

**Tugas Besar 1**  
**IF3070 Dasar Intelelegensi Artifisial**  
**Semester I 2025/2026**

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

**Oleh,**

Wisa Ahmaduta Dinutama	18223003
Persada Ramiiza Abyudaya	18223033
Inggried Amelia Deswany	18223035



**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2025**

## DAFTAR ISI

### **BAB I**

#### **DESKRIPSI persoalan..... 4**

1. Persoalan dan Representasi.....	4
1.1. Daftar Variabel.....	4
1.2. Ketentuan Representasi.....	5

### **BAB II**

#### **PEMBAHASAN..... 6**

1. Pemilihan Fungsi Objektif.....	6
2. Implementasi Algoritma Local Search.....	7
2.1. Objective Function.....	7
2.1.1. Kode dan Penjelasan Implementasi.....	7
2.2. Hill-Climbing (HC).....	10
2.2.1. Kode dan Penjelasan Implementasi.....	12
2.2.1.1. Stochastic Hill-Climbing.....	12
2.2.1.2. Steepest Ascent Hill-Climbing.....	16
2.2.1.3. Hill-Climbing with Sideways Move.....	21
2.2.1.4. Random Restart Hill-Climbing.....	29
2.3. Simulated Annealing (SA).....	40
2.3.1. Kode dan Penjelasan Implementasi.....	40
2.4. Genetic Algorithm (GA).....	54
2.4.1. Kode dan Penjelasan Implementasi.....	54
3. Implementasi Visualisasi dan Eksperimen.....	60
3.1. Eksperimen.....	60
3.2. Eksperimen 1: Hill Climbing.....	61
3.2.1. Stochastic Hill-Climbing.....	61
3.2.2. Steepest Ascent Hill-Climbing.....	64
3.2.3. Hill-Climbing with Sideways Move.....	68
3.2.4. Random Restart Hill-Climbing.....	71
3.3. Eksperimen 2: Simulated Annealing.....	75
3.4. Eksperimen 3: Genetic Algorithm.....	82
4. Hasil Eksperimen dan Analisis.....	85
4.1. Stochastic Hill-Climbing.....	85
1. Eksperimen 1 - Stochastic Hill-Climbing.....	86
2. Eksperimen 2 - Stochastic Hill-Climbing.....	87
3. Eksperimen 3 - Stochastic Hill-Climbing.....	88

4.2. Steepest Ascent Hill-Climbing.....	90
1. Eksperimen 1 - Steepest Ascent Hill-Climbing.....	90
2. Eksperimen 2 - Steepest Ascent Hill-Climbing.....	92
3. Eksperimen 3 - Steepest Ascent Hill-Climbing.....	93
4.3. Hill-Climbing with Sideways Move.....	95
1. Eksperimen 1 - Hill-Climbing with Sideways Move.....	95
2. Eksperimen 2 - Hill-Climbing with Sideways Move.....	97
3. Eksperimen 3 - Hill-Climbing with Sideways Move.....	98
4.4. Random Restart Hill-Climbing.....	100
1. Eksperimen 1 - Random Restart Hill-Climbing.....	100
2. Eksperimen 2 - Random Restart Hill-Climbing.....	102
3. Eksperimen 3 - Random Restart Hill-Climbing.....	104
4.5. Simulated Annealing.....	105
1. Eksperimen 1 - Simulated Annealing.....	106
2. Eksperimen 2 - Simulated Annealing.....	108
3. Eksperimen 3 - Simulated Annealing.....	110
4.6. Genetic Algorithm.....	112
1. Variasi Jumlah Populasi (Probabilitas Mutasi = 0.15).....	113
2. Variasi Jumlah Maksimal Iterasi (Probabilitas Mutasi = 0.15).....	127
3. Variasi Jumlah Populasi (Probabilitas Mutasi = 0.5).....	141
4. Variasi Jumlah Iterasi (Probabilitas Mutasi = 0.5).....	154
<b>BAB III</b>	
<b>KESIMPULAN DAN SARAN.....</b>	<b>168</b>
1. Kesimpulan.....	168
2. Saran.....	168
<b>BAB IV</b>	
<b>PEMBAGIAN TUGAS.....</b>	<b>169</b>
1. Pembagian Tugas.....	169
<b>REFERENSI.....</b>	<b>171</b>
<b>LAMPIRAN.....</b>	<b>173</b>
Lampiran 1. Tautan GitHub.....	173
Lampiran 2. Tautan Google Colab.....	173

## BAB I

# DESKRIPSI persoalan

### 1. Persoalan dan Representasi

Dalam tugas besar ini, permasalahan utama yang dihadapi adalah optimasi alokasi barang ke dalam kontainer. Permasalahan tersebut adalah “*Bin Packing Problem*” yang memiliki tujuan untuk menempatkan sekumpulan barang dengan ukuran yang berbeda-beda ke dalam sejumlah kontainer. Adapun syarat dari kontainer itu adalah kapasitas kontainer yang sama dan dengan mengusahakan jumlah kontainer sesedikit mungkin. Terdapat spesifikasi dan aturan yang menyarankan implementasi fungsi objektif dengan sistem penalti besar untuk pelanggaran kapasitas.

#### 1.1. Daftar Variabel

Berikut merupakan barang dengan atribut dan ukuran dalam satuan seragam dan kontainer dengan kapasitas yang seragam.

**Tabel 1.1** Daftar Spesifikasi Variabel

Entitas	Atribut	Contoh	Keterangan
Barang	ID Barang	BRG001	Kode unik untuk setiap barang.
	Ukuran	5	Ukuran atau berat barang (dalam unit yang sama, misal: kg atau m <sup>3</sup> ).
Kontainer	Kapasitas	20	Setiap kontainer memiliki kapasitas yang sama dan seragam.

Adapun pemetaan dari daftar variabel dengan implementasi pada kode adalah sebagai berikut:

**Tabel 1.2** Pemetaan Implementasi Kode

Kode	Deskripsi

items: Dict[str, int]	Berfungsi untuk memetakan ID barang ke ukuran atau berat (contoh di kode: '1':5, '2':9, ...).
containers: List[List[str]]	Merupakan kumpulan kontainer di mana tiap kontainer berisi list ID barang.
max_capacity: int	Merupakan kapasitas kontainer (contoh pengujian menggunakan 50).
<b>Helper Function (Fungsi Pembantu)</b>	
initialize()	Membuat state awal (acak), sejalan dengan inisialisasi bebas di spesifikasi.
shift() dan swap()	Merupakan operator move yang diperbolehkan.

## 1.2. Ketentuan Representasi

Struktur data yang digunakan untuk merepresentasikan dibebaskan sesuai kebutuhan oleh spesifikasi yang tertera. Dalam implementasi, digunakan *list of lists* di mana *state* merupakan alokasi dari *item* yang akan dimasukkan atau dialokasikan ke dalam kontainer. *List* internal akan merepresentasikan satu kontainer dan berisi ID barang yang tersedia di dalamnya. Sementara itu, inisialisasi atau *initial state* juga dibebaskan dengan pilihan metode acak atau menggunakan algoritma heuristik yang sederhana. Dalam implementasi, digunakan inisialisasi acak untuk mencapai maksimum dengan hasil *random* yang dihasilkan oleh algoritma acak. Terdapat dua *move* yang diperbolehkan, yaitu memindahkan satu barang dari satu kontainer ke kontainer lain (yang sudah ada atau yang baru) dan menukar dua barang dari dua kontainer yang berbeda. Kedua *move* ini diimplementasikan dalam *swap* dan *shift*.

## BAB II

### PEMBAHASAN

#### 1. Pemilihan Fungsi Objektif

Fungsi objektif dirancang untuk menyeimbangkan validitas solusi, jumlah kontainer, dan kepadatan isi. Penalti besar pada *overload* atau *overflow* menjamin hanya solusi valid yang dipertimbangkan. Bobot terbesar diberikan pada jumlah kontainer ( $w_{\text{CONT}}$ ) sebagai fokus utama optimasi, disusul penalti ringan untuk ruang kosong ( $w_{\text{UNUSED}}$ ) dan reward kepadatan ( $w_{\text{PACK}}$ ) guna membedakan solusi dengan jumlah kontainer sama.

**Tabel 2.1** Pemilihan Fungsi Objektif

Komponen	Rumus	Alasan	Relevansi
Jumlah Kontainer	$\text{num\_cont} * w_{\text{CONT}}$	Memberi penalti untuk setiap kontainer yang digunakan. Semakin banyak kontainer, semakin besar nilainya.	Tujuan utama adalah meminimalkan jumlah kontainer yang digunakan.
Penalti <i>Overflow</i>	$(w - \max_{\text{capacity}}) * w_{\text{OVER}}$ diakumulasikan ke $\text{overflow\_penalty}$	Jika ada kontainer yang isinya melebihi kapasitas, diberikan penalti yang besar.	Setiap kontainer yang total ukuran barangnya melebihi kapasitas harus diberi penalti yang sangat besar.
Ruang Kosong Total	$\text{total\_unused} * w_{\text{UNUSED}}$	Penalti kecil untuk total ruang kosong (memadatkan	Memprioritaskan <i>state</i> yang menggunakan lebih

		barang).	sedikit kontainer dan lebih padat
Reward Kepadatan	- W_PACK * sum_w2	Menggunakan sum of squares of container weights sebagai reward. Semakin berat beberapa kontainer (lebih padat), nilai objektif makin kecil.	Spesifikasi memperbolehkan reward untuk kepadatan dan membebaskan perancangan formula.
Bobot (Weights)	W_CONT=1_000, W_OVER=1_000_000, W_UNUSED=W_PACK=0 .001	Bobot dipilih agar urutan prioritas sesuai kebutuhan.	Menunjukkan pertimbangan besar pengaruh tiap faktor.

## 2. Implementasi Algoritma Local Search

Sesuai cakupan yang diwajibkan, implementasi memuat (i) satu varian *hill-climbing*, (ii) *Simulated Annealing*, (iii) *Genetic Algorithm*, dan eksperimen dengan *logging state*, objektif, *plot*, atau durasi.

### 2.1. Objective Function

*Objective function* atau fungsi nilai objektif diperlukan untuk menggabungkan penalti dan reward agar algoritma *local search* dapat menilai “seberapa baik” suatu *state* atau solusi.

#### 2.1.1. Kode dan Penjelasan Implementasi

Berikut merupakan kode implementasi dari algoritma *objective function*.

```
@staticmethod
def objective_fun(containers, max_capacity,
```

```
items) :  
    W_CONT    = 1_000  
    W_OVER    = 1_000_000  
    W_UNUSED= 0.001  
    W_PACK    = 0.001  
  
    num_cont = len(containers)  
    total_unused = 0  
    sum_w2 = 0  
    overflow_penalty = 0  
  
    for container in containers:  
        w = sum(items[i] for i in container)  
        if w > max_capacity:  
            overflow_penalty += (w -  
max_capacity) * W_OVER  
        else:  
            total_unused += (max_capacity - w)  
            sum_w2 += w * w  
  
            penalty = (  
                num_cont * W_CONT +  
                total_unused * W_UNUSED +  
                overflow_penalty -  
                W_PACK * sum_w2  
            )  
    return penalty
```

Berikut merupakan langkah dari implementasi *objective function*.

1. Fungsi nilai objektif menghitung nilai penalti di mana semakin kecil nilai tersebut, solusi akan dianggap semakin baik.

2. Pertama, ditentukan terlebih dahulu beberapa bobot (*hyperparameter*) yang akan digunakan dalam algoritma.
  - a. **w\_CONT** → Merupakan parameter untuk penalti setiap kontainer.
  - b. **w\_OVER** → Merupakan parameter untuk penalti kalau kapasitas terlampaui.
  - c. **w\_PACK** → Merupakan parameter untuk *reward* jika kotak terisi dengan padat.
3. Hal selanjutnya yang akan dilakukan adalah menghitung berapa banyak kontainer yang dipakai (`num_cont`) dan siapkan variabel untuk menampung total ruang kosong (`total_unused`), jumlah kuadrat berat (`sum_w2`), dan penalti overflow (`overflow_penalty`).
4. Kemudian, tiap kontainer akan diperiksa satu-per-satu. Untuk setiap kontainer kita jumlahkan berat semua item di dalamnya menjadi `w`.
5. Jika `w` lebih besar dari `max_capacity`, akan ditambahkan penalti overflow sebesar selisih dikali `w_OVER`, agar solusi yang melanggar kapasitas akan terasa “sangat mahal”.
6. Jika `w` tidak melebihi kapasitas, kita tambahkan ruang kosong kontainer itu ke `total_unused` (yaitu `max_capacity - w`), agar solusi dengan ruang berlebih mendapat penalti kecil.
7. Untuk setiap kontainer, akan ditambahkan juga dengan `w * w` ke `sum_w2` yang akan dipakai untuk memberi insentif agar beberapa kontainer penuh (kuadrat membuat kondisi penuh mendapat nilai lebih besar).
8. Setelah semua kontainer dihitung, kita gabungkan semua komponen menjadi nilai penalti akhir, yaitu `num_cont * w_CONT + total_unused * w_UNUSED + overflow_penalty - w_PACK * sum_w2`.

9. Setiap komponen punya perannya:  $\text{num\_cont} * \text{W\_CONT}$  memberi penalti jika terlalu banyak kontainer,  $\text{total\_unused} * \text{W\_UNUSED}$  mendorong agar ruang digunakan seefisien mungkin,  $\text{overflow\_penalty}$  dengan bobot besar ( $\text{W\_OVER}$ ) membuat solusi yang melebihi kapasitas langsung ditolak, dan  $\text{W\_PACK} * \text{sum\_w2}$  menjadi *reward* bagi solusi yang padat karena kontainernya terisi penuh.
10. Fungsi lalu mengembalikan nilai penalti total tersebut dan algoritma akan berusaha meminimalkan nilai hal tersebut agar solusi jadi sedikit kotak, tidak *overflow*, sedikit ruang kosong, dan beberapa kotak benar-benar penuh.

## 2.2. Hill-Climbing (HC)

Berikut merupakan implementasi dari algoritma *local search* yang mengimplementasikan beberapa varian dari *hill-climbing*, sebagaimana yang tertera pada spesifikasi wajib dan bonus.

**Tabel 2.2** Hill-Climbing (HC) Implementation

Varian	Deskripsi
<i>Stochastic Hill-Climbing</i>	Proses ini memanfaatkan banyak iterasi hingga pencarian berhenti dilakukan. Dalam implementasi <i>stochastic hill-climbing</i> , digunakan fungsi <i>move</i> yang memilih <i>item</i> dan aksi ( <i>shift</i> atau <i>swap</i> ) secara acak (inisialisasi acak). Solusi akan diterima apabila objektif sudah menurun. Eksperimen merekam nilai objektif per <i>batch</i> iterasi untuk <i>plotting</i> .

<i>Steepest Ascent Hill-Climbing</i>	Proses ini memanfaatkan banyak iterasi hingga pencarian berhenti dilakukan. Varian ini membentuk semua <i>neighbour</i> unik via <i>shift</i> (termasuk kontainer baru) dan <i>swap</i> yang mengevaluasi semuanya. Kemudian, algoritma akan memilih <i>neighbour</i> terbaik dengan nilai objektif minimum. <i>Class</i> sudah menyimpan riwayat nilai objektif ( <i>history</i> ) untuk visualisasi.
<i>Hill-Climbing with Sideways Move</i>	Konsep dan implementasi dari <i>hill-climbing with sideways move</i> tidak berbeda jauh dengan <i>steepest ascent hill-climbing</i> . Adapun langkah perbedaan adalah langkah <i>sideways</i> (nilai objektif sama) sampai batas maksimum ( <i>max_sideways</i> ). Setelah mencapai batas, algoritma akan berhenti dengan memenuhi ketentuan “tambahkan parameter <i>maximum sideways move</i> ”. <i>Logging</i> menyertakan total <i>sideways</i> & durasi.
<i>Random Restart Hill-Climbing</i>	Menjalankan <i>hill-climbing</i> dari beberapa inisialisasi acak dengan parameter penting <i>max_restarts</i> dan <i>iters_per_restart</i> . Kode sudah menyiapkan ringkasan objektif terbaik per restart, total iterasi, dan durasi.

	Terdapat dua <i>plot</i> yang merupakan kurva min <i>per restart</i> dan riwayat <i>best restart</i> .
--	--

### 2.2.1. Kode dan Penjelasan Implementasi

#### 2.2.1.1. Stochastic Hill-Climbing

Berikut merupakan kode implementasi dari algoritma *stochastic hill-climbing*.

```
class StochasticHillClimbing(BinPackingBase):  
    @staticmethod  
    def move(containers, items, max_capacity):  
        non_empty_containers = [c for c in  
containers if c]  
        if not non_empty_containers:  
            return containers  
  
        random_container =  
rand.choice(non_empty_containers)  
        random_item =  
rand.choice(random_container)  
        random_move = rand.randint(0,1)  
        successor = copy.deepcopy(containers)  
  
        if random_move == 0:  
            all_items = [item for container in  
non_empty_containers for item in container]  
            if len(all_items) < 2:  
                return containers  
            random_item2 = rand.choice(all_items)  
            BinPackingBase.swap(random_item,  
random_item2, successor)  
        else:
```

```
        target_container_index = rand.randint(0,
len(successor))

        BinPackingBase.shift(random_item,
target_container_index, successor)

successor = [c for c in successor if c]

if BinPackingBase.objective_fun(successor,
max_capacity, items) <
BinPackingBase.objective_fun(containers,
max_capacity, items):

    return successor
else:
    return containers

@staticmethod
def run(items, max_capacity=50,
max_iter=500, seed=None):
    if seed is not None:
        rand.seed(seed)

    time_start = time.time()
    containers = []
    if items:
        BinPackingBase.initialize(containers,
items)
        containers = [c for c in containers if
c]
        if not containers:
            containers =
[[list(items.keys())[0]]]
            for item in
list(items.keys())[1:]:
                BinPackingBase.shift(item,
```

```
rand.randint(0, len(containers)), containers)
    containers = [c for c in
containers if c]

    initial_state = copy.deepcopy(containers)
    objective_values =
[BinPackingBase.objective_fun(containers,
max_capacity, items)]

    for _ in range(max_iter):
        containers =
StochasticHillClimbing.move(containers, items,
max_capacity)
        if not containers:
            containers = [[]]

    objective_values.append(BinPackingBase.objective_fun(containers, max_capacity, items))

    duration = time.time() - time_start
    return initial_state, containers,
objective_values, duration
```

Implementasi algoritma ini memanfaatkan dua *static method* (dua bagian), yaitu *move()* dan *run()*. Metode *move()* merupakan metode untuk memindahkan atau mengubah solusi secara acak. Salah satu contohnya adalah mencoba langkah baru untuk melihat apakah hasilnya bisa lebih baik dari *current state*. Kemudian *run()* digunakan untuk menjalankan keseluruhan proses pencarian solusi dengan proses iterasi sampai bertemu dengan hasil yang terbaik. Berikut merupakan implementasi dari bagian pertama yang mengimplementasikan *move()*.

1. Pertama, program akan mencari kontainer yang masih berisi *item*. Jika semua kontainer kosong, berarti tidak ada yang bisa dipindah. Sehingga, solusi dikembalikan apa adanya (dibiarkan menjadi *current state*).
2. Dari kontainer yang dipilih, akan diambil satu *item* atau barang secara acak. Kemudian *move* yang diambil bisa berupa *swap* dan *shift* dan digunakan *move()* agar perubahan tidak mengubah solusi asli.
3. Jika yang dipilih adalah *swap*, program akan memilih satu *item* lain secara acak dari semua kontainer. Kemudian, kedua *item* atau barang akan ditukar tempatnya.
4. Jika yang dipilih adalah *shift*, program akan memilih satu kontainer acak (tujuan). *Item* atau barang akan dipindahkan ke kontainer yang sudah ada.
5. Setelah perpindahan (*shift*) atau pertukaran (*swap*), ada kemungkinan ada kontainer yang kosong. Kontainer tersebut akan dihapus agar solusi tetap efisien.
6. Program akan menghitung nilai objektif (misalnya berapa banyak kontainer dipakai atau seberapa optimal kapasitasnya). Semakin kecil nilai objektif maka akan semakin baik. Sehingga, jika solusi baru lebih baik maka solusi tersebut akan dipakai.

Berikut merupakan implementasi dari bagian pertama yang mengimplementasikan *run()*.

1. Pertama, program akan mencatat waktu mulai agar nanti bisa tahu berapa lama proses berjalan.
2. Kemudian, algoritma akan membuat daftar kontainer yang kosong untuk memulai proses. Fungsi *initialize()* akan dipanggil untuk mengisi *item* atau barang awal. Jika hasil awal masih kosong, buat satu kontainer berisi *item*

atau barang pertama, lalu pindahkan *item* atau barang lain secara acak.

3. Selanjutnya, program akan menyimpan salinan dari solusi pertama. Kemudian, nilai objektif awal akan dihitung sebagai pembanding di iterasi berikutnya.
4. Program akan mengeksekusi *move()* berulang kali hingga mencapai batas maksimum untuk iterasi (*max\_iter*). Setiap langkah ini dijalankan, hasil yang baru akan disimpan dan nilai objektif akan diperbarui. Jika terjadi *error*, akan dibuat kontainer baru sehingga algoritma dapat tetap berjalan.
5. Setelah selesai, program menghitung lama waktu eksekusi. Kemudian akan menampilkan solusi awal, solusi akhir, riwayat nilai objektif di tiap iterasi, dan durasi waktu yang dibutuhkan.

#### 2.2.1.2. *Steepest Ascent Hill-Climbing*

Berikut merupakan kode implementasi dari algoritma *steepest ascent hill-climbing*.

```
class SteepestHill(BinPackingBase):  
    @staticmethod  
    def clean_nonempty(containers):  
        return [c for c in containers if  
len(c) > 0]  
  
    @staticmethod  
    def score(state, items, max_capacity):  
        return  
BinPackingBase.objective_fun(state,  
max_capacity, items)
```

```
@staticmethod
def neighbors_all(state, items,
max_capacity):
    base =
SteepestHill.clean_nonempty(copy.deepcopy(s
tate))
    m = len(base)
    seen = set()
    neighbors = []

    dst_range = range(m + 1)
    for i in range(m):
        for item in base[i]:
            for j in dst_range:
                if j == i:
                    continue
                s = copy.deepcopy(base)

                BinPackingBase.shift(item, j, s)
                s =
                SteepestHill.clean_nonempty(s)
                key = tuple(tuple(c)
for c in s)
                if key not in seen:
                    seen.add(key)
                    neighbors.append(s)

                for a in range(m):
                    for item_a in base[a]:
                        for b in range(a + 1, m):
```

```
for item_b in base[b]:
    s =
    copy.deepcopy(base)

    BinPackingBase.swap(item_a, item_b, s)
    s =
    SteepestHill.clean_nonempty(s)
    key =
    tuple(tuple(c) for c in s)
    if key not in seen:
        seen.add(key)

    neighbors.append(s)

    return neighbors

@staticmethod
def run(items, max_capacity=50,
max_iter=500, seed=None):
    if seed is not None:
        rand.seed(seed)

    containers = []
    state =
    BinPackingBase.initialize(containers,
items)
    start_time = time.time()
    state =
    SteepestHill.clean_nonempty(copy.deepcopy(c
ontainers))
    initial_state = state
```

```
        best_val = SteepestHill.score(state,
items, max_capacity)
        history = [best_val]

        for it in range(1, max_iter + 1):
            best_neighbor = None
            best_neighbor_val = best_val

            for nb in
SteepestHill.neighbors_all(state, items,
max_capacity):
                val = SteepestHill.score(nb,
items, max_capacity)
                if val < best_neighbor_val:
                    best_neighbor_val = val
                    best_neighbor = nb

                if best_neighbor is None or
best_neighbor_val >= best_val:
                    duration = time.time() -
start_time
                    return initial_state, state,
best_val, it, history, duration

            state, best_val = best_neighbor,
best_neighbor_val
            history.append(best_val)
            duration = time.time() - start_time
            return initial_state, state,
best_val, it, history, duration
```

Berikut merupakan mekanisme dari implementasi *steepest ascent hill-climbing*.

1. Kelas `SteepestHill` dibuat dari turunan `BinPackingBase`, jadi otomatis bisa pakai fungsi dasar seperti `shift()`, `swap()`, `initialize()`, dan `objective_fun()` untuk bantu proses perhitungan.
2. Terdapat fungsi `clean_nonempty()` bertugas membersihkan kontainer kosong setelah item dipindah dan `score()` menghitung nilai total penalti dari suatu solusi yang nilainya menunjukkan seberapa baik solusi tersebut.
3. Fungsi `neighbors_all()` menghasilkan semua kemungkinan solusi baru (tetangga) dari kondisi sekarang dengan dua cara,
  - a. *SHIFT* → Memindahkan satu item ke kontainer lain atau ke kontainer baru.
  - b. *SWAP* → Menukar posisi dua item dari dua kontainer berbeda.
4. Saat algoritma membuat tetangga (*neighbour*), algoritma akan menyalin *state* asli agar tidak berubah. Kemudian, hasil dapat dipastikan bernilai unik (tidak duplikat) sebelum nantinya dimasukkan ke daftar tetangga.
5. Fungsi `run()` adalah bagian utama yang menjalankan seluruh proses pencarian solusi terbaik menggunakan metode *steepest hill-climbing*. Kemudian, algoritma akan membuat solusi awal dengan menempatkan semua *item* atau barang secara acak ke dalam beberapa kontainer.
6. Selanjutnya nilai objektif awal (penalti awal) akan dihitung dan disimpan untuk jadi pembanding di tiap langkah.
7. Selanjutnya algoritma akan melakukan iterasi berulang dengan melakukan hal-hal berikut ini,

- a. Membuat semua kemungkinan tetangga dari solusi terkini.
- b. Menghitung nilai dari setiap tetangga.
- c. Memilih satu tetangga dengan nilai terbaik, yaitu nilai penalti terkecil.
8. Jika tidak ada tetangga yang lebih baik, proses akan berhenti karena sudah mencapai kondisi *local optimum* atau lokal optima (tidak bisa diperbaiki lagi). Tetapi, jika ada solusi yang lebih baik, algoritma akan berpindah ke solusi tersebut dan berlanjut ke iterasi berikutnya.
9. Jika ada solusi yang lebih baik, algoritma akan berpindah ke solusi tersebut dan lanjut ke iterasi berikutnya. Proses ini akan diulang sampai mencapai batas iterasi maksimum dan tidak ada perbaikan lagi.
10. Hasil akhir adalah algoritma akan menghasilkan hasil akhir sebagai berikut,
  - a. Solusi awal dan solusi akhir
  - b. Nilai objektif terbaik
  - c. Jumlah iterasi
  - d. Riwayat nilai objektif tiap iterasi
  - e. Waktu eksekusi total

#### **2.2.1.3. *Hill-Climbing with Sideways Move***

Berikut merupakan kode implementasi dari algoritma *hill-climbing with sideways move*.

```
import copy, time
import numpy as np

class SidewaysHill(BinPackingBase):
```

```
@staticmethod
def cleanup_nonempty(containers):
    return [c for c in containers if
len(c) > 0]

@staticmethod
def score(state, items, max_capacity):
    return
BinPackingBase.objective_fun(state,
max_capacity, items)

@staticmethod
def neighbors_all(state, items,
max_capacity):
    base =
SidewaysHill.cleanup_nonempty(copy.deepcopy
(state))
    m = len(base)
    seen = set()
    neighbors = []

    dst_range = range(m + 1)
    for i in range(m):
        for item in base[i]:
            for j in dst_range:
                if j == i:
                    continue
                s = copy.deepcopy(base)

BinPackingBase.shift(item, j, s)
    s =
```

```
SidewaysHill.cleanup_nonempty(s)
    key = tuple(tuple(c)
for c in s)
    if key not in seen:
        seen.add(key)
        neighbors.append(s)

    for a in range(m):
        for item_a in base[a]:
            for b in range(a + 1, m):
                for item_b in base[b]:
                    s =
copy.deepcopy(base)

BinPackingBase.swap(item_a, item_b, s)
    s =
SidewaysHill.cleanup_nonempty(s)
    key =
tuple(tuple(c) for c in s)
    if key not in seen:
        seen.add(key)

neighbors.append(s)
    return neighbors

@staticmethod
def run(items, max_capacity=50,
max_iter=2000, max_sideways=50, seed=None):
    if seed is not None:
        rand.seed(seed)
    containers = []
```

```
state =
BinPackingBase.initialize(containers,
items)

state =
SidewaysHill.cleanup_nonempty(copy.deepcopy
(containers))

initial_state = state
best_val =
SidewaysHill.score(state, items,
max_capacity)

history = [best_val]

start = time.time()
consecutive_sideways = 0
total_sideways = 0

for it in range(1, max_iter + 1):
    best_neighbor = None
    best_neighbor_val = None

    for nb in
        SidewaysHill.neighbors_all(state, items,
max_capacity):
        val =
        SidewaysHill.score(nb, items, max_capacity)
        if (best_neighbor is None)
or (val < best_neighbor_val):
            best_neighbor = nb
            best_neighbor_val = val

    if best_neighbor is None:
```

```
duration = time.time() -
start
return initial_state,
state, best_val, it, history, duration,
total_sideways

if best_neighbor_val <
best_val:
    state = best_neighbor
    best_val =
best_neighbor_val
    history.append(best_val)
    consecutive_sideways = 0
elif best_neighbor_val ==
best_val:
    consecutive_sideways += 1
    total_sideways += 1
    if consecutive_sideways >
max_sideways:
        duration = time.time()
- start
        return initial_state,
state, best_val, it, history, duration,
total_sideways
        state = best_neighbor
        history.append(best_val)
else:
    duration = time.time() -
start
    return initial_state,
state, best_val, it, history, duration,
```

```
total_sideways

    duration = time.time() - start
    return initial_state, state,
best_val, it, history, duration,
total_sideways
```

Berikut merupakan mekanisme dari implementasi *sideways move hill-climbing*.

1. Kelas SidewaysHill adalah variasi dari algoritma Hill-Climbing yang memperbolehkan bergerak ke solusi tetangga dengan nilai objektif sama (sideways move), bukan hanya yang lebih baik. Kelas ini mewarisi (inherits) semua fungsi dari BinPackingBase, jadi bisa langsung menggunakan fungsi dasar seperti shift(), swap(), dan objective\_fun().
2. Tujuan dari *sideways move* adalah untuk menghindari terjebak di *plateau* yang merupakan bagian datar pada grafik nilai objektif. Hal ini dilakukan agar algoritma bisa terus mencari solusi yang lebih baik.
3. Terdapat batas maksimal dari *sideways move* agar tidak berakhir dengan iterasi yang terus berputar tanpa hasil.
4. Adapun fungsi cleanup\_nonempty() yang digunakan untuk menghapus kontainer kosong. Kemudian terdapat fungsi score() menghitung nilai penalti (*objective value*) dari suatu solusi, semakin kecil nilainya berarti solusi semakin baik.
5. Fungsi neighbors\_all() menghasilkan semua kemungkinan solusi baru (tetangga) dari konfigurasi sekarang, melalui dua cara berikut:

- a. *SHIFT* → Memindahkan satu *item* atau barang ke kontainer lain atau membuat kontainer baru.
- b. *SWAP* → Menukar dua *item* atau barang dari dua kontainer yang berbeda.
6. Saat membuat tetangga, algoritma akan membuat dan menyimpan salinan dari *state* agar data asli tidak berubah. Kemudian, tetangga yang unik akan disimpan agar tidak dihitung dua kali.
7. Terdapat fungsi `run()` yang menjalankan seluruh proses pencarian solusi menggunakan *hill-climbing with sideways move*. Algoritma sendiri akan dimulai dengan membuat solusi secara acak. Kemudian nilai objektif pertama akan dijadikan acuan (`best_val`).
8. Terdapat fungsi penghitung *sideways*, yaitu `consecutive_sideways` yang menunjukkan berapa kali berturut-turut *sideways* terjadi dan `total_sideways` yang menunjukkan jumlah keseluruhan *sideways*.
9. Terdapat beberapa mekanisme yang dilakukan algoritma di tiap iterasi:
  - a. Membentuk semua tetangga dari state saat ini.
  - b. Menghitung nilai objektif masing-masing tetangga.
  - c. Memilih tetangga dengan nilai terbaik (paling kecil).
10. Berikut merupakan beberapa kondisional yang mungkin terjadi dalam algoritma:

**Tabel 2.3 Conditionals Sideways**

Kondisi	Penanganan
---------	------------

Jika ada solusi yang lebih baik	Algoritma berpindah ke solusi itu dan mengatur ulang penghitung sideways ke nol.
Jika nilai tetangga sama dengan nilai sekarang	Maka dilakukan sideways move: 1. Naikkan penghitung sideways. 2. Jika sudah melebihi batas <code>max_sideways</code> , algoritma berhenti. 3. Jika belum, tetap lanjut ke solusi itu (harapannya bisa keluar dari <i>plateau</i> nanti).
Jika semua tetangga lebih buruk	Menandakan bahwa sudah mencapai <i>local optimum</i> dan algoritma berhenti.
Jika batas iterasi maksimum tercapai	Algoritma juga berhenti secara otomatis dan menampilkan hasil akhir.

11. Hasil akhir dari algoritma adalah sebagai berikut.

- a. Solusi awal dan akhir
- b. Nilai objektif terbaik
- c. Jumlah iterasi
- d. Riwayat nilai tiap iterasi (untuk grafik)
- e. Durasi eksekusi
- f. Dan total *sideways move* yang dilakukan.

#### 2.2.1.4. Random Restart Hill-Climbing

Berikut merupakan kode implementasi dari algoritma *random restart hill-climbing*.

```
class RandomRestartHill(BinPackingBase):  
    @staticmethod  
    def cleanup_nonempty(containers):  
        return [c for c in containers if  
len(c) > 0]  
  
    @staticmethod  
    def normalize_state(state):  
        norm = [tuple(sorted(c)) for c in  
state if len(c) > 0]  
        norm.sort()  
        return tuple(norm)  
  
    @staticmethod  
    def score(state, items, max_capacity):  
        return  
BinPackingBase.objective_fun(state,  
max_capacity, items)  
  
    @staticmethod  
    def neighbors_all(state, items,  
max_capacity, allow_new_bin=True):  
        base =  
RandomRestartHill.cleanup_nonempty(copy.deepcopy(state))  
        m = len(base)  
        seen = set()  
        neighbors = []
```

```
        dst_range = range(m + 1) if
allow_new_bin else range(m)
        for i in range(m):
            for item in base[i]:
                for j in dst_range:
                    if j == i:
                        continue
                    s = copy.deepcopy(base)

BinPackingBase.shift(item, j, s)
s =
RandomRestartHill.cleanup_nonempty(s)
key =
RandomRestartHill.normalize_state(s)
if key not in seen:
    seen.add(key)
    neighbors.append(s)

for a in range(m):
    for item_a in base[a]:
        for b in range(a + 1, m):
            for item_b in base[b]:
                s =
copy.deepcopy(base)

BinPackingBase.swap(item_a, item_b, s)
s =
RandomRestartHill.cleanup_nonempty(s)
key =
RandomRestartHill.normalize_state(s)
```

```
        if key not in seen:
            seen.add(key)

    neighbors.append(s)

    return neighbors

    @staticmethod
    def single_run(initial_state, items,
max_capacity=50, max_iter=500,
                    allow_new_bin=True):
        state =
RandomRestartHill.cleanup_nonempty(copy.deepcopy(initial_state))
        best_val =
RandomRestartHill.score(state, items,
max_capacity)
        history = [best_val]

        for it in range(1, max_iter + 1):
            best_neighbor,
best_neighbor_val = None, best_val
            for nb in
RandomRestartHill.neighbors_all(state,
items, max_capacity, allow_new_bin):
                val =
RandomRestartHill.score(nb, items,
max_capacity)
                if val < best_neighbor_val:
                    best_neighbor,
best_neighbor_val = nb, val
```

```
        if best_neighbor is None or
best_neighbor_val >= best_val:
            return state, best_val,
it-1, history

        state, best_val =
best_neighbor, best_neighbor_val
        history.append(best_val)

    return state, best_val, max_iter,
history

@staticmethod
def run(items, max_capacity=50,
       max_restarts=20,
       iters_per_restart=500,
       allow_new_bin=True):
    start = time.time()

    best_state, best_obj = None,
float('inf')
    best_history = None
    restart_objectives = []
    total_iters = 0
    last_init_state = None

    all_histories = []
    per_restart_iters = []

    for r in range(1, max_restarts +
1):
```

```
        init_state = []

BinPackingBase.initialize(init_state,
                           items)

        init_state =
RandomRestartHill.cleanup_nonempty(copy.deepcopy(init_state))

        last_init_state = init_state

state_r, obj_r, iters_r, hist_r
= RandomRestartHill.single_run(
    init_state, items,
    max_capacity=max_capacity,
    max_iter=iters_per_restart,
    allow_new_bin=allow_new_bin
)

total_iters += iters_r

restart_objectives.append(obj_r)

all_histories.append(hist_r)

per_restart_iters.append(len(hist_r))

        if obj_r < best_obj:
            best_obj = obj_r
            best_state = state_r
            best_history = hist_r

duration = time.time() - start

return last_init_state, best_state,
```

```
best_obj, len(restart_objectives),
total_iters, restart_objectives,
best_history, duration, all_histories,
per_restart_iters

    @staticmethod
    def
plot_objective_all_histories(all_histories,
duration=None, title="Nilai Objektif
(Gabungan Restart)":

    import matplotlib.pyplot as plt

    xs = []
    ys = []
    restart_boundaries = []
    offset = 0

    per_restart_iters = []

    for r, hist in
enumerate(all_histories, start=1):
        n = len(hist)
        if n == 0:
            continue
        local_x = list(range(offset +
1, offset + n + 1))
        xs.extend(local_x)
        ys.extend(hist)
        offset += n
```

```
restart_boundaries.append(offset)
per_restart_iters.append(n)

if len(xs) == 0:
    print("Tidak ada history untuk
dipetakan.")
    return

fig, ax = plt.subplots(figsize=(10,
5))
ax.plot(xs, ys, marker='o',
linestyle='-', linewidth=1.8)

dur_txt = f" (Durasi:
{duration:.3f} detik)" if duration is not
None else ""
ax.set_title(f"{title}{dur_txt}",
pad=14)
ax.set_xlabel("Iterasi (akumulatif
lintas restart)")
ax.set_ylabel("Nilai Objektif")
ax.grid(True, linestyle='--',
alpha=0.5)

ax.text(xs[0], ys[0],
f"{ys[0]:.3f}", ha='right', va='bottom',
bbox=dict(facecolor='white',
edgecolor='none', alpha=0.8))
ax.scatter([xs[0]], [ys[0]], s=50,
zorder=3)
```

```
        ax.text(xs[-1], ys[-1],
f"{{ys[-1]:.3f}}", ha='left', va='bottom',
bbox=dict(facecolor='white',
edgecolor='none', alpha=0.8))

        ax.scatter([xs[-1]], [ys[-1]],
s=50, zorder=3)

        cum_iters = 0
        y_top = ax.get_ylim()[1]

        for idx, (boundary_x, n_iter) in
enumerate(zip(restart_boundaries,
per_restart_iters), start=1):
            if idx <
len(restart_boundaries):
                ax.axvline(boundary_x +
0.5, linestyle='--', alpha=0.4)

            start_x = boundary_x - n_iter +
1
            mid_x = start_x + (n_iter - 1)
/ 2

            ax.text(
                mid_x, y_top,
                n_iter,
                ha='center', va='bottom',
                fontsize=8,
                rotation=0,
```

```
bbox=dict(facecolor='white',
edgecolor='none', alpha=0.7,
boxstyle='round,pad=0.2')

)

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

Berikut merupakan mekanisme dari implementasi *random restart hill-climbing*.

1. Kelas RandomRestartHill merupakan algoritma *hill-climbing* yang menambahkan fitur *restart* acak. Algoritma akan menghindari jebakan *local optimum* dengan cara mengulang proses pencarian dari titik awal yang berbeda berkali-kali dan kemudian memilih hasil verbosterbaik. Kelas ini juga mewarisi fungsi penting seperti `shift()`, `swap()`, dan `objective_fun()`.
2. Terdapat fungsi `cleanup_nonempty()` yang digunakan untuk menghapus kontainer kosong dan fungsi `normalize_state()` yang membuat setiap konfigurasi (*state*) jadi bentuk unik dapat dikenali dengan urutan yang berbeda. Terakhir, terdapat fungsi `score()` menghitung nilai objektif dari suatu solusi.
3. Fungsi `neighbors_all()` menghasilkan semua kemungkinan solusi baru (tetangga) dari konfigurasi sekarang, melalui dua cara berikut:
  - a. *SHIFT* → Memindahkan satu *item* atau barang ke kontainer lain atau membuat kontainer baru.

- b. *SWAP* → Menukar dua *item* atau barang dari dua kontainer yang berbeda.
4. Kemudian, terdapat fungsi *single\_run()* yang akan menjalankan satu kali proses *hill-climbing* biasa tanpa *restart*. Proses ini diawali dengan membuat salinan *state* awal, menghitung nilai objektifnya, dan menyimpan nilai tersebut ke dalam, daftar *history*.
5. Setiap iterasi, algoritma mencari tetangga dengan nilai objektif paling kecil.

**Tabel 2.4 Conditional Random Restart**

Kondisi	Penanganan
Jika semua tetangga sama atau lebih buruk (tidak ada perbaikan)	Proses akan dihentikan
Jika ada perbaikan	<i>State</i> akan diganti dengan tetangga terbaik dan proses berlanjut hingga mencapai batas iterasi.

12. Setelah satu kali *hill-climbing* selesai, hasilnya akan dikembalikan berupa *state* terbaik dengan nilai objektif terbaik. Jumlah iterasi dan riwayat perubahan akan dihasilkan juga.
13. Fungsi *run()* bertugas melakukan banyak kali *hill-climbing* dengan titik awal acak (*random restart*). Setiap *restart* dimulai dari konfigurasi acak baru yang akan dijalankan dengan *single\_run()*. Nilai hasil akhir dari tiap restart disimpan untuk dibandingkan.
14. Jika dari beberapa restart ditemukan hasil dengan nilai objektif lebih kecil dari sebelumnya, maka hasil itu

dijadikan solusi terbaik sejauh ini. Proses *restart* akan diulang hingga mencapai jumlah maksimum yang ditemukan (`max_restarts`). Sistem juga akan mencatat riwayat semua nilai objektif agar bisa digambarkan dalam bentuk grafik. Setelah semua restart selesai, fungsi mengembalikan seluruh data penting.

15. Fungsi `plot_objective_all_histories()` akan digunakan untuk memvisualisasikan perubahan nilai objektif dari semua *restart* sekaligus. Hasil dijalankan lewat pemanggilan fungsi `run()`, lalu divisualisasikan untuk melihat:
- Konfigurasi awal acak (*initial state*)
  - Solusi terbaik akhir (*best state*)
  - Grafik perubahan nilai objektif dari semua *restart*.

### 2.3. Simulated Annealing (SA)

*Simulated annealing* merupakan algoritma yang menggunakan probabilitas penerimaan solusi berdasarkan formula  $e^{\frac{\Delta E}{T}}$  untuk menghindari terjebak di *local optima*. *Simulated annealing* merupakan metode optimasi probabilistik yang terinspirasi dari proses pendinginan lambat material logam (konsep fisika) untuk menemukan solusi terbaik atau mendekati optimal pada masalah yang kompleks.

#### 2.3.1. Kode dan Penjelasan Implementasi

```
class SimulatedAnnealing(BinPackingBase):  
    @staticmethod  
    def _cleanup_nonempty(containers:  
        List[List[str]]) -> List[List[str]]:  
        return [c for c in containers if c]
```

```
@staticmethod
def random_init(items: Dict[str, int], rng:
random.Random, p_new: float = 0.15) ->
List[List[str]]:
    containers: List[List[str]] = []
    keys = list(items.keys())
    rng.shuffle(keys)
    for it in keys:
        if not containers or rng.random() < p_new:
            containers.append([it])
        else:
            containers[rng.randrange(len(containers))].append(it)
    return
SimulatedAnnealing._cleanup_nonempty(containers)

@staticmethod
def geometric_cooling(T0: float, alpha: float,
Tend: float = 1e-3) -> Callable[[int], float]:
    if not (0 < alpha < 1):
        raise ValueError("alpha harus di (0,1)")
    return lambda k: max(Tend, T0 * (alpha ** k))

@staticmethod
def random_neighbor(containers: List[List[str]],
rng: random.Random) -> List[List[str]]:
    new_state = [c[:] for c in containers if c]
    if not new_state: return []
    all_items = [it for c in new_state for it in c]
    if not all_items: return []
    if rng.random() < 0.5 and len(new_state) > 1
```

```
and len(all_items) >= 2:
    i1, i2 = rng.sample(range(len(new_state)), 2)
    if not new_state[i1] or not new_state[i2]:
        return []
    a = rng.choice(new_state[i1])
    b = rng.choice(new_state[i2])
    if a == b: return []
    SimulatedAnnealing.swap(a, b, new_state)
else:
    x = rng.choice(all_items)
    tgt = rng.randint(0, len(new_state))
    src = next((i for i, c in
enumerate(new_state) if x in c), None)
    if src is None: return []
    if tgt < len(new_state) and src == tgt and
len(new_state[src]) == 1:
        return []
    SimulatedAnnealing.shift(x, tgt, new_state)

return

SimulatedAnnealing._cleanup_nonempty(new_state)

@staticmethod
def estimate_T0(state: List[List[str]], items:
Dict[str,int], max_capacity:int,
                rng: random.Random,
samples:int=128, p0:float=0.8) -> float:
    base = SimulatedAnnealing.objective_fun(state,
max_capacity, items)
    deltas = []
    for _ in range(samples):
        nbh =
SimulatedAnnealing.random_neighbor(state, rng)
        if not nbh: continue
```

```
        dc = SimulatedAnnealing.objective_fun(nbh,
max_capacity, items) - base
        if dc > 0: deltas.append(dc)
        if not deltas: return 1.0
        mean_pos = sum(deltas)/len(deltas)
        p0 = max(1e-6, min(0.999, p0))
        return max(1e-9, -mean_pos / math.log(p0))

    @staticmethod
    def optimize(initial_containers: List[List[str]],
                 items: Dict[str,int],
                 max_capacity:int,
                 schedule: Callable[[int], float],
                 max_iters:int=50_000,
                 max_stagnant:int=5_000,
                 seed=182,
                 stuck_window:int=100) ->
        Dict[str,Any]:
        rng = random.Random(seed)
        current =
            SimulatedAnnealing._cleanup_nonempty(copy.deepcopy(
initial_containers))
        curr_cost =
            SimulatedAnnealing.objective_fun(current,
max_capacity, items)

        best = copy.deepcopy(current)
        best_cost = curr_cost

        hist_best = [best_cost]
        hist_curr = [curr_cost]
        prob_hist = []
        temp_hist = []
        delta_hist = []
```

```
stagnant = 0
no_improve_best = 0
stuck_events = 0

moves_log = []
accepted_worse_idx = []
worse_snapshots = []

final_T = schedule(0)
t0 = time.time()

for k in range(max_iters):
    T = schedule(k); final_T = T
    temp_hist.append(T)
    if T <= 1e-9:
        break

    state_before = copy.deepcopy(current)
    curr_before = curr_cost

    cand =
    SimulatedAnnealing.random_neighbor(current, rng)
    if not cand:
        hist_best.append(best_cost);
    hist_curr.append(curr_cost)
        prob_hist.append(None)
        delta_hist.append(None)
        stagnant += 1
        no_improve_best += 1

        if no_improve_best >= stuck_window:
            stuck_events += 1
            no_improve_best = 0
```

```
        if stagnant >= max_stagnant:
            break
        continue

        c_cost =
SimulatedAnnealing.objective_fun(cand,
max_capacity, items)
        d = c_cost - curr_cost
        delta_hist.append(d)

        if d < 0:
            prob = None
            accept = True
        else:
            exp_arg = -d / T if T > 0 else -1e9
            prob = 0.0 if exp_arg < -700 else
math.exp(exp_arg)
            accept = (rng.random() < prob)

        mv = {
            "iter": k,
            "T": T,
            "delta": d,
            "worse": (d > 0),
            "prob": prob,
            "accepted": accept,
            "curr_obj_before": curr_before,
            "cand_obj": c_cost,
            "state_before": state_before,
        }
        moves_log.append(mv)

        if accept:
            current, curr_cost = cand, c_cost
```

```
        if curr_cost < best_cost:
            best, best_cost = copy.deepcopy(current),
curr_cost
            stagnant = 0
            no_improve_best = 0
        else:
            if d < 0:
                no_improve_best += 1
            elif d == 0:
                no_improve_best += 1
            else:
                accepted_worse_idx.append(k)
                worse_snapshots.append({
                    "iter": k,
                    "state": copy.deepcopy(current),
                    "obj": curr_cost
                })
                stagnant += 1
                no_improve_best += 1
        else:
            stagnant += 1
            no_improve_best += 1

    if no_improve_best >= stuck_window:
        stuck_events += 1
        no_improve_best = 0

    hist_best.append(best_cost)
    hist_curr.append(curr_cost)
    if stagnant >= max_stagnant:
        break

return {
    "best_state": best,
```

```
        "best_cost": best_cost,
        "history_best": hist_best,
        "history_current": hist_curr,
        "probability_history": prob_hist,
        "temperature_history": temp_hist,
        "delta_history": delta_hist,
        "accepted_worse_idx": accepted_worse_idx,
        "worse_snapshots": worse_snapshots,
        "moves_log": moves_log,
        "final_temperature": final_T,
        "time_sec": time.time() - t0,
        "iterations": len(hist_curr)-1,
        "stuck_events": stuck_events
    }

    @staticmethod
    def run(items: Dict[str,int],
max_capacity:int=50, max_iter:int=500,
seed:int=42, alpha:float=0.995,
p_new:float=0.2,
stuck_window:int=100):
    rng = random.Random(seed)
    containers =
SimulatedAnnealing.random_init(items, rng,
p_new=p_new)
    initial_state = copy.deepcopy(containers)

    T0 =
SimulatedAnnealing.estimate_T0(initial_state,
items, max_capacity, rng, samples=128, p0=0.8)
    schedule =
SimulatedAnnealing.geometric_cooling(T0, alpha,
Tend=1e-3)
```

```
start = time.time()
res = SimulatedAnnealing.optimize(
    initial_containers=initial_state,
    items=items,
    max_capacity=max_capacity,
    schedule=schedule,
    max_iters=max_iter,
    max_stagnant=max_iter//5,
    seed=seed,
    stuck_window=stuck_window
)
duration = time.time() - start

return {
    "initial_state": initial_state,
    "final_state": res["best_state"],
    "objective_values": res["history_best"],
    "history_current": res["history_current"],
    "probability_history":
        res["probability_history"],
    "temperature_values":
        res["temperature_history"],
    "delta_history": res["delta_history"],
    "accepted_worse_idx":
        res["accepted_worse_idx"],
    "worse_snapshots": res["worse_snapshots"],
    "moves_log": res["moves_log"],
    "final_temperature":
        res["final_temperature"],
    "stuck_events": res["stuck_events"],
    "iterations": res["iterations"],
    "duration": duration
}
```

```
import matplotlib.pyplot as plt

def plot_sa_worse_markers(history_current:
List[float],
                           accepted_worse_idx:
List[int],
                           title: str = "SA:
Objective vs Iteration (x = accepted worse)"):

    plt.figure(figsize=(8,4))
    plt.plot(history_current, label="current
objective")
    if accepted_worse_idx:
        y = [history_current[i] for i in
accepted_worse_idx if 0 <= i <
len(history_current)]
        x = [i for i in accepted_worse_idx if 0 <= i <
len(history_current)]
        plt.scatter(x, y, marker='x', s=60,
label="worse move accepted")
        plt.xlabel("iteration"); plt.ylabel("objective")
        plt.title(title); plt.legend();
        plt.tight_layout(); plt.show()

def plot_delta_and_accept(moves_log:
List[Dict[str,Any]]),
                           title: str = "SA: Δ per
iteration (x = accepted worse)"):

    delta = [m["delta"] for m in moves_log]
    accepted = [i for i,m in enumerate(moves_log) if
m["worse"] and m["accepted"]]
    plt.figure(figsize=(8,4))
    plt.plot(delta, label="delta (cand - curr)")
    if accepted:
        y = [delta[i] for i in accepted]
```

```
plt.scatter(accepted, y, marker='x', s=60,
label="accepted worse")

plt.axhline(0, linestyle="--")
plt.xlabel("iteration"); plt.ylabel("delta")
plt.title(title); plt.legend();
plt.tight_layout(); plt.show()

def _draw_state_bars(containers: List[List[str]],
                     items: Dict[str,int],
                     ax,
                     title: str = "",
                     max_capacity: Optional[int] =
None):
    totals = [sum(items[it] for it in c) for c in
containers]
    x = range(len(containers))
    bottom = [0]*len(containers)

    unique_items = sorted({it for c in containers for
it in c})
    for it in unique_items:
        heights = []
        for idx, c in enumerate(containers):
            h = items[it] if it in c else 0
            heights.append(h)
        ax.bar(list(x), heights, bottom=bottom)
        bottom = [b+h for b,h in zip(bottom, heights)]
    ax.set_xticks(list(x))
    ax.set_xticklabels([f"C{idx}\n({tot})" for idx,
tot in enumerate(totals)])
    ax.set_ylabel("load")
    if max_capacity is not None:
        ax.axhline(max_capacity, linestyle="--")
    ax.set_title(title)
```

```
def show_worse_transition(res: Dict[str,Any],  
                         items: Dict[str,int],  
                         max_capacity: int,  
                         k_pick: Optional[int] =  
                           None):  
    if not res["accepted_worse_idx"]:  
        print("Tidak ada langkah worse yang diterima.")  
        return  
    k = res["accepted_worse_idx"][-1] if k_pick is  
None else k_pick  
  
    mv = res["moves_log"][k]  
    state_before = mv.get("state_before")  
    state_after = None  
    for snap in res["worse_snapshots"]:  
        if snap["iter"] == k:  
            state_after = snap["state"]  
            obj_after = snap["obj"]  
            break  
  
    if state_before is None or state_after is None:  
        print("State sebelum/sesudah tidak lengkap di  
log. Pastikan logging di optimize() sudah benar.")  
        return  
  
    fig, axs = plt.subplots(1, 2, figsize=(10,4),  
                          sharey=True)  
    _draw_state_bars(state_before, items, axs[0],  
                     title=f"Before (iter  
{k})\nobj={mv['curr_obj_before']:.2f}",  
                     max_capacity=max_capacity)  
    _draw_state_bars(state_after, items, axs[1],  
                     title=f"After WORSE (iter
```

```

{k})\nobj={obj_after:.2f}",
max_capacity=max_capacity)

fig.suptitle("Accepted worse move: before →
after")

plt.tight_layout(); plt.show()

```

Implementasi algoritma ini mengandalkan konsep fisika, khususnya proses pendinginan logam, yang bertujuan untuk menghindari *stuck* pada solusi lokal dan mencari solusi optimal global. Proses diawali dengan algoritma yang dimulai dengan suhu yang tinggi ( $T_0$ ) sehingga memungkinkan penerimaan solusi yang lebih buruk (menghindari terjebak di solusi lokal). Seiring berjalannya iterasi, suhu diturunkan (dengan jadwal pendinginan tertentu). Hal ini akan membuat probabilitas menerima solusi yang lebih buruk juga menurun. Suhu berfungsi sebagai parameter untuk menentukan kemungkinan penerimaan solusi yang lebih buruk.

**Tabel 2.2 Rumus Simulated Annealing**

Simbol	Penjelasan
$e^{\frac{\Delta E}{T}}$	Rumus untuk menentukan kemungkinan penerimaan solusi yang lebih buruk
$\Delta E$	Perubahan energi (atau perubahan biaya)
$T$	Suhu yang digunakan dalam perhitungan.

Terdapat tiga langkah utama secara umum dalam implementasi *simulated annealing*:

1. Dimulai dengan solusi awal, yang dapat diperoleh baik secara acak maupun heuristik. Pada implementasi ini, digunakan inisialisasi acak untuk menghasilkan *state* secara acak. Pada tahap ini, suhu awal yang tinggi ditetapkan (*set* suhu awal yang tinggi).

2. *Neighborhood generation* yang menciptakan solusi tetangga dengan sedikit perubahan pada solusi yang ada, seperti melakukan *swap* atau *shift* dalam konteks bin *packing*.

3. Evaluasi solusi tetangga dilakukan berdasarkan biaya atau fungsi objektif. Solusi dapat berpindah ke tetangga jika probabilitas berdasarkan rumus *simulated annealing* melebihi *threshold* yang ditentukan, sehingga memungkinkan pergerakan ke solusi tersebut.

Berikut merupakan mekanisme dari implementasi algoritma *simulated annealing*.

1. Kelas Simulated Annealing mewarisi fungsi-fungsi penting dari BinPackingBase, sehingga bisa langsung menggunakan fungsi `shift()`, `swap()`, dan `objective_fun()`. Terdapat fungsi `_cleanup_nonempty()` dipakai untuk membersihkan kontainer kosong.

2. Terdapat `random_init()` membuat solusi awal secara acak. Setiap *item* atau barang bisa masuk ke kontainer yang sudah ada atau membuat kontainer baru dengan probabilitas tertentu (`p_new`). Semakin besar `p_new`, semakin banyak kontainer kecil yang terbentuk di awal.

3. Adapun `geometric_cooling()` membuat fungsi suhu yang menurun secara bertahap menggunakan rumus geometrik.

$$Tk = \max(T_{\text{end}}, T_0 \times \alpha^k).$$

Di sini, `T0` adalah suhu awal, `alpha` laju pendinginan, dan `Tend` suhu minimum.

4. Terdapat fungsi `random_neighbor()` yang menghasilkan satu tetangga acak dari *state* saat ini, baik dengan menukar dua *item* atau barang antar kontainer (*swap*) atau memindahkan satu item ke tempat lain (*shift*).

5. Kemudian, terdapat `estimate_T0()` menghitung suhu awal secara otomatis agar peluang menerima solusi buruk di awal sekitar 80%
6. Fungsi `optimize()` adalah proses secara bertahap mencari solusi terbaik dengan menerima solusi buruk di awal (untuk eksplorasi) dan makin selektif saat suhu menurun (untuk eksploitasi) hingga sistem mencapai kondisi stabil.
7. Kemudian, fungsi `run()` menyiapkan solusi awal, menentukan suhu awal dan jadwal pendinginan, lalu menjalankan proses optimasi hingga menghasilkan solusi terbaik beserta waktu eksekusinya.
8. Adapun fungsi tambahan seperti `plot_sa_worse_markers()`, `plot_delta_and_accept()`, `show_worse_transition()` digunakan untuk memvisualisasikan kapan SA menerima solusi buruk, bagaimana perubahan  $\Delta$  terjadi, serta bagaimana pergerakan menuju solusi akhir.

## 2.4. Genetic Algorithm (GA)

*Genetic Algorithm* merupakan algoritma yang terinspirasi dari evolusi biologis dengan menggunakan prinsip seleksi alam, persilangan (*crossover*), dan mutasi untuk menemukan solusi terbaik secara bertahap dari satu generasi ke generasi selanjutnya.

### 2.4.1. Kode dan Penjelasan Implementasi

```
class Genetic(BinPackingBase):  
    @staticmethod  
    def fitness_fun(containers, max_capacity, items):  
        return 1 / (1 +  
BinPackingBase.objective_fun(containers,
```

```
max_capacity, items))

    @staticmethod
    def reverse_fitness(fitness_value):
        return (1 / fitness_value) - 1

    @staticmethod
    def populate(population, size, items):
        for _ in range(size):
            containers = []
            BinPackingBase.initialize(containers, items)
            population.append(containers)

    @staticmethod
    def select(population, max_capacity, items):
        population_fitness =
[Genetic.fitness_fun(individual, max_capacity,
items) for individual in population]
        return random.choices(population,
weights=population_fitness, k=len(population))

    @staticmethod
    def clean(individual_to_clean,
individual_to_compare):
        compare_set = {item for container in
individual_to_compare for item in container}
        cleaned_individual = []

        for container in individual_to_clean:
            cleaned_container = [item for item in
container if item not in compare_set]
            if cleaned_container:
                cleaned_individual.append(cleaned_container)
```

```
        return cleaned_individual

    @staticmethod
    def crossover(parent1, parent2):
        max_point = min(len(parent1), len(parent2)) - 1
        if max_point < 1:
            return parent1[:], parent2[:]

        crossover_point = rand.randint(0, max_point)

        offspring1_half1 = parent1[:crossover_point+1]
        offspring1_half2 = Genetic.clean(parent2,
offspring1_half1)
        offspring1 = offspring1_half1 +
offspring1_half2

        offspring2_half1 = parent2[:crossover_point+1]
        offspring2_half2 = Genetic.clean(parent1,
offspring2_half1)
        offspring2 = offspring2_half1 +
offspring2_half2

        return offspring1, offspring2

    @staticmethod
    def mutate(containers, items, mutation_rate=0.1):
        if rand.random() > mutation_rate:
            return containers

        successor = copy.deepcopy(containers)

        non_empty = [c for c in successor if c]
        if not non_empty:
```

```
        return successor

    random_container = rand.choice(non_empty)
    random_item = rand.choice(random_container)
    random_move = rand.randint(0, 1)

    if random_move == 0:
        other_containers = [c for c in successor if c
is not random_container]
        if not other_containers:
            return successor
        random_container2 =
rand.choice(other_containers)
        random_item2 = rand.choice(random_container2)
        BinPackingBase.swap(random_item,
random_item2, successor)
    else:
        BinPackingBase.shift(random_item,
rand.randint(0, len(successor) - 1), successor)

    successor = [c for c in successor if c]
    return successor

@staticmethod
def run(items, max_capacity, population_size,
max_iteration, mutation_rate):
    population = []
    Genetic.populate(population, population_size,
items)
    initial_population = population

    best_initial_individual =
min(initial_population, key=lambda x:
BinPackingBase.objective_fun(x, max_capacity,
```

```
items))

    min_initial_objective_value =
BinPackingBase.objective_fun(best_initial_individual, max_capacity, items)

    min_objective_values =
[min_initial_objective_value]
    avg_objective_values = []

    initial_objectives =
[BinPackingBase.objective_fun(ind, max_capacity,
items) for ind in population]

avg_objective_values.append(np.mean(initial_objectives))

    start_time = time.time()
    for i in range(max_iteration):
        all_objectives =
[BinPackingBase.objective_fun(ind, max_capacity,
items) for ind in population]

avg_objective_values.append(np.mean(all_objectives))

    population_fitness =
[Genetic.fitness_fun(individual, max_capacity,
items) for individual in population]

min_objective_values.append(Genetic.reverse_fitness(
max(population_fitness)))

    population = random.choices(population,
weights=population_fitness, k=len(population))
```

```
new_population = []

for j in range(0, len(population), 2):
    child1, child2 =
        Genetic.crossover(population[j], population[j+1])
    new_population.append(child1)
    new_population.append(child2)
population = new_population
for k in range(0, len(population)):
    population[k] =
        Genetic.mutate(population[k], items, mutation_rate)

final_time = time.time()
duration = final_time - start_time

best_final_individual = min(population,
key=lambda x: BinPackingBase.objective_fun(x,
max_capacity, items))

return min_objective_values,
avg_objective_values, best_initial_individual,
best_final_individual, duration
```

Berikut merupakan mekanisme dari implementasi algoritma *genetic algorithm*.

1. Kelas Genetic mewarisi fungsi dari BinPackingBase, sehingga dapat langsung memakai metode seperti objective\_fun() untuk menilai kualitas solusi dan mengatur distribusi item antar kontainer.
2. Terdapat fungsi fitness\_fun() mengubah nilai objektif (semakin kecil semakin baik) menjadi nilai fitness (semakin besar semakin baik) dengan rumus:

$$fitness = \frac{1}{1+f_{obj}}$$

Hal ini membantu algoritma memilih solusi yang lebih optimal dengan probabilitas lebih tinggi. Semakin kecil nilai objektif (artinya solusi makin bagus), semakin besar skor fitness-nya. Ini membuat solusi terbaik punya peluang lebih besar untuk terpilih dalam proses seleksi.

3. Pertama, `reverse_fitness()` berfungsi untuk membalik hasil perhitungan sebelumnya, yaitu mengubah nilai fitness kembali menjadi nilai objektif agar hasil algoritma bisa lebih mudah dianalisis atau divisualisasikan.
4. Kemudian, `populate()` digunakan untuk membentuk populasi awal yang berisi beberapa solusi acak dan tiap solusi menggambarkan cara penempatan *item* atau barang ke dalam kontainer yang dibuat dengan bantuan `BinPackingBase.initialize()`.
5. Selanjutnya, `select()` menjalankan proses *Roulette Wheel Selection*, yaitu memilih individu berdasarkan tingkat *fitness*-nya maka semakin tinggi *fitness*-nya, semakin besar pula peluang individu tersebut terpilih menjadi *parent* di generasi berikutnya.
6. Fungsi `clean()` bertugas membersihkan hasil *crossover* supaya tidak ada item yang muncul dua kali, dengan cara membandingkan dua *parent* dan menghapus item yang duplikat di *offspring* agar setiap item hanya muncul satu kali dalam solusi.
7. Pertama, fungsi `crossover()` menjadi tahap reproduksi yang menggabungkan dua *parent* untuk membentuk dua *offspring* baru. Algoritma menentukan titik potong acak (*crossover point*), lalu bagian awal dari *parent* pertama digabung dengan bagian akhir dari *parent* kedua berlaku untuk sebaliknya, sehingga terbentuk dua anak dengan kombinasi genetik dari kedua *parent*.

8. Kemudian, fungsi `mutate()` digunakan untuk menambahkan variasi acak ke dalam populasi agar algoritma tidak cepat terjebak pada solusi lokal. Proses ini dilakukan dengan probabilitas tertentu sesuai nilai `mutation_rate`.
9. Selanjutnya, ada dua jenis mutasi yang digunakan, yaitu *swap*, yang menukar dua *item* atau barang antar kontainer, dan *shift*, yang memindahkan satu *item* atau barang ke kontainer lain.
10. Setelah itu, sistem akan membersihkan kontainer kosong yang muncul setelah mutasi.
11. Terakhir, fungsi `run()` menjadi inti dari seluruh proses evolusi dalam *Genetic Algorithm*. Fungsi ini membentuk populasi awal dengan `populate()`, menghitung nilai objektif tiap individu, lalu menjalankan iterasi hingga `max_iteration` melalui proses seleksi, *crossover*, dan mutasi. Nilai objektif terbaik tiap generasi disimpan untuk memantau perkembangan, hingga akhirnya ditemukan individu terbaik sebagai solusi optimal.

### 3. Implementasi Visualisasi dan Eksperimen

#### 3.1. Eksperimen

```
EXPERIMENT_ITEMS = {  
    "BRG-01": 8, "BRG-02": 2, "BRG-03": 5, "BRG-04":  
4, "BRG-05": 7,  
    "BRG-06": 1, "BRG-07": 3, "BRG-08": 6, "BRG-09":  
9, "BRG-10": 2,  
    "BRG-11": 8, "BRG-12": 4, "BRG-13": 5, "BRG-14":  
1, "BRG-15": 7,  
    "BRG-16": 3, "BRG-17": 6, "BRG-18": 9, "BRG-19":  
2, "BRG-20": 5,  
    "BRG-21": 4, "BRG-22": 6, "BRG-23": 3, "BRG-24":  
7, "BRG-25": 1,
```

```
"BRG-26": 8
}

EXPERIMENT_CAPACITY = 20

NUM_RUNS = 3
SEEDS = [67, 182, 69]

print("==" * 60)
print("SETUP EKSPERIMEN TUGAS BESAR")
print("==" * 60)
print(f"Dataset: {len(EXPERIMENT_ITEMS)} items")
print(f"Kapasitas: {EXPERIMENT_CAPACITY}")
print(f"Jumlah run per algoritma: {NUM_RUNS}")
print(f"Seeds: {SEEDS}")
print("==" * 60)
```

### 3.2. Eksperimen 1: Hill Climbing

#### 3.2.1. Stochastic Hill-Climbing

```
print("\n" + "==" * 60)
print("EKSPERIMEN 1: STOCHASTIC HILL CLIMBING")
print("==" * 60)

hc_results = []
MAX_ITER_HC = 5000

for run_idx, seed in enumerate(SEEDS, 1):
    print(f"\n--- Run {run_idx} (Seed: {seed})
---")

    init_state, final_state, history, duration =
```

```
StochasticHillClimbing.run(  
    EXPERIMENT_ITEMS,  
    max_capacity=EXPERIMENT_CAPACITY,  
    max_iter=MAX_ITER_HC,  
    seed=seed  
)  
  
result = {  
    'run': run_idx,  
    'seed': seed,  
    'initial_state': init_state,  
    'final_state': final_state,  
    'history': history,  
    'final_objective': history[-1],  
    'duration': duration,  
    'iterations': len(history) - 1  
}  
hc_results.append(result)  
  
print(f"Initial Objective:  
{history[0]:.2f}")  
print(f"Final Objective: {history[-1]:.2f}")  
print(f"Improvement: {((history[0] -  
history[-1]) / history[0] * 100):.2f}%")  
print(f"Iterasi: {result['iterations']}")  
print(f"Durasi: {duration:.3f} detik")  
  
print("\n" + "=" * 60)  
print("RINGKASAN STOCHASTIC HILL CLIMBING")  
print("=" * 60)  
final_objectives = [r['final_objective'] for r
```

```
in hc_results]
durations = [r['duration'] for r in hc_results]
print(f"Objective Function Akhir:")
print(f"  Min: {min(final_objectives):.2f}")
print(f"  Max: {max(final_objectives):.2f}")
print(f"  Rata-rata:
{np.mean(final_objectives):.2f}")
print(f"  Std Dev:
{np.std(final_objectives):.2f}")
print(f"\nDurasi:")
print(f"  Rata-rata: {np.mean(durations):.3f}
detik")
print("==" * 60)

print("\n VISUALISASI STOCHASTIC HILL CLIMBING")
for result in hc_results:
    run_id = result['run']
    final_obj = result['final_objective']

    print(f"\n🏁 RUN {run_id} - Seed:
{result['seed']} | Final Objective:
{final_obj:.2f}")

    Visualizer.visualize_state(
        result['initial_state'],
        EXPERIMENT_ITEMS,
        EXPERIMENT_CAPACITY,
        f"Stochastic HC - Initial State Run
{run_id}",
    )
```

```
Visualizer.visualize_state(
    result['final_state'],
    EXPERIMENT_ITEMS,
    EXPERIMENT_CAPACITY,
    f"Stochastic HC - Final State Run {run_id}"
)

Visualizer.plot_objective_values(
    result['history'],
    result['duration'],
    title=f"Stochastic HC - Nilai Objektif per
Iterasi Run {run_id} (Seed {result['seed']})"
)
```

### 3.2.2. *Steepest Ascent Hill-Climbing*

```
print("\n" + "=" * 60)
print("EKSPERIMEN 1.2: STEEPEST ASCENT HILL
CLIMBING")
print("=" * 60)

hc_results = []
MAX_ITER_HC = 5000

for run_idx, seed in enumerate(SEEDS, 1):
    print(f"\n--- Run {run_idx} (Seed: {seed})
---")

    initial_state, final_state, best_val,
iterations_run, history, duration =
SteepestHill.run(
```

```
        EXPERIMENT_ITEMS,
        max_capacity=EXPERIMENT_CAPACITY,
        max_iter=MAX_ITER_HC,
        seed=seed
    )

result = {
    'run': run_idx,
    'seed': seed,
    'initial_state': initial_state,
    'final_state': final_state,
    'history': history,
    'final_objective': best_val,
    'duration': duration,
    'iterations': iterations_run
}
hc_results.append(result)

print(f"Initial Objective:
{history[0]:.2f}")

print(f"Final Objective:
{result['final_objective']:.2f}")

improvement = ((history[0] -
result['final_objective']) / history[0] * 100)
if history[0] != 0 else 0
print(f"Improvement: {improvement:.2f}%")
print(f"Iterasi: {result['iterations']} ")
print(f"Durasi: {duration:.3f} detik")

print("\n" + "=" * 60)
```

```
print("RINGKASAN STEEPEST ASCENT HILL CLIMBING")
print("=" * 60)
final_objectives = [r['final_objective'] for r
in hc_results]
durations = [r['duration'] for r in hc_results]
print(f"Objective Function Akhir:")
print(f"  Min: {min(final_objectives):.2f}")
print(f"  Max: {max(final_objectives):.2f}")
print(f"  Rata-rata:
{np.mean(final_objectives):.2f}")
print(f"  Std Dev:
{np.std(final_objectives):.2f}")
print(f"\nDurasi:")
print(f"  Rata-rata: {np.mean(durations):.3f}
detik")
print("=" * 60)

print("\n VISUALISASI STEEPEST ASCENT HILL
CLIMBING")

for result in hc_results:
    run_id = result['run']
    final_obj = result['final_objective']

    print(f"\n🏁 RUN {run_id} – Seed:
{result['seed']} | Final Objective:
{final_obj:.2f}")

    Visualizer.visualize_state(
        result['initial_state'],
        EXPERIMENT_ITEMS,
```

```
        EXPERIMENT_CAPACITY,  
        f"STEEPEST ASCENT HC - Initial State Run  
{run_id}",  
    )  
  
    Visualizer.visualize_state(  
        result['final_state'],  
        EXPERIMENT_ITEMS,  
        EXPERIMENT_CAPACITY,  
        f"STEEPEST ASCENT HC - Final State Run  
{run_id}"  
    )  
  
    Visualizer.plot_objective_values(  
        result['history'],  
        result['duration'],  
        title=f"STEEPEST ASCENT HC - Nilai Objektif  
per Iterasi Run {run_id} (Seed  
{result['seed']})"  
    )
```

### 3.2.3. *Hill-Climbing with Sideways Move*

```
print("\n" + "=" * 60)  
print("EKSPERIMEN 1.3: HILL CLIMBING WITH  
SIDEWAYS MOVE")  
print("=" * 60)  
  
hc_results = []  
MAX_ITER_HC = 5000
```

```
for run_idx, seed in enumerate(SEEDS, 1):
    print(f"\n--- Run {run_idx} (Seed: {seed})")
    ---")

    initial_state, final_state, best_val,
iterations_run, history, duration =
SteepestHill.run(
    EXPERIMENT_ITEMS,
    max_capacity=EXPERIMENT_CAPACITY,
    max_iter=MAX_ITER_HC,
    seed=seed
)

result = {
    'run': run_idx,
    'seed': seed,
    'initial_state': initial_state,
    'final_state': final_state,
    'history': history,
    'final_objective': best_val,
    'duration': duration,
    'iterations': iterations_run
}
hc_results.append(result)

print(f"Initial Objective:
{history[0]:.2f}")
print(f"Final Objective:
{result['final_objective']:.2f}")
improvement = ((history[0] -
result['final_objective'])) / history[0] * 100)
```

```
if history[0] != 0 else 0
    print(f"Improvement: {improvement:.2f} %")
    print(f"Iterasi: {result['iterations']} ")
    print(f"Durasi: {duration:.3f} detik")

print("\n" + "=" * 60)
print("RINGKASAN STEEPEST ASCENT HILL CLIMBING")
print("=" * 60)
final_objectives = [r['final_objective'] for r
in hc_results]
durations = [r['duration'] for r in hc_results]
print(f"Objective Function Akhir:")
print(f"  Min: {min(final_objectives):.2f}")
print(f"  Max: {max(final_objectives):.2f}")
print(f"  Rata-rata:
{np.mean(final_objectives):.2f}")
print(f"  Std Dev:
{np.std(final_objectives):.2f}")
print(f"\nDurasi:")
print(f"  Rata-rata: {np.mean(durations):.3f}
detik")
print("=" * 60)

print("\n HILL CLIMBING WITH SIDEWAYS MOVE")

for result in hc_results:
    run_id = result['run']
    final_obj = result['final_objective']

    print(f"\n🏁 RUN {run_id} – Seed:
```

```
{result['seed']} | Final Objective:  
{final_obj:.2f})  
  
Visualizer.visualize_state(  
    result['initial_state'],  
    EXPERIMENT_ITEMS,  
    EXPERIMENT_CAPACITY,  
    f"HC WITH SIDEWAYS MOVE - Initial State Run  
{run_id}",  
)  
  
Visualizer.visualize_state(  
    result['final_state'],  
    EXPERIMENT_ITEMS,  
    EXPERIMENT_CAPACITY,  
    f"HC WITH SIDEWAYS MOVE - Final State Run  
{run_id}"  
)  
  
Visualizer.plot_objective_values(  
    result['history'],  
    result['duration'],  
    title=f"HC WITH SIDEWAYS MOVE - Nilai  
Objektif per Iterasi Run {run_id} (Seed  
{result['seed']})"  
)
```

### 3.2.4. Random Restart Hill-Climbing

```
print("\n" + "=" * 60)  
print("EKSPERIMEN 1.4: RANDOM RESTART HILL")
```

```
CLIMBING")
print("=" * 60)

rr_results = []
MAX_RESTARTS = 20
ITERS_PER_RESTART = 500

for run_idx, seed in enumerate(SEEDS, 1):
    print(f"\n--- Run {run_idx} (Seed: {seed})\n---")

    np.random.seed(seed)
    random.seed(seed)

    initial_state, final_state, best_val,
    restarts_done, total_iters, restart_objectives,
    best_history, duration, all_histories,
    per_restart_iters = RandomRestartHill.run(
        EXPERIMENT_ITEMS,
        max_capacity=EXPERIMENT_CAPACITY,
        max_restarts=MAX_RESTARTS,
        iters_per_restart=ITERS_PER_RESTART,
        allow_new_bin=True
    )

    initial_obj_from_state =
BinPackingBase.objective_fun(initial_state,
EXPERIMENT_CAPACITY, EXPERIMENT_ITEMS)

    result = {
        'run': run_idx,
        'seed': seed,
```

```
        'initial_state': initial_state,
        'final_state': final_state,
        'restart_objectives':
restart_objectives,
        'final_objective': best_val,
        'best_val': best_val,
        'restarts_done': restarts_done,
        'total_iters': total_iters,
        'duration': duration,
        'all_histories': all_histories,
        'per_restart_iters': per_restart_iters,
        'best_history': best_history,
        'initial_obj_from_state':
initial_obj_from_state

    }

rr_results.append(result)

    print(f"Initial Objective:
{initial_obj_from_state:.2f}")

    print(f"Final Objective:
{result['final_objective']:.2f}")

    print(f"Total Iterasi: {total_iters}")
    print(f"Durasi: {duration:.3f} detik")
    print(f"Jumlah Restart: {restarts_done}")
    print(f"Iterasi per Restart:
{per_restart_iters}")

    # print(f"Objective Terbaik:
{best_val:.2f}")

    # print(f"Objective Tiap Restart: {[ '%.2f' %
```

```
v for v in restart_objectives] }")  
  
print("\n" + "=" * 60)  
print("RINGKASAN RANDOM RESTART HILL CLIMBING")  
print("=" * 60)  
  
final_objectives = [r['best_val'] for r in  
rr_results]  
durations = [r['duration'] for r in rr_results]  
total_iters_list = [r['total_iters'] for r in  
rr_results]  
  
print(f"Objective Function Akhir:")  
print(f"  Min: {min(final_objectives):.2f}")  
print(f"  Max: {max(final_objectives):.2f}")  
print(f"  Rata-rata:  
  {np.mean(final_objectives):.2f}")  
print(f"  Std Dev:  
  {np.std(final_objectives):.2f}")  
print(f"\nIterasi Total:")  
print(f"  Rata-rata:  
  {np.mean(total_iters_list):.1f}")  
print(f"  Std Dev:  
  {np.std(total_iters_list):.1f}")  
print(f"\nDurasi:")  
print(f"  Rata-rata: {np.mean(durations):.3f}  
detik")  
print(f"  Std Dev: {np.std(durations):.3f}  
detik")  
print("=" * 60)
```

```
print("\n RANDOM RESTART HILL CLIMBING")

for result in rr_results:
    run_id = result['run']
    seed = result['seed']
    final_obj = result['best_val']

    print(f"\n🏁 RUN {run_id} - Seed: {seed} |"
          f"Final Objective: {final_obj:.2f}")

    Visualizer.visualize_state(
        result['initial_state'],
        EXPERIMENT_ITEMS,
        EXPERIMENT_CAPACITY,
        f"RANDOM RESTART HC - Initial State Run"
        f"{run_id}"
    )

    Visualizer.visualize_state(
        result['final_state'],
        EXPERIMENT_ITEMS,
        EXPERIMENT_CAPACITY,
        f"RANDOM RESTART HC - Final State Run"
        f"{run_id}"
    )

RandomRestartHill.plot_objective_all_histories(
    result['all_histories'],
    duration=result['duration'],
```

```
        title=f"RANDOM RESTART HC - Gabungan Nilai  
Objektif per Iterasi Run {run_id} (Seed {seed})"  
    )
```

### 3.3. Eksperimen 2: Simulated Annealing

```
import matplotlib.pyplot as plt  
import numpy as np  
  
print("\n" + "=" * 60)  
print("EKSPERIMEN 2: SIMULATED ANNEALING")  
print("=" * 60)  
  
sa_results = []  
MAX_ITER_SA = 5000  
  
for run_idx, seed in enumerate(SEEDS, 1):  
    print(f"\n--- Run {run_idx} (Seed: {seed}) ---")  
  
    result_dict = SimulatedAnnealing.run(  
        items=EXPERIMENT_ITEMS,  
        max_capacity=EXPERIMENT_CAPACITY,  
        max_iter=MAX_ITER_SA,  
        seed=seed,  
        alpha=0.995  
    )  
  
    episodes, total_stagnant, frac =  
    Visualizer.stuck_metrics(  
        result_dict["objective_values"], window=150  
    )
```

```
prob_hist          =
result_dict.get("probability_values",
result_dict.get("probability_history", []))
temp_hist          =
result_dict.get("temperature_values",
result_dict.get("temperature_history", []))
history_current   = result_dict["history_current"]
moves_log          = result_dict.get("moves_log",
[])
accepted_worse_idx =
result_dict.get("accepted_worse_idx", [])
delta_hist          =
result_dict.get("delta_history", [m.get("delta") for
m in moves_log] if moves_log else [])
result = {
    'run': run_idx,
    'seed': seed,
    'initial_state':
result_dict["initial_state"],
    'final_state': result_dict["final_state"],
    'history': result_dict["objective_values"],
    'history_current': history_current,
    'probability_history': prob_hist,
    'temperature_history': temp_hist,
    'delta_history': delta_hist,
    'final_objective':
result_dict["objective_values"][-1],
    'duration': result_dict["duration"],
    'iterations':
```

```
len(result_dict["objective_values"]) - 1,
        'stuck_episodes': episodes,
        'stuck_total': total_stagnant,
        'stuck_fraction': frac,
        'accepted_worse': len(accepted_worse_idx),
        'accepted_worse_idx': accepted_worse_idx
    }
sa_results.append(result)

    print(f"Initial Objective:
{result['history'][0]:.2f}")
    print(f"Final Objective:
{result['final_objective']:.2f}")
    print(f"Improvement: {((result['history'][0] -
result['final_objective']) / result['history'][0] * 100):.2f}%)")
    print(f"Iterasi: {result['iterations']} ")
    print(f"Durasi: {result['duration']:.3f} detik")
    print(f"Stuck Episodes: {episodes}")
    print(f"Stuck Fraction: {frac*100:.2f} %")
    print(f"Accepted Worse Moves:
{result['accepted_worse']} ")

    Visualizer.visualize_state(
        result['initial_state'],
        EXPERIMENT_ITEMS,
        EXPERIMENT_CAPACITY,
        f"SA - Initial State Run {run_idx}"
    )

    Visualizer.visualize_state(
```

```
        result['final_state'],
        EXPERIMENT_ITEMS,
        EXPERIMENT_CAPACITY,
        f"SA - Final State Run {run_idx}"
    )

    Visualizer.plot_objective_values(
        result['history_current'],
        result['duration'],
        title=f"SA - Nilai Objektif per Iterasi Run
{run_idx} (Seed {result['seed']})"
    )

    acceptance_probs = []
    iterations_with_worse = []

    for i, (delta, temp) in enumerate(zip(delta_hist,
temp_hist)):
        if delta is not None and delta > 0 and temp >
0:
            prob = np.exp(-delta / temp)
            acceptance_probs.append(prob)
            iterations_with_worse.append(i)

    if acceptance_probs:
        plt.figure(figsize=(12, 6))
        plt.plot(iterations_with_worse,
acceptance_probs, 'o-', linewidth=2, markersize=4,
alpha=0.7)

    if accepted_worse_idx:
```

```
accepted_probs = []
accepted_iters = []
for idx in accepted_worse_idx:
    if idx in iterations_with_worse:
        pos =
iterations_with_worse.index(idx)

accepted_probs.append(acceptance_probs[pos])
accepted_iters.append(idx)

if accepted_probs:
    plt.scatter(accepted_iters,
accepted_probs, color='red', s=100,
                marker='x', linewidth=3,
label='Accepted Worse Move', zorder=5)

    plt.xlabel("Iterasi", fontsize=12,
fontweight='bold')
    plt.ylabel(r"Probabilitas Akseptansi
$e^{-\Delta/T}$", fontsize=12, fontweight='bold')
    plt.title(f"Run {run_idx} (seed={seed}) - SA:
Probabilitas Akseptansi vs Iterasi",
            fontsize=14, fontweight='bold')
    plt.ylim(-0.05, 1.05)
    plt.grid(True, alpha=0.3)
    plt.legend(fontsize=10)
    plt.tight_layout()
    plt.show()

Visualizer.plot_eet(
    probability_history=prob_hist,
```

```
temperature_history=temp_hist,
history_current=history_current,
title=f"Run {run_idx} (seed={seed}) - SA:
\$e^{{-\Delta/T}}\$ vs Iterasi"
)

plt.figure(figsize=(8,4))
plt.plot(history_current, label="current
objective")
if accepted_worse_idx:
    xw = [i for i in accepted_worse_idx if 0 <= i
< len(history_current)]
    yw = [history_current[i] for i in xw]
    plt.scatter(xw, yw, marker='x', s=60,
label="worse move accepted", color='red')
    plt.xlabel("iteration"); plt.ylabel("objective")
    plt.title(f"Run {run_idx} (seed={seed}) - SA
objective (\times worse accepted)")
    plt.legend(); plt.tight_layout(); plt.show()

if delta_hist:
    plt.figure(figsize=(8,4))
    plt.plot(delta_hist, label="delta (cand -
curr)")
    acc_idx = ([i for i,m in enumerate(moves_log)
if m.get("worse") and m.get("accepted")]
            if moves_log else [i for i in
accepted_worse_idx if 0 <= i < len(delta_hist)])
    if acc_idx:
        y = [delta_hist[i] for i in acc_idx]
        plt.scatter(acc_idx, y, marker='x', s=60,
```

```
label="accepted worse", color='red')

    plt.axhline(0, linestyle="--")
    plt.xlabel("iteration"); plt.ylabel("delta")
    plt.title(f"Run {run_idx} - SA Δ per
iteration (x accepted worse)")

    plt.legend(); plt.tight_layout(); plt.show()

print("\n" + "=" * 60)
print("RINGKASAN SIMULATED ANNEALING")
print("=" * 60)

final_objectives = [r['final_objective'] for r in
sa_results]
durations = [r['duration'] for r in sa_results]
stuck_eps = [r['stuck_episodes'] for r in sa_results]
accepted_worse_all = [r['accepted_worse'] for r in
sa_results]

print("Objective Function Akhir:")
print(f" Min: {min(final_objectives):.2f}")
print(f" Max: {max(final_objectives):.2f}")
print(f" Rata-rata:
{np.mean(final_objectives):.2f}")
print(f" Std Dev: {np.std(final_objectives):.2f}")

print("\nDurasi:")
print(f" Rata-rata: {np.mean(durations):.3f} detik")

print("\nStuck Episodes:")
print(f" Rata-rata: {np.mean(stuck_eps):.1f}")

print("\nAccepted Worse Moves:")
```

```
print(f" Rata-rata:  
{np.mean(accepted_worse_all):.1f}")  
print(f" Min-Max :  
{int(np.min(accepted_worse_all))}-{int(np.max(accepted_worse_all))}")  
print("==" * 60)
```

### 3.4. Eksperimen 3: Genetic Algorithm

```
print("\nEKSPERIMENT GENETIC ALGORITHM")  
population_base = 8  
max_iteration_base = 200  
mutation = 0.15  
population_variations = [2, 8, 16]  
max_iteration_variations = [100, 200, 500]  
mutation_rate_variations = [0.15, 0.5]  
  
print("\n==== Eksperimen 2: Variasi Jumlah Generasi  
====")  
for mutation in mutation_rate_variations:  
    print(f"\nMenjalankan GA dengan Mutation Rate =  
{mutation}")  
    for pop_size in population_variations:  
        for i in range(3):  
            print(f"\nMenjalankan GA dengan Populasi =  
{pop_size}")  
  
            min_objective_values, best_initial_individual,  
            best_final_individual, duration = Genetic.run(  
                items=EXPERIMENT_ITEMS,  
                max_capacity=EXPERIMENT_CAPACITY,
```

```
        population_size=pop_size,
        max_iteration=max_iteration_base,
        mutation_rate=mutation,
    )

    Visualizer.visualize_state(
        best_initial_individual,
        EXPERIMENT_ITEMS,
        EXPERIMENT_CAPACITY,
        f"GENETIC - Individu Terbaik Generasi Pertama
(Jumlah Populasi: {pop_size}, Jumlah Iterasi:
{max_iteration_base}, Probabilitas Mutasi:
{mutation})"
    )

    Visualizer.visualize_state(
        best_final_individual,
        EXPERIMENT_ITEMS,
        EXPERIMENT_CAPACITY,
        f"GENETIC - Individu Terbaik Generasi
Terakhir (Jumlah Populasi: {pop_size}, Jumlah
Iterasi: {max_iteration_base}, Probabilitas Mutasi:
{mutation})"
    )

    Visualizer.plot_objective_values(
        min_objective_values,
        duration,
        title=f"GENETIC - Nilai Objektif Minimum per
Generasi (Jumlah Populasi: {pop_size}, Jumlah
Iterasi: {max_iteration_base}, Probabilitas Mutasi:
```

```
{mutation} )"  
)  
  
for gens in max_iteration_variations:  
    for i in range(3):  
        print(f"\nMenjalankan GA dengan Generasi =  
{gens}")  
        min_objective_values, best_initial_individual,  
        best_final_individual, duration = Genetic.run(  
            items=EXPERIMENT_ITEMS,  
            max_capacity=EXPERIMENT_CAPACITY,  
            population_size=8,  
            max_iteration=gens,  
            mutation_rate=mutation  
        )  
  
        Visualizer.visualize_state(  
            best_initial_individual,  
            EXPERIMENT_ITEMS,  
            EXPERIMENT_CAPACITY,  
            f"GENETIC - Individu Terbaik Generasi Pertama  
(Jumlah Populasi: {population_base}, Jumlah Iterasi:  
{gens}, Probabilitas Mutasi: {mutation})"  
        )  
  
        Visualizer.visualize_state(  
            best_final_individual,  
            EXPERIMENT_ITEMS,  
            EXPERIMENT_CAPACITY,  
            f"GENETIC - Individu Terbaik Generasi  
Terakhir (Jumlah Populasi: {population_base}, Jumlah
```

```
Iterasi: {gens}, Probabilitas Mutasi: {mutation})"  
)  
  
Visualizer.plot_objective_values(  
    min_objective_values,  
    duration,  
    title=f"GENETIC - Nilai Objektif Minimum per  
Generasi (Jumlah Populasi: {population_base}, Jumlah  
Iterasi: {gens}, Probabilitas Mutasi: {mutation})"  
)
```

## 4. Hasil Eksperimen dan Analisis

### 4.1. Stochastic Hill-Climbing

Secara umum, semua eksperimen dimulai dengan nilai objektif yang tinggi (yang menandakan pengemasan yang buruk) dan secara progresif mencapai nilai yang jauh lebih rendah. Hal ini menunjukkan adanya peningkatan secara signifikan dalam efisiensi pengemasan. Pada semua eksperimen, state akhir menggunakan hanya 7 kontainer dengan rata-rata pemanfaatan yang cukup tinggi. Hal ini menunjukkan bahwa algoritma berhasil mengemas item ke dalam kontainer dengan efisien. Kemudian, terdapat rata-rata *improvement* 61,09% dengan peningkatan tertinggi (99.65%) pada Eksperimen 2 dan yang terendah (56.3%) pada Eksperimen 1. Didapatkan bahwa perbedaan ini dapat disebabkan oleh konfigurasi awal dan jumlah iterasi yang dilakukan pada setiap eksperimen.

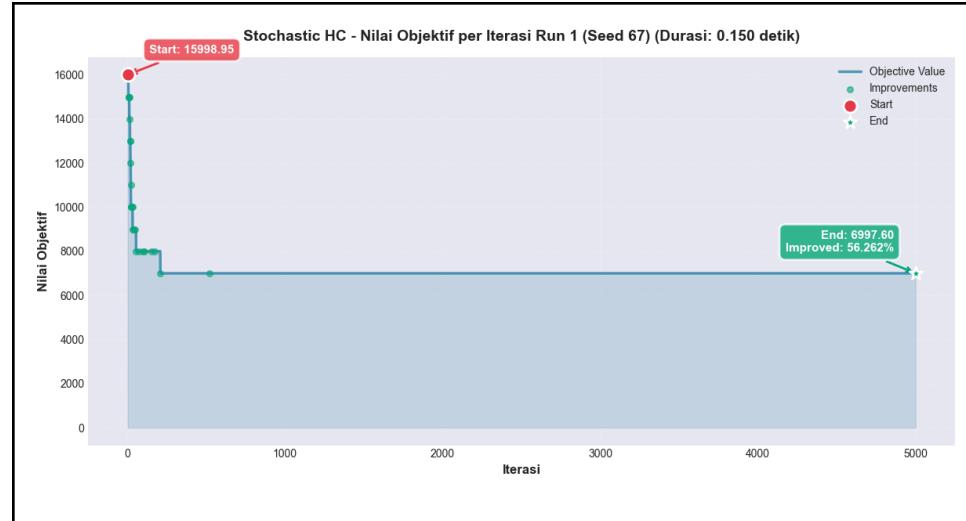
**Kesimpulan:** Algoritma *Stochastic Hill-Climbing* terbukti efektif dalam menyelesaikan *Bin Packing Problem* dengan mengurangi jumlah kontainer yang digunakan dan meningkatkan pemanfaatan ruang.

#### 1. Eksperimen 1 - Stochastic Hill-Climbing

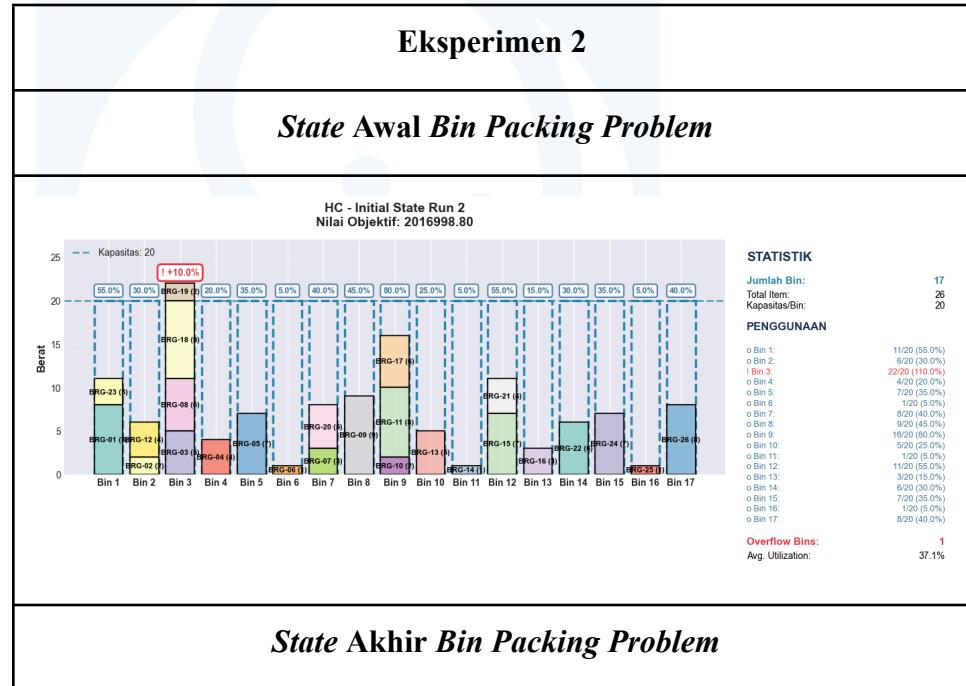
##### Eksperimen 1

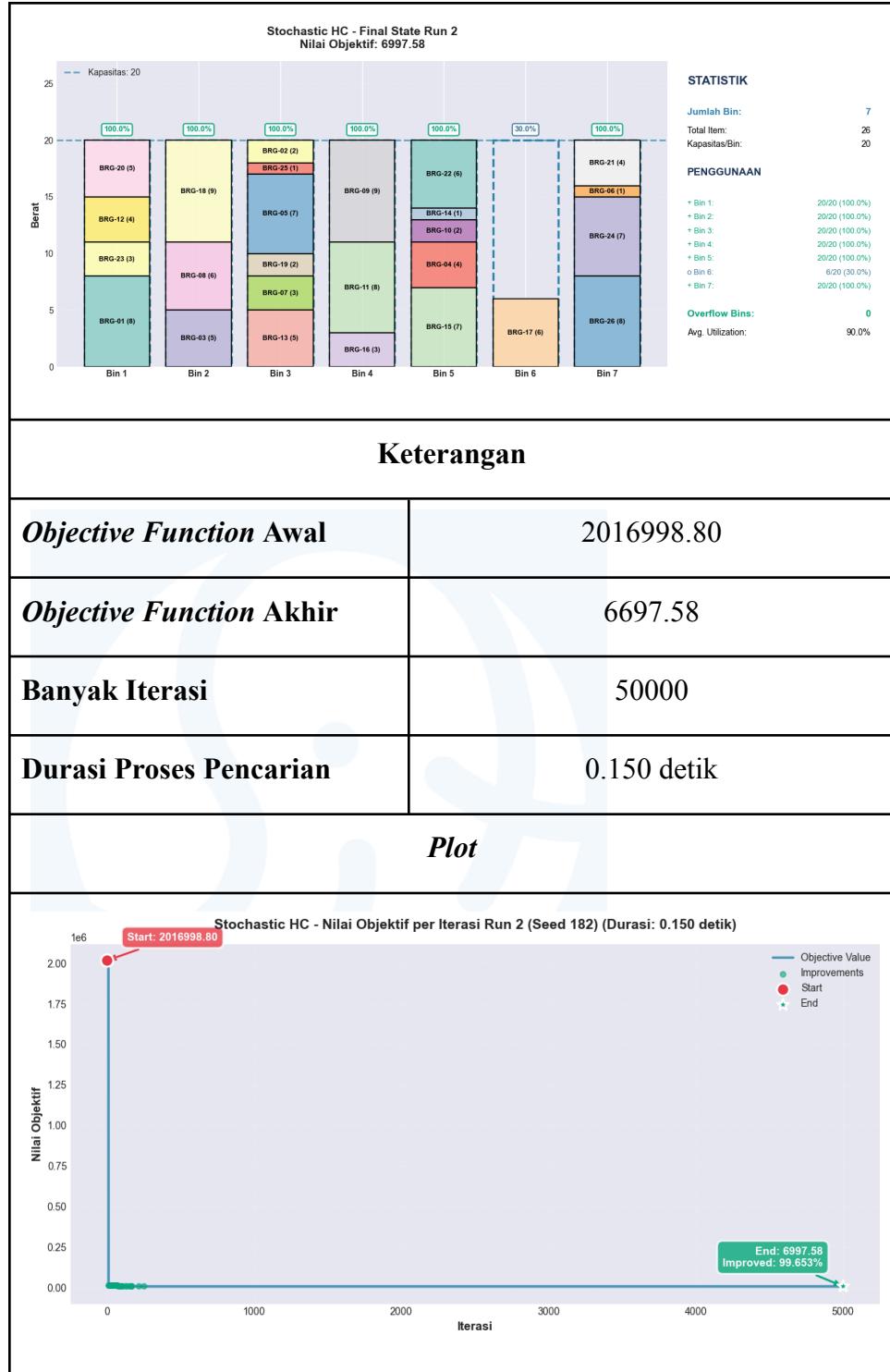
##### State Awal Bin Packing Problem





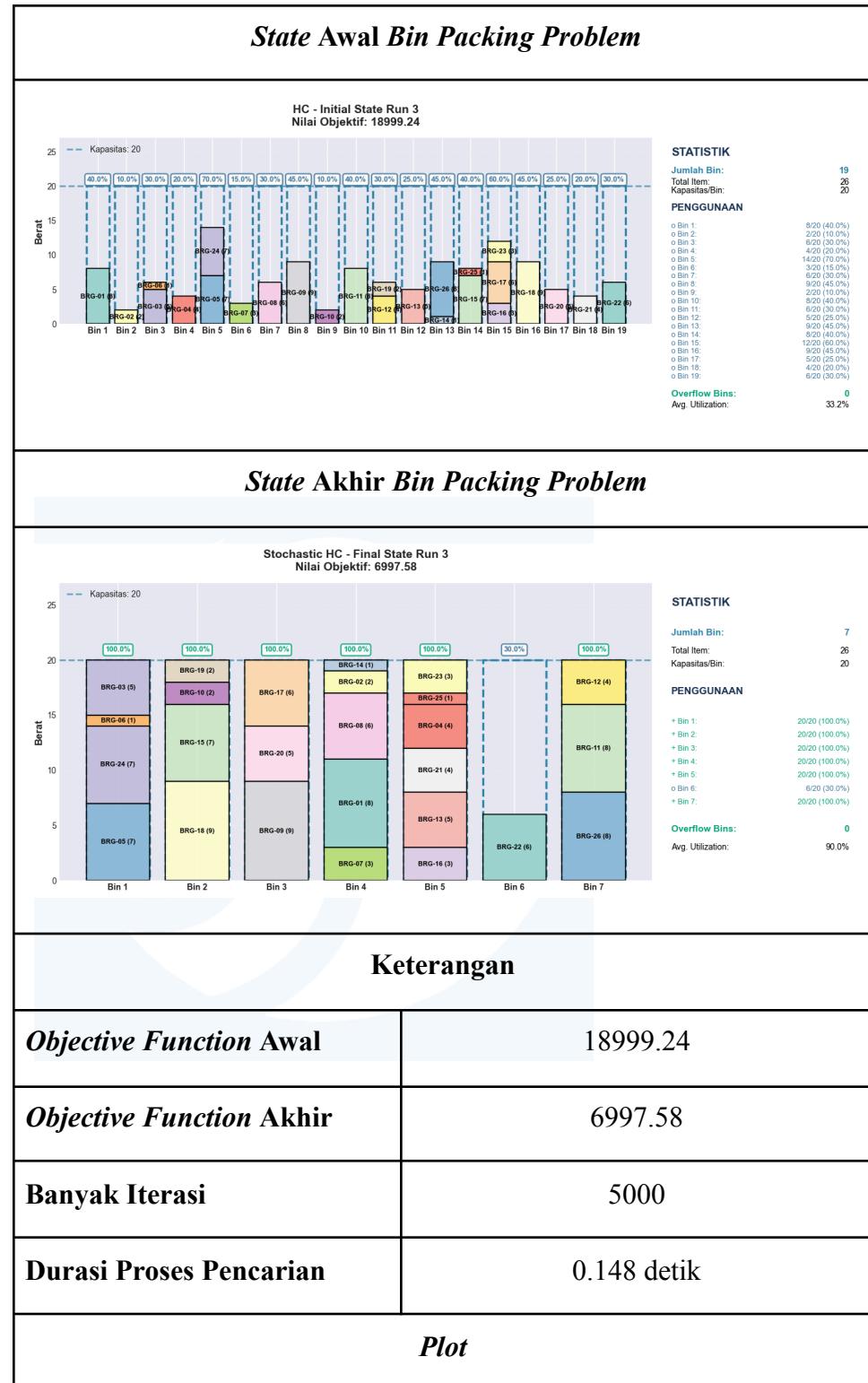
## 2. Eksperimen 2 - Stochastic Hill-Climbing

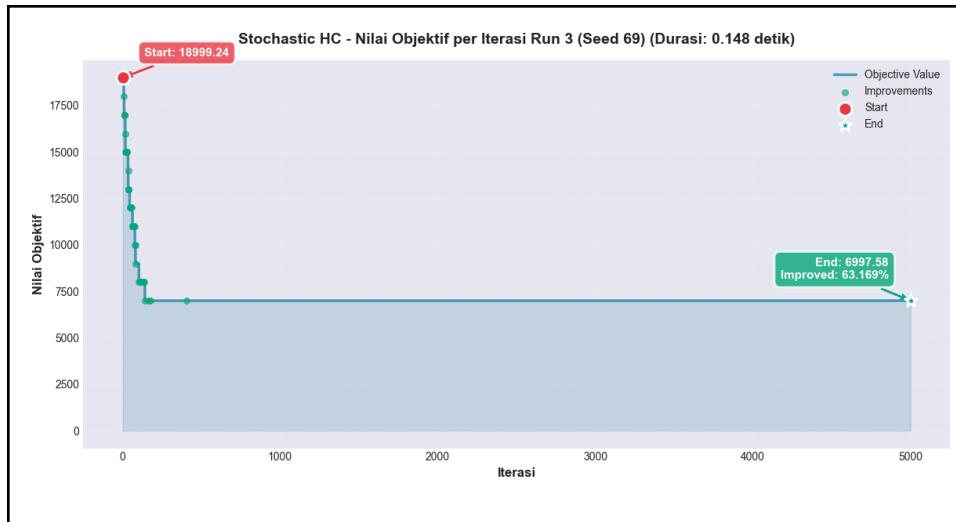




### 3. Eksperimen 3 - Stochastic Hill-Climbing

#### Eksperimen 3





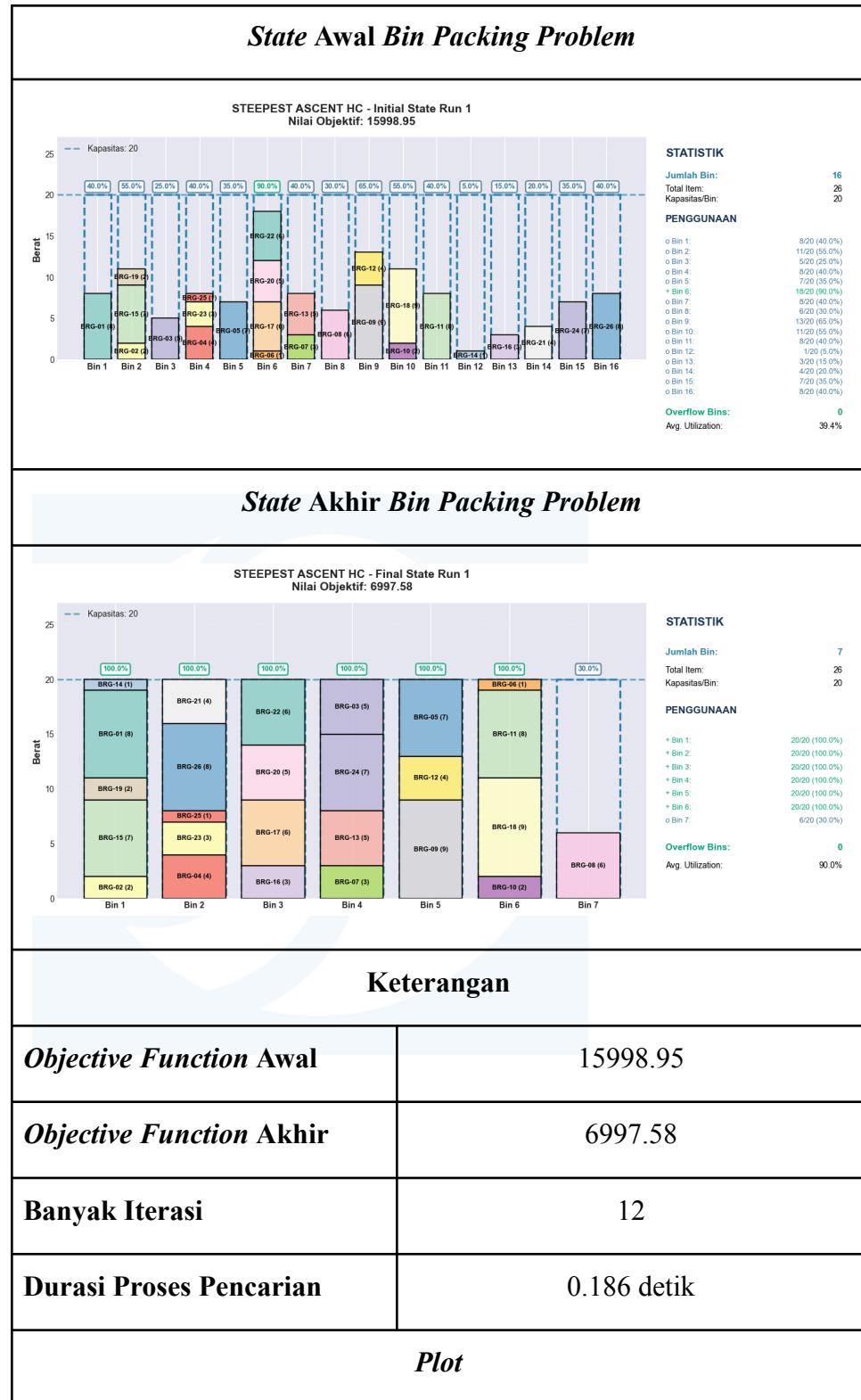
#### 4.2. Steepest Ascent Hill-Climbing

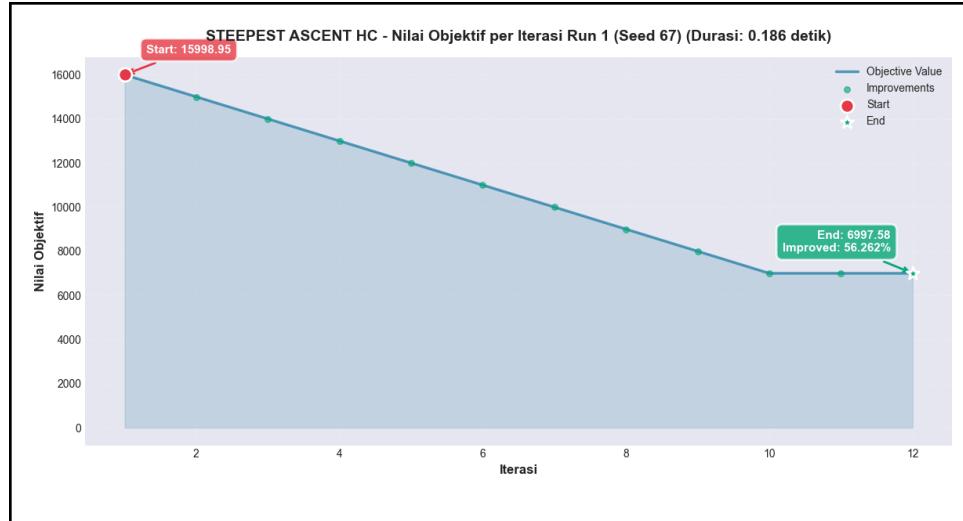
Secara umum, semua eksperimen dimulai dengan nilai objektif yang tinggi (yang menandakan pengemasan yang buruk) dengan nilai 15,998.95, 20,169,898.80, dan 18,999.24. Adapun jumlah kontainer yang berbeda-beda tiap eksperimen dan secara progresif mencapai nilai objektif yang jauh lebih rendah. Hal ini menunjukkan adanya peningkatan secara signifikan dalam efisiensi pengemasan. Pada setiap eksperimen, state akhir menggunakan hanya 7 kontainer, dan pemanfaatan ruang rata-rata mencapai 90%, yang menunjukkan bahwa algoritma berhasil memaksimalkan kapasitas setiap kontainer. Rata-rata persentase *improvement* atau peningkatan adalah 73,02%. Persentase peningkatan yang dicapai bervariasi, dengan peningkatan tertinggi (99.65%) pada eksperimen kedua dan yang terendah (56.26%) pada eksperimen pertama. Hal ini mungkin dipengaruhi oleh konfigurasi awal dan jumlah iterasi yang digunakan di setiap eksperimen.

**Kesimpulan:** Algoritma *Steepest Ascent Hill-Climbing* terbukti efektif dalam menyelesaikan *Bin Packing Problem* dengan mengurangi jumlah kontainer yang digunakan dan meningkatkan pemanfaatan ruang.

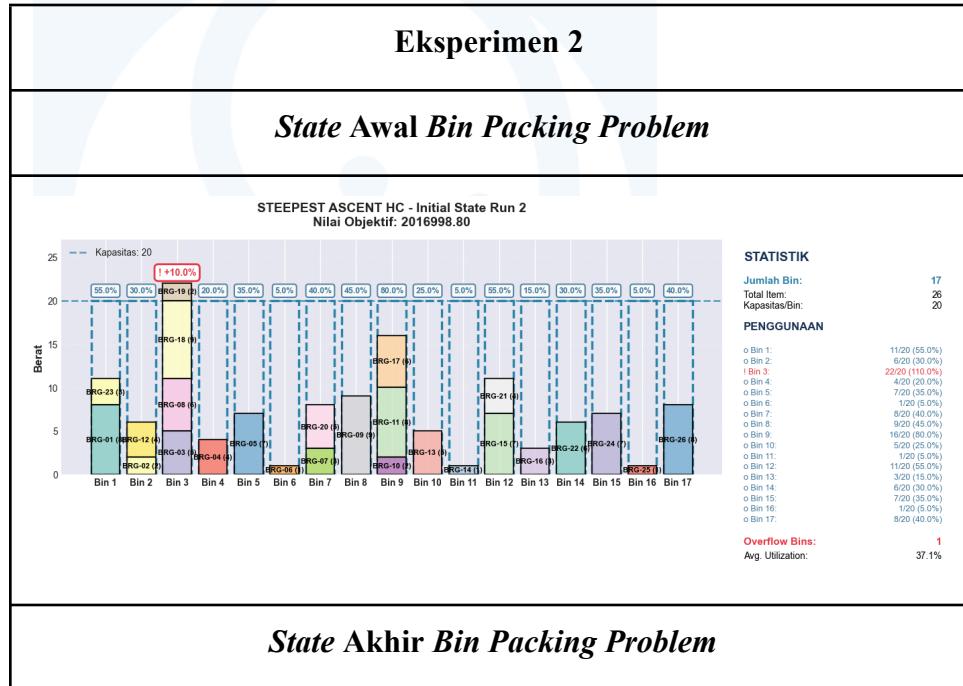
##### 1. Eksperimen 1 - Steepest Ascent Hill-Climbing

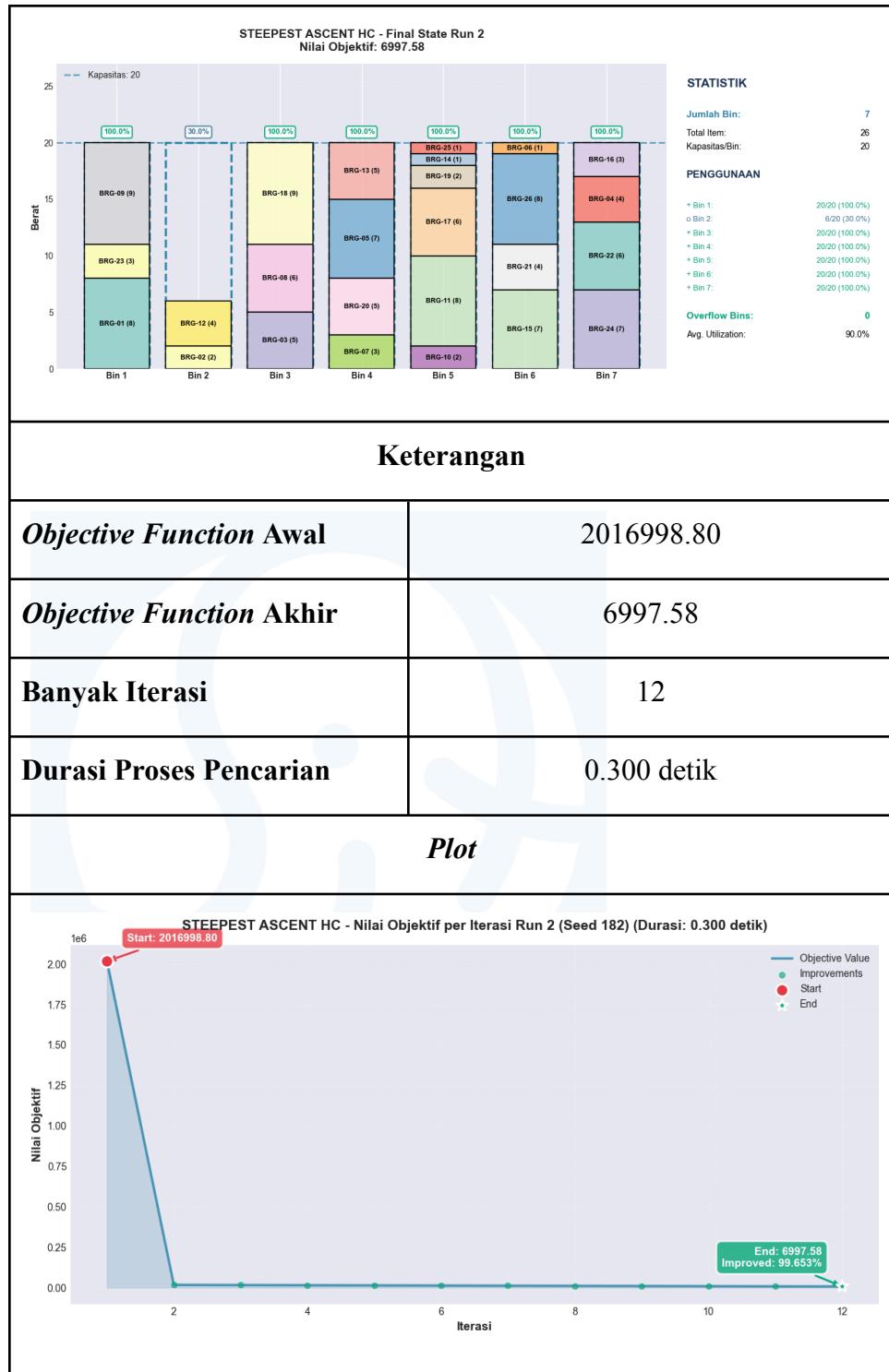
##### Eksperimen 1





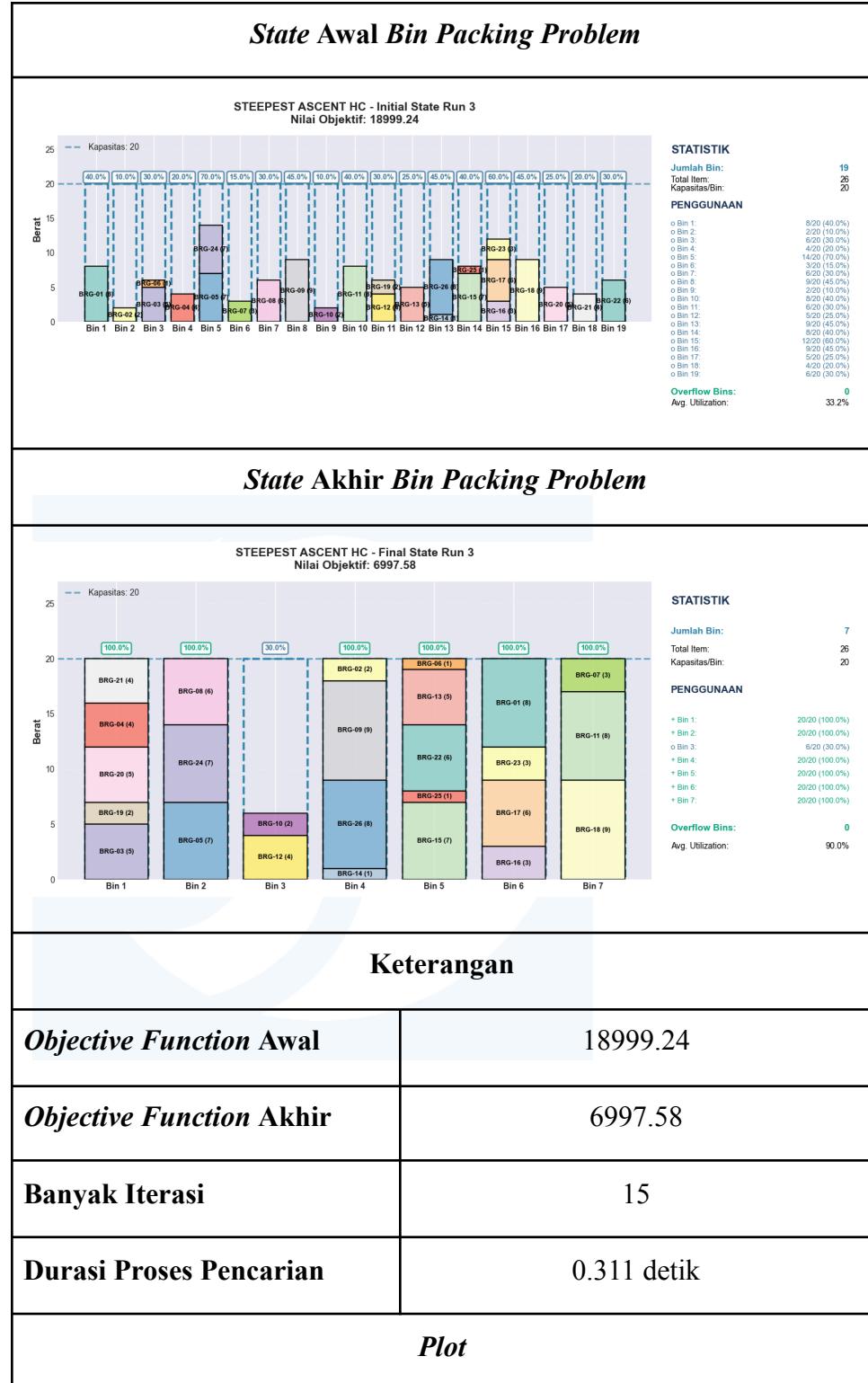
## 2. Eksperimen 2 - Steepest Ascent Hill-Climbing

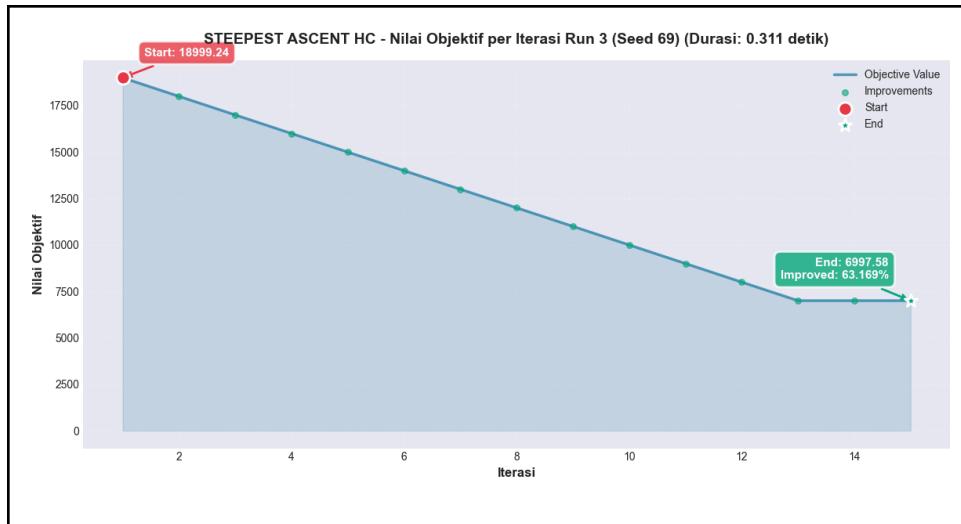




### 3. Eksperimen 3 - Steepest Ascent Hill-Climbing

#### Eksperimen 3





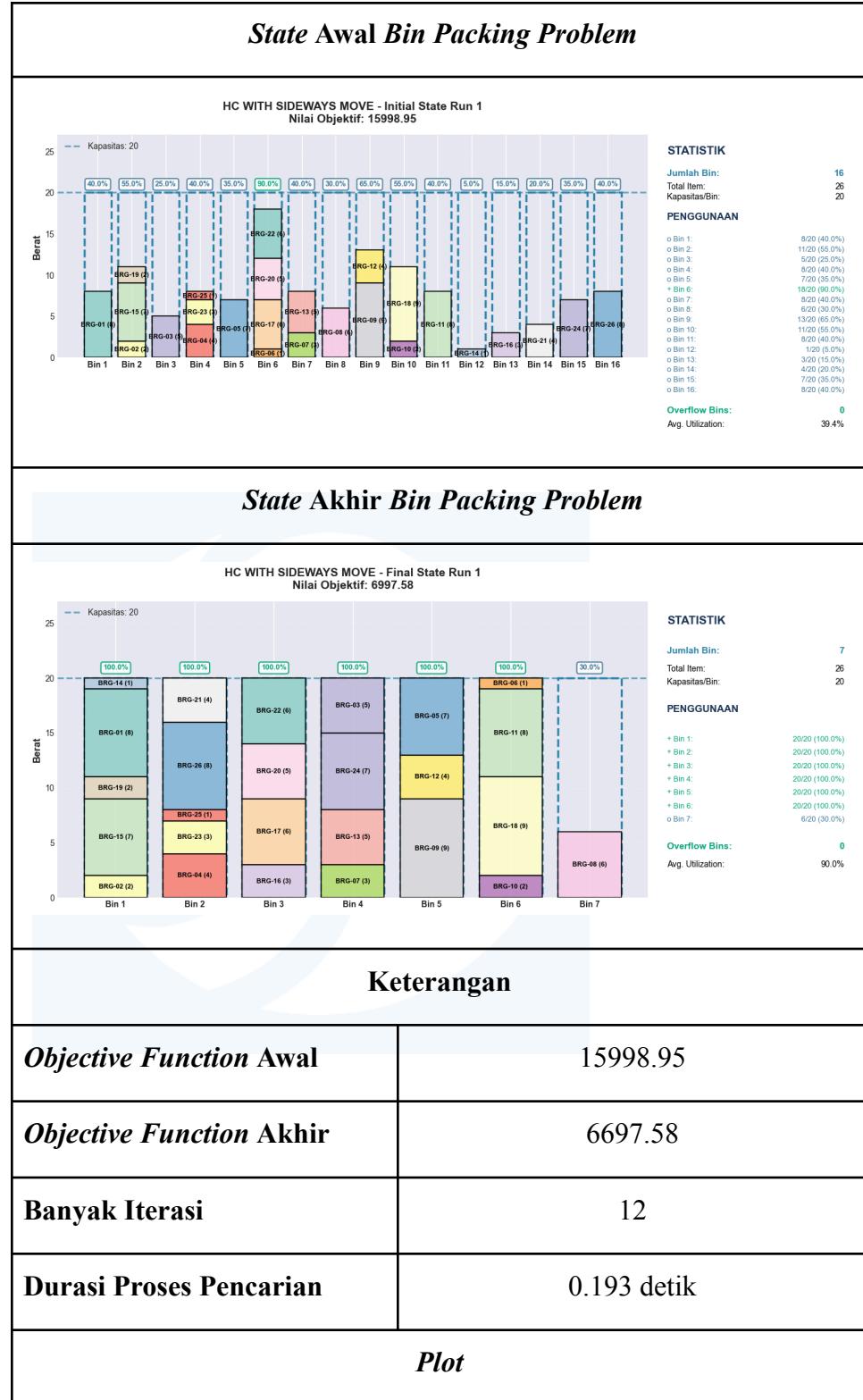
#### 4.3. Hill-Climbing with Sideways Move

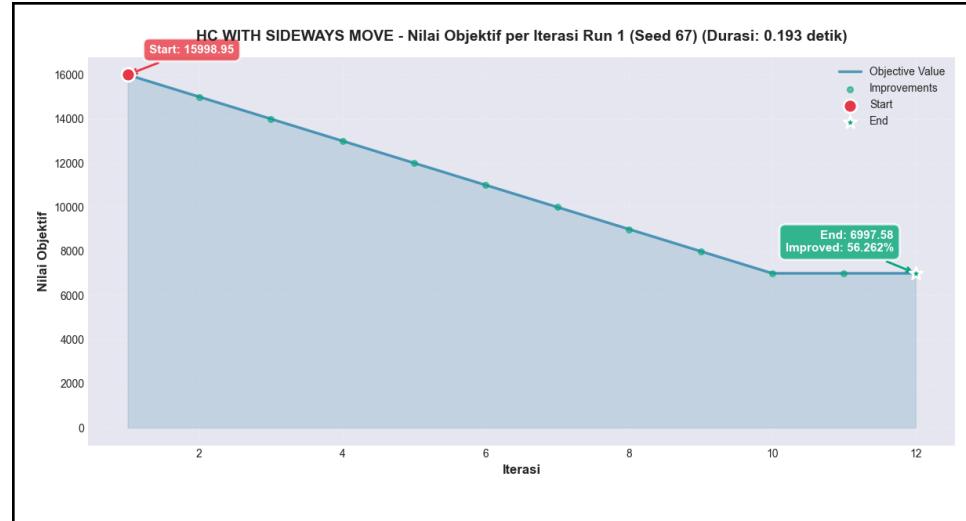
Secara umum, semua eksperimen dimulai dengan nilai objektif yang tinggi, yaitu 15,998.95, 20,169,898.80, dan 18,999.24, yang mengindikasikan pengemasan yang tidak efisien. Jumlah kontainer yang digunakan bervariasi di tiap eksperimen, namun secara progresif nilai objektif mengalami penurunan yang signifikan. Hal ini menunjukkan adanya peningkatan efisiensi pengemasan yang jelas. Pada setiap eksperimen, *state akhir* menggunakan hanya 7 kontainer, dengan pemanfaatan ruang rata-rata mencapai 90%, yang menunjukkan bahwa algoritma berhasil memaksimalkan kapasitas setiap kontainer. Rata-rata persentase peningkatan atau *improvement* yang dicapai adalah 73,02%, dengan peningkatan tertinggi (99.65%) pada eksperimen kedua dan yang terendah (56.26%) pada eksperimen pertama. Perbedaan ini kemungkinan dipengaruhi oleh konfigurasi awal dan jumlah iterasi yang digunakan di setiap eksperimen.

**Kesimpulan:** Algoritma *Hill-Climbing with Sideways Move* terbukti efektif dalam menyelesaikan *Bin Packing Problem* dengan mengurangi jumlah kontainer yang digunakan dan meningkatkan pemanfaatan ruang.

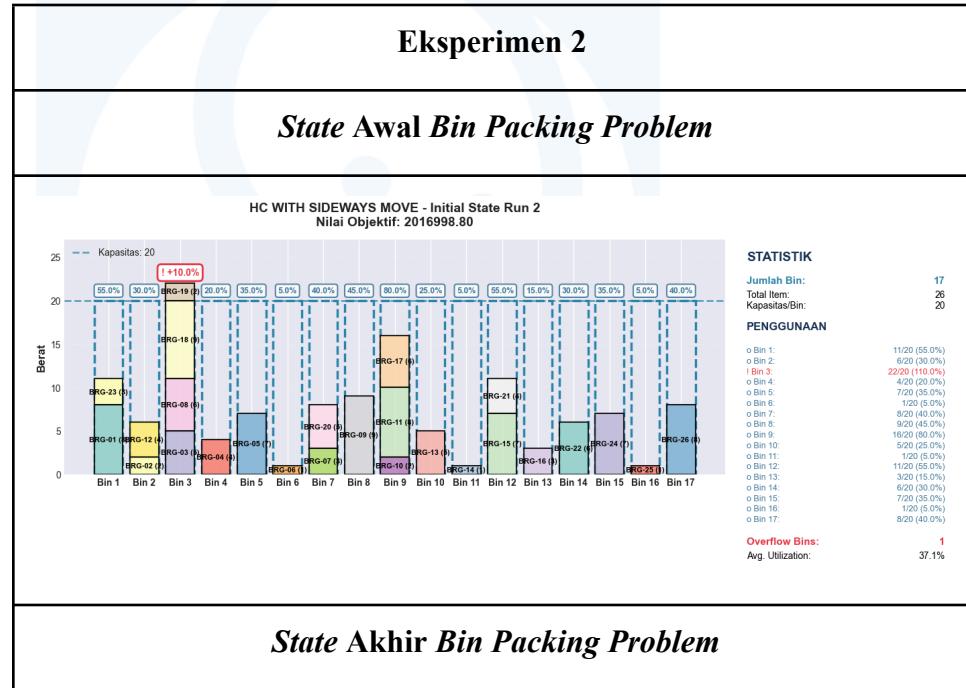
##### 1. Eksperimen 1 - Hill-Climbing with Sideways Move

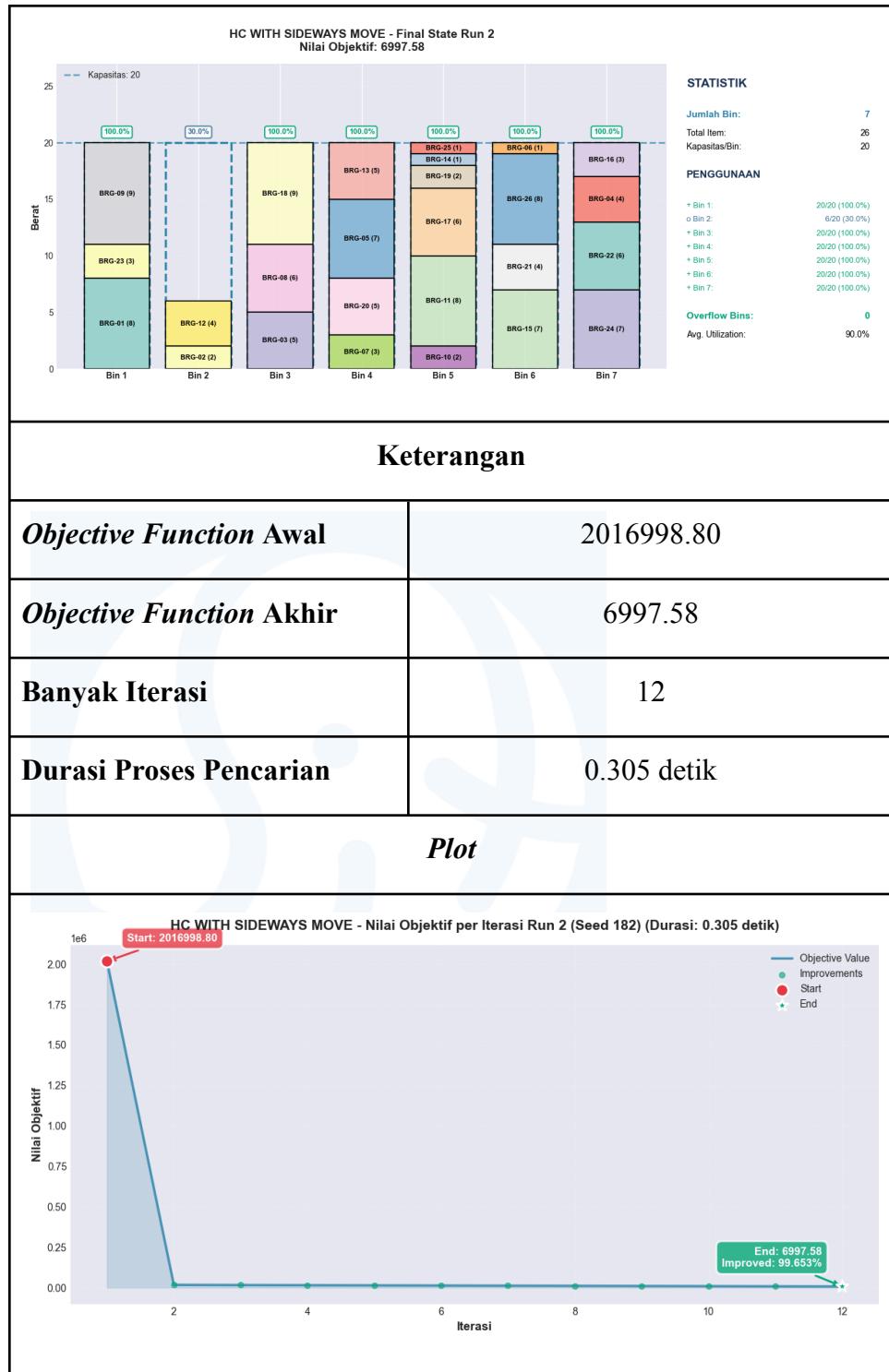
##### Eksperimen 1





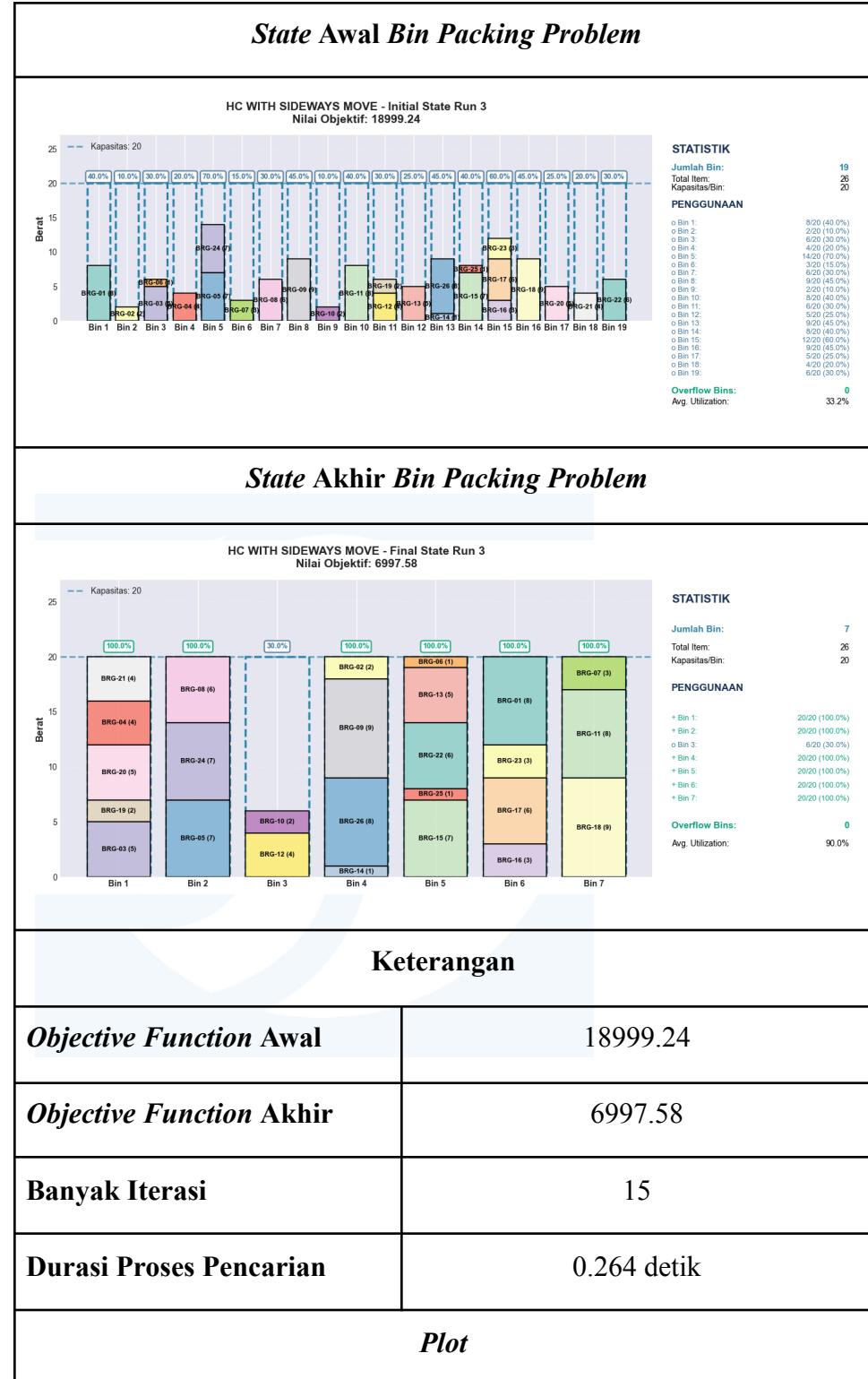
## 2. Eksperimen 2 - Hill-Climbing with Sideways Move

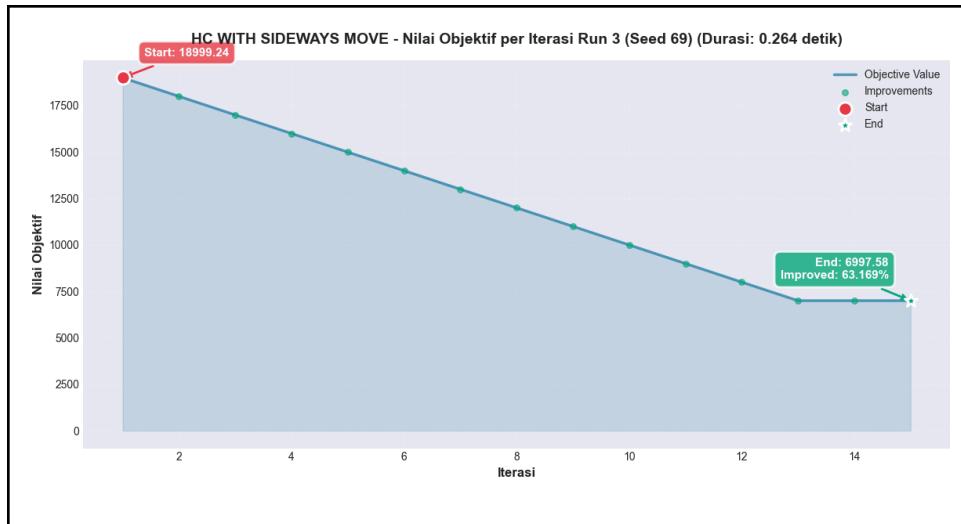




### 3. Eksperimen 3 - Hill-Climbing with Sideways Move

#### Eksperimen 3





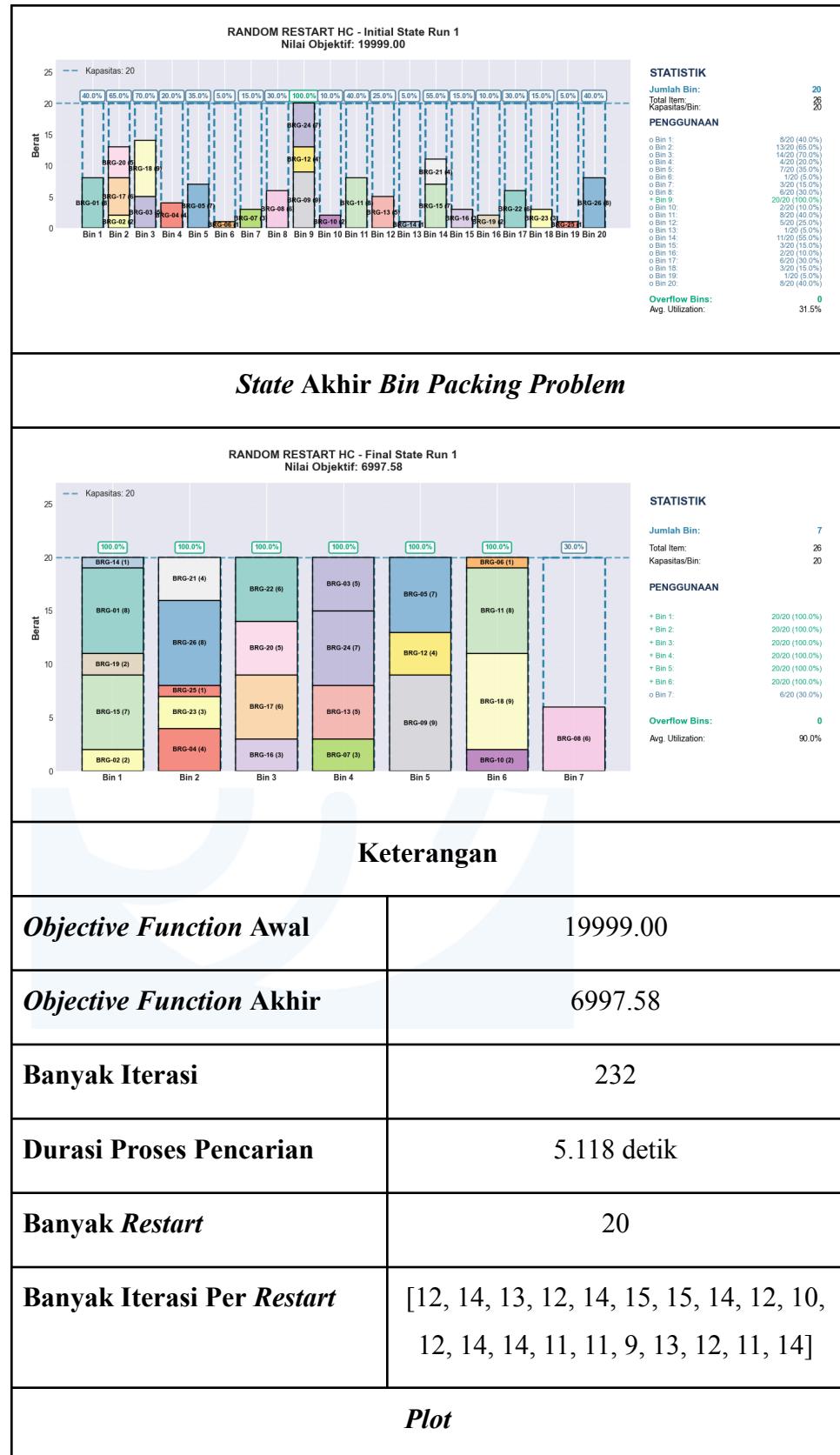
#### 4.4. Random Restart Hill-Climbing

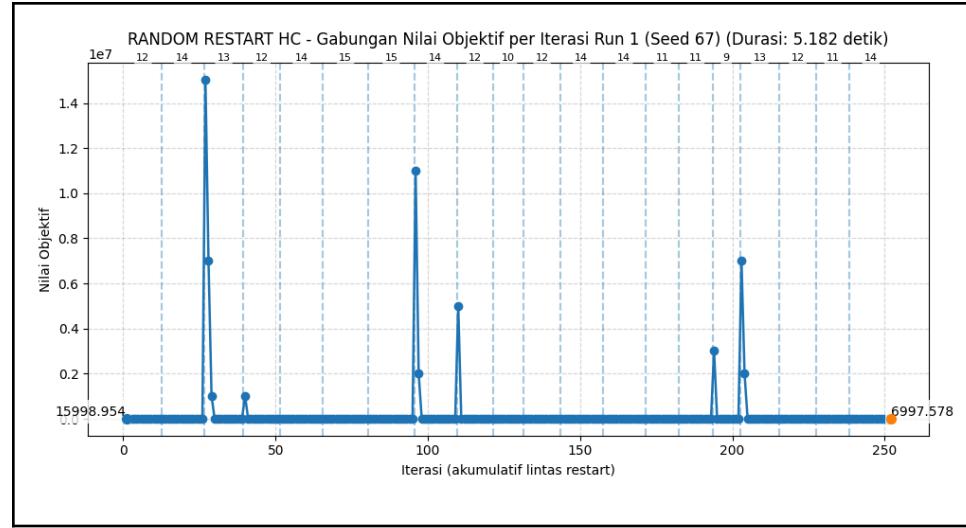
Secara umum, semua eksperimen dimulai dengan nilai objektif yang tinggi, yaitu 19999.00, 17999.15, dan 18999.06 yang mengindikasikan pengemasan awal yang tidak efisien. Jumlah kontainer yang digunakan berbeda-beda di setiap eksperimen, namun secara progresif nilai objektif mengalami penurunan yang signifikan hingga mencapai 6997.58 pada seluruh eksperimen. Hal ini menunjukkan peningkatan efisiensi pengemasan yang jelas. Pada setiap eksperimen, state akhir menggunakan hanya 7 kontainer dengan pemanfaatan ruang rata-rata mencapai 90%, menandakan algoritma berhasil memaksimalkan kapasitas setiap kontainer.

**Kesimpulan:** Algoritma *Random Restart Hill-Climbing* terbukti efektif dalam menyelesaikan *Bin Packing Problem* dengan meningkatkan efisiensi pengemasan dan menghindari jebakan local optimum melalui mekanisme restart acak.

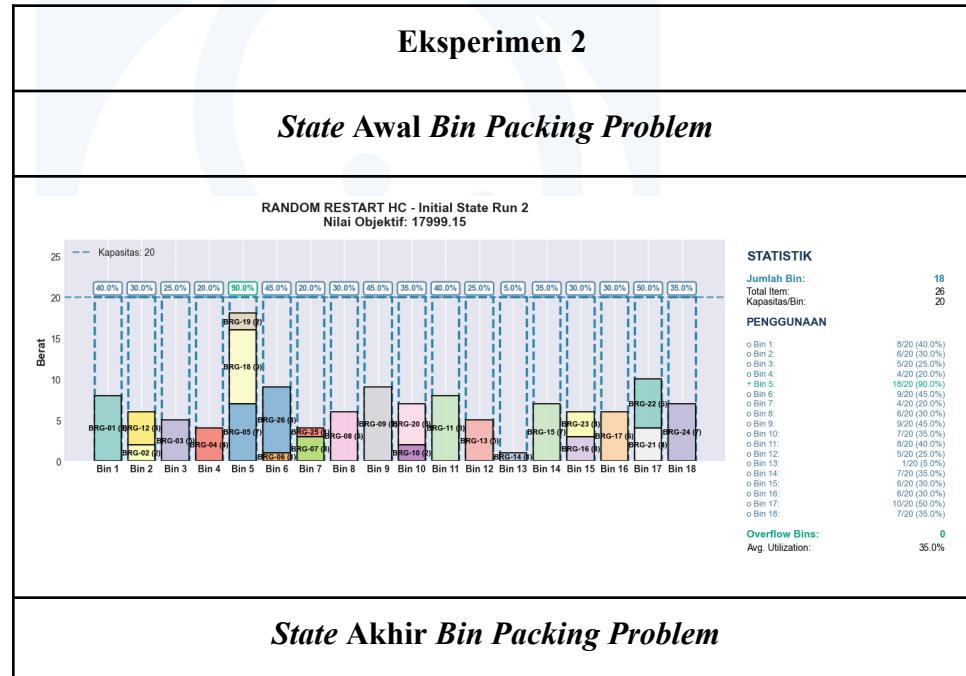
##### 1. Eksperimen 1 - Random Restart Hill-Climbing

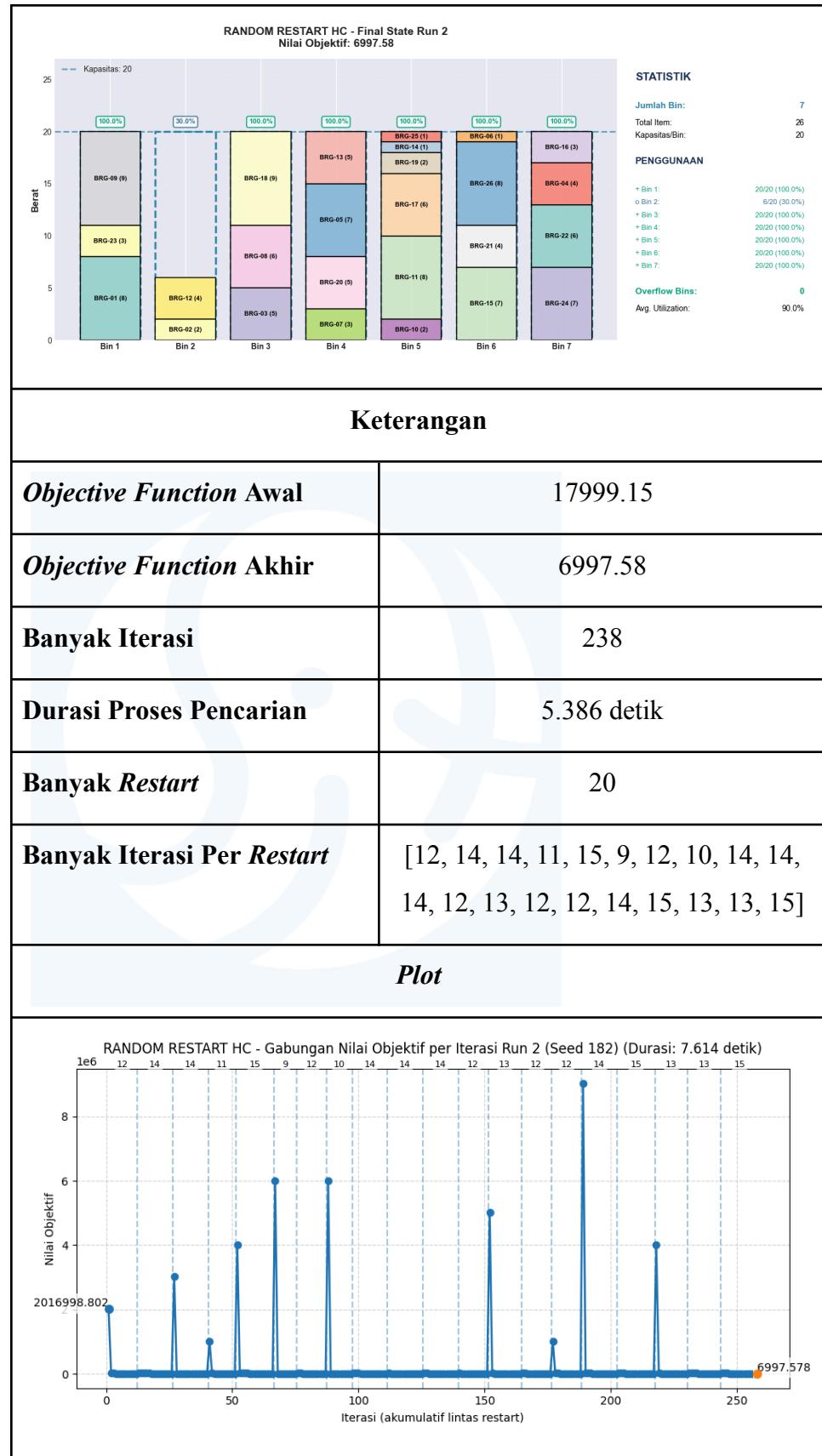
Eksperimen 1
State Awal Bin Packing Problem



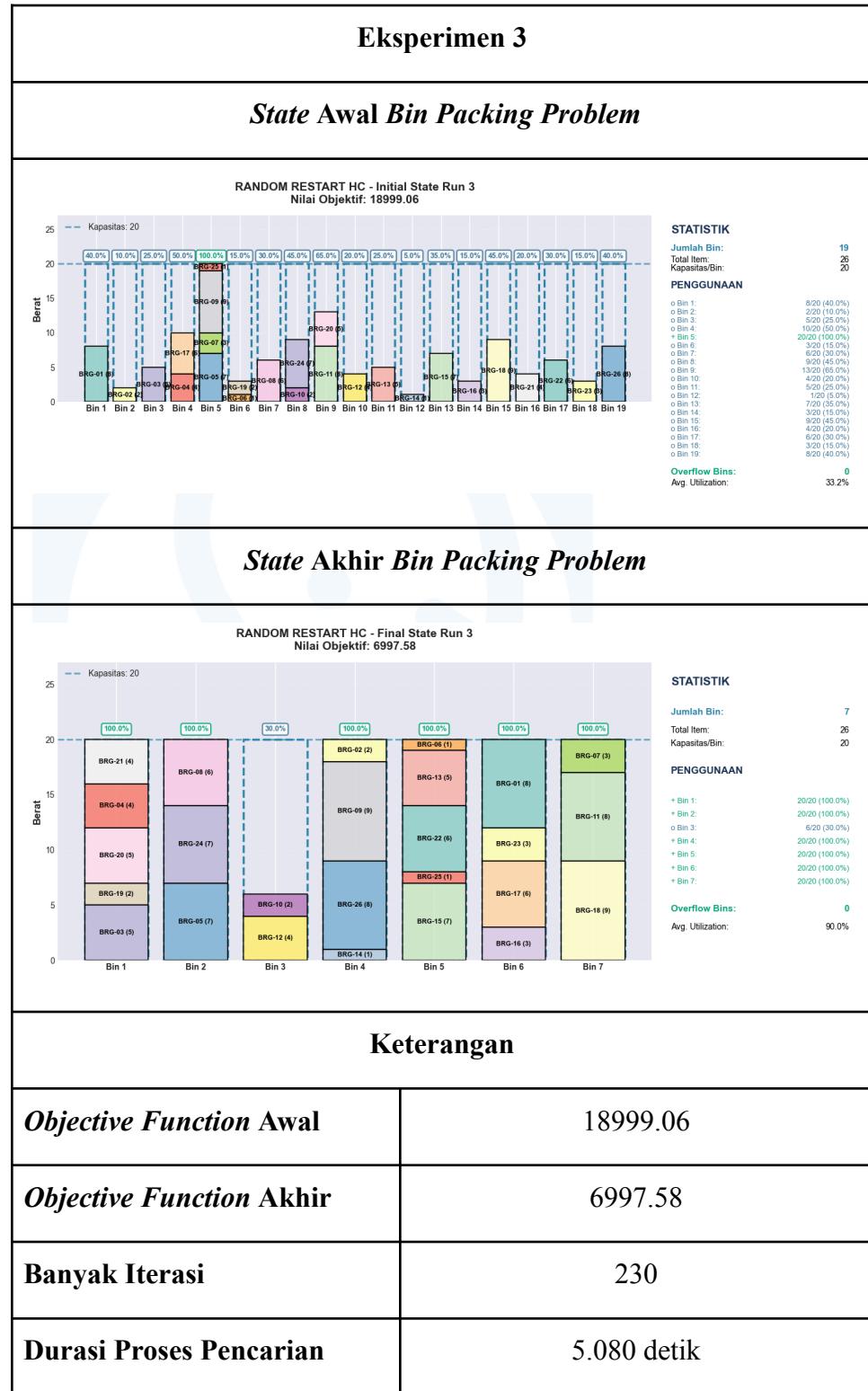


## 2. Eksperimen 2 - Random Restart Hill-Climbing





### 3. Eksperimen 3 - Random Restart Hill-Climbing



<b>Banyak Restart</b>	20																		
<b>Banyak Iterasi Per Restart</b>	[15, 14, 11, 13, 12, 14, 12, 13, 15, 11, 11, 13, 11, 12, 13, 12, 12, 13, 10, 13]																		
<b>Plot</b>																			
<p>RANDOM RESTART HC - Gabungan Nilai Objektif per Iterasi Run 3 (Seed 69) (Durasi: 8.552 detik)</p> <p>Nilai Objektif</p> <p>Iterasi (akumulatif lintas restart)</p> <table border="1"> <caption>Data points from the Objective Value vs Iteration plot</caption> <thead> <tr> <th>Iterasi (akumulatif lintas restart)</th> <th>Nilai Objektif</th> </tr> </thead> <tbody> <tr><td>0</td><td>1e6</td></tr> <tr><td>0</td><td>18999.236</td></tr> <tr><td>~100</td><td>1.0</td></tr> <tr><td>~140</td><td>4.0</td></tr> <tr><td>~170</td><td>4.0</td></tr> <tr><td>~190</td><td>2.0</td></tr> <tr><td>~200</td><td>2.0</td></tr> <tr><td>250</td><td>6997.578</td></tr> </tbody> </table>		Iterasi (akumulatif lintas restart)	Nilai Objektif	0	1e6	0	18999.236	~100	1.0	~140	4.0	~170	4.0	~190	2.0	~200	2.0	250	6997.578
Iterasi (akumulatif lintas restart)	Nilai Objektif																		
0	1e6																		
0	18999.236																		
~100	1.0																		
~140	4.0																		
~170	4.0																		
~190	2.0																		
~200	2.0																		
250	6997.578																		

#### 4.5. Simulated Annealing

Secara umum, semua eksperimen pada algoritma *Simulated Annealing* dimulai dengan nilai objektif yang sangat tinggi, yaitu 25,007,996.99, 39,006,995.63, dan 25,008,997.23, yang mengindikasikan kondisi awal pengemasan yang sangat tidak efisien dan masih jauh dari optimum. Selama proses pencarian, nilai objektif menurun secara signifikan hingga mencapai 6,997.58 pada seluruh eksperimen, menunjukkan peningkatan efisiensi pengemasan yang substansial.

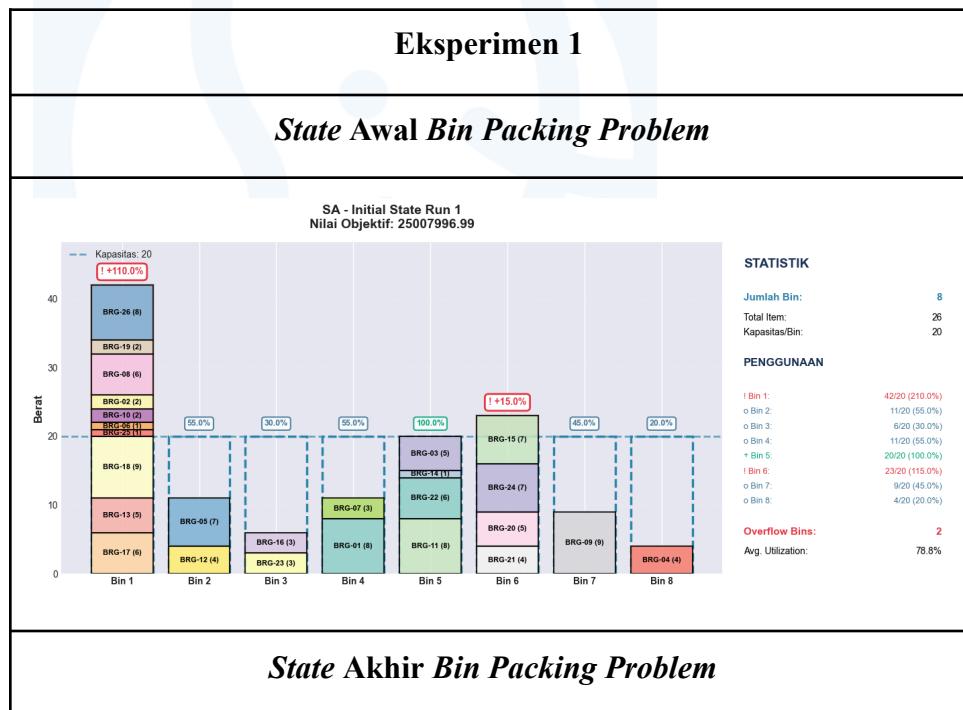
Jumlah kontainer yang digunakan pada *state* akhir tetap 7 kontainer, dengan rata-rata pemanfaatan ruang sekitar 90%, menandakan algoritma mampu memaksimalkan kapasitas setiap kontainer secara konsisten. Durasi proses pencarian tergolong cepat, berkisar antara 0.098 detik hingga 0.196 detik, dengan jumlah iterasi tinggi (hingga 5000 iterasi) yang memungkinkan eksplorasi luas terhadap ruang solusi. Frekuensi terjebak di *local optima* juga relatif rendah,

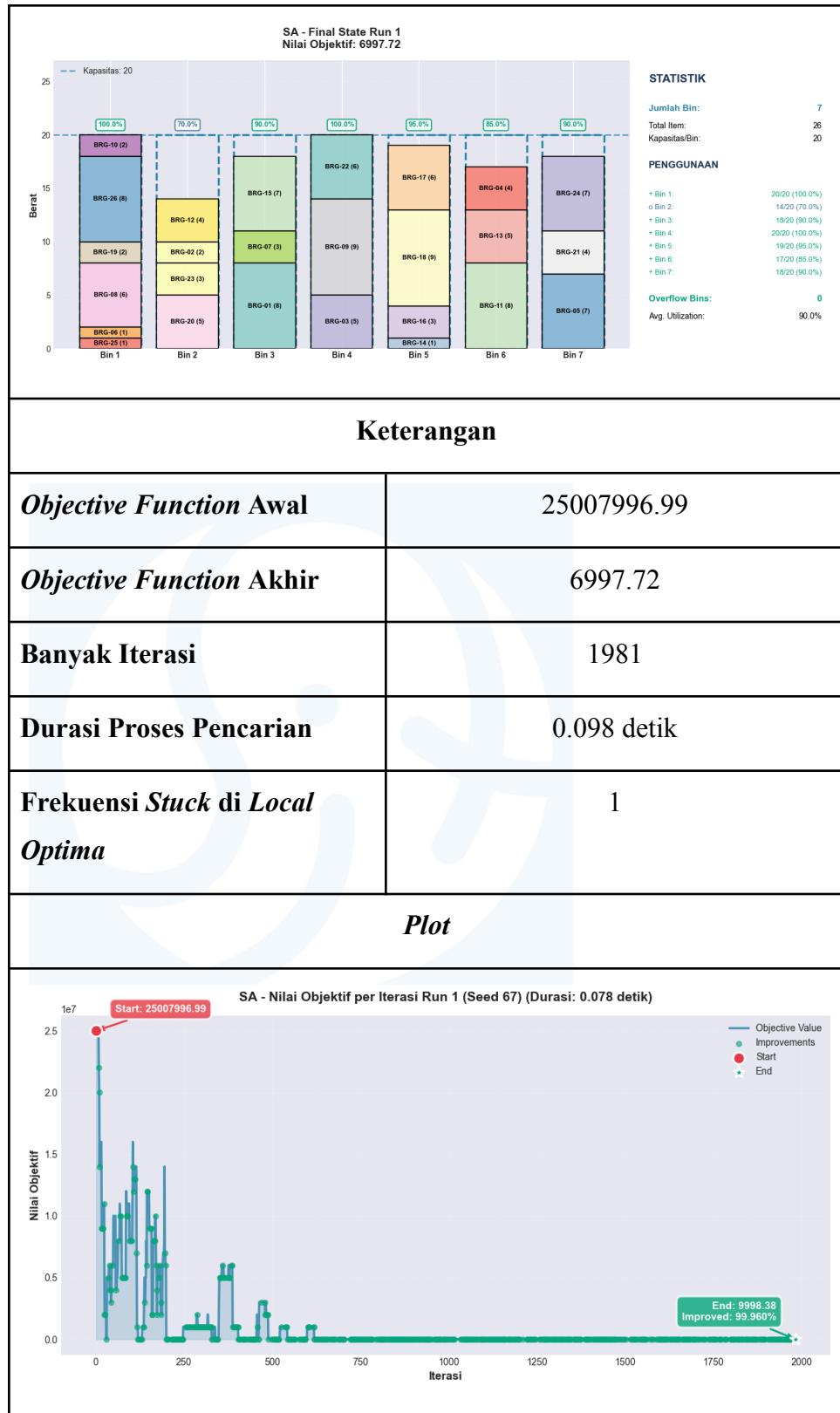
menunjukkan mekanisme penerimaan solusi yang lebih buruk (*acceptance of worse states*) berfungsi dengan baik selama suhu masih tinggi

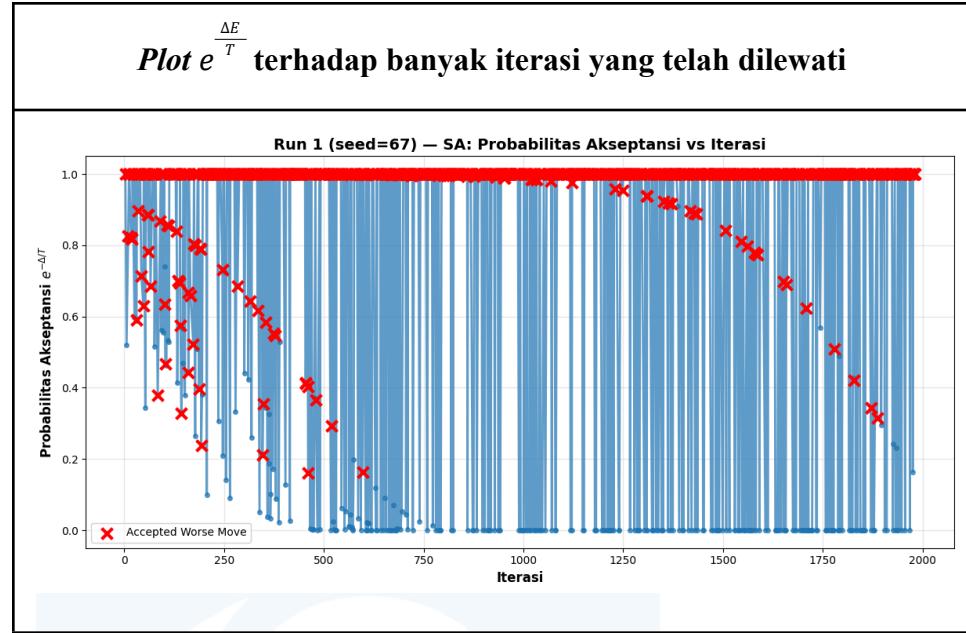
Rata-rata persentase peningkatan (*improvement*) yang dicapai adalah 99.97%, dengan peningkatan tertinggi (99.98%) pada eksperimen kedua dan yang terendah (99.97%) pada eksperimen pertama. Perbedaan kecil ini menunjukkan konsistensi performa algoritma pada berbagai inisialisasi acak.

**Kesimpulan:** Algoritma Simulated Annealing terbukti sangat efektif dalam menyelesaikan *Bin Packing Problem* dengan efisiensi pengemasan tertinggi di antara semua varian hill-climbing. Mekanisme *probabilistic acceptance* dan *cooling schedule* memungkinkan algoritma keluar dari jebakan *local optimum* dan mencapai solusi mendekati *global optimum* dengan stabilitas tinggi serta waktu eksekusi yang efisien.

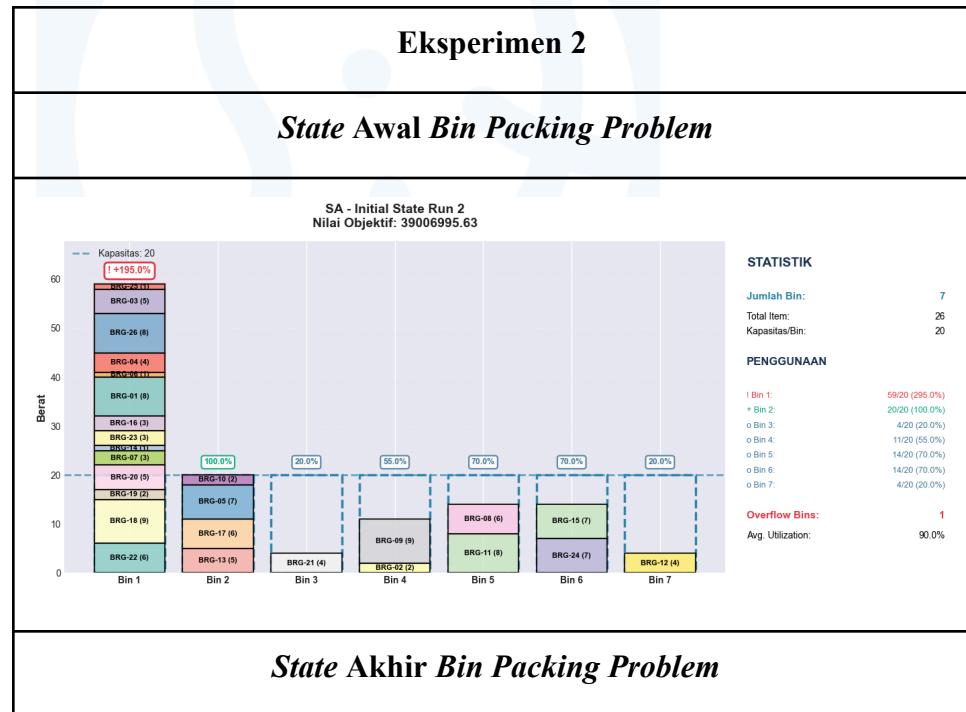
### 1. Eksperimen 1 - Simulated Annealing

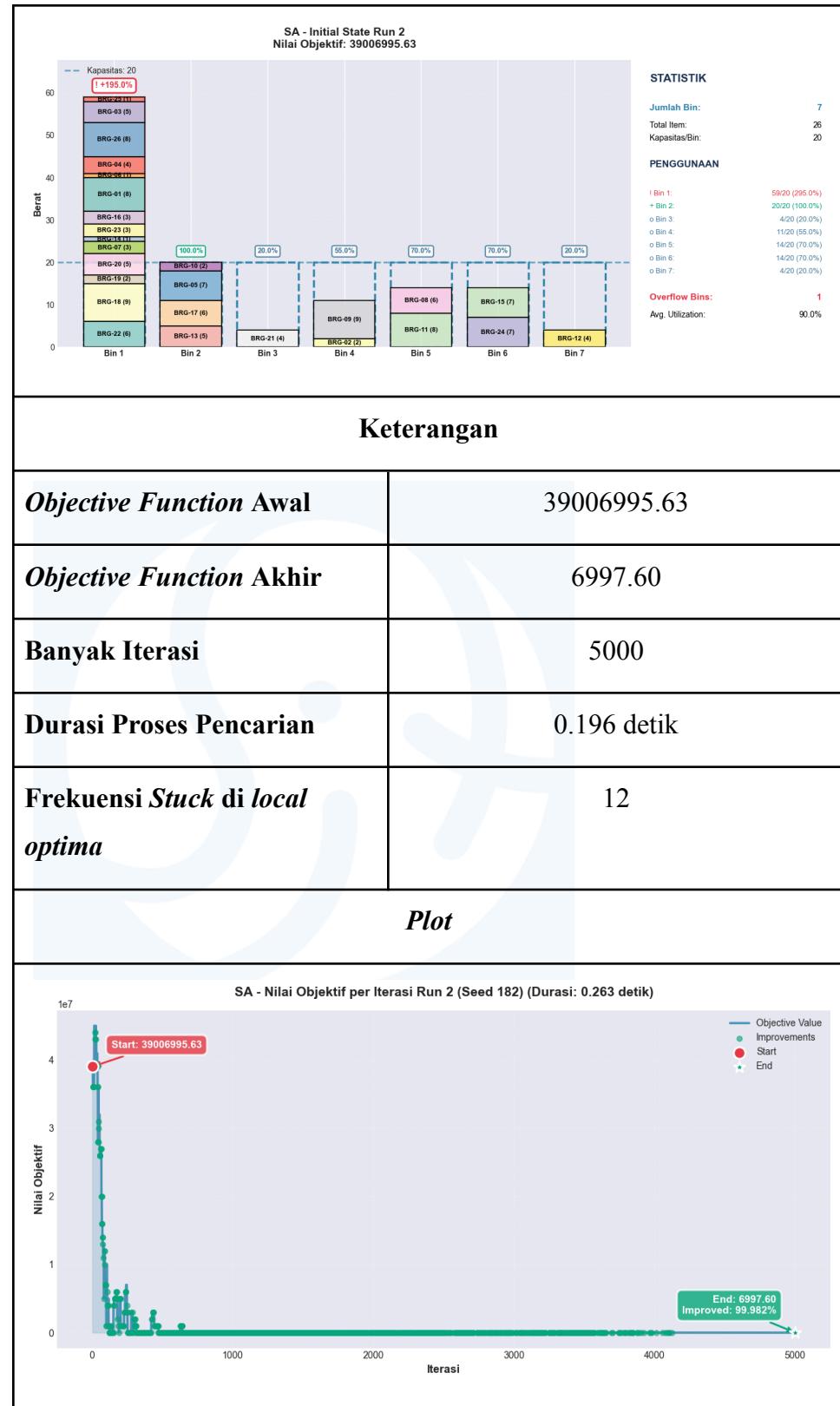


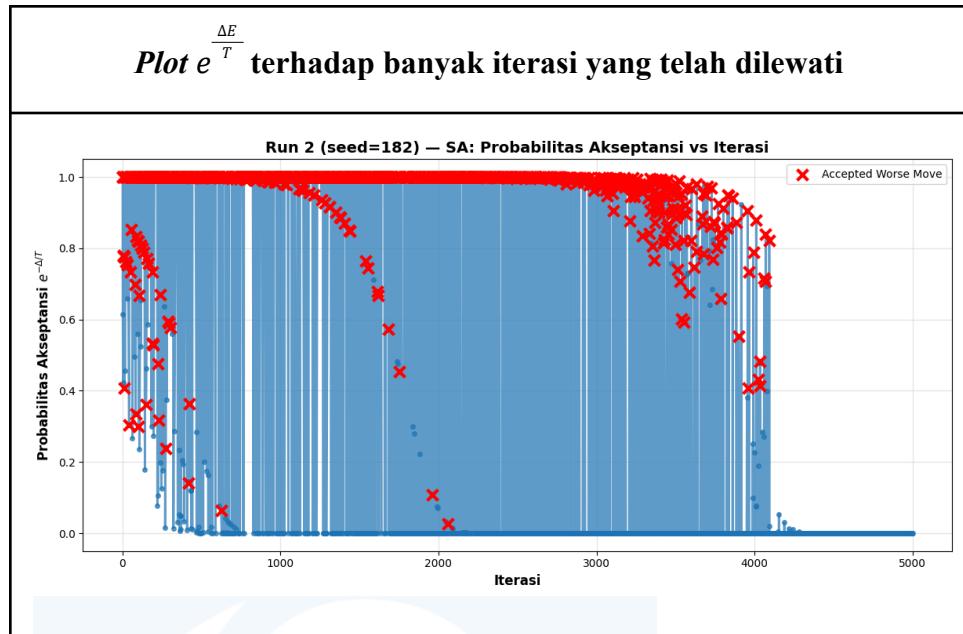




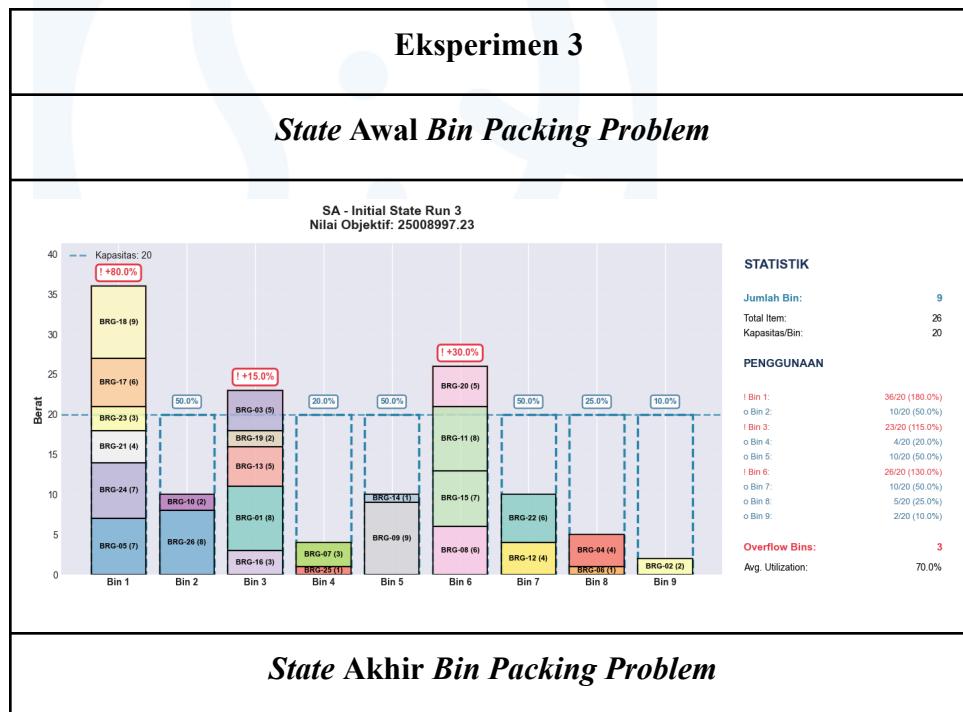
## 2. Eksperimen 2 - Simulated Annealing

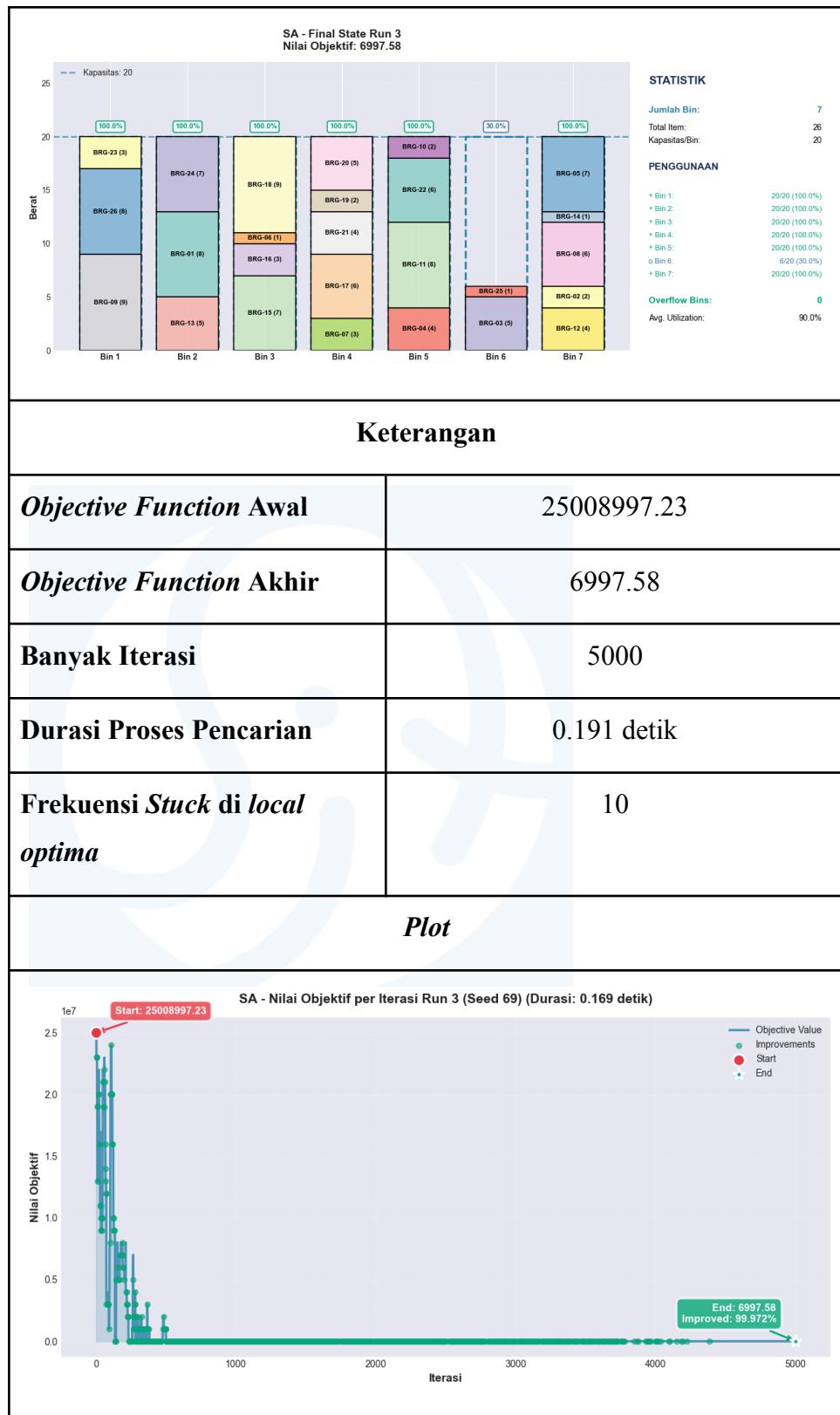


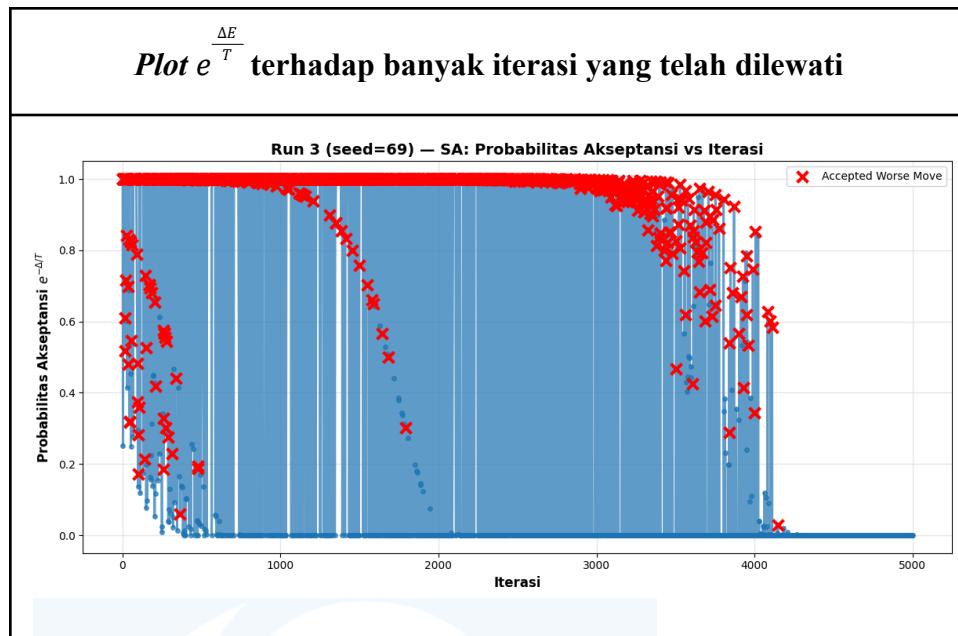




### **3. Eksperimen 3 - *Simulated Annealing***







#### 4.6. Genetic Algorithm

Eksperimen dilakukan dengan dua parameter utama yaitu jumlah populasi dan banyak iterasi, serta dua variasi probabilitas mutasi yang berbeda (0.15 dan 0.5). Ketika jumlah populasi dijadikan kontrol dengan nilai tetap (8), dan banyak iterasi divariasikan menjadi 100, 200, dan 500, hasil menunjukkan bahwa peningkatan jumlah iterasi secara umum menghasilkan nilai objective function akhir yang lebih baik, dengan konvergensi yang lebih stabil pada iterasi tinggi. Namun, setelah titik tertentu, peningkatan iterasi memberikan perbaikan yang semakin kecil, seperti terlihat pada plot eksperimen dengan probabilitas mutasi 0.15, dengan iterasi sebanyak 200, yang malah mengalami kenaikan nilai menjadi 8997.87.

Sebaliknya, ketika banyak iterasi dijadikan kontrol (200) dan jumlah populasi divariasikan menjadi 2, 8, dan 16, terlihat bahwa populasi yang lebih besar cenderung mencapai nilai *objective function* yang lebih dekat ke global optimum, karena keragaman genetik yang lebih tinggi memungkinkan eksplorasi ruang solusi yang lebih luas. Namun, peningkatan populasi juga berbanding lurus dengan durasi proses pencarian, di mana populasi terbesar (16) memerlukan waktu rata-rata 0.324 detik untuk konvergen.

Dari segi pengaruh probabilitas mutasi, eksperimen menunjukkan bahwa nilai probabilitas mutasi yang rendah (0.15) menghasilkan konvergensi cepat namun berisiko terjebak di *local optimum*, sementara nilai probabilitas mutasi lebih tinggi (0.5) membantu keluar dari *local optimum*. Pada umumnya, implementasi algoritma *genetics* biasanya menghasilkan hasil nilai *objective function* yang lebih fluktuatif. Namun, algoritma *genetics* kami mengimplementasikan *elitism*, sebuah pendekatan yang wajibkan algoritma untuk selalu meneruskan individu terbaik pada satu generasi ke generasi berikutnya. Dengan pendekatan tersebut, eksplorasi yang dilakukan dengan nilai probabilitas mutasi yang tinggi bisa lebih bebas tanpa efek samping fluktuasi yang tinggi karena adanya jaminan untuk meneruskan individu terbaik pada tiap generasi.

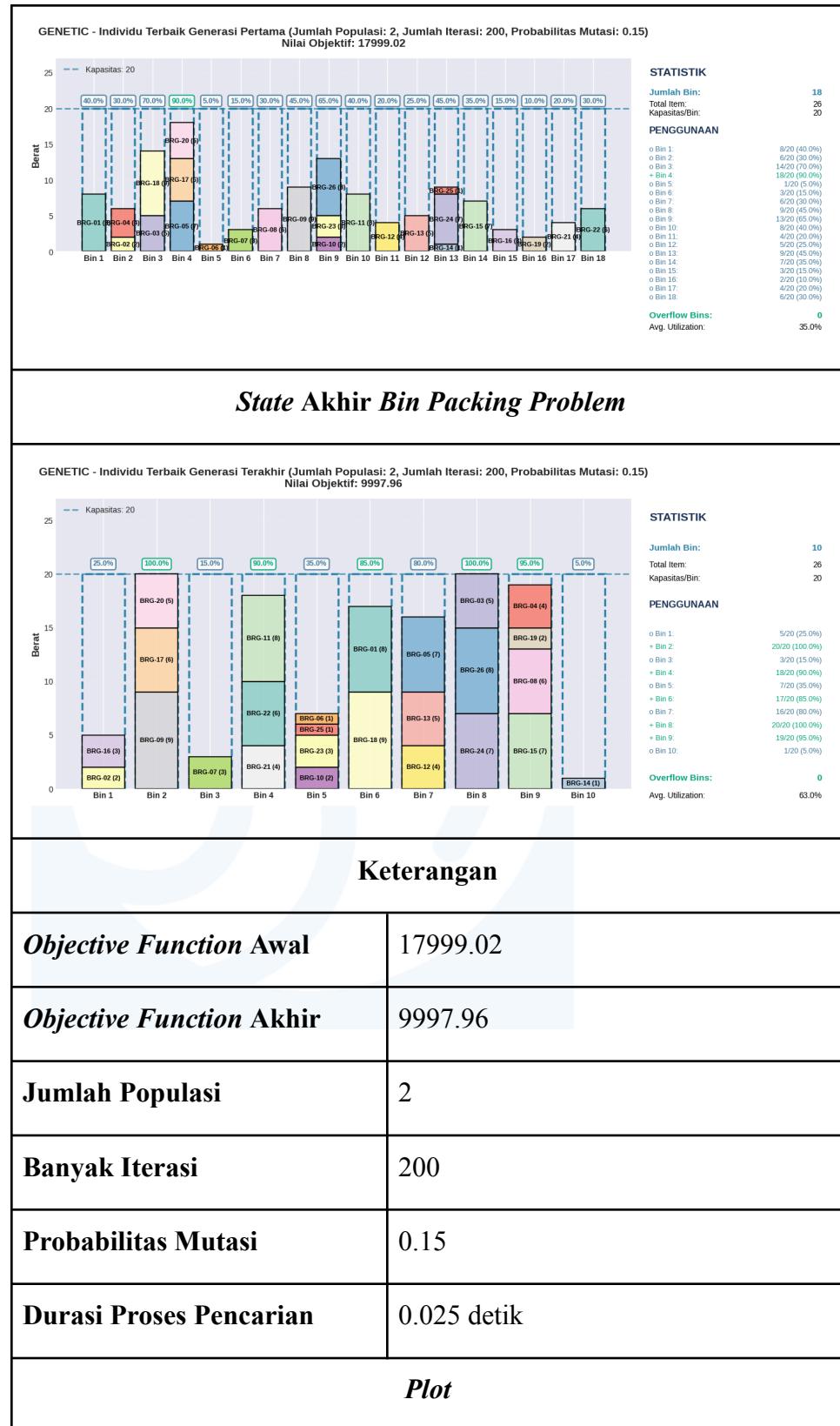
**Kesimpulan:** Algoritma *Genetics* menunjukkan konsistensi hasil akhir yang cukup tinggi antar percobaan, dengan nilai *objective function* terbaik sebesar 6997.58 dengan durasi pencarian tercepat selama 0.247 detik pada konfigurasi probabilitas mutasi 0.15, jumlah populasi 8, dan banyak iterasi 500. Dibandingkan dengan algoritma local search lain seperti Hill Climbing atau Simulated Annealing, GA mampu mencapai hasil yang lebih dekat ke global optimum karena mekanisme crossover dan mutasi yang menjaga keberagaman solusi. Meskipun demikian, durasi pencarinya relatif lebih lama dibandingkan local search sederhana karena proses evaluasi banyak individu dalam populasi setiap iterasi.

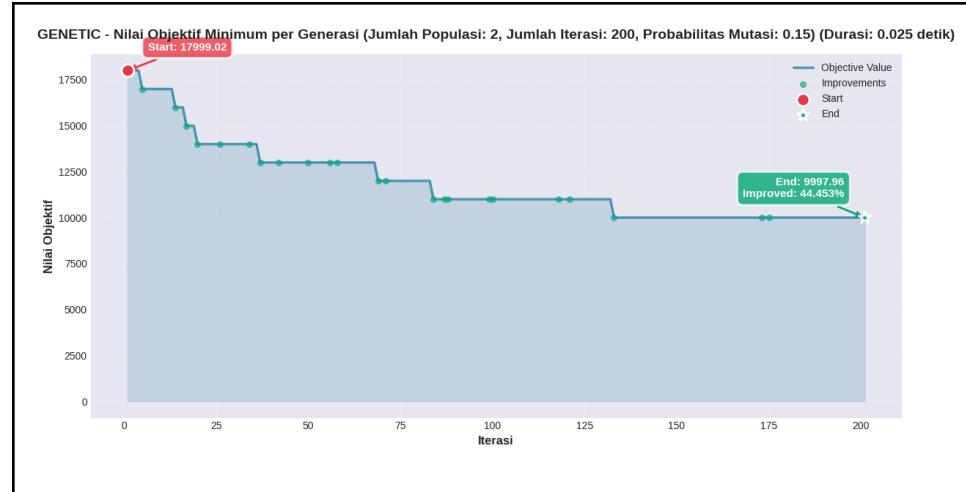
## 1. Variasi Jumlah Populasi (Probabilitas Mutasi = 0.15)

- a. Populasi = 2

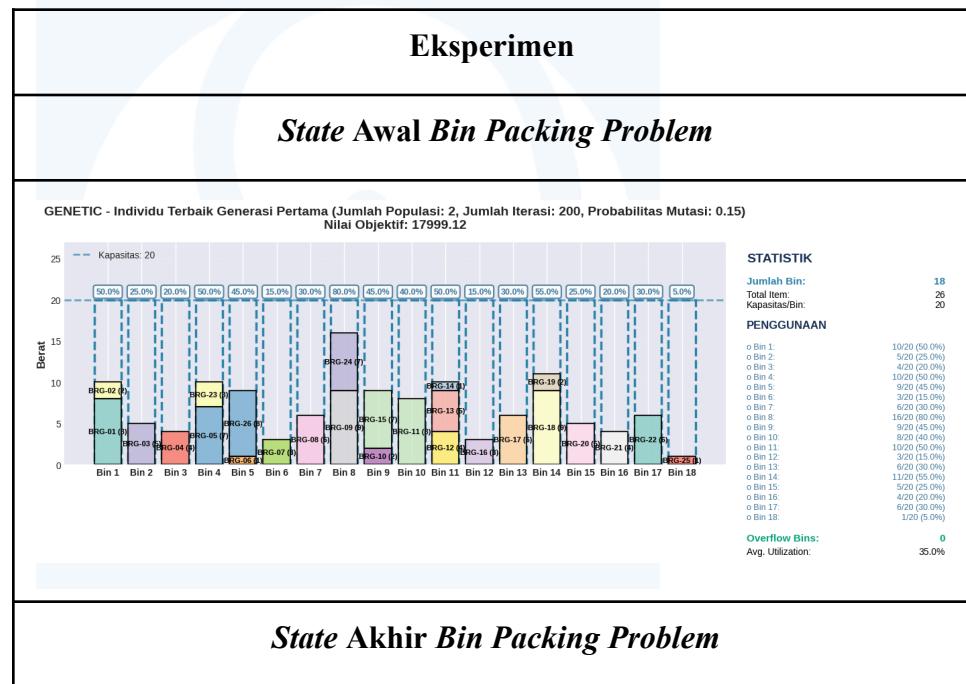
- i. Percobaan 1

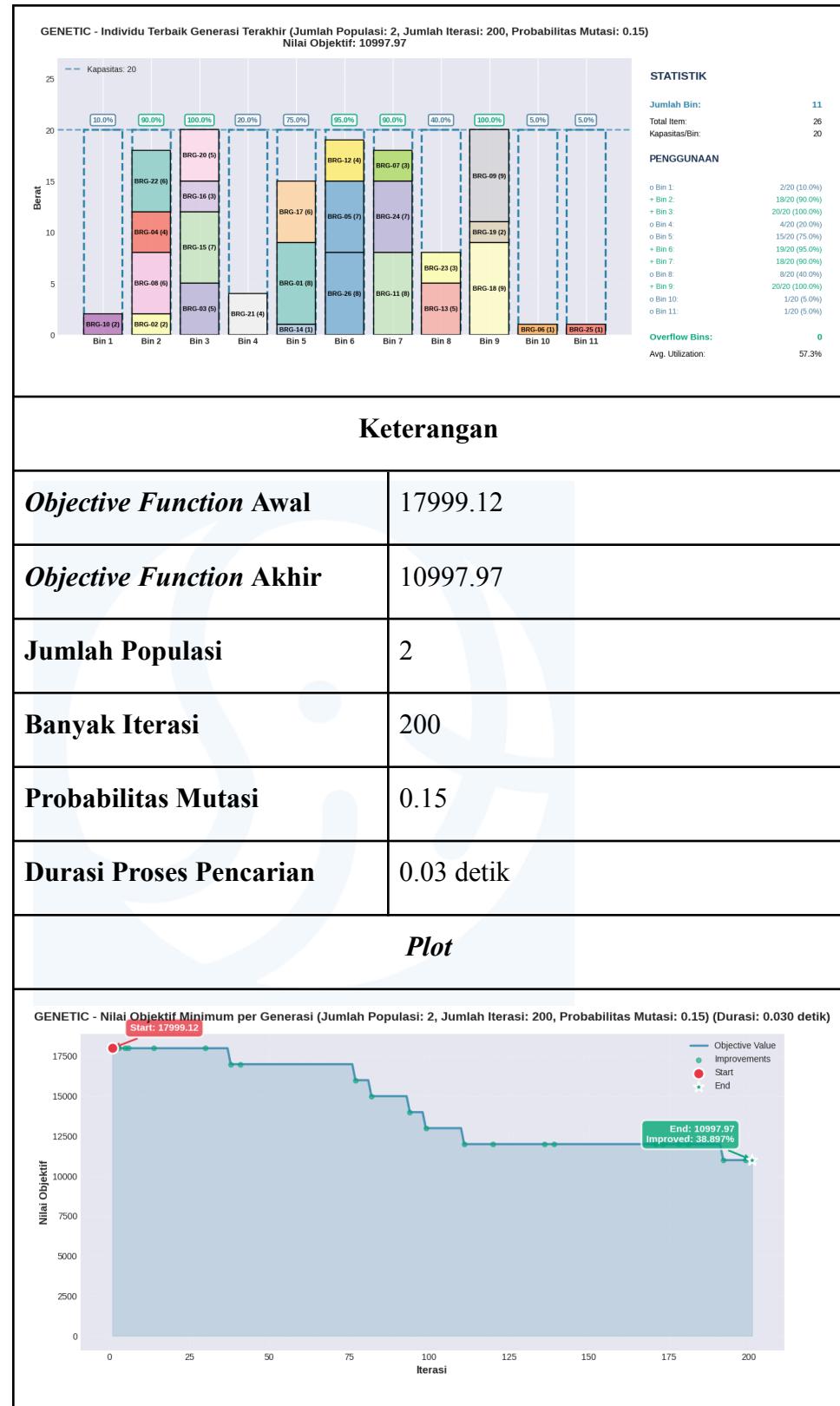
Eksperimen
<b>State Awal Bin Packing Problem</b>



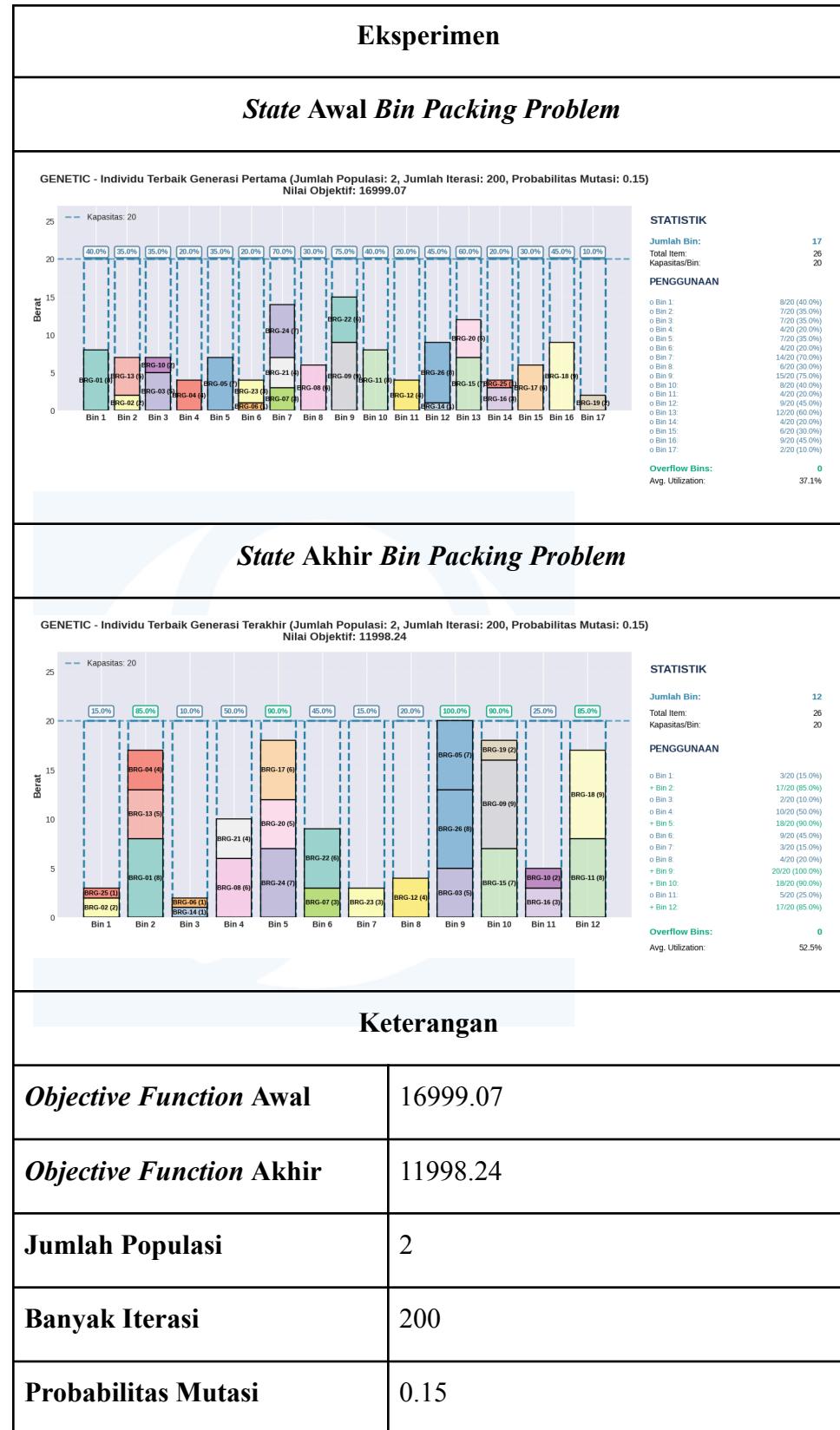


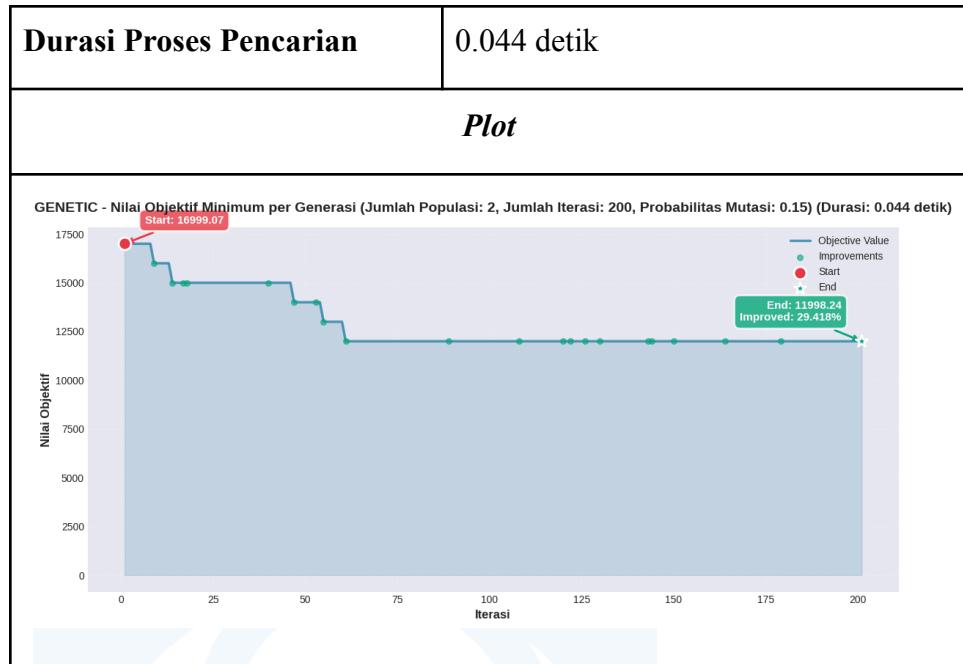
## ii. Percobaan 2





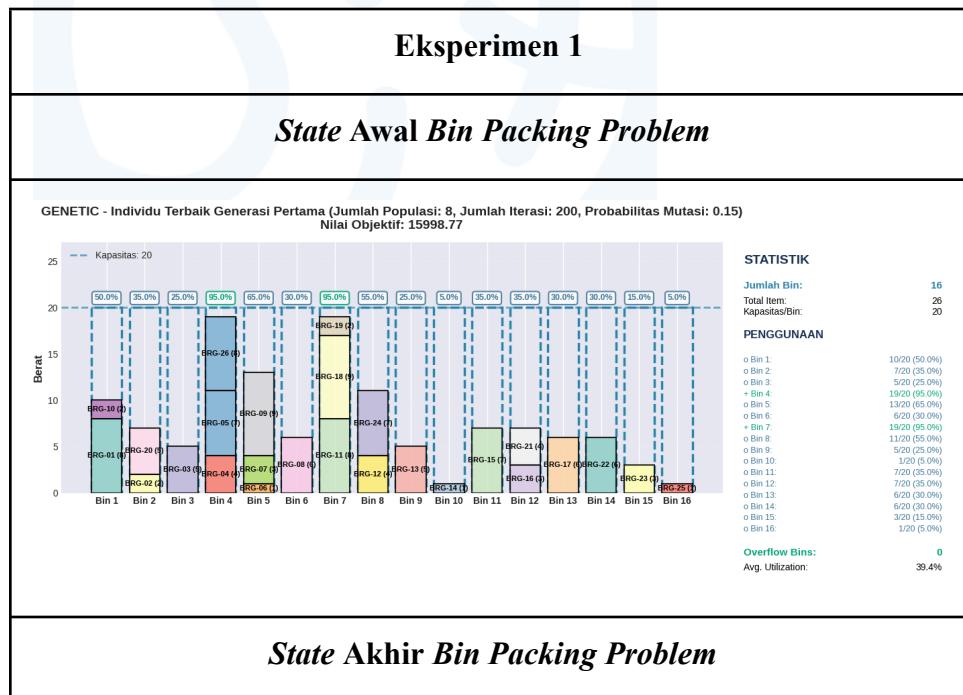
### iii. Percobaan 3

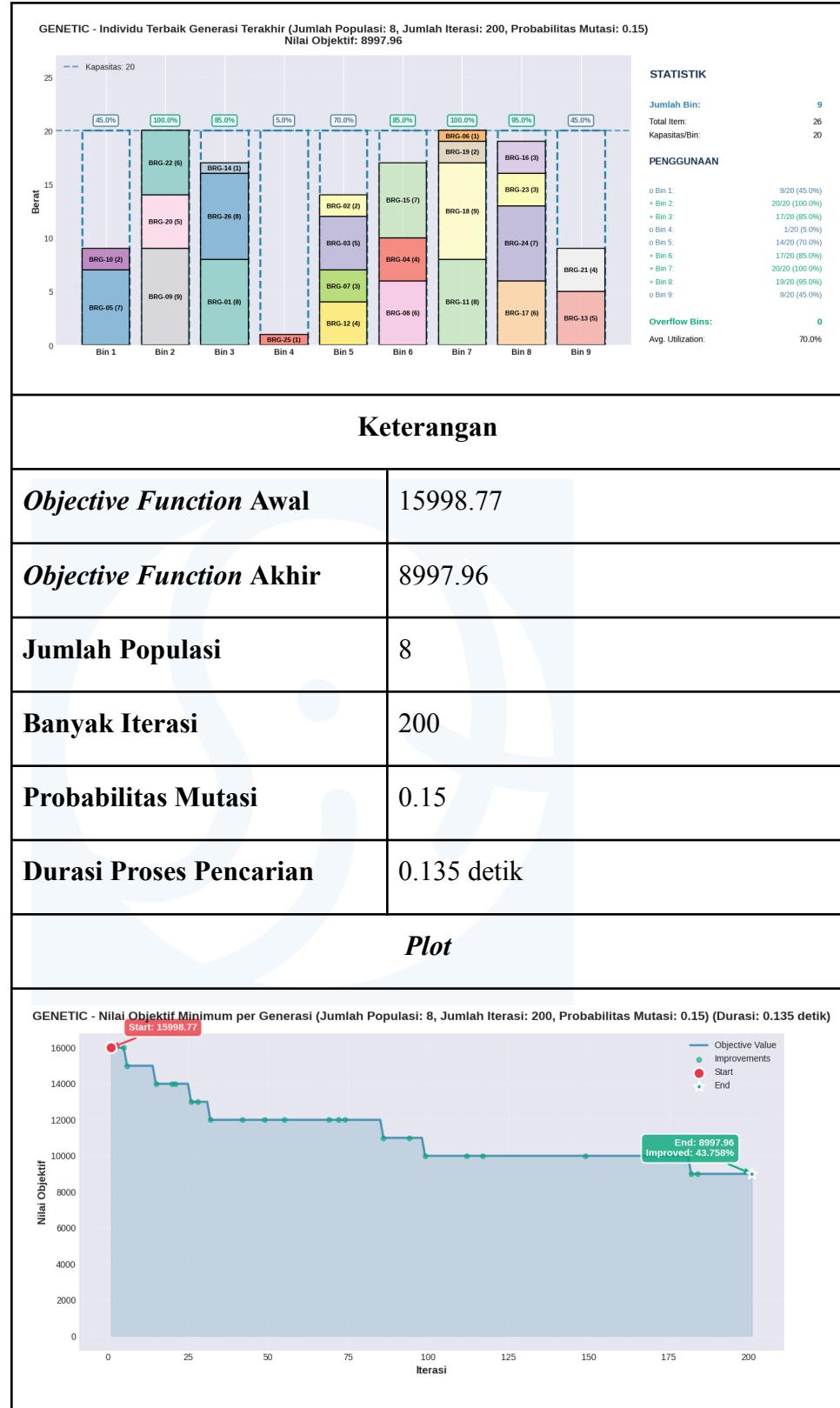




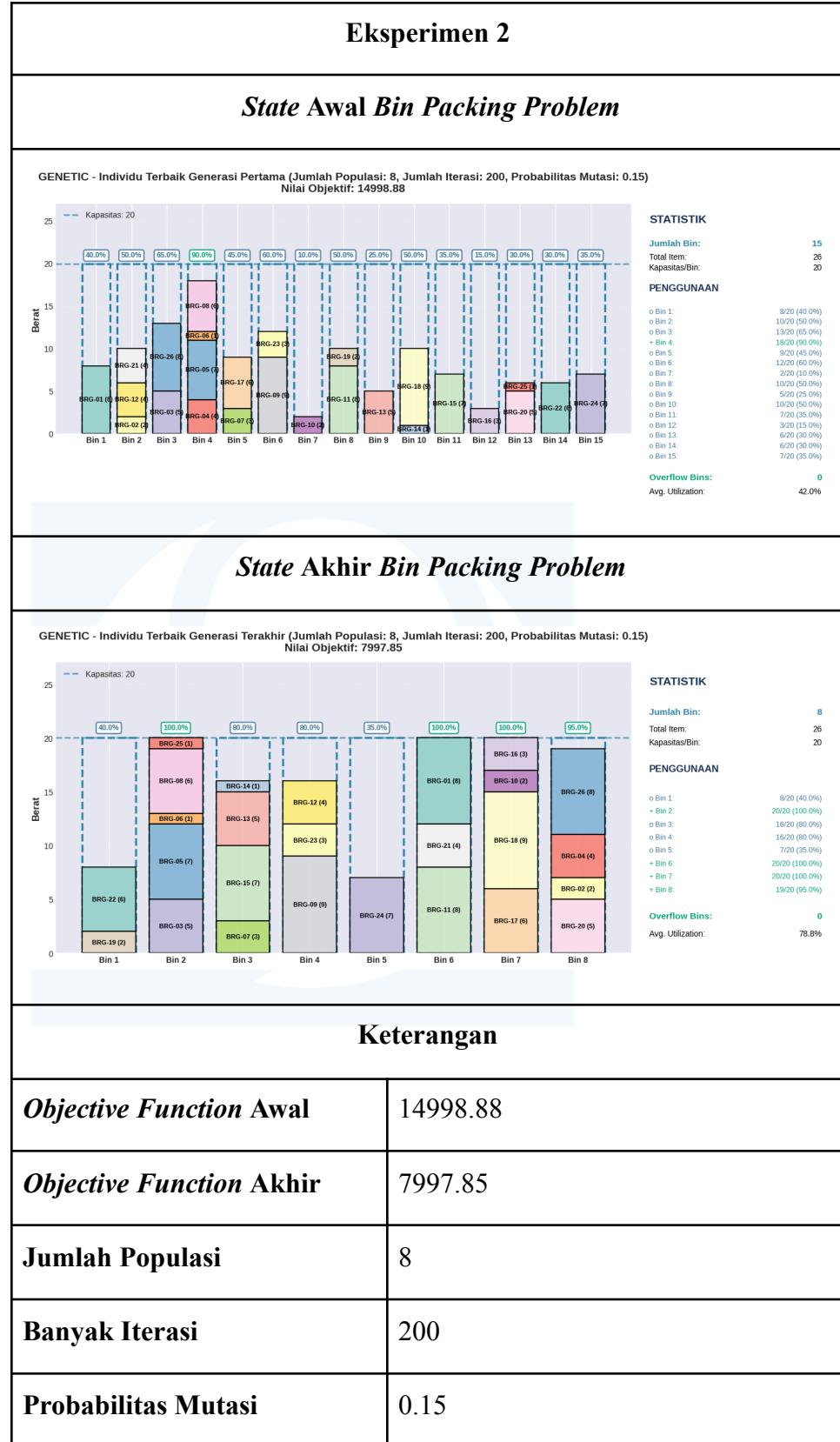
b. Populasi = 8

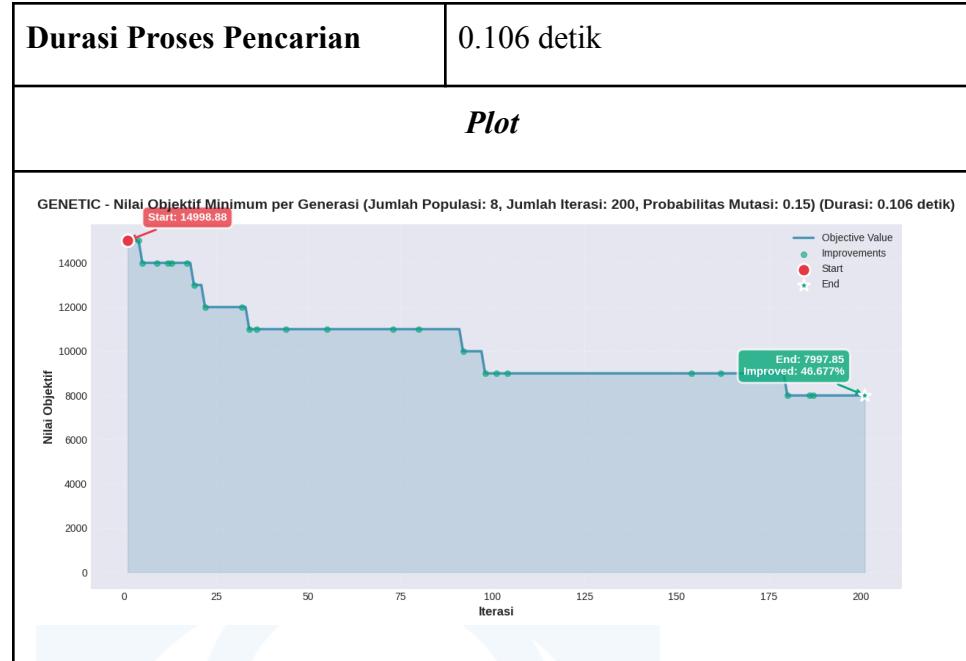
i. Eksperimen 1



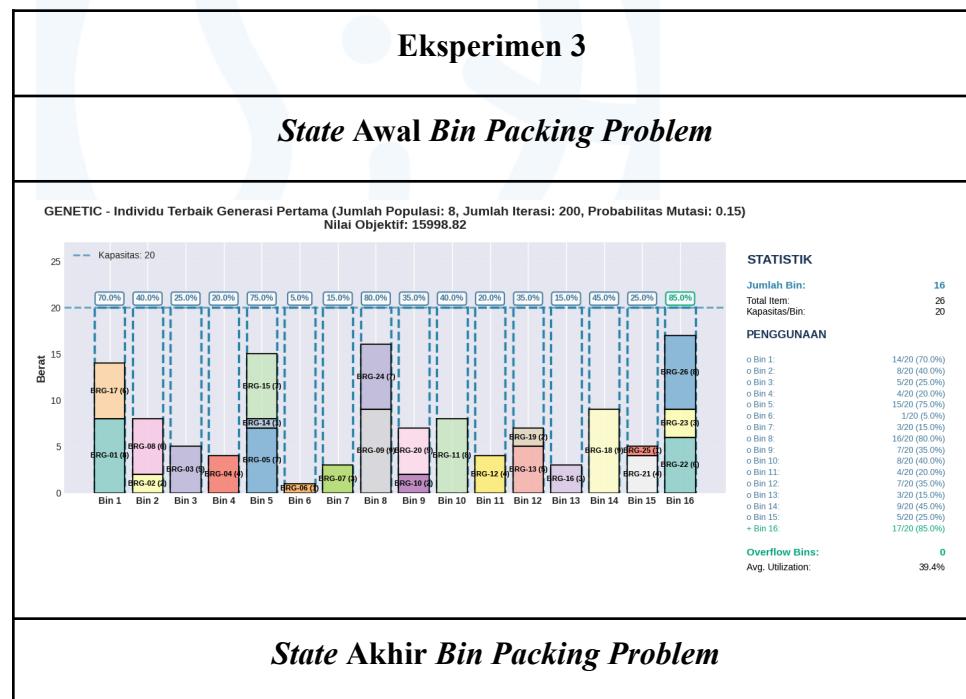


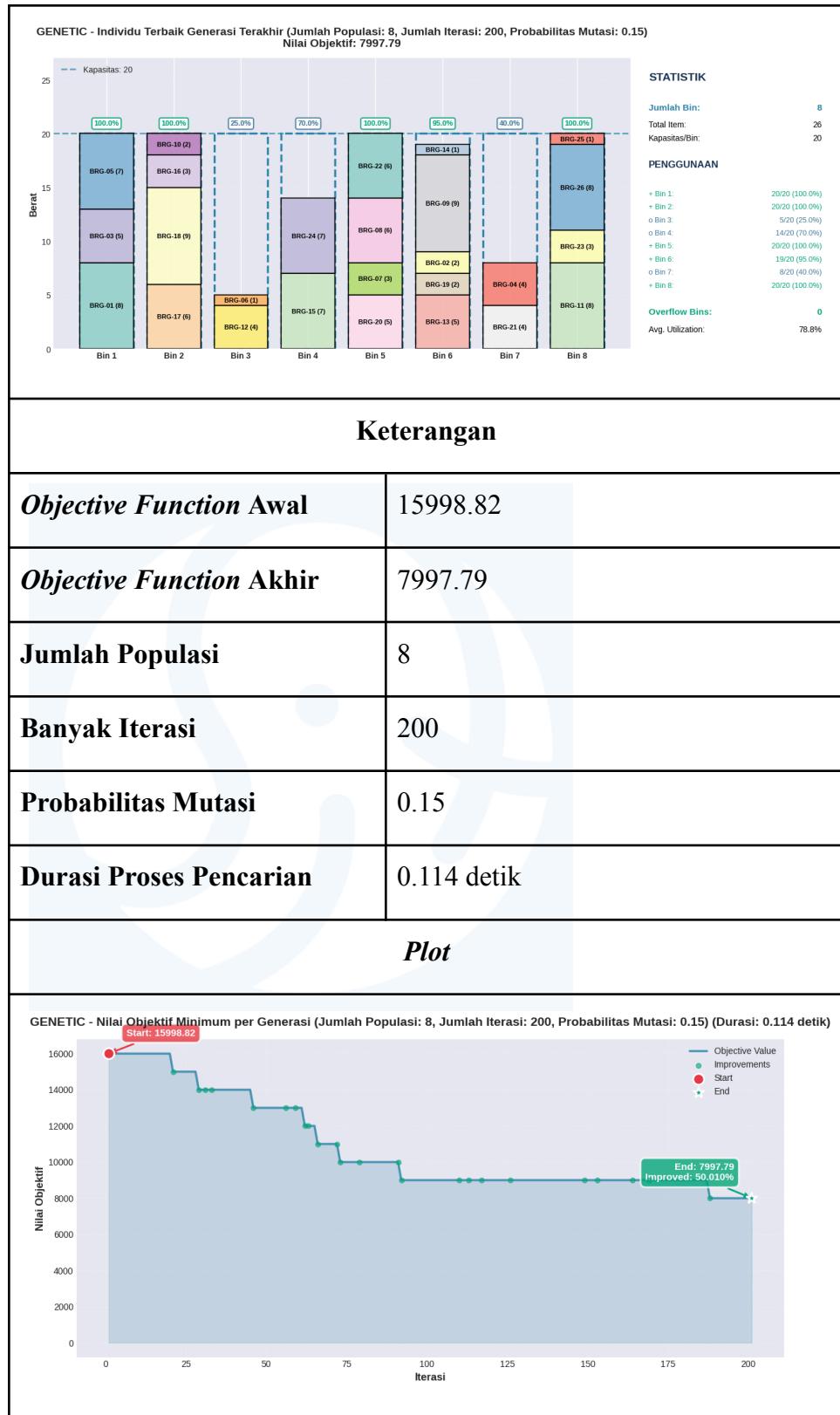
## ii. Eksperimen 2





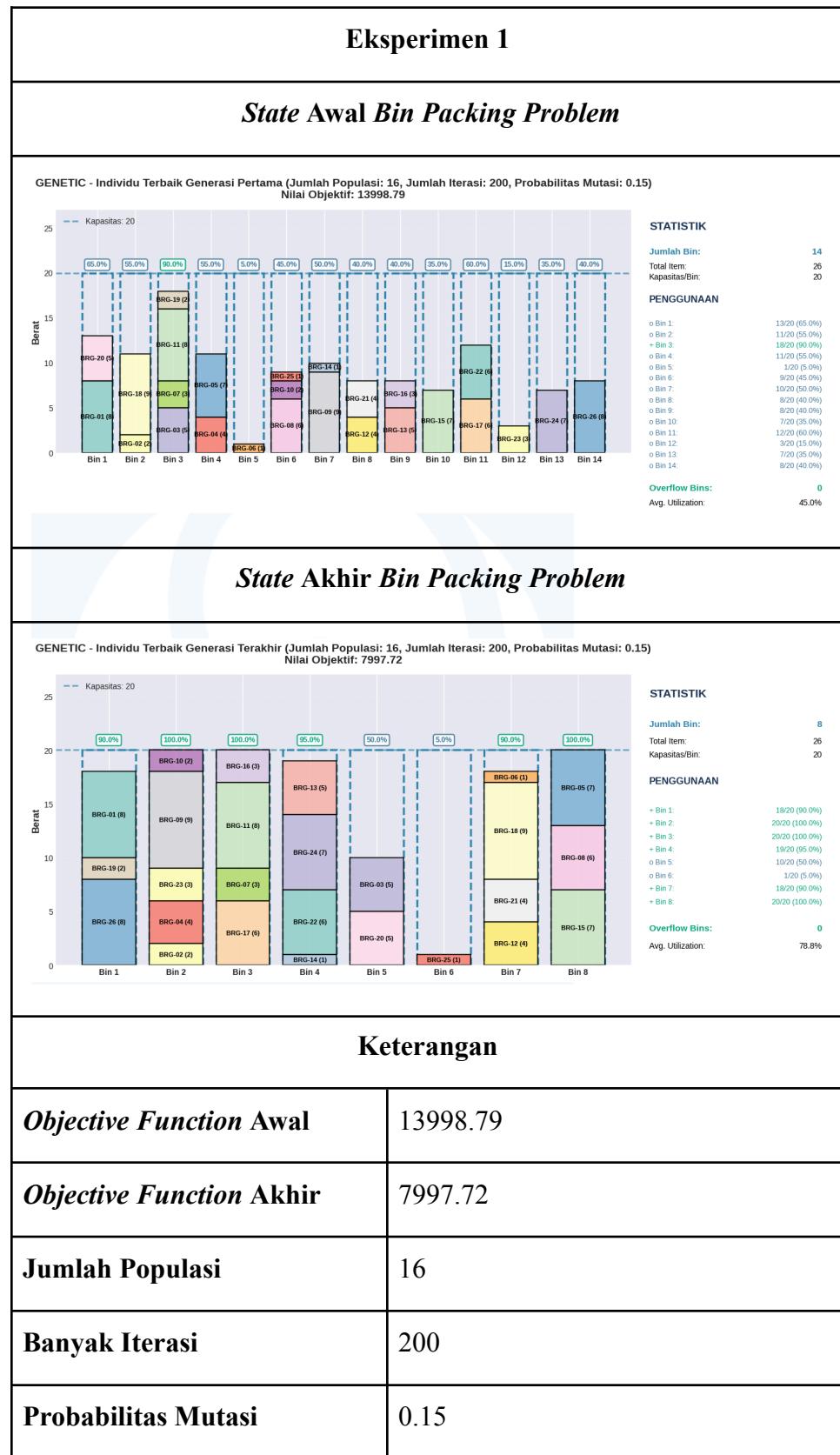
### iii. Eksperimen 3

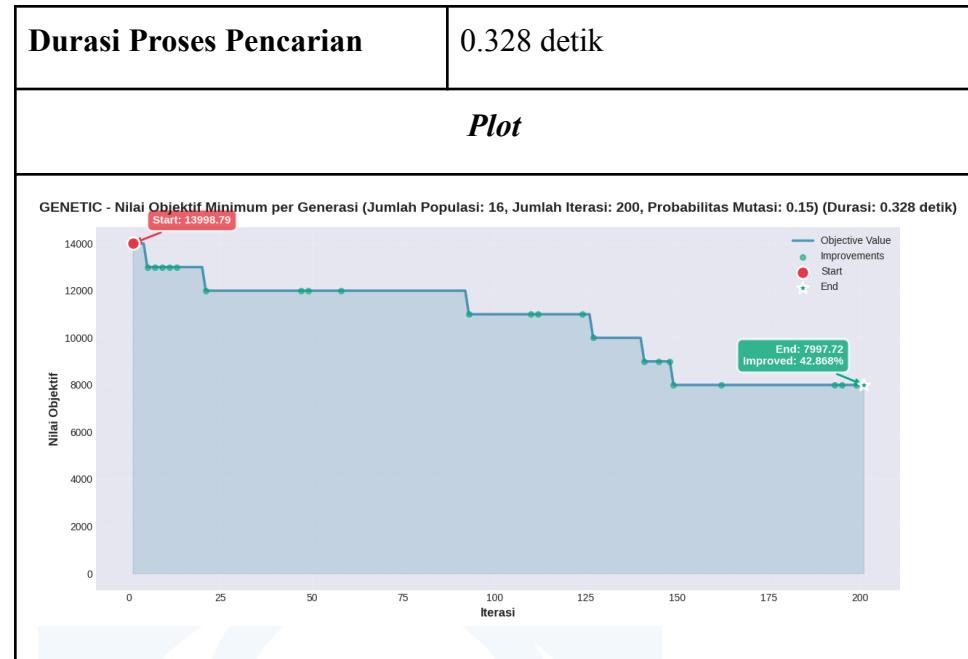




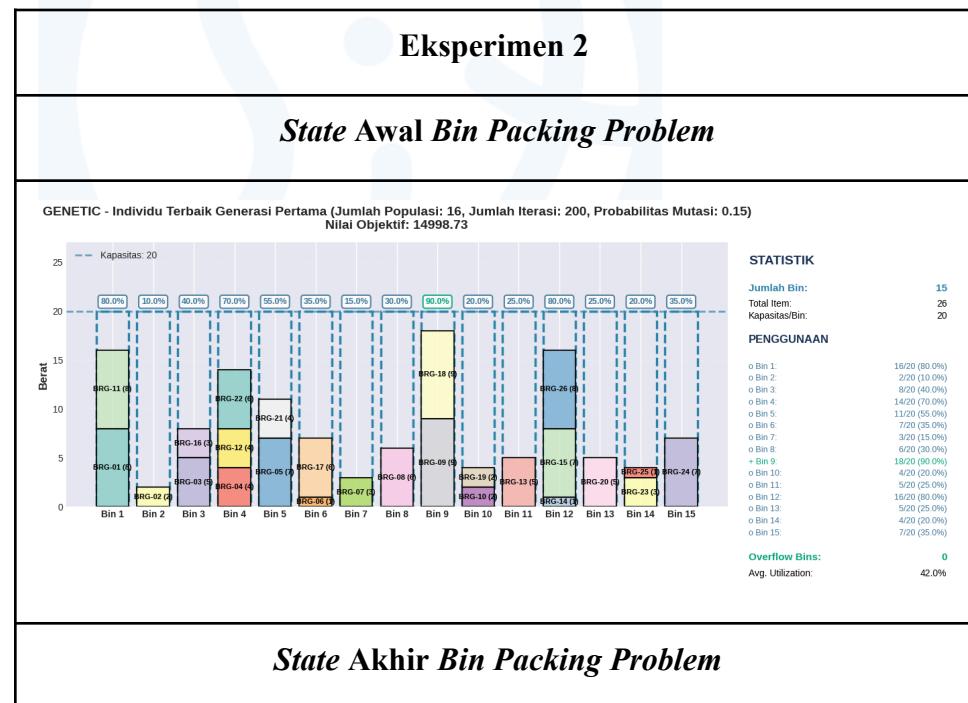
c. Populasi = 16

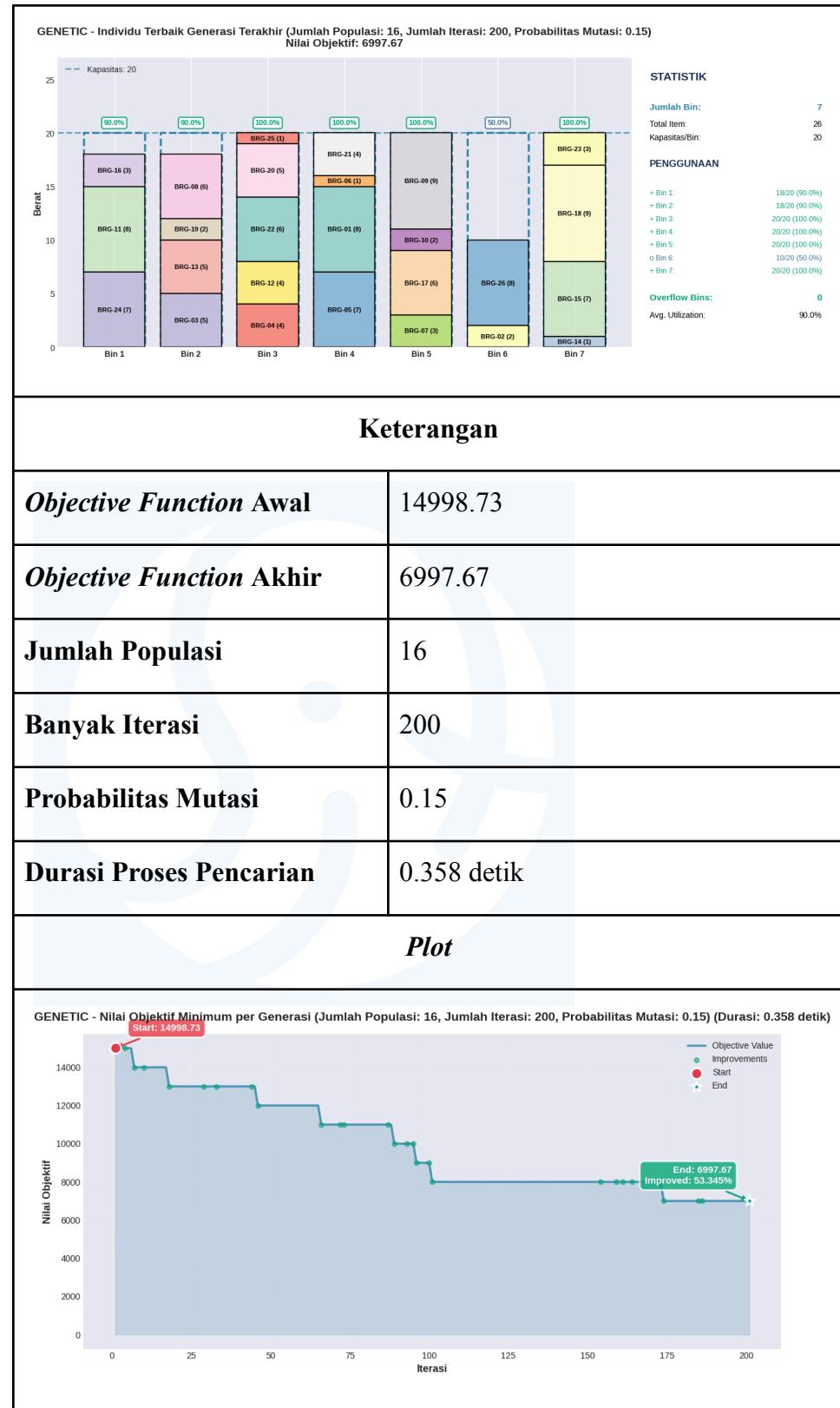
i. Eksperimen 1



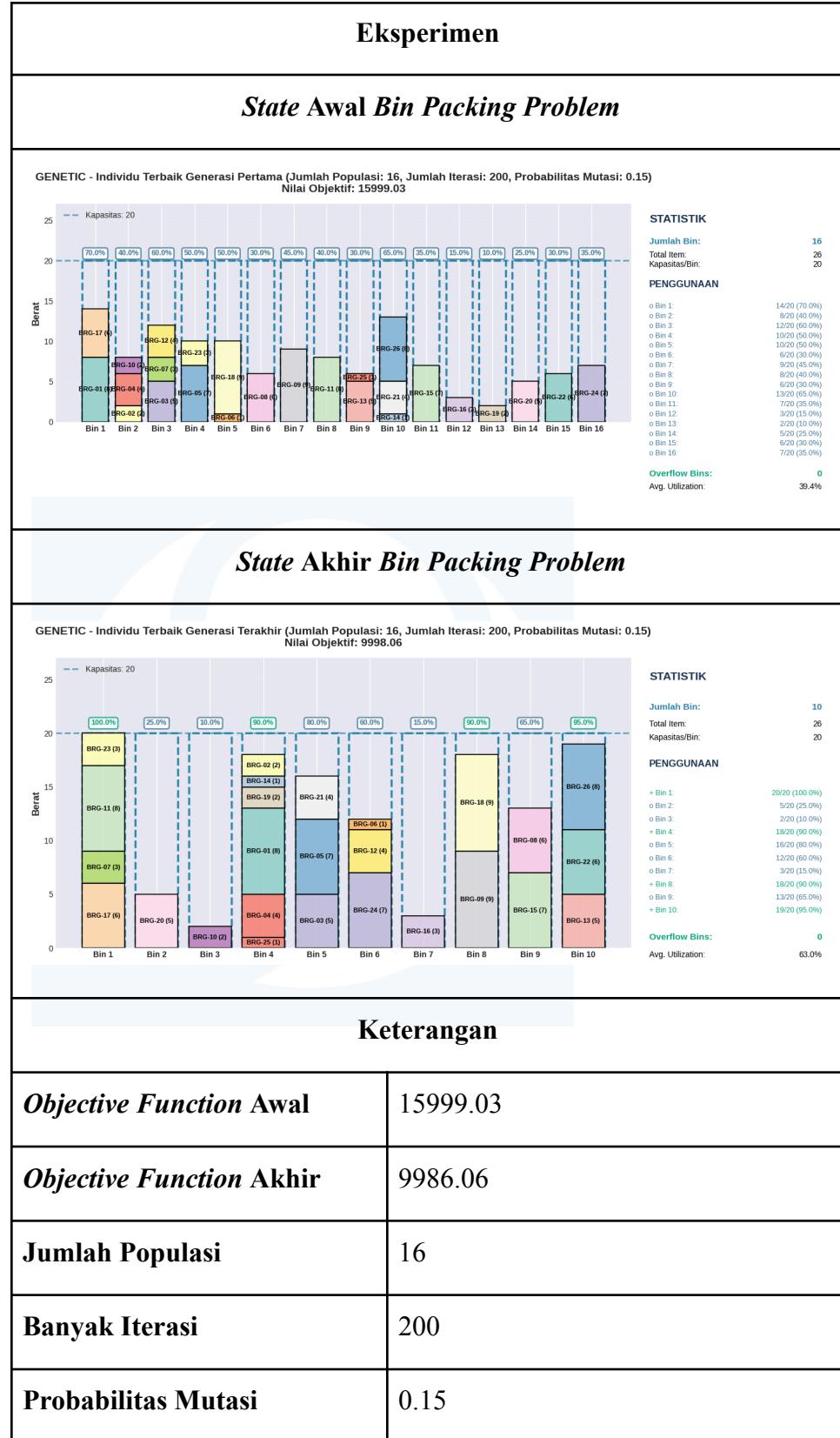


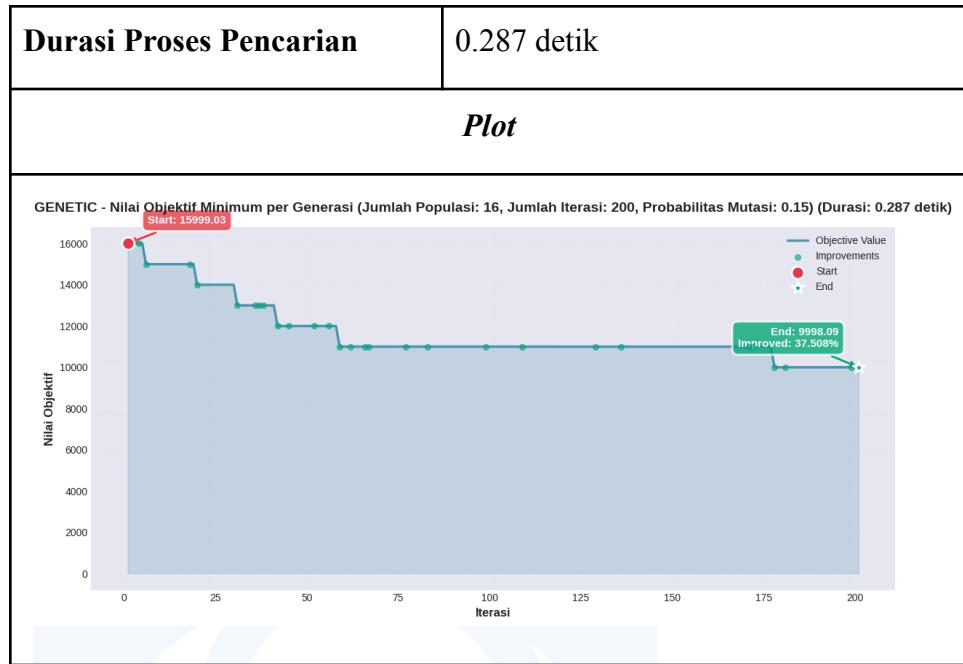
## ii. Eksperimen 2





### iii. Eksperimen 3

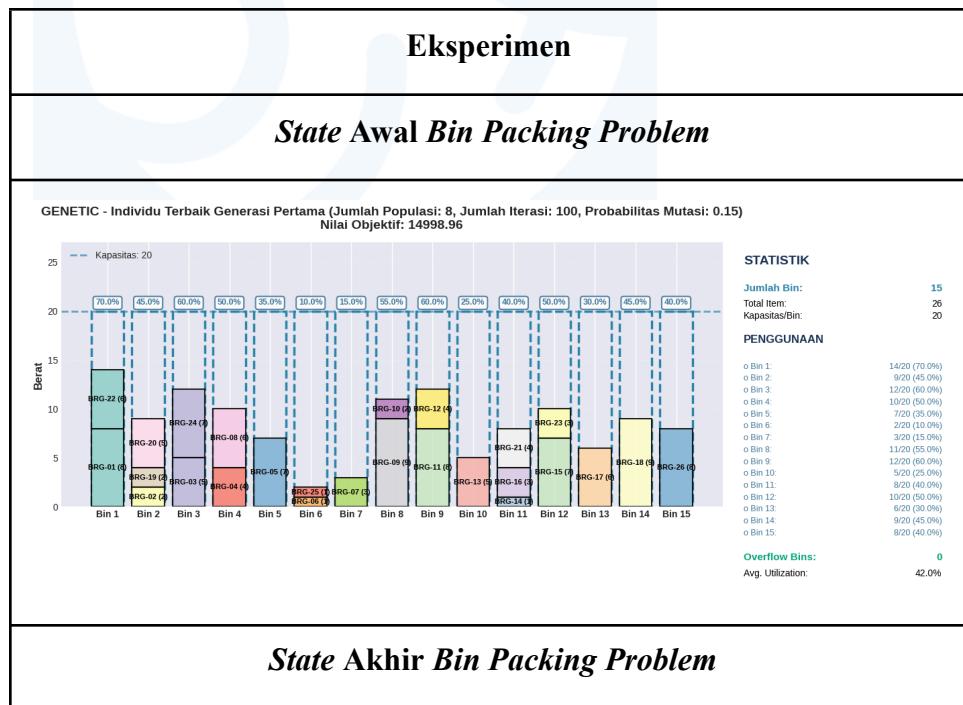


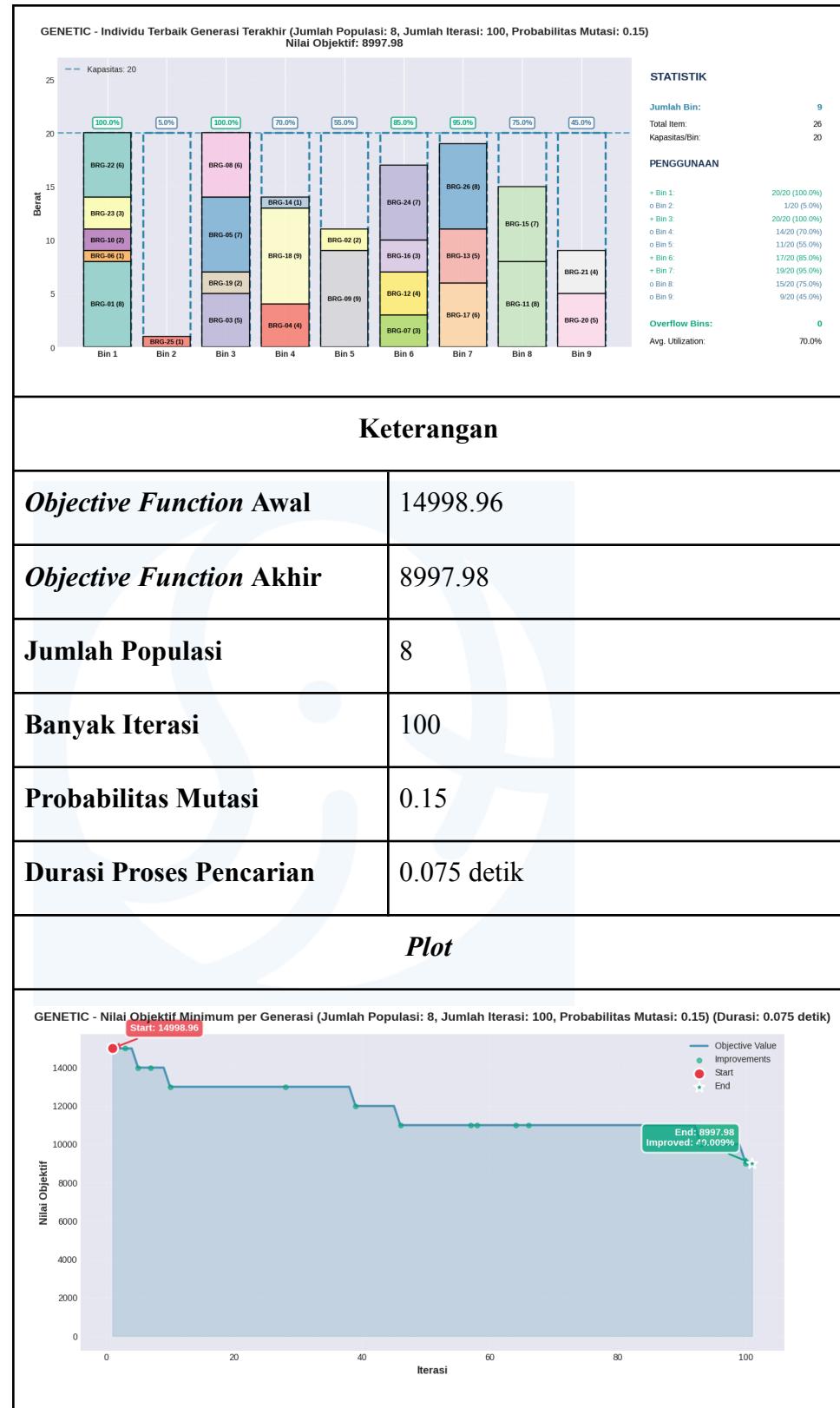


## 2. Variasi Jumlah Maksimal Iterasi (Probabilitas Mutasi = 0.15)

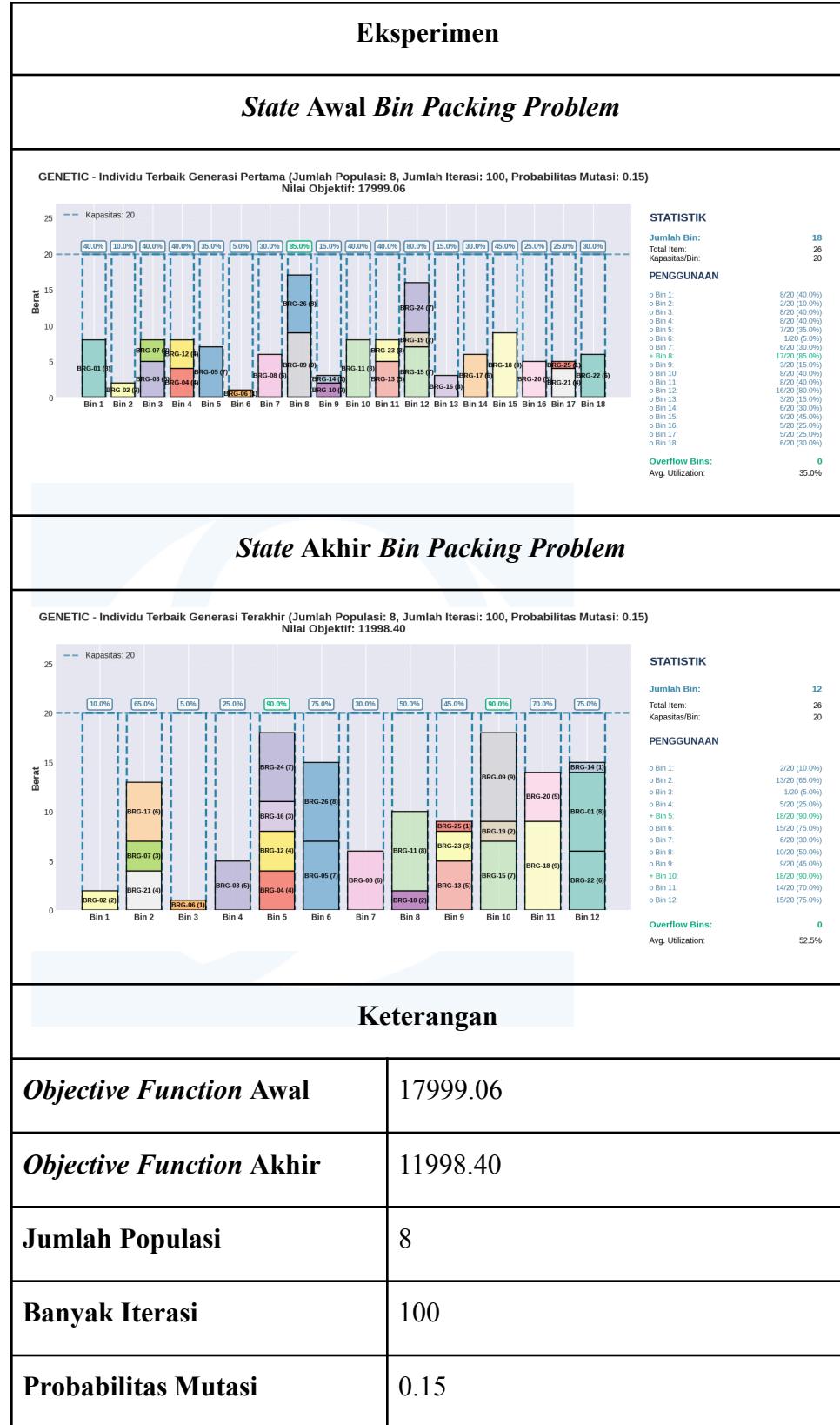
a. Iterasi = 100

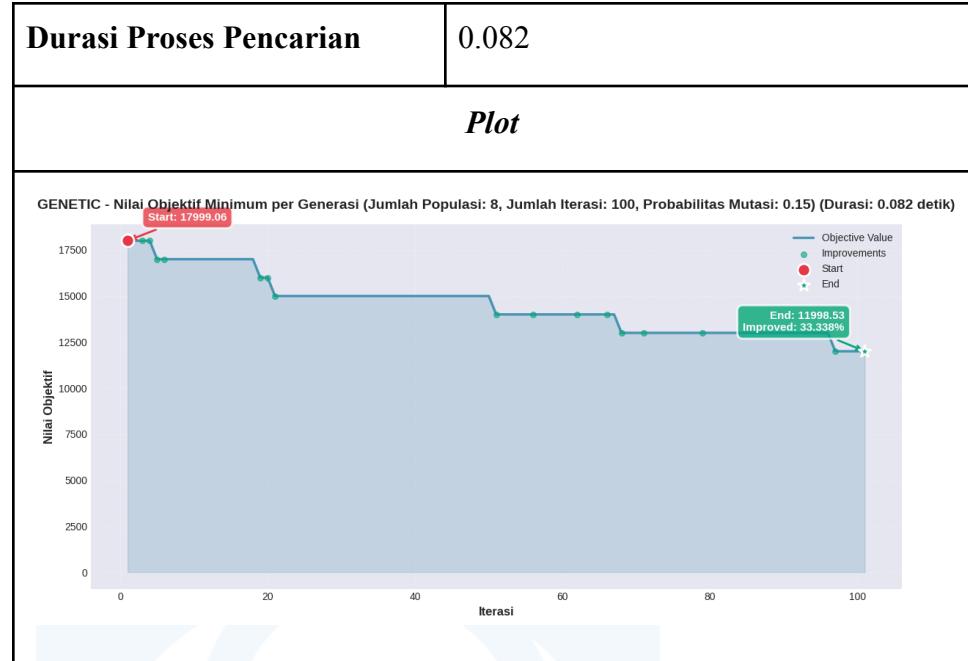
i. Eksperimen 1



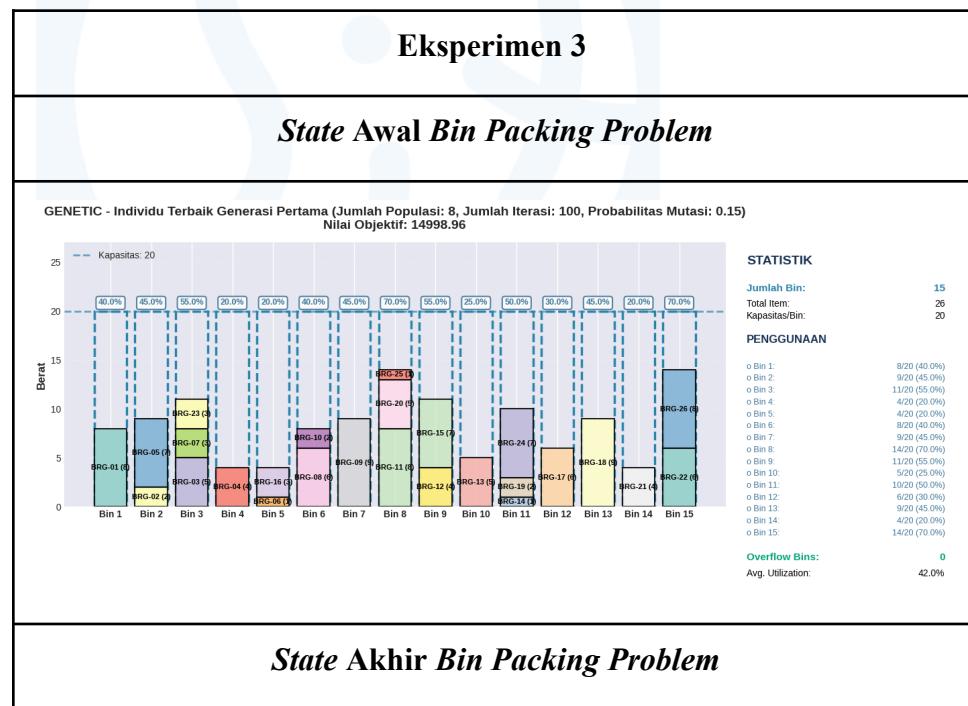


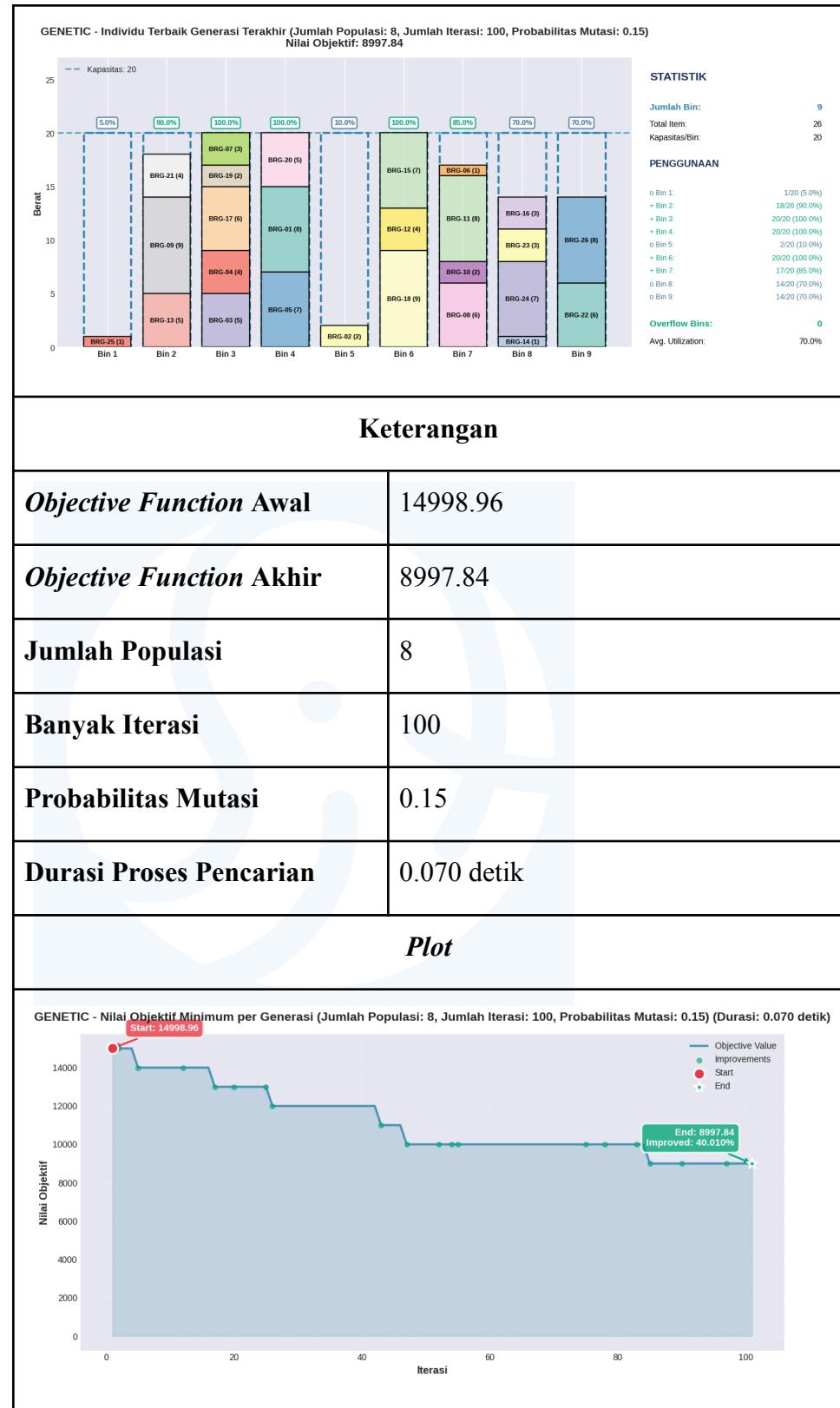
## ii. Eksperimen 2





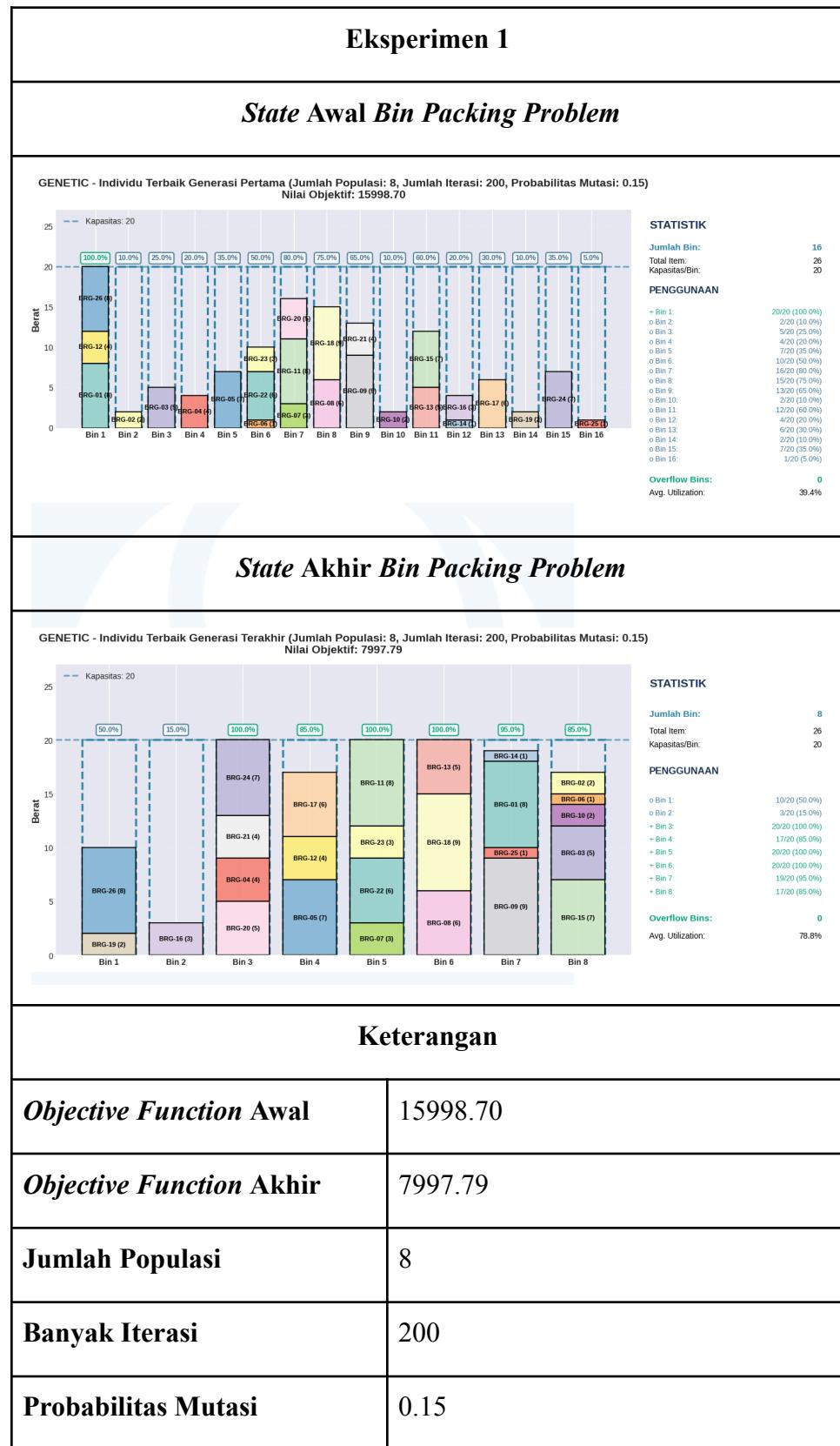
### iii. Eksperimen 3

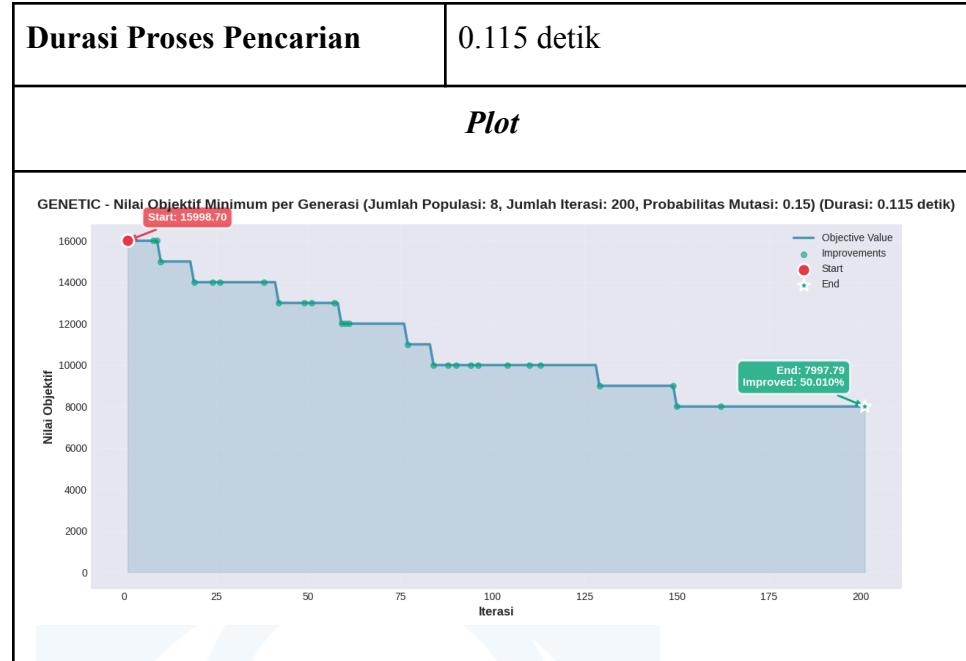




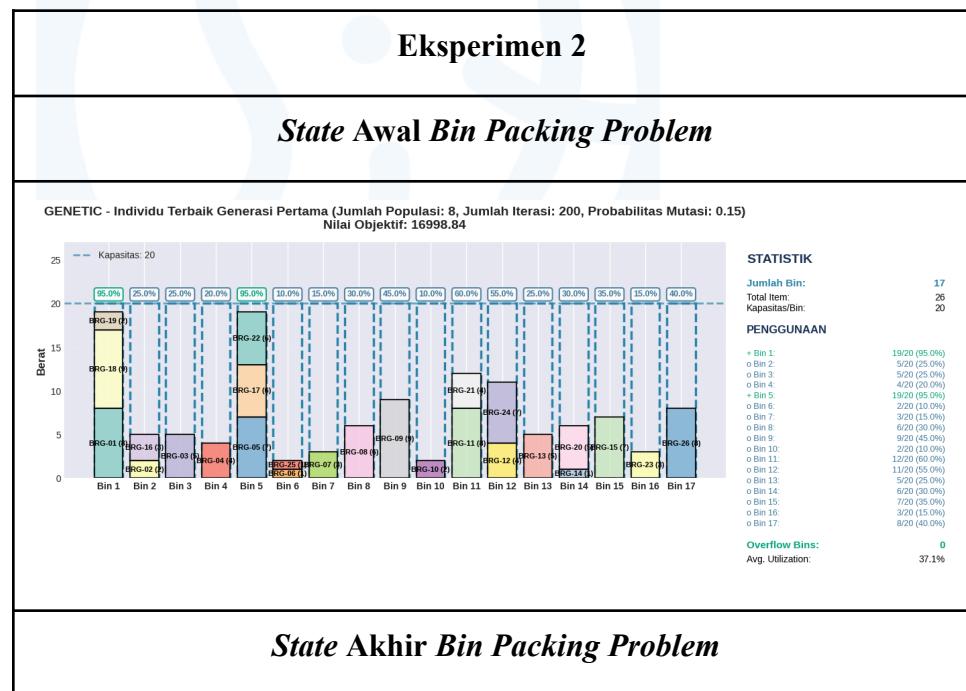
b. Iterasi = 200

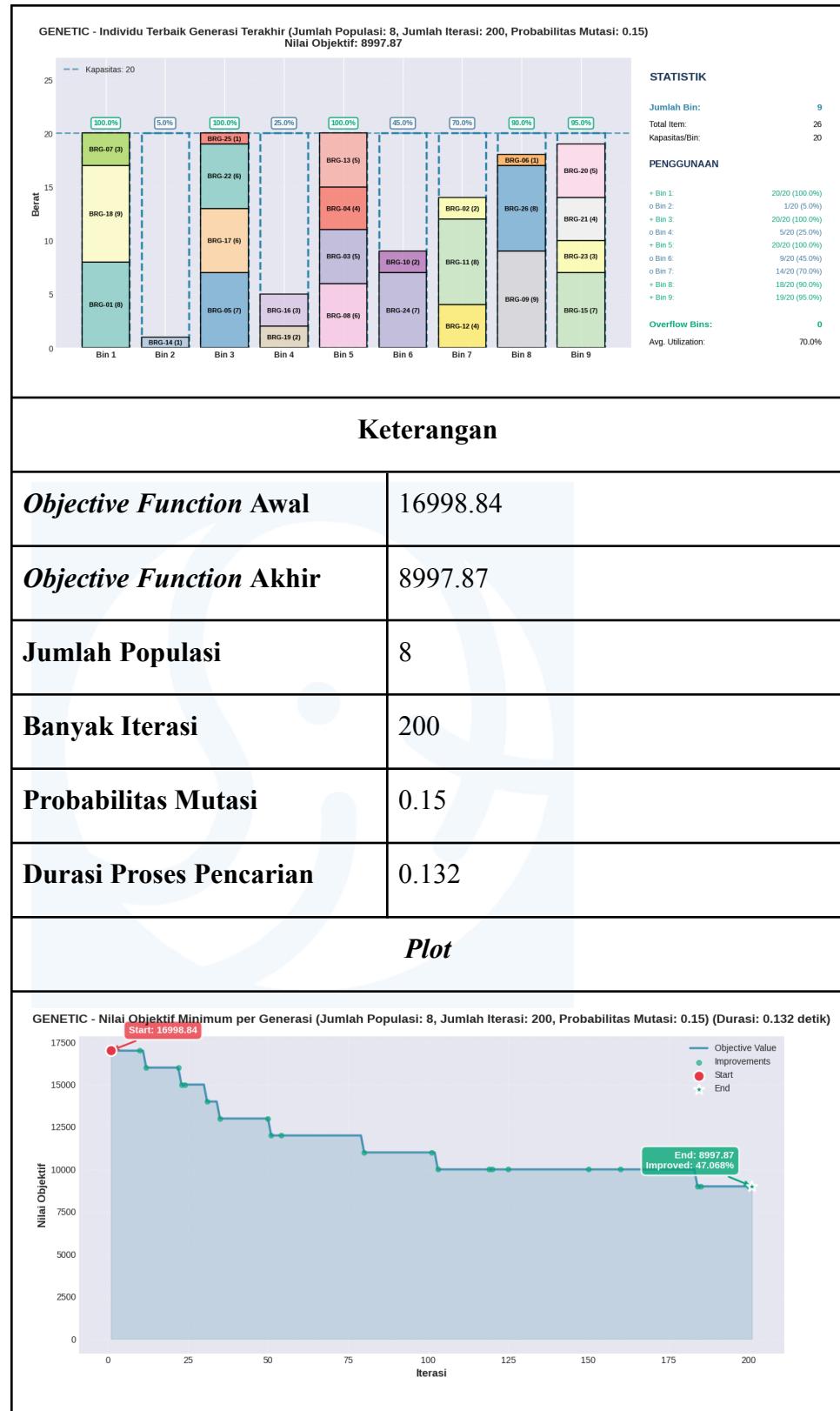
i. Eksperimen 1



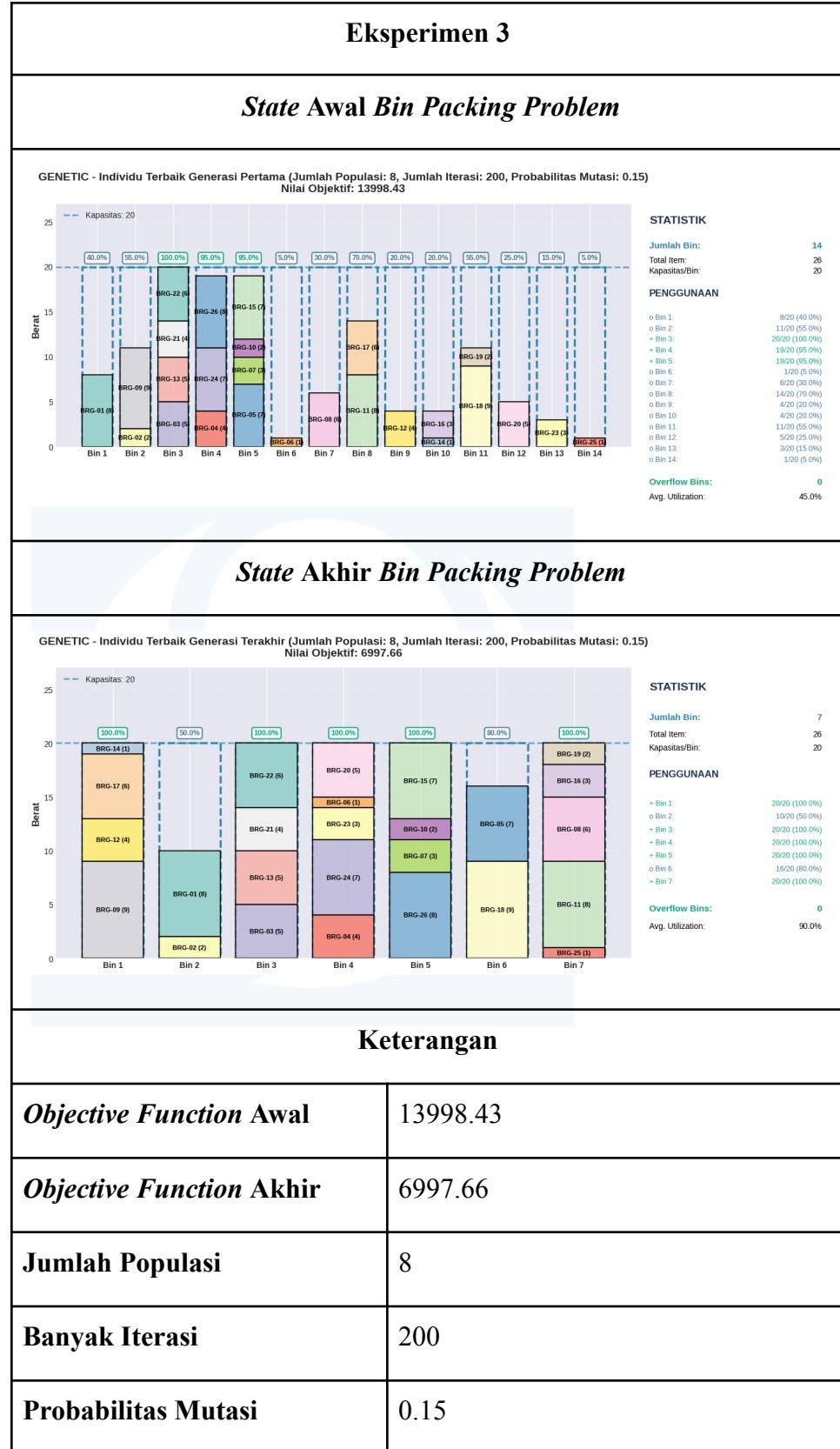


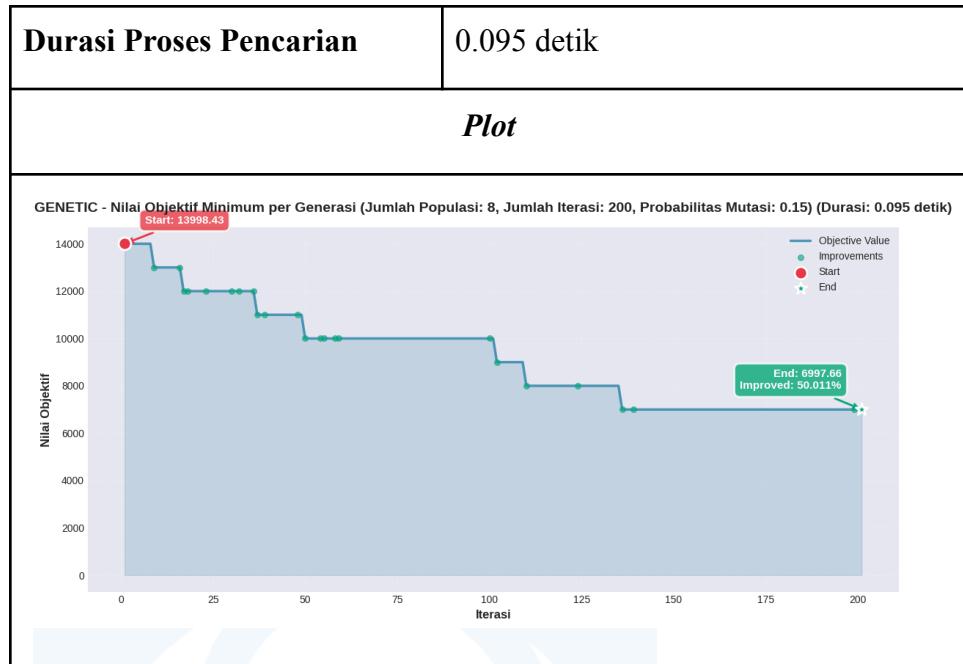
## ii. Eksperimen 2





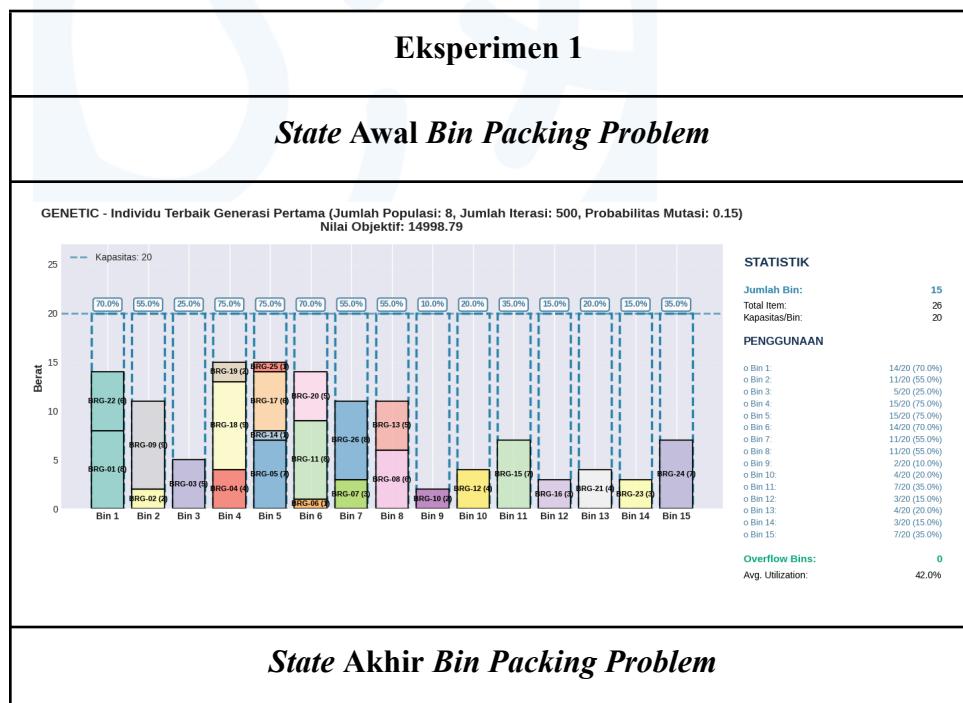
### iii. Eksperimen 3

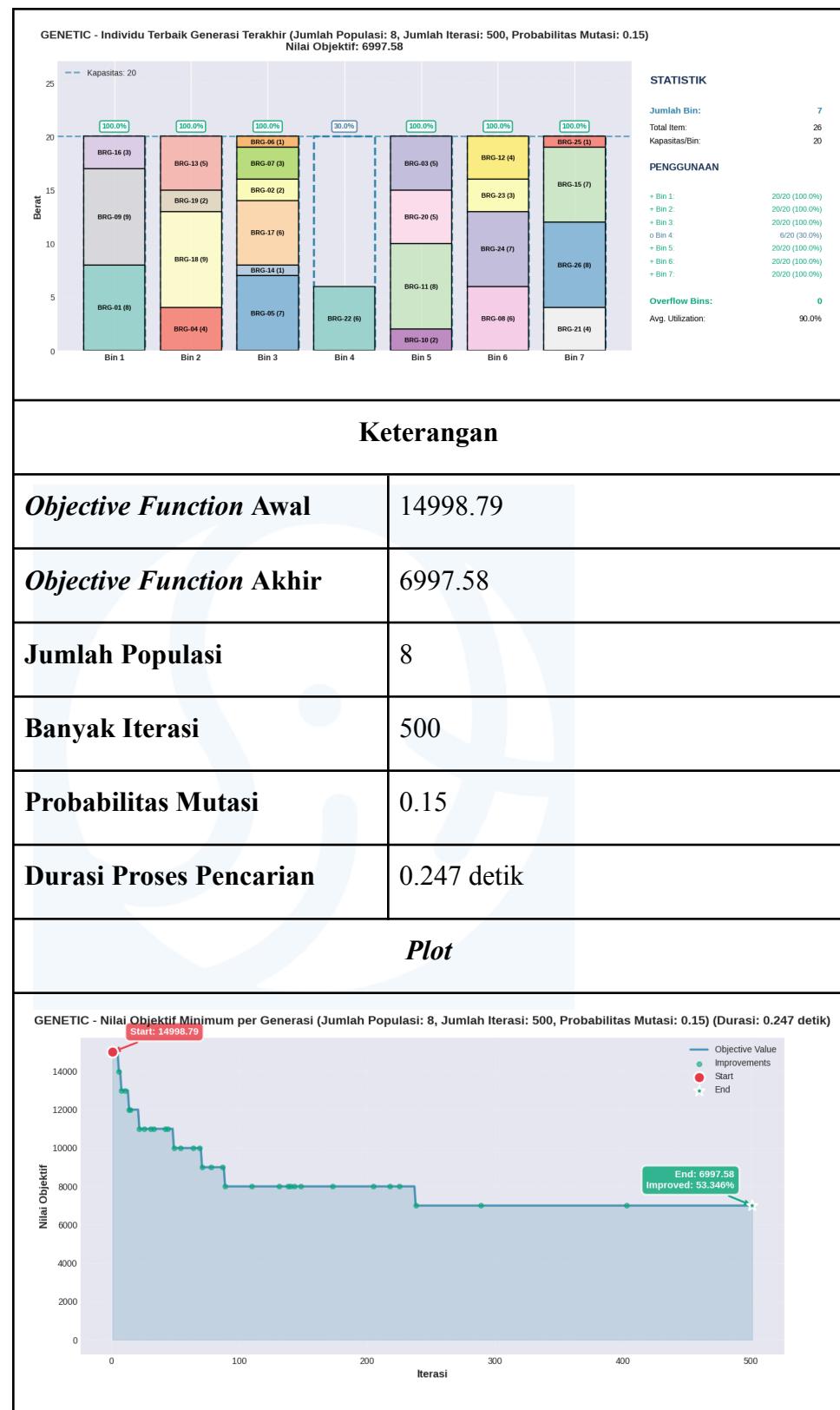




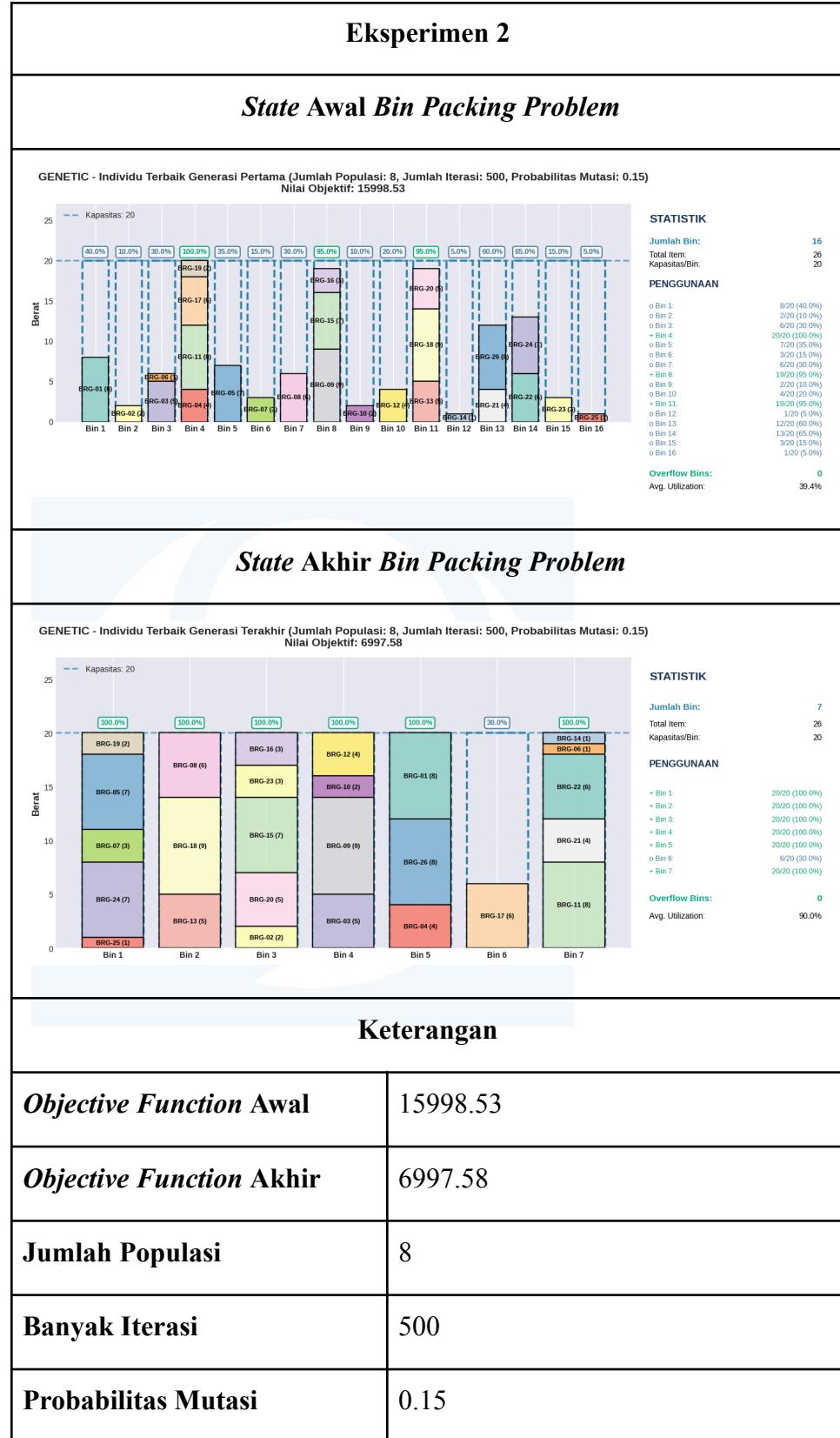
c. Iterasi = 500

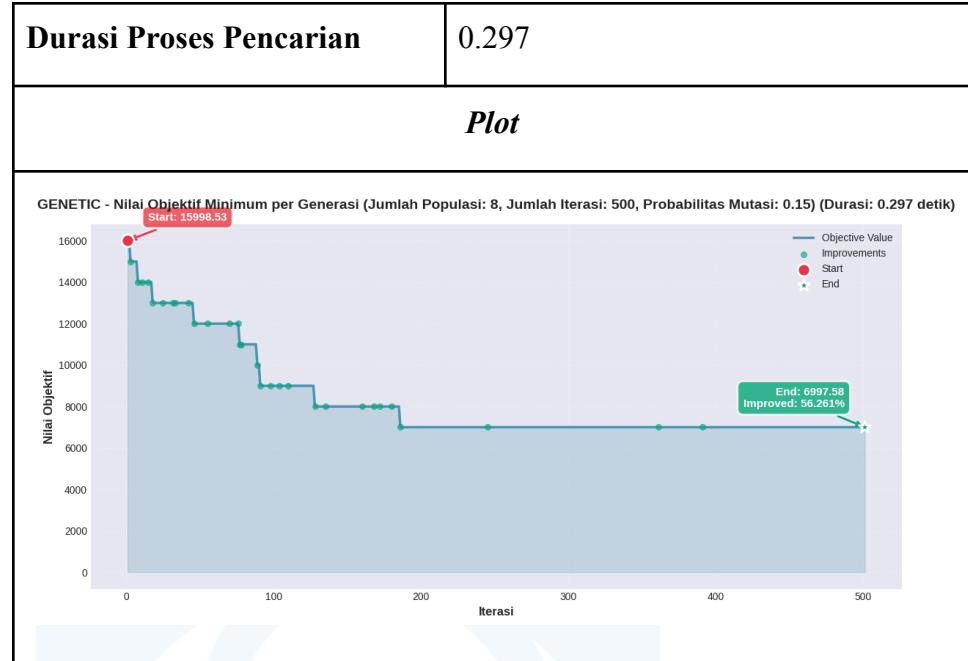
i. Eksperimen 1



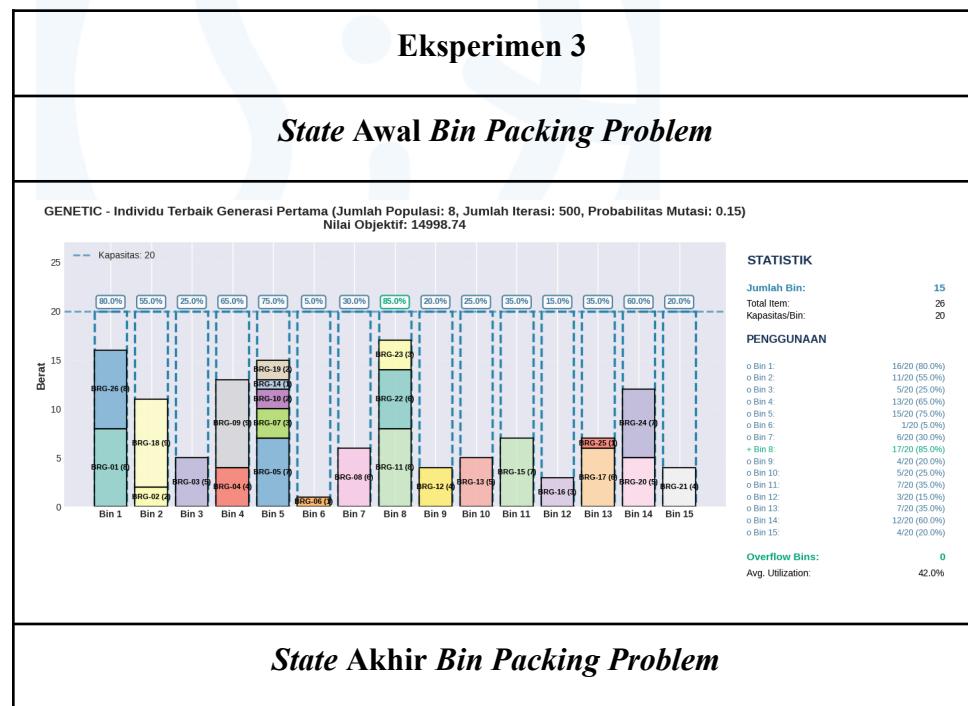


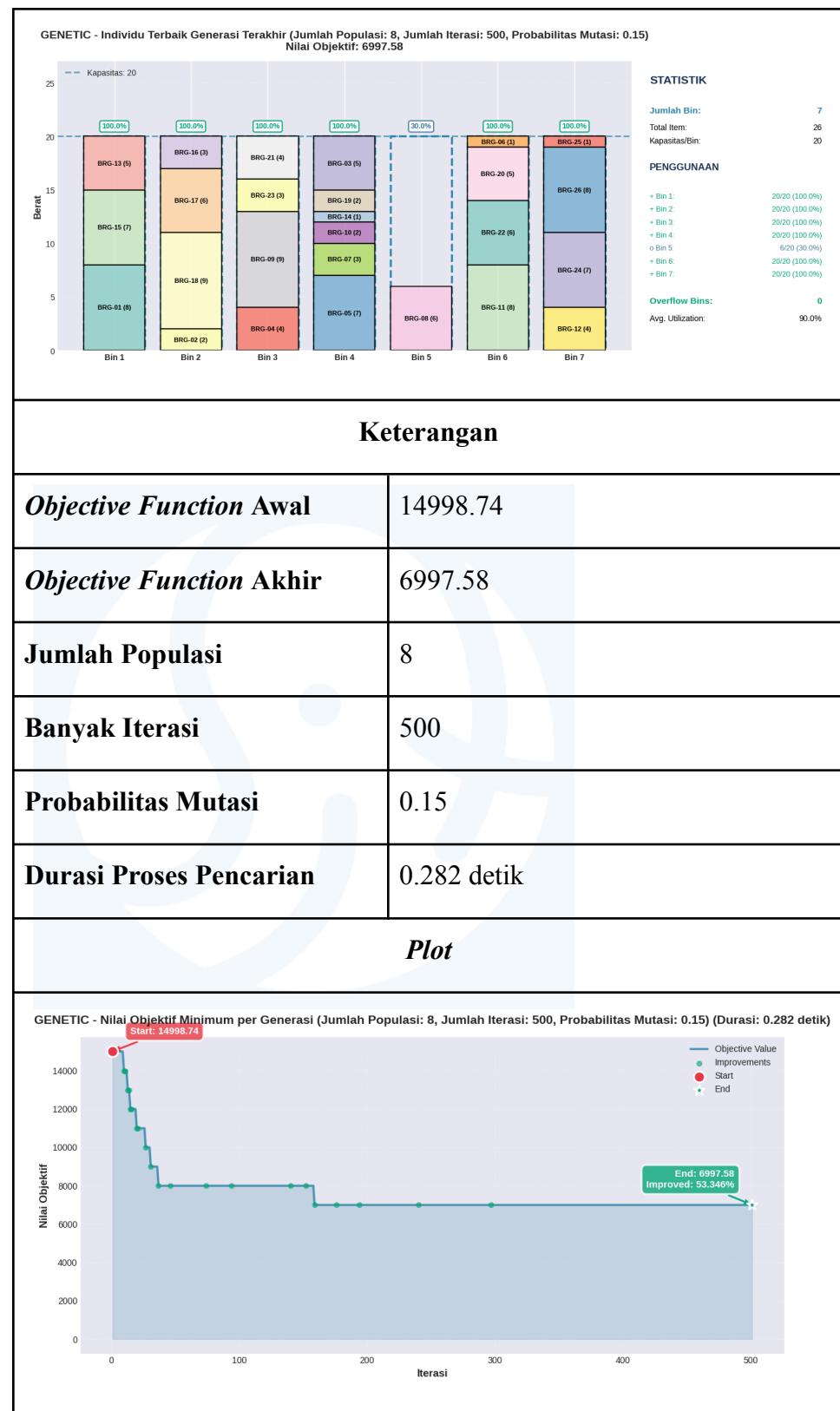
## ii. Eksperimen 2





### iii. Eksperimen 3

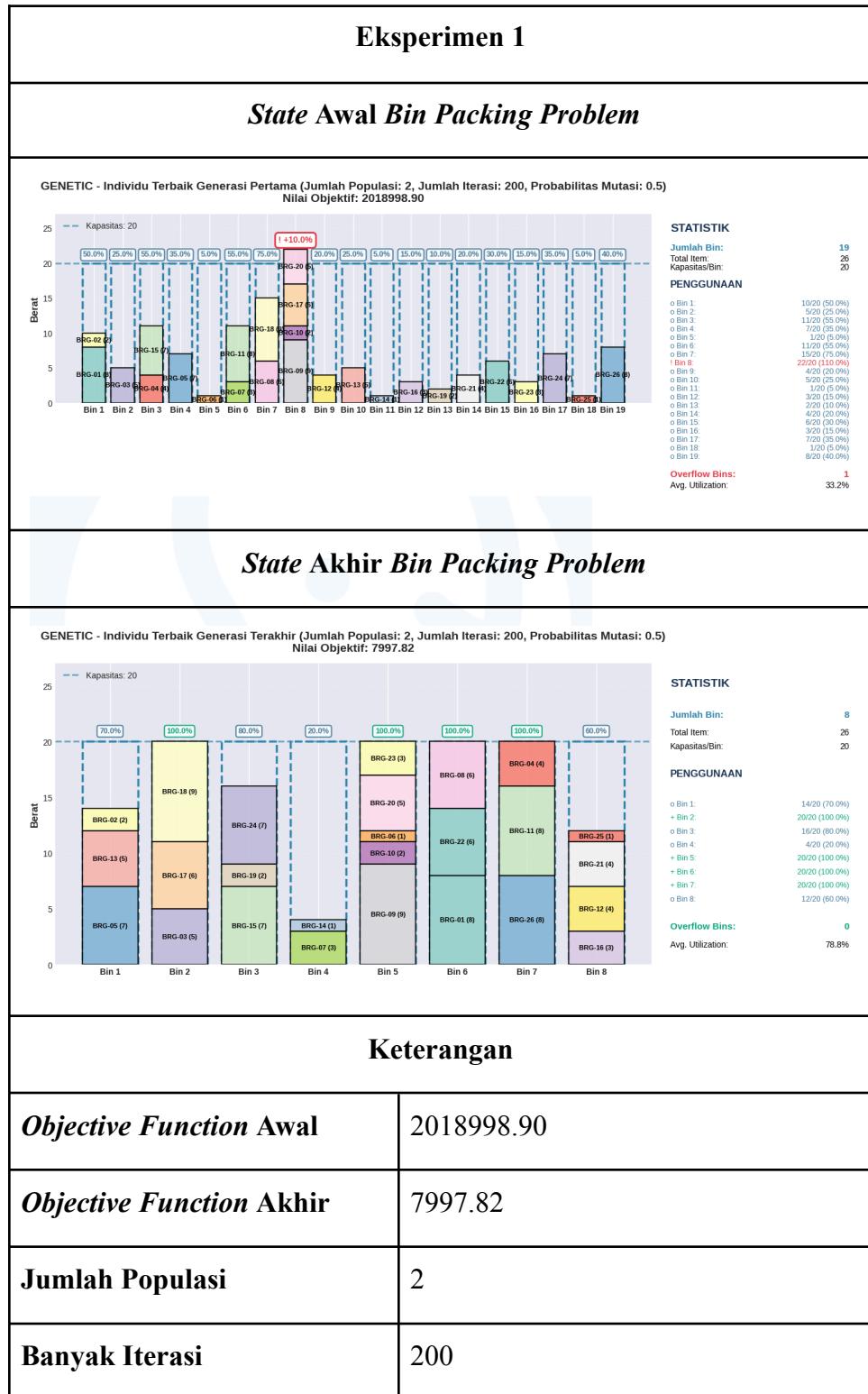


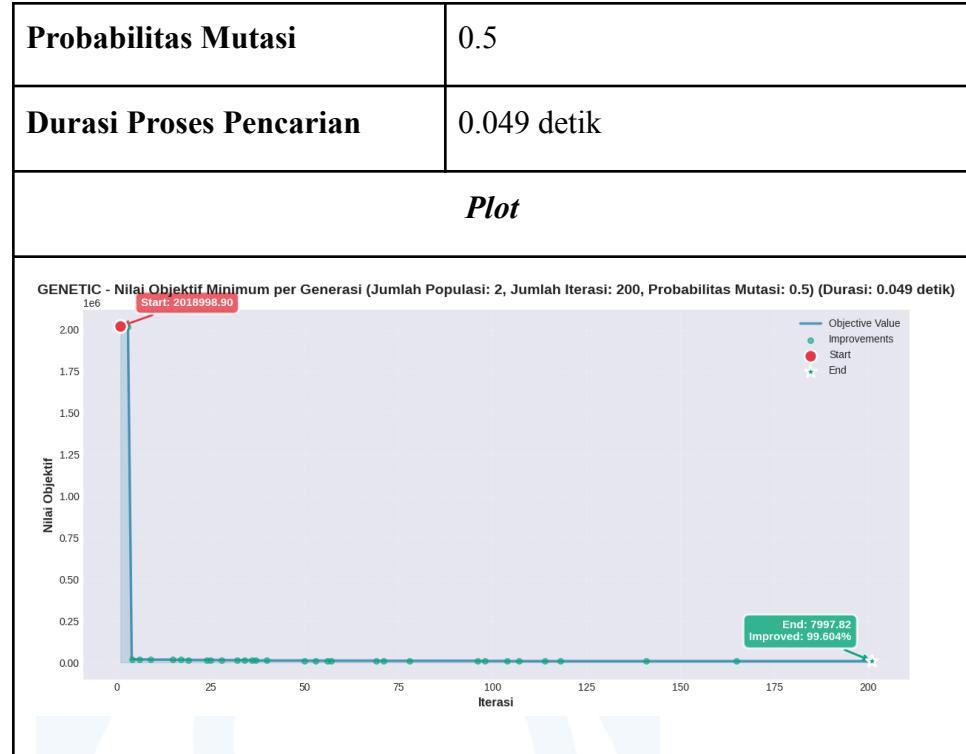


### 3. Variasi Jumlah Populasi (Probabilitas Mutasi = 0.5)

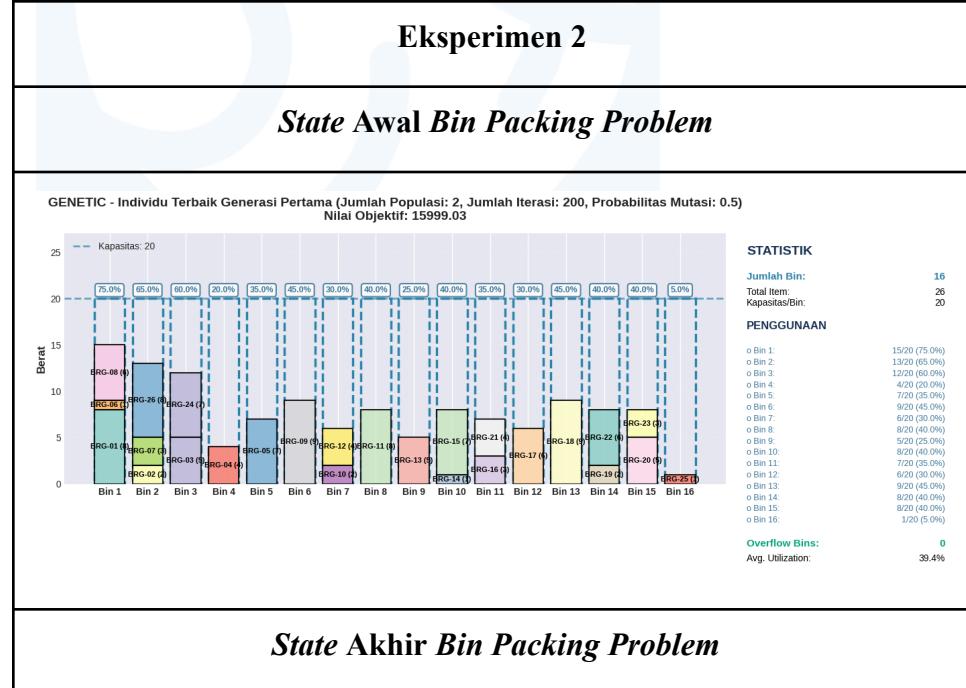
#### a. Jumlah Populasi = 2

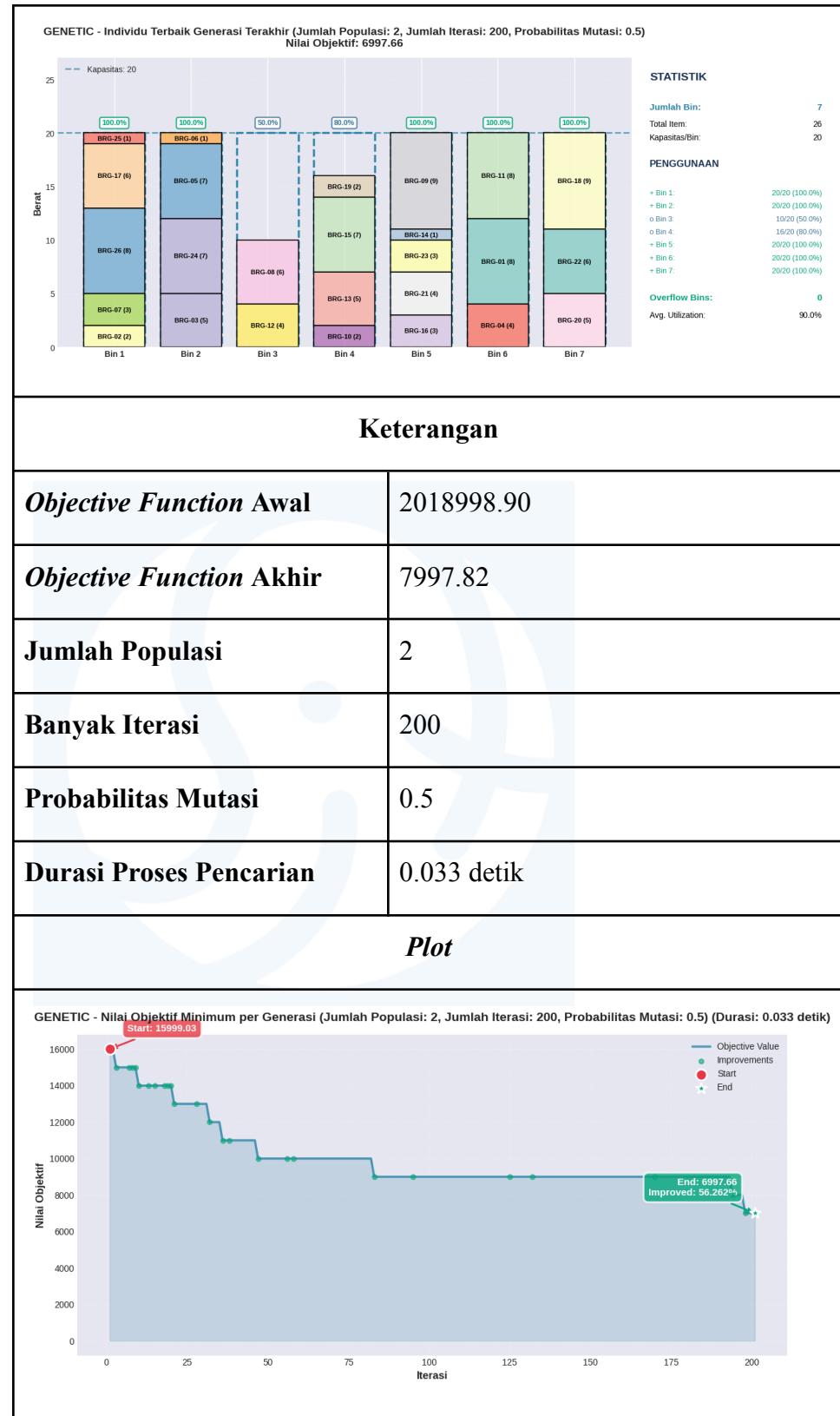
##### i. Eksperimen 1



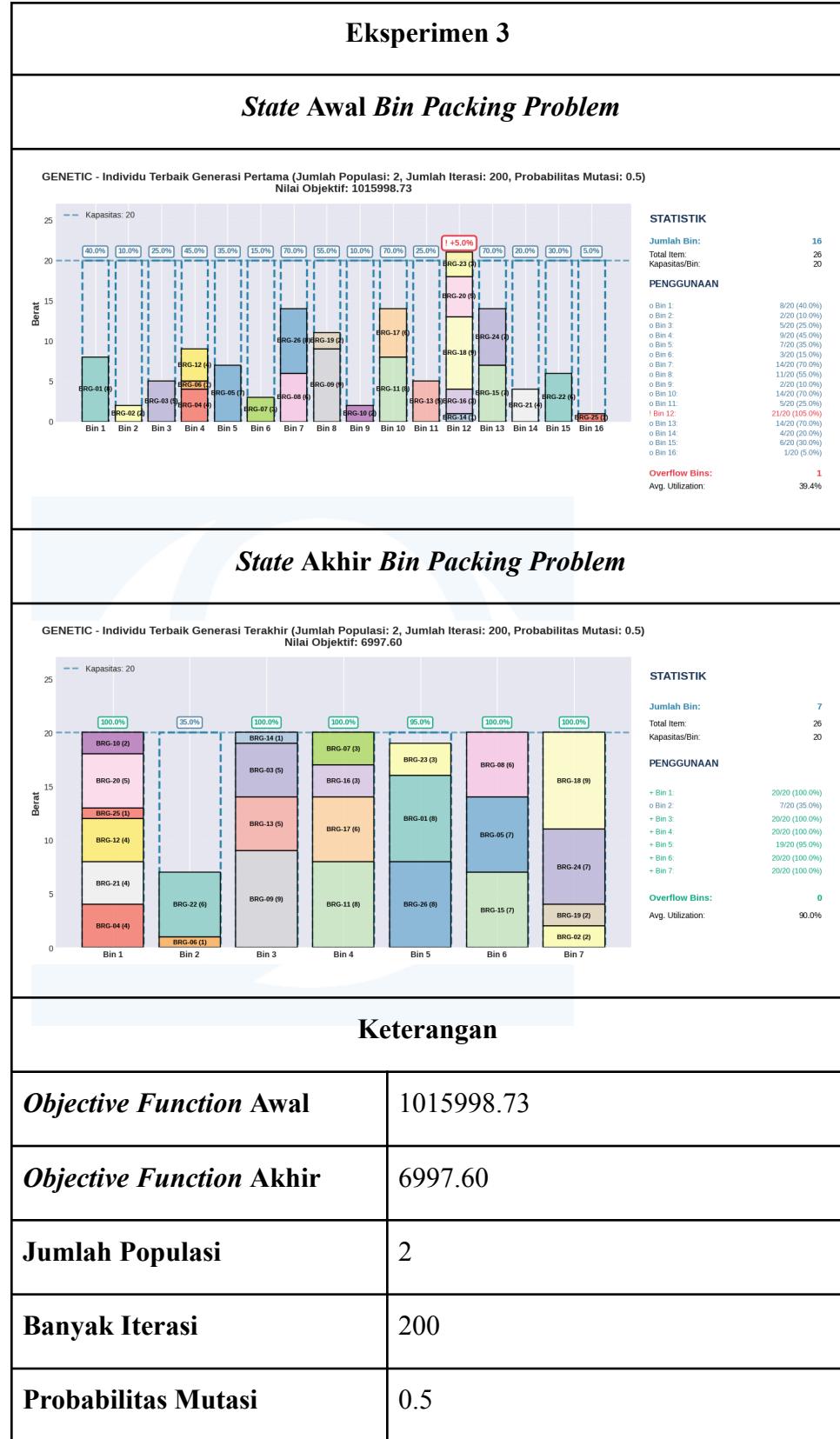


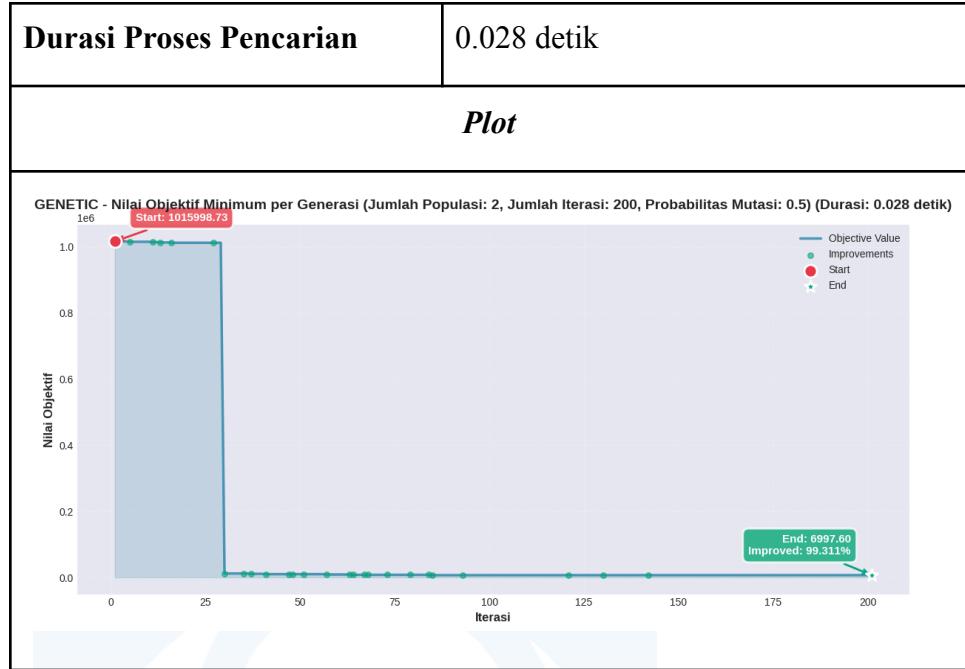
## ii. Eksperimen 2





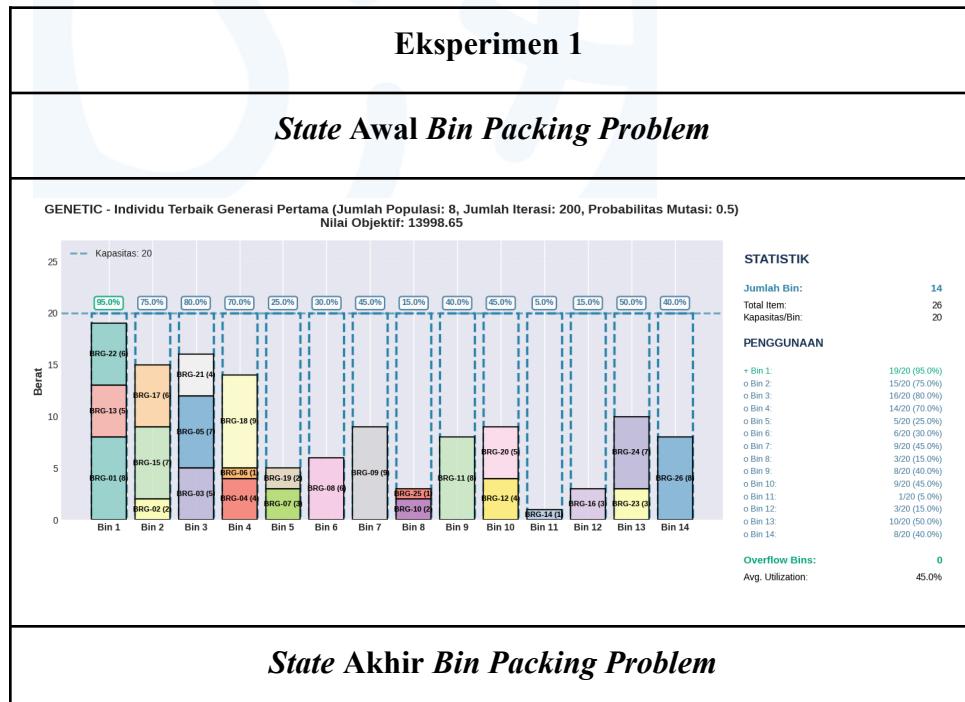
### iii. Eksperimen 3

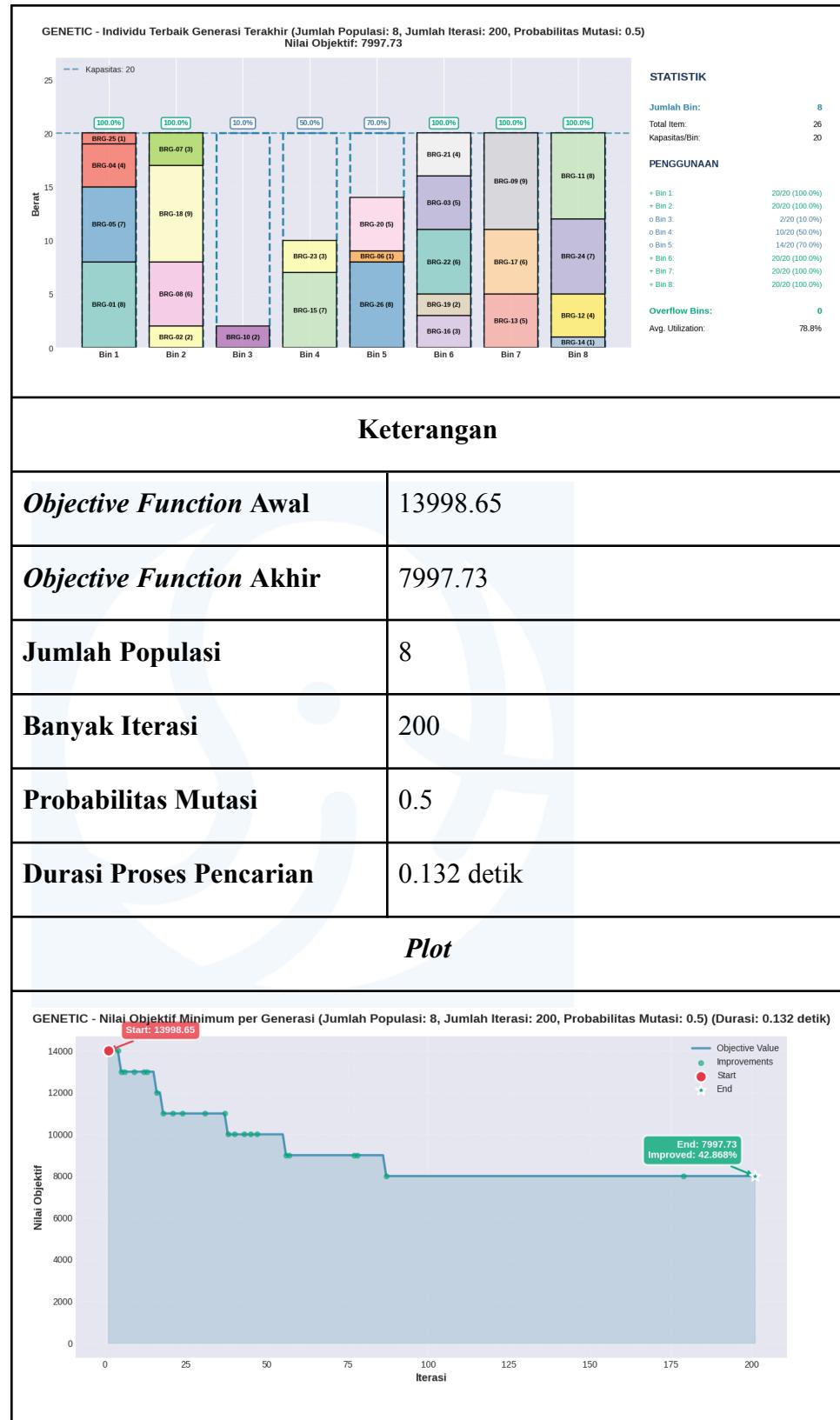




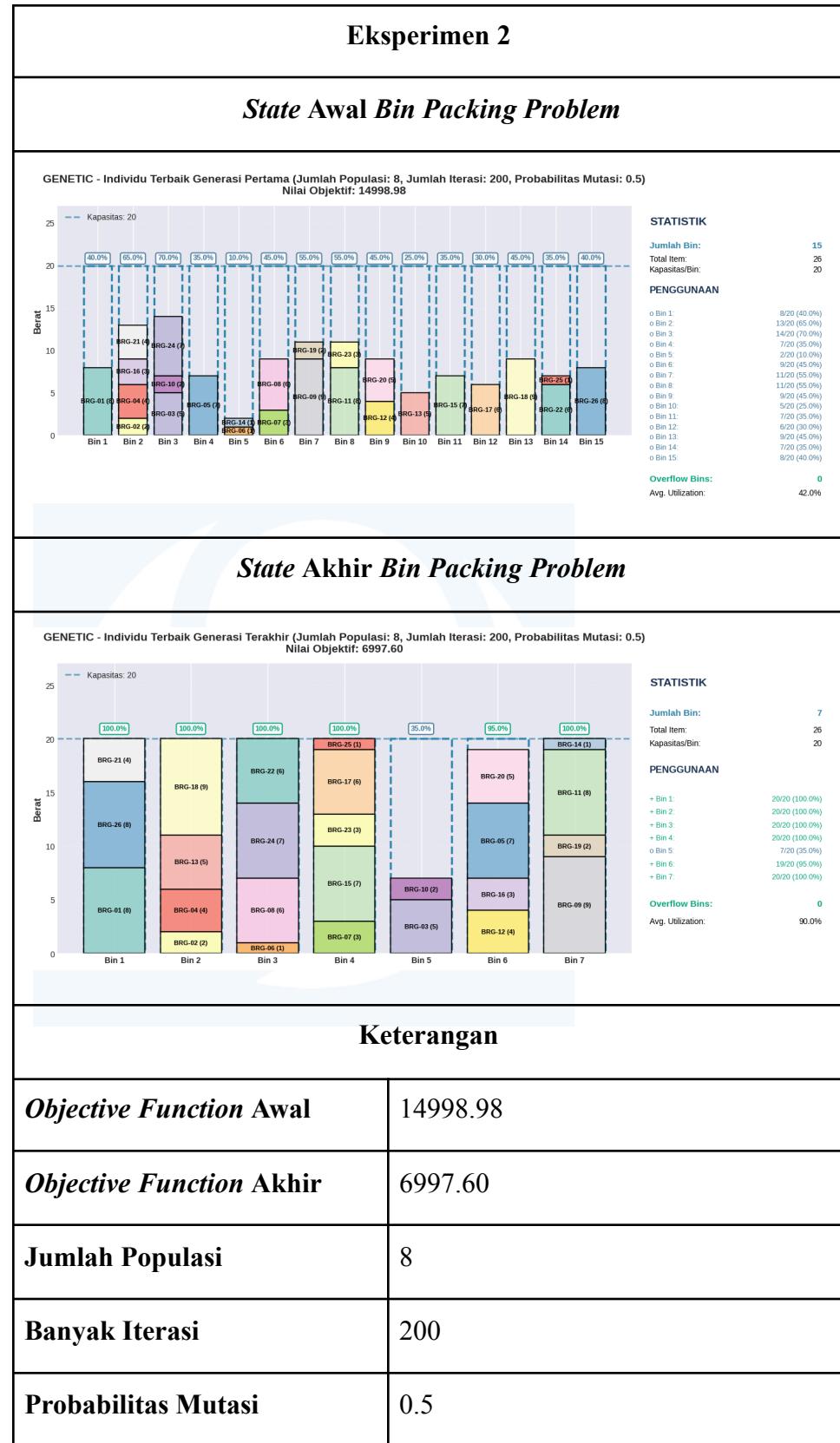
b. Iterasi = 8

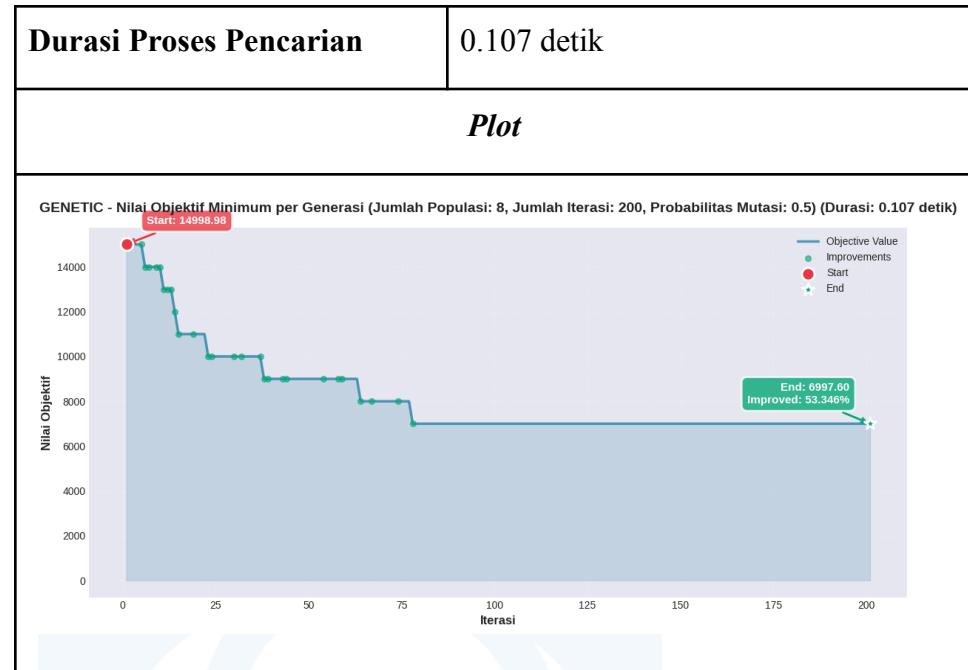
i. Eksperimen 1



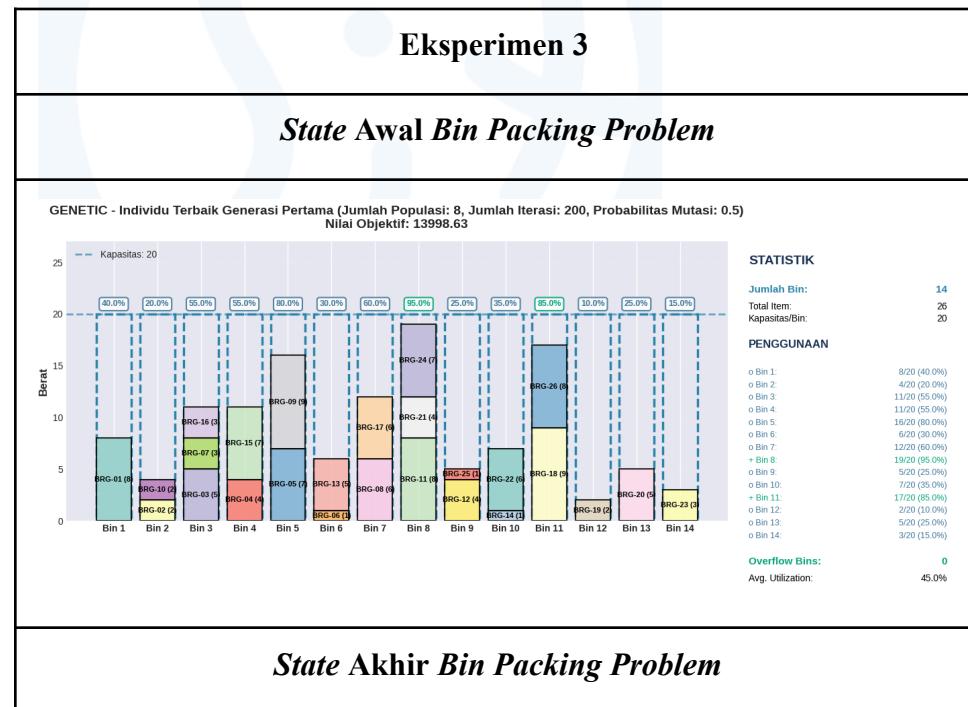


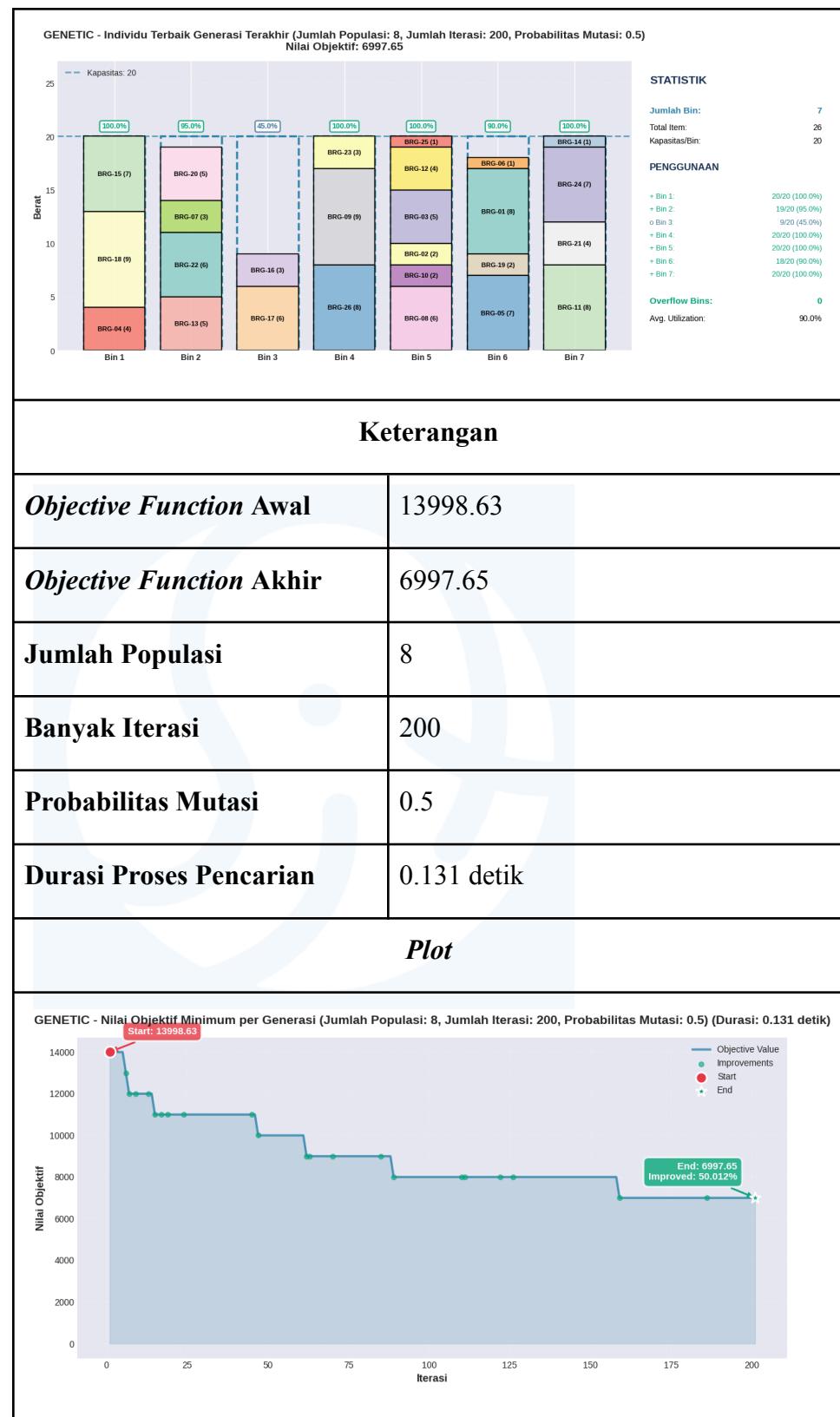
## ii. Eksperimen 2





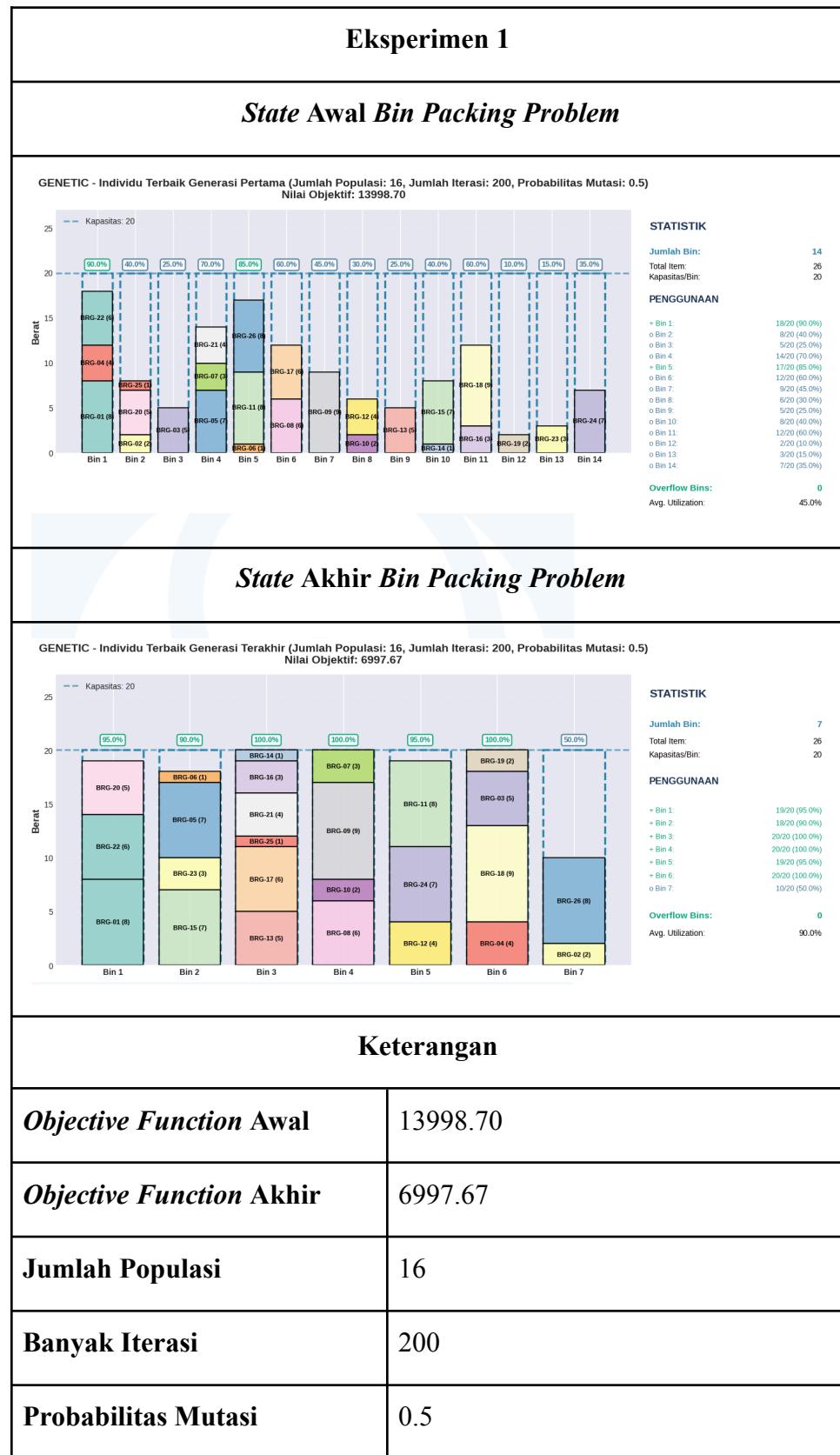
### iii. Eksperimen 3

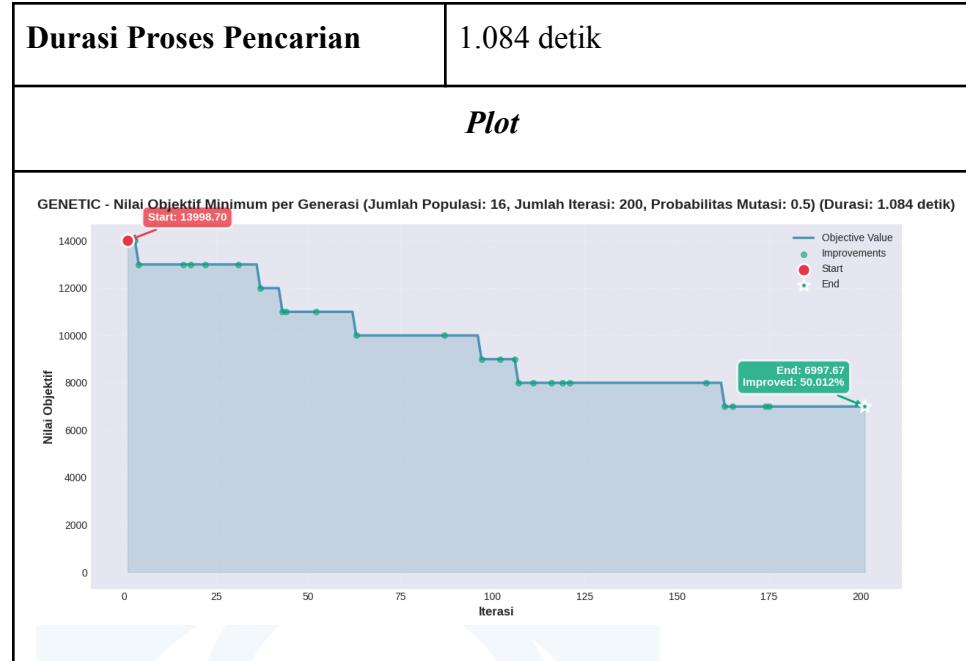




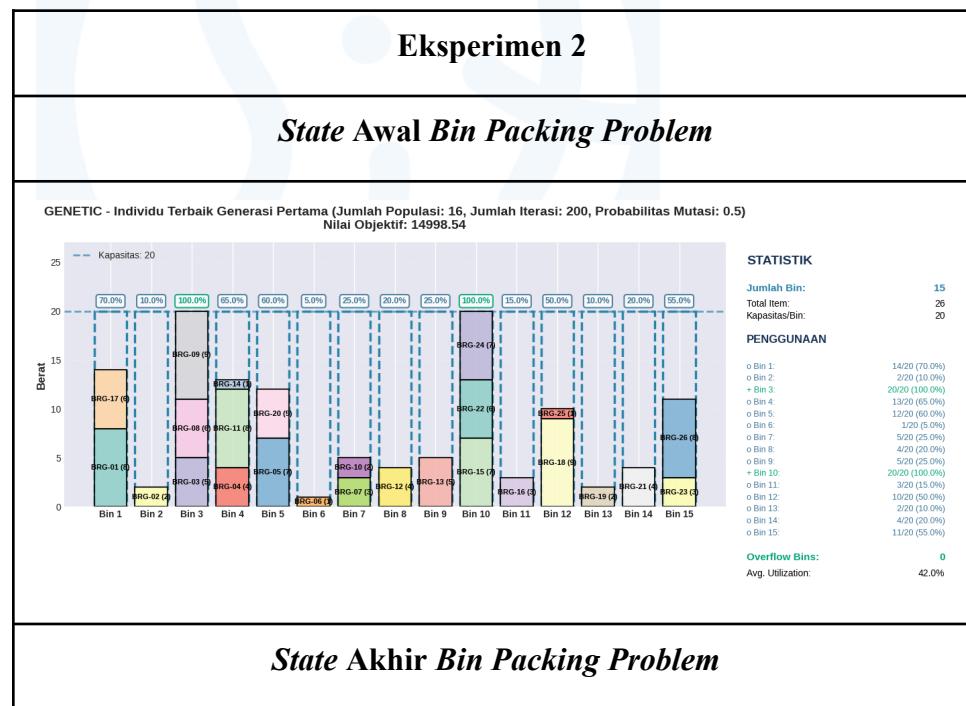
c. Jumlah Populasi = 16

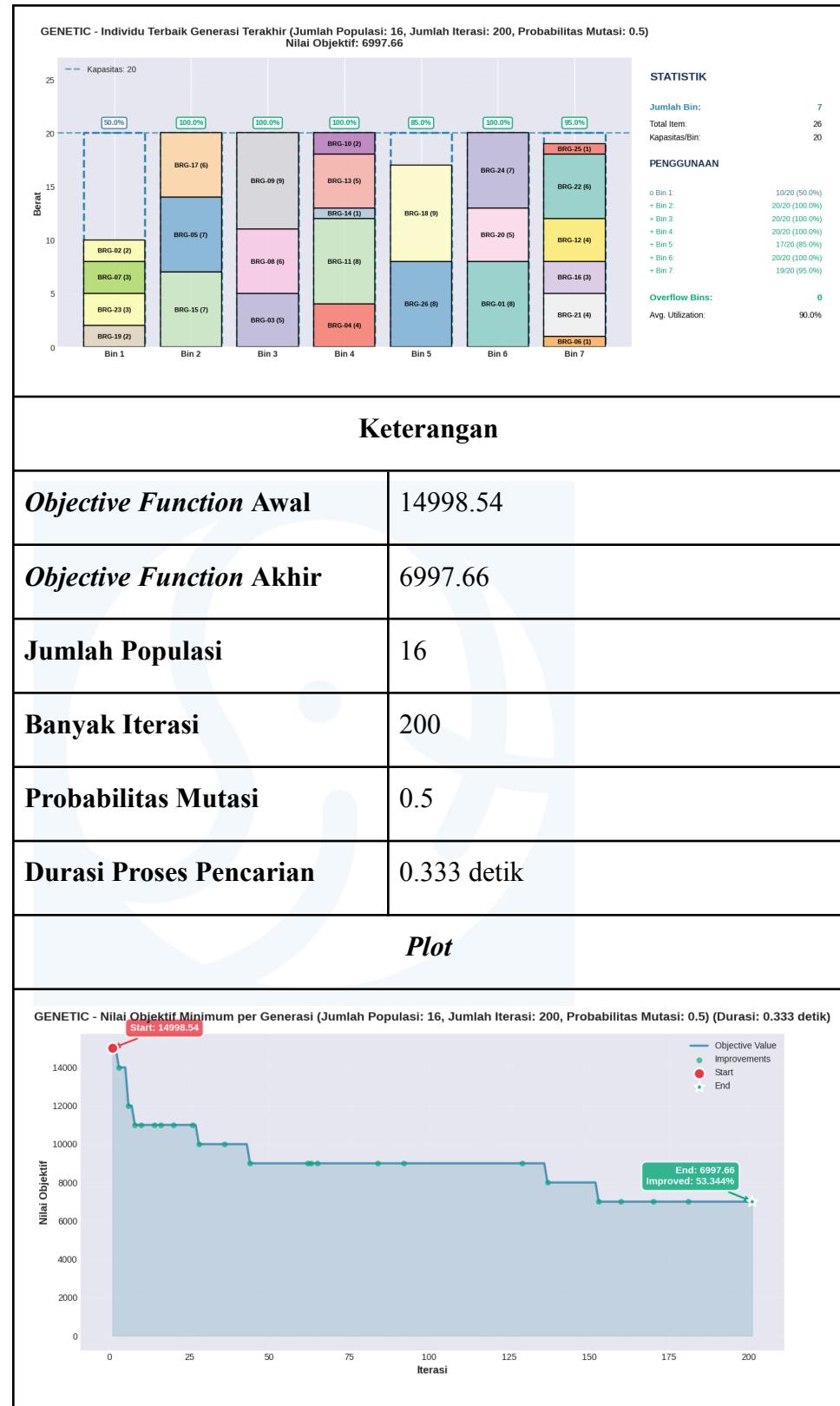
i. Eksperimen 1



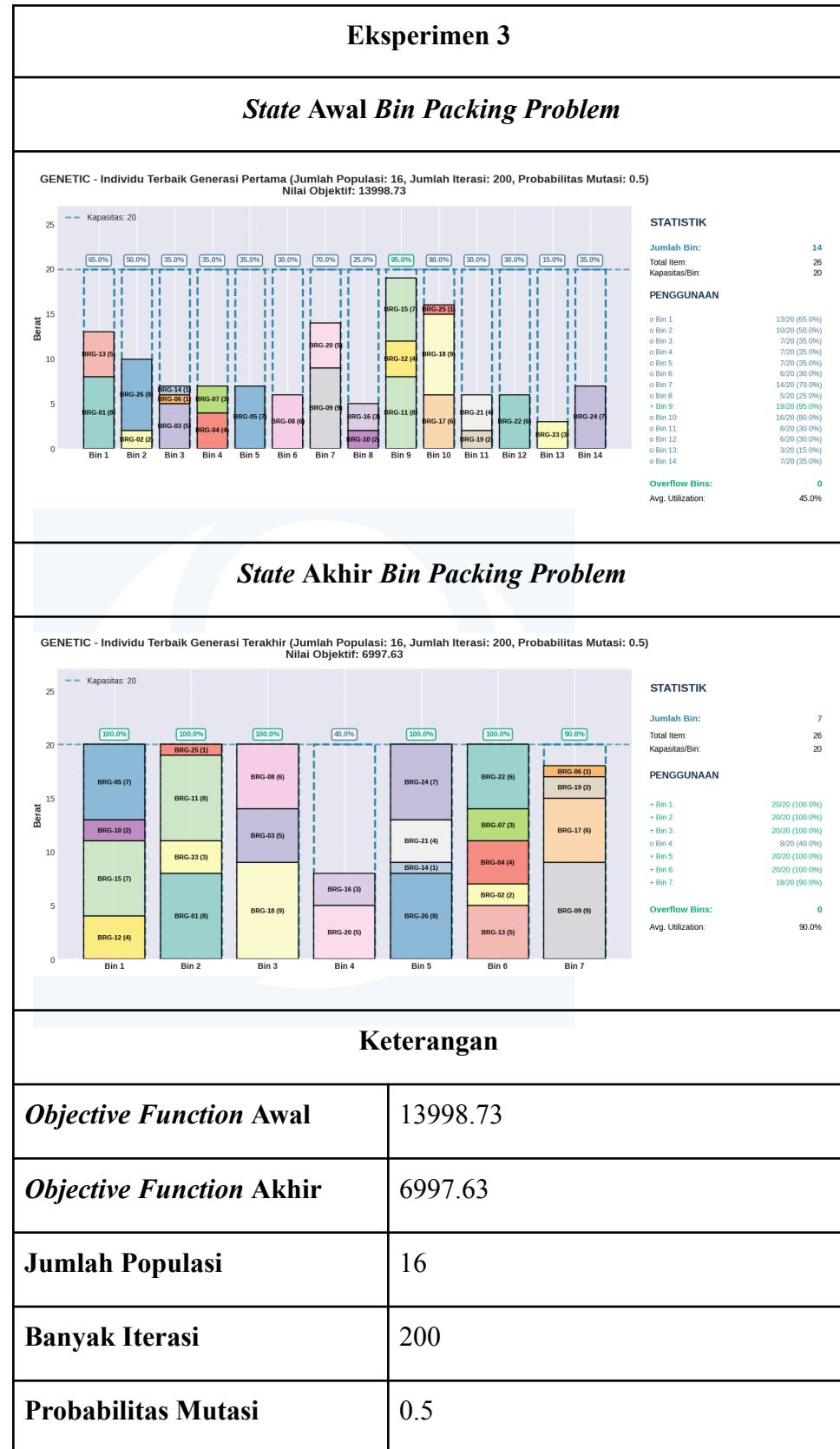


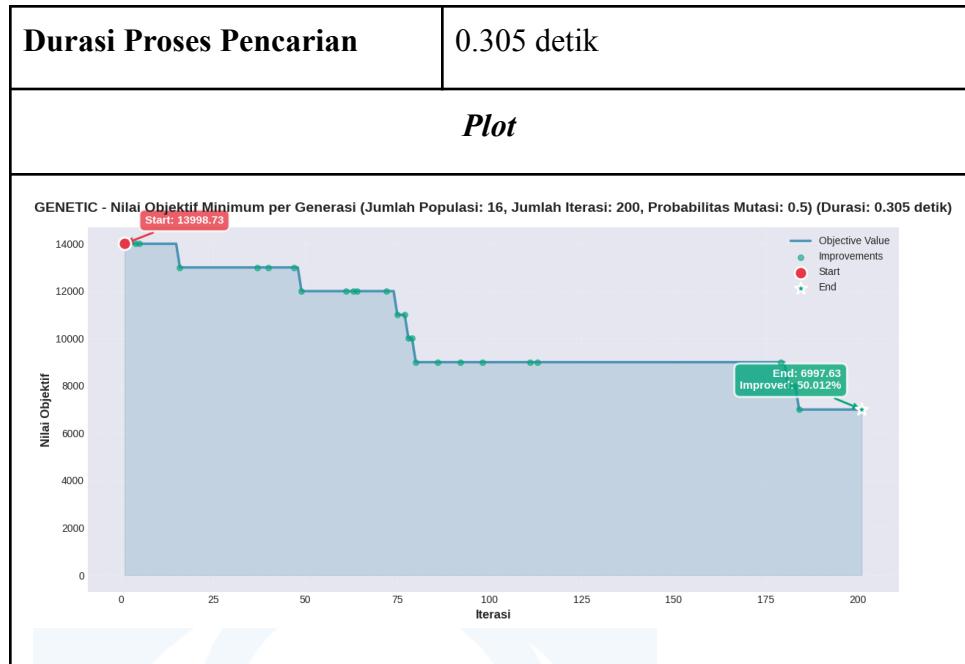
## ii. Eksperimen 2





### iii. Eksperimen 3

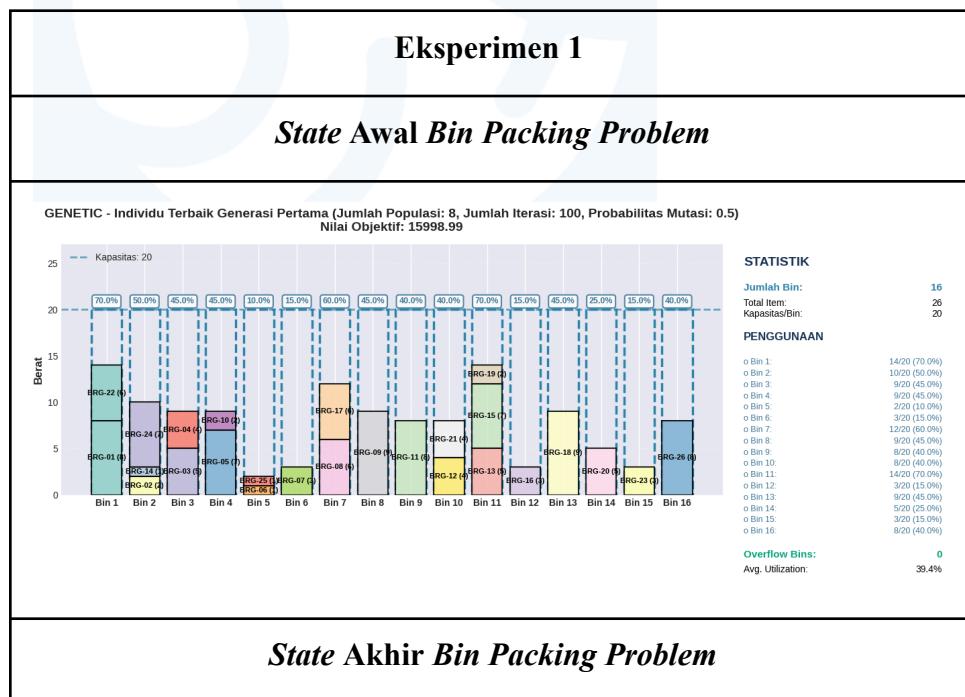


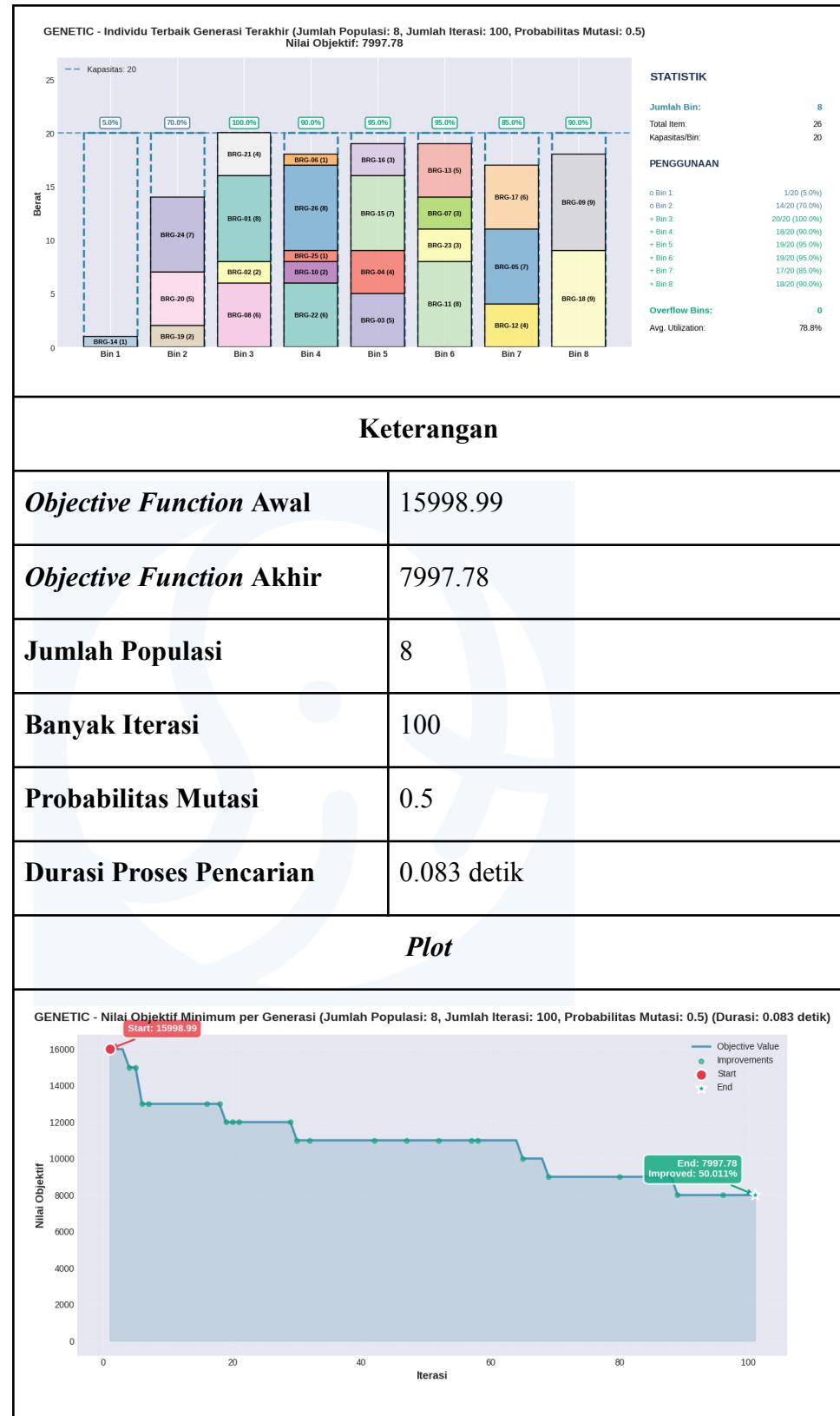


#### 4. Variasi Jumlah Iterasi (Probabilitas Mutasi = 0.5)

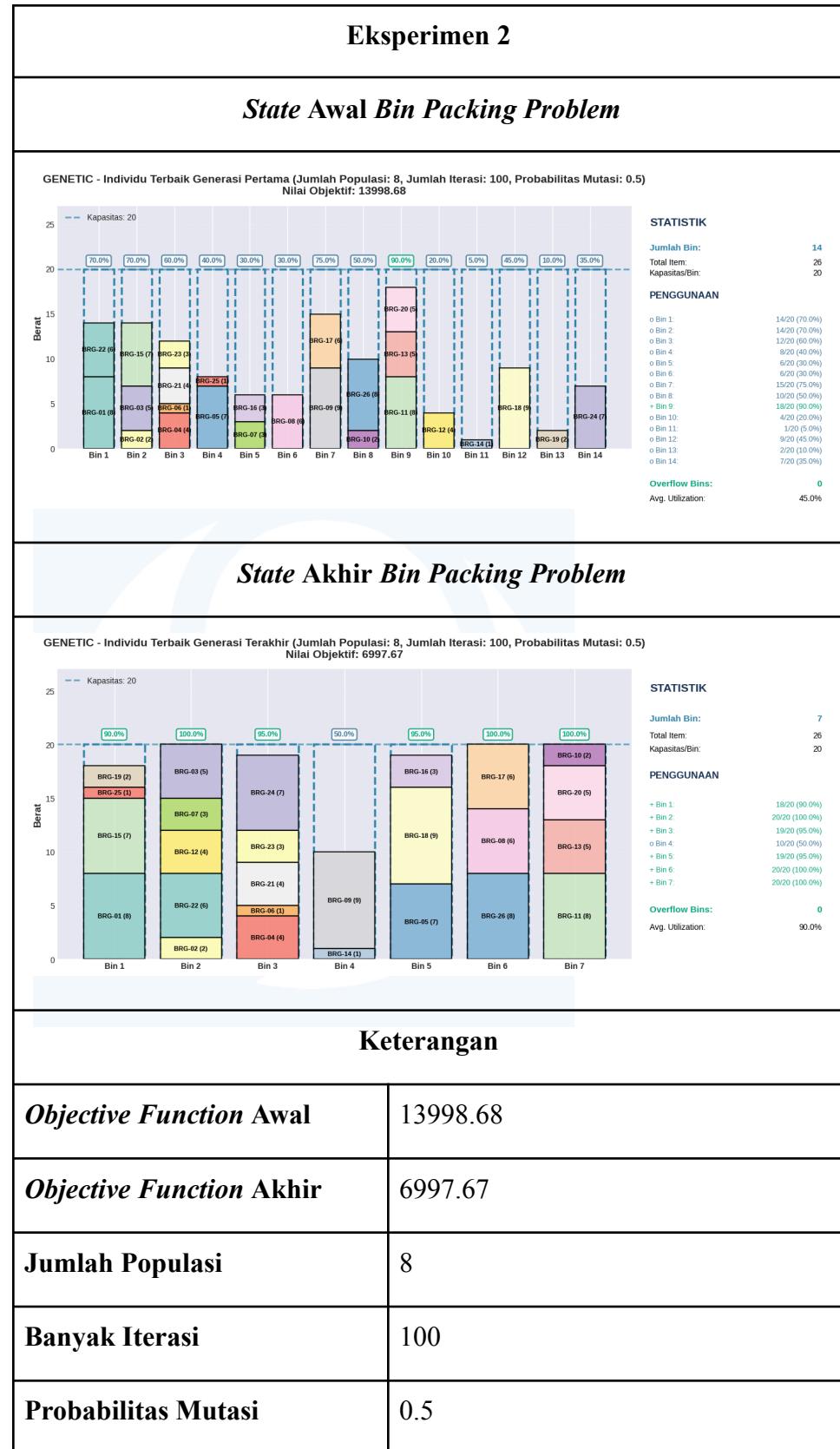
a. Iterasi = 100

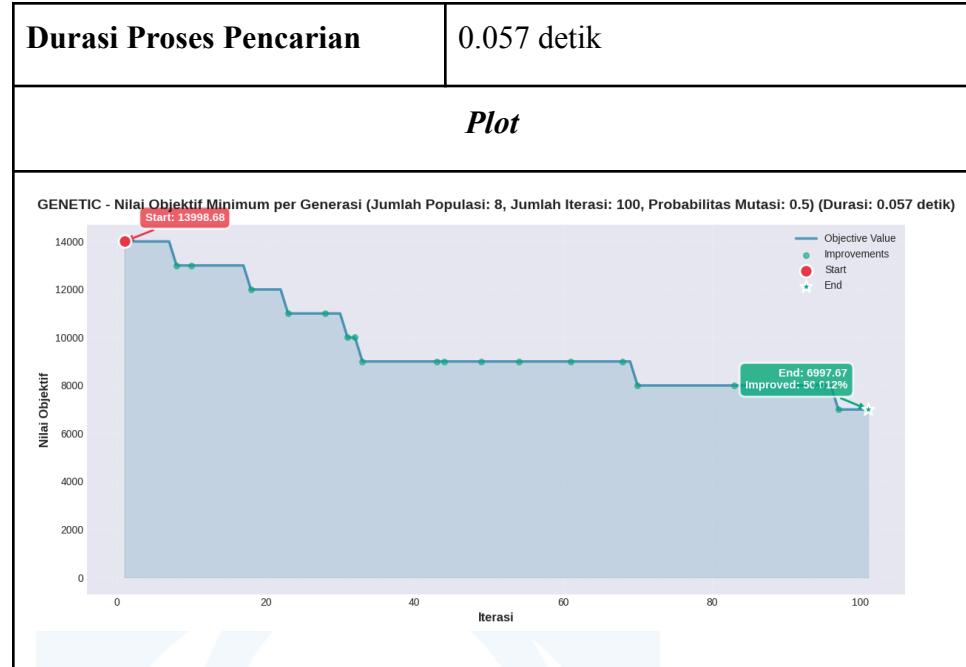
i. Eksperimen 1



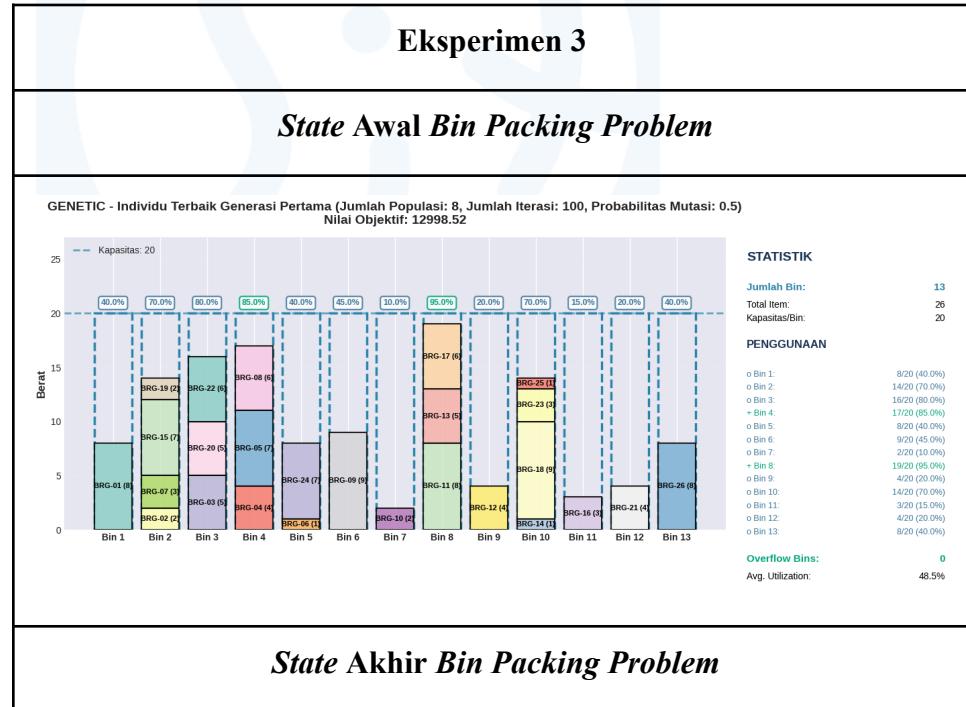


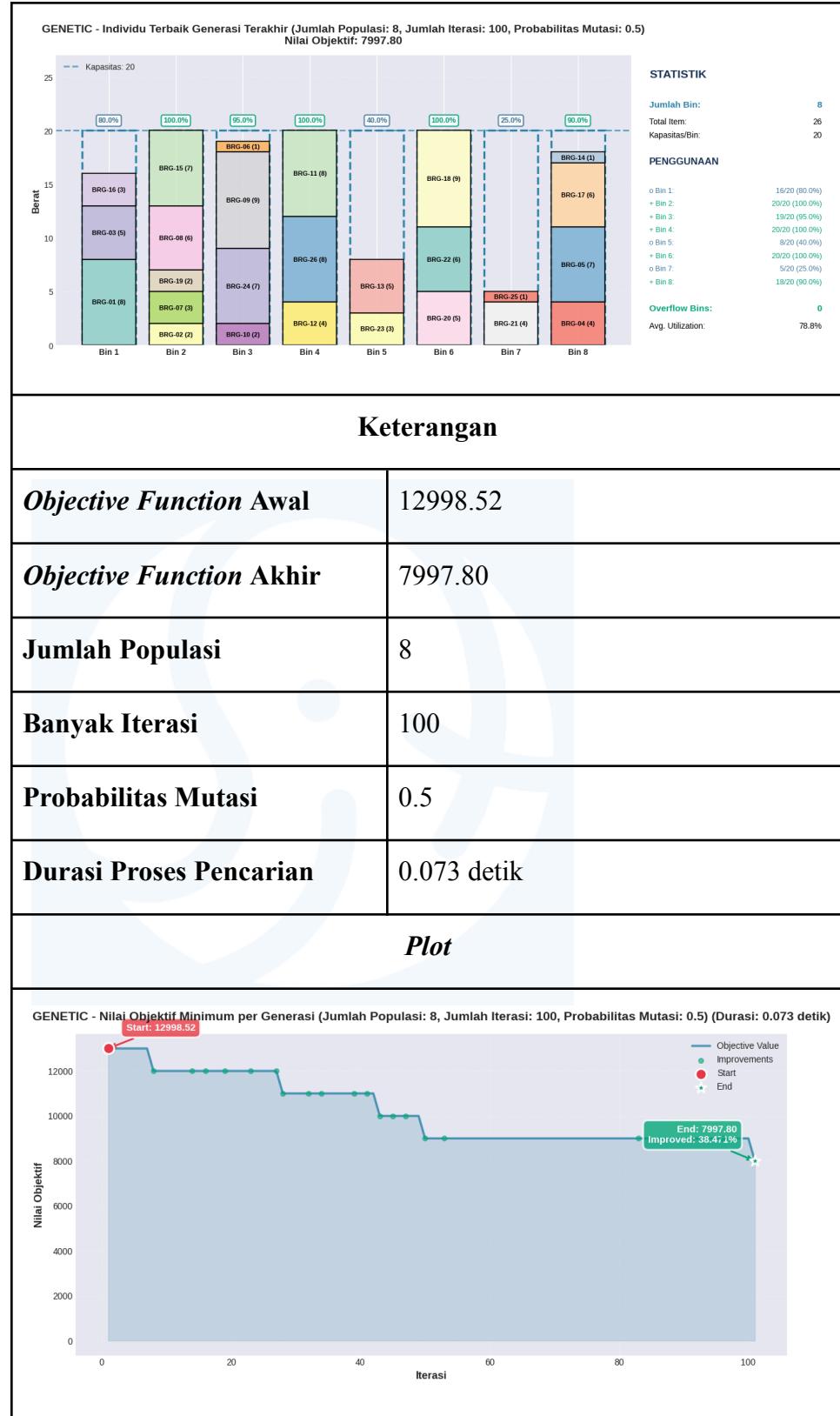
## ii. Eksperimen 2





### iii. Eksperimen 3

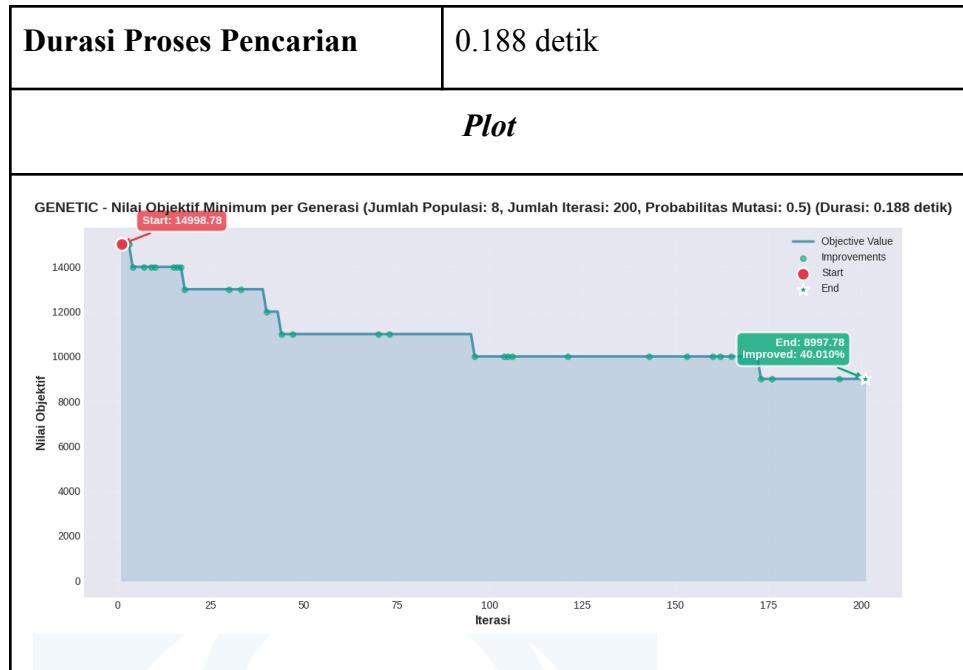




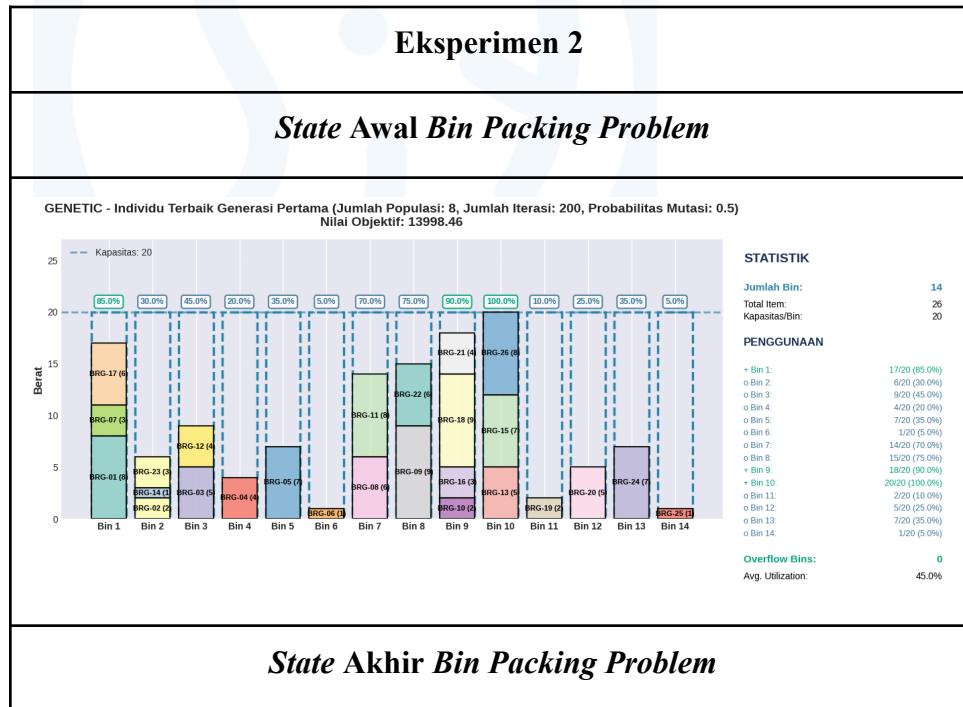
b. Iterasi = 200

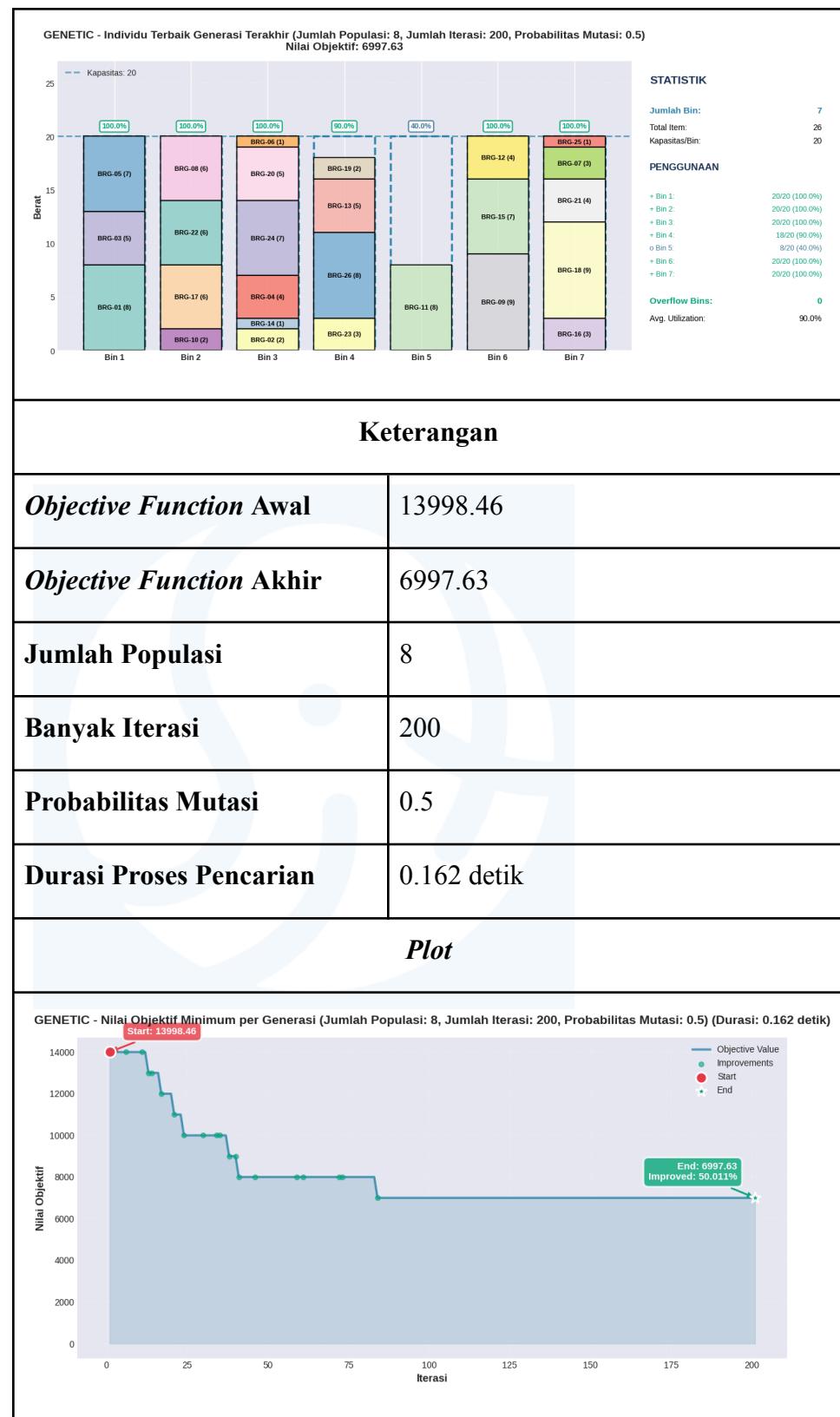
i. Eksperimen 1



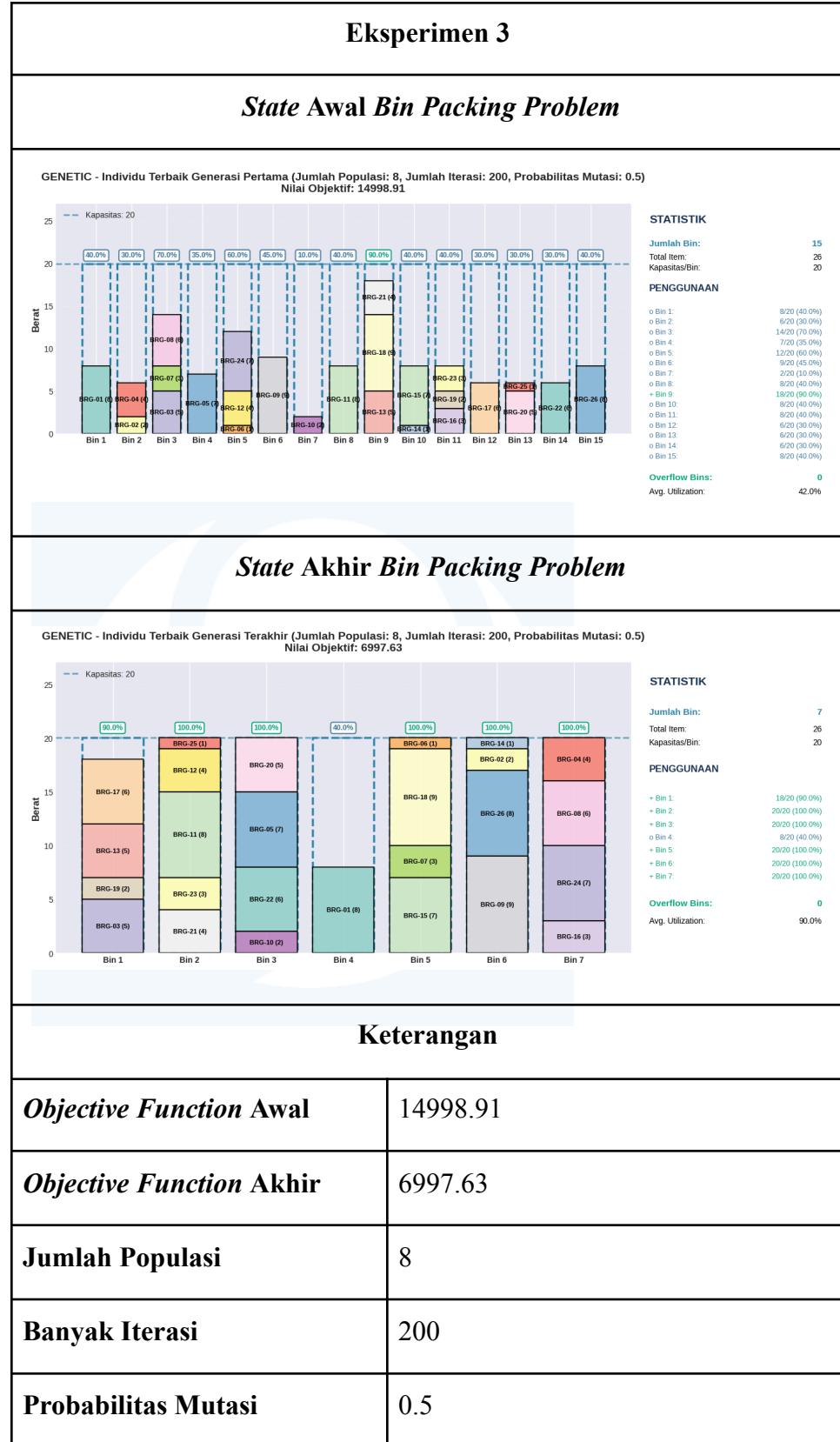


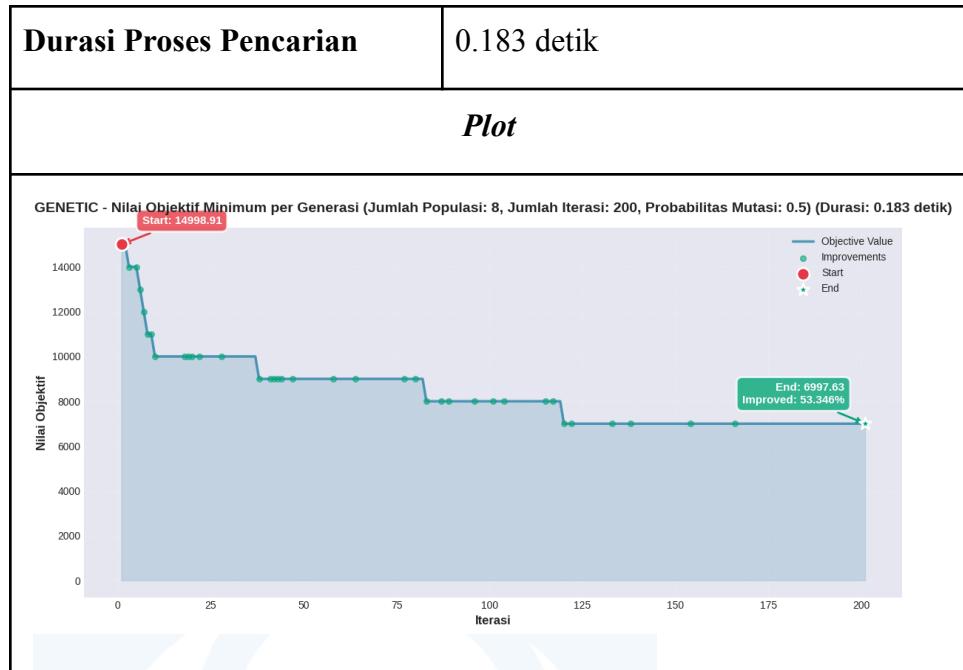
## ii. Eksperimen 2





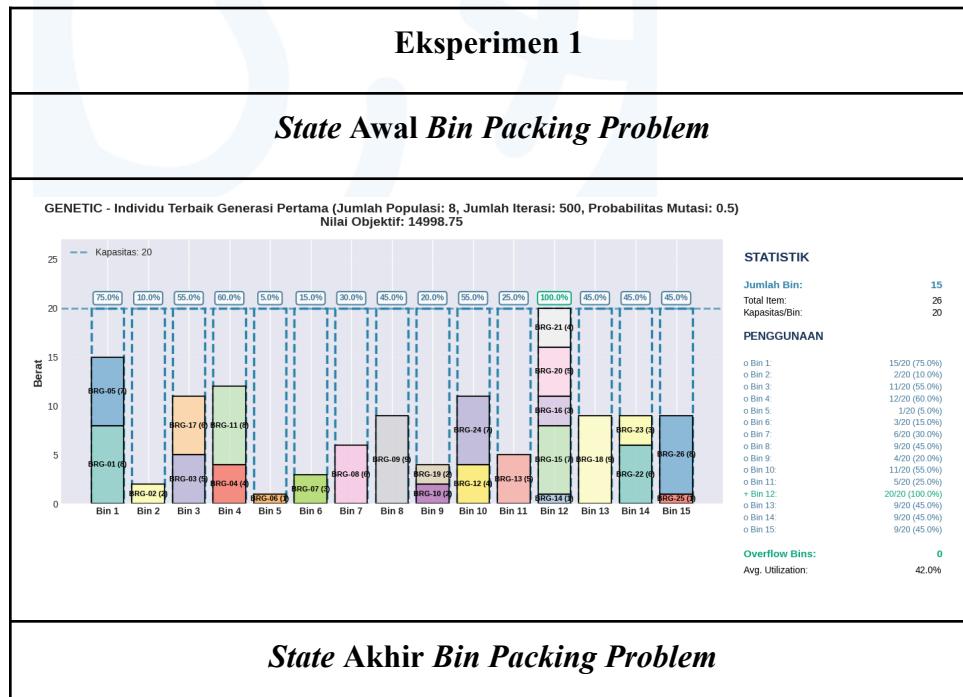
### iii. Eksperimen 3

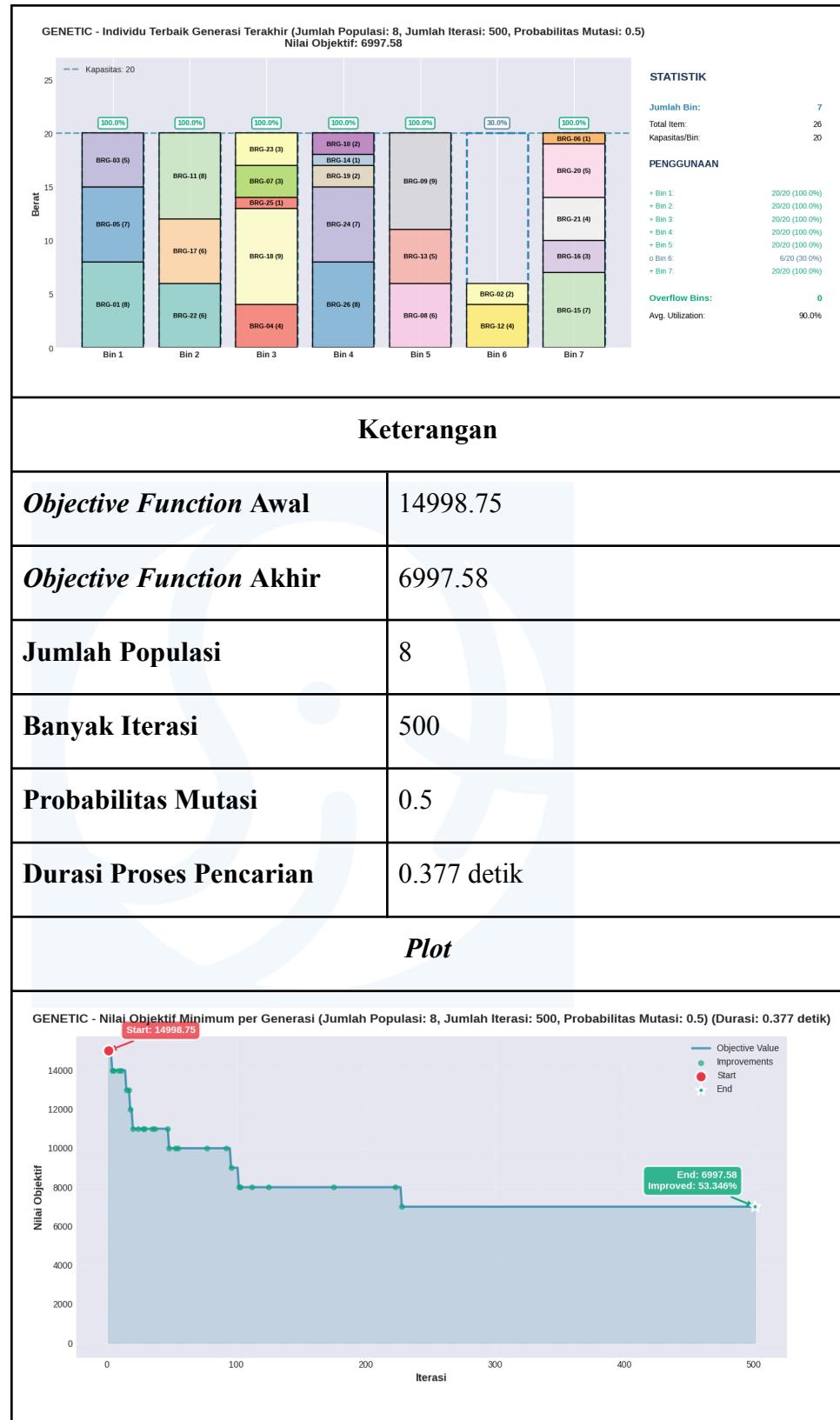




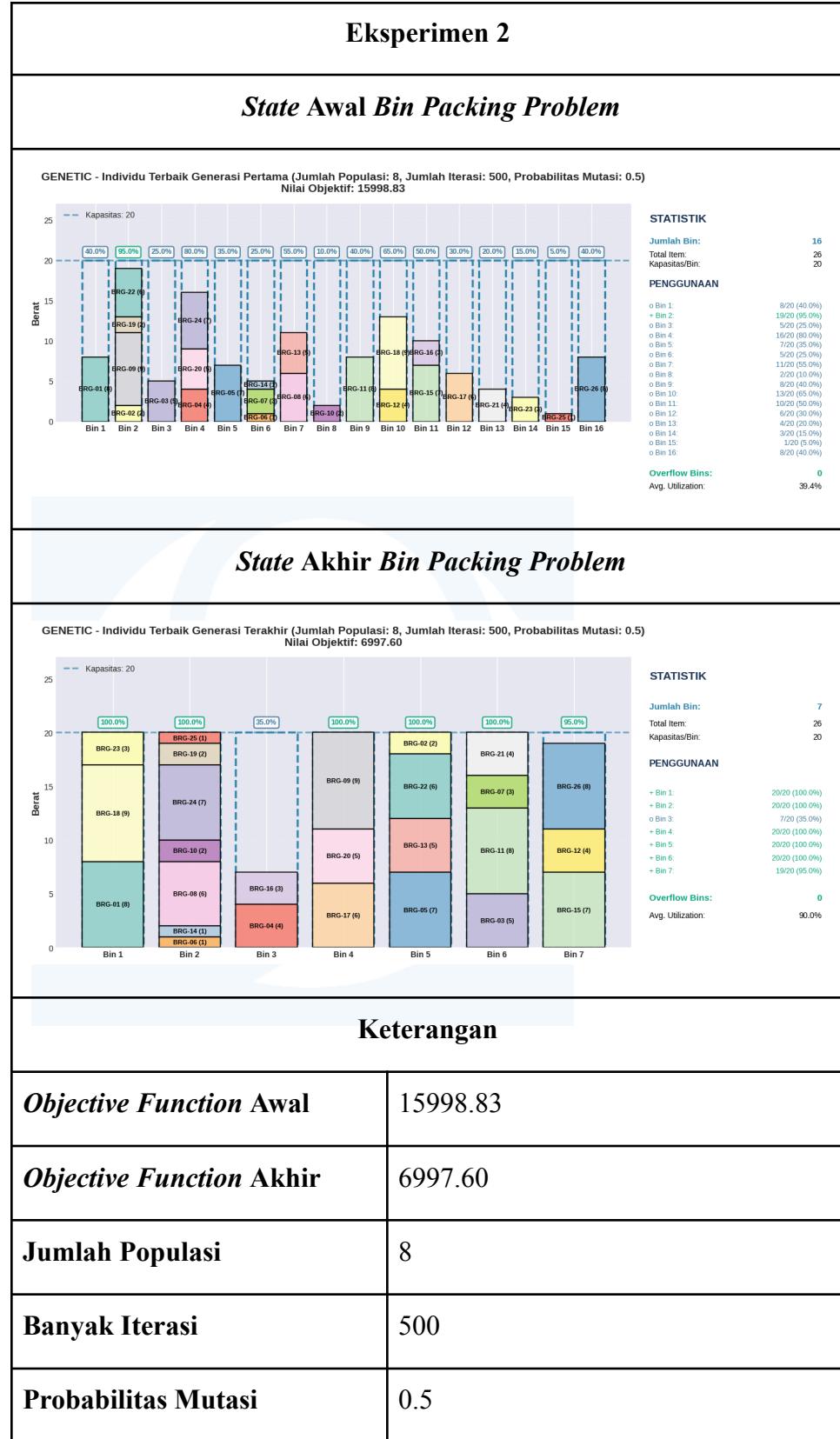
c. Iterasi = 500

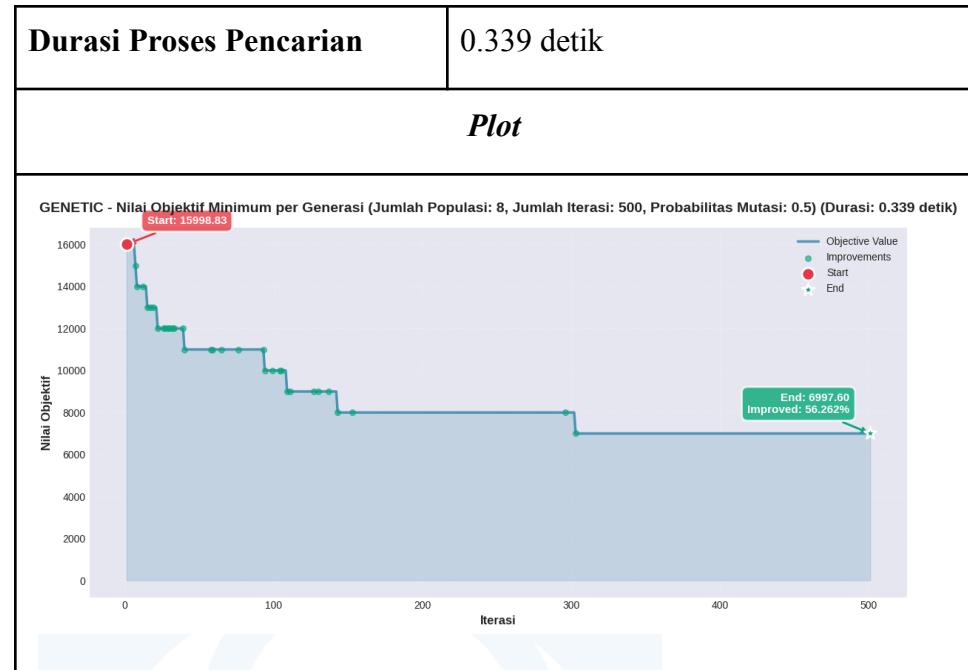
i. Eksperimen 1



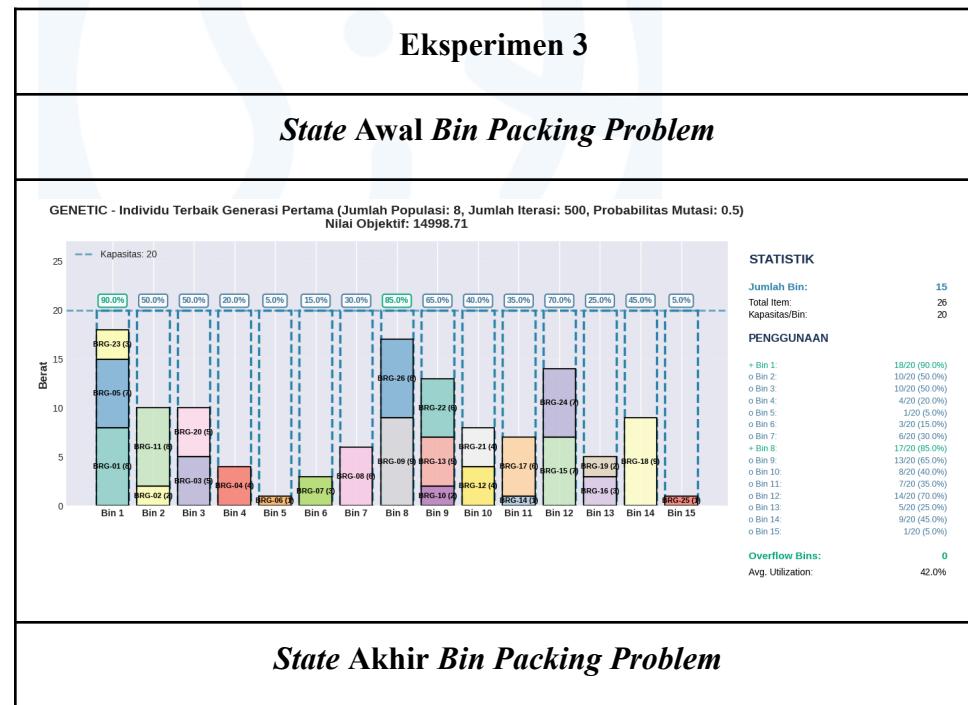


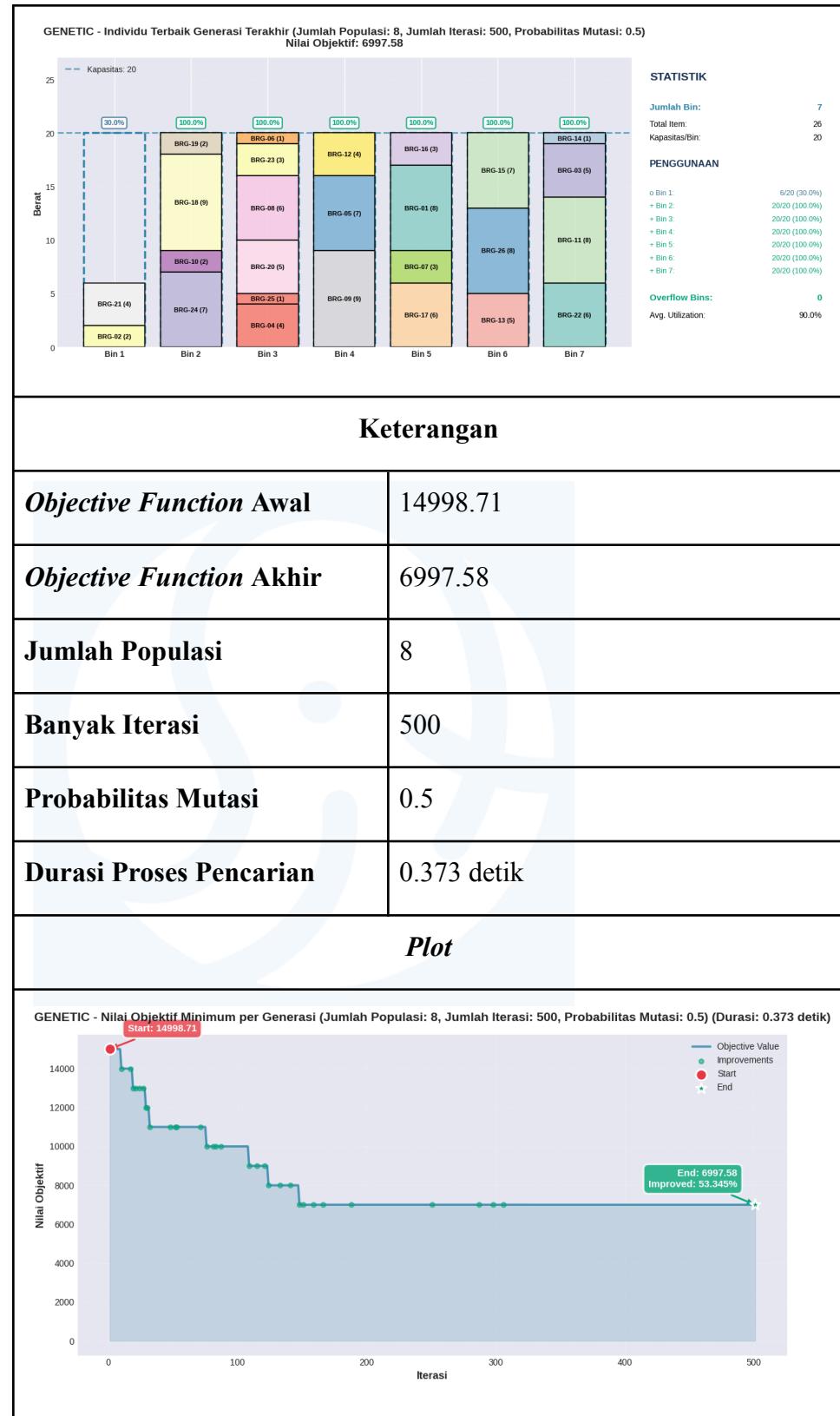
## ii. Eksperimen 2





### iii. Eksperimen 3





## **BAB III**

### **KESIMPULAN DAN SARAN**



## 1. Kesimpulan

Secara umum, hasil eksperimen menunjukkan bahwa setiap algoritma memiliki keunggulan dan kelemahannya sendiri. *Steepest Ascent* bekerja cepat, tetapi sering terjebak pada *local optima* karena ruang pencarinya sempit. Penambahan *sideways move* sedikit membantu keluar dari *plateau*, namun peningkatan hasilnya terbatas. Adapun algoritma yang lebih berkemampuan untuk keluar dari *plateau* adalah *stochastic hill-climbing*, karena kemampuannya untuk berpindah ke tetangga secara acak sepenuhnya, meskipun harus mengorbankan stabilitas.

*Random Restart Hill-Climbing* memperbaiki kelemahan tersebut dengan melakukan pencarian dari beberapa titik awal, sehingga solusi yang diperoleh cenderung lebih baik dan stabil tanpa menambah kompleksitas berlebih. Adapun kekurangan dari algoritma ini adalah waktunya yang cenderung lebih lama karena sangat bergantung dengan jumlah *reset* yang dilakukan.

*Simulated Annealing* menunjukkan performa paling konsisten dan mendekati global optimum berkat mekanisme penerimaan solusi yang lebih buruk di awal proses, meskipun memiliki waktu komputasinya lebih lama.

Sementara itu, *Genetic Algorithm* mampu mengeksplorasi banyak solusi sekaligus dan hasilnya sangat bergantung pada pengaturan populasi serta tingkat mutasi. Dengan konfigurasi yang tepat, kualitas solusinya dapat menyamai atau bahkan melampaui *Simulated Annealing*. Meskipun begitu, perlu diingat bahwa *Genetics Algorithm* memiliki parameter yang lebih banyak dibandingkan algoritma lain sehingga diperlukan berbagai iterasi percobaan untuk dapat mencapai parameter algoritma yang paling optimal.

Secara keseluruhan, algoritma dengan eksplorasi lebih luas seperti *Simulated Annealing* dan *Genetic Algorithm* menghasilkan solusi terbaik, sedangkan algoritma *hill-climbing* unggul dalam kecepatan. Pilihan akhirnya sangat bergantung dengan kompromisasi antara efisiensi waktu atau kualitas hasil.

## **2. Saran**

Berdasarkan eksperimen dengan beberapa algoritma dan mekanisme yang berbeda-beda. Berikut merupakan saran dan masukkan untuk penelitian kedepannya:

1. Penelitian lanjutan perlu dilakukan dengan variasi parameter yang lebih luas, seperti jumlah *item* atau barang, kapasitas kontainer, bobot penalti (*weighting factors*), serta suhu awal dan laju pendinginan pada *Simulated Annealing*. Hal ini penting untuk memahami sensitivitas parameter terhadap kualitas solusi dan waktu konvergensi algoritma.
2. Kombinasi metode seperti *Simulated Annealing* dan *Genetic Algorithm* atau *Hill-Climbing with Adaptive Restart* berpotensi menghasilkan hasil yang lebih cepat dan stabil. Selain itu, visualisasi hasil algoritma dapat ditingkatkan agar lebih interaktif dan informatif untuk memantau dinamika proses pencarian solusi.
3. Penelitian juga dapat dikembangkan ke versi dua atau tiga dimensi dengan mempertimbangkan rotasi dan bentuk fisik item agar lebih relevan terhadap kebutuhan industri.

## **BAB IV**

### **PEMBAGIAN TUGAS**

#### **1. Pembagian Tugas**

Berikut merupakan pembagian tugas dalam pengerojaan tugas besar IF3070 Dasar Intelektualisasi Artifisial.

**Tabel 4.1** Pembagian Tugas

<b>NIM</b>	<b>Nama</b>	<b>Pembagian Tugas</b>
18223003	Wisa Ahmaduta Dinutama	Mengerjakan <i>Objective Function</i>
		Mengerjakan <i>Bin Packing Base</i>
		Mengerjakan visualisasi dari algoritma-algoritma yang ada
		Mengerjakan <i>Genetics Algorithm</i>
		Mengerjakan <i>Stochastic Hill-Climbing</i>
		Mengoreksi dan memperbaiki visualisasi dari algoritma <i>hill-climbing</i> dan <i>simulated annealing</i>
		Mengerjakan visualisasi dari <i>Genetics Algorithm</i>
		Mengerjakan laporan bagian <i>Genetics Algorithm</i>
18223033	Persada Ramiiza Abyudaya	Mengerjakan <i>Simulated Annealing</i>
		Mengerjakan visualisasi dari algoritma-algoritma yang ada

		Mengerjakan laporan BAB 2, Referensi
182223035	Inggried Amelia Deswenty	Mengerjakan <i>Steepest Ascent Hill-Climbing</i>
		Mengerjakan <i>Random Restart Hill-Climbing</i>
		Mengerjakan <i>Hill-Climbing with Sideways Move</i>
		Menyediakan <i>template</i> dan mengerjakan laporan BAB 1, 2, 3, 4, Referensi

## REFERENSI

Brownlee, J. (2021). *Stochastic hill climbing in Python from scratch*. Machine Learning Mastery.

<https://machinelearningmastery.com/stochastic-hill-climbing-in-python-from-scratch/>

Google OR-Tools. (2024). *The bin packing problem*. Google Developers.

[https://developers.google.com/optimization/bin/bin\\_packing](https://developers.google.com/optimization/bin/bin_packing)

Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2), 311–329. <https://doi.org/10.1287/moor.13.2.311>

Levine, J., & Ducatelle, F. (2004). Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7), 705–716. <https://doi.org/10.1057/palgrave.jors.2601723>

Nourani, Y., & Andresen, B. (1998). A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41), 8373–8385. <https://doi.org/10.1088/0305-4470/31/41/003>

Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. A. (2007). A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, 53(10), 1605–1614. <https://doi.org/10.1016/j.camwa.2006.08.039>

Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Prentice Hall.

Sonuc, E., Sen, B., & Bayir, S. (2017). Solving bin packing problem using simulated annealing. *International Journal of Mechanical and Production Engineering*, 5(3), 21–23.

Weise, T., Wu, Z., Chiong, R., Tang, K., & Lässig, J. (2024). Randomized local search on the 2D rectangular bin packing problem with item rotation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '24)*. Association for Computing Machinery. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

Zhang, B., et al. (2024). A GAN-based genetic algorithm for solving the 3D bin packing problem. *Scientific Reports*, 14, 8199. <https://doi.org/10.1038/s41598-024-8199-5>



## **LAMPIRAN**

### **Lampiran 1. Tautan GitHub**

[Tautan GitHub](#)

### **Lampiran 2. Tautan Google Colab**

 [BinPacking.ipynb](#)

