LAPORAN TUGAS BESAR 1

IF3170 INTELIGENSI ARTIFISIAL

Pencarian Solusi Diagonal Magic Cube dengan Local Search

Disusun Oleh:

13522069 Nabila Shikoofa Muida

13522096 Novelya Putri Ramadhani

13522102 Hayya Zuhailii Kinasih

13522104 Diana Tri Handayani

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

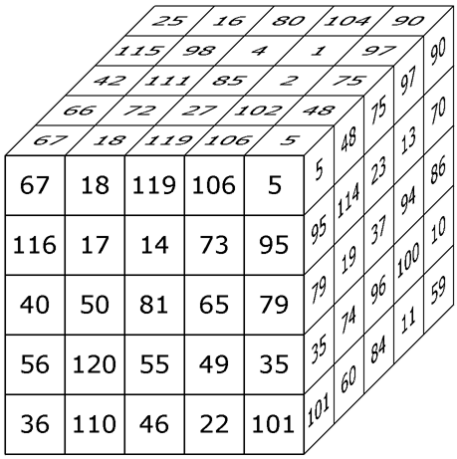
DAFTAR ISI

DAFTAR ISI.....1
BAB I
DESKRIPSI MASALAH..... 3
1.1. Deskripsi Persoalan.....3
1.2. Spesifikasi Tugas..... 4
BAB II
LANDASAN TEORI..... 6
2.1. Diagonal Magic Cube..... 6
2.1.1. Magic Number..... 6
2.1.2. Sifat Diagonal Magic Cube.....7
2.2. Hill-Climbing Algorithm..... 8
2.2.1. Steepest Ascent Hill-climbing..... 9
2.2.2. Hill-climbing with Sideways Move.....9
2.2.3. Random Restart Hill-climbing.....9
2.2.4. Stochastic Hill-climbing..... 10
2.3. Algoritma Simulated Annealing..... 10
2.4. Genetic Algorithm.....11
BAB III
IMPLEMENTASI..... 13
3.1. Objective Function.....13
3.2. Implementasi Algoritma..... 13
3.2.1. Algoritma Base Algorithm.....13
3.2.2. Algoritma Data Processing..... 15
3.2.3. Algoritma Steepest Ascent Hill-Climbing.....18
3.2.4. Algoritma Hill-Climbing with Sideways Move.....19
3.2.5. Algoritma Random Restart Hill-Climbing.....20
3.2.6. Algoritma Stochastic Hill-Climbing.....21
3.2.7. Algoritma Simulated Annealing..... 22
3.2.8. Algoritma Genetic.....24
3.2.9. Algoritma Visualisasi.....27
BAB IV
HASIL EKSPERIMEN DAN ANALISIS.....29
4.1. Hasil Eksperimen Algoritma Local Search.....29
4.1.1. Steepest Ascent Hill-climbing..... 29
4.1.2. Hill-climbing with Sideways Move.....35

4.1.3. Random Restart Hill-climbing.....41
4.1.4. Stochastic Hill-climbing..... 47
4.1.5. Simulated Annealing.....53
4.1.6. Genetic Algorithm..... 62
4.2. Perbandingan Waktu Algoritma Local Search.....98
4.3. Perbandingan Hasil Solusi Algoritma Local Search.....98
4.4. Perbandingan Hasil Genetic Algorithm.....99
4.5. Analisis Algoritma Local Search..... 100
BAB V
PENGERJAAN BONUS..... 105
5.1. Algoritma Player.....105
5.2. Dokumentasi..... 107
BAB VI
PENUTUP.....108
6.1. Kesimpulan..... 108
6.2. Saran..... 108
LAMPIRAN.....109
DAFTAR PUSTAKA.....110

BAB I
DESKRIPSI MASALAH

1.1. Deskripsi Persoalan

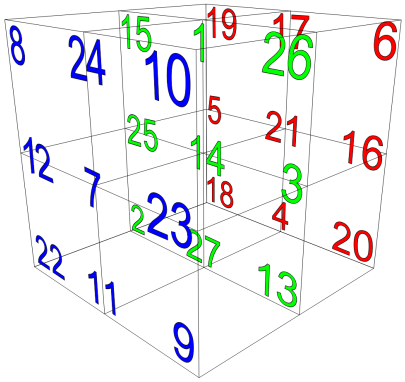


Gambar 1.1. Diagonal Magic Cube 5x5x5
(Sumber: https://www.trump.de/magic-squares/magic-cubes/cubes-1.html)

Diagonal magic cube merupakan kubus yang tersusun dari angka 1 hingga n^3 tanpa pengulangan dengan n adalah panjang sisi pada kubus tersebut. Angka-angka pada tersusun sedemikian rupa sehingga properti-properti berikut terpenuhi:

- Terdapat satu angka yang merupakan magic number dari kubus tersebut (Magic number tidak harus termasuk dalam rentang 1 hingga n^3, magic number juga bukan termasuk ke dalam angka yang harus dimasukkan ke dalam kubus)
- Jumlah angka-angka untuk setiap baris sama dengan magic number
- Jumlah angka-angka untuk setiap kolom sama dengan magic number
- Jumlah angka-angka untuk setiap tiang sama dengan magic number
- Jumlah angka-angka untuk seluruh diagonal ruang pada kubus sama dengan magic number
- Jumlah angka-angka untuk seluruh diagonal pada suatu potongan bidang dari kubus sama dengan magic number

Berikut ilustrasi dari potongan bidang yang ada pada suatu kubus berukuran 3:



Gambar 1.2. Diagonal Magic Cube 3x3x3
(Sumber: https://en.wikipedia.org/wiki/Magic_cube)

- Terdapat 9 potongan bidang, yaitu:

8	24	10	15	1	26	19	17	6
12	7	23	25	14	3	5	21	16
22	11	9	2	27	13	18	4	20

19	17	6	5	21	16	18	4	20
15	1	26	25	14	3	2	27	13
8	24	10	12	7	23	22	11	9

8	15	19	12	25	5	22	2	18
24	1	17	7	14	21	11	27	4
10	26	6	23	3	16	9	13	20

Gambar 1.3. Diagonal Bidang Magic Cube 3x3x3
(Sumber: Spesifikasi Tugas Kecil 1 Inteligensi Buatan)

1.2. Spesifikasi Tugas

Dalam penyelesaian masalah Diagonal Magic Cube ini, terdapat beberapa hal yang harus dilakukan, yaitu:

1. Menjelaskan *objective function* yang digunakan beserta alasan pemilihannya.
2. Menjelaskan proses pencarian solusi menggunakan algoritma *Local Search* untuk mencari solusi Diagonal Search dengan memperhatikan *feasibility*, termasuk:
    - Steepest Ascent Hill-climbing
    - Hill-climbing with Sideways Move

    - Random Restart Hill-climbing
    - Stochastic Hill-climbing
    - Simulated Annealing
    - Genetic Algorithm
3. Menjelaskan algoritma *Local Search* yang dianggap terbaik untuk menyelesaikan masalah ini, beserta alasan pemilihannya.
4. Mengidentifikasi kelebihan dan kekurangan pendekatan *Local Search* dibandingkan dengan metode *complete search*.
5. Menyusun rencana kelas atau fungsi yang digunakan, baik dalam bentuk pseudocode maupun deskripsi, untuk mengimplementasikan seluruh algoritma *Local Search* yang disebutkan.

BAB II
LANDASAN TEORI

2.1. Diagonal Magic Cube

2.1.1. Magic Number

Magic Number adalah nilai konstanta dalam Magic Cube berukuran $n \, x \, n \, x \, n$, yang diisi dengan bilangan bulat dari 1 hingga n^3. Tujuan dari Magic Cube ini adalah untuk memastikan bahwa jumlah dari setiap baris, kolom, tiang, dan diagonal memiliki nilai yang sama. Nilai tersebut dikenal sebagai konstanta magis atau *magic constant*, dan biasanya dilambangkan dengan M. Adapun langkah-langkah untuk menentukan nilai Magic Number antara lain:

1. Menghitung jumlah total dari seluruh angka dalam kubus.

   Jumlah seluruh angka dalam Magic Cube dapat dihitung dengan menggunakan rumus jumlah deret aritmatika:

   Jumlah angka = 1 + 2 + 3 + … + n = $\frac{n^3 (n^3+1)}{2}$

2. Menghitung jumlah baris dalam Magic Cube.

   Setiap lapisan memiliki *n* baris dan terdapat *n* lapisan untuk setiap Cube, sehingga terdapat n^2 baris.

3. Membagi jumlah total dari seluruh angka dengan jumlah baris dari Magic Cube.

   Jumlah total dari semua angka dalam kubus adalah $\frac{n^3 (n^3+1)}{2}$, dan ada n^2 baris, maka jumlah pada setiap baris yang merupakan konstanta magis M dapat dihitung dengan membagi jumlah total angka dengan jumlah baris.

$$M = \frac{Jumlah\ Total\ Angka}{Jumlah\ Baris} = \frac{\frac{n^3 (n^3+1)}{2}}{n^2}$$

$$M = \frac{n (n^3+1)}{2}$$

Jadi, konstanta magis M untuk Magic Cube berukuran $n\ x\ n\ x\ n$ adalah:

$$M = \frac{n\,(n^3+1)}{2}$$

4. Menghitung Magic Number untuk kubus ukuran *5 x 5 x5*

   Dengan n = 5:

$$M = \frac{n\,(n^3+1)}{2} = \frac{5\,(5^3+1)}{2} = 315$$

Sehingga, pencarian solusi diagonal magic cube dengan local search dalam laporan ini akan menggunakan nilai 315.

2.1.2. Sifat Diagonal Magic Cube

Pada **Diagonal Magic Cube**, ada sifat khusus yang berlaku pada diagonal-diagonal kubus. Diagonal Magic Cube memastikan bahwa jumlah angka-angka pada diagonal tertentu, baik dalam dimensi dua maupun tiga, juga sama dengan magic number. Ini berlaku untuk:

- **Baris**: Setiap baris dari kubus harus memiliki jumlah yang sama dengan magic number.
- **Kolom**: Setiap kolom dari kubus juga memiliki jumlah yang sama dengan magic number.
- **Tiang**: Setiap tiang vertikal dalam kubus harus berjumlah sama dengan magic number.
- **Diagonal**: Setiap diagonal dalam kubus, baik yang terdapat pada satu bidang kubus maupun yang memotong ruang kubus secara tiga dimensi, harus memenuhi sifat ini.

Sifat-sifat diagonal ini dapat dijelaskan dalam dua kategori:

1) **Diagonal Bidang**

   Pada setiap bidang dalam kubus, terdapat beberapa diagonal yang memenuhi sifat magic cube. Bidang ini bisa berupa sisi depan, belakang, atas, bawah, kiri, atau kanan dari kubus. Diagonal-diagonal ini harus memiliki jumlah angka yang sama dengan magic number.

a) **Lapis Baris (Horizontal Layer)**: Diagonal-diagonal yang memotong setiap lapisan horizontal.
b) **Lapis Kolom (Front to Back Vertical Layer)**: Diagonal-diagonal yang memotong setiap lapisan dari depan ke belakang.
c) **Lapis Tiang (Left to Right Vertical Layer)**: Diagonal-diagonal yang memotong setiap lapisan dari kiri ke kanan.

Setiap lapisan dari tiga dimensi (baris, kolom, dan tiang) berkontribusi terhadap dua diagonal utama, yang secara keseluruhan menghasilkan **2 x 3 x *n* diagonal** yang memenuhi sifat magic number dalam Diagonal Magic Cube.

2) **Diagonal Ruang**

Diagonal ruang adalah diagonal yang memotong kubus dari satu sudut ke sudut yang berlawanan, melewati berbagai lapisan bidang di dalam kubus. Pada setiap kubus $n \, x \, n \, x \, n$, terdapat **4 diagonal ruang**. Pada Diagonal Magic Cube, jumlah angka pada setiap diagonal ruang ini juga harus sama dengan magic number.

2.2. Hill-Climbing Algorithm

Hill-Climbing adalah salah satu teknik dalam pencarian solusi yang digunakan untuk mengoptimalkan suatu fungsi objektif. Algoritma ini bekerja dengan terus bergerak ke arah nilai yang lebih tinggi (dalam konteks pencarian nilai maksimum dari suatu fungsi objektif) atau nilai yang lebih rendah (dalam konteks pencarian minimum dari suatu fungsi biaya).

Hill-climbing mencari solusi dengan mengevaluasi *successor* dari solusi saat ini, yaitu solusi yang dihasilkan dari modifikasi kecil pada solusi sekarang. Hill-Climbing sering kali disebut sebagai pencarian lokal yang rakus karena selalu memilih langkah terbaik tanpa mempertimbangkan konsekuensi jangka panjang. Oleh karena itu, algoritma ini rentan terjebak pada local maxima (solusi lokal terbaik), ridges (rangkaian local maxima), dan plateaus (area datar tanpa peningkatan). Selain itu, karena sifatnya yang tidak mengizinkan pergerakan ke arah yang lebih buruk, Hill-Climbing dianggap sebagai pencarian yang tidak lengkap dan bisa berhenti pada solusi yang tidak optimal.

2.2.1. *Steepest Ascent Hill-climbing*

Pada algoritma *Steepest Ascent Hill-climbing,* tetangga dengan nilai fungsi objektif tertinggi dipilih pada setiap iterasi. Algoritma akan berhenti ketika tidak ada tetangga yang lebih baik dari solusi saat ini. Algoritma ini sangat efisien dalam menemukan local maxima tetapi sering kali terjebak pada titik-titik *local maximum, flat local maximum, atau shoulder*. Berikut adalah pseudocode-nya.

```
function HILL-CLIMBING(problem) returns a state that is local maximum
    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

2.2.2. *Hill-climbing with Sideways Move*

Algoritma *Hill-climbing with Sideways Move* mirip dengan *Steepest Ascent Hill-climbing,* hanya saja algoritma ini memperbolehkan untuk bergerak ke tetangga dengan nilai yang sama. Sehingga, algoritma ini dapat membantuk untuk keluar dari *flat local maximum atau shoulder.* Berikut adalah pseudocode-nya.

```
function HILL-CLIMBING(problem) returns a state that is local maximum
    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE < current.VALUE then return current.STATE
        current ← neighbor
```

2.2.3. *Random Restart Hill-climbing*

Jika algoritma terjebak di **local maxima**, varian ini memungkinkan algoritma untuk melakukan restart dari titik acak yang baru. Konsepnya sederhana: "*If at first you don't succeed, try, try again*." Algoritma ini akan melakukan pencarian ulang dari keadaan awal yang acak sampai solusi global

terbaik ditemukan. Dengan mencoba banyak permulaan, algoritma ini meningkatkan peluang untuk menemukan **global maximum**.

2.2.4. *Stochastic Hill-climbing*

Stochastic Hill-Climbing memilih tetangga secara acak dari kumpulan tetangga yang tersedia, dan jika tetangga tersebut memiliki nilai yang lebih baik, maka algoritma akan berpindah ke tetangga tersebut. Pseudocode-nya sebagai berikut:

```
function HILL-CLIMBING(problem)returns a state that is local maximum
    current ← MAKE-NODE(problem.INITIAL-STATE)
    repeat nmax times
        neighbor ← a random successor of current
        if neighbor.VALUE > current.VALUE
        then current ← neighbor
```

2.3. Algoritma *Simulated Annealing*

Simulated Annealing adalah algoritma pencarian solusi yang terinspirasi dari proses pendinginan logam dalam metalurgi. Algoritma Simulated Annealing adalah metode optimasi yang menggabungkan hill climbing dengan random walk untuk menghindari terjebak di local maxima. Pada algoritma hill-climbing pada umumnya, pergerakan selalu dilakukan ke tetangga yang lebih baik (meningkatkan nilai objektif). Namun, algoritma ini mudah terjebak pada local maxima, yaitu solusi lokal yang tampak optimal tetapi bukan solusi terbaik secara global. *Simulated annealing* memperbaiki kelemahan ini dengan mengizinkan pergerakan ke solusi yang lebih buruk, tetapi dengan probabilitas tertentu, yang menurun seiring waktu. Sehingga, algoritma ini memungkinkan untuk menjelajahi ruang solusi lebih luas dan menghindari terjebak di local maxima.

Pada setiap iterasi, algoritma bergerak menuju tetangga yang lebih baik, tetapi juga bisa pindah ke tetangga yang lebih buruk dengan probabilitas tertentu yang ditentukan oleh ekspresi $e^{\frac{\Delta E}{T}}$, di mana T adalah "temperatur" yang secara bertahap menurun selama proses

berjalan, dan ΔE adalah perbedaan antara nilai tetangga dan nilai saat ini (yaitu E = neighbor.value - current.value).

Jika tetangga lebih baik (ΔE > 0), algoritma akan berpindah ke tetangga tersebut, mirip dengan stochastic hill climbing. Namun, jika tetangga lebih buruk (ΔE < 0), algoritma tetap bisa pindah ke tetangga tersebut dengan probabilitas $e^{\frac{\Delta E}{T}}$. Saat T tinggi (misalnya E = -5, T = 100 dengan probabilitas 0,95), maka diperbolehkan untuk pindah ke tetangga, menyerupai random walk. Ketika T mendekati 0 (misalnya E = -5, T = 1 dengan probabilitas 0,007), maka tidak diperbolehkan pindah ke tetangga dan algoritma lebih menyerupai stochastic hill climbing. Berikut adalah pseudocode dari Simulated Annealing:

```
function SIMULATED-ANNEALING(problem, schedule) return a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"

    Current ← MAKE-NODE(problem.INITIAL-STATE)
    for t=1 to ∞ do
        T ← schedule(t)
        if T=0 then return current
        next ← a randomly selected successor of current
        ΔE ← next.VALUE - current.VALUE
        if ΔE > 0 then current ← next
        else current <- next only with probability e^(ΔE/T)
```

2.4. *Genetic Algorithm*

Genetic Algorithm (GA) adalah metode optimasi yang terinspirasi oleh proses evolusi alami, di mana setiap solusi dari suatu masalah direpresentasikan sebagai sebuah state atau individu. State ini biasanya direpresentasikan dalam bentuk string dengan alfabet terbatas, sering kali berupa string yang terdiri dari 0 dan 1, di mana setiap karakter mewakili satu variabel. Untuk mengevaluasi seberapa baik suatu state, digunakan fungsi evaluasi atau lebih dikenal sebagai fitness function, di mana nilai yang lebih tinggi menunjukkan state yang lebih baik. Sebagai contoh, pada masalah n-queen, fitness function dapat berupa jumlah pasangan ratu yang tidak saling menyerang. Nilai minimum fitness adalah 0, dan nilai

maksimumnya adalah jumlah pasangan ratu yang tidak saling menyerang, misalnya 28 pada masalah 8-queen.

Proses Genetic Algorithm dimulai dengan membangkitkan populasi awal yang terdiri dari k state yang dihasilkan secara acak. Setiap individu dalam populasi tersebut kemudian dievaluasi menggunakan fitness function. Setelah itu, algoritma akan menghasilkan generasi berikutnya dengan menggunakan tiga operasi utama: seleksi, crossover, dan mutasi. Seleksi dilakukan untuk memilih state (individu) yang lebih baik berdasarkan nilai fitness-nya. Selanjutnya, crossover digunakan untuk menggabungkan dua individu (parent) yang dipilih untuk membentuk individu baru (offspring), dan mutasi dilakukan dengan mengubah sedikit bagian dari offspring tersebut untuk menjaga keragaman solusi.

Secara keseluruhan, Genetic Algorithm bekerja dengan memulai dari populasi state acak, lalu secara bertahap menghasilkan generasi baru melalui kombinasi dua state parent (dengan seleksi, crossover, dan mutasi). Tujuan akhirnya adalah menemukan state dengan fitness tertinggi, yang mewakili solusi optimal dari masalah yang dihadapi. Genetic Algorithm sangat efektif dalam menjelajahi ruang solusi yang besar dan cocok untuk masalah-masalah optimasi kompleks yang memiliki banyak local optimum.

```
function GENETIC-ALGORITHM(population, FITNESS-FN) return an individual
    inputs: population, a set of individuals
            FITNESS-FN, a function that measures the fitness of an individual

    repeat
        new_population ← empty set
        for i=1 to SIZE(population) do
            x ← RANDOM-SELECTION(population, FITNESS_FN)
            y ← RANDOM-SELECTION(population, FITNESS_FN)
            child ← REPRODUCE(x,y)
            if (small random probability) then child ← MUTATE(child)
            add child to new_population
        population ← new_population
    until some individual is fit enough, or enough time has elapsed
    return the best individual in population, according to FITNESS-FN
```

BAB III
IMPLEMENTASI

3.1. Objective Function

Objective function yang akan digunakan harus dapat mengukur seberapa dekat susunan angka pada kubus terhadap kondisi optimal yang memenuhi sifat-sifat Diagonal Magic Cube. Kubus ini harus mencapai *magic number* pada setiap baris, kolom, dan diagonal.

Pendekatan dilakukan dengan menghitung perbedaan antara jumlah setiap baris, kolom, dan diagonal terhadap nilai target optimal dari Diagonal Magic Cube, yaitu 315 berdasarkan perhitungan magic number untuk kubus 5x5x5 pada landasan teori. Sehingga, objective function dapat dirumuskan sebagai jumlah selisih mutlak antar jumlah tiap baris, kolom, dan diagonal dengan 315. Nilai objective function yang semakin mendekati 0, menandakan semakin mendekati kondisi susunan kubus yang optimal terhadap Diagonal Magic Cube. Berikut adalah bentuk perumusannya:

$$f(value) = -(|\sum(315 - jumlah\ baris)| + |\sum(315 - jumlah\ kolom)|$$

$$+ |\Sigma(315 - jumlah\ diagonal\ sisi)| + |\Sigma(315 - jumlah\ diagonal\ ruang)|)$$

Nilai fungsi yang mendekati 0 menunjukkan bahwa susunan angka pada kubus semakin mendekati kondisi optimal. Pendekatan *objective function* ini sensitif terhadap perubahan kecil dalam susunan angka dan dapat mendeteksi perbaikan secara bertahap.

3.2. Implementasi Algoritma

3.2.1. Algoritma Base Algorithm

Base Algorithm merupakan kelas dasar yang dirancang untuk menyediakan kerangka umum bagi semua algoritma pencarian local search yang diimplementasikan dalam proyek ini. Kelas ini membantu untuk mengurangi redundansi kode dan memastikan bahwa setiap algoritma yang menggunakannya memiliki struktur yang konsisten dan mudah diikuti. Dengan menggunakan pendekatan berbasis objek, kelas Base Algorithm memungkinkan setiap algoritma turunan untuk fokus pada logika spesifiknya tanpa perlu mengimplementasikan ulang fungsi-fungsi dasar.

```python
from abc import ABC, abstractmethod
import numpy as np
from utils.data_processing import evaluate_cube
import time

class BaseAlgorithm(ABC):
    def __init__(self, cube: np.ndarray, max_iter: int):
        self.init_cube = cube.copy()
        self.cube = cube.copy()
        self.max_iter = max_iter
        self.iter = 0
        self.tracker = []
        self.start_time = None
        self.end_time = None

    @abstractmethod
    def run(self):
        """ Implement for each algorithm """
        pass

    def evaluate(self) -> int:
        return evaluate_cube(self.cube)

    def track_value(self):
        value = self.evaluate()
        self.tracker.append(value)

    def get_results(self) -> dict:
        duration = self.end_time - self.start_time
        results = {
            "initial_state": self.init_cube,
            "final_state" : self.cube,
            "initial_value": evaluate_cube(self.init_cube),
            "final_value": self.evaluate(),
            "iterations": self.iter,
            "tracker": self.tracker,
            "duration": duration
        }
        return results

    def reset(self):
        self.cube = self.init_cube.copy()
        self.iter = 0
        self.tracker = []
        self.start_time = None
        self.end_time = None

```

Gambar 3.2.1. Source Code Base Algorithm

3.2.2. Algoritma Data Processing

Algoritma Data Processing berisi fungsi-fungsi untuk mempersiapkan, mengelola, dan memproses data yang digunakan dalam algoritma local search yang melibatkan inisialisasi state awal dari cube, evaluasi terhadap state cube, dan pemeriksaan apakah konfigurasi cube saat ini telah mencapai solusi yang optimal.

```python
def init_cube(n: int) -> np.ndarray:
    numbers = np.arange(1, n**3 + 1)
    np.random.shuffle(numbers)
    return numbers.reshape((n, n, n))

def get_rows(cube: np.ndarray) -> np.ndarray:
    n = cube.shape[0]
    rows = []
    for layer in range(n):
        for row in range(n):
            rows.append(cube[layer, row, :])
    return np.array(rows)

def get_cols(cube: np.ndarray) -> np.ndarray:
    n = cube.shape[0]
    cols = []
    for layer in range(n):
        for col in range(n):
            cols.append(cube[layer, :, col])
    return np.array(cols)

def get_pillars(cube: np.ndarray) -> np.ndarray:
    n = cube.shape[0]
    pills = []
    for row in range(n):
        for col in range(n):
            pills.append(cube[:, row, col])
    return np.array(pills)
```

Gambar 3.2.2.1. Source Code Data Processing (1)

```python
def get_side_diagonals(cube: np.ndarray) -> np.ndarray:
    n = cube.shape[0]
    diagonals = []

    # diagonal sisi tiap layer
    for layer in range(n):
        diagonals.append([cube[layer, i, i] for i in range(n)])
        diagonals.append([cube[layer, i, n-i-1] for i in range(n)])

    # diagonal sisi tiap baris
    for row in range(n):
        diagonals.append([cube[i, row, i] for i in range(n)])
        diagonals.append([cube[i, row, n-i-1] for i in range(n)])

    # diagonal sisi tiap kolom
    for col in range(n):
        diagonals.append([cube[i, i, col] for i in range(n)])
        diagonals.append([cube[i, n-i-1, col] for i in range(n)])

    return np.array(diagonals)

def get_space_diagonals(cube: np.ndarray) -> np.ndarray:
    n = cube.shape[0]
    diagonals = []
    diagonals.append([cube[i, i, i] for i in range(n)])
    diagonals.append([cube[i, i, n-i-1] for i in range(n)])
    diagonals.append([cube[i, n-i-1, i] for i in range(n)])
    diagonals.append([cube[i, n-i-1, n-i-1] for i in range(n)])
    return np.array(diagonals)

def evaluate_cube(cube: np.ndarray) -> int:
    n = cube.shape[0]
    magic_number = 1/2 * n * (n**3 + 1)
    h = 0

    rows = get_rows(cube)
    cols = get_cols(cube)
    pills = get_pillars(cube)
    side_diagonals = get_side_diagonals(cube)
    space_diagonals = get_space_diagonals(cube)

    for component in [rows, cols, pills, side_diagonals, space_diagonals]:
        for c in component:
            h -= abs(np.sum(c) - magic_number)

    return h
```

Gambar 3.2.2.2. Source Code Data Processing (2)

```python
def swap(cube: np.ndarray, a: tuple, b: tuple) -> np.ndarray:
    cube = cube.copy()
    cube[a], cube[b] = cube[b], cube[a]
    return cube

def random_swap(cube: np.ndarray) -> np.ndarray:
    n = cube.shape[0]
    a = np.random.randint(0, n, 3)
    b = np.random.randint(0, n, 3)
    while np.array_equal(a, b):
        b = np.random.randint(0, n, 3)
    return swap(cube, tuple(a), tuple(b))

def best_neighbor(cube: np.ndarray) -> np.ndarray:
    n = cube.shape[0]
    best_neighbor = cube.copy()
    best_value = evaluate_cube(swap(cube, (0, 0, 0), (0, 0, 1)))

    indices = [(i, j, k) for i in range(n) for j in range(n) for k in range(n)]
    for (i, j, k), (x, y, z) in combinations(indices, 2):
        neighbor = swap(cube, (i, j, k), (x, y, z))
        neighbor_value = evaluate_cube(neighbor)
        if neighbor_value > best_value:
            best_neighbor = neighbor
            best_value = neighbor_value

    return best_neighbor

def is_goal(cube: np.ndarray) -> bool:
    return evaluate_cube(cube) == 0
```

Gambar 3.2.2.3. Source Code Data Processing (3)

3.2.3. Algoritma *Steepest Ascent Hill-Climbing*

Algoritma ini melakukan pencarian menggunakan metode *Steepest Ascent Hill-Climbing*. Maksimal iterasi dilakukan sebagai masukan karena permasalahan studi kasus ini yang sangat kompleks sehingga untuk menunggu benar-benar diselesaikan membutuhkan waktu yang sangat lama dan terdapat keterbatasan komputasional, yang mana juga berlaku untuk algoritma *Hill-climbing with Sideways Move* dan *Random Restart Hill-climbing*. Pada *Steepest Ascent*, *neighbor* dipilih berdasarkan nilai *objective function* terbaik dari semua kemungkinan *neighbor*. Pencarian akan berhenti ketika nilai *objective function* dari *neighbor* tidak lebih besar dari nilai saat ini.

```python
from algorithms.base_algorithm import BaseAlgorithm
from utils.data_processing import best_neighbor, evaluate_cube
import time

class SteepestAscent(BaseAlgorithm):
    def run(self):
        self.start_time = time.time()
        self.iter = 0
        while self.iter < self.max_iter:
            self.track_value()
            neighbor = best_neighbor(self.cube)
            if evaluate_cube(neighbor) > self.evaluate():
                self.cube = neighbor
            else:
                break
            self.iter += 1
        self.end_time = time.time()
```

Gambar 3.2.3. Source Code Steepest Ascent Hill-Climbing

3.2.4. Algoritma *Hill-Climbing with Sideways Move*

Algoritma ini melakukan pencarian menggunakan metode *Hill-Climbing with Sideways Move* dengan variabel iterasi maksimal dan jumlah *sideways move* maksimal berdasarkan masukan fungsi. Algoritma ini mirip dengan *Steepest Ascent*, namun memperbolehkan *neighbor* yang nilai *objective function*-nya sama untuk dipilih, dengan batasan maksimal yang telah ditentukan tadi.

```python
from algorithms.base_algorithm import BaseAlgorithm
from utils.data_processing import best_neighbor, evaluate_cube
import time

class SidewaysMove(BaseAlgorithm):
    def __init__(self, cube, max_iter, max_sideways_moves):
        super().__init__(cube, max_iter)
        self.max_sideways_moves = max_sideways_moves

    def run(self):
        self.start_time = time.time()
        self.iter = 0
        sideways_count = 0

        while self.iter < self.max_iter:
            self.track_value()
            neighbor = best_neighbor(self.cube)

            if evaluate_cube(neighbor) > self.evaluate():
                self.cube = neighbor
                sideways_count = 0
            elif evaluate_cube(neighbor) == self.evaluate():
                self.cube = neighbor
                sideways_count += 1
                if sideways_count >= self.max_sideways_moves:
                    break
            else:
                break

            self.iter += 1

        self.end_time = time.time()
```

Gambar 3.2.4. Source Code Hill-Climbing with Sideways Move

3.2.5. Algoritma *Random Restart Hill-Climbing*

Algoritma ini melakukan pencarian menggunakan metode *Random Restart Hill-Climbing* dengan variabel iterasi maksimal dan jumlah *restart* maksimal berdasarkan masukan fungsi. Algoritma ini merupakan algoritma *Steepest Ascent* yang dieksekusi berulang-ulang sebanyak jumlah *restart* masukan.

```python
import time
from algorithms.base_algorithm import BaseAlgorithm
from utils.data_processing import is_goal, init_cube
from algorithms.hill_climbing.steepest_ascent import SteepestAscent

class RandomRestart(BaseAlgorithm):
    def __init__(self, cube, max_iter, max_restart):
        super().__init__(cube, max_iter)
        self.max_restart = max_restart
        self.results = []
        self.restarts = 0

    def run(self):
        self.start_time = time.time()
        self.iter = 0
        steepest_algo = SteepestAscent(self.cube, self.max_iter)

        for _ in range(self.max_restart):
            steepest_algo.run()
            self.results.append(steepest_algo)
            if (is_goal(steepest_algo.cube)):
                break
            else:
                self.cube = init_cube(self.cube.shape[0])
                steepest_algo = SteepestAscent(self.cube, self.max_iter)
                self.restarts += 1

        m = self.results[0]
        m_val = m.evaluate()
        for result in self.results:
            self.tracker += result.tracker
            self.iter += result.iter

            res_val = result.evaluate()
            if (res_val > m_val):
                m = result
                m_val = res_val

        self.cube = m.cube
        self.end_time = time.time()

    def get_results(self):
        result = super().get_results()
        result.update({'restarts': self.restarts,
                       'iter_per_restart': [e.iter for e in self.results]
                       })
        return result


```

Gambar 3.2.5. Source Code Random Restart Hill-Climbing

3.2.6. Algoritma *Stochastic Hill-Climbing*

Algoritma ini melakukan pencarian menggunakan metode *Stochastic Hill-Climbing* dengan iterasi maksimal berdasarkan masukan fungsi. Berbeda dengan ketiga algoritma *Hill-Climbing* sebelumnya, *neighbor* langsung dipilih secara random. Pencarian akan berhenti ketika nilai *objective function* dari *neighbor* tidak lebih besar dari nilai saat ini.

```python
from algorithms.base_algorithm import BaseAlgorithm
from utils.data_processing import evaluate_cube, random_swap
import time

class Stochastic(BaseAlgorithm):
    def run(self):
        self.start_time = time.time()
        self.iter = 0
        while self.iter < self.max_iter:
            c_val = self.evaluate()
            self.tracker.append(c_val)

            neighbor = random_swap(self.cube)
            n_val = evaluate_cube(neighbor)

            if c_val < n_val:
                self.cube = neighbor
            if n_val == 0:
                break

            self.iter += 1
        self.end_time = time.time()

```

Gambar 3.2.6. Source Code Stochastic Hill-Climbing

3.2.7. Algoritma *Simulated Annealing*

Algoritma ini melakukan pencarian menggunakan metode *Simulated Annealing.* Algoritma memiliki parameter tambahan, yaitu suhu awal (*initial_temp*), dan laju pendinginan (*cooling_rate*). Suhu awal (*temperature*) digunakan untuk menentukan probabilitas penerimaan solusi suboptimal, yang akan berkurang seiring waktu mengikuti *cooling rate*. Suhu diperbarui setiap iterasi dengan mengalikan suhu saat ini dengan *cooling_rate*. Pada setiap iterasi, algoritma memilih tetangga acak dari kubus (dengan menggunakan *random_swap*) dan menghitung perbedaan nilai objektif (*delta_e*). Jika tetangga memiliki nilai yang lebih baik, maka akan langsung diterima. Namun, jika nilai tetangga lebih buruk, solusi tersebut masih bisa diterima dengan probabilitas tertentu yang berkurang seiring penurunan suhu. Algoritma juga melacak frekuensi "stuck" di lokal optima (dengan *stuck_freq*) ketika tidak ada perbaikan yang terjadi dalam sejumlah iterasi tertentu (*stuck_threshold*).

```python
import numpy as np
import math
from algorithms.base_algorithm import BaseAlgorithm
from utils.data_processing import evaluate_cube, random_swap
import time

class SimulatedAnnealing(BaseAlgorithm):
    def __init__(self, cube: np.ndarray, max_iter: int, initial_temp: float, cooling_rate: float):
        super().__init__(cube, max_iter)
        self.initial_temp = initial_temp
        self.cooling_rate = cooling_rate
        self.prob_values = []
        self.stuck_freq = 0

    def run(self):
        self.start_time = time.time()
        self.iter = 0
        temperature = self.initial_temp
        stuck_threshold = 10
        consec_stuck = 0

        while self.iter < self.max_iter and temperature > 0:
            self.track_value()

            neighbor = random_swap(self.cube)
            delta_e = evaluate_cube(neighbor) - self.evaluate()

            if delta_e > 0:
                # Accept the neighbor if it has a higher value
                self.cube = neighbor
                consec_stuck = 0
            else:
                # Accept the neighbor with a certain probability
                probability = math.exp(delta_e / temperature)
                self.prob_values.append(probability)

                if np.random.rand() < probability:
                    self.cube = neighbor
                    consec_stuck = 0
                else:
                    consec_stuck += 1
            # Increase stuck frequency if stuck for a certain number of iterations
            if consec_stuck >= stuck_threshold:
                self.stuck_freq += 1
                consec_stuck = 0
            # Update temperature
            temperature *= self.cooling_rate
            self.iter += 1
        self.end_time = time.time()

    def get_results(self) -> dict:
        results = super().get_results()
        results.update({
            "prob_values": self.prob_values,
            "stuck_freq": self.stuck_freq
        })
        return results
```

Gambar 3.2.7. Source Code Simulated Annealing

3.2.8. Algoritma *Genetic*

Algoritma ini melakukan pencarian menggunakan metode *Genetic Algorithm.* Algoritma genetika ini dimulai dengan menginisialisasi populasi awal yang terdiri dari beberapa state acak sesuai dengan ukuran populasi yang ditentukan (population size). Setiap state dalam populasi tersebut kemudian dievaluasