LAPORAN TUGAS BESAR 1

“AI Bin Packing”

IF3070 DASAR INTELIGENSI ARTIFISIAL

Dosen Mata Kuliah: Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

Disusun Oleh:

Kelompok 46

Keane Putra Lim	/ 18223056
Sebastian Albern Nugroho	/ 18223074
M Khalfani Shaquille	/ 18223104

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
DESKRIPSI PERSOALAN

Tugas besar ini fokus pada menyelesaikan masalah optimasi yaitu Bin Packing Problem. Permasalahan utama ini adalah bagaimana menempatkan sekumpulan barang dengan berbagai ukuran ke dalam kontainer yang identik.

Pada tugas ini, terdapat aturan yang sudah ditentukan yaitu total ukuran barang yang ditempatkan di dalam satu kontainer tidak boleh melebihi kapasitas kontainer tersebut. Penyelesaian masalah ini dilakukan dengan menggunakan tiga algoritma yaitu algoritma Hill-Climbing Search, algoritma Simulated Annealing, dan algoritma Genetic Algorithm. Pemenang dari penyelesaian permasalahan ini adalah algoritma yang dapat menyusun barang pada container dengan jumlah kontainer terkecil.

Dalam tugas ini, kami akan merancang dan mengimplementasikan beberapa algoritma local search untuk mencari solusi yang paling optimal (atau mendekati optimal) untuk masalah Bin Packing yang diberikan.

BAB II

PEMBAHASAN

2.1. Objective Function

Objective Function adalah sebuah fungsi penting yang mengatur kualitas dari solusi yang ada. Dalam masalah *bin packing*, tujuan utama dari *Objective Function* adalah untuk meminimalisir jumlah kontainer yang akan digunakan. Cara kerja dari fungsi ini cukup sederhana. Setiap algoritma akan berusaha menghasilkan solusi yang meminimalkan nilai dari fungsi ini.

Dalam kasus ini, ada 3 acuan dalam membuat *Objective Function*. Fungsi harus memberikan penalti besar kepada kontainer dengan isi melebihi kapasitasi, fungsi harus memberikan skor berdasarkan jumlah kontainer serta kepadatan kontainer, dan kombinasi perhitungan keduanya. Selain itu, ada implementasi bonus yang juga memengaruhi perhitungan *Objective Function*. Bonus tersebut meliputi perhitungan barang rapuh serta barang tidak kompatibel. Barang rapuh berarti akan ada penalti apabila suatu barang rapuh diletakkan di dalam sebuah kontainer bersama barang lain yang total ukurannya melebihi ambang batas (di dalam program ditetapkan batasnya adalah 50). Barang tidak kompatibel berarti sebuah barang dengan tipe tertentu harus diletakkan di dalam satu kontainer dengan tipe yang sama.

Perhitungan penalti kapasitas berlebih dilakukan dengan fungsi sebagai berikut:

```
OVERFILL_PENALTY_MULTIPLIER = 1_000_000

overfill_penalty = 0.0
for kontainer in state.kontainer_list:
    if kontainer.muatan_saat_ini > kontainer.kapasitas:
        overfill_penalty += (kontainer.muatan_saat_ini -
kontainer.kapasitas) * OVERFILL_PENALTY_MULTIPLIER

if overfill_penalty > 0:
    return overfill_penalty
```

Perhitungan dilakukan dengan menghitung selisih muatan kontainer dengan kapasitas kontainer, lalu mengalikannya dengan *multiplier* dengan nilai yang sangat besar. Ini dilakukan untuk memastikan skor penalti yang sangat besar bagi solusi dengan kontainer melebihi kapasitas.

Perhitungan jumlah kontainer sangat sederhana. Perhitungan hanya dilakukan dengan menghitung jumlah kontainer yang digunakan.

```
container_score = len(state.kontainer_list)
```

Jumlah kontainer akan secara langsung berpengaruh kepada skor. Semakin banyak menggunakan kontainer maka skor akan semakin besar. Semakin sedikit menggunakan kontainer maka skor akan semakin kecil.

Perhitungan kepadatan kontainer akan memprioritaskan kontainer yang lebih padat dibandingkan kontainer yang lebih kosong. Perhitungannya dilakukan sebagai berikut:

```
total_density_score = 0
if container_score > 0:
    for k in state.kontainer_list:
        density = k.muatan_saat_ini / k.kapasitas
        total_density_score += (1 - density**2)
    density_bonus = total_density_score / container_score
else:
    density_bonus = 0
```

Penalti untuk barang rapuh dihitung dengan memberikan penalti besar kepada kontainer yang memiliki barang rapuh dan memiliki kapasitas di atas ambang batas. Perhitungannya dilakukan sebagai berikut:

```
FRAGILE_PENALTY = 500_000

fragile_penalty = 0.0
if config.use_fragile_constraint:
```

```
    for k in state.kontainer_list:
        if any(b.rapuh for b in k.barang_di_dalam):
            muatan_non_rapuh = sum(b.ukuran for b in
k.barang_di_dalam if not b.rapuh)
            if muatan_non_rapuh > config.fragile_threshold:
                fragile_penalty += FRAGILE_PENALTY
```

Jika ada kontainer yang melanggar ketentuan *fragile*, maka skor sebesar 500000 akan ditambahkan ke skor objektif secara keseluruhan. Namun jika tidak melebihi ketentuan, maka tidak ada penalti yang akan ditambahkan.

Penalti barang inkompatibel dihitung dengan menghitung elemen yang tidak sesuai dengan pasangan kompatibel. Jika ditemukan kontainer dengan barang yang tidak kompatibel, maka nilai penalti akan ditambahkan ke nilai skor objektif. Berikut adalah perhitungannya:

```
FRAGILE_PENALTY = 500_000

fragile_penalty = 0.0
if config.use_fragile_constraint:
    for k in state.kontainer_list:
        if any(b.rapuh for b in k.barang_di_dalam):
            muatan_non_rapuh = sum(b.ukuran for b in
k.barang_di_dalam if not b.rapuh)
            if muatan_non_rapuh > config.fragile_threshold:
                fragile_penalty += FRAGILE_PENALTY
```

Setelah selesai melakukan perhitungan, maka seluruh skor penalti akan digabungkan untuk membentuk sebuah skor total.

```
total_score = (container_score + density_bonus +
fragile_penalty + incompatible_penalty)
```

Setiap algoritma akan berusaha untuk mencari skor total terendah sebagai solusinya. Skor terendah akan memastikan solusi yang dihasilkan sudah benar, sesuai ketentuan, serta merupakan solusi paling efisien yang bisa dicapai oleh algoritma tersebut.

2.2. Constraint

Constraint yang digunakan untuk eksperimen adalah *incompatible* dan rapuh. *Incompatibility* digunakan untuk memisahkan tipe-tipe barang di dalam kontainer. Jika *constraint incompatible* dinyalakan, maka barang "makanan" dan "kimia" tidak boleh dimasukkan ke dalam satu kontainer yang sama. Ini diatur di dalam parameter berikut:

```
incompatible_pairs: list[tuple[str, str]] =
field(default_factory=lambda: [('makanan', 'kimia')])
```

Constraint kedua yang digunakan adalah rapuh. Barang yang rapuh tidak boleh dimasukkan ke dalam kontainer dengan isi melebihi ambang batas. Ambang batas ditetapkan sendiri di dalam program. Untuk kasus ini, ambang batas ditetapkan di angka 50.

```
fragile_threshold: int = 50
```

2.3. Utilities

Utilities atau utils adalah fungsi yang digunakan untuk mendukung proses optimasi dan pencarian solusi. Ada beberapa fungsi yang digunakan secara luas di seluruh algoritma. Fungsi-fungsi tersebut didefinisikan sebagai berikut:

2.3.1. moves.py

moves.py digunakan untuk menghasilkan keadaan tetangga dari keadaan saat ini. Keadaan tetangga merupakan variasi kecil dari solusi saat ini. Ini dibuat dengan melakukan perubahan kecil. *Moves* dilakukan dengan dua cara yaitu memindahkan barang ke kontainer lain atau menukar 2 barang dalam kontainer yang berbeda. Berikut adalah kode yang digunakan :

```
import random
from typing import List, Tuple

from src.core.data_structures import State, Kontainer,
Barang

def get_random_neighbor(state: State) -> State:

    # Menghasilkan keadaan tetangga secara acak dengan
melakukan salah satu dari dua operasi:
    # 1. Memindahkan satu barang secara acak ke
kontainer lain (bisa kontainer baru).
    # 2. Menukar dua barang secara acak dari dua
kontainer yang berbeda.

    new_state = state.salin()

    # Pilih antara memindahkan (move) atau menukar
(swap) barang
    if random.random() < 0.7 or
len(new_state.kontainer_list) < 2: # Lebih sering
memindahkan
        return _relocate_random_item(new_state)
    else:
        return _swap_random_items(new_state)

def _relocate_random_item(state: State) -> State:
    # Memindahkan satu barang acak ke kontainer acak
(termasuk kemungkinan membuat kontainer baru)

    # Pilih kontainer yang tidak kosong
    non_empty_containers = [k for k in
state.kontainer_list if k.barang_di_dalam]
    if not non_empty_containers:
        return state
```

```
    source_container =
random.choice(non_empty_containers)
    item_to_move =
random.choice(source_container.barang_di_dalam)

    # Pilih kontainer tujuan, bisa juga membuat yang
baru
    # Peluang 1/N untuk membuat kontainer baru, dimana N
adalah jumlah kontainer
    if random.random() < 1 / (len(state.kontainer_list)
+ 1):
        # Buat kontainer baru
        new_container_id = max([k.id for k in
state.kontainer_list]) + 1 if state.kontainer_list else
1
        # Asumsi kapasitas sama untuk semua, ambil dari
kontainer pertama
        new_capacity = state.kontainer_list[0].kapasitas
if state.kontainer_list else 100
        target_container =
Kontainer(id=new_container_id, kapasitas=new_capacity)
        state.kontainer_list.append(target_container)
    else:
        # Pilih dari kontainer yang sudah ada
        target_container =
random.choice(state.kontainer_list)

    # Pindahkan barang

source_container.barang_di_dalam.remove(item_to_move)
    target_container.tambah_barang(item_to_move)

    # Hapus kontainer kosong jika ada (kecuali hanya ada
satu)
    if not source_container.barang_di_dalam and
len(state.kontainer_list) > 1:
```

```
        state.kontainer_list.remove(source_container)

    return state

def _swap_random_items(state: State) -> State:
    # Menukar dua barang acak dari dua kontainer yang
berbeda

    # Pilih dua kontainer berbeda yang tidak kosong
    non_empty_containers = [k for k in
state.kontainer_list if k.barang_di_dalam]
    if len(non_empty_containers) < 2:
        return state # Tidak cukup kontainer untuk
menukar

    container1, container2 =
random.sample(non_empty_containers, 2)

    item1 = random.choice(container1.barang_di_dalam)
    item2 = random.choice(container2.barang_di_dalam)

    # Tukar barang
    container1.barang_di_dalam.remove(item1)
    container2.barang_di_dalam.remove(item2)
    container1.tambah_barang(item2)
    container2.tambah_barang(item1)

    return state

def get_all_neighbors(state: State) -> List[State]:
    """
    Menghasilkan semua kemungkinan keadaan tetangga dari
keadaan saat ini dengan menerapkan
    semua kemungkinan gerakan relokasi dan pertukaran.

    Args:
```

```
        state: Keadaan saat ini.

    Returns:
        Sebuah list dari semua kemungkinan keadaan
tetangga.
    """
    neighbors = get_all_relocation_moves(state)
    neighbors.extend(get_all_swap_moves(state))
    return neighbors

def get_all_relocation_moves(state: State) ->
List[State]:
    """
    Menghasilkan semua keadaan tetangga dengan
memindahkan satu barang ke kontainer lain
    (termasuk yang baru).

    Args:
        state: Keadaan saat ini.

    Returns:
        Sebuah list keadaan tetangga yang dibuat oleh
gerakan relokasi.
    """
    neighbors = []
    all_items = [item for container in
state.kontainer_list for item in
container.barang_di_dalam]

    for item_to_move in all_items:
        source_container = next((c for c in
state.kontainer_list if item_to_move in
c.barang_di_dalam), None)
        if not source_container:
            continue
```

```
        # Coba pindah ke setiap kontainer lain yang ada
        for target_container in state.kontainer_list:
            if source_container.id ==
target_container.id:
                continue

            if
target_container.bisa_tambah_barang(item_to_move):
                new_state = state.salin()

                # Ambil referensi objek dari keadaan
baru
                new_source_container = next(c for c in
new_state.kontainer_list if c.id == source_container.id)
                new_target_container = next(c for c in
new_state.kontainer_list if c.id == target_container.id)
                new_item = next(i for i in
new_source_container.barang_di_dalam if i.id ==
item_to_move.id)

                # Lakukan pemindahan

new_source_container.barang_di_dalam.remove(new_item)

new_target_container.barang_di_dalam.append(new_item)

                # Hapus kontainer asal jika menjadi
kosong
                if not
new_source_container.barang_di_dalam:

new_state.kontainer_list.remove(new_source_container)

                neighbors.append(new_state)

        # Coba pindah ke kontainer baru
```

```
        new_state_for_new_container = state.salin()
        source_in_new_state = next(c for c in
new_state_for_new_container.kontainer_list if c.id ==
source_container.id)
        item_in_new_state = next(i for i in
source_in_new_state.barang_di_dalam if i.id ==
item_to_move.id)


source_in_new_state.barang_di_dalam.remove(item_in_new_s
tate)

        new_container_id = max(c.id for c in
new_state_for_new_container.kontainer_list) + 1 if
new_state_for_new_container.kontainer_list else 1
        new_container = Kontainer(id=new_container_id,
kapasitas=source_container.kapasitas,
barang_di_dalam=[item_in_new_state])

new_state_for_new_container.kontainer_list.append(new_co
ntainer)

        if not source_in_new_state.barang_di_dalam:

new_state_for_new_container.kontainer_list.remove(source
_in_new_state)

        neighbors.append(new_state_for_new_container)

    return neighbors

def get_all_swap_moves(state: State) -> List[State]:
    """
    Menghasilkan semua keadaan tetangga dengan menukar
dua barang dari dua kontainer yang berbeda.
```

```
    Args:
        state: Keadaan saat ini.

    Returns:
        Sebuah list keadaan tetangga yang dibuat oleh
gerakan pertukaran.
    """
    neighbors = []

    # Loop semua pasangan kontainer yang unik
    for i in range(len(state.kontainer_list)):
        for j in range(i + 1,
len(state.kontainer_list)):
            container1 = state.kontainer_list[i]
            container2 = state.kontainer_list[j]

            # Loop semua pasangan barang dari dua
kontainer tersebut
            for item1 in container1.barang_di_dalam:
                for item2 in container2.barang_di_dalam:
                    # Cek apakah pertukaran valid (tidak
melebihi kapasitas)
                    new_load1 =
container1.muatan_saat_ini - item1.ukuran + item2.ukuran
                    new_load2 =
container2.muatan_saat_ini - item2.ukuran + item1.ukuran

                    if new_load1 <= container1.kapasitas
and new_load2 <= container2.kapasitas:
                        new_state = state.salin()

                        # Ambil referensi objek dari
keadaan baru
                        new_container1 = next(c for c in
new_state.kontainer_list if c.id == container1.id)
                        new_container2 = next(c for c in
```

```
new_state.kontainer_list if c.id == container2.id)
                        new_item1 = next(it for it in
new_container1.barang_di_dalam if it.id == item1.id)
                        new_item2 = next(it for it in
new_container2.barang_di_dalam if it.id == item2.id)

                        # Lakukan pertukaran

new_container1.barang_di_dalam.remove(new_item1)

new_container2.barang_di_dalam.remove(new_item2)

new_container1.barang_di_dalam.append(new_item2)

new_container2.barang_di_dalam.append(new_item1)

                        neighbors.append(new_state)

    return neighbors
```

Gerakan pertama adalah memindahkan barang ke kontainer lain. Ada 2 varian dalam gerakan ini yaitu perpindahan acak dan sistematis. Cara kerja keduanya hampir sama, yaitu memindahkan barang ke kontainer yang berbeda. Perbedaannya terletak di metode pemindahannya. Untuk perpindahan acak, algoritma akan mencari satu barang secara random dan memindahkannya ke kontainer lain. Sedangkan untuk perpindahan sistematis, algoritma akan mencari setiap kemungkinan perpindahan untuk semua barang. Kemudian algoritma akan memindahkan barang tersebut ke kontainer yang benar-benar baru.

Gerakan kedua adalah pertukaran barang. Sama seperti perpindahan barang, gerakan ini juga memiliki dua varian yaitu pertukaran acak dan pertukaran sistematis. Perbedaannya juga sama seperti gerakan perpindahan barang dimana pertukaran acak akan menukar 2 barang di 2 kontainer yang berbeda secara acak sedangkan pertukaran sistematis akan menukar setiap kemungkinan tetangga yang ada.

2.3.2. data_structures.py

Struktur data adalah fondasi dari seluruh proyek ini. File ini bertanggung jawab untuk mendefinisikan berbagai kelas utama yang akan digunakan oleh program. Fungsi akan digunakan untuk merepresentasikan setiap komponen di dalam masalah *bin packing*. Ada tiga kelas di dalam *file* ini.

```
from dataclasses import dataclass, field
from typing import List, Optional

@dataclass
class Barang:
    # Merepresentasikan satu barang dengan semua
atributnya.
    id: str
    ukuran: int
    tipe: Optional[str] = None
    rapuh: bool = False

@dataclass
class Kontainer:
    # Merepresentasikan satu kontainer.
    id: int
    kapasitas: int
    barang_di_dalam: List[Barang] =
field(default_factory=list)

    @property
    def muatan_saat_ini(self) -> int:
        # Menghitung total ukuran barang di dalam
kontainer.
        return sum(b.ukuran for b in
self.barang_di_dalam)

    @property
```

```
    def sisa_kapasitas(self) -> int:
        # Menghitung sisa kapasitas yang tersedia.
        return self.kapasitas - self.muatan_saat_ini

    def bisa_tambah_barang(self, barang: Barang) ->
bool:
        # Mengecek apakah sebuah barang masih muat.
        return self.sisa_kapasitas >= barang.ukuran

    def tambah_barang(self, barang: Barang):
        # Menambahkan barang ke dalam kontainer.
        self.barang_di_dalam.append(barang)

@dataclass
class State:
    # Merepresentasikan sebuah solusi lengkap (alokasi
barang ke kontainer).
    kontainer_list: List[Kontainer]
    barang_belum_dialokasi: List[Barang] =
field(default_factory=list)

    def salin(self) -> 'State':
        # Membuat deep copy dari state saat ini untuk
eksplorasi oleh algoritma.
        return State(
            kontainer_list=[
                Kontainer(
                    id=k.id,
                    kapasitas=k.kapasitas,

barang_di_dalam=list(k.barang_di_dalam)
                ) for k in self.kontainer_list
            ],

barang_belum_dialokasi=list(self.barang_belum_dialokasi)
        )
```

Kelas pertama adalah Barang. Sesuai dengan namanya, kelas ini digunakan untuk merepresentasikan satu unit barang yang kemudian akan dimasukkan ke dalam kontainer. Ada beberapa atribut di dalam kelas ini yaitu id berupa string, ukuran berupa integer, tipe berupa string, serta rapuh yang berupa boolean.

Kelas kedua adalah Kontainer yang bertujuan untuk merepresentasikan satu unit kontainer yang digunakan untuk menampung barang. Ada tiga atribut pada kelas ini yaitu id, kapasitas, serta barang_di_dalam.

Kelas ketiga adalah State. State bertujuan untuk merepresentasikan sebuah solusi lengkap dalam satu waktu tertentu. State akan terus berubah dan dievaluasi hingga menemukan state terbaik. Atribut dalam state mencakup kontainer_list dan barang_belum_dialokasi.

2.3.3. Initial_state.py

File ini berisikan fungsi-fungsi untuk membuat sebuah keadaan awal dari algoritma. Keadaan awal merupakan solusi pertama yang mungkin saja belum optimal. Solusi awal ini akan menjadi titik awal bagi seluruh algoritma dalam proses pencarian solusi.

```
import random
from typing import List, Sequence
from src.core.data_structures import Barang, Kontainer,
State
from src.utils.state_utils import renumber_container_ids

def generate_ffd_state(daftar_barang: list[Barang],
kapasitas_kontainer: int) -> State:
    # Membuat state awal menggunakan heuristik First Fit
Decreasing (FFD).
    # Pisahkan barang yang bisa dimuat dan yang tidak
(karena terlalu besar)
```

```
    barang_untuk_ditempatkan = []
    barang_belum_dialokasi = []
    for barang in daftar_barang:
        if barang.ukuran > kapasitas_kontainer:
            barang_belum_dialokasi.append(barang)
        else:
            barang_untuk_ditempatkan.append(barang)

    # Barang diurutkan dari yang terbesar ke terkecil,
lalu dimasukkan ke kontainer pertama yang muat.
    barang_diurutkan = sorted(barang_untuk_ditempatkan,
key=lambda b: b.ukuran, reverse=True)

    kontainer_list: list[Kontainer] = []
    id_kontainer_selanjutnya = 0

    for barang in barang_diurutkan:
        ditempatkan = False
        # Coba tempatkan di kontainer yang ada
        for kontainer in kontainer_list:
            if kontainer.bisa_tambah_barang(barang):
                kontainer.tambah_barang(barang)
                ditempatkan = True
                break

        # Jika tidak muat di mana pun, buat kontainer
baru
        if not ditempatkan:
            kontainer_baru =
Kontainer(id=id_kontainer_selanjutnya,
kapasitas=kapasitas_kontainer)
            kontainer_baru.tambah_barang(barang)
            kontainer_list.append(kontainer_baru)
            id_kontainer_selanjutnya += 1

    return State(kontainer_list=kontainer_list,
```

```
barang_belum_dialokasi=barang_belum_dialokasi)

def generate_random_state(items: Sequence[Barang],
kapasitas: int, rng: random.Random) -> State:
    """
    Membuat state awal secara acak.

    Barang-barang (dalam urutan acak) ditempatkan ke
dalam kontainer yang dipilih
    secara acak. Jika tidak ada kontainer yang muat,
kontainer baru akan dibuat.

    Args:
        items: Sekumpulan barang yang akan ditempatkan.
        kapasitas: Kapasitas untuk setiap kontainer
baru.
        rng: Generator angka acak untuk memastikan hasil
yang dapat direplikasi.

    Returns:
        Sebuah state baru yang dihasilkan secara acak.
    """
    # Pisahkan barang yang bisa dimuat dan yang tidak
    barang_untuk_ditempatkan = []
    barang_belum_dialokasi = []
    for barang in items:
        if barang.ukuran > kapasitas:
            barang_belum_dialokasi.append(barang)
        else:
            barang_untuk_ditempatkan.append(barang)

    kontainer_list: List[Kontainer] = []
    # Acak urutan barang untuk variasi penempatan
    if barang_untuk_ditempatkan:
        for barang in
rng.sample(barang_untuk_ditempatkan,
```

```
len(barang_untuk_ditempatkan)):
            # Coba tempatkan di kontainer yang ada
terlebih dahulu
            ditempatkan = False
            if kontainer_list:
                # Acak urutan kontainer yang akan dicoba
                kandidat_kontainer =
rng.sample(kontainer_list, len(kontainer_list))
                for kontainer in kandidat_kontainer:
                    if
kontainer.bisa_tambah_barang(barang):

kontainer.barang_di_dalam.append(barang)
                        ditempatkan = True
                        break

            # Jika tidak muat dimanapun, buat kontainer
baru
            if not ditempatkan:
                kontainer_baru =
Kontainer(id=len(kontainer_list), kapasitas=kapasitas,
barang_di_dalam=[barang])
                kontainer_list.append(kontainer_baru)

    renumber_container_ids(kontainer_list)
    return State(kontainer_list=kontainer_list,
barang_belum_dialokasi=barang_belum_dialokasi)
```

Ada dua metode yang digunakan untuk menentukan *initial state* yaitu:

2.3.3.1. **First Fit Decreasing**

First Fit Decreasing akan membuat sebuah keadaan awal menggunakan heuristik. Ini merupakan metode yang cukup banyak digunakan dan seringkali menghasilkan solusi awal yang cukup baik. Cara kerjanya cukup sederhana yaitu dengan meletakkan barang berdasarkan urutan ukurannya

menurun. Jika kontainer pertama tidak muat lagi, maka kontainer kedua akan diisi dan seterusnya.

