Tugas Besar IF 3070

Dasar Artificial Intelligence

Disusun oleh :
Kelompok - 42
Brandon Theodore Ferinov / 18223020
Rafli Dwi Nugraha / 18223038
Luckman Fakhmanidris Arvasirri / 18223041

Kamis, 30 Oktober 2025

Deskripsi Persoalan

Tugas ini bertujuan untuk menempatkan sekumpulan barang dengan ukuran yang berbeda-beda ke dalam sejumlah kontainer dengan kapasitas yang sama, dengan tujuan menggunakan jumlah kontainer sesedikit mungkin.

Daftar Variabel

Entitas	Atribut	Contoh	Keterangan
Barang	ID Barang	BRG001	Kode unik untuk setiap barang.
	Ukuran	25	Ukuran atau berat barang (dalam unit yang sama, misal: kg atau m^3).
Kontainer	Kapasitas	100	Setiap kontainer memiliki kapasitas yang sama dan seragam.

Ketentuan Representasi

- Struktur data yang digunakan untuk representasi bebas. Misal, menggunakan list of lists.
- Representasi **state** adalah alokasi setiap Barang ke salah satu Kontainer. Contoh representasi: sebuah list yang berisi beberapa list lain, di mana setiap list internal merepresentasikan satu kontainer dan berisi ID barang di dalamnya.
  Contoh: [['BRG001', 'BRG005'], ['BRG002', 'BRG003'], ['BRG004']] berarti ada 3 kontainer yang digunakan.
- Inisialisasi (**initial state**) **bebas**, bisa secara **acak** atau menggunakan **algoritma heuristik** sederhana (misal: First Fit), di mana barang-barang ditempatkan satu per satu ke dalam kontainer.
- **Move** yang diperbolehkan tiap iterasi:
  1. Memindahkan satu barang dari satu kontainer ke kontainer lain (yang sudah ada atau yang baru).
  2. Menukar dua barang dari dua kontainer yang berbeda.

Contoh Input

```
{
  “kapasitas_kontaine”: 100,
  “barang”: [
    { “id”: “BRG001”, “ukuran”: 40 },
    { “id”: “BRG002”, “ukuran”: 55 },
    { “id”: “BRG003”, “ukuran”: 25 },
    { “id”: “BRG004”, “ukuran”: 60 },
    { “id”: “BRG005”, “ukuran”: 30 },
    { “id”: “BRG006”, “ukuran”: 45 },
    { “id”: “BRG007”, “ukuran”: 50 }
  ]
}
```

Contoh Output

Total Kontainer Digunakan: 4

1. Kontainer 1 (Total: 95/100):
   - BRG002 (55)
   - BRG006 (45)
2. Kontainer 2 (Total: 100/100):
   - BRG004 (60)
   - BRG001 (40)
3. Kontainer 3 (Total: 55/100):
   - BRG003 (25)
   - BRG005 (30)
4. Kontainer 4 (Total: 50/100):
   - BRG007 (50).

Objective Function

Tujuan utama adalah **meminimalkan jumlah kontainer yang digunakan**.

Berikut adalah acuan untuk membuat fungsi objektif:

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

Cakupan

Anda akan diminta untuk mengimplementasikan rencana yang telah Anda buat pada Tugas Besar 1. Berikut merupakan hal-hal yang perlu dilakukan oleh setiap kelompok:

- Implementasikan 3 algoritma local search dengan rincian sebagai berikut:
    - Salah satu algoritma hill-climbing
    - Simulated Annealing
    - Genetic Algorithm
- Lakukan eksperimen dengan skema sebagai berikut:
    - Jalankan setiap algoritma sebanyak **3 kali**, kemudian catat beberapa hal berikut:
        - Berlaku untuk semua algoritma
            - State awal dan akhir
            - Nilai objective function akhir yang dicapai
            - Plot nilai objective function terhadap banyak iterasi yang telah dilewati
            - Durasi proses pencarian
        - Berlaku hanya untuk Steepest Ascent Hill-Climbing dan Stochastic Hill-Climbing
            - Banyak iterasi hingga proses pencarian berhenti
        - Berlaku hanya untuk Hill-Climbing with Sideways Move
            - Banyak iterasi hingga proses pencarian berhenti
            - Note: Tambahkan parameter maximum sideways move, dimana ketika banyak sideways move yang dilakukan sudah mencapai maksimum, pencarian dihentikan
        - Berlaku hanya untuk Random Restart Hill-Climbing
            - Banyak restart
            - Banyak iterasi per restart
            - Note: Tambahkan parameter maximum restart, dimana ketika banyak restart sudah mencapai maksimum, pencarian dihentikan.
        - Berlaku hanya untuk Simulated Annealing
            - Plot $e^{\frac{\Delta E}{T}}$ terhadap banyak iterasi yang telah dilewati
            - Frekuensi ‘stuck’ di local optima
    - Khusus untuk Genetic Algorithm, lakukan beberapa hal berikut:
        - Terdapat 2 parameter yang dapat diubah, yaitu **jumlah populasi** dan **banyak iterasi.**
        - Jadikan jumlah populasi sebagai kontrol, kemudian pilih **3 variasi** banyak iterasi yang berbeda. Jalankan program sebanyak masing-masing **3 kali** untuk setiap konfigurasi parameter.
        - Jadikan banyak iterasi sebagai kontrol, kemudian pilih **3 variasi** jumlah populasi yang berbeda. Jalankan program sebanyak masing-masing **3 kali** untuk setiap konfigurasi parameter.
        - Untuk setiap eksperimen, catat beberapa hal berikut:
            - State awal dan akhir
            - Nilai objective function akhir yang dicapai

        - Plot nilai objective function terhadap banyak iterasi yang telah dilewati (Cukup plot nilai objective function maksimum dan rata-rata dari populasi terhadap banyak iterasi yang telah dilewati. Jika sudah terlanjur membuat plot untuk tiap individu pada populasi tidak menjadi masalah, silahkan diberikan keterangan saja maksud plotnya apa)
        - Pengaruh nilai probabilitas mutasi (**coba minimal 2**)
        - Jumlah populasi
        - Banyak iterasi
        - Durasi proses pencarian
- Lakukan analisis terhadap hasil eksperimen, berikut merupakan beberapa pertanyaan yang dapat menjadi acuan untuk analisis yang Anda lakukan (Anda boleh menambahkan beberapa pertanyaan tambahan jika dirasa perlu untuk dijelaskan):
    - Seberapa dekat tiap-tiap algoritma bisa mendekati global optima dan mengapa hasilnya demikian?
    - Bagaimana perbandingan hasil pencarian tiap-tiap algoritma dengan algoritma local search yang lain?
    - Bagaimana perbandingan durasi proses pencarian tiap algoritma relatif terhadap algoritma lainnya?
    - Seberapa konsisten hasil akhir yang didapatkan dari tiap-tiap eksperimen yang dilakukan?
    - Bagaimana pengaruh banyak iterasi dan jumlah populasi terhadap hasil akhir pencarian pada Genetic Algorithm?
    - dst…
- Program yang dibuat harus bisa **memvisualisasikan** state awal, state akhir, dan juga hasil eksperimennya (sesuaikan informasi hasil eksperimen yang ditampilkan dengan ketentuan yang telah dijelaskan di poin sebelum ini).
- Cara visualisasi dibebaskan kepada masing-masing kelompok.
- Gunakan bahasa **Python**.
- Diperbolehkan untuk menggunakan heuristik yang Anda buat sendiri atau dari referensi lain untuk optimasi pencarian solusi, asalkan masih dalam lingkup local search. Jangan lupa jelaskan heuristik yang Anda pakai di laporan.

Spesifikasi Bonus

Hanya kerjakan spesifikasi ini jika Anda sudah menyelesaikan **seluruh spesifikasi wajib**. Berikut merupakan beberapa spesifikasi bonus yang dapat Anda kerjakan:

1. **Buatlah implementasi untuk seluruh algoritma hill-climbing yang diajarkan.**
2. **Tambahkan Aturan Batasan Antar Barang (*Item Constraints*).** Modifikasi masalah dengan menambahkan batasan baru yang membuat pengepakan lebih kompleks. Tambahkan aturan ini sebagai komponen dalam *objective function* Anda (misalnya, dengan menambahkan penalti baru). Anda dipersilahkan untuk **membuat batasan**

sendiri.
Contoh batasan yang bisa ditambahkan:

a. **Barang Rapuh (Fragile Items):** Tambahkan atribut **rapuh** pada beberapa barang. Sebuah barang rapuh tidak boleh ditempatkan dalam satu kontainer bersama barang lain yang total ukurannya melebihi nilai tertentu (misal, 50). Situasi Ini mensimulasikan barang rapuh yang tidak boleh tertindih beban berat.
b. **Barang Tidak Kompatibel (Incompatible Items):** Tambahkan atribut **tipe** pada setiap barang (contoh: ‘makanan’, ‘kimia’, ‘pakaian’). Buat aturan bahwa barang dengan tipe tertentu tidak boleh berada dalam satu kontainer yang sama (misalnya, ‘makanan’ tidak boleh digabung dengan ‘kimia’).

Pembahasan

I. Pemilihan Objective Function

Objective function dibuat sebagai nilai dari kesesuaian *packing* barang dalam kontainer. Pemilihan objective function dilakukan dengan menggunakan 2 bobot penghitungan, yaitu dengan menghitung value dari kapasitas, dan juga menghitung value dari kepadatan dari container. Semakin tinggi nilai dari objective function maka nilai kesesuaiannya semakin rendah, sebaliknya jika nilainya semakin mendekati 0 maka kesesuaiannya semakin optimal.

A. Perhitungan Value Kapasitas

Value kapasitas dari suatu container dihitung dengan nilai selisih kelebihan ukuran kapasitas kontainer. Jika total ukuran barang melebihi kapasitas, maka akan diberikan penalti sebesar nilai selisihnya dikali 2.

$$Value = (total\ ukuran\ barang - kapasitas)\ x\ 2$$

B. Perhitungan Kepadatan Kontainer

Value kepadatan kontainer dihitung dengan berdasarkan jumlah dari kontainer yang digunakan dan kepadatan setiap kontainer. Perhitungan dilakukan dengan menghitung kepadatan dari setiap kontainer, lalu nantinya dilakukan perhitungan dengan jumlah kontainer.

$$Value = (1 - avg(kepadatan))\ x\ jumlah\ kontainer$$

Dengan perhitungan Value kapasitas dan Value Kepadatan, dapat disimpulkan bahwa perhitungan untuk objective function keseluruhan adalah sebagai berikut.

$$Obj.function = ((total\ ukuran\ barang - kapasitas)\ x\ 2) + (1 - avg(kepadatan))\ x\ jumlah\ kontainer$$

II. Penjelasan Implementasi Algoritma

A. Entities

1. Kelas Barang

Nama Kelas	Atribut	Deskripsi
Barang	ID: String	Atribut yang menyimpan Kode dari barang
	ukuran: Integer	Atribut untuk ukuran dari barang

Source code:

```
class Barang:
    def __init__(self, ID: str, ukuran: int):
        self.ID = ID
        self.ukuran = ukuran

    def __eq__(self, other):
        if not isinstance(other, Barang):
            return NotImplemented
        return self.value == other.value

    def __str__(self) -> str:
        return f"ID: {self.ID}\nUkuran: {self.ukuran}"
```

2. Kelas Container

Nama Kelas	Atribut	Deskripsi
Container	kapasitas: Integer	Atribut untuk kapasitas maksikum yang dapat ditampung oleh container
	daftar_barang: List of Barang	List yang memuat semua barang yang ada dalam kontainer

Source code:

```
class Container:
    def __init__(self, kapasitas: int):
        self.kapasitas = kapasitas
        self.daftar_barang: List[Barang] = []

    def add_barang(self, barang: Barang) -> bool:
        total_ukuran = self.hitung_ukuran() + barang.ukuran
        if total_ukuran <= self.kapasitas:
            self.daftar_barang.append(barang)
            return True
        return False

    def remove_barang(self, barang_keluar: Barang):
        self.daftar_barang = [b for b in self.daftar_barang if b.ID !=
barang_keluar.ID]

    def hitung_ukuran(self) -> int:
        return sum(barang.ukuran for barang in self.daftar_barang)

    def reset_container(self):
        self.daftar_barang: List[Barang] = []

    def has_barang(self, barang1: Barang) -> bool:
        for barang in self.daftar_barang:
            if barang.ID == barang1.ID:
                return True
        return False

    def __str__(self) -> str:
        result = f"Kontainer (Total:
{self.hitung_ukuran()}/{self.kapasitas}):"
        for barang in self.daftar_barang:
            result += f"\n{barang.ID} ({barang.ukuran})"
        return result
```

Nama Method	Deskripsi
add_barang	Method yang digunakan untuk menambahkan barang ke dalam kontainer. Jika ukuran barang melebihi kapasitas, maka tidak akan ditambahkan. Jika berhasil ditambahkan akan mengembalikan True, jika tidak akan mengembalikan nilai False.

Source Code:

```
def add_barang(self, barang: Barang) -> bool:
        total_ukuran = self.hitung_ukuran() + barang.ukuran
        if total_ukuran <= self.kapasitas:
            self.daftar_barang.append(barang)
            return True
        return False
```

remove_barang	Method untuk menghapus barang dari kontainer. Penghapusan dilakukan dengan cara mengambil semua barang ke daftar_barang tetapi jika ditemukan barang yang ingin dihapus, maka barang tersebut tidak ditambahkan kembali ke list barang.

Source Code:

```
def remove_barang(self, barang_keluar: Barang):
            self.daftar_barang = [b for b in self.daftar_barang if b.ID != barang_keluar.ID]
```

hitung_ukuran	Method untuk menghitung total ukuran semua barang yang ada dalam kontainer.

Source Code:

```
def hitung_ukuran(self) -> int:
        return sum(barang.ukuran for barang in self.daftar_barang)
```

reset_container	Method yang digunakan untuk mengosongkan kontainer

Source Code:

```
def reset_container(self):
        self.daftar_barang: List[Barang] = []
```

has_barang	Method yang digunakan untuk mengecek apakah barang ada dalam kontainer. Jika barang ditemukan maka akan mengembalikan nilai True, jika tidak akan mengembalikan

	nilai False.

Source Code:

```
    def has_barang(self, barang1: Barang) -> bool:
        for barang in self.daftar_barang:
            if barang.ID == barang1.ID:
                return True
        return False
```

3. Kelas State

Nama Kelas	Atribut	Deskripsi
State	list_barang: List of Barang	List barang yang akan dimasukan kedalam kontainer
	list_container: List of Container	List dari container pada state
	objective_function: Integer	nilai objective function pada state

Source code:

```
class State:
    def __init__(self, list_barang: list[Barang]):
        self.list_barang = list_barang
        self.list_container: List[Container] = []
        self.objective_function = 0

    def __str__(self) -> str:
        if not self.list_container:
            return "State: No containers"

        result = []
        for i, container in enumerate(self.list_container, 1):
            container_str = f"Kontainer {i} (Total:
{container.hitung_ukuran()}/{container.kapasitas}):"
            for barang in container.daftar_barang:
                container_str += f"\n{barang.ID} ({barang.ukuran})"
            result.append(container_str)
```

```
        return "\n\n".join(result)


    def initiate_random(self, kapasitas: int):
        self.list_container = []

        barang_packing = self.list_barang.copy()
        random.shuffle(barang_packing)

        kontainer = Container(kapasitas)

        for barang in barang_packing:
            if not kontainer.add_barang(barang):
                if kontainer.daftar_barang:
                    self.list_container.append(kontainer)
                kontainer = Container(kapasitas)
                kontainer.add_barang(barang)

        if kontainer.daftar_barang:
            self.list_container.append(kontainer)

    def move_exist(self, barang: Barang, kontainer1: Container, kontainer2:
Container):
        if kontainer1.hitung_ukuran() + barang.ukuran > kontainer1.kapasitas:
            raise ValueError("Barang Melebihi Kapasitas")
        kontainer1.add_barang(barang)
        kontainer2.remove_barang(barang)

    def move_to_empty(self, barang:Barang, kapasitas:int):
        newcontainer = Container(kapasitas)
        newcontainer.add_barang(barang)
        for kontainer in self.list_container:
            kontainer.remove_barang(barang)
        self.list_container.append(newcontainer)
```

```
    def swap_barang(self, barang1: Barang, barang2: Barang):
        container1 = None
        container2 = None

        for container in self.list_container:
            if container.has_barang(barang1):
                container1 = container
            if container.has_barang(barang2):
                container2 = container

        if not container1 or not container2:
            raise ValueError("Barang tidak ditemukan")

        if container1 == container2:
            return

        new_size1 = container1.hitung_ukuran() - barang1.ukuran +
barang2.ukuran
        new_size2 = container2.hitung_ukuran() - barang2.ukuran +
barang1.ukuran

        if new_size1 > container1.kapasitas or new_size2 >
container2.kapasitas:
            raise ValueError("Kapasitas tidak cukup")

        container1.remove_barang(barang1)
        container2.remove_barang(barang2)
        container1.add_barang(barang2)
        container2.add_barang(barang1)

    def generate_neighbour(self, kapasitas: int):
        list_of_states = []
        for i in range ((len(self.list_barang) ** 2) - 1):
            state = State(self.list_barang)
            state.initiate_random(kapasitas)
            list_of_states.append(state)

        return list_of_states
```

Nama Method	Deskripsi
initiate_random	Method untuk inisialisasi state secara random, barang akan dishuffle dan dimasukkan kedalam container sesuai urutan. JIka kontainer tidak dapat memuat barang baru, maka akan dibuat kontainer baru, diisi hingga tidak bisa memuat lagi dan ditambahkan ke dalam list_container

Source Code:

```
def initiate_random(self, kapasitas: int):
        self.list_container = []

        barang_packing = self.list_barang.copy()
        random.shuffle(barang_packing)

        kontainer = Container(kapasitas)

        for barang in barang_packing:
            if not kontainer.add_barang(barang):
                if kontainer.daftar_barang:
                    self.list_container.append(kontainer)
                kontainer = Container(kapasitas)
                kontainer.add_barang(barang)

        if kontainer.daftar_barang:
            self.list_container.append(kontainer)
```

move_exist	Method untuk memindahkan barang dalam satu kontainer ke kontainer lain. Jika kontainer yang dituju tidak dapat memuat barang yang akan dipindahkan, maka akan mengeluarkan ValueError.

Source Code:

```
def move_exist(self, barang: Barang, kontainer1: Container, kontainer2:
Container):
        if kontainer1.hitung_ukuran() + barang.ukuran > kontainer1.kapasitas:
            raise ValueError("Barang Melebihi Kapasitas")
```

```
        kontainer1.add_barang(barang)
        kontainer2.remove_barang(barang)
```

move_to_empty	Method untuk memindahkan barang yang ada dalam kontainer ke kontainer kosong baru.

