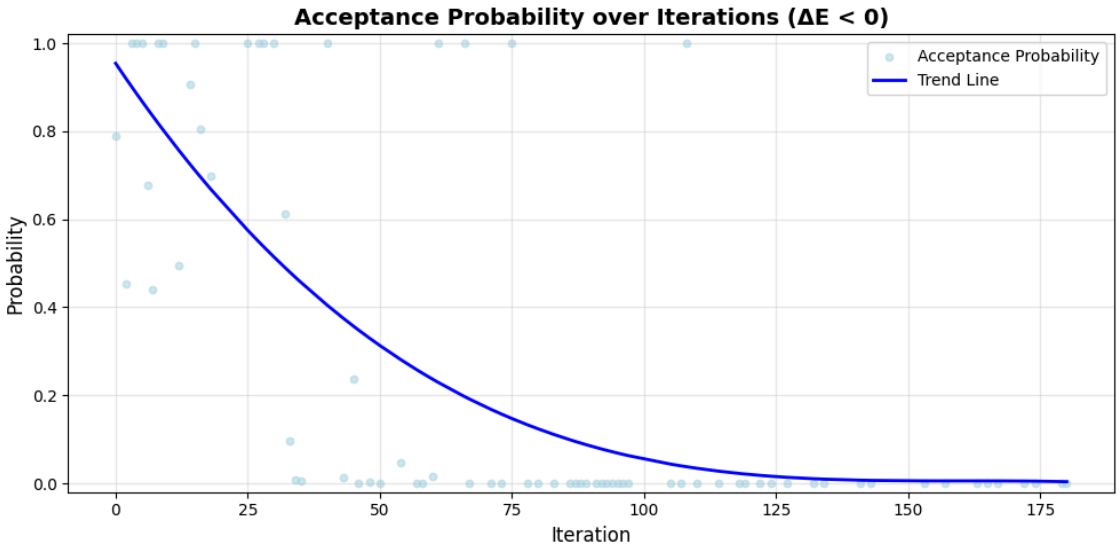
Gambar 15. Objective Value dan State Akhir Run Ketiga

2.3.3 *Simulated Annealing*

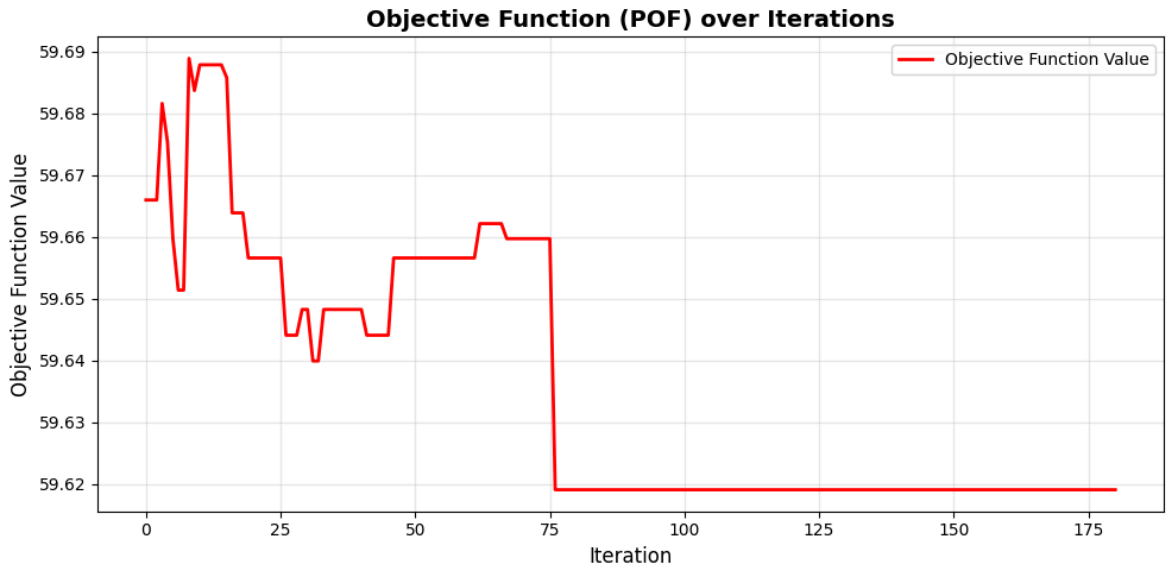
Tabel 3. Hasil Eksperimen *Simulated Annealing*

Run	*Final Objective Value*	*State* Awal	*State* Akhir	Durasi Pencarian	Iterasi hingga berhenti	Frekuensi "Stuck"
1	59.66 → 59.62	Kontainer 1: ['BRG008'] Kontainer 2: ['BRG002'] Kontainer 3: ['BRG004', 'BRG001', 'BRG005'] Kontainer 4: ['BRG009', 'BRG003'] Kontainer 5: ['BRG006', 'BRG007'] Kontainer 6: ['BRG010']	Kontainer 1: ['BRG004'] Kontainer 2: ['BRG008'] Kontainer 3: ['BRG010', 'BRG001', 'BRG003'] Kontainer 4: ['BRG007', 'BRG006'] Kontainer 5: ['BRG005', 'BRG009'] Kontainer 6: ['BRG002']	0.008 detik	181	26
2	59.68 → 59.64	Kontainer 1: ['BRG002', 'BRG001'] Kontainer 2: ['BRG006', 'BRG004']	Kontainer 1: ['BRG003', 'BRG009'] Kontainer 2: ['BRG001', 'BRG006'] Kontainer 3: ['BRG008',	0.0112 detik	178	26

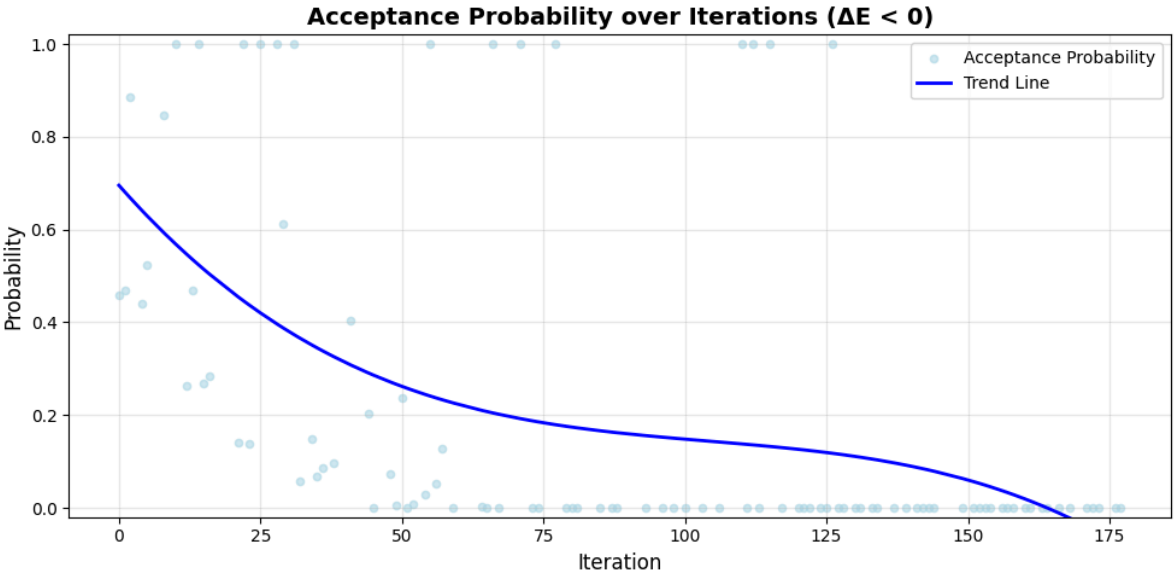
		Kontainer 3: ['BRG010', 'BRG003'] Kontainer 4: ['BRG009', 'BRG005', 'BRG007'] Kontainer 5: ['BRG008']	'BRG004'] Kontainer 4: ['BRG010', 'BRG005'] Kontainer 5: ['BRG007'] Kontainer 6: ['BRG002']			
3	49.65 → 49.63	Kontainer 1: ['BRG002'] Kontainer 2: ['BRG006', 'BRG001', 'BRG010'] Kontainer 3: ['BRG005', 'BRG009'] Kontainer 4: ['BRG008', 'BRG003', 'BRG004'] Kontainer 5: ['BRG007']	Kontainer 1: ['BRG010'] Kontainer 2: ['BRG007', 'BRG001', 'BRG004'] Kontainer 3: ['BRG005', 'BRG008'] Kontainer 4: ['BRG009', 'BRG003', 'BRG006'] Kontainer 5: ['BRG002']	0.006 detik	181	30



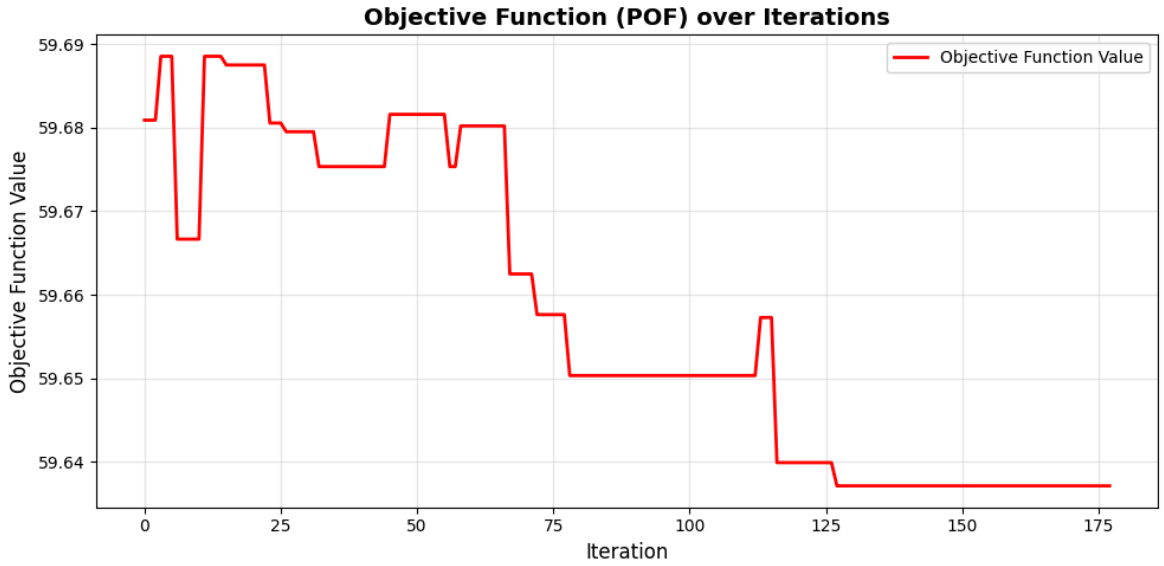
Gambar 16. *Acceptance Probability over Iterations* Pertama



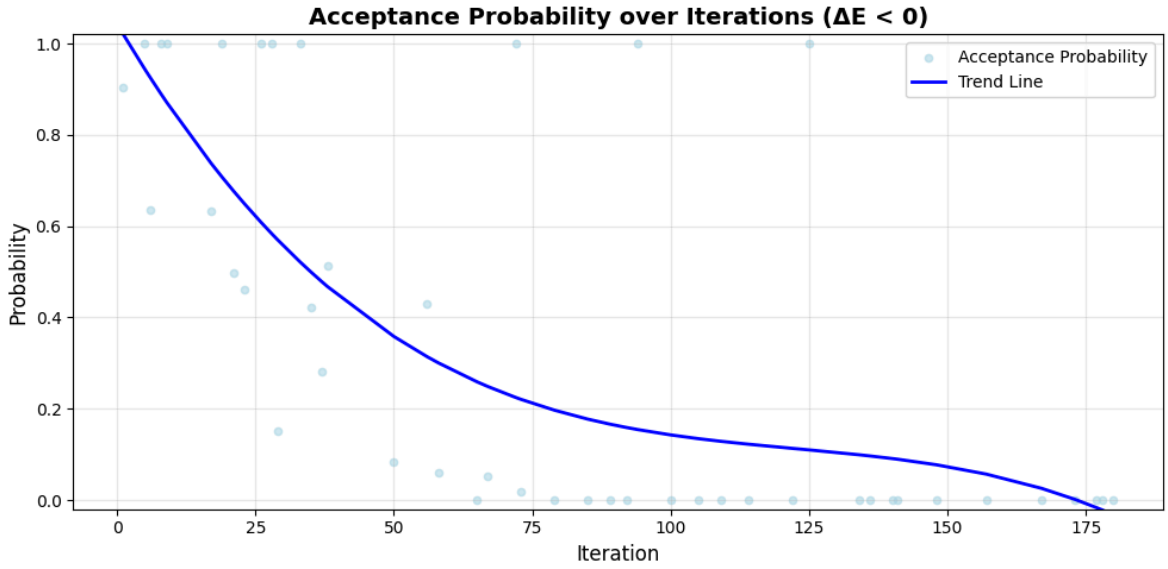
Gambar 17. *Objective Value* dan *State* Akhir *Run* Pertama



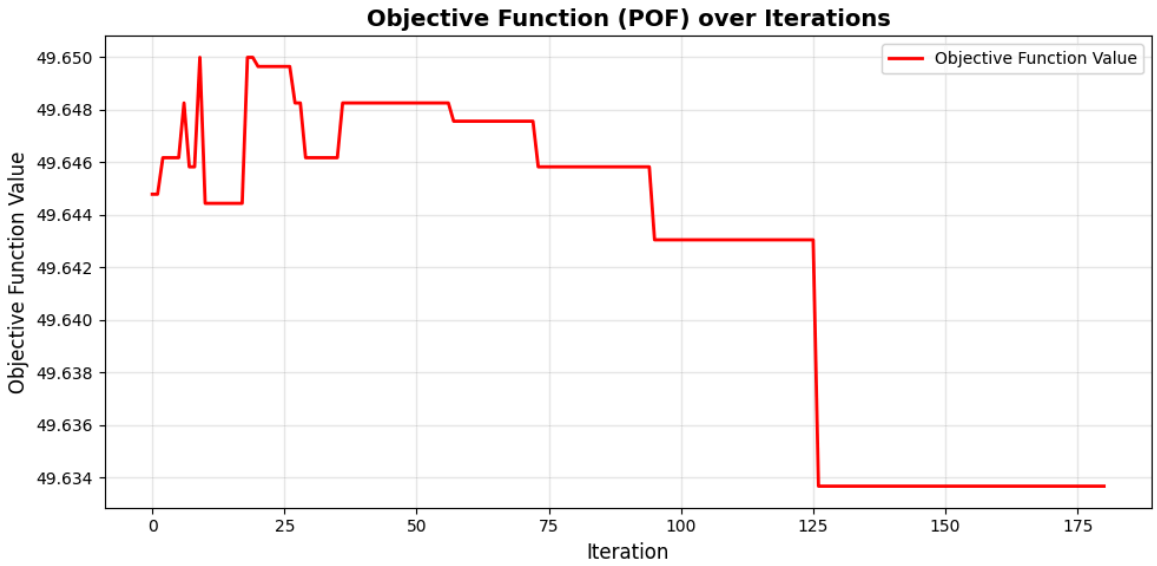
Gambar 18. *Acceptance Probability over Iterations* Kedua



Gambar 19. *Objective Function over Iterations* Kedua



Gambar 20. *Acceptance Probability over Iterations* Ketiga



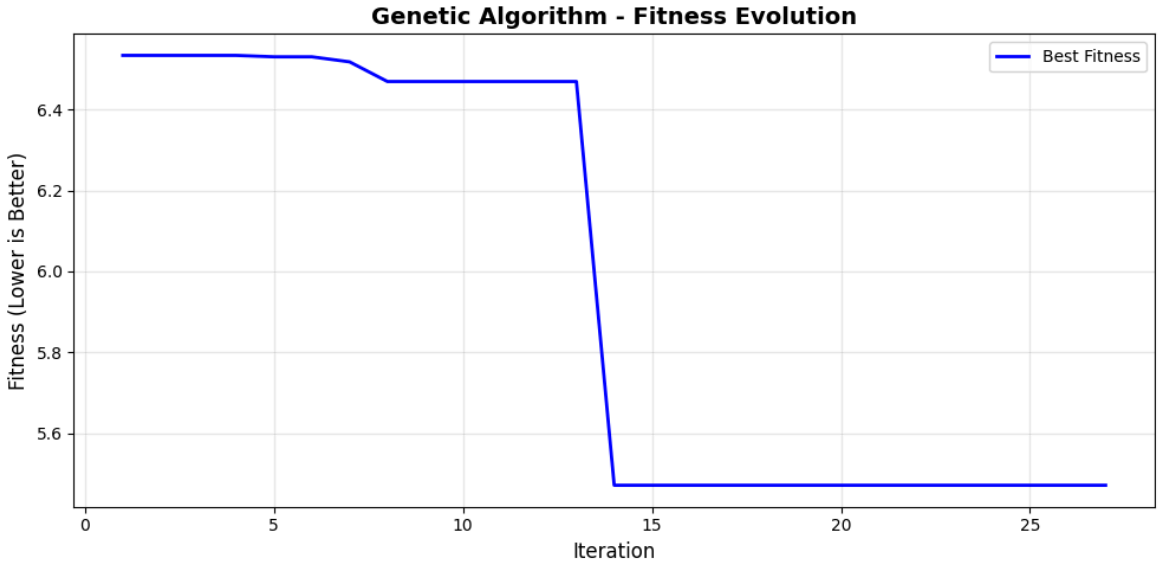
Gambar 21. *Objective Function over Iterations* Ketiga

2.3.4 *Genetic Algorithm*

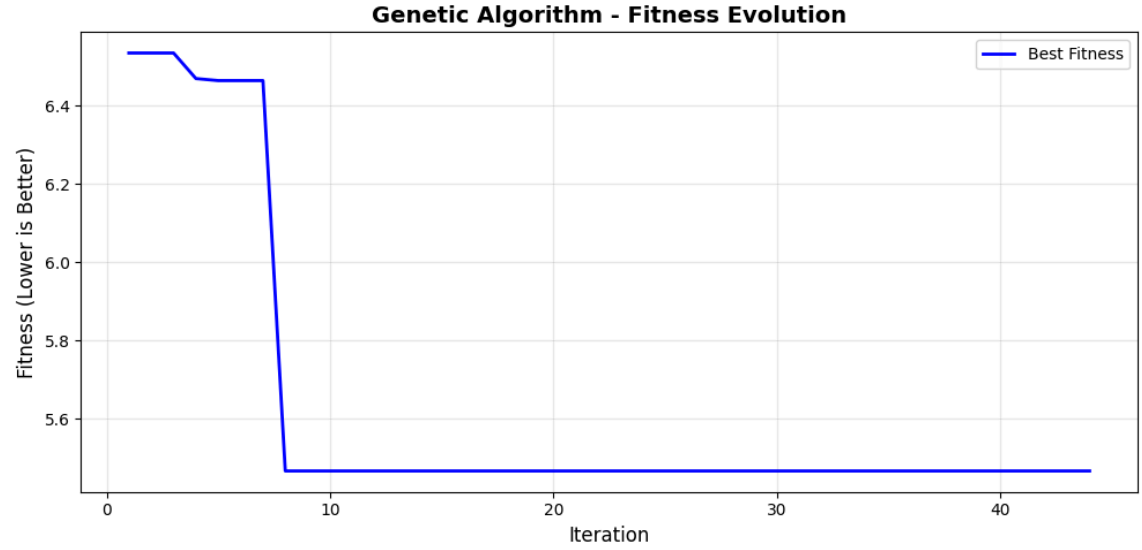
Tabel 4. Hasil Eksperimen *Genetic Algorithm*

Run	Mutate	*Final Objective Value*	*State* Awal	*State* Akhir	Durasi Pencarian	Iterasi hingga berhenti	Jumlah Populasi
***Controlled Population* (*Population* = 8)**							
1	0.05	7.59 → 5.47	[2, 1, 5, 1, 6, 3, 4, 1, 2, 3, 8, 5, 4, 6, 7]	[2, 5, 5, 3, 5, 6, 4, 1, 2, 5, 2, 1, 4, 6, 3]	0.014	33	**8**
	0.2	6.54 → 5.47	[5, 7, 7, 6, 7, 3, 2, 7, 5, 5, 3, 4, 2, 6, 1]	[2, 3, 4, 4, 3, 1, 2, 6, 5, 5, 6, 4, 2, 3, 1]	0.013	27	
2	0.05	6.54 → 5.46	[4, 7, 6, 1, 5, 3, 2, 1, 1, 4, 3, 2, 5, 7, 6]	[4, 4, 5, 1, 5, 4, 2, 1, 1, 3, 4, 3, 5, 6, 2]	0.017	44	
	0.2	7.58 → 5.47	[8, 2, 6, 1, 2, 1, 2, 3, 7, 1, 7, 4, 6, 8, 5]	[2, 2, 6, 1, 2, 1, 3, 5, 4, 1, 4, 2, 6, 5, 3]	0.019	37	
3	0.05	6.53 → 5.46	[6, 5, 4, 6, 2, 6, 6, 1, 3, 5, 4, 2, 3, 1, 7]	[5, 5, 2, 6, 2, 6, 1, 4, 3, 5, 6, 2, 3, 4, 1]	0.0201	51	
	0.2	6.53 → 5.46	[4, 2, 2, 2, 3, 5, 4, 2, 1, 1, 4, 6, 7, 3, 5]	[4, 1, 3, 2, 4, 3, 2, 1, 6, 2, 1, 3, 6, 5, 4]	0.02	42	
***Controlled Iterations* (*Iterations* = 50)**							
1	0.05	8.6 → 5.46	[6, 7, 4, 6, 5, 5, 1, 4, 7, 1, 6, 3, 8, 2, 9]	[3, 1, 6, 1, 1, 4, 3, 5, 1, 2, 6, 2, 3, 5, 4]	0.2881	50	100
	0.2	7.58 → 5.47	[2, 4, 8, 2, 3, 2, 4, 8, 7, 4, 5, 3, 7, 1, 6]	[4, 3, 6, 3, 6, 1, 1, 1, 2, 5, 6, 5, 2, 4, 3]	0.3049		98
2	0.05	5.47 → 5.46	[3, 6, 1, 3, 1, 6, 4, 2, 5, 1, 5, 6, 4, 2, 3]	[4, 3, 5, 3, 1, 1, 4, 6, 1, 2, 5, 2, 4, 6, 3]	0.244		95

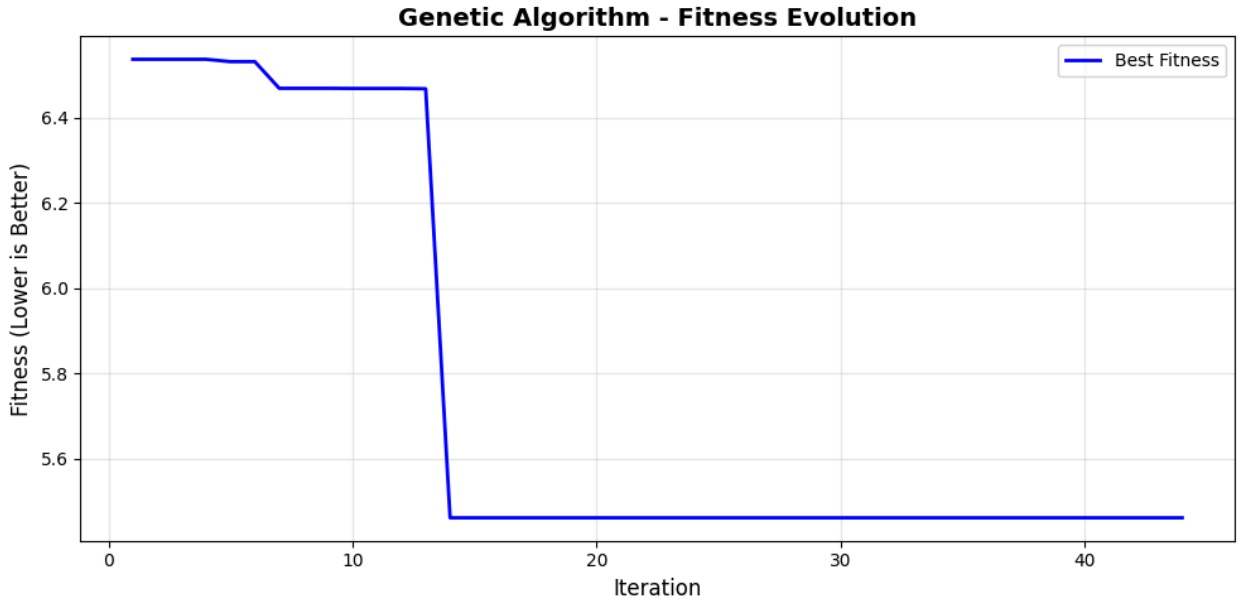
	0.2	7.56 → 5.46	[2, 5, 4, 4, 8, 2, 5, 5, 7, 2, 4, 1, 7, 6, 3]	[1, 1, 5, 4, 4, 1, 5, 6, 3, 5, 1, 4, 3, 6, 2]	0.277		90
3	0.05	6.52 → 5.46	[3, 4, 1, 3, 6, 4, 2, 1, 1, 7, 6, 4, 5, 3, 2]	[6, 3, 6, 2, 6, 3, 4, 5, 2, 1, 6, 1, 3, 5, 4]	0.228		89
	0.2	6.53 → 5.46	[7, 2, 4, 1, 1, 6, 7, 2, 1, 2, 5, 6, 7, 4, 3]	[1, 5, 5, 2, 1, 2, 5, 5, 3, 4, 2, 4, 3, 6, 1]	0.304		97



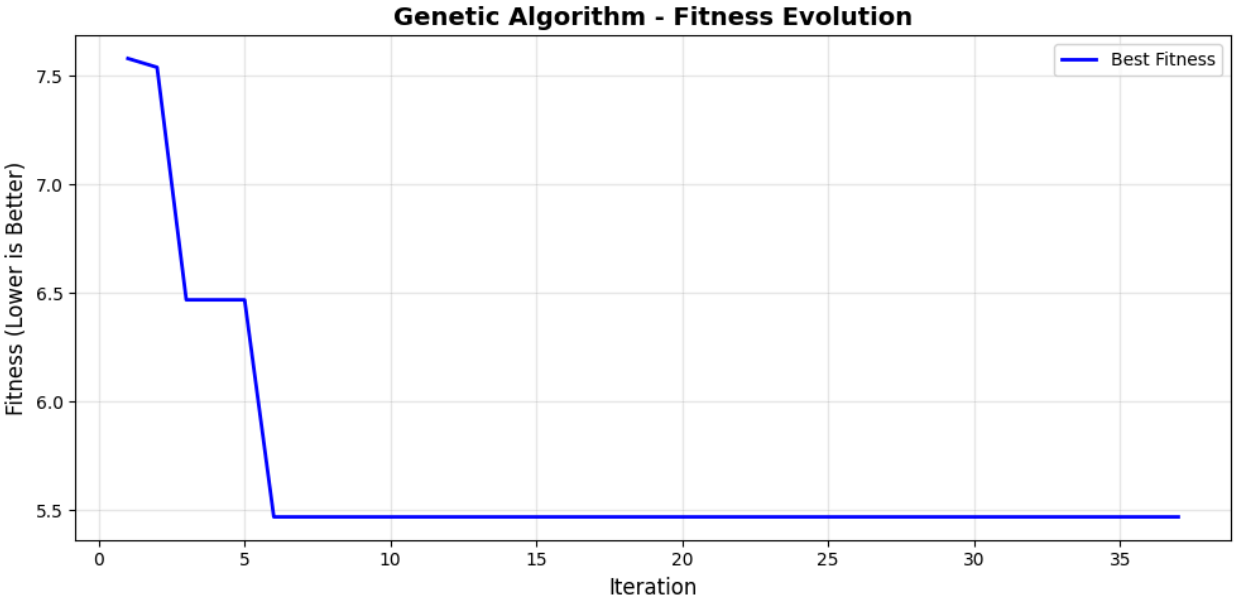
Gambar 22. *Objective Function (Fitness) over Iteration C.Population* Pertama (0.05)



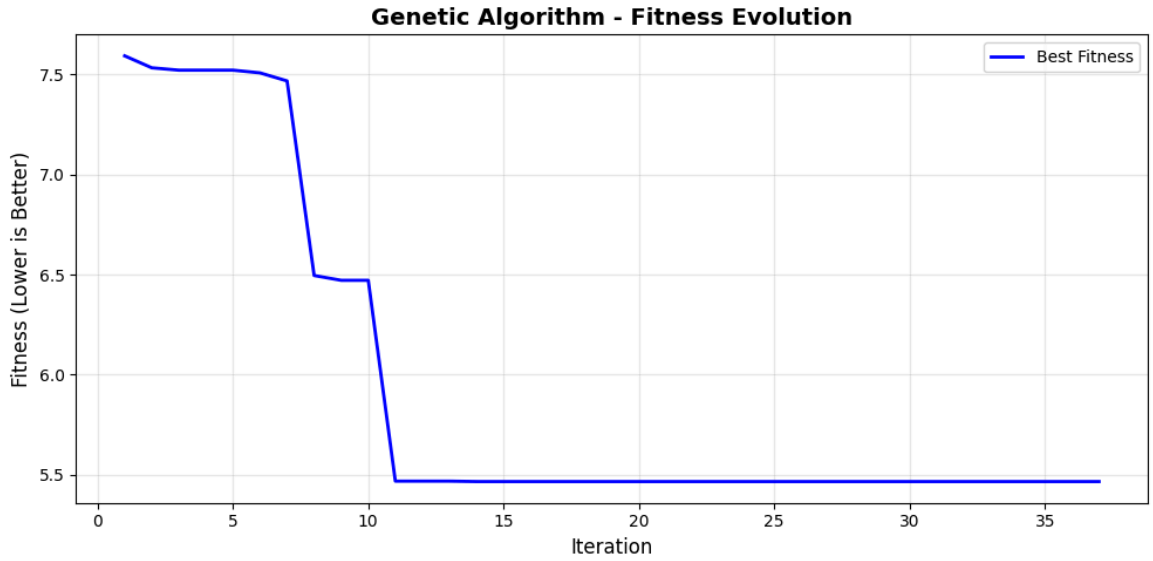
Gambar 23. *Objective Function (Fitness) over Iteration C.Population* Pertama (0.2)



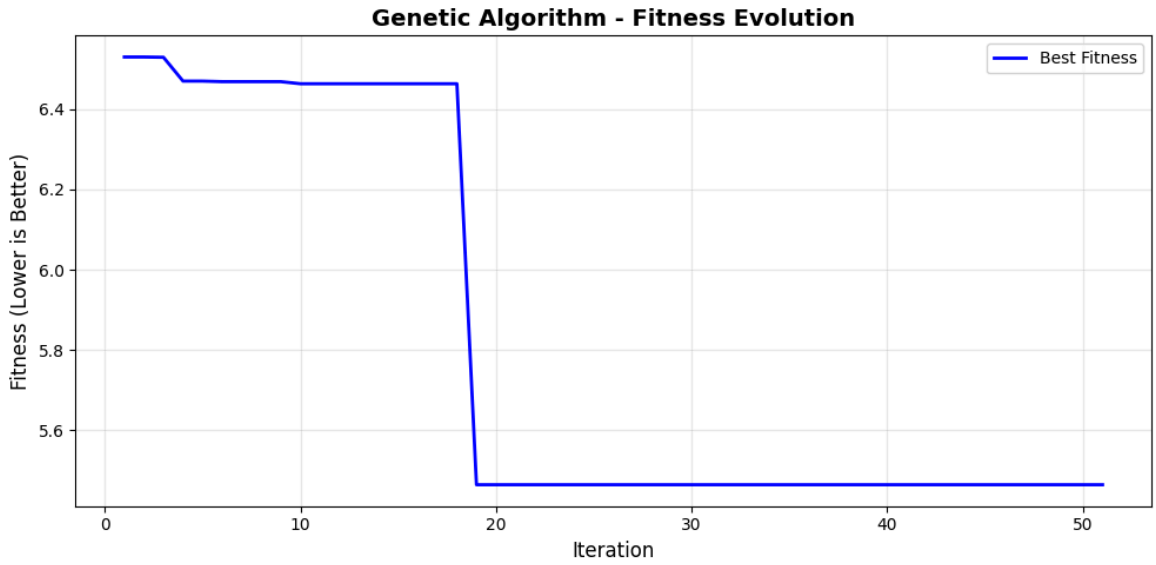
Gambar 24. *Objective Function (Fitness) over Iteration C.Population* Kedua (0.05)



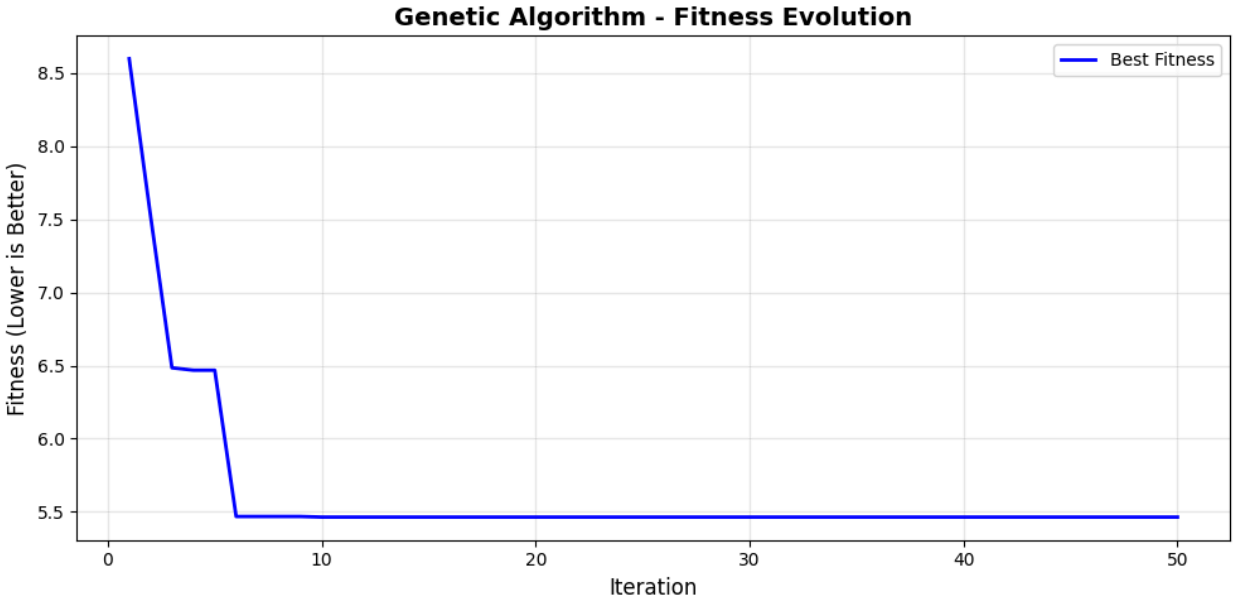
Gambar 25. *Objective Function (Fitness) over Iteration C.Population* Kedua (0.2)



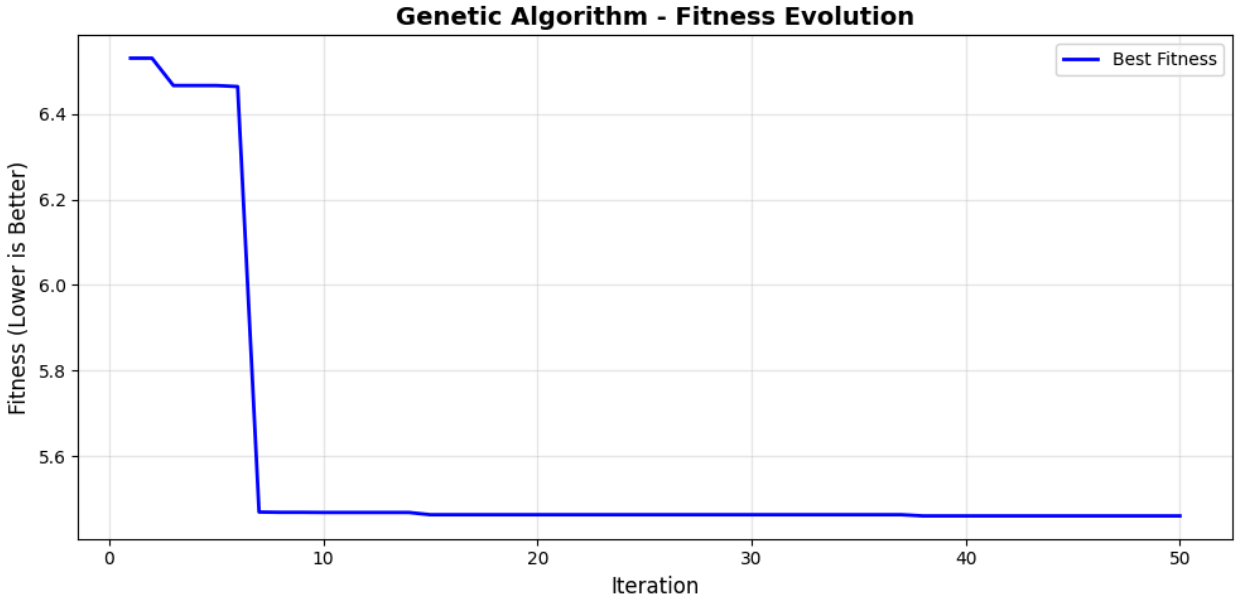
Gambar 26. *Objective Function (Fitness) over Iteration C.Population* Ketiga (0.05)



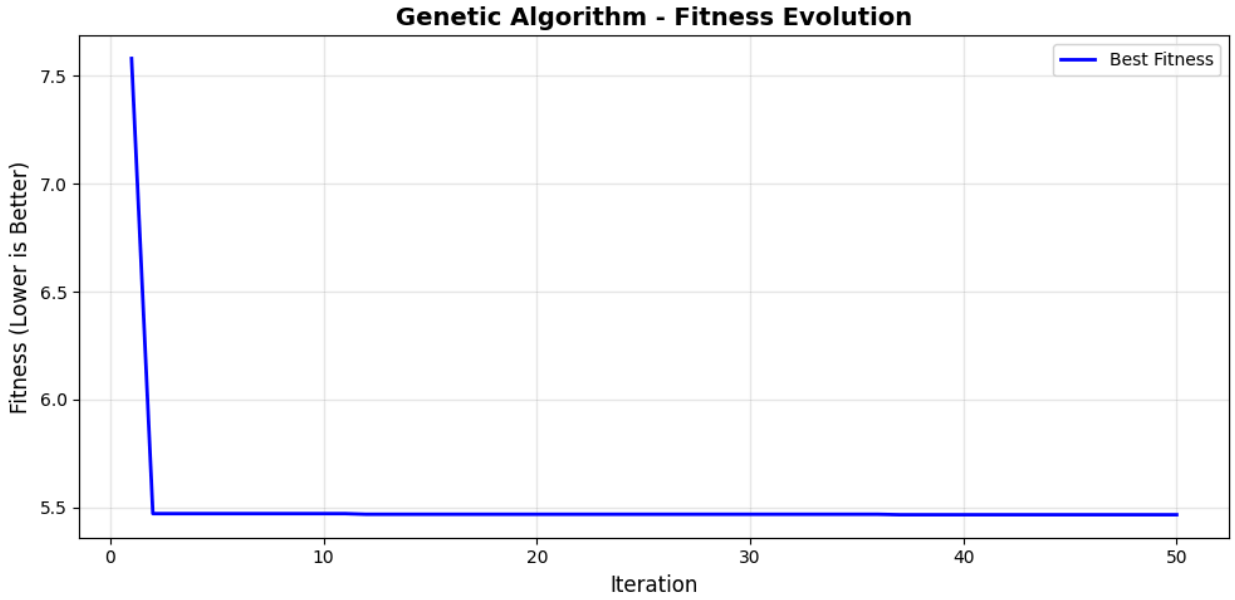
Gambar 27. *Objective Function (Fitness) over Iteration C.Population* Ketiga (0.2)



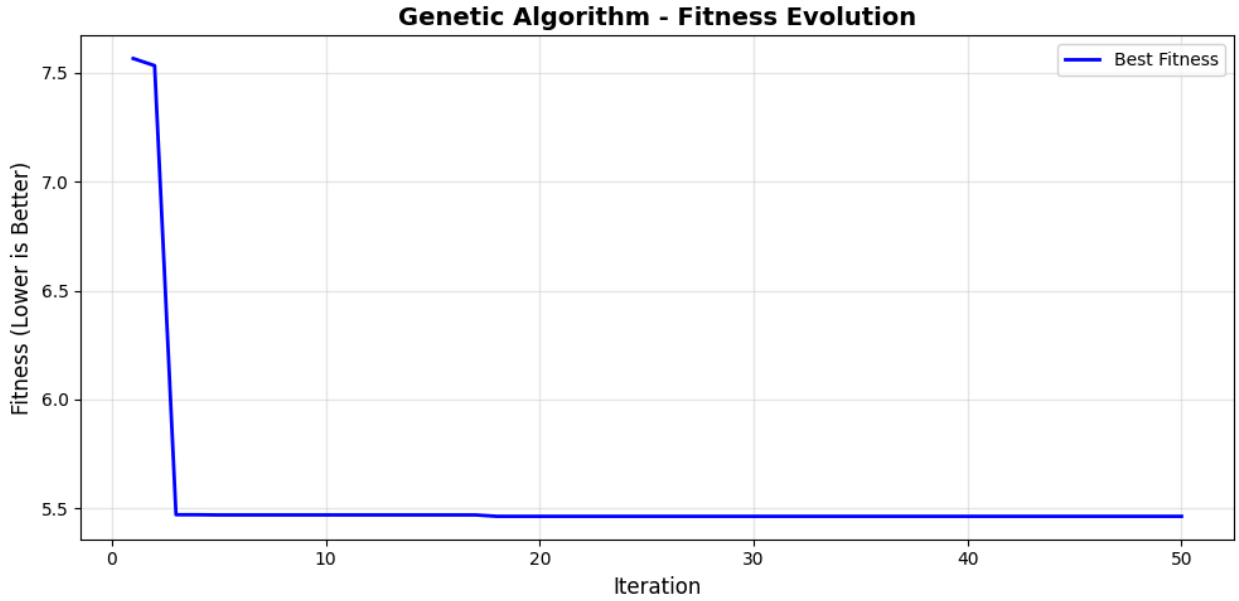
Gambar 28. *Objective Function (Fitness) over Iteration C.Iteration* Pertama (0.05)



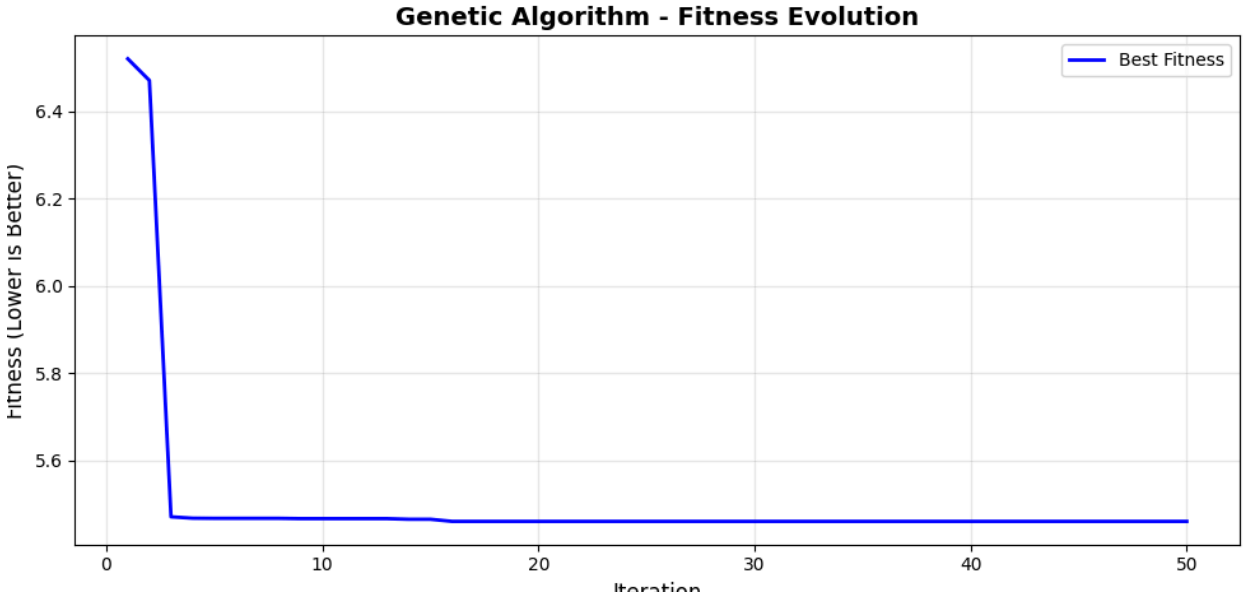
Gambar 29. *Objective Function (Fitness) over Iteration C.Iteration* Pertama (0.2)



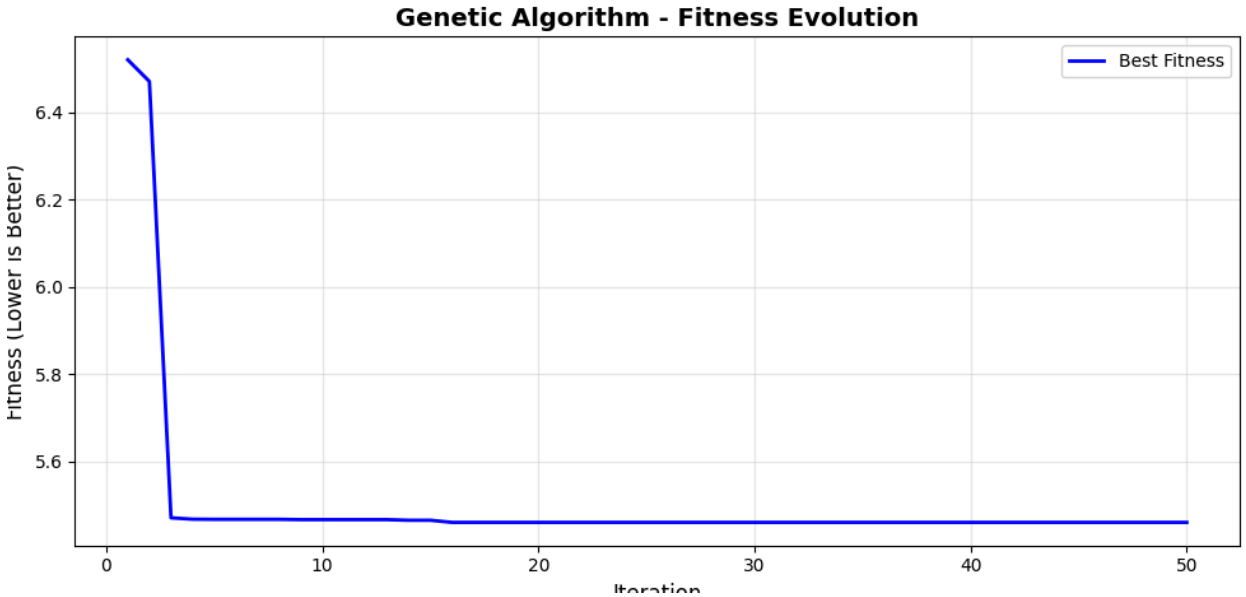
Gambar 30. *Objective Function (Fitness) over Iteration C.Iteration* Kedua (0.05)



Gambar 31. *Objective Function (Fitness) over Iteration C.Iteration* Kedua (0.2)



Gambar 32. *Objective Function (Fitness) over Iteration C.Iteration* Ketiga (0.05)



Gambar 33. *Objective Function (Fitness) over Iteration C.Iteration* Ketiga (0.2)

2.3.5 Analisis

Setiap algoritma mendekati *global maximum* dengan cara yang berbeda-beda. Algoritma *Hill Climbing Steepest Ascent* selalu memilih tetangga yang *state*-nya lebih baik. Tetangga yang lebih baik artinya selalu menurunkan nilai *objective function*. Dalam

konteks *bin packing problems* ini, algoritma hanya akan melakukan *swap* terhadap barang yang memaksimalkan penggunaan kapasitas kontainer. Penentuan *initial state* (penempatan barang secara acak di awal) sangat menentukan hasil akhir algoritma ini karena tidak bisa melakukan eksplorasi, seperti algoritma lain. Setiap kali menemukan tetangga yang lebih buruk, algoritma ini akan langsung berhenti. Oleh karena itu, algoritma ini cenderung terjebak di *local maximum*. Algoritma *Hill Climbing Stochastic* memilih tetangga secara acak. Artinya algortima bisa melakukan *swap* terhadap pasangan barang yang tidak menurunkan nilai *objective function*. Keuntungan dari mekanisme ini adalah algoritma masih bisa keluar dari *local maximum*, tetapi hanya kemungkinan kecil saja. Algoritma lain, seperti *Simulated Annealing* dan *Genetic*, sangat dekat dengan *global maximum*. Alasan utamanya adalah algoritma ini masih melakukan *swap* terhadap pasangan yang lebih "buruk" dengan penilaian tertentu. Algoritma *Simulated Annealing* menggunakan probabilitas yang bergantung pada variabel `temperature` dan algoritma *Genetic* menggunakan proses *selection*, *crossover*, dan *mutation*. Dengan demikian, algoritma yang paling dekat dengan *global maximum* adalah *genetic, simulated annealing, stochastic hill climbing*, dan *steepest ascent hill climbing*.

Berdasarkan visualisasi yang sudah dibuat pada bagian 2.3.1 – 2.3.4, setiap algoritma juga membutuhkan waktu pemrosesan yang berbeda-beda. Algoritma *Steepest Hill Climbing* membutuhkan waktu pemrosesan yang paling sedikit karena setiap iterasi hanya melakukan *swap* terhadap barang yang menurunkan nilai *objective function*. Algoritma ini tidak membutuhkan eksplorasi yang luas, seperti Algoritma *Stochastic Hill Climbing*, *Simulated Annealing*, dan *Genetic.* Untuk mencari keseimbangan antara durasi pemrosesan dan kualitas solusi, algoritma *Simulated Annealing* sangat sesuai. Akan tetapi, solusi dari algoritma *Genetic* tetap yang terbaik. Walaupun demikian, *case* yang besar juga akan membuat durasi pemrosesan algoritma *genetic* meningkat drastis.

Algoritma juga akan memberikan solusi yang berbeda di setiap *run*-nya. Algoritma Genetic memiliki konsistensi solusi paling tinggi karena diversitas dari populasi awal. Selain itu, algoritma ini tidak terlalu bergantung dengan *Initial State*, berbeda halnya dengan Algoritma Steepest Hill Climbing yang sangat bergantung pada penempatan acak di penentuan solusi awal. Dalam konteks Algoritma Genetic, banyaknya iterasi akan membuat solusi akhir yang lebih baik. Selain itu, populasi yang lebih besar akan meningkatkan diversitas sehingga solusi semakin mendekati *global maximum*.

BAB III
KESIMPULAN DAN SARAN

3.1 Kesimpulan

Pada permasalahan diatas, dapat dilihat berbagai algoritma *local search* yang diimplementasikan untuk menyelesaikan permasalahan pengisian barang di dalam berbagai kontainer secara padat dan efektif. Pengembangan algoritma berupa *Hill-Climbing Steepest Ascent* dengan *constraint max iteration*, *Hill-Climbing Stochastic* dengan *constraint max iteration*, *Simulated Annealing* dengan *approach Geometric Cooling* untuk temperaturnya, dan menentukans secara final minimum temperatur untuk dianggap valid dalam algorimta, serta *Genetics Algorithm* yang menggunakan konsep sesuai dengan buku Artificial Intelligence: A Modern Approach. Seluruh algoritma memiliki *output* yang sesuai dan diharapkan oleh kami dengan pencatatan yang lengkap dan menyeluruh dari segi *state*, *objective function*, dan lainnya. Untuk keempat algoritma, dapat disarankan dua (2) utama dengan *Genetic Algorithm* dikarenakan cepat dalam jumlah data yang relatif sedikit. Jika ingin mengembangkan yang jauh leibh besar data, *Simulated Annealing* menjadi algoritma yang sesuai untuk kondisi tersebut menyesuaikan dengan temperatur awal yang dicari dengan kesesuaian data dari *input*, *alpha* yang dapt dimanipulasi secara teratur untuk mendapatkan akurasi dan efektivitas algoritma yang kuat untuk data yang besar, dan minimum temperatur yang dapat disesuaikan dengan efesientivitas algoritma dalam perhitungan di akhir.

Semua itu didasari dan didukung dengan fungsi objektif yang telah dijelaskan sebelumnya. Ini menjadi kunci dalam *progress* dan dasar penilaian dari antara algoritma untuk melihat performa dan waktu yang diperlukan oleh algoritma untuk menyelesaikan suatu *problem*.

3.2 Saran

Hill-Climbing dan Simulated Annealing masih berfokus pada menukar dua barang Sebagaimana didefinisikan dalam deskripsi persoalan, sistem akan memindahkan suatu barang ke kontainer lain, pun termasuk ke kontainer yang baru dibuat jika perlu. Hal tersebut masih perlu lebih diuji. Penambahan operasi yang dapat sistem lakukan akan memperluas ruang pencarian dan memperbaiki hasil dari fungsi objektif.

Optimasi dari Initial State dapat meningkatkan performa dari seluruh algoritma local search secara signifikan dengan memulai dari solusi yang lebih baik. Dapat dikembangkan dengan implementasi *Best Fit Decreasing* (BFD) ataupun *First-fit Decreasing* (FFD).

Melakukan eksperimen pada *dataset* yang berukuran sangat besar dan kompleks, hingga ribuan *test-case*, untuk menguji skalabilitas dan performa setiap algoritma untuk *menghandle dataset* yang besar.

BAB IV
PEMBAGIAN TUGAS

Tabel 5. Pembagian Tugas Anggota Kelompok

NIM	Nama	Bagian Pengerjaan
18223011	Samuel Chris Michael Bagasta S.	- Visualisasi *Hill-Climbing Steepest Ascent* - Mengerjakan Deskripsi Persoalan (Bab 1), Pembahasan *Steepest Ascent Hill Climbing* (Bab 2), Pembahasan *Stochastic Hill Climbing* (Bab 2), Analisis (Bab 2), Hasil eksperimen *Steepest Ascent Hill Climbing* (Bab 2)
18223057	Stanislaus Ardy Bramantyo	- Algoritma dasar *Hill-Climbing Steepest Ascent, Hill-Climbing Stochastic, Simulated Annealing, dan Genetics.* - Membantu dalam beberapa visualisasi (Genetics & Simulated Annealing) - Mengerjakan Hasil Eksperimen *Simulated Annealing* & *Genetics* (Bab 2). Kesimpulan dan Saran (Bab 3)
18223097	Audy Alicia Renatha Tirayoh	- Visualisasi *Hill-Climbing Stochastic* dan sebagian *Simulated Annealing* - Mengerjakan Deskripsi

		Persoalan(Bab 1), Pembahasan *Genetic Algorithm*(Bab 2), dan *formatting* laporan - Membuat *file* JSON untuk *test-case*(*case* ke-1 sampai *case* ke-4) - Membuat *file* README.md

REFERENSI

1. Delorme, A., Iori, M., & Martello, S. (2016). *Bin packing and cutting stock problems: Mathematical models and exact algorithms. European Journal of Operational Research, 255*(1), 1–20.
2. Falkenauer, E. (1996). *A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics, 2*(1), 5–30.
3. GeeksforGeeks. (n.d.). *Introduction to hill climbing in artificial intelligence.* Retrieved October 29, 2024, from https://www.geeksforgeeks.org/artificial-intelligence/introduction-hill-climbing-artificial-intelligence/
4. GeeksforGeeks. (n.d.). *What is simulated annealing.* Retrieved October 29, 2024, from https://www.geeksforgeeks.org/artificial-intelligence/what-is-simulated-annealing/
5. Lodi, A., Martello, S., & Monaci, M. (2002). *Two-dimensional bin packing problems: A survey. European Journal of Operational Research, 141*(2), 241–252.
6. Russell, S., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Prentice Hall.