2.3.3.2.Random State

Berbeda dengan FFD, random state akan membuat *initial state* dengan sangat acak. Tujuannya adalah menghasilkan solusi awal yang sangat bervariatif. Cara kerjanya sangat sederhana yaitu dengan mengambil satu barang lalu menempatkannya di kontainer acak.

2.4. Local Search

2.4.1. Steepest Ascent Hill Climbing

Steepest Ascent Hill Climbing (Pendakian Bukit dengan Kemiringan Terbesar) adalah algoritma optimasi yang termasuk dalam keluarga local search (pencarian lokal). Algoritma ini bersifat deterministik dan greedy (rakus), yang berarti pada setiap langkahnya, ia selalu memilih langkah terbaik yang tersedia di sekitarnya untuk menuju solusi yang lebih optimal.

Algoritma ini bekerja dengan 2 prinsip. Pertama berupa pencarian lokal dimana algoritma hanya mempertimbangkan keadaan-keadaan yang bersebelahan (tetangga) dari keadaan saat ini. Kedua adalah greedy approach yakni algoritma tidak pernah memilih langkah yang akan memperburuk solusi. Keputusan yang diambil pada setiap langkah bersifat final dan tidak mempertimbangkan dampak jangka panjang.

Proses kerja dari Steepest Ascent Hill Climbing dapat diuraikan sebagai berikut:

1. Inisialisasi

   Inisialisasi dimulai dari sebuah keadaan_awal (solusi awal), yang bisa dihasilkan melalui heuristik (seperti FFD) atau secara acak.
2. Iterasi
   a. Dari keadaan_saat_ini, terdapat semua kemungkinan keadaan_tetangga yang bisa dicapai dengan satu langkah

(misalnya, semua kemungkinan pemindahan satu barang atau semua kemungkinan pertukaran dua barang).

b. Mencari tetangga terbaik dengan menghitung skor fungsi objektif untuk setiap keadaan tetangga tersebut. Lalu, mengidentifikasi satu tetangga yang memiliki skor paling rendah (paling baik), yang kita sebut tetangga_terbaik.
c. Membandingkan skor tetangga_terbaik dengan skor_saat_ini untuk membuat keputusan. Jika skor(tetangga_terbaik) < skor_saat_ini, ditemukan sebuah langkah yang memperbaiki solusi. Perbarui keadaan_saat_ini menjadi tetangga_terbaik dan lanjutkan ke iterasi berikutnya. Jika `skor(tetangga_terbaik) >= skor_saat_ini`, artinya, tidak ada satu pun tetangga yang memiliki solusi lebih baik daripada keadaan saat ini. Algoritma telah mencapai sebuah "puncak".

3. Algoritma determinasi/berhenti ketika tidak ada lagi langkah yang bisa diambil untuk memperbaiki skor. Keadaan saat ini dikembalikan sebagai solusi akhir.

Pada proyek ini, algoritma Steepest Ascent Hill Climbing diimplementasikan dengan mendefinisikan keadaan sebagai suatu konfigurasi penempatan barang di dalam kontainer. Kualitas suatu keadaan dievaluasi melalui fungsi objektif berupa skor yang ditentukan berdasarkan jumlah kontainer yang digunakan dan tingkat efisiensi penempatannya. Ruang pencarian dijelajahi dengan menghasilkan semua tetangga yang mungkin melalui fungsi get_all_neighbors, yang mencakup seluruh kemungkinan pemindahan satu barang ke kontainer lain serta semua kemungkinan pertukaran dua barang antar kontainer. Prinsip langkah paling curam diwujudkan dengan menghitung skor untuk setiap tetangga yang dihasilkan, kemudian memilih satu langkah tertentu seperti memindahkan Barang A ke Kontainer 3 yang memberikan penurunan skor terbesar, sehingga memastikan perpindahan ke keadaan tetangga yang paling optimal pada setiap iterasi.

```
def steepest_ascent_hill_climbing(
    initial_state: State,
    config: ObjectiveConfig,
    max_iter: int
) -> Tuple[State, List[float]]:
    """
    Mengimplementasikan Steepest Ascent Hill Climbing.

    Pada setiap iterasi, algoritma ini mengevaluasi semua keadaan
tetangga dan
    memilih yang memberikan penurunan skor terbesar (paling
curam). Pencarian
    berhenti jika tidak ada tetangga yang lebih baik atau iterasi
maksimum tercapai.

    Args:
        initial_state: Keadaan awal untuk memulai pencarian.
        config: Konfigurasi untuk fungsi objektif.
        max_iter: Jumlah iterasi maksimum.

    Returns:
        Tuple berisi (keadaan terbaik yang ditemukan, histori
skor).
    """
    current_state = initial_state.salin()
    current_score = calculate_objective(current_state, config)
    score_history = [current_score]

    for _ in range(max_iter):
        neighbors = get_all_neighbors(current_state)
        if not neighbors:
            break

        neighbor_scores = [calculate_objective(n, config) for n
in neighbors]
```

```
        best_neighbor_score = min(neighbor_scores)
        best_neighbor_index =
neighbor_scores.index(best_neighbor_score)
        best_neighbor = neighbors[best_neighbor_index]

        if best_neighbor_score < current_score:
            current_state = best_neighbor
            current_score = best_neighbor_score
            score_history.append(current_score)
        else:
            break

    return current_state, score_history
```

2.4.2. Hill-Climbing with Sideways Move

Hill-Climbing with Sideways Move merupakan varian dari algoritma Hill Climbing yang memperbolehkan langkah horizontal (sideways) ketika tidak ada tetangga yang lebih baik. Dalam implementasi standar, algoritma hanya bergerak ketika menemukan tetangga dengan nilai fitness lebih tinggi. Namun, dalam versi ini, algoritma diperbolehkan bergerak ke tetangga dengan nilai fitness yang sama ketika tidak ada tetangga yang lebih baik.

Mekanisme algoritma ini hampir identik dengan Steepest Ascent, namun dengan satu tambahan aturan pada langkah pengambilan keputusan:

1. Melakukan inisialisasi dan mengevaluasi tetangga seperti algoritma Steepest Ascent. Algoritma ini memiliki proses yang sama seperti algoritma Steepest Ascent mulai dari keadaan awal, lalu pada setiap iterasi, menghasilkan dan evaluasi semua keadaan tetangga untuk menemukan tetangga_terbaik.

2. Jika skor(tetangga_terbaik) < skor_saat_ini, maka keputusan diterima. Algoritma beralih ke keadaan yang lebih baik (langkah menanjak). Jika skor(tetangga_terbaik) == skor_saat_ini, maka keputusan tetap diterima. Ini adalah sideways move. Algoritma beralih ke keadaan dengan skor yang sama untuk melanjutkan eksplorasi. Jika skor(tetangga_terbaik) > skor_saat_ini, maka keputusan ditolak dan algoritma berhenti.

```
def hill_climbing_with_sideways_moves(
    initial_state: State,
    config: ObjectiveConfig,
    max_iter: int,
    max_sideways_moves: int
) -> Tuple[State, List[float]]:
    """
    Mengimplementasikan Hill Climbing yang mengizinkan sideways
moves.

    Varian ini mirip dengan Steepest Ascent, tetapi jika tidak
ada tetangga yang
    lebih baik, ia akan menerima tetangga dengan skor yang sama.
Ini berguna
    untuk melintasi 'plateau' dalam lanskap pencarian. Jumlah
sideways moves
    dibatasi oleh `max_sideways_moves`.

    Args:
        initial_state: Keadaan awal untuk memulai pencarian.
        config: Konfigurasi untuk fungsi objektif.
        max_iter: Jumlah iterasi maksimum.
        max_sideways_moves: Jumlah maksimum langkah menyamping
yang diizinkan secara berurutan.

    Returns:
```

```
        Tuple berisi (keadaan terbaik yang ditemukan, histori
skor).
    """
    current_state = initial_state.salin()
    current_score = calculate_objective(current_state, config)
    score_history = [current_score]
    sideways_moves_count = 0

    for _ in range(max_iter):
        neighbors = get_all_neighbors(current_state)
        if not neighbors:
            break

        neighbor_scores = [calculate_objective(n, config) for n
in neighbors]

        best_neighbor_score = min(neighbor_scores)
        best_neighbor_index =
neighbor_scores.index(best_neighbor_score)
        best_neighbor = neighbors[best_neighbor_index]

        if best_neighbor_score < current_score:
            current_state = best_neighbor
            current_score = best_neighbor_score
            score_history.append(current_score)
            sideways_moves_count = 0
        elif best_neighbor_score == current_score and
sideways_moves_count < max_sideways_moves:
            current_state = best_neighbor
            sideways_moves_count += 1
        else:
            break

    return current_state, score_history
```

2.4.3. Stochastic Hill-Climbing

Stochastic Hill-Climbing adalah varian algoritma Hill Climbing yang menggunakan *randomness* dalam proses pemilihan tetangga. Berbeda dengan Steepest Ascent Hill Climbing atau Simple Hill Climbing, Stochastic Hill Climbing memilih tetangga secara acak dari sekumpulan tetangga yang memiliki kualitas lebih baik.

Mekanisme proses dari stochastic hill climbing meliputi inisialisasi, explorasi tetangga dan mencari langkah yang lebih baik, melakukan pemilihan langkah secara acak dan terminasi algoritma. Berikut adalah penjelasan yang lebih detail.

1. Inisialisasi
   Algoritma dimulai dengan sebuah keadaan awal secara acak.
2. Eksplorasi tetangga
   Pada setiap iterasi, algoritma akan menghasilkan semua kemungkinan keadaan tetangga dari keadaan saat ini. Sebuah tetangga adalah keadaan baru yang dihasilkan dari perubahan kecil, seperti memindahkan satu barang ke kontainer lain atau menukar dua barang di antara dua kontainer.
3. Mencari langkah yang lebih baik
   Algoritma mengevaluasi skor dari setiap tetangga yang telah dihasilkan. Ia akan mengidentifikasi semua tetangga yang memiliki skor lebih rendah (lebih baik) daripada nilai kondisi sekarang.
4. Memilih langkah secara acak
   Algoritma mengumpulkan semua tetangga yang lebih baik ke dalam sebuah daftar, lalu memilih salah satu dari mereka secara acak. Pemilihan acak ini memberikan elemen ketidakpastian yang memungkinkan algoritma menjelajahi jalur pencarian yang berbeda, tidak hanya jalur yang paling curam.
5. Terminasi algoritma
   Proses pencarian akan berhenti jika salah satu dari dua kondisi terpenuhi yaitu tidak ada lagi tetangga yang memiliki skor lebih baik dari keadaan saat ini, yang menandakan algoritma telah mencapai

"optimum lokal"atau jumlah iterasi telah mencapai batas max_iter yang ditentukan.

```
def stochastic_hill_climbing(
    initial_state: State,
    config: ObjectiveConfig,
    max_iter: int
) -> Tuple[State, List[float]]:
    """
    Mengimplementasikan Stochastic Hill Climbing.

    Pada setiap iterasi, algoritma ini menemukan semua keadaan
tetangga yang
    memiliki skor lebih baik daripada keadaan saat ini, lalu
memilih salah satu
    dari mereka secara acak untuk menjadi keadaan berikutnya.

    Args:
        initial_state: Keadaan awal untuk memulai pencarian.
        config: Konfigurasi untuk fungsi objektif.
        max_iter: Jumlah iterasi maksimum.

    Returns:
        Tuple berisi (keadaan terbaik yang ditemukan, histori
skor).
    """
    current_state = initial_state.salin()
    current_score = calculate_objective(current_state, config)
    score_history = [current_score]

    for _ in range(max_iter):
        neighbors = get_all_neighbors(current_state)
        if not neighbors:
            break

        better_neighbors = []
```

```
        for neighbor in neighbors:
            neighbor_score = calculate_objective(neighbor,
config)
            if neighbor_score < current_score:
                better_neighbors.append((neighbor,
neighbor_score))

        if better_neighbors:
            chosen_neighbor, chosen_score =
random.choice(better_neighbors)
            current_state = chosen_neighbor
            current_score = chosen_score
            score_history.append(current_score)
        else:
            break

    return current_state, score_history
```

2.4.4. Random Restart Hill-Climbing

Random-Restart Hill Climbing adalah sebuah meta-algoritma probabilistik yang dirancang untuk meningkatkan kemampuan algoritma Hill Climbing dalam menemukan solusi optimal global.

Tujuan utamanya adalah untuk mengatasi masalah optimum lokal, yaitu kondisi di mana algoritma Hill Climbing standar terjebak pada sebuah solusi puncak yang bukan merupakan solusi terbaik secara keseluruhan. Dengan memulai pencarian dari berbagai titik yang berbeda, kemungkinan untuk menemukan optimum global menjadi jauh lebih besar.

Mekanisme dari Random Restart Hill Climbing mencakup inisialisasi global dan proses restart yang berulang. Detail mekanisme adalah sebagai berikut:

1. Inisialisasi

   Algoritma menyimpan satu solusi terbaik secara keseluruhan, best_state_overall. Pada awalnya, solusi ini adalah keadaan awal yang diberikan.

2. Proses *restart* ulang
    a. Membuat keadaan awal secara acak
    b. Menjalankan pencarian lokal untuk menemukan puncak lokal dari area tersebut
    c. Membandingkan hasil yang ditemukan dari pencarian lokal current_best_state dengan solusi terbaik yang ditemukan dari semua restart sebelumnya best_state_overall
    d. Memperbarui solusi terbaik global jika pencarian lokal saat ini lebih baik dibandingkan best_state_overall

```
def random_restart_hill_climbing(
    initial_state: State,
    config: ObjectiveConfig,
    num_restarts: int,
    max_iter_per_restart: int,
    rng: Optional[random.Random] = None,
    kapasitas_kontainer: Optional[int] = None
) -> Tuple[State, List[float]]:
    """
    Mengimplementasikan Random-Restart Hill Climbing.

    Algoritma ini menjalankan Steepest Ascent Hill Climbing
beberapa kali (`num_restarts`).
    Setiap restart dimulai dari sebuah keadaan awal yang dibuat
secara acak.
    Solusi terbaik yang ditemukan dari semua restart akan menjadi
hasil akhir.

    Args:
        initial_state: Digunakan untuk mengekstrak daftar barang
dan kapasitas.
        config: Konfigurasi untuk fungsi objektif.
        num_restarts: Berapa kali pencarian akan diulang dari
```

```
awal.
        max_iter_per_restart: Jumlah iterasi maksimum untuk
setiap proses Hill Climbing.
        rng: Generator angka acak untuk pembuatan state.
        kapasitas_kontainer: Kapasitas kontainer (opsional).

    Returns:
        Tuple berisi (keadaan terbaik yang ditemukan, histori
skor dari pencarian terbaik).
    """
    rng = rng or random.Random()
    best_state_overall = initial_state.salin()
    best_score_overall = calculate_objective(best_state_overall,
config)
    best_history = [best_score_overall]

    all_items = extract_all_items(initial_state)
    if not all_items:
        return best_state_overall, best_history

    kapasitas = resolve_capacity(initial_state,
kapasitas_kontainer)

    for i in range(num_restarts):
        print(f"  Restarting search ({i + 1}/{num_restarts})...")
        random_start_state = generate_random_state(all_items,
kapasitas, rng)

        current_best_state, current_history =
steepest_ascent_hill_climbing(
            initial_state=random_start_state,
            config=config,
            max_iter=max_iter_per_restart
        )
        current_best_score = current_history[-1]
```

```
        if current_best_score < best_score_overall:
            best_score_overall = current_best_score
            best_state_overall = current_best_state
            best_history = current_history

    return best_state_overall, best_history
```

2.4.5. Simulated Annealing

Simulated Annealing adalah algoritma metaheuristic yang digunakan untuk mencari solusi mendekati optimal untuk masalah yang besar dan kompleks, di mana metode pencarian eksak (yang selalu menemukan solusi terbaik) membutuhkan waktu yang terlalu lama. Algoritma ini mampu melakukan "lompatan" ke solusi yang lebih buruk untuk sementara waktu, agar terhindar dari jebakan optimum lokal.

Algoritma ini memiliki mekanisme utama yaitu inisialisasi, looping utama, dan mengeluarkan solusi yang terbaik. Mekanisme secara detail mencakup:

1. Inisialisasi
    - Menetukan suhu awal (T) yang tinggi,
    - Membangun sebuah solusi awal (S) secara acak, dan
    - Menentukan jadwal pendinginan (cooling schedule).
2. Loop Utama
    - Membuat solusi baru (S') dengan memodifikasi solusi saat ini (S) secara acak.
    - Mengevaluasi perubahan dengan menghitung selisih kualitas (ΔE) antara solusi baru dan lama dengan rumus ΔE = Cost(S') - Cost(S), di mana Cost adalah fungsi yang ingin diminimalkan.
    - Jika ΔE < 0 (solusi baru lebih baik), solusi baru selalu diterima. Jika ΔE >= 0 (solusi baru lebih buruk), solusi baru dapat diterima dengan probabilitas e^(-ΔE / T). Keputusan probabilistik ini memungkinkan pelarian dari optimum lokal.

- Menurunkan suhu (T) secara iteratif berdasarkan jadwal pendinginan, misalnya T baru = α * T lama, di mana α adalah konstanta pendinginan (seperti 0.95). Penurunan suhu ini secara bertahap mengurangi kemungkinan penerimaan solusi yang lebih buruk. Di awal (saat T tinggi), probabilitas menerima langkah buruk cukup besar, membuat pencarian bersifat eksploratif. Seiring T menurun, probabilitasnya mengecil, membuat algoritma menjadi lebih "serakah" (eksploitatif) dan fokus pada pencarian di sekitar solusi terbaik yang telah ditemukan.

3. Menghasilkan solusi

Dalam implementasi proyek ini, konsep-konsep Simulated Annealing (SA) diwujudkan melalui empat komponen utama. State didefinisikan sebagai sebuah objek yang merepresentasikan konfigurasi lengkap dari alokasi semua barang ke dalam sekumpulan kontainer. Setiap state menggambarkan satu kemungkinan solusi dari permasalahan pengepakan yang sedang dioptimalkan.

Kualitas dari suatu state dievaluasi melalui Fungsi Objektif (calculate_objective), yang menghitung skor berdasarkan tiga faktor kunci. Faktor-faktor tersebut meliputi jumlah kontainer yang digunakan, tingkat kepadatan alokasi, dan penalti untuk setiap pelanggaran constraint. Fungsi ini berperan sebagai panduan bagi algoritma untuk menentukan arah pencarian solusi yang lebih baik.

Untuk mengeksplorasi ruang pencarian, Operator Gerak (get_random_neighbor) menghasilkan state baru yang bertetangga dengan melakukan modifikasi acak pada state saat ini. Operator ini dapat melakukan berbagai jenis perubahan, seperti memindahkan satu barang ke kontainer lain atau menukar dua barang antara kontainer yang berbeda, sehingga menciptakan variasi dalam solusi yang diuji.

Fleksibilitas algoritma dikendalikan oleh Parameter yang dapat dikonfigurasi, yaitu suhu_awal dan cooling_rate. Parameter-parameter ini memungkinkan dilakukan eksperimen untuk menyelidiki pengaruhnya terhadap kinerja algoritma dalam konteks permasalahan yang spesifik.

Secara keseluruhan, mekanisme ini memungkinkan SA untuk secara sistematis menjelajahi berbagai konfigurasi pengepakan. Melalui iterasi yang terkontrol, algoritma bertujuan menemukan solusi yang efektif dalam meminimalkan jumlah kontainer yang digunakan sekaligus memenuhi berbagai constraint yang berlaku.

```
import math
import random
from typing import List, Tuple

from src.core.data_structures import State
from src.core.objective_function import calculate_objective,
ObjectiveConfig
from src.algorithms.utils.moves import get_random_neighbor

def simulated_annealing(
    keadaan_awal: State,
    suhu_awal: float,
    cooling_rate: float,
    max_iter: int,
    config: ObjectiveConfig
) -> Tuple[State, List[float], List[float]]:

    #     keadaan_awal: State awal untuk memulai pencarian.
    #     suhu_awal: Temperatur awal untuk proses annealing.
    #     cooling_rate: Faktor pengurangan temperatur (e.g.,
0.99).
    #     max_iter: Jumlah iterasi maksimum yang akan
dijalankan.
    #     config: Konfigurasi untuk fungsi objektif (e.g.,
penggunaan constraint bonus).
    #

    # Salin keadaan awal untuk menghindari modifikasi objek
aslinya
```

```
    keadaan_saat_ini = keadaan_awal.salin()
    skor_saat_ini = calculate_objective(keadaan_saat_ini,
config)

    # Inisialisasi state terbaik global dengan keadaan awal
    keadaan_terbaik_global = keadaan_saat_ini
    skor_terbaik_global = skor_saat_ini

    histori_skor = [skor_saat_ini]
    histori_probabilitas = [1.0] # Probabilitas awal adalah 1.0
    suhu = suhu_awal

    for _ in range(max_iter):
        # Hasilkan tetangga secara acak
        keadaan_tetangga =
get_random_neighbor(keadaan_saat_ini)

        # Hitung skor tetangga
        skor_tetangga = calculate_objective(keadaan_tetangga,
config)

        # Hitung perbedaan energi (skor)
        delta_e = skor_tetangga - skor_saat_ini

        # Tentukan apakah akan menerima keadaan baru
        if delta_e < 0:
            probabilitas_penerimaan = 1.0
            keadaan_saat_ini = keadaan_tetangga
            skor_saat_ini = skor_tetangga
        else:
            # Jika tetangga lebih buruk, terima dengan
probabilitas tertentu
            # Ini adalah inti dari SA untuk keluar dari optimum
lokal
            probabilitas_penerimaan = math.exp(-delta_e / suhu)
            if random.random() < probabilitas_penerimaan:
```

```
                keadaan_saat_ini = keadaan_tetangga
                skor_saat_ini = skor_tetangga

        # Perbarui solusi terbaik global jika ditemukan yang
lebih baik
        if skor_saat_ini < skor_terbaik_global:
            keadaan_terbaik_global = keadaan_saat_ini
            skor_terbaik_global = skor_saat_ini

        # Simpan data saat ini untuk analisis
        histori_skor.append(skor_saat_ini)
        histori_probabilitas.append(probabilitas_penerimaan)

        # Turunkan suhu
        suhu *= cooling_rate

    return keadaan_terbaik_global, histori_skor,
histori_probabilitas
```

2.4.6. Genetic Algorithm

Genetic Algorithm adalah sebuah algoritma yang menggunakan prinsip evolusi DNA genetik. Algoritma ini akan meniru cara makhluk hidup dalam seleksi alam, perkawinan, serta mutasi. Algoritma ini sangat cocok untuk diterapkan ke dalam masalah optimasi karena kemampuannya yang sangat baik dalam memecahkan masalah kompleks.

Algoritma ini bekerja dengan cara mengelompokkan sekumpulan kandidat solusi yang disebut sebagai populasi, lalu melakukan seleksi agar mendapatkan solusi terbaik. Setiap solusi di dalam populasi disebut dengan individu dan setiap individu direpresentasikan dengan kromosom. Dalam masalah *bin packing*, kromosom merepresentasikan urutan penempatan barang di dalam kontainer.

Di dalam permasalahan *bin packing*, pertama algoritma akan membuat sebuah *initial state*. Dalam kasus ini, *initial state* bisa dipilih antara menggunakan

algoritma *First Fit Decreasing* atau *Random State* (dengan pilihan *default* yaitu *First Fit Decreasing*).

Algoritma kemudian akan masuk ke fase *loop* yang mana akan terus berjalan sebanyak *max_generations* yang telah ditetapkan. Dalam satu loop, populasi baru akan diciptakan. Ini akan disebut sebagai satu generasi. Setelah itu, proses seleksi akan dimulai. Dua individu akan dipilih sebagai *parent* dari populasi saat ini. Pemilihan orang tua menggunakan metode *Tournament Selection*. Metode ini akan mengambil sebagian kecil secara acak dari populasi. Lalu individu dengan skor terbaik akan dipilih menjadi salah satu *parent*. Proses ini akan diulangi lagi untuk mendapatkan *parent* kedua.

Setelah didapatkan kedua orang tua, maka akan ada kemungkinan terjadi *crossover* sesuai dengan *crossover rate*. Jika berhasil, maka *parents* akan bersilangan dan membentuk dua *child*. Jika gagal, maka *parents* hanya akan disalin ke generasi berikutnya. Proses persilangan akan dilakukan di *cut point* yang dipilih secara acak.

Setelah *crossover* akan dilakukan mutasi. Setiap *child* yang baru dibuat akan memiliki kemungkinan mutasi. Ada dua cara mutasi yang bisa terjadi yaitu pindah barang dari satu kontainer ke kontainer lain, atau menukar dua barang dalam dua kontainer. Mutasi penting dilakukan untuk memperkenalkan varian baru ke dalam populasi.

Loop akan terus dijalankan hingga mencapai *max_generations*. Setelah selesai, fungsi akan mengembalikan solusi terbaik yang pernah ditemukan selama algoritma bekerja.

```
from __future__ import annotations

import random
from typing import Dict, List, Optional, Sequence, Tuple

from src.core.data_structures import Barang, Kontainer,
State
from src.core.objective_function import ObjectiveConfig,
calculate_objective
from src.core.initial_state import generate_random_state
```

```
from src.utils.state_utils import
renumber_container_ids, extract_all_items,
resolve_capacity


def genetic_algorithm(
    initial_state: State,
    config: ObjectiveConfig,
    *,
    kapasitas_kontainer: Optional[int] = None,
    max_generations: int = 100,
    population_size: int = 30,
    crossover_rate: float = 0.8,
    mutation_rate: float = 0.2,
    tournament_size: int = 3,
    elitism: int = 1,
    rng: Optional[random.Random] = None,
) -> Tuple[State, List[float]]:
    """
    Menjalankan Genetic Algorithm untuk masalah Bin
Packing.

    Args:
        initial_state: State awal yang digunakan sebagai
baseline populasi.
        config: Konfigurasi fungsi objektif.
        kapasitas_kontainer: Kapasitas kontainer
(opsional, akan diambil dari state bila tidak
disediakan).
        max_generations: Jumlah generasi yang
disimulasikan.
        population_size: Jumlah individu dalam populasi.
        crossover_rate: Peluang crossover antar pasangan
orang tua.
        mutation_rate: Peluang mutasi diterapkan pada
individu.
```

```
        tournament_size: Ukuran turnamen untuk seleksi.
        elitism: Jumlah individu terbaik yang dibawa
langsung ke generasi berikutnya.
        rng: Random generator agar eksperimen dapat
direplikasi.

    Returns:
        Pasangan (State terbaik, histori skor terbaik
per generasi).
    """
    if population_size < 2:
        raise ValueError("population_size minimal
bernilai 2 agar GA dapat bekerja.")
    if max_generations < 0:
        raise ValueError("max_generations tidak boleh
negatif.")
    if tournament_size < 1:
        raise ValueError("tournament_size minimal
bernilai 1.")
    if elitism < 0:
        raise ValueError("elitism tidak boleh negatif.")

    rng = rng or random.Random()
    items = extract_all_items(initial_state)
    if not items:
        base_score = calculate_objective(initial_state,
config)
        return initial_state.salin(), [base_score]

    kapasitas = resolve_capacity(initial_state,
kapasitas_kontainer)

    population: List[State] = [initial_state.salin()]
    while len(population) < population_size:
        population.append(generate_random_state(items,
kapasitas, rng))
```

```
    scores = [calculate_objective(individu, config) for
individu in population]
    best_idx = min(range(len(population)), key=lambda
idx: scores[idx])
    best_state = population[best_idx].salin()
    best_score = scores[best_idx]
    history: List[float] = [best_score]

    for _ in range(max_generations):
        new_population: List[State] = []
        if elitism > 0:
            elite_indices = _top_indices(scores,
elitism)
            for idx in elite_indices:

new_population.append(population[idx].salin())

        while len(new_population) < population_size:
            parent1 = _tournament_selection(population,
scores, tournament_size, rng)
            parent2 = _tournament_selection(population,
scores, tournament_size, rng)

            if rng.random() < crossover_rate:
                child1, child2 = _crossover(parent1,
parent2, items, kapasitas, rng)
            else:
                child1, child2 = parent1.salin(),
parent2.salin()

            if rng.random() < mutation_rate:
                _mutate_state(child1, kapasitas, rng)
            if rng.random() < mutation_rate:
                _mutate_state(child2, kapasitas, rng)
```

```
            new_population.append(child1)
            if len(new_population) < population_size:
                new_population.append(child2)

        population = new_population
        scores = [calculate_objective(individu, config)
for individu in population]
        generation_best_idx =
min(range(len(population)), key=lambda idx: scores[idx])
        generation_best_score =
scores[generation_best_idx]

        if generation_best_score < best_score:
            best_score = generation_best_score
            best_state =
population[generation_best_idx].salin()

        history.append(best_score)

    return best_state, history







def _tournament_selection(
    population: Sequence[State],
    scores: Sequence[float],
    tournament_size: int,
    rng: random.Random,
) -> State:
    size = min(tournament_size, len(population))
    kandidat_idx = rng.sample(range(len(population)),
size)
    best_idx = min(kandidat_idx, key=lambda idx:
```

```
scores[idx])
    return population[best_idx]


def _crossover(
    parent1: State,
    parent2: State,
    items: Sequence[Barang],
    kapasitas: int,
    rng: random.Random,
) -> Tuple[State, State]:
    if len(items) < 2:
        return parent1.salin(), parent2.salin()

    assignment1 = _item_assignment(parent1)
    assignment2 = _item_assignment(parent2)
    cut_point = rng.randint(1, len(items) - 1)

    child1_map: Dict[str, Optional[int]] = {}
    child2_map: Dict[str, Optional[int]] = {}
    for idx, barang in enumerate(items):
        if idx < cut_point:
            child1_map[barang.id] =
assignment1.get(barang.id)
            child2_map[barang.id] =
assignment2.get(barang.id)
        else:
            child1_map[barang.id] =
assignment2.get(barang.id)
            child2_map[barang.id] =
assignment1.get(barang.id)

    return (
        _build_state_from_assignment(child1_map, items,
kapasitas),
        _build_state_from_assignment(child2_map, items,
```

```
kapasitas),
    )


def _item_assignment(state: State) -> Dict[str,
Optional[int]]:
    assignment: Dict[str, Optional[int]] = {}
    for kontainer in state.kontainer_list:
        for barang in kontainer.barang_di_dalam:
            assignment[barang.id] = kontainer.id
    for barang in state.barang_belum_dialokasi:
        assignment[barang.id] = None
    return assignment


def _build_state_from_assignment(
    assignment_map: Dict[str, Optional[int]],
    items: Sequence[Barang],
    kapasitas: int,
) -> State:
    grouped: Dict[Optional[int], List[Barang]] = {}
    for barang in items:
        key = assignment_map.get(barang.id)
        grouped.setdefault(key, []).append(barang)

    kontainer_list: List[Kontainer] = []
    for _, group_items in sorted(grouped.items(),
key=lambda item: str(item[0])):
        if not group_items:
            continue
        kontainer = Kontainer(id=len(kontainer_list),
kapasitas=kapasitas, barang_di_dalam=[])
        for barang in group_items:
            if kontainer.bisa_tambah_barang(barang):
                kontainer.barang_di_dalam.append(barang)
            else:
```

```
                kontainer_list.append(kontainer)
                kontainer =
Kontainer(id=len(kontainer_list), kapasitas=kapasitas,
barang_di_dalam=[barang])
        kontainer_list.append(kontainer)

    renumber_container_ids(kontainer_list)
    return State(kontainer_list=kontainer_list,
barang_belum_dialokasi=[])

def _mutate_state(state: State, kapasitas: int, rng:
random.Random) -> None:
    if not state.kontainer_list:
        return

    operasi_pindah = rng.random() < 0.5

    if operasi_pindah:
        kontainer_dengan_barang = [k for k in
state.kontainer_list if k.barang_di_dalam]
        if not kontainer_dengan_barang:
            return
        sumber = rng.choice(kontainer_dengan_barang)
        barang = rng.choice(sumber.barang_di_dalam)
        sumber.barang_di_dalam.remove(barang)

        kandidat = [k for k in state.kontainer_list if k
is not sumber and k.bisa_tambah_barang(barang)]
        if kandidat:
            tujuan = rng.choice(kandidat)
            tujuan.barang_di_dalam.append(barang)
        else:
            kontainer_baru =
Kontainer(id=len(state.kontainer_list),
kapasitas=kapasitas, barang_di_dalam=[barang])
            state.kontainer_list.append(kontainer_baru)
```

```
        if not sumber.barang_di_dalam:
            state.kontainer_list.remove(sumber)
    else:
        if len(state.kontainer_list) < 2:
            return
        kontainer_dengan_barang = [k for k in
state.kontainer_list if k.barang_di_dalam]
        if len(kontainer_dengan_barang) < 2:
            return
        c1, c2 = rng.sample(kontainer_dengan_barang, 2)
        b1 = rng.choice(c1.barang_di_dalam)
        b2 = rng.choice(c2.barang_di_dalam)
        muatan1 = c1.muatan_saat_ini - b1.ukuran +
b2.ukuran
        muatan2 = c2.muatan_saat_ini - b2.ukuran +
b1.ukuran
        if muatan1 <= kapasitas and muatan2 <=
kapasitas:
            c1.barang_di_dalam.remove(b1)
            c2.barang_di_dalam.remove(b2)
            c1.barang_di_dalam.append(b2)
            c2.barang_di_dalam.append(b1)

    _remove_empty_and_renumber(state)


def _remove_empty_and_renumber(state: State) -> None:
    state.kontainer_list = [k for k in
state.kontainer_list if k.barang_di_dalam]
    renumber_container_ids(state.kontainer_list)



def _top_indices(scores: Sequence[float], jumlah: int)
-> List[int]:
```

```
    jumlah = min(jumlah, len(scores))
    return sorted(range(len(scores)), key=lambda idx:
scores[idx])[:jumlah]
```

2.5. Integrasi

2.5.1. main.py

File main.py berlaku sebagai pusat kendali dari project ini. Fungsi pada main.py menyatukan semua komponen lain seperti algoritma, fungsi objektif dan lainnya agar bisa dijalankan secara optimal. Terdapat beberapa komponen penting pada file main.py ini yakni:

1. Inisialisasi Pencatatan Hasil

   Sebelum eksekusi, skrip secara otomatis membuat sebuah file CSV unik di dalam direktori src/results/csv dengan nama berbasis timestamp. Mekanisme ini memastikan bahwa hasil dari setiap eksekusi disimpan secara terpisah, mencegah penumpukan data, dan mempermudah analisis perbandingan performa di kemudian hari.
2. Konfigurasi Fungsi Objektif

   Skrip menginstansiasi ObjectiveConfig berdasarkan argumen baris perintah seperti --enable_fragile dan --enable_incompatible. Hal ini memungkinkan fungsi objektif untuk secara dinamis menerapkan aturan penalti sesuai dengan konfigurasi eksperimen yang diinginkan, membuat evaluasi solusi menjadi fleksibel.
3. Persiapan Data dan Keadaan Awal

   Skrip membuat data masalah (daftar barang dan kapasitas kontainer) dari file JSON yang ditentukan menggunakan fungsi parse_problem. Setelah itu, sebuah keadaan awal (initial_state) dibuat menggunakan metode yang dipilih , yang akan menjadi titik awal bai algoritma untuk memulai proses optimisasi.
4. Eksekusi Algoritma dan Pengukuran Kinerja

   Bagian ii dari skrip ini adalah blok kondisional yang memanggil fungsi algoritma (simulated_annealing, genetic_algorithm, dll.) sesuai dengan

argumen --algoritma yang diberikan. Waktu eksekusi diukur secara presisi dengan mencatat waktu sistem sebelum dan sesudah pemanggilan fungsi algoritma, yang hasilnya digunakan untuk analisis efisiensi.

5. Pelaporan dan Penyimpanan Hasil: Setelah eksekusi selesai, skrip menampilkan ringkasan solusi akhir di konsol untuk umpan balik langsung. Selanjutnya, sema data relevan dari eksekusi tersebut termasuk parameter yang digunakan, skor awal dan akhir, serta durasi disimpan sebagai satu baris baru di dalam file CSV yang telah disiapkan. Ini menciptakan sebuah catatan eksperimen yang komprehensif dan terstruktur.

```
import argparse
import random
import time
import csv
import os
from datetime import datetime
from typing import List, Optional
from src.core.data_structures import State, Kontainer
from src.core.initial_state import generate_ffd_state,
generate_random_state
from src.core.objective_function import ObjectiveConfig,
calculate_objective
from src.utils.file_parser import parse_problem
from src.algorithms.simulated_annealing import
simulated_annealing
from src.algorithms.genetic_algorithm import genetic_algorithm
from src.algorithms.hill_climbing import (
    steepest_ascent_hill_climbing,
    stochastic_hill_climbing,
    hill_climbing_with_sideways_moves,
    random_restart_hill_climbing
)
```

```
def print_state_summary(state: State, title: str):
    """Mencetak ringkasan keadaan (solusi) ke konsol."""
    print(f"\n--- {title} ---")
    print(f"Total Kontainer Digunakan:
{len(state.kontainer_list)}")
    for i, kontainer in enumerate(sorted(state.kontainer_list,
key=lambda k: k.id)):
        item_ids = [item.id for item in
kontainer.barang_di_dalam]
        print(f"  Kontainer {kontainer.id} (Muatan:
{kontainer.muatan_saat_ini}/{kontainer.kapasitas}): {item_ids}")

def main():
    parser = argparse.ArgumentParser(description="AI Bin
Packaging Solver")
    # Argumen Umum
    parser.add_argument("--algoritma", type=str, required=True,
choices=['sa', 'hc', 'ga'], help="Algoritma yang akan
dijalankan.")
    parser.add_argument("--data_file", type=str, required=True,
help="Path ke file data JSON.")
    parser.add_argument("--seed", type=int, default=None,
help="Seed RNG (opsional) untuk replikasi hasil.")
    parser.add_argument("--run_count", type=int, default=1,
help="Jumlah eksekusi per skenario.")
    parser.add_argument("--max_iter", type=int, default=1000,
help="Jumlah iterasi maksimum (untuk SA, HC). Juga sebagai
fallback untuk max_generasi GA.")
    parser.add_argument("--initial_state_method", type=str,
default='ffd', choices=['ffd', 'random'], help="Metode pembuatan
state awal.")

    # Argumen GA
    parser.add_argument("--max_generasi", type=int, default=None,
```

```
help="Jumlah generasi maksimum untuk GA.")
    parser.add_argument("--populasi_size", type=int, default=30,
help="Ukuran populasi untuk GA.")
    parser.add_argument("--crossover_rate", type=float,
default=0.8, help="Peluang crossover untuk GA.")
    parser.add_argument("--mutation_rate", type=float,
default=0.2, help="Peluang mutasi untuk GA.")
    parser.add_argument("--tournament_size", type=int, default=3,
help="Ukuran turnamen seleksi GA.")
    parser.add_argument("--elitism", type=int, default=1,
help="Jumlah individu elit yang dipertahankan GA.")

    # Argumen SA
    parser.add_argument("--suhu_awal", type=float,
default=1000.0, help="Suhu awal untuk SA.")
    parser.add_argument("--cooling_rate", type=float,
default=0.99, help="Cooling rate untuk SA.")

    # Argumen HC
    parser.add_argument("--hc_variant", type=str,
default='steepest', choices=['steepest', 'stochastic',
'sideways', 'random_restart'], help="Varian Hill Climbing yang
akan digunakan.")
    parser.add_argument("--max_sideways_moves", type=int,
default=10, help="Jumlah maksimum gerakan menyamping untuk varian
'sideways'.")
    parser.add_argument("--num_restarts", type=int, default=5,
help="Jumlah restart untuk varian 'random_restart'.")

    # Argumen Constraint
    parser.add_argument("--enable_fragile", action="store_true",
help="Aktifkan constraint barang rapuh.")
    parser.add_argument("--enable_incompatible",
action="store_true", help="Aktifkan constraint barang tidak
kompatibel.")
```

```
    args = parser.parse_args()

    # Pengaturan File CSV
    results_dir = os.path.join("src", "results", "csv")
    os.makedirs(results_dir, exist_ok=True)
    timestamp_str = datetime.now().strftime("%Y%m%d_%H%M%S")
    csv_filename = os.path.join(results_dir,
f"results_{timestamp_str}.csv")

    csv_header = [
        'timestamp', 'algorithm', 'hc_variant', 'data_file',
'run_id', 'initial_state_method',
        'initial_score', 'final_score', 'duration_seconds',
'iterations', 'num_containers_initial', 'num_containers_final',
        'fragile_enabled', 'incompatible_enabled', 'seed',
'max_iter_generations', 'population_size', 'crossover_rate',
        'mutation_rate', 'tournament_size', 'elitism',
'initial_temp', 'cooling_rate', 'max_sideways_moves',
'num_restarts'
    ]

    with open(csv_filename, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(csv_header)

    print(f"Menyimpan hasil ke: {csv_filename}")

    # Konfigurasi Fungsi Objektif
    obj_config = ObjectiveConfig(
        use_fragile_constraint=args.enable_fragile,
        use_incompatible_constraint=args.enable_incompatible
    )

    # Baca data problem
    try:
        items, container_capacity = parse_problem(args.data_file)
```

```
    except FileNotFoundError:
        print(f"Error: File data tidak ditemukan di
'{args.data_file}'")
        return

    # Jalankan Eksperimen
    for i in range(args.run_count):
        print(f"\n================ RUN {i + 1} / {args.run_count}
================")

        # Pilih metode pembuatan state awal
        rng_for_initial_state = random.Random(args.seed) if
args.seed is not None else random.Random()
        if args.initial_state_method == 'random':
            keadaan_awal = generate_random_state(items,
container_capacity, rng_for_initial_state)
            method_name = "Acak"
        else: # Default ke ffd
            keadaan_awal = generate_ffd_state(items,
container_capacity)
            method_name = "FFD"

        skor_awal = calculate_objective(keadaan_awal, obj_config)
        print_state_summary(keadaan_awal, f"Keadaan Awal
({method_name})")
        print(f"Skor Awal: {skor_awal:.2f}")

        start_time = time.time()
        rng = random.Random(args.seed) if args.seed is not None
else None
        histori_probabilitas: Optional[List[float]] = None
        iterations = 0

        full_algo_name = args.algoritma
        if args.algoritma == 'hc':
            full_algo_name = f"hc_{args.hc_variant}"
```

```
        if args.algoritma == 'sa':
            keadaan_akhir, histori_skor, histori_probabilitas =
simulated_annealing(
                keadaan_awal=keadaan_awal,
                suhu_awal=args.suhu_awal,
                cooling_rate=args.cooling_rate,
                max_iter=args.max_iter,
                config=obj_config
            )
            iterations = len(histori_skor) -1
        elif args.algoritma == 'hc':
            print(f"\nMenjalankan Hill Climbing (Varian:
{args.hc_variant})...")
            if args.hc_variant == 'steepest':
                keadaan_akhir, histori_skor =
steepest_ascent_hill_climbing(
                    initial_state=keadaan_awal,
config=obj_config, max_iter=args.max_iter
                )
            elif args.hc_variant == 'stochastic':
                keadaan_akhir, histori_skor =
stochastic_hill_climbing(
                    initial_state=keadaan_awal,
config=obj_config, max_iter=args.max_iter
                )
            elif args.hc_variant == 'sideways':
                keadaan_akhir, histori_skor =
hill_climbing_with_sideways_moves(
                    initial_state=keadaan_awal,
config=obj_config, max_iter=args.max_iter,
max_sideways_moves=args.max_sideways_moves
                )
            elif args.hc_variant == 'random_restart':
                keadaan_akhir, histori_skor =
random_restart_hill_climbing(
```

```
                    initial_state=keadaan_awal,
                    config=obj_config,
                    num_restarts=args.num_restarts,
                    max_iter_per_restart=args.max_iter,
                    rng=rng,
                    kapasitas_kontainer=container_capacity
                )
            else:
                print(f"Error: Varian Hill Climbing
'{args.hc_variant}' tidak dikenal.")
                return
            iterations = len(histori_skor) - 1

        elif args.algoritma == 'ga':
            max_generasi = args.max_generasi if args.max_generasi
is not None else args.max_iter
            keadaan_akhir, histori_skor = genetic_algorithm(
                initial_state=keadaan_awal,
                config=obj_config,
                kapasitas_kontainer=container_capacity,
                max_generations=max_generasi,
                population_size=args.populasi_size,
                crossover_rate=args.crossover_rate,
                mutation_rate=args.mutation_rate,
                tournament_size=args.tournament_size,
                elitism=args.elitism,
                rng=rng,
            )
            iterations = len(histori_skor) - 1
        else:
            print(f"Algoritma '{args.algoritma}' tidak dikenal.")
            return

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

```
        skor_akhir = calculate_objective(keadaan_akhir,
obj_config)
        print_state_summary(keadaan_akhir, f"Keadaan Akhir
({full_algo_name.upper()})")
        print(f"Skor Akhir: {skor_akhir:.2f}")
        print(f"Durasi Eksekusi: {durasi:.4f} detik")

        # Simpan hasil ke CSV
        with open(csv_filename, 'a', newline='') as f:
            writer = csv.writer(f)
            row_data = [
                datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
full_algo_name, args.hc_variant if args.algoritma == 'hc' else
'N/A',
                os.path.basename(args.data_file), i + 1,
args.initial_state_method, f"{skor_awal:.4f}",
f"{skor_akhir:.4f}",
                f"{durasi:.4f}", iterations,
len(keadaan_awal.kontainer_list),
len(keadaan_akhir.kontainer_list),
                args.enable_fragile, args.enable_incompatible,
args.seed,
                args.max_iter if args.algoritma != 'ga' else
args.max_generasi,
                args.populasi_size if args.algoritma == 'ga' else
'N/A',
                args.crossover_rate if args.algoritma == 'ga'
else 'N/A',
                args.mutation_rate if args.algoritma == 'ga' else
'N/A',
                args.tournament_size if args.algoritma == 'ga'
else 'N/A',
                args.elitism if args.algoritma == 'ga' else
'N/A',
                args.suhu_awal if args.algoritma == 'sa' else
'N/A',
```

```
                args.cooling_rate if args.algoritma == 'sa' else
'N/A',
                args.max_sideways_moves if args.algoritma == 'hc'
and args.hc_variant == 'sideways' else 'N/A',
                args.num_restarts if args.algoritma == 'hc' and
args.hc_variant == 'random_restart' else 'N/A'
            ]
            writer.writerow(row_data)

if __name__ == "__main__":
    main()
```

2.6. Metodologi Eksperimen

Eksperimen dilakukan untuk mengevaluasi dan membandingkan kinerja berbagai algoritma pencarian lokal dan metaheuristik dalam menyelesaikan masalah Bin Packing Problem (BPP) dengan batasan tambahan. Setiap skenario eksperimen dijalankan tiga kali dengan initial state yang berbeda untuk mengumpulkan data kinerja

2.6.1. Data Masalah

Seluruh eksperimen dalam penelitian ini menggunakan satu set data masalah yang konsisten, yang terdapat dalam file **problem.json.** File ini secara spesifik menguraikan konfigurasi masalah BPP sebagai berikut:

- Kapasitas Kontainer: Setiap kontainer memiliki kapasitas maksimum sebesar 150 unit. Asumsi ini berlaku seragam untuk semua kontainer yang digunakan.
- Daftar Barang: Terdapat total 48 barang yang harus ditempatkan ke dalam kontainer. Setiap barang memiliki atribut unik yang mencakup:
    - Id: Pengenal unik untuk setiap barang (misalnya, “BRG001”).
    - Ukuran: Dimensi atau berat barang yang menentukan seberapa banyak ruang yang dibutuhkan dalam kontainer.
    - Tipe: Kategori barang (misalnya, “makanan”, “elektronik”, “kimia”) yang relevan untuk penerapan batas inkompatibilitas.

- Rapuh: sebuah nilai boolean (true/false) yang menunjukkan apakah barang tersebut rapuh, relevan untuk penerapan batasan kerapuhan.

Konten dari **problem.json** secara lengkap adalah sebagai berikut:

```
{
  "kapasitas_kontainer": 150,
  "barang": [
    { "id": "BRG001", "ukuran": 37, "tipe": "kimia" },
    { "id": "BRG002", "ukuran": 54, "tipe":
"elektronik", "rapuh": true },
    { "id": "BRG003", "ukuran": 19, "tipe": "buku" },
    { "id": "BRG004", "ukuran": 88, "tipe": "makanan" },
    { "id": "BRG005", "ukuran": 63, "tipe": "pakaian" },
    { "id": "BRG006", "ukuran": 28, "tipe": "perabot" },
    { "id": "BRG007", "ukuran": 46, "tipe": "makanan" },
    { "id": "BRG008", "ukuran": 33, "tipe": "kimia" },
    { "id": "BRG009", "ukuran": 21, "tipe": "obat" },
    { "id": "BRG010", "ukuran": 12, "tipe": "perabot" },
    { "id": "BRG011", "ukuran": 41, "tipe": "buku",
"rapuh": true },
    { "id": "BRG012", "ukuran": 30, "tipe": "pakaian" },
    { "id": "BRG013", "ukuran": 18, "tipe": "kimia" },
    { "id": "BRG014", "ukuran": 65, "tipe": "elektronik"
},
    { "id": "BRG015", "ukuran": 92, "tipe": "makanan" },
    { "id": "BRG016", "ukuran": 24, "tipe": "kimia" },
    { "id": "BRG017", "ukuran": 16, "tipe": "makanan" },
    { "id": "BRG018", "ukuran": 50, "tipe": "pakaian" },
    { "id": "BRG019", "ukuran": 47, "tipe": "elektronik"
},
    { "id": "BRG020", "ukuran": 26, "tipe": "buku" },
    { "id": "BRG021", "ukuran": 9, "tipe": "obat" },
    { "id": "BRG022", "ukuran": 44, "tipe": "perabot" },
    { "id": "BRG023", "ukuran": 34, "tipe": "kimia" },
    { "id": "BRG024", "ukuran": 56, "tipe": "makanan" },
```

```
    { "id": "BRG025", "ukuran": 70, "tipe": "kimia" },
    { "id": "BRG026", "ukuran": 52, "tipe":
"elektronik", "rapuh": true },
    { "id": "BRG027", "ukuran": 29, "tipe": "makanan" },
    { "id": "BRG028", "ukuran": 38, "tipe": "perabot" },
    { "id": "BRG029", "ukuran": 14, "tipe": "kimia" },
    { "id": "BRG030", "ukuran": 7, "tipe": "obat" },
    { "id": "BRG031", "ukuran": 5, "tipe": "makanan" },
    { "id": "BRG032", "ukuran": 48, "tipe": "elektronik"
},
    { "id": "BRG033", "ukuran": 31, "tipe": "pakaian" },
    { "id": "BRG034", "ukuran": 22, "tipe": "elektronik"
},
    { "id": "BRG035", "ukuran": 110, "tipe": "kimia" },
    { "id": "BRG036", "ukuran": 27, "tipe": "pakaian",
"rapuh": true },
    { "id": "BRG037", "ukuran": 39, "tipe": "makanan",
"rapuh": true },
    { "id": "BRG038", "ukuran": 43, "tipe": "buku" },
    { "id": "BRG039", "ukuran": 32, "tipe": "obat" },
    { "id": "BRG040", "ukuran": 25, "tipe": "perabot" },
    { "id": "BRG041", "ukuran": 11, "tipe": "kimia" },
    { "id": "BRG042", "ukuran": 20, "tipe": "pakaian" },
    { "id": "BRG043", "ukuran": 36, "tipe": "elektronik"
},
    { "id": "BRG044", "ukuran": 45, "tipe": "perabot" },
    { "id": "BRG045", "ukuran": 23, "tipe": "makanan" },
    { "id": "BRG046", "ukuran": 17, "tipe": "buku" },
    { "id": "BRG047", "ukuran": 13, "tipe": "perabot" },
    { "id": "BRG048", "ukuran": 6, "tipe": "makanan" }
  ]
}
```

2.6.2. Konfigurasi Batasan (Constraints)

semua eksperimen diaktifkan dengan dua batasan bonus yang memengaruhi perhitungan fungsi objektif:

a. **Barang Rapuh (Fragile Items):** Batasan ini memberikan penalti yang signifikan jika sebuah barang yang ditandai sebagai rapuh=True ditempatkan dalam kontainer yang sama dengan barang lain yang total ukurannya melebihi ambang batas 50 unit. Penalti ini dirancang untuk mencegah penumpukan berlebihan pada barang rapuh.

b. **Barang Tidak Kompatibel (Incompatible Items):** Batasan ini memberlakukan penalti besar jika barang-barang dengan atribut tipe yang telah ditentukan sebagai tidak kompatibel (secara spesifik, pasangan 'makanan' dan 'kimia') ditempatkan dalam kontainer yang sama.

2.6.3. Metode Inisialisasi Keadaan Awal (Initial State)

Eksperimen dilakukan untuk mengevaluasi dan membandingkan kinerja berbagai algoritma pencarian lokal dan metaheuristik dalam menyelesaikan masalah Bin Packing Problem (BPP) dengan batasan tambahan. Setiap skenario eksperimen dijalankan tiga kali dengan initial state yang berbeda untuk mengumpulkan data kinerja

a. **Acak (Seed 42):** Keadaan awal dihasilkan dengan menempatkan barang-barang secara acak ke dalam kontainer yang ada atau kontainer baru. Penggunaan random seed 42 memastikan bahwa urutan penempatan acak ini dapat direproduksi, memungkinkan perbandingan yang konsisten antar eksperimen.

b. **Acak (Seed 1337):** Mirip dengan metode acak sebelumnya, namun menggunakan random seed 1337. Ini menyediakan titik awal acak yang berbeda, memungkinkan evaluasi bagaimana variasi dalam keadaan awal acak memengaruhi jalur pencarian dan hasil akhir algoritma.

c. **First Fit Decreasing (FFD)**: Ini adalah heuristik greedy di mana barang-barang diurutkan berdasarkan ukurannya dari terbesar ke terkecil. Setiap barang kemudian ditempatkan ke dalam kontainer pertama yang memiliki ruang yang cukup. Jika tidak ada kontainer yang muat, kontainer baru akan dibuat. Metode ini cenderung menghasilkan solusi awal yang relatif padat.

2.6.4. Algoritma yang Diuji

Penelitian ini mengimplementasikan dan menguji tiga kategori utama algoritma pencarian, masing-masing dengan beberapa varian:

a. Hill Climbing (HC): Pada algoritma ini diterapkan **max_iter** sebesar 1000 untuk seluruh varian: Steepest Ascent, Stochastic, Sideways Moves (max_sideways_moves=20), dan Random Restart (num_restarts=10)
b. Simulated Annealing (SA): Algoritma ini menerapkan varian default (suhu_awal = 1000.0, cooling_rate = 0.99)
c. Genetic Algorithm (GA): Algoritma ini menggunakan *Tournament Selection (tournament_size = 3),* Crossover menggunakan *Single-Point Crossover* (crossover_rate = 0.8), Mutasi menggunakan operator pemindahan barang acak (mutation_rate bervariasi), Elitism (elitism = 1) dipertahankan. Adapun varian yang diuji adalah: Variasi Jumlah Generasi (populasi_size = 50, mutation_rate = 0.2) max_generasi = 100, 300, 500, Variasi Ukuran Populasi (max_generasi = 200, mutation_rate = 0.2) populasi_size = 20, 50, 100, Variasi Tingkat Mutasi (populasi_size = 50, max_generasi = 200): mutation_rate = 0.1, 0.4

2.6.5. Pencatatan dan Visualisasi Hasil

Untuk setiap eksekusi algoritma, setidaknya informasi berikut dicatat:

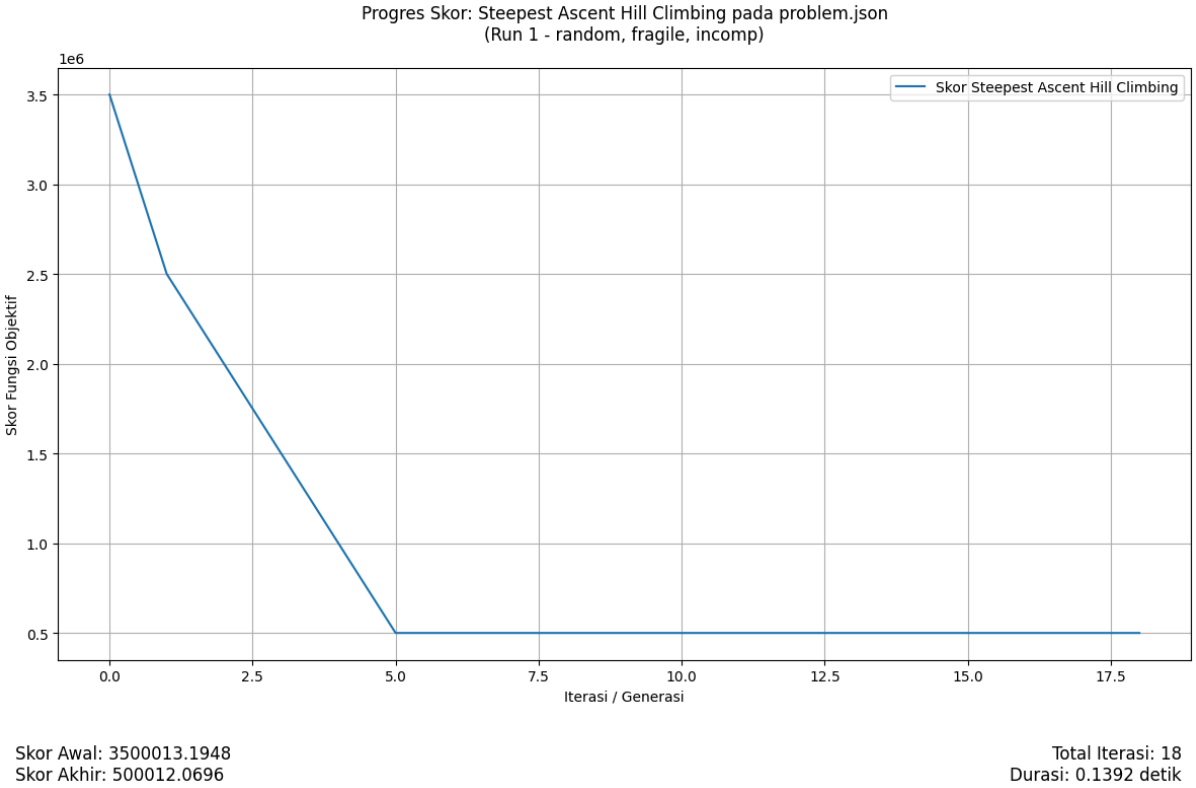
a. Skor fungsi objektif awal dan akhir.
b. Jumlah kontainer awal dan akhir.
c. Durasi eksekusi.
d. Jumlah iterasi/generasi yang dicapai.
e. Histori skor selama proses pencarian

2.7. Hasil Eksperimen

2.7.1. Eksperimen 1

Pada eksperimen pertama, setelah dilakukan inisialisas, didapatkan bahwa initial state cost adalah 3500013,19. Initial state ini akan digunakan pada seluruh algoritma untuk menentukan seberapa dekat algoritma tersebut mendekati global optima.

2.7.1.1 Steepest Ascent Hill-Climbing



Gambar 2.7.1. Hasil Eksperimen Steepest Ascent Hill-Climbing

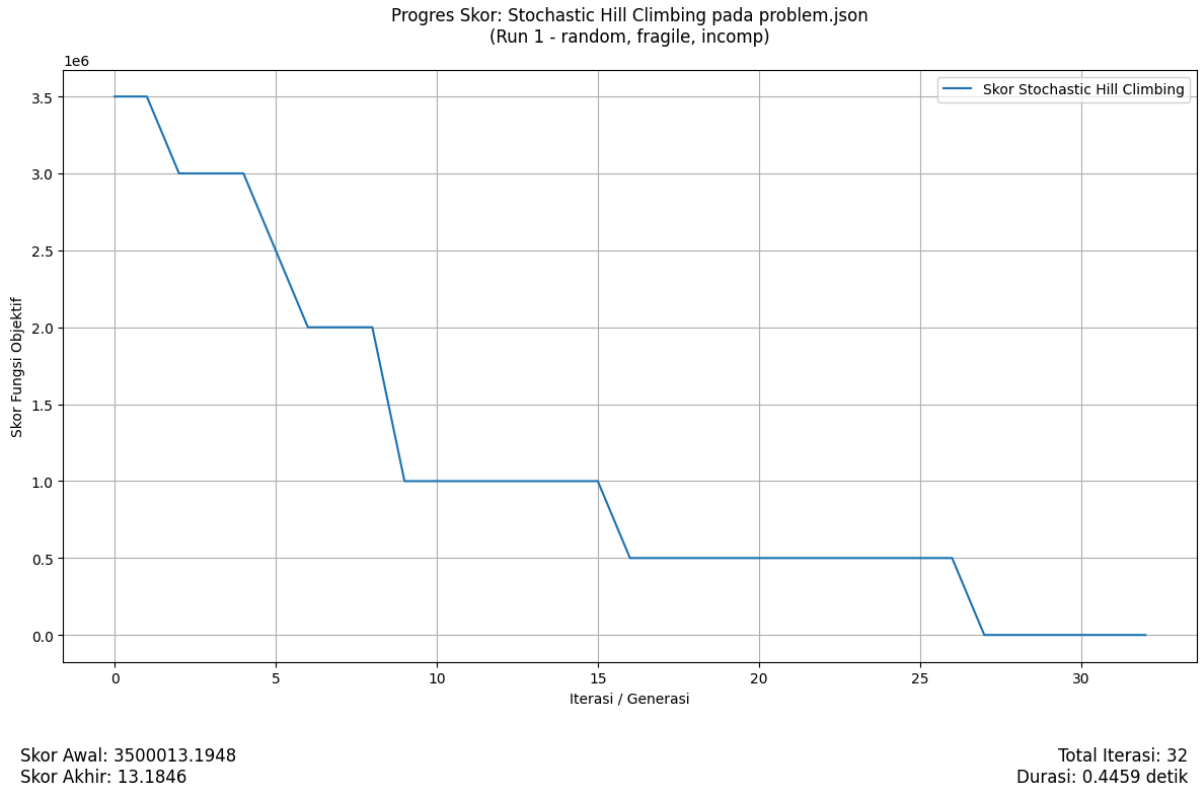
```
RUN 1/1: Menyimpan output CLI ke
src\results\Steepest_Ascent_Hill_Climbing\logs\log_Steep
est_Ascent_Hill_Climbing_problem_random_fragile_incomp_r
un1_20251030_200412.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Steepest Ascent Hill
Climbing
  - Data File             : src/data/problem.json
  - Initial State         : random
  - Iterasi Maks          : 1000
  - Seed                  : 42
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
-------------------------------------
```

```
--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 13
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Steepest Ascent Hill Climbing...

--- Keadaan Akhir (Steepest Ascent Hill Climbing) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG041', 'BRG008',
'BRG002', 'BRG026']
  Kontainer 1 (Muatan: 148/150): ['BRG015', 'BRG009',
'BRG045', 'BRG010']
```

```
  Kontainer 2 (Muatan: 150/150): ['BRG007', 'BRG006',
'BRG005', 'BRG047']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 150/150): ['BRG017', 'BRG020',
'BRG001', 'BRG048', 'BRG023', 'BRG030', 'BRG016']
  Kontainer 6 (Muatan: 150/150): ['BRG031', 'BRG040',
'BRG022', 'BRG019', 'BRG027']
  Kontainer 7 (Muatan: 150/150): ['BRG032', 'BRG042',
'BRG029', 'BRG013', 'BRG018']
  Kontainer 8 (Muatan: 150/150): ['BRG012', 'BRG044',
'BRG024', 'BRG003']
  Kontainer 9 (Muatan: 150/150): ['BRG036', 'BRG011',
'BRG037', 'BRG038']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG033',
'BRG034', 'BRG021']
  Kontainer 11 (Muatan: 70/150): ['BRG025']
Skor Akhir: 500012.07
Durasi Eksekusi: 0.1392 detik
Plot skor disimpan di:
src\results\Steepest_Ascent_Hill_Climbing\plots\Steepest
_Ascent_Hill_Climbing_problem_random_fragile_incomp_run1
_20251030_200412.png
```

Pada algoritma steepest ascent hill-climbing, didapatkan final cost yaitu 500012.07 Seluruh proses ini membutuhkan waktu selama 0.1392 detik dan dengan total iterasi yaitu 18.

2.7.1.2 Stochastic Hill-Climbing



Gambar 2.7.2 Hasil Eksperimen Stochastic Hill-Climbing

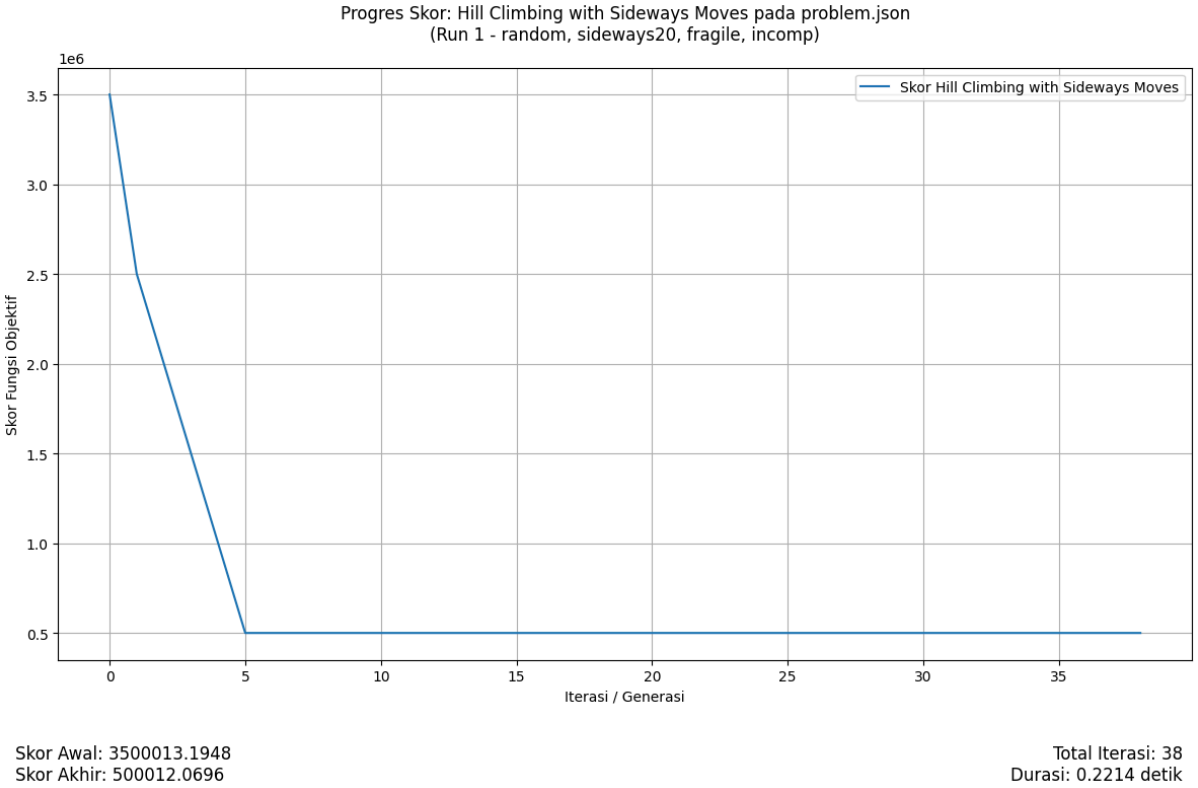
```
RUN 1/1: Menyimpan output CLI ke
src\results\Stochastic_Hill_Climbing\logs\log_Stochastic_Hi
ll_Climbing_problem_random_fragile_incomp_run1_20251030_200
413.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Stochastic Hill Climbing
  - Data File             : src/data/problem.json
  - Initial State         : random
  - Iterasi Maks          : 1000
  - Seed                  : 42
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
----------------------------------------

--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 13
```

```
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Stochastic Hill Climbing...

--- Keadaan Akhir (Stochastic Hill Climbing) ---
Total Kontainer Digunakan: 13
  Kontainer 0 (Muatan: 144/150): ['BRG035', 'BRG023']
  Kontainer 1 (Muatan: 150/150): ['BRG015', 'BRG012',
'BRG006']
  Kontainer 2 (Muatan: 150/150): ['BRG007', 'BRG038',
'BRG027', 'BRG010', 'BRG042']
  Kontainer 3 (Muatan: 57/150): ['BRG016', 'BRG008']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
```

```
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 136/150): ['BRG017', 'BRG020',
'BRG048', 'BRG030', 'BRG028', 'BRG009', 'BRG034']
  Kontainer 6 (Muatan: 150/150): ['BRG031', 'BRG036',
'BRG011', 'BRG002', 'BRG045']
  Kontainer 7 (Muatan: 150/150): ['BRG032', 'BRG003',
'BRG025', 'BRG047']
  Kontainer 8 (Muatan: 150/150): ['BRG044', 'BRG021',
'BRG013', 'BRG019', 'BRG033']
  Kontainer 9 (Muatan: 150/150): ['BRG005', 'BRG040',
'BRG029', 'BRG041', 'BRG001']
  Kontainer 10 (Muatan: 150/150): ['BRG022', 'BRG018',
'BRG024']
  Kontainer 11 (Muatan: 91/150): ['BRG037', 'BRG026']
  Kontainer 12 (Muatan: 88/150): ['BRG004']
Skor Akhir: 13.18
Durasi Eksekusi: 0.4459 detik
Plot skor disimpan di:
src\results\Stochastic_Hill_Climbing\plots\Stochastic_Hill_
Climbing_problem_random_fragile_incomp_run1_20251030_200413
.png
```

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 13.18. Seluruh proses ini membutuhkan waktu selama 0.4459 detik dan dengan total iterasi yaitu 32

2.7.1.3 Hill-Climbing with Sideways Moves



Gambar 2.7.3 Hasil Eksperimen Hill-Climbing with Sideways Moves

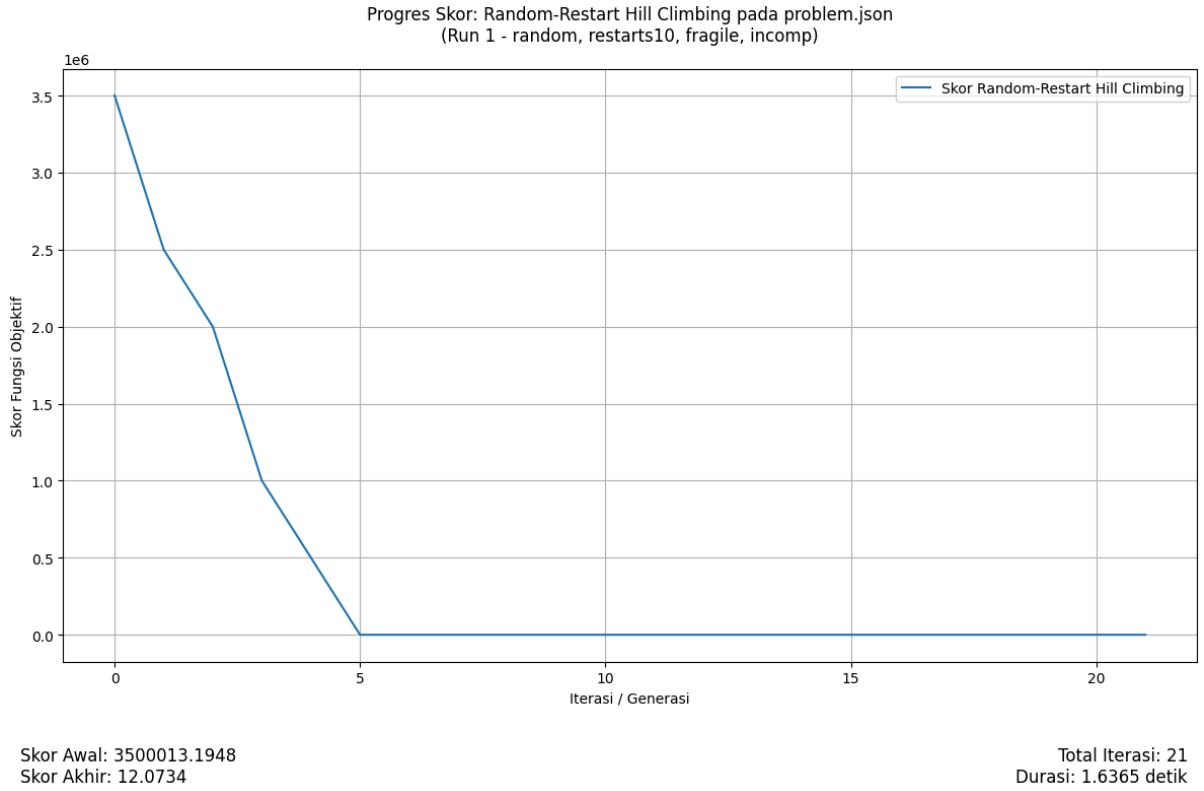
```
    RUN 1/1: Menyimpan output CLI ke
    src\results\Hill_Climbing_with_Sideways_Moves\logs\log
    _Hill_Climbing_with_Sideways_Moves_problem_random_side
    ways20_fragile_incomp_run1_20251030_200415.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Hill Climbing with Sideways
Moves
  - Data File             : src/data/problem.json
  - Initial State         : random
  - Iterasi Maks          : 1000
  - Seed                  : 42
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
-------------------------------------
```

```
--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 13
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Hill Climbing with Sideways Moves...

--- Keadaan Akhir (Hill Climbing with Sideways Moves) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG041', 'BRG008',
'BRG002', 'BRG026']
  Kontainer 1 (Muatan: 148/150): ['BRG015', 'BRG009',
'BRG045', 'BRG010']
  Kontainer 2 (Muatan: 150/150): ['BRG007', 'BRG006',
```

```
'BRG005', 'BRG047']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 150/150): ['BRG017', 'BRG020',
'BRG001', 'BRG048', 'BRG023', 'BRG030', 'BRG016']
  Kontainer 6 (Muatan: 150/150): ['BRG031', 'BRG040',
'BRG022', 'BRG019', 'BRG027']
  Kontainer 7 (Muatan: 150/150): ['BRG032', 'BRG042',
'BRG029', 'BRG013', 'BRG018']
  Kontainer 8 (Muatan: 150/150): ['BRG012', 'BRG044',
'BRG024', 'BRG003']
  Kontainer 9 (Muatan: 150/150): ['BRG036', 'BRG011',
'BRG037', 'BRG038']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG033',
'BRG034', 'BRG021']
  Kontainer 31 (Muatan: 70/150): ['BRG025']
Skor Akhir: 500012.07
Durasi Eksekusi: 0.2214 detik
Plot skor disimpan di:
src\results\Hill_Climbing_with_Sideways_Moves\plots\Hill_Cl
imbing_with_Sideways_Moves_problem_random_sideways20_fragil
e_incomp_run1_20251030_200415.png
```

Pada algoritma Hill-Climbing with Sideways Moves, didapatkan final cost yaitu 500012.0696. Seluruh proses ini membutuhkan waktu selama 0.2214 detik dan dengan total iterasi yaitu 38

2.7.1.4 Random Restart Hill-Climbing



Gambar 2.7.4 Hasil Eksperimen Random Restart Hill-Climbing

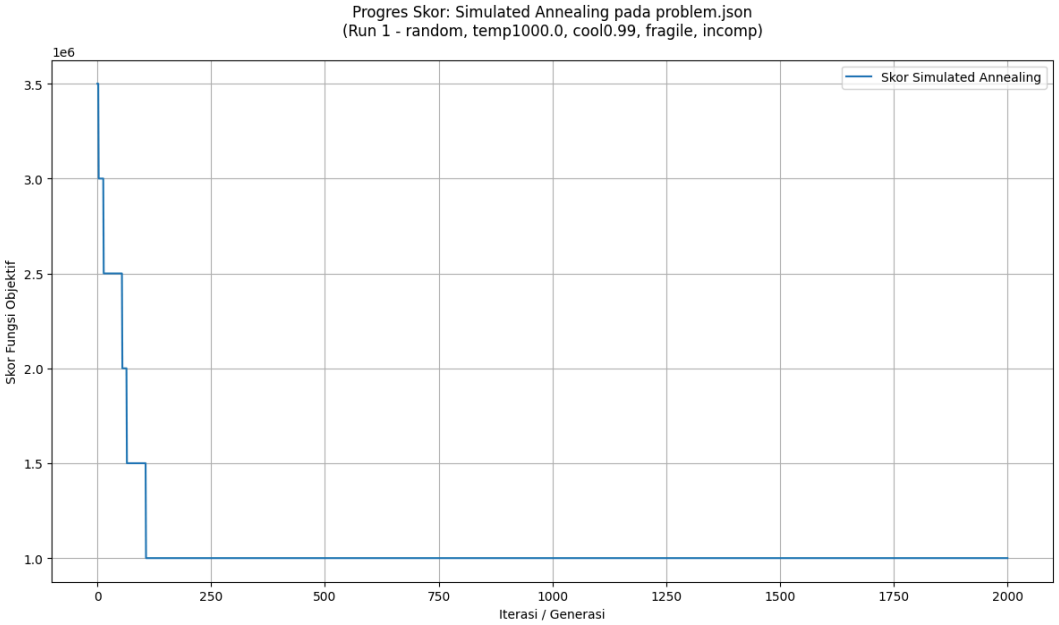
```
RUN 1/1: Menyimpan output CLI ke
src\results\Random-Restart_Hill_Climbing\logs\log_Random-Re
start_Hill_Climbing_problem_random_restarts10_fragile_incom
p_run1_20251030_200416.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Random-Restart Hill Climbing
  - Data File             : src/data/problem.json
  - Initial State         : random
  - Iterasi Maks          : 200
  - Seed                  : 42
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
-------------------------------------
```

```
--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 13
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Random-Restart Hill Climbing...
  Running initial search on the provided start state...
  Restarting search (1/10)...
  Restarting search (2/10)...
  Restarting search (3/10)...
  Restarting search (4/10)...
  Restarting search (5/10)...
  Restarting search (6/10)...
  Restarting search (7/10)...
```

```
  Restarting search (8/10)...
  Restarting search (9/10)...
  Restarting search (10/10)...

--- Keadaan Akhir (Random-Restart Hill Climbing) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 146/150): ['BRG002', 'BRG048',
'BRG026', 'BRG031', 'BRG027']
  Kontainer 1 (Muatan: 150/150): ['BRG037', 'BRG036',
'BRG011', 'BRG042', 'BRG045']
  Kontainer 2 (Muatan: 150/150): ['BRG017', 'BRG007',
'BRG004']
  Kontainer 3 (Muatan: 150/150): ['BRG023', 'BRG033',
'BRG047', 'BRG030', 'BRG014']
  Kontainer 4 (Muatan: 145/150): ['BRG035', 'BRG041',
'BRG016']
  Kontainer 5 (Muatan: 150/150): ['BRG005', 'BRG046',
'BRG025']
  Kontainer 6 (Muatan: 150/150): ['BRG022', 'BRG001',
'BRG020', 'BRG040', 'BRG013']
  Kontainer 7 (Muatan: 77/150): ['BRG024', 'BRG009']
  Kontainer 8 (Muatan: 150/150): ['BRG038', 'BRG021',
'BRG039', 'BRG003', 'BRG008', 'BRG029']
  Kontainer 9 (Muatan: 149/150): ['BRG018', 'BRG019',
'BRG034', 'BRG012']
  Kontainer 10 (Muatan: 150/150): ['BRG032', 'BRG043',
'BRG028', 'BRG006']
  Kontainer 12 (Muatan: 149/150): ['BRG010', 'BRG044',
'BRG015']
Skor Akhir: 12.07
Durasi Eksekusi: 1.6365 detik
Plot skor disimpan di:
src\results\Random-Restart_Hill_Climbing\plots\Random-Resta
rt_Hill_Climbing_problem_random_restarts10_fragile_incomp_r
un1_20251030_200416.png
```

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 12.0734. Seluruh proses ini membutuhkan waktu selama 1.6365 detik dan dengan total iterasi yaitu 21 dan total restart adalah 10.

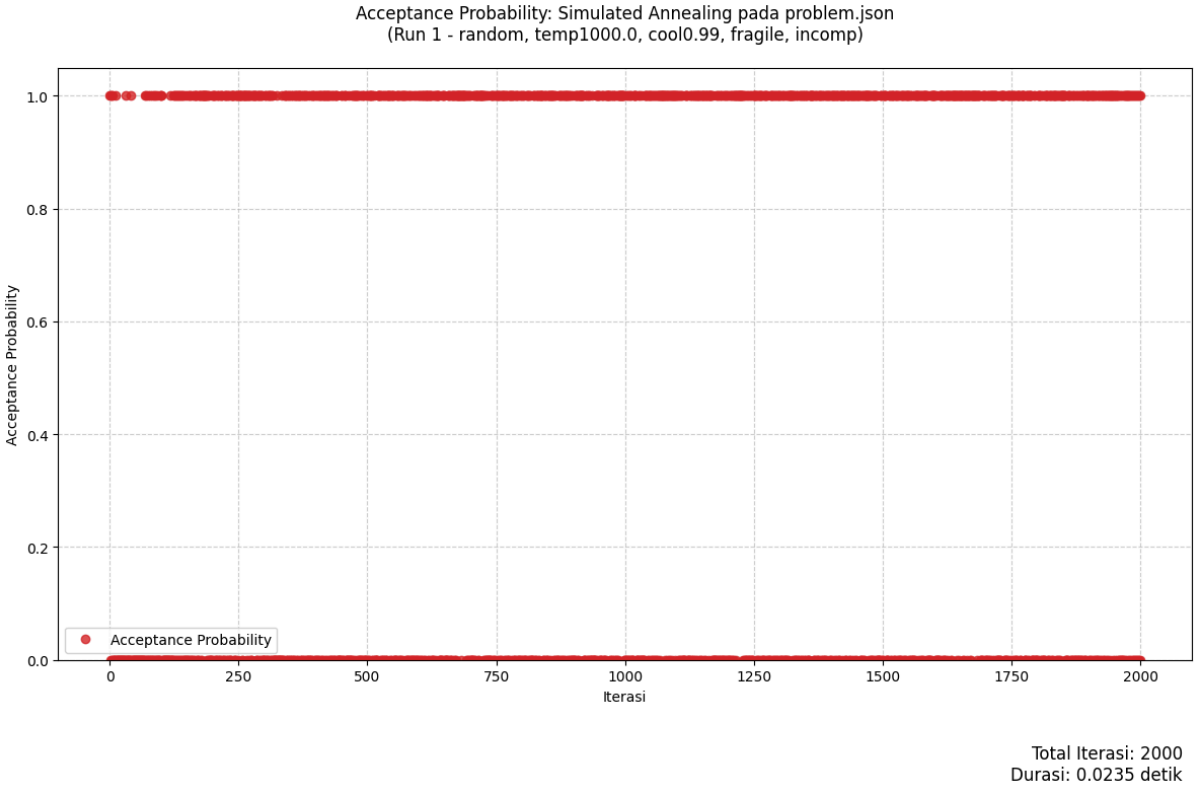
2.7.1.5 Simulated Annealing



Skor Awal: 3500013.1948
Skor Akhir: 1000000.0000

Total Iterasi: 2000
Durasi: 0.0252 detik

Gambar 2.7.5 Progres Skor Simulated Annealing Eksperimen 1



Gambar 2.7.6 Acceptance Probability Simulated Annealing Eksperimen 1

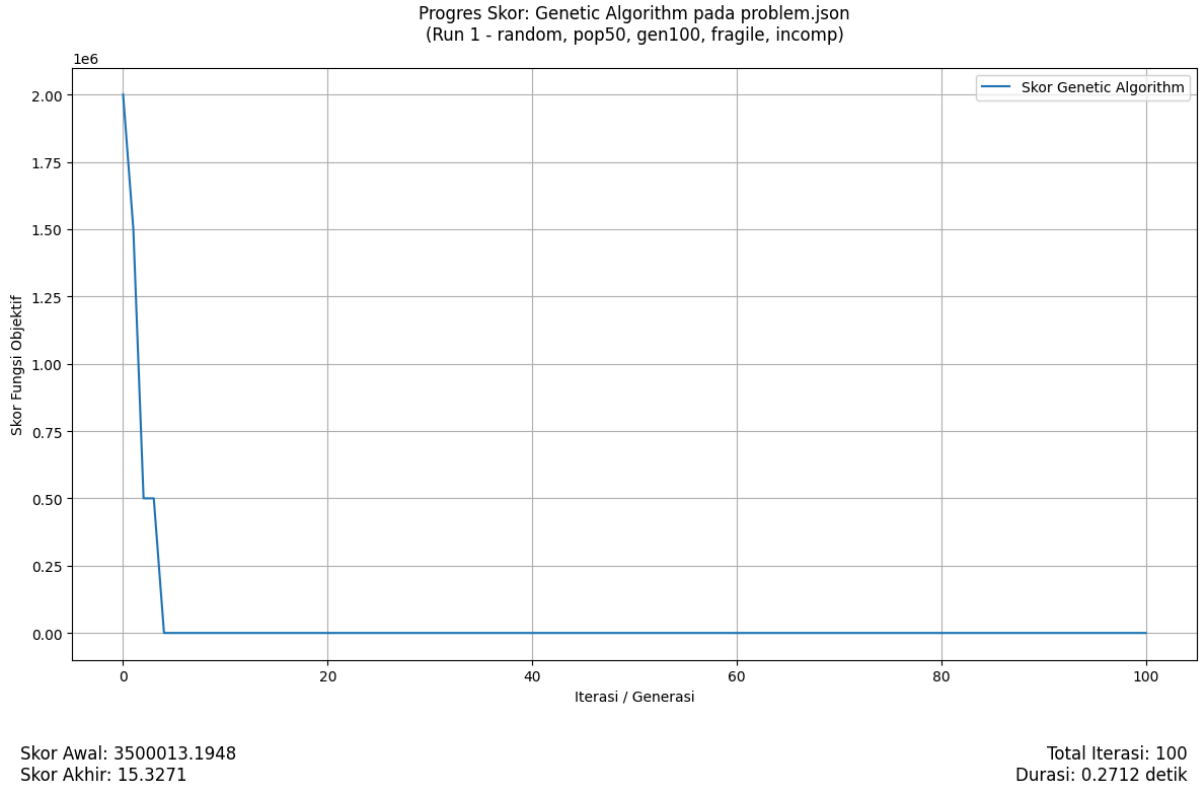
```
RUN 1/1: Menyimpan output CLI ke
src\results\Simulated_Annealing\logs\log_Simulated_Annealin
g_problem_random_temp1000.0_cool0.99_fragile_incomp_run1_20
251030_200418.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Simulated Annealing
  - Data File             : src/data/problem.json
  - Initial State         : random
  - Iterasi Maks          : 2000
  - Seed                  : 42
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
-------------------------------------


--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 13
```

```
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Simulated Annealing...

--- Keadaan Akhir (Simulated Annealing) ---
Total Kontainer Digunakan: 13
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 136/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003']
  Kontainer 3 (Muatan: 121/150): ['BRG018', 'BRG042',
'BRG023', 'BRG046']
  Kontainer 4 (Muatan: 151/150): ['BRG039', 'BRG035',
```

```
'BRG021']
  Kontainer 5 (Muatan: 136/150): ['BRG017', 'BRG020',
'BRG001', 'BRG048', 'BRG030', 'BRG041', 'BRG008']
  Kontainer 6 (Muatan: 146/150): ['BRG033', 'BRG034',
'BRG031', 'BRG004']
  Kontainer 7 (Muatan: 132/150): ['BRG032', 'BRG029',
'BRG013', 'BRG026']
  Kontainer 8 (Muatan: 127/150): ['BRG012', 'BRG027',
'BRG044', 'BRG045']
  Kontainer 9 (Muatan: 139/150): ['BRG040', 'BRG025',
'BRG022']
  Kontainer 10 (Muatan: 130/150): ['BRG043', 'BRG028',
'BRG024']
  Kontainer 11 (Muatan: 140/150): ['BRG014', 'BRG005',
'BRG010']
  Kontainer 14 (Muatan: 115/150): ['BRG011', 'BRG036',
'BRG019']
  Kontainer 15 (Muatan: 93/150): ['BRG037', 'BRG002']
Skor Akhir: 1000000.00
Durasi Eksekusi: 0.0252 detik
Plot skor disimpan di:
src\results\Simulated_Annealing\plots\Simulated_Annealing_p
roblem_random_temp1000.0_cool0.99_fragile_incomp_run1_20251
030_200418_score.png
Plot probabilitas SA disimpan di:
src\results\Simulated_Annealing\plots\Simulated_Annealing_p
roblem_random_temp1000.0_cool0.99_fragile_incomp_run1_20251
030_200418_prob.png
```

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 1000000.0 Seluruh proses ini membutuhkan waktu selama 0.0252 detik dan dengan total iterasi yaitu 2000

2.7.1.6 Genetic Algorithm pop=50, gen=100



Gambar 2.7.7 Progres Skor Genetic Algorithm Pop = 50 Gen = 50

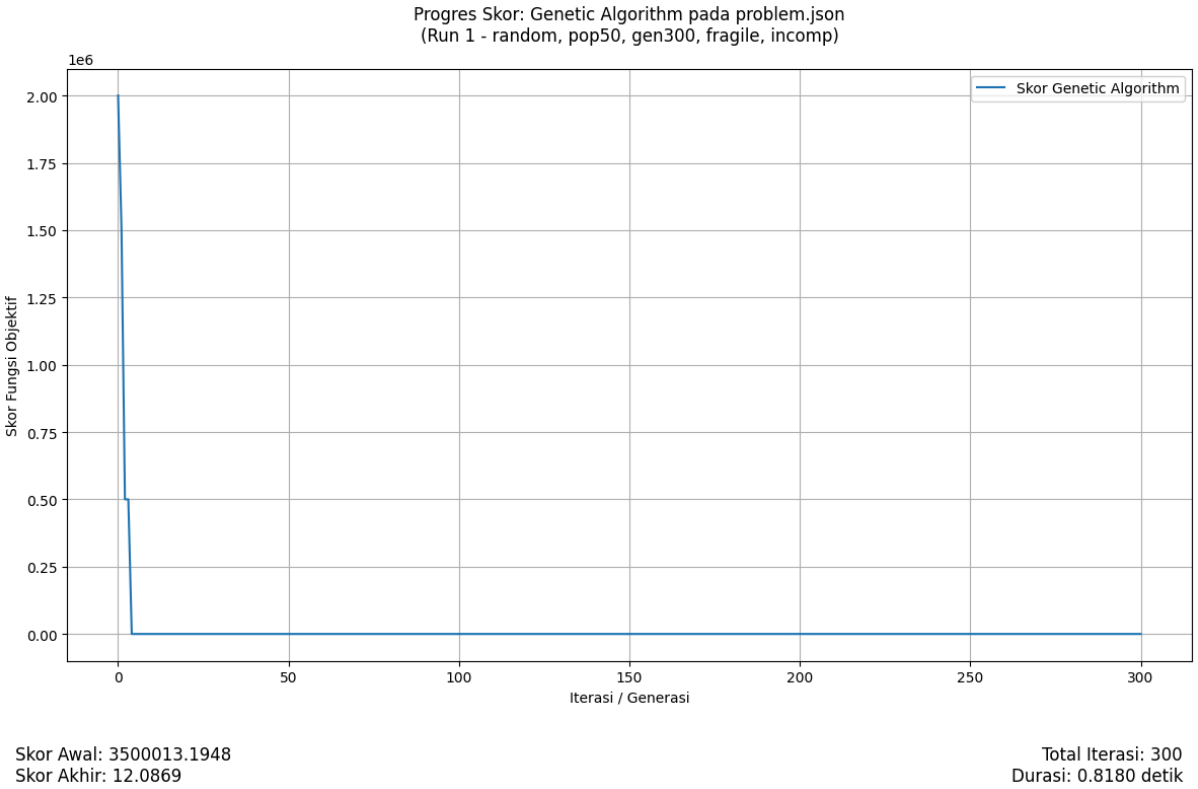
```
RUN 1/1: Menyimpan output CLI ke
src\results\Genetic_Algorithm\logs\log_Genetic_Algorithm_pr
oblem_random_pop50_gen100_fragile_incomp_run1_20251030_2004
21.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Genetic Algorithm
  - Data File             : src/data/problem.json
  - Initial State         : random
  - Generasi Maks         : 100
  - Ukuran Populasi       : 50
  - Seed                  : 42
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
```

```
------------------------------------

--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 13
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Genetic Algorithm...

--- Keadaan Akhir (Genetic Algorithm) ---
Total Kontainer Digunakan: 15
  Kontainer 0 (Muatan: 150/150): ['BRG015', 'BRG043',
'BRG034']
  Kontainer 1 (Muatan: 143/150): ['BRG030', 'BRG032',
```

```
'BRG042', 'BRG010', 'BRG024']
  Kontainer 2 (Muatan: 140/150): ['BRG002', 'BRG044',
'BRG011']
  Kontainer 3 (Muatan: 118/150): ['BRG026', 'BRG037',
'BRG036']
  Kontainer 4 (Muatan: 137/150): ['BRG046', 'BRG039',
'BRG004']
  Kontainer 5 (Muatan: 136/150): ['BRG006', 'BRG017',
'BRG027', 'BRG005']
  Kontainer 6 (Muatan: 147/150): ['BRG003', 'BRG028',
'BRG040', 'BRG014']
  Kontainer 7 (Muatan: 14/150): ['BRG029']
  Kontainer 8 (Muatan: 23/150): ['BRG045']
  Kontainer 9 (Muatan: 48/150): ['BRG038', 'BRG031']
  Kontainer 10 (Muatan: 150/150): ['BRG008', 'BRG047',
'BRG023', 'BRG025']
  Kontainer 11 (Muatan: 140/150): ['BRG035', 'BRG012']
  Kontainer 12 (Muatan: 111/150): ['BRG009', 'BRG007',
'BRG022']
  Kontainer 13 (Muatan: 149/150): ['BRG041', 'BRG018',
'BRG016', 'BRG021', 'BRG001', 'BRG013']
  Kontainer 14 (Muatan: 110/150): ['BRG020', 'BRG048',
'BRG019', 'BRG033']
Skor Akhir: 15.33
Durasi Eksekusi: 0.2712 detik
Plot skor disimpan di:
src\results\Genetic_Algorithm\plots\Genetic_Algorithm_probl
em_random_pop50_gen100_fragile_incomp_run1_20251030_200421.
png
```

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 15.3271 Seluruh proses ini membutuhkan waktu selama 0.2712 detik dan dengan total iterasi yaitu 100

2.7.1.7 Genetic Algorithm pop=50, gen=300



Gambar 2.7.8 Progres Skor Genetic Algorithm Pop = 50 Gen = 300

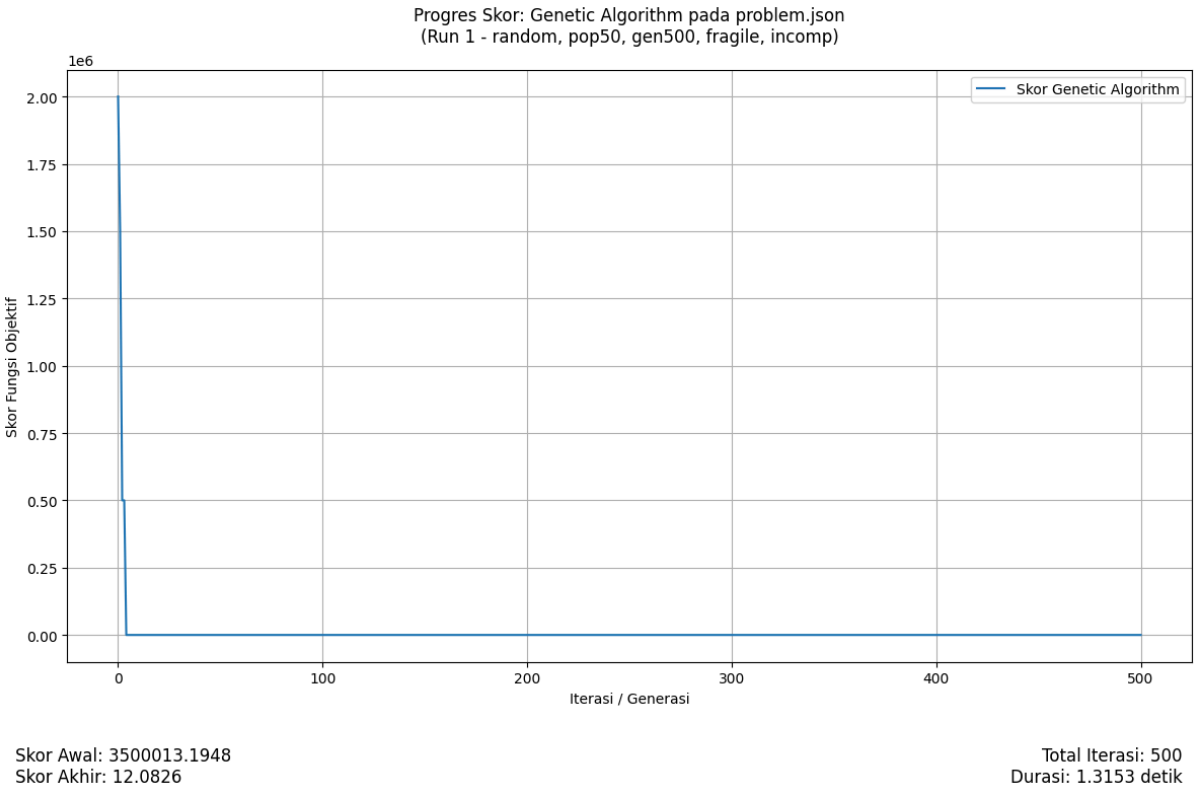
```
RUN 1/1: Menyimpan output CLI ke
src\results\Genetic_Algorithm\logs\log_Genetic_Algorithm_pr
oblem_random_pop50_gen300_fragile_incomp_run1_20251030_2004
22.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Genetic Algorithm
  - Data File             : src/data/problem.json
  - Initial State         : random
  - Generasi Maks         : 300
  - Ukuran Populasi       : 50
  - Seed                  : 42
  - Constraint Rapuh      : Aktif
```

```
  - Constraint Inkompatibel : Aktif
--------------------------------------

--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 13
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Genetic Algorithm...

--- Keadaan Akhir (Genetic Algorithm) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG015', 'BRG043',
'BRG034']
```

```
  Kontainer 1 (Muatan: 150/150): ['BRG030', 'BRG032',
'BRG042', 'BRG010', 'BRG022', 'BRG003']
  Kontainer 2 (Muatan: 117/150): ['BRG028', 'BRG014',
'BRG029']
  Kontainer 3 (Muatan: 150/150): ['BRG008', 'BRG047',
'BRG023', 'BRG025']
  Kontainer 4 (Muatan: 140/150): ['BRG035', 'BRG012']
  Kontainer 5 (Muatan: 149/150): ['BRG009', 'BRG007',
'BRG038', 'BRG017', 'BRG045']
  Kontainer 6 (Muatan: 149/150): ['BRG041', 'BRG018',
'BRG016', 'BRG021', 'BRG001', 'BRG013']
  Kontainer 7 (Muatan: 135/150): ['BRG020', 'BRG048',
'BRG033', 'BRG019', 'BRG040']
  Kontainer 8 (Muatan: 145/150): ['BRG002', 'BRG031',
'BRG044', 'BRG011']
  Kontainer 9 (Muatan: 150/150): ['BRG039', 'BRG026',
'BRG037', 'BRG036']
  Kontainer 10 (Muatan: 133/150): ['BRG006', 'BRG046',
'BRG004']
  Kontainer 11 (Muatan: 148/150): ['BRG027', 'BRG005',
'BRG024']
Skor Akhir: 12.09
Durasi Eksekusi: 0.8180 detik
Plot skor disimpan di:
src\results\Genetic_Algorithm\plots\Genetic_Algorithm_probl
em_random_pop50_gen300_fragile_incomp_run1_20251030_200422.
png
```

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 12.0869 Seluruh proses ini membutuhkan waktu selama 0.8180 detik dan dengan total iterasi yaitu 300

2.7.1.8 Genetic Algorithm pop=50, gen=500



Gambar 2.7.9 Progres Skor Genetic Algorithm Pop = 50 Gen = 500

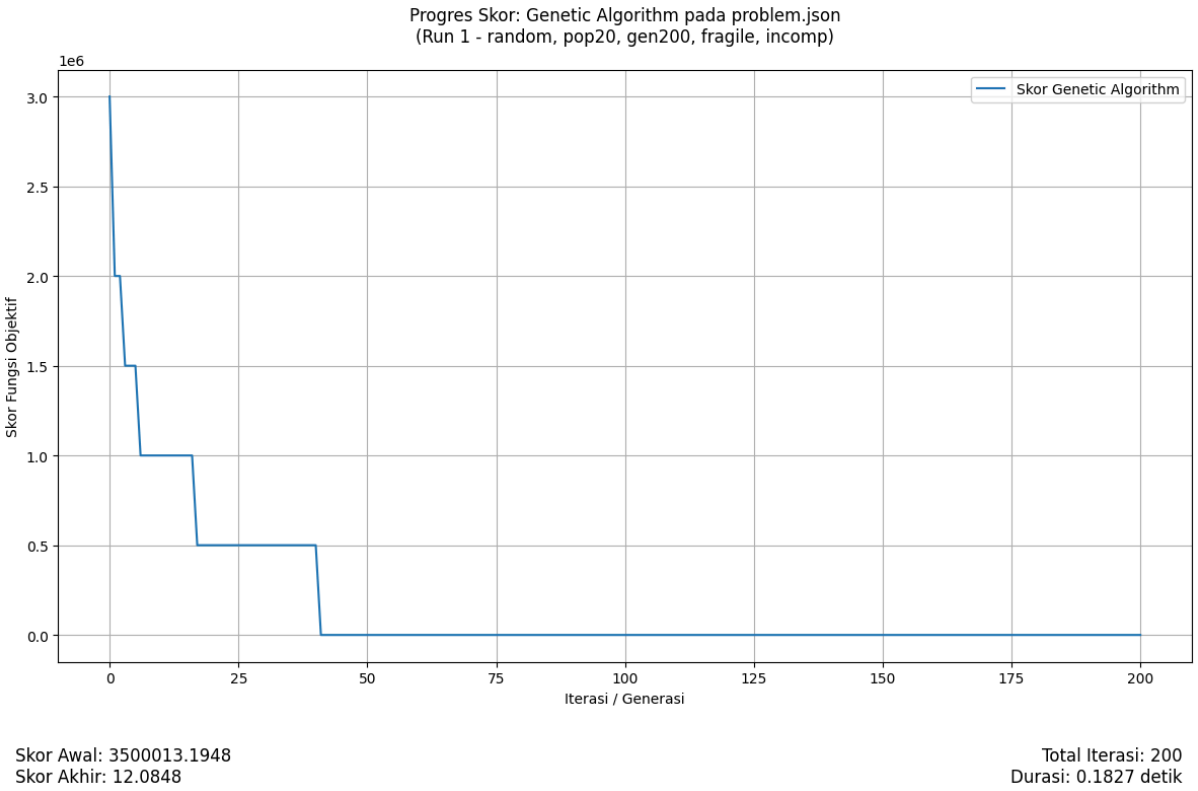
```
RUN 1/1: Menyimpan output CLI ke
src\results\Genetic_Algorithm\logs\log_Genetic_Algorithm_pr
oblem_random_pop50_gen500_fragile_incomp_run1_20251030_2004
24.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Genetic Algorithm
  - Data File             : src/data/problem.json
  - Initial State         : random
  - Generasi Maks         : 500
  - Ukuran Populasi       : 50
  - Seed                  : 42
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
--------------------------------------


--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 13
```

```
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Genetic Algorithm...

--- Keadaan Akhir (Genetic Algorithm) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG015', 'BRG043',
'BRG034']
  Kontainer 1 (Muatan: 150/150): ['BRG003', 'BRG030',
'BRG032', 'BRG042', 'BRG010', 'BRG022']
  Kontainer 2 (Muatan: 133/150): ['BRG006', 'BRG046',
'BRG004']
  Kontainer 3 (Muatan: 148/150): ['BRG027', 'BRG005',
```

```
'BRG024']
  Kontainer 4 (Muatan: 147/150): ['BRG028', 'BRG014',
'BRG029', 'BRG012']
  Kontainer 5 (Muatan: 150/150): ['BRG008', 'BRG047',
'BRG023', 'BRG025']
  Kontainer 6 (Muatan: 141/150): ['BRG035', 'BRG033']
  Kontainer 7 (Muatan: 149/150): ['BRG009', 'BRG007',
'BRG038', 'BRG017', 'BRG045']
  Kontainer 8 (Muatan: 149/150): ['BRG041', 'BRG018',
'BRG016', 'BRG021', 'BRG001', 'BRG013']
  Kontainer 9 (Muatan: 149/150): ['BRG020', 'BRG048',
'BRG019', 'BRG040', 'BRG044']
  Kontainer 10 (Muatan: 100/150): ['BRG002', 'BRG031',
'BRG011']
  Kontainer 11 (Muatan: 150/150): ['BRG039', 'BRG026',
'BRG037', 'BRG036']
Skor Akhir: 12.08
Durasi Eksekusi: 1.3153 detik
Plot skor disimpan di:
src\results\Genetic_Algorithm\plots\Genetic_Algorithm_probl
em_random_pop50_gen500_fragile_incomp_run1_20251030_200424.
png
```

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 13.18Seluruh proses ini membutuhkan waktu selama 0.4459 detik dan dengan total iterasi yaitu 1000

2.7.1.9 Genetic Algorithm pop=20, gen=200



Gambar 2.7.10 Progres Skor Genetic Algorithm Pop = 20 Gen = 200

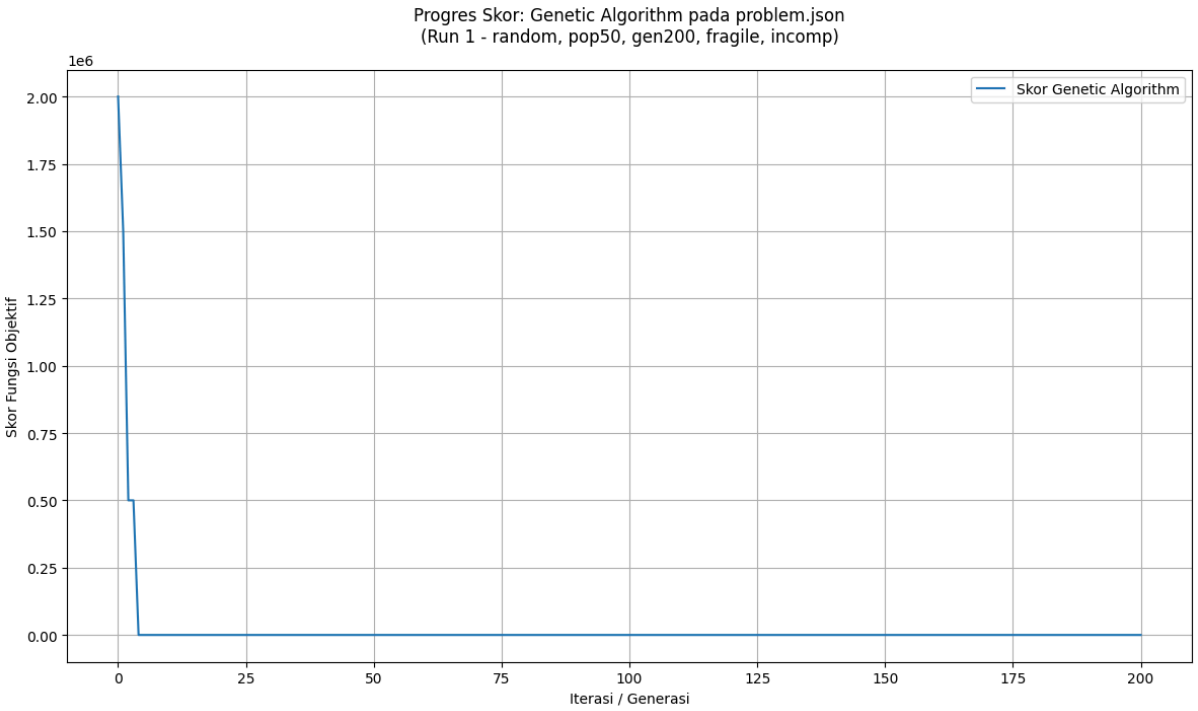
```
RUN 1/1: Menyimpan output CLI ke
src\results\Genetic_Algorithm\logs\log_Genetic_Algorithm_pr
oblem_random_pop20_gen200_fragile_incomp_run1_20251030_2004
26.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Genetic Algorithm
  - Data File             : src/data/problem.json
  - Initial State         : random
  - Generasi Maks         : 200
  - Ukuran Populasi       : 20
  - Seed                  : 42
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
--------------------------------------


--- Keadaan Awal (Acak) ---
```

```
Total Kontainer Digunakan: 13
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Genetic Algorithm...

--- Keadaan Akhir (Genetic Algorithm) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG041', 'BRG018',
'BRG003', 'BRG044', 'BRG040']
  Kontainer 1 (Muatan: 148/150): ['BRG015', 'BRG048',
'BRG042', 'BRG012']
  Kontainer 2 (Muatan: 138/150): ['BRG009', 'BRG027',
'BRG004']
```

```
  Kontainer 3 (Muatan: 149/150): ['BRG007', 'BRG019',
'BRG024']
  Kontainer 4 (Muatan: 147/150): ['BRG020', 'BRG001',
'BRG029', 'BRG025']
  Kontainer 5 (Muatan: 146/150): ['BRG035', 'BRG043']
  Kontainer 6 (Muatan: 147/150): ['BRG045', 'BRG031',
'BRG010', 'BRG005', 'BRG022']
  Kontainer 7 (Muatan: 150/150): ['BRG006', 'BRG038',
'BRG021', 'BRG039', 'BRG017', 'BRG034']
  Kontainer 8 (Muatan: 144/150): ['BRG016', 'BRG028',
'BRG046', 'BRG014']
  Kontainer 9 (Muatan: 105/150): ['BRG008', 'BRG047',
'BRG023', 'BRG013', 'BRG030']
  Kontainer 10 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG036']
  Kontainer 11 (Muatan: 143/150): ['BRG002', 'BRG032',
'BRG011']
Skor Akhir: 12.08
Durasi Eksekusi: 0.1827 detik
Plot skor disimpan di:
src\results\Genetic_Algorithm\plots\Genetic_Algorithm_probl
em_random_pop20_gen200_fragile_incomp_run1_20251030_200426.
png
```

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 12.0848 Seluruh proses ini membutuhkan waktu selama 0.1827 detik dan dengan total iterasi yaitu 200

2.7.1.10 Genetic Algorithm pop=50, gen=200



Skor Awal: 3500013.1948
Skor Akhir: 13.1848

Total Iterasi: 200
Durasi: 0.5647 detik

```
RUN 1/1: Menyimpan output CLI ke
src\results\Genetic_Algorithm\logs\log_Genetic_Algorithm_pr
oblem_random_pop50_gen200_fragile_incomp_run1_20251030_2004
27.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma              : Genetic Algorithm
  - Data File              : src/data/problem.json
  - Initial State          : random
  - Generasi Maks          : 200
  - Ukuran Populasi        : 50
  - Seed                   : 42
  - Constraint Rapuh       : Aktif
  - Constraint Inkompatibel : Aktif
--------------------------------------


--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 13
  Kontainer 0 (Muatan: 148/150): ['BRG041', 'BRG008',
```

```
'BRG002', 'BRG018']
  Kontainer 1 (Muatan: 150/150): ['BRG016', 'BRG015',
'BRG009', 'BRG047']
  Kontainer 2 (Muatan: 145/150): ['BRG007', 'BRG006',
'BRG038', 'BRG003', 'BRG021']
  Kontainer 3 (Muatan: 148/150): ['BRG035', 'BRG028']
  Kontainer 4 (Muatan: 150/150): ['BRG046', 'BRG039',
'BRG014', 'BRG043']
  Kontainer 5 (Muatan: 149/150): ['BRG017', 'BRG020',
'BRG001', 'BRG045', 'BRG048', 'BRG023', 'BRG030']
  Kontainer 6 (Muatan: 149/150): ['BRG026', 'BRG037',
'BRG033', 'BRG034', 'BRG031']
  Kontainer 7 (Muatan: 129/150): ['BRG032', 'BRG019',
'BRG042', 'BRG029']
  Kontainer 8 (Muatan: 116/150): ['BRG012', 'BRG027',
'BRG044', 'BRG010']
  Kontainer 9 (Muatan: 115/150): ['BRG005', 'BRG036',
'BRG040']
  Kontainer 10 (Muatan: 150/150): ['BRG004', 'BRG013',
'BRG022']
  Kontainer 11 (Muatan: 111/150): ['BRG011', 'BRG025']
  Kontainer 12 (Muatan: 56/150): ['BRG024']
Skor Awal: 3500013.19

Menjalankan Genetic Algorithm...

--- Keadaan Akhir (Genetic Algorithm) ---
Total Kontainer Digunakan: 13
  Kontainer 0 (Muatan: 150/150): ['BRG015', 'BRG043',
'BRG034']
  Kontainer 1 (Muatan: 87/150): ['BRG030', 'BRG032',
'BRG042', 'BRG010']
  Kontainer 2 (Muatan: 135/150): ['BRG020', 'BRG048',
'BRG033', 'BRG019', 'BRG040']
  Kontainer 3 (Muatan: 145/150): ['BRG002', 'BRG031',
'BRG044', 'BRG011']
```

```
  Kontainer 4 (Muatan: 146/150): ['BRG026', 'BRG037',
'BRG036', 'BRG006']
  Kontainer 5 (Muatan: 137/150): ['BRG046', 'BRG039',
'BRG004']
  Kontainer 6 (Muatan: 148/150): ['BRG027', 'BRG005',
'BRG024']
  Kontainer 7 (Muatan: 136/150): ['BRG003', 'BRG028',
'BRG014', 'BRG029']
  Kontainer 8 (Muatan: 43/150): ['BRG038']
  Kontainer 9 (Muatan: 150/150): ['BRG008', 'BRG047',
'BRG023', 'BRG025']
  Kontainer 10 (Muatan: 140/150): ['BRG035', 'BRG012']
  Kontainer 11 (Muatan: 150/150): ['BRG009', 'BRG007',
'BRG017', 'BRG045', 'BRG022']
  Kontainer 12 (Muatan: 149/150): ['BRG041', 'BRG018',
'BRG016', 'BRG021', 'BRG001', 'BRG013']
Skor Akhir: 13.18
Durasi Eksekusi: 0.5647 detik
Plot skor disimpan di:
src\results\Genetic_Algorithm\plots\Genetic_Algorithm_probl
em_random_pop50_gen200_fragile_incomp_run1_20251030_200427.
png
```

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 13.18Seluruh proses ini membutuhkan waktu selama 0.5647 detik dan dengan total iterasi yaitu 200

2.7.1.11 Genetic Algorithm pop=100, gen=200

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 13.18Seluruh proses ini membutuhkan waktu selama 0.4459 detik dan dengan total iterasi yaitu 1000

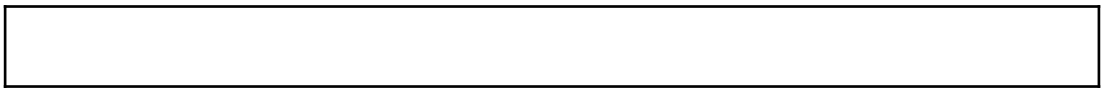
2.7.1.12 Genetic Algorithm pop=50, gen=200, mut=0.1

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 13.18Seluruh proses ini membutuhkan waktu selama 0.4459 detik dan dengan total iterasi yaitu 1000

2.7.1.13 Genetic Algorithm pop=50, gen=200, mut=0.4

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 13.18Seluruh proses ini membutuhkan waktu selama 0.5647 detik dan dengan total iterasi yaitu 1000

2.7.1.14 Genetic Algorithm pop=20, gen=200

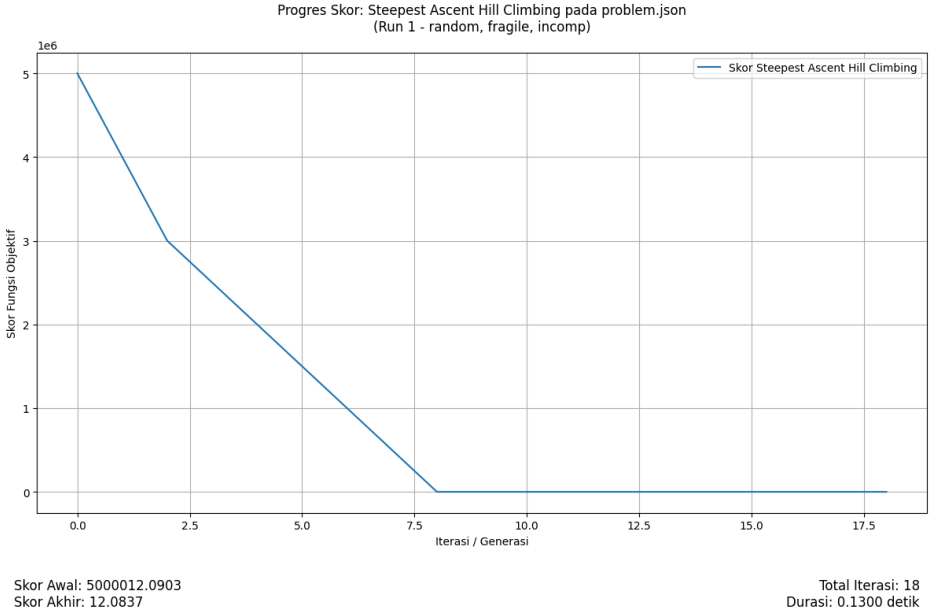


Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 13.18Seluruh proses ini membutuhkan waktu selama 0.4459 detik dan dengan total iterasi yaitu 1000

2.7.2. Eksperimen 2

Pada eksperimen kedua, setelah dilakukan inisialisasi, didapatkan bahwa initial state cost adalah 5000012,09. Initial state ini akan digunakan pada seluruh algoritma untuk menentukan seberapa dekat algoritma tersebut mendekati global optima.

2.7.2.1. Steepest Ascent Hill-Climbing




```
RUN 1/1: Menyimpan output CLI ke
src\results\Steepest_Ascent_Hill_Climbing\logs
\log_Steepest_Ascent_Hill_Climbing_problem_ran
dom_fragile_incomp_run1_20251030_200433.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma              : Steepest Ascent
Hill Climbing
  - Data File              :
src/data/problem.json
  - Initial State          : random
```

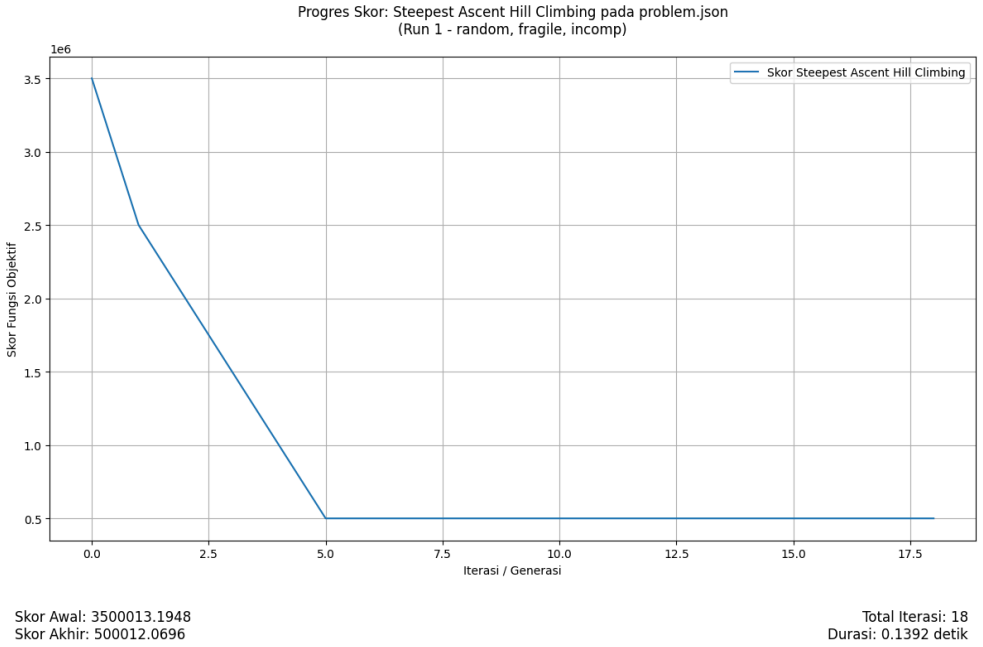
```
  - Iterasi Maks          : 1000
  - Seed                  : 1337
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
--------------------------------------

--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG040',
'BRG035', 'BRG048', 'BRG021']
  Kontainer 1 (Muatan: 147/150): ['BRG046',
'BRG024', 'BRG037', 'BRG045', 'BRG010']
  Kontainer 2 (Muatan: 142/150): ['BRG038',
'BRG011', 'BRG022', 'BRG029']
  Kontainer 3 (Muatan: 148/150): ['BRG025',
'BRG020', 'BRG026']
  Kontainer 4 (Muatan: 143/150): ['BRG014',
'BRG039', 'BRG007']
  Kontainer 5 (Muatan: 147/150): ['BRG028',
'BRG044', 'BRG003', 'BRG017', 'BRG041',
'BRG013']
  Kontainer 6 (Muatan: 143/150): ['BRG023',
'BRG012', 'BRG027', 'BRG001', 'BRG047']
  Kontainer 7 (Muatan: 136/150): ['BRG032',
'BRG004']
  Kontainer 8 (Muatan: 145/150): ['BRG015',
'BRG008', 'BRG042']
  Kontainer 9 (Muatan: 137/150): ['BRG006',
'BRG036', 'BRG016', 'BRG033', 'BRG031',
'BRG034']
  Kontainer 10 (Muatan: 138/150): ['BRG005',
'BRG019', 'BRG009', 'BRG030']
  Kontainer 11 (Muatan: 140/150): ['BRG018',
```

```
'BRG043', 'BRG002']
Skor Awal: 5000012.09

Menjalankan Steepest Ascent Hill Climbing...

--- Keadaan Akhir (Steepest Ascent Hill
Climbing) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 135/150): ['BRG040',
'BRG035']
  Kontainer 1 (Muatan: 150/150): ['BRG024',
'BRG045', 'BRG010', 'BRG017', 'BRG038']
  Kontainer 2 (Muatan: 149/150): ['BRG011',
'BRG026', 'BRG037', 'BRG046']
  Kontainer 3 (Muatan: 150/150): ['BRG025',
'BRG032', 'BRG039']
  Kontainer 4 (Muatan: 146/150): ['BRG014',
'BRG007', 'BRG012', 'BRG031']
  Kontainer 5 (Muatan: 150/150): ['BRG028',
'BRG003', 'BRG041', 'BRG013', 'BRG029',
'BRG018']
  Kontainer 6 (Muatan: 149/150): ['BRG023',
'BRG001', 'BRG008', 'BRG044']
  Kontainer 7 (Muatan: 132/150): ['BRG004',
'BRG022']
  Kontainer 8 (Muatan: 150/150): ['BRG015',
'BRG042', 'BRG027', 'BRG021']
  Kontainer 9 (Muatan: 150/150): ['BRG006',
'BRG033', 'BRG034', 'BRG043', 'BRG030',
'BRG020']
  Kontainer 10 (Muatan: 150/150): ['BRG005',
'BRG019', 'BRG009', 'BRG047', 'BRG048']
  Kontainer 11 (Muatan: 105/150): ['BRG002',
```

```
'BRG036', 'BRG016']
Skor Akhir: 12.08
Durasi Eksekusi: 0.1300 detik
Plot skor disimpan di:
src\results\Steepest_Ascent_Hill_Climbing\plot
s\Steepest_Ascent_Hill_Climbing_problem_random
_fragile_incomp_run1_20251030_200433.png
```

Pada algoritma steepest ascent hill-climbing, didapatkan final cost yaitu 12,08. Seluruh proses ini membutuhkan waktu selama 0.13 detik dan dengan total iterasi yaitu 18.

2.7.2.2.Stochastic Hill Climbing




```
RUN 1/1: Menyimpan output CLI ke
src\results\Steepest_Ascent_Hill_Climbing\logs
\log_Steepest_Ascent_Hill_Climbing_problem_ran
```

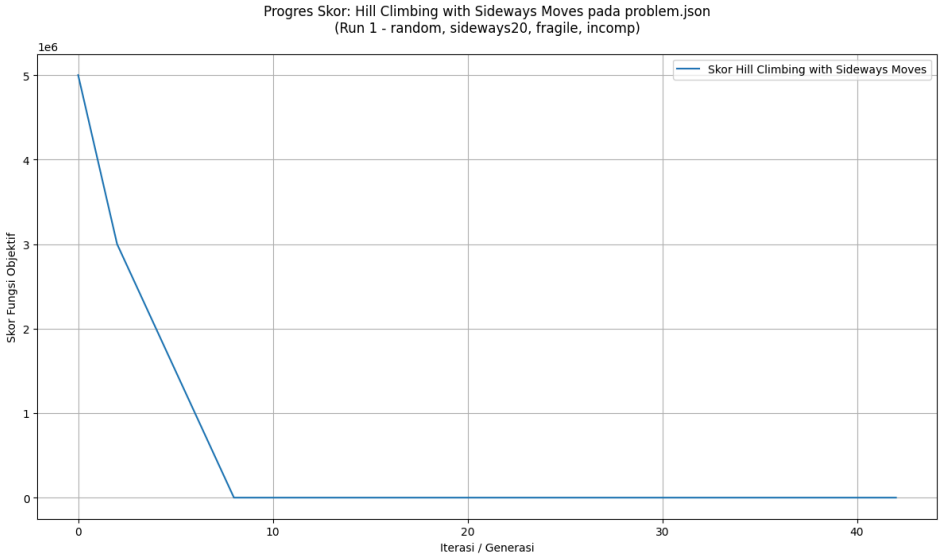
```
dom_fragile_incomp_run1_20251030_200433.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Steepest Ascent
Hill Climbing
  - Data File             :
src/data/problem.json
  - Initial State         : random
  - Iterasi Maks          : 1000
  - Seed                  : 1337
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
--------------------------------------

--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG040',
'BRG035', 'BRG048', 'BRG021']
  Kontainer 1 (Muatan: 147/150): ['BRG046',
'BRG024', 'BRG037', 'BRG045', 'BRG010']
  Kontainer 2 (Muatan: 142/150): ['BRG038',
'BRG011', 'BRG022', 'BRG029']
  Kontainer 3 (Muatan: 148/150): ['BRG025',
'BRG020', 'BRG026']
  Kontainer 4 (Muatan: 143/150): ['BRG014',
'BRG039', 'BRG007']
  Kontainer 5 (Muatan: 147/150): ['BRG028',
'BRG044', 'BRG003', 'BRG017', 'BRG041',
'BRG013']
  Kontainer 6 (Muatan: 143/150): ['BRG023',
'BRG012', 'BRG027', 'BRG001', 'BRG047']
  Kontainer 7 (Muatan: 136/150): ['BRG032',
'BRG004']
```

```
  Kontainer 8 (Muatan: 145/150): ['BRG015',
'BRG008', 'BRG042']
  Kontainer 9 (Muatan: 137/150): ['BRG006',
'BRG036', 'BRG016', 'BRG033', 'BRG031',
'BRG034']
  Kontainer 10 (Muatan: 138/150): ['BRG005',
'BRG019', 'BRG009', 'BRG030']
  Kontainer 11 (Muatan: 140/150): ['BRG018',
'BRG043', 'BRG002']
Skor Awal: 5000012.09

Menjalankan Steepest Ascent Hill Climbing...

--- Keadaan Akhir (Steepest Ascent Hill
Climbing) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 135/150): ['BRG040',
'BRG035']
  Kontainer 1 (Muatan: 150/150): ['BRG024',
'BRG045', 'BRG010', 'BRG017', 'BRG038']
  Kontainer 2 (Muatan: 149/150): ['BRG011',
'BRG026', 'BRG037', 'BRG046']
  Kontainer 3 (Muatan: 150/150): ['BRG025',
'BRG032', 'BRG039']
  Kontainer 4 (Muatan: 146/150): ['BRG014',
'BRG007', 'BRG012', 'BRG031']
  Kontainer 5 (Muatan: 150/150): ['BRG028',
'BRG003', 'BRG041', 'BRG013', 'BRG029',
'BRG018']
  Kontainer 6 (Muatan: 149/150): ['BRG023',
'BRG001', 'BRG008', 'BRG044']
  Kontainer 7 (Muatan: 132/150): ['BRG004',
'BRG022']
```

```
  Kontainer 8 (Muatan: 150/150): ['BRG015',
'BRG042', 'BRG027', 'BRG021']
  Kontainer 9 (Muatan: 150/150): ['BRG006',
'BRG033', 'BRG034', 'BRG043', 'BRG030',
'BRG020']
  Kontainer 10 (Muatan: 150/150): ['BRG005',
'BRG019', 'BRG009', 'BRG047', 'BRG048']
  Kontainer 11 (Muatan: 105/150): ['BRG002',
'BRG036', 'BRG016']
Skor Akhir: 12.08
Durasi Eksekusi: 0.1300 detik
Plot skor disimpan di:
src\results\Steepest_Ascent_Hill_Climbing\plot
s\Steepest_Ascent_Hill_Climbing_problem_random
_fragile_incomp_run1_20251030_200433.png
```

Pada algoritma stochastic hill-climbing, didapatkan final cost yaitu 12,08. Seluruh proses ini membutuhkan waktu selama 0.1300 detik dan dengan total iterasi yaitu 18.

2.7.2.3. Hill Climbing with Sideways Move



Skor Awal: 5000012.0903
Skor Akhir: 12.0794

Total Iterasi: 42
Durasi: 0.2405 detik

```
RUN 1/1: Menyimpan output CLI ke
src\results\Hill_Climbing_with_Sideways_Moves\
logs\log_Hill_Climbing_with_Sideways_Moves_pro
blem_random_sideways20_fragile_incomp_run1_202
51030_200435.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Hill Climbing with
Sideways Moves
  - Data File             :
src/data/problem.json
  - Initial State         : random
  - Iterasi Maks          : 1000
  - Seed                  : 1337
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
-------------------------------------
```

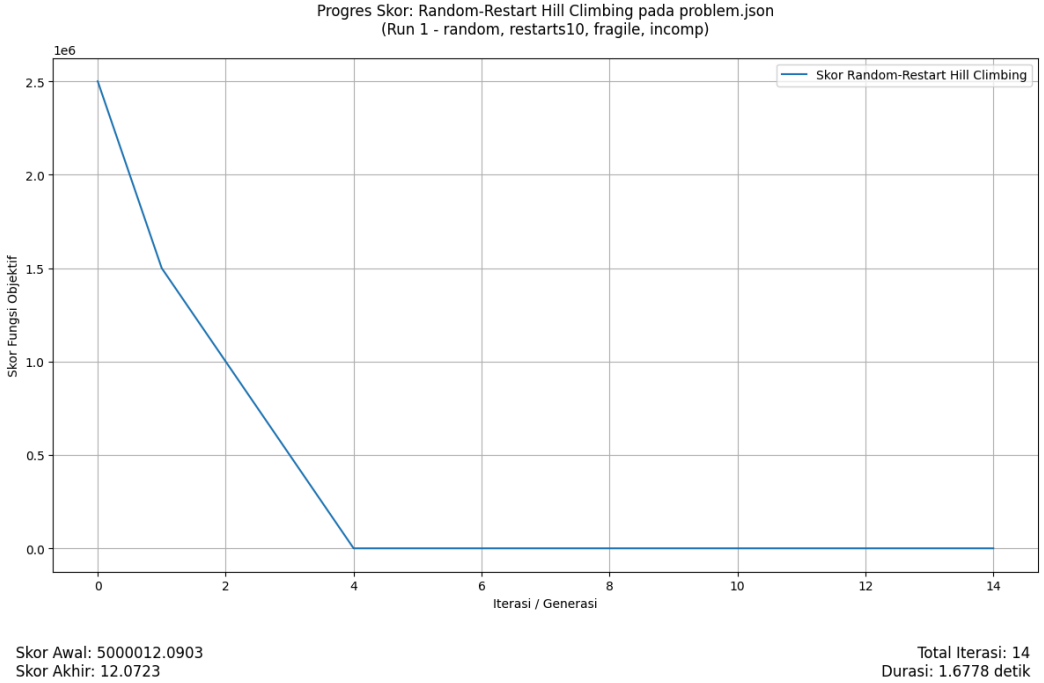
```
--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG040',
'BRG035', 'BRG048', 'BRG021']
  Kontainer 1 (Muatan: 147/150): ['BRG046',
'BRG024', 'BRG037', 'BRG045', 'BRG010']
  Kontainer 2 (Muatan: 142/150): ['BRG038',
'BRG011', 'BRG022', 'BRG029']
  Kontainer 3 (Muatan: 148/150): ['BRG025',
'BRG020', 'BRG026']
  Kontainer 4 (Muatan: 143/150): ['BRG014',
'BRG039', 'BRG007']
  Kontainer 5 (Muatan: 147/150): ['BRG028',
'BRG044', 'BRG003', 'BRG017', 'BRG041',
'BRG013']
  Kontainer 6 (Muatan: 143/150): ['BRG023',
'BRG012', 'BRG027', 'BRG001', 'BRG047']
  Kontainer 7 (Muatan: 136/150): ['BRG032',
'BRG004']
  Kontainer 8 (Muatan: 145/150): ['BRG015',
'BRG008', 'BRG042']
  Kontainer 9 (Muatan: 137/150): ['BRG006',
'BRG036', 'BRG016', 'BRG033', 'BRG031',
'BRG034']
  Kontainer 10 (Muatan: 138/150): ['BRG005',
'BRG019', 'BRG009', 'BRG030']
  Kontainer 11 (Muatan: 140/150): ['BRG018',
'BRG043', 'BRG002']
Skor Awal: 5000012.09

Menjalankan Hill Climbing with Sideways
Moves...
```

```
--- Keadaan Akhir (Hill Climbing with Sideways
Moves) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 134/150): ['BRG035',
'BRG016']
  Kontainer 1 (Muatan: 150/150): ['BRG024',
'BRG045', 'BRG010', 'BRG017', 'BRG038']
  Kontainer 2 (Muatan: 91/150): ['BRG026',
'BRG037']
  Kontainer 3 (Muatan: 150/150): ['BRG025',
'BRG032', 'BRG039']
  Kontainer 4 (Muatan: 146/150): ['BRG014',
'BRG007', 'BRG012', 'BRG031']
  Kontainer 5 (Muatan: 150/150): ['BRG028',
'BRG003', 'BRG041', 'BRG013', 'BRG029',
'BRG018']
  Kontainer 6 (Muatan: 148/150): ['BRG023',
'BRG001', 'BRG008', 'BRG022']
  Kontainer 7 (Muatan: 150/150): ['BRG004',
'BRG046', 'BRG044']
  Kontainer 8 (Muatan: 150/150): ['BRG015',
'BRG042', 'BRG027', 'BRG021']
  Kontainer 9 (Muatan: 150/150): ['BRG006',
'BRG033', 'BRG034', 'BRG043', 'BRG030',
'BRG020']
  Kontainer 10 (Muatan: 150/150): ['BRG005',
'BRG019', 'BRG009', 'BRG047', 'BRG048']
  Kontainer 11 (Muatan: 147/150): ['BRG002',
'BRG036', 'BRG011', 'BRG040']
Skor Akhir: 12.08
Durasi Eksekusi: 0.2405 detik
Plot skor disimpan di:
src\results\Hill_Climbing_with_Sideways_Moves\
```

```
plots\Hill_Climbing_with_Sideways_Moves_proble
m_random_sideways20_fragile_incomp_run1_202510
30_200435.png
```

Pada algoritma hill-climbing with sideways move, didapatkan final cost yaitu 12,08. Seluruh proses ini membutuhkan waktu selama 0.2405 detik dan dengan total iterasi yaitu 42.

2.7.2.4. **Random Restart Hill-Climbing**




```
RUN 1/1: Menyimpan output CLI ke
src\results\Random-Restart_Hill_Climbing\logs\
log_Random-Restart_Hill_Climbing_problem_rando
m_restarts10_fragile_incomp_run1_20251030_2004
36.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
```

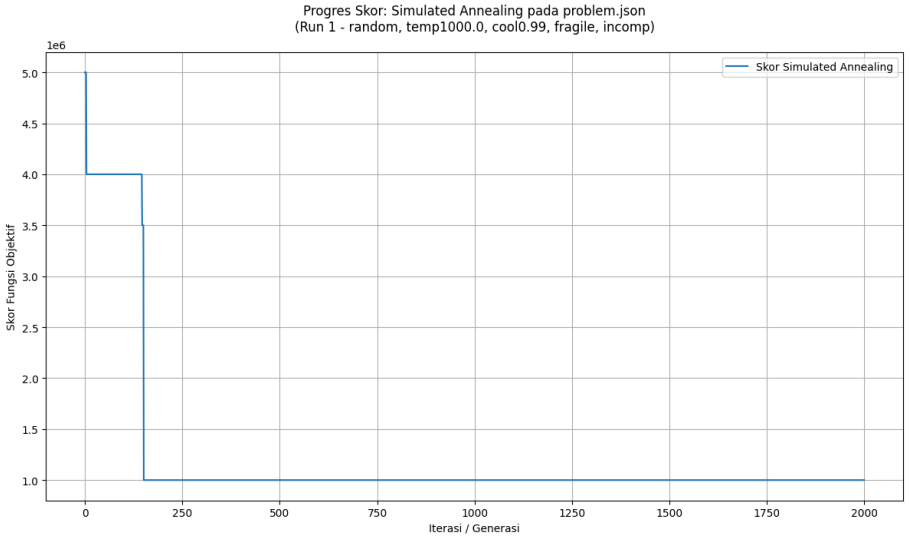
```
  - Algoritma              : Random-Restart
Hill Climbing
  - Data File              :
src/data/problem.json
  - Initial State          : random
  - Iterasi Maks           : 200
  - Seed                   : 1337
  - Constraint Rapuh       : Aktif
  - Constraint Inkompatibel : Aktif
--------------------------------------

--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG040',
'BRG035', 'BRG048', 'BRG021']
  Kontainer 1 (Muatan: 147/150): ['BRG046',
'BRG024', 'BRG037', 'BRG045', 'BRG010']
  Kontainer 2 (Muatan: 142/150): ['BRG038',
'BRG011', 'BRG022', 'BRG029']
  Kontainer 3 (Muatan: 148/150): ['BRG025',
'BRG020', 'BRG026']
  Kontainer 4 (Muatan: 143/150): ['BRG014',
'BRG039', 'BRG007']
  Kontainer 5 (Muatan: 147/150): ['BRG028',
'BRG044', 'BRG003', 'BRG017', 'BRG041',
'BRG013']
  Kontainer 6 (Muatan: 143/150): ['BRG023',
'BRG012', 'BRG027', 'BRG001', 'BRG047']
  Kontainer 7 (Muatan: 136/150): ['BRG032',
'BRG004']
  Kontainer 8 (Muatan: 145/150): ['BRG015',
'BRG008', 'BRG042']
  Kontainer 9 (Muatan: 137/150): ['BRG006',
```

```
'BRG036', 'BRG016', 'BRG033', 'BRG031',
'BRG034']
  Kontainer 10 (Muatan: 138/150): ['BRG005',
'BRG019', 'BRG009', 'BRG030']
  Kontainer 11 (Muatan: 140/150): ['BRG018',
'BRG043', 'BRG002']
Skor Awal: 5000012.09

Menjalankan Random-Restart Hill Climbing...
  Running initial search on the provided start
state...
  Restarting search (1/10)...
  Restarting search (2/10)...
  Restarting search (3/10)...
  Restarting search (4/10)...
  Restarting search (5/10)...
  Restarting search (6/10)...
  Restarting search (7/10)...
  Restarting search (8/10)...
  Restarting search (9/10)...
  Restarting search (10/10)...

--- Keadaan Akhir (Random-Restart Hill
Climbing) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG002',
'BRG037', 'BRG031', 'BRG026']
  Kontainer 1 (Muatan: 75/150): ['BRG011',
'BRG036', 'BRG030']
  Kontainer 2 (Muatan: 150/150): ['BRG014',
'BRG032', 'BRG003', 'BRG013']
  Kontainer 3 (Muatan: 150/150): ['BRG025',
'BRG005', 'BRG046']
```

```
  Kontainer 4 (Muatan: 150/150): ['BRG019',
'BRG028', 'BRG001', 'BRG006']
  Kontainer 5 (Muatan: 149/150): ['BRG039',
'BRG045', 'BRG038', 'BRG020', 'BRG040']
  Kontainer 6 (Muatan: 150/150): ['BRG027',
'BRG017', 'BRG042', 'BRG021', 'BRG007',
'BRG012']
  Kontainer 7 (Muatan: 150/150): ['BRG018',
'BRG022', 'BRG008', 'BRG010', 'BRG041']
  Kontainer 8 (Muatan: 150/150): ['BRG009',
'BRG004', 'BRG047', 'BRG034', 'BRG048']
  Kontainer 9 (Muatan: 150/150): ['BRG043',
'BRG016', 'BRG033', 'BRG044', 'BRG029']
  Kontainer 10 (Muatan: 148/150): ['BRG015',
'BRG024']
  Kontainer 11 (Muatan: 144/150): ['BRG035',
'BRG023']
Skor Akhir: 12.07
Durasi Eksekusi: 1.6778 detik
Plot skor disimpan di:
src\results\Random-Restart_Hill_Climbing\plots
\Random-Restart_Hill_Climbing_problem_random_r
estarts10_fragile_incomp_run1_20251030_200436.
png
```

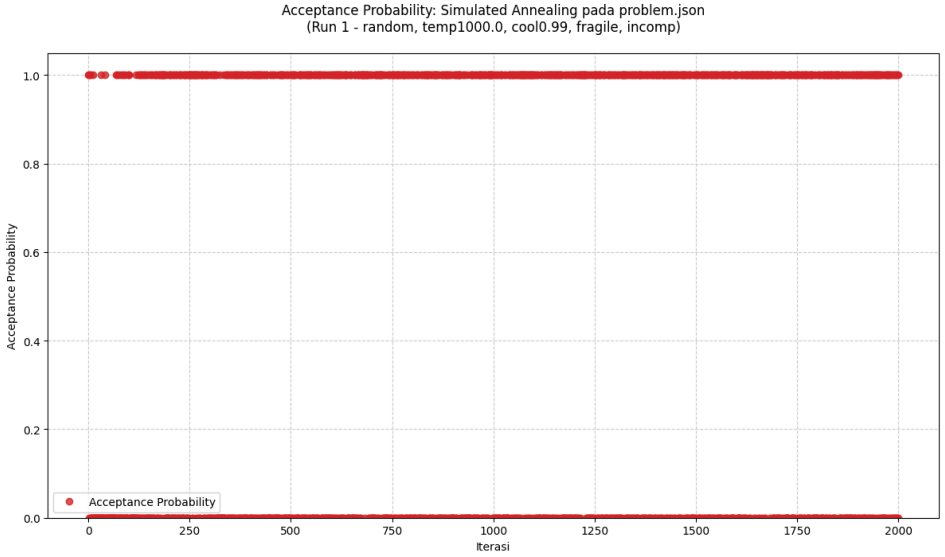
Pada algoritma random restart hill-climbing, didapatkan final cost yaitu 12,08. Seluruh proses ini membutuhkan waktu selama 0.2405 detik dan dengan total iterasi yaitu 42, dan total restart 10.

2.7.2.5. **Simulated Annealing**



Skor Awal: 5000012.0903
Skor Akhir: 1000000.0000

Total Iterasi: 2000
Durasi: 0.0235 detik



Total Iterasi: 2000
Durasi: 0.0235 detik

```
RUN 1/1: Menyimpan output CLI ke
src\results\Simulated_Annealing\logs\log_Simul
ated_Annealing_problem_random_temp1000.0_cool0
.99_fragile_incomp_run1_20251030_200439.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Simulated
Annealing
  - Data File             :
src/data/problem.json
  - Initial State         : random
  - Iterasi Maks          : 2000
```

```
  - Seed                   : 1337
  - Constraint Rapuh       : Aktif
  - Constraint Inkompatibel : Aktif
-------------------------------------

--- Keadaan Awal (Acak) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 150/150): ['BRG040',
'BRG035', 'BRG048', 'BRG021']
  Kontainer 1 (Muatan: 147/150): ['BRG046',
'BRG024', 'BRG037', 'BRG045', 'BRG010']
  Kontainer 2 (Muatan: 142/150): ['BRG038',
'BRG011', 'BRG022', 'BRG029']
  Kontainer 3 (Muatan: 148/150): ['BRG025',
'BRG020', 'BRG026']
  Kontainer 4 (Muatan: 143/150): ['BRG014',
'BRG039', 'BRG007']
  Kontainer 5 (Muatan: 147/150): ['BRG028',
'BRG044', 'BRG003', 'BRG017', 'BRG041',
'BRG013']
  Kontainer 6 (Muatan: 143/150): ['BRG023',
'BRG012', 'BRG027', 'BRG001', 'BRG047']
  Kontainer 7 (Muatan: 136/150): ['BRG032',
'BRG004']
  Kontainer 8 (Muatan: 145/150): ['BRG015',
'BRG008', 'BRG042']
  Kontainer 9 (Muatan: 137/150): ['BRG006',
'BRG036', 'BRG016', 'BRG033', 'BRG031',
'BRG034']
  Kontainer 10 (Muatan: 138/150): ['BRG005',
'BRG019', 'BRG009', 'BRG030']
  Kontainer 11 (Muatan: 140/150): ['BRG018',
'BRG043', 'BRG002']
Skor Awal: 5000012.09

Menjalankan Simulated Annealing...

--- Keadaan Akhir (Simulated Annealing) ---
Total Kontainer Digunakan: 14
  Kontainer 0 (Muatan: 148/150): ['BRG035',
'BRG021', 'BRG030', 'BRG034']
  Kontainer 1 (Muatan: 113/150): ['BRG046',
'BRG037', 'BRG045', 'BRG023']
  Kontainer 2 (Muatan: 141/150): ['BRG011',
'BRG026', 'BRG032']
  Kontainer 3 (Muatan: 148/150): ['BRG025',
'BRG020', 'BRG048', 'BRG007']
  Kontainer 4 (Muatan: 140/150): ['BRG014',
'BRG039', 'BRG038']
  Kontainer 5 (Muatan: 128/150): ['BRG028',
```

```
'BRG044', 'BRG003', 'BRG009', 'BRG031']
  Kontainer 6 (Muatan: 151/150): ['BRG012',
'BRG027', 'BRG041', 'BRG010', 'BRG024',
'BRG047']
  Kontainer 7 (Muatan: 132/150): ['BRG004',
'BRG022']
  Kontainer 8 (Muatan: 112/150): ['BRG015',
'BRG042']
  Kontainer 9 (Muatan: 118/150): ['BRG006',
'BRG036', 'BRG016', 'BRG040', 'BRG029']
  Kontainer 10 (Muatan: 128/150): ['BRG005',
'BRG013', 'BRG017', 'BRG033']
  Kontainer 11 (Muatan: 140/150): ['BRG018',
'BRG043', 'BRG002']
  Kontainer 12 (Muatan: 84/150): ['BRG019',
'BRG001']
  Kontainer 13 (Muatan: 33/150): ['BRG008']
Skor Akhir: 1000000.00
Durasi Eksekusi: 0.0235 detik
Plot skor disimpan di:
src\results\Simulated_Annealing\plots\Simulate
d_Annealing_problem_random_temp1000.0_cool0.99
_fragile_incomp_run1_20251030_200439_score.png
Plot probabilitas SA disimpan di:
src\results\Simulated_Annealing\plots\Simulate
d_Annealing_problem_random_temp1000.0_cool0.99
_fragile_incomp_run1_20251030_200439_prob.png
```

Pada algoritma simulated annealing, didapatkan final cost yaitu 1000000,00. Seluruh proses ini membutuhkan waktu selama 0.0235 detik dan dengan total iterasi yaitu 2000.

2.7.2.6. **Genetic Algorithm Varian Generasi**

2.7.2.7. **Genetic Algorithm Varian Populasi**

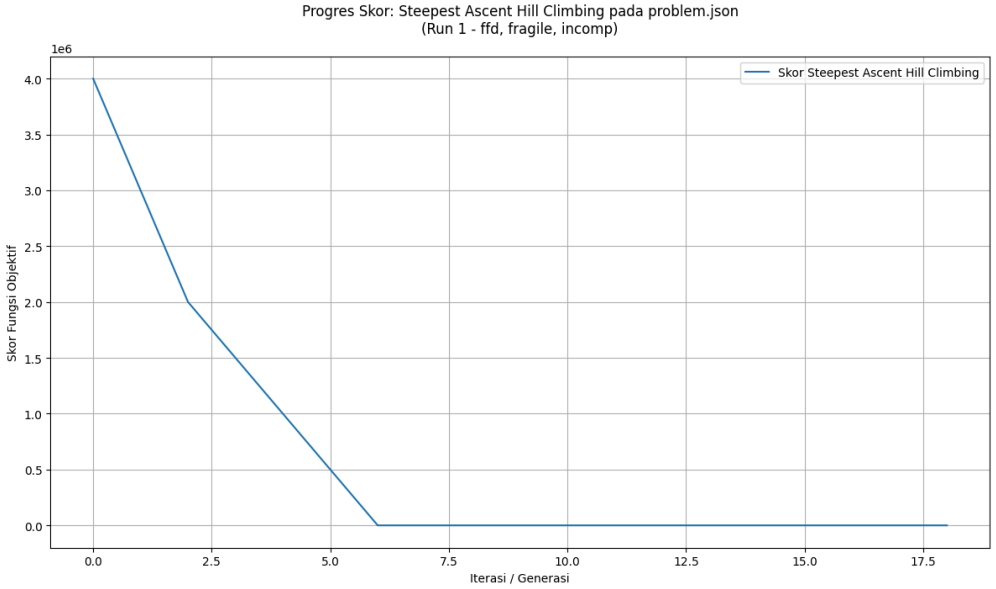
2.7.2.8. **Genetic Algorithm Varian Mutasi**

Pada eksperimen kedua, setelah dilakukan inisialisasi, didapatkan bahwa initial state cost adalah 5000012,09. Initial state ini akan digunakan pada seluruh algoritma untuk menentukan seberapa dekat algoritma tersebut mendekati global optima.

2.7.3. Eksperimen 3

Pada eksperimen ini, seluruh algorithm menggunakan FFD sebagai heuristik.

2.7.3.1. **Steepest Ascent Hill-Climbing**




```
RUN 1/1: Menyimpan output CLI ke
src\results\Steepest_Ascent_Hill_Climbing\logs
\log_Steepest_Ascent_Hill_Climbing_problem_ffd
_fragile_incomp_run1_20251030_200454.txt
================ RUN 1 / 1 ================
Konfigurasi Run:
  - Algoritma             : Steepest Ascent
Hill Climbing
  - Data File             :
src/data/problem.json
  - Initial State         : ffd
  - Iterasi Maks          : 1000
  - Constraint Rapuh      : Aktif
  - Constraint Inkompatibel : Aktif
--------------------------------------

--- Keadaan Awal (FFD) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 149/150): ['BRG035',
'BRG037']
  Kontainer 1 (Muatan: 148/150): ['BRG015',
'BRG024']
  Kontainer 2 (Muatan: 149/150): ['BRG004',
'BRG002', 'BRG030']
  Kontainer 3 (Muatan: 149/150): ['BRG025',
'BRG014', 'BRG029']
  Kontainer 4 (Muatan: 149/150): ['BRG005',
```

```
'BRG026', 'BRG023']
  Kontainer 5 (Muatan: 150/150): ['BRG018',
'BRG032', 'BRG019', 'BRG031']
  Kontainer 6 (Muatan: 148/150): ['BRG007',
'BRG044', 'BRG022', 'BRG047']
  Kontainer 7 (Muatan: 150/150): ['BRG038',
'BRG011', 'BRG028', 'BRG006']
  Kontainer 8 (Muatan: 150/150): ['BRG001',
'BRG043', 'BRG008', 'BRG039', 'BRG010']
  Kontainer 9 (Muatan: 149/150): ['BRG033',
'BRG012', 'BRG027', 'BRG036', 'BRG020',
'BRG048']
  Kontainer 10 (Muatan: 146/150): ['BRG040',
'BRG016', 'BRG045', 'BRG034', 'BRG009',
'BRG042', 'BRG041']
  Kontainer 11 (Muatan: 79/150): ['BRG003',
'BRG013', 'BRG046', 'BRG017', 'BRG021']
Skor Awal: 4000012.07

Menjalankan Steepest Ascent Hill Climbing...

--- Keadaan Akhir (Steepest Ascent Hill
Climbing) ---
Total Kontainer Digunakan: 12
  Kontainer 0 (Muatan: 136/150): ['BRG037',
'BRG026', 'BRG044']
  Kontainer 1 (Muatan: 148/150): ['BRG015',
'BRG024']
  Kontainer 2 (Muatan: 150/150): ['BRG013',
'BRG035', 'BRG047', 'BRG021']
  Kontainer 3 (Muatan: 149/150): ['BRG025',
'BRG014', 'BRG029']
  Kontainer 5 (Muatan: 150/150): ['BRG018',
'BRG032', 'BRG019', 'BRG031']
  Kontainer 6 (Muatan: 86/150): ['BRG022',
'BRG033', 'BRG041']
  Kontainer 7 (Muatan: 150/150): ['BRG038',
'BRG028', 'BRG005', 'BRG048']
  Kontainer 8 (Muatan: 150/150): ['BRG001',
'BRG043', 'BRG008', 'BRG039', 'BRG010']
  Kontainer 9 (Muatan: 150/150): ['BRG012',
'BRG027', 'BRG045', 'BRG034', 'BRG007']
  Kontainer 10 (Muatan: 150/150): ['BRG040',
'BRG016', 'BRG009', 'BRG042', 'BRG023',
'BRG020']
  Kontainer 11 (Muatan: 147/150): ['BRG003',
'BRG046', 'BRG017', 'BRG004', 'BRG030']
  Kontainer 12 (Muatan: 150/150): ['BRG036',
'BRG011', 'BRG006', 'BRG002']
Skor Akhir: 12.08
```

```
Durasi Eksekusi: 0.1387 detik
Plot skor disimpan di:
src\results\Steepest_Ascent_Hill_Climbing\plot
s\Steepest_Ascent_Hill_Climbing_problem_ffd_fr
agile_incomp_run1_20251030_200454.png
```

2.7.3.2. Stochastic Hill Climbing

2.7.3.3. Hill Climbing with Sideways Move

2.7.3.4.Random Restart Hill Climbing

2.7.3.5. Simulated Annealing

2.7.3.6. Genetic Algorithm varian Generasi

2.7.3.7.Genetic Algorithm varian Populasi

2.7.3.8.Genetic Algorithm varian Mutasi

2.8. Bonus

Pada project ini, semua spesifikasi bonus telah diimplementasikan yakni:

1. Implementasi seluruh algoritma Hill-Climbing
2. Implementasi constraint barang rapuh
3. Implementasi constraint barang kompatibel

BAB III
KESIMPULAN DAN SARAN

3.1. Kesimpulan

Berdasarkan hasil eksperimen yang sudah ada, dapat disimpulkan bahwa algoritma yang paling efektif serta efisien dalam menemukan solusi adalah *Random-Restart Hill-Climbing Algorithm* serta *Genetic Algorithm*. Kedua algoritma ini terbukti paling efektif dan konsisten dalam menemukan solusi terbaik dan skor paling rendah. Keduanya berhasil mengurangi jumlah kontainer dari keadaan awal sembari memenuhi semua *constraints* yang ada.

Sebaliknya, algoritma yang terbukti kurang efektif adalah *Simulated Annealing*. Dengan konfigurasi saat ini, algoritma ini terhitung kurang efektif. Hal ini bisa disebabkan beberapa hal seperti suhu awal dan cooling rate yang belum optimal. Hal ini bisa menyebabkan algoritma "mendingin" terlalu cepat sebelum sempat mencari semua solusi.

3.2. Saran

Untuk mendapatkan peningkatan pada hasil eksperimen berikutnya, penulis menyarankan untuk mengoptimalkan parameter dari *Simulated Annealing*. *Cooling rate* bisa diperlambat atau suhu awal bisa ditingkatkan. Ini akan memberikan algoritma lebih banyak waktu serta kesempatan untuk mengeksplorasi seluruh solusi yang ada.

PEMBAGIAN TUGAS

NIM	Nama	Tugas
18223056	Keane Putra Lim	● Membuat algoritma *Simulated Annealing* ● Melakukan testing ● Mengerjakan laporan bagian deskirpsi persoalan, algoritma *hill climb, simulated annealing*, integrasi, dan hasil eksperimen
18223074	Sebastian Albern Nugroho	● Membuat algoritma *Genetic Algorithm* ● Melakukan testing ● Mengerjakan laporan bagian *Objective Function, Utilities, Genetic Algorithm*, kesimpulan, dan saran.
18223104	M Khalfani Shaquille	● Membuat algoritma *Hill Climb* ● Melakukan testing ● Mengerjakan laporan bagian metodologi eksperimen dan hasil eksperimen.

REFERENSI

Institut Teknologi Bandung. (n.d.). Beyond classical search: Hill climbing [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/194228-Beyond-Classical-Search/1693804849395_IF3170_Materi3_Seg03_BeyondClassicalSearch_HillClimbing.pdf

Institut Teknologi Bandung. (n.d.). Beyond classical search: Local search [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/194228-Beyond-Classical-Search/1693804818592_IF3170_Materi03_Seg01_BeyondClassicalSearch_LocalSearch.pdf

Institut Teknologi Bandung. (n.d.). Beyond classical search: Simulated annealing [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/194228-Beyond-Classical-Search/1693804872404_IF3170_Materi03_Seg04_BeyondClassicalSearch_SimulatedAnnealing.pdf

Institut Teknologi Bandung. (n.d.). Beyond classical search: Genetic algorithm [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/194228-Beyond-Classical-Search/1693969141836_IF3170_Materi03_Seg05_BeyondClassicalSearch.pdf