Source Code:

```
def move_to_empty(self, barang:Barang, kapasitas:int):
        newcontainer = Container(kapasitas)
        newcontainer.add_barang(barang)
        for kontainer in self.list_container:
            kontainer.remove_barang(barang)
        self.list_container.append(newcontainer)
```

swap_barang	Method untuk menukar barang dalam satu kontainer dengan barang di kontainer lain. Jika salah satu kontainer tidak dapat menampung barang yang akan ditukar, maka akan mengeluarkan ValueError

Source Code:

```
def swap_barang(self, barang1: Barang, barang2: Barang):
        container1 = None
        container2 = None

        for container in self.list_container:
            if container.has_barang(barang1):
                container1 = container
            if container.has_barang(barang2):
                container2 = container

        if not container1 or not container2:
            raise ValueError("Barang tidak ditemukan")

        if container1 == container2:
            return

        new_size1 = container1.hitung_ukuran() - barang1.ukuran +
```

```
barang2.ukuran
        new_size2 = container2.hitung_ukuran() - barang2.ukuran +
barang1.ukuran

        if new_size1 > container1.kapasitas or new_size2 >
container2.kapasitas:
            raise ValueError("Kapasitas tidak cukup")

        container1.remove_barang(barang1)
        container2.remove_barang(barang2)
        container1.add_barang(barang2)
        container2.add_barang(barang1)
```

generate_neighbour	Method yang digunakan untuk *generate* list of state berdasarkan list barang yang sudah ada.

Source Code:

```
    def generate_neighbour(self, kapasitas: int):
        list_of_states = []
        for i in range ((len(self.list_barang) ** 2) - 1):
            state = State(self.list_barang)
            state.initiate_random(kapasitas)
            list_of_states.append(state)

        return list_of_states
```

B. Algorithm

1. Objective Function

Nama Fungsi	Deskripsi
hitung_value_kapasitas	Fungsi untuk menghitung value dari kapasitas pada kontainer. Jika ukuran total barang melebihi kapasitas, maka akan dikenakan penalti sebesar 2 kali dari selisih ukuran.
Source code:	

```python
def hitung_value_kapasitas(state: State, kapasitas: int) -> int:
    result = 0

    for container in state.list_container:
        if container.hitung_ukuran() > kapasitas:
            result += (container.hitung_ukuran() - kapasitas)*2

    return result
```

hitung_value_kepadatan	Fungsi yang digunakan untuk menghitung value kepadatan kontainer dan jumlah kontainer. Kepadatan setiap kontainer akan dihitung dan dibuat rata-rata, lalu selanjutnya akan dihitung berdasarkan jumlah kontainer.

Source code:

```python
def hitung_value_kepadatan(state: State) -> int:
    result = 0
    total_sisa = 0

    for container in state.list_container:
        if container.hitung_ukuran() <= container.kapasitas:
            kepadatan = container.hitung_ukuran() / container.kapasitas

    kepadatan = kepadatan / len(state.list_container)

    result = (1 - kepadatan) * len(state.list_container)

    return result
```

objective_function	Fungsi untuk menghitung objective function dengan menjumlahkan value kapasitas dan value kepadatan.

Source code:

```python
def objective_function(current_state: State, kapasitas: int):
    current_state.objective_function = hitung_value_kapasitas(current_state,
```

```
kapasitas) + hitung_value_kepadatan(current_state)
```

hitung_of	Fungsi untuk menghitung objective function dari semua State dalam list of State

```
def hitung_of(daftar: List[State], kapasitas: int):
    for state in daftar:
        objective_function(state, kapasitas)
```

2. Simulated Annealing

Nama Fungsi	Deskripsi
generate_neighbor	Fungsi generate_neighbor digunakan untuk men-*generate* neighbor yang nanti akan digunakan untuk mencari neighbor mana yang memiliki fitness level paling kecil

Source code:

```
def generate_neighbour(list_of_barang, kapasitas):
   list_of_states = []
   for i in range ((len(list_of_barang) ** 2) - 1):
       state = State(list_of_barang)
       state.initiate_random(kapasitas)
       list_of_states.append(state)

   return list_of_states
```

generate_barang	Fungsi yang digunakan untuk men-*generate* barang secara random untuk inisialisasi class State

Source code:

```
def generate_barang():
   list_of_barang = []
   for i in range(10):
       barang = Barang(ID=f"BRG{i+1}", ukuran=random.randint(5, 50))
       list_of_barang.append(barang)
```

```
    return list_of_barang
```

visualize_state	Fungsi untuk memvisualisasikan initial state dan end state

Source code:

```
def visualize_state(state):
    plt.figure(figsize=(8, 5))
    y_positions = range(len(state.list_container))

    for i, container in enumerate(state.list_container):
        start = 0
        for barang in container.daftar_barang:
            plt.barh(
                y=i,
                width=barang.ukuran,
                left=start,
                label=barang.ID if i == 0 else "",
            )
            start += barang.ukuran
        plt.axvline(container.kapasitas, color='black', linestyle='--',
linewidth=1)

    plt.yticks(y_positions, [f"Container {i+1}" for i in y_positions])
    plt.xlabel("Kapasitas")
    plt.title("State")
    plt.tight_layout()
    plt.show()
```

fitness_over_iteration	Fungsi untuk menampilkan grafik fitness level per iterasi

Source code:

```
def fitness_over_iteration(fitness_history):
    plt.figure(figsize=(8, 4))
    plt.plot(fitness_history, color='blue', linewidth=2)
    plt.title("Fitness Progress Over Iterations")
    plt.xlabel("Iteration")
    plt.ylabel("Fitness Value (Objective Function)")
```

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

delta_e_over_iteration	Fungsi untuk menampilkan nilai $e^{\frac{\Delta E}{T}}$ per iterasi yang terjadi

Source code:

```
def delta_e_over_iteration(x):
    plt.figure(figsize=(8, 4))
    plt.plot(x, color='blue', linewidth=2)
    plt.title("Delta E / T Over Iterations")
    plt.xlabel("Iteration")
    plt.ylabel("e ^(-ΔE / T)")
    plt.tight_layout()
    plt.show()
```

local_optima	Fungsi untuk menghitung berapa kali algoritma “stuck” pada local optima dengan threshold yang sudah ditentukan

Source code:

```
def local_optima(best_fitness, last_best_fitness, no_improve_steps,
stuck_counter, stuck_threshold):
    if best_fitness < last_best_fitness:
        no_improve_steps = 0
        last_best_fitness = best_fitness
    else:
        no_improve_steps += 1
        if no_improve_steps >= stuck_threshold:
            stuck_counter += 1
            no_improve_steps = 0

    return no_improve_steps, stuck_counter, last_best_fitness
```

simulated_annealing	Fungsi utama dari kode ini yang berguna untuk menjalankan algoritma simulated annealing

Source code:

```
def simulated_annealing(start_temp, end_temp, prob, iteration,
list_of_barang):
```

```
start = time.time()
state = State(list_of_barang)
state.initiate_random(kapasitas=100)
objective_function(state, kapasitas=100)
print(f"\nInitial Fitness: {state.objective_function}")
print("Initial State:")
for container in state.list_container:
    print(container)

visualize_state(state)

current = state
current_fitness = state.objective_function

best = copy.deepcopy(state)
best_fitness = current_fitness

list_of_states = generate_neighbour(list_of_barang, kapasitas=100)
rand_state = random.choice(list_of_states)
objective_function(rand_state, 100)

fitness_history = [current_fitness]
delta_e_history = []

stuck_counter = 0
no_improve_steps = 0
stuck_threshold = 100
last_best_fitness = best_fitness

T = start_temp

while T > end_temp:
    for i in range(iteration):
        new_state = random.choice(list_of_states)
        objective_function(new_state, 100)
        new_fitness = new_state.objective_function

        delta_e = new_fitness - current_fitness
```

```
            if delta_e < 0 or math.exp(-delta_e / T) > random.random():
                current = new_state
                current_fitness = new_fitness

                fitness_history.append(current_fitness)
                delta_e_history.append(math.exp(-delta_e / T))
                if current_fitness < best_fitness:
                    best = copy.deepcopy(current)
                    best_fitness = current_fitness

            no_improve_steps, stuck_counter, last_best_fitness =
local_optima(best_fitness, last_best_fitness, no_improve_steps,
stuck_counter, stuck_threshold)
        T *= prob
    end_time = time.time()
    execution_time = (end_time - start)
    fitness_over_iteration(fitness_history)
    delta_e_over_iteration(delta_e_history)
    visualize_state(best)
    print(f"\nTotal Stuck in Local Optima: {stuck_counter}")
    print(f"\nExecution Time: {execution_time:.4f} s")
    print(f"\nBest Fitness: {best_fitness}")
    return best.list_container
```

3. Steepest Ascent Hill Climbing

Nama Kelas	Deskripsi
HCsteepest	Kelas ini merupakan kelas yang digunakan untuk mengimplementasikan Algoritma Steepest Ascent Hill Climbing untuk *packing bin.* Kelas memuat atribut initial State, initial state value, final state, final state value, time execution, dan total iteration.
Nama Atribut	**Deskripsi**
initial_state: State	State Awal untuk algoritma
initial_state_value: Integer	Value objective function dari initial state

final_state: State	State akhir yang didapatkan dari algoritma
neighbours: List of State	List yang digunakan untuk menyimpan State untuk algoritma
iteration: Integer	jumlah iterasi yang dilakukan oleh algoritma
time_execution: Integer	Waktu yang dibutuhkan untuk menjalankan algoritma
objective_history: List of Integer	List yang digunakan untuk menyimpan objective function setiap iterasi

Source code:

```
lass HCsteepest():
    def __init__(self, state: State):
        self.initial_state = state
        self.initial_state_value = 0
        self.final_state = None
        self.final_state_value = 0
        self.neighbours = None
        self.iteration = 0
        self.time_execution = 0
        self.objective_history = []

    def run(self, kapasitas, list_barang: List[Barang]):
        start = time.time()

        self.initial_state.initiate_random(kapasitas)
        objective_function(self.initial_state, kapasitas)
        self.initial_state_value = self.initial_state.objective_function
        print(f"Initial Value: {self.initial_state_value}")
        self.neighbour_value = 0

        while True:
            neighbours = self.initial_state.generate_neighbour(kapasitas)
            hitung_of(neighbours, kapasitas)
            neighbour = neighbours[0]

            best_neighbour = min(neighbours, key=lambda n:
n.objective_function)
            best_value = best_neighbour.objective_function
```

```
            self.objective_history.append(best_value)
            self.iteration += 1
            for successor in neighbours[1:]:
                if neighbour.objective_function > successor.objective_function:
                    neighbour = successor

            objective_function(neighbour, kapasitas)
            self.neighbour_value = neighbour.objective_function

            if self.neighbour_value >= self.initial_state_value:
                self.final_state = self.initial_state
                self.final_state_value = self.initial_state_value

                end = time.time()

                self.time_execution = (end - start) * 1000
                plt.figure(figsize=(7, 4))
                plt.plot(range(len(self.objective_history)),
self.objective_history, marker='o')
                plt.title("Objective Function vs Iteration")
                plt.xlabel("Iteration")
                plt.ylabel("Objective Function Value")
                plt.grid(True, linestyle="--", alpha=0.6)
                plt.tight_layout()
                plt.show()
                return self.initial_state

            self.initial_state = neighbour
            self.initial_state_value = neighbour.objective_function
```

4. Genetic Algorithm

Nama Fungsi	Deskripsi
create_initial_population	Fungsi yang digunakan untuk menciptakan

	populasi awal yang nantinya akan diproses

Source code:

```
def create_initial_population(self):
        self.population = []
        for _ in range(self.pop_size):
            state = State(self.list_barang)
            state.initiate_random(self.kapasitas)
            objective_function(state, self.kapasitas)
            self.population.append(state)
```

select_parent	Fungsi yang digunakan untuk menentukan parent dari populasi berikutnya

Source code:

```
def select_parent(self) -> State:
        tournament = random.sample(self.population, self.tournament_size)
        winner = min(tournament, key=lambda s: s.objective_function)
        return winner
```

crossover	Fungsi yang digunakan untuk melakukan crossover antar parent yang telah dipilih menggunakan fungsi select_parent

Source code:

```
def crossover(self, parent1: State, parent2: State) -> State:
        child = copy.deepcopy(parent2)
        if not parent1.list_container:
            return child

        container_to_inject = random.choice(parent1.list_container)
        items_in_container = {b.ID for b in container_to_inject.daftar_barang}
        new_child_containers: List[Container] = []
        for container in child.list_container:
            items_to_keep = [b for b in container.daftar_barang if b.ID not in
items_in_container]
            if items_to_keep:
                new_container = Container(self.kapasitas)
                for item in items_to_keep:
                    new_container.add_barang(item)
```

```
                new_child_containers.append(new_container)
        new_child_containers.append(copy.deepcopy(container_to_inject))
        child.list_container = new_child_containers
        objective_function(child, self.kapasitas)
        return child
```

mutate	Fungsi yang digunakan untuk melakukan mutasi suatu individu

Source code:

```
def mutate(self, state: State):
        if random.random() < self.mutation_prob:
            mutation_type = random.randint(1, 2)
            if mutation_type == 1 and len(self.list_barang) >= 2:
                try:
                    b1, b2 = random.sample(self.list_barang, 2)
                    state.swap_barang(b1, b2)
                except ValueError:
                    self.mutate_move_to_empty(state)
            elif mutation_type == 2:
                self.mutate_move_to_empty(state)
            else:
                self.mutate_move_to_empty(state)
            objective_function(state, self.kapasitas)
```

mutate_move_to_empty	Fungsi yang melakukan mutasi individu dengan cara move to empty

Source code:

```
def mutate_move_to_empty(self, state: State):
        try:
            barang_to_move = random.choice(self.list_barang)
            state.move_to_empty(barang_to_move, self.kapasitas)
        except Exception:
            pass
```

get_best_individual	Fungsi yang digunakan untuk mengambil individual dengan fitness value terbaik

Source code:

```
def get_best_individuals(self, n: int) -> List[State]:
        sorted_pop = sorted(self.population, key=lambda s:
```

```
s.objective_function)
        return sorted_pop[:n]
```

record_history	Fungsi yang digunakan untuk menyimpan populasi-populasi sebelumnya

Source code:

```
def record_history(self):
        best_fitness = min(s.objective_function for s in self.population)
        avg_fitness = sum(s.objective_function for s in self.population) /
self.pop_size
        self.history_best_fitness.append(best_fitness)
        self.history_avg_fitness.append(avg_fitness)
```

run	Fungsi yang digunakan untuk menjalankan algoritma genetik secara keseluruhan

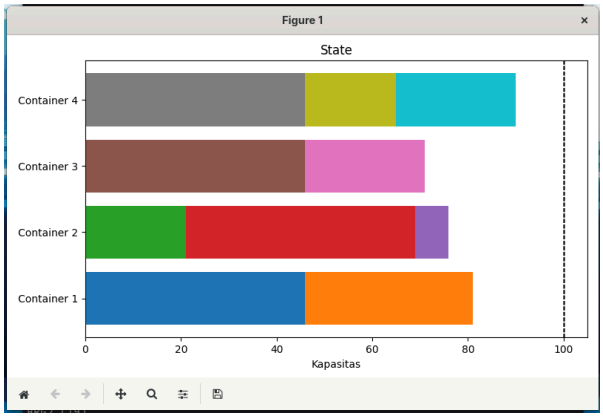
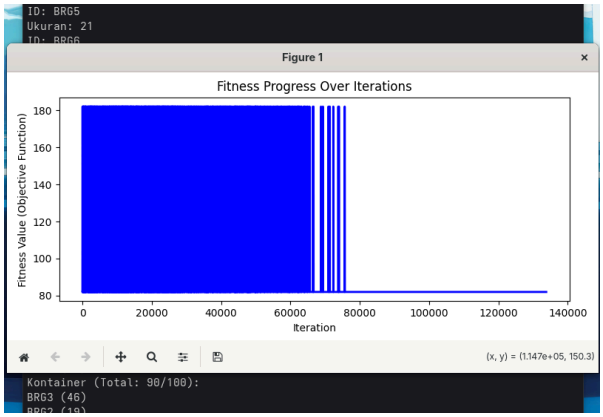
Source code:

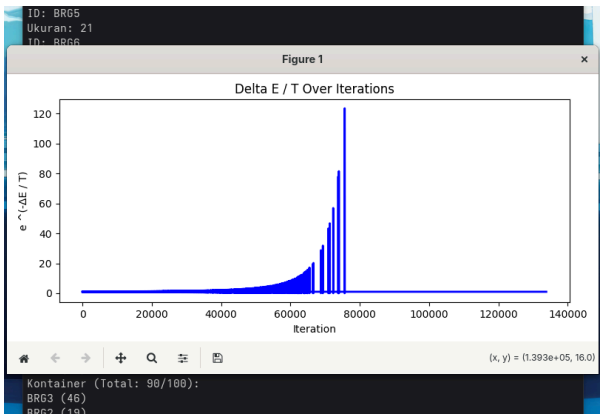
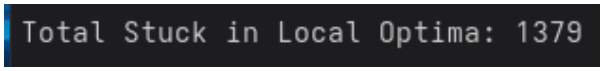
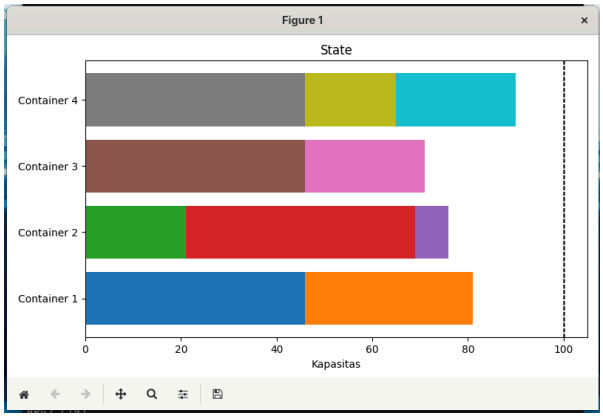
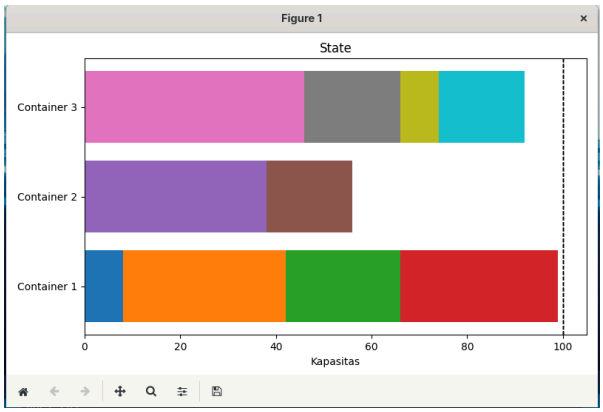
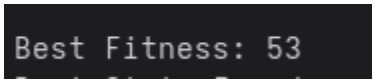
```
def run(self) -> Tuple[State, State, float, List[float], List[float]]:
        start_time = time.time()
        self.create_initial_population()
        initial_best_state = copy.deepcopy(self.get_best_individuals(1)[0])
        self.record_history()
        for _ in range(self.iterations):
            new_population = []
            elites = self.get_best_individuals(self.elitism_count)
            new_population.extend(copy.deepcopy(elite) for elite in elites)
            while len(new_population) < self.pop_size:
                parent1 = self.select_parent()
                parent2 = self.select_parent()
                child = self.crossover(parent1, parent2)
                self.mutate(child)
                new_population.append(child)
            self.population = new_population
            self.record_history()
        end_time = time.time()
        duration = end_time - start_time
        final_best_state = self.get_best_individuals(1)[0]
        return initial_best_state, final_best_state, duration,
self.history_best_fitness, self.history_avg_fitness
```

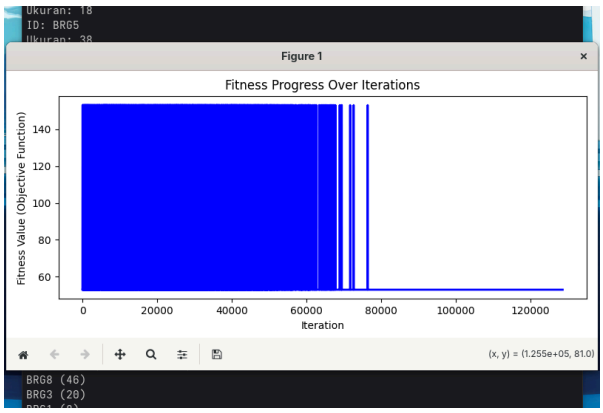
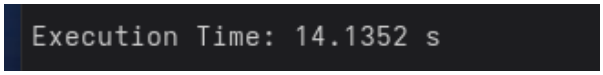
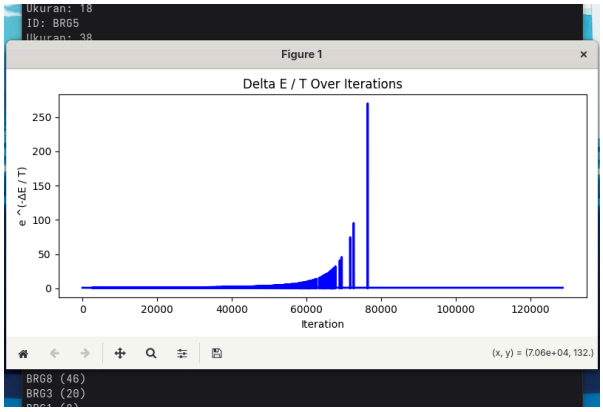
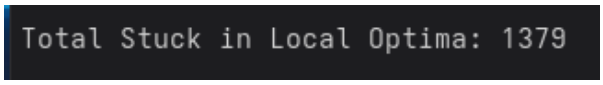
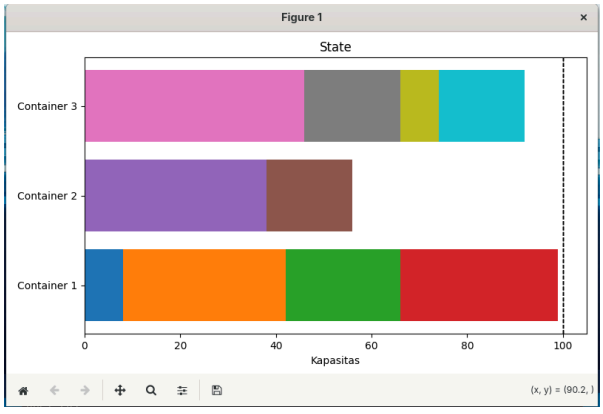
III. Hasil Eksperimen dan Analsis

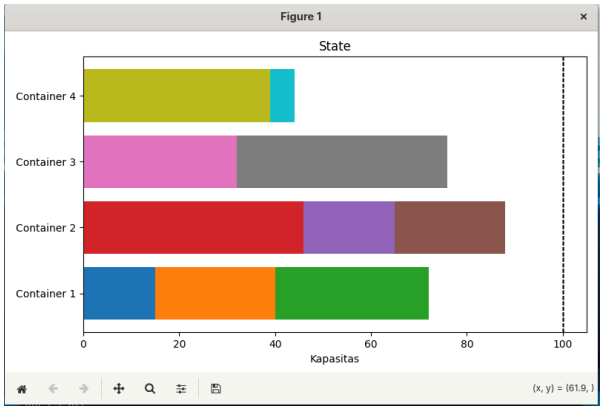
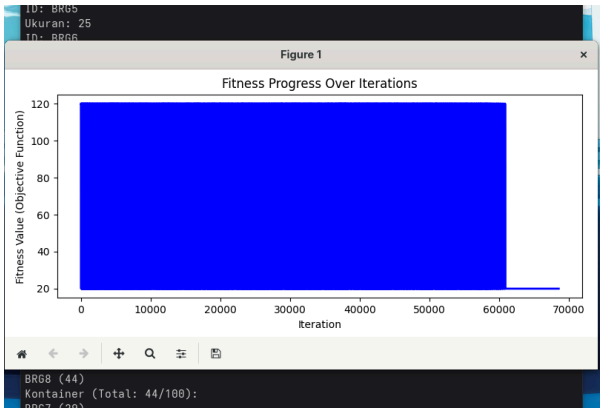
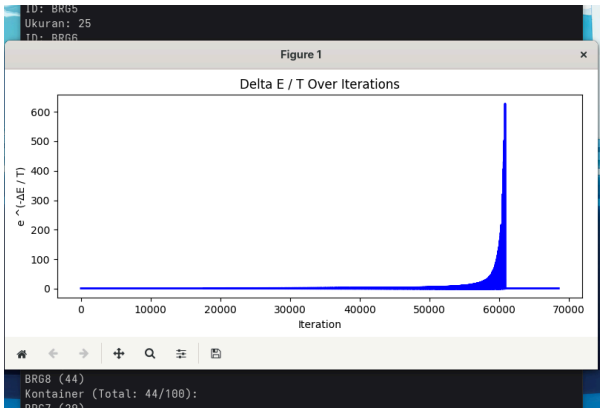
A. Hasil Eksperimen

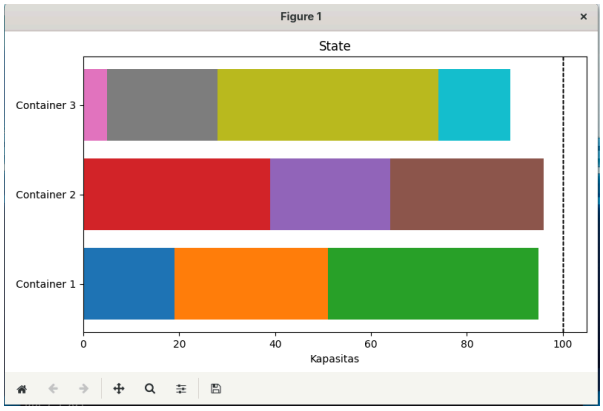
1. Simulated Annealing

Hasil	Dokumentasi
Initial State #1	
Objective Function Akhir (Fitness Level) #1	
Plot Objective Function per Iterasi #1	
Durasi Proses Pencarian #1	

Plot $e^{\frac{\Delta E}{T}}$ per iterasi #1	
Frekuensi Stuck Local Optima #1	
End State #1	
Initial State #2	
Objective Function Akhir (Fitness Level) #2	

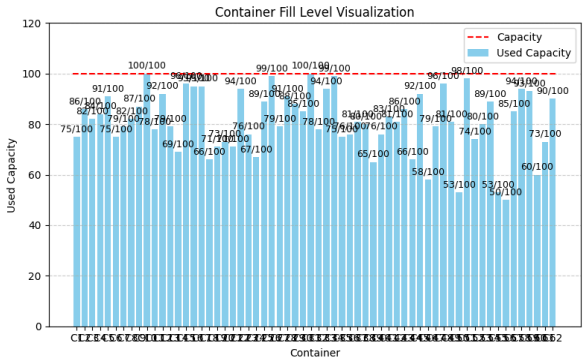
Plot Objective Function per Iterasi #2	
Durasi Proses Pencarian #2	
Plot $e^{\frac{\Delta E}{T}}$ per iterasi #2	
Frekuensi Stuck Local Optima #2	
End State #2	

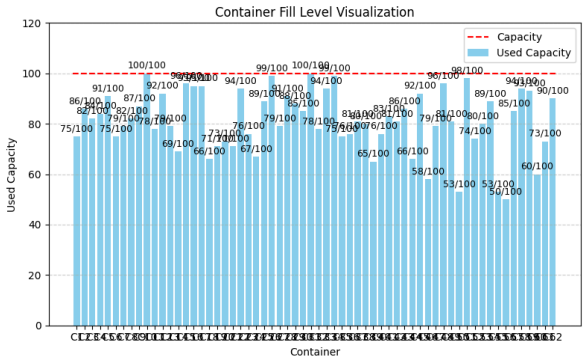
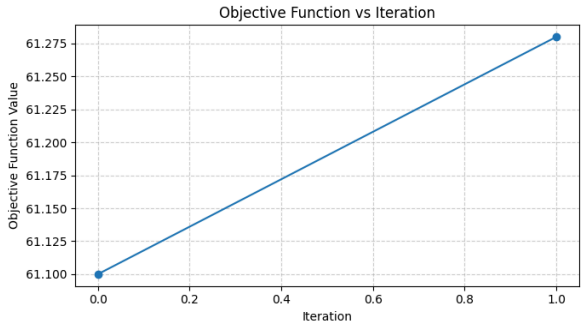
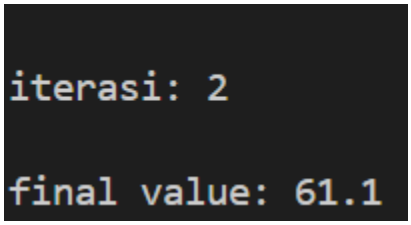
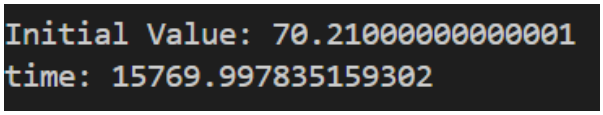
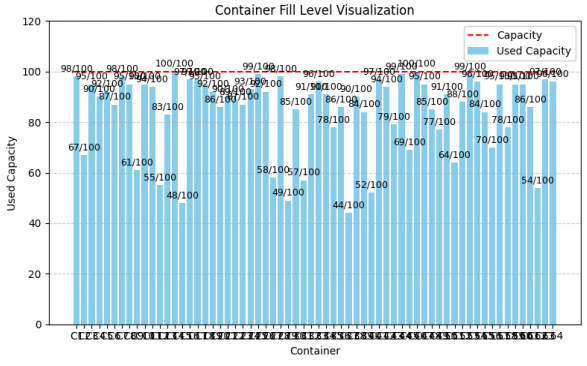
Initial State #3	
Objective Function Akhir (Fitness Level) #3	`Best Fitness: 20`
Plot Objective Function per Iterasi #3	
Durasi Proses Pencarian #3	`Execution Time: 15.6789 s`
Plot $e^{\frac{\Delta E}{T}}$ per iterasi #3	
Frekuensi Stuck Local Optima #3	`Total Stuck in Local Optima: 1378`

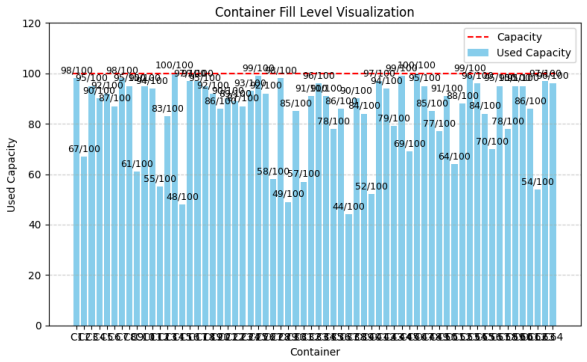
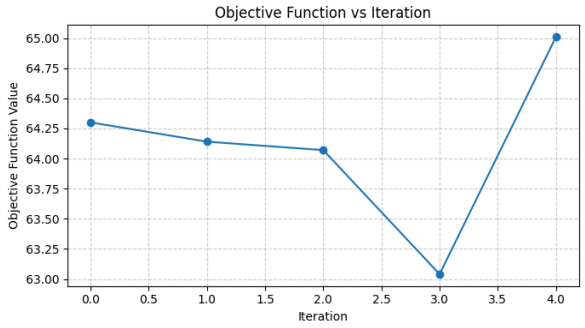
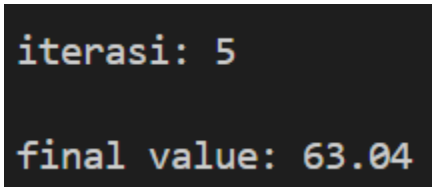
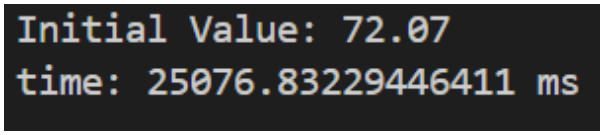
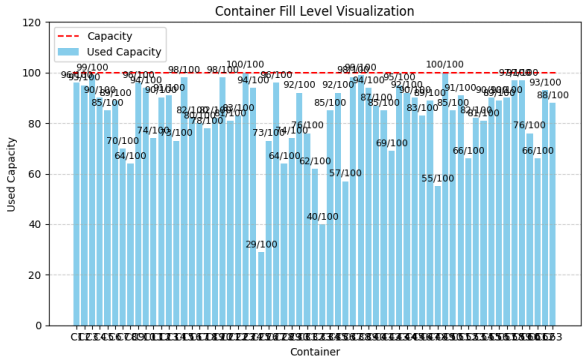
End State #3	

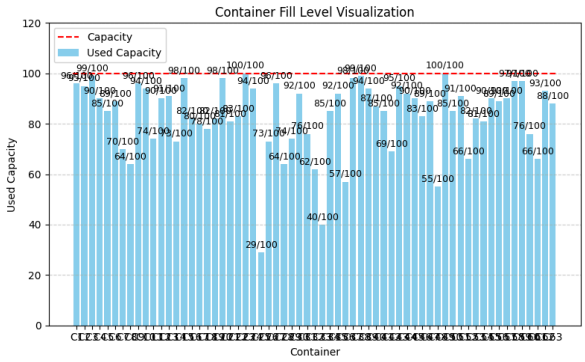
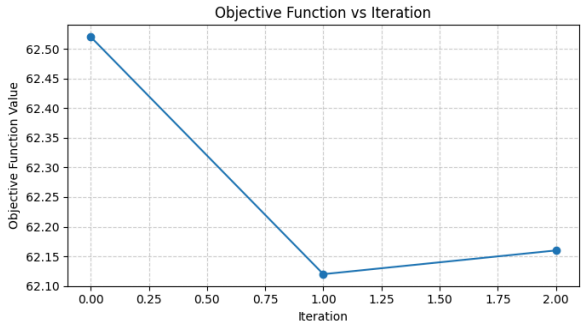
Penjelasan:
Dari hasil run sebanyak 3 kali yang sudah dilakukan, dapat dilihat bahwa simulated annealing **tidak selalu akan menemukan global maxima** hal tersebut terbukti dengan dari 3 kali run, hanya 1 yang dapat menghasilkan hasil yang lebih baik dari solusi awal. Selain itu dapat dilihat juga bahwa simulated annealing biasanya memakan waktu yang lebih lama dari algoritma lain karena iterasi yang dilakukan sangatlah banyak, walaupun jumlah iterasi bisa ditentukan dari temperatur awal, temperatur akhir, prob, dan jumlah iterasi, namun semakin banyak iterasi yang dilakukan, semakin baik pula solusinya

2. Steepest Ascent Hill Climbing

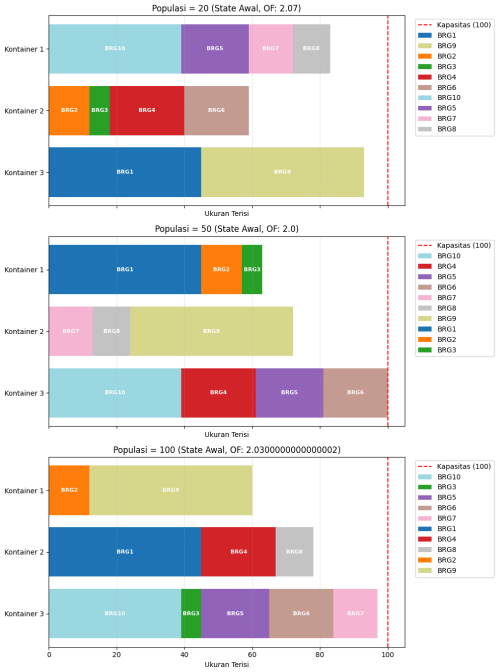
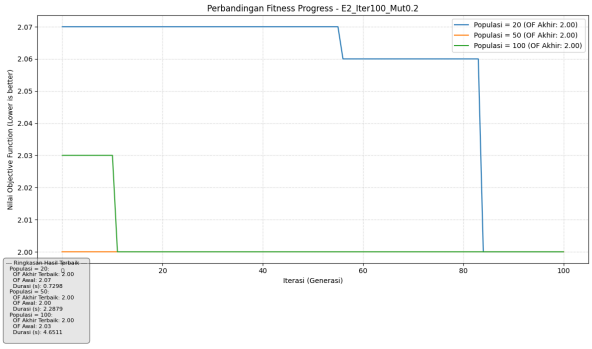
Hasil	Dokumentasi
State Awal #1	

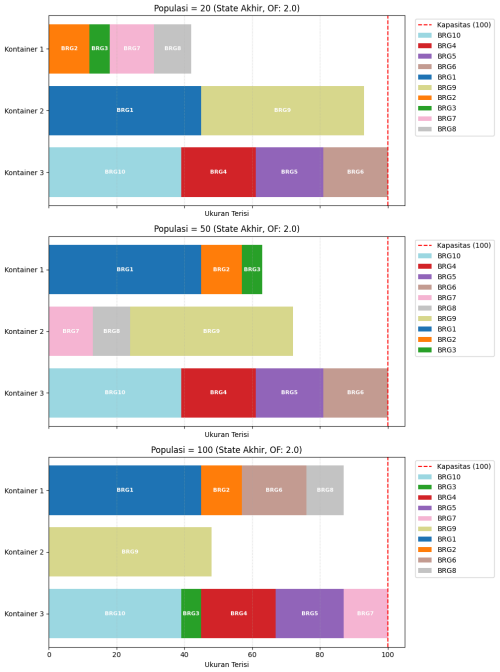
Objection Function Akhir #1	
Plot objective function terhadap banyak iterasi yang telah dilewati #1	
Banyak iterasi #1	
Durasi proses pencarian #1	
State Awal #2	

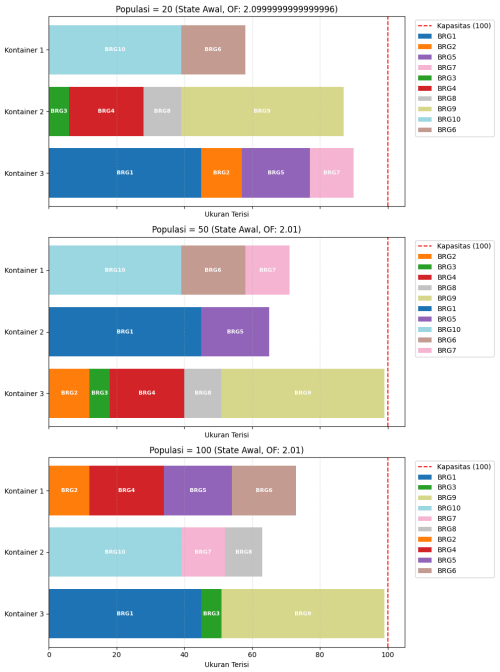
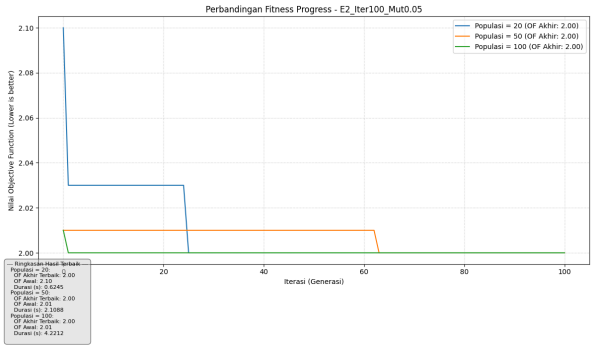
Objection Function Akhir #2	
Plot objective function terhadap banyak iterasi yang telah dilewati #2	
Banyak iterasi #2	
Durasi proses pencarian #2	
State Awal #3	

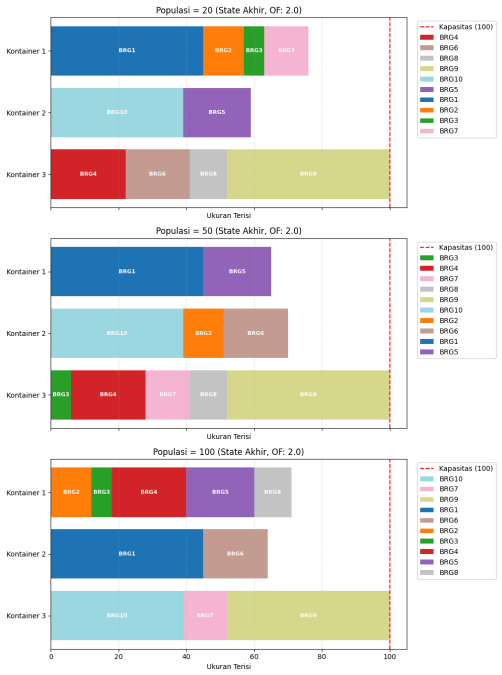
Objection Function Akhir #3	
Plot objective function terhadap banyak iterasi yang telah dilewati #3	
Banyak iterasi #3	iterasi: 3 final value: 62.12
Durasi proses pencarian #3	Initial Value: 69.37 time: 18270.286560058594 ms
Penjelasan: Algoritma yang diimplementasikan sebenarnya lebih ke Steepest Descent Hill Climbing, karena value yang dicari adalah value minimal. Algoritma ini tidak menjamin dapat menemukan Global Maxima, karena akan berhenti ketika semua neighbour-nya tidak lebih baik. Dengan algoritma ini jika menemukan local maxima maka iterasi juga akan berhenti. Dari hasil run algoritma, jumlah iterasi terhitung sedikit, karena algoritma bisa saja terjebak di local maxima.	

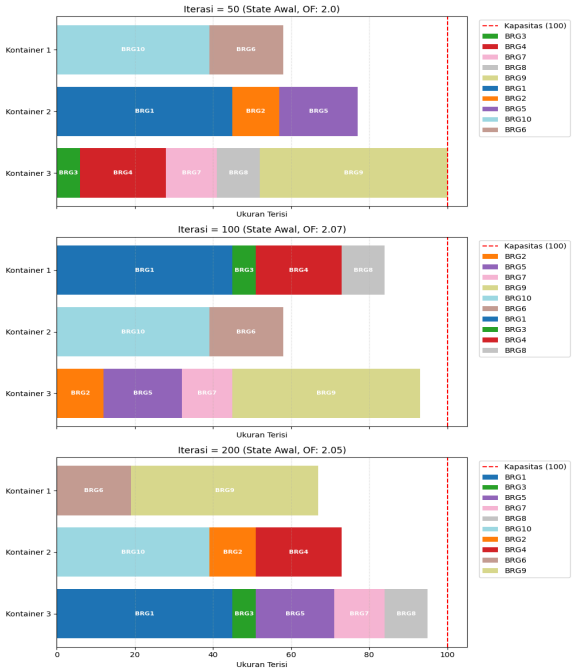
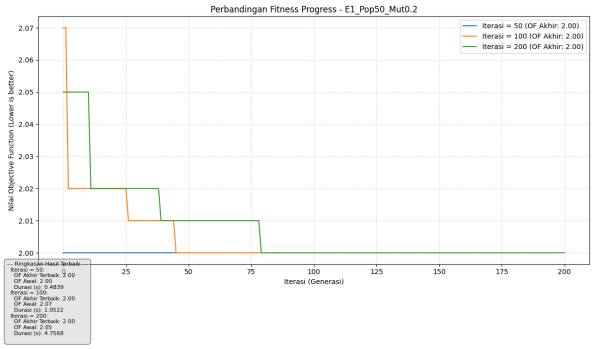
3. Genetic Algorithm

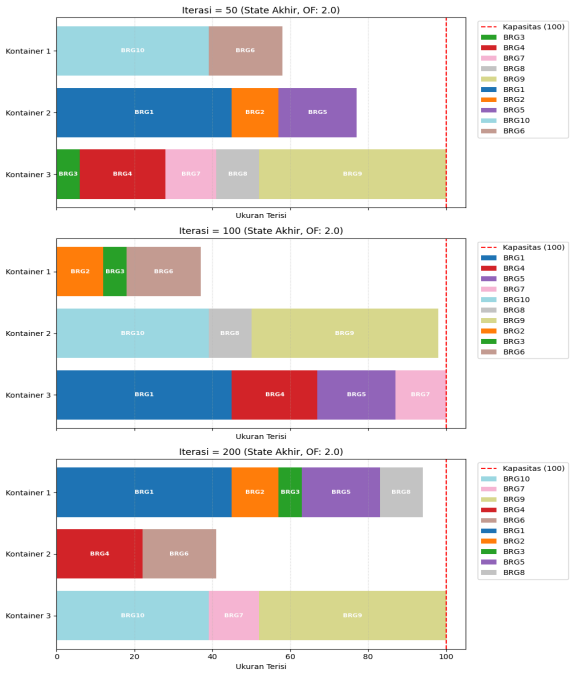
Hasil	Dokumentasi
Banyak Iterasi sebagai kontrol (Probabilitas mutasi 0.2); Iterasi = 100	
Jumlah Populasi Eksperimen #1, #2, dan #3	20, 50, 100
State Awal Eksperimen #1, #2, dan #3	
Plot objective function terhadap banyak iterasi yang telah dilewati eksperimen #1, #2, dan #3	

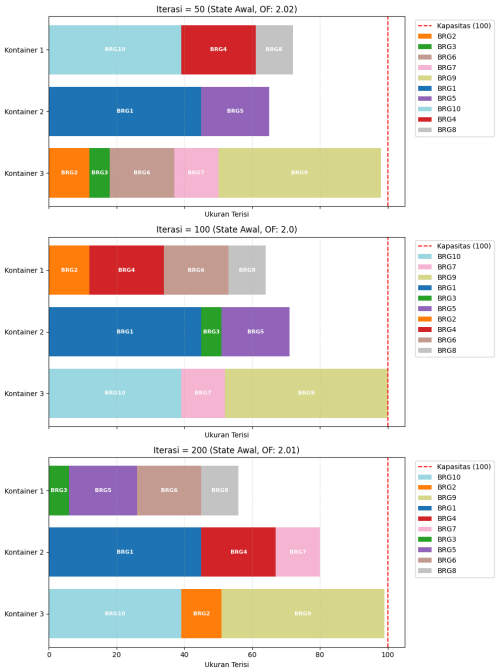
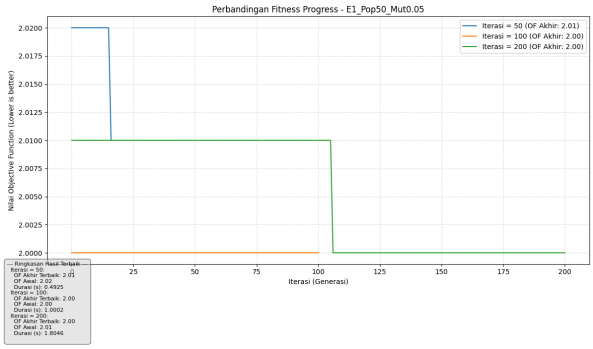
Objection Function Akhir dan Durasi proses pencarian eksperimen #1, #2, dan #3	Populasi = 20: OF Akhir Terbaik: 2.00 OF Awal: 2.07 Durasi (s): 0.7298 Populasi = 50: OF Akhir Terbaik: 2.00 OF Awal: 2.00 Durasi (s): 2.2879 Populasi = 100: OF Akhir Terbaik: 2.00 OF Awal: 2.03 Durasi (s): 4.6511
State Akhir Eksperimen #1, #2, dan #3	
Banyak Iterasi sebagai kontrol (Probabilitas mutasi 0.05); Iterasi = 100	
Jumlah Populasi Eksperimen #1, #2, dan #3	20, 50, 100

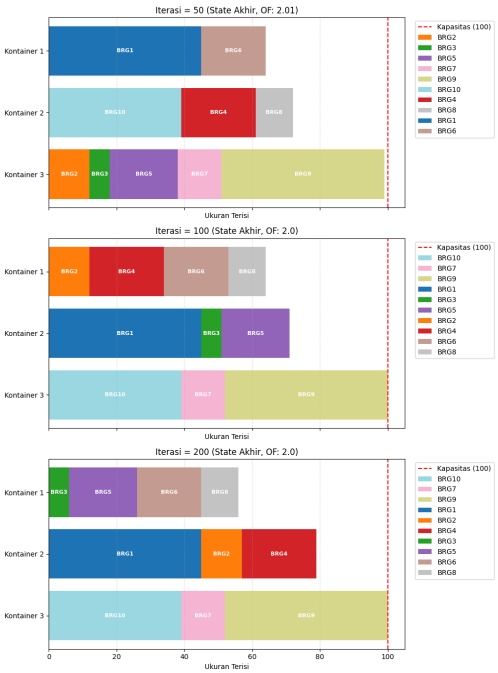
State Awal Eksperimen #1, #2, dan #3	
Plot objective function terhadap banyak iterasi yang telah dilewati eksperimen #1, #2, dan #3	

Objection Function Akhir dan Durasi proses pencarian eksperimen #1, #2, dan #3	Populasi = 20: OF Akhir Terbaik: 2.00 OF Awal: 2.10 Durasi (s): 0.6245 Populasi = 50: OF Akhir Terbaik: 2.00 OF Awal: 2.01 Durasi (s): 2.1088 Populasi = 100: OF Akhir Terbaik: 2.00 OF Awal: 2.01 Durasi (s): 4.2212
State Akhir Eksperimen #1, #2, dan #3	
Pengaruh nilai probabilitas mutasi	Pengaruh probabilitas mutasi tidak dapat diukur secara langsung jika melihat perbandingan kedua plot
Jumlah Populasi sebagai kontrol (Probabilitas mutasi = 0.2); Populasi = 50	
Banyak Iterasi eksperimen #1, #2, dan #3	50, 100, dan 200

State Awal Eksperimen #1, #2, dan #3	
Plot objective function terhadap banyak iterasi yang telah dilewati eksperimen #1, #2, dan #3	

Objection Function Akhir dan Durasi proses pencarian eksperimen #1, #2, dan #3	Iterasi = 50: OF Akhir Terbaik: 2.00 OF Awal: 2.00 Durasi (s): 0.4839 Iterasi = 100: OF Akhir Terbaik: 2.00 OF Awal: 2.07 Durasi (s): 1.0522 Iterasi = 200: OF Akhir Terbaik: 2.00 OF Awal: 2.05 Durasi (s): 4.7568
State Akhir Eksperimen #1, #2, dan #3	
Jumlah Populasi sebagai kontrol (Probabilitas mutasi 0.05); Populasi = 50	
Banyak Iterasi Eksperimen #1, #2, dan #3	50, 100, dan 200

State Awal Eksperimen #1, #2, dan #3	
Plot objective function terhadap banyak iterasi yang telah dilewati eksperimen #1, #2, dan #3	

Objection Function Akhir dan Durasi proses pencarian eksperimen #1, #2, dan #3	Iterasi = 50: OF Akhir Terbaik: 2.01 OF Awal: 2.02 Durasi (s): 0.4925 Iterasi = 100: OF Akhir Terbaik: 2.00 OF Awal: 2.00 Durasi (s): 1.0002 Iterasi = 200: OF Akhir Terbaik: 2.00 OF Awal: 2.01 Durasi (s): 1.8046
State Akhir Eksperimen #1, #2, dan #3	
Pengaruh nilai probabilitas mutasi	Pengaruh probabilitas mutasi tidak dapat diukur secara langsung jika melihat perbandingan kedua plot
Penjelasan: Hasil eksperimen di atas menunjukkan bahwa Genetic Algorithm sangat efektif karena mampu menemukan solusi global optima terus menerus. Dari 12 percobaan yang dilakukan, semuanya dapat mencapai optimal global dan tidak ada yang stuck di local optimum. Namun tentunya efisiensi dari Genetic Algorithm sendiri juga ditentukan oleh *tuning* dari variabel-variabelnya (probabilitas mutasi, iterasi, dan populasi)	

B. Analisis

Berdasarkan data yang dihasilkan dari 3 algoritma, dapat terlihat bahwa Genetic Algorithm merupakan algoritma yang paling efektif untuk mendapatkan global optima, diikuti oleh simulated annealing, dan hill climbing pada urutan terakhir, hal ini bisa terjadi karena hill climbing sangat mudah terjebak pada local optima, sementara simulated annealing masih mempunyai cara untuk keluar dari local optima, terakhir untuk Genetic Algorithm sendiri mempunyai cara yang memiliki tingkat keberhasilan tinggi untuk keluar dari local optima. Dari hasil pencarian, dapat terlihat juga bahwa simulated annealing sering terjebak dalam local optima, sementara simulated annealing dan genetic algorithm mempunyai kemungkinan yang jauh lebih kecil untuk terjebak pada local optima dan mempunyai kemungkinan lebih besar untuk menemukan global optima.

Untuk waktu yang dibutuhkan sendiri, hill climbing membutuhkan waktu yang paling sedikit, lalu untuk genetic dan simulated annealing algorithm, waktu eksekusi sangat bergantung pada konfigurasi yang dilakukan seperti berapa banyak iterasi yang dilakukan pada simulated annealing dan tuning pada genetic algorithm. Hasil yang didapatkan pun memiliki kekonsistenan nya masing masing, hill climbing dan simulated annealing terbilang kurang konsisten karena terkadang masih stuck di local optima, sementara genetic algorithm lebih konsisten karena lebih sering mendapatkan global optima.

Tentunya genetic algorithm memiliki tantangan sendiri untuk mencapai tingkat akurasi tersebut, yaitu *tuning* banyak iterasi serta jumlah populasi yang sesuai. Semakin besar populasi, semakin banyak variasi *parent* yang ada sehingga akan memiliki banyak variasi solusi pula. Banyak iterasi juga sangat berpengaruh karena mempengaruhi banyak kemungkinan terjadinya persilangan antar populasi. Keduanya harus dikalibrasi dengan tepat agar tetap memberikan hasil maksimal dengan seefisien mungkin

IV. Kesimpulan dan Saran

Dari banyaknya eksperimen-eksperimen yang telah dilakukan, dapat disimpulkan bahwa walaupun memiliki algoritma yang paling rumit, Genetic Algorithm terbukti sangat efektif dalam mencari global optima dibandingkan Simulated Annealing ataupun Hill Climbing. Tidak hanya itu, tingkat keefisienan yang dapat dicapai Genetic Algorithm juga dapat jauh lebih unggul dibanding Simulated Annealing dan Hill Climbing jika memiliki *tuning* serta algoritma yang tepat

Saran dalam melakukan pengerjaan tugas ini adalah sebaiknya dilakukan dalam jangka waktu yang lama (menyicil). Selain itu, perbanyak membaca referensi agar tidak kesulitan dalam memahami serta membuat algoritmanya, sehingga algoritma yang dihasilkan lebih optimal.

V. Pembagian Tugas Tiap Kelompok

Nama Mahasiswa	:	Brandon Theodore Ferrinov
NIM	:	18223020

NO	**KEGIATAN**
1	Membuat algoritma simulated annealing
2	Membuat repository github
3	Membuat template dokumen laporan
4	Mengerjakan laporan bagian simulated annealing
5	Debugging Objective Function

Nama Mahasiswa	:	Rafli Dwi Nugraha
NIM	:	18223038

NO	**KEGIATAN**
1	Membuat algoritma hill climbing
2	Membuat objective function
3	Membuat class State, Barang, dan Container
4	Mengerjakan laporan bagian hill climbing

Nama Mahasiswa	:	Luckman Fakhmanidris Arvasirri
NIM	:	18223041

NO	**KEGIATAN**
1	Membuat algoritma genetic
2	Mengerjakan laporan bagian genetic algorithm
3	
4	

Referensi

https://www.geeksforgeeks.org/dsa/implement-simulated-annealing-in-python/
https://www.geeksforgeeks.org/dsa/simulated-annealing/
https://machinelearningmastery.com/simulated-annealing-from-scratch-in-python/
https://www.machinelearningplus.com/machine-learning/simulated-annealing-algorithm-explained-from-scratch-python/
https://gist.github.com/MNoorFawi/4dcf29d69e1708cd60405bd2f0f55700

https://ai.stackexchange.com/questions/240/what-exactly-are-genetic-algorithms-and-what-sort-of-problems-are-they-good-for
https://medium.com/@AnasBrital98/genetic-algorithm-explained-76dfbc5de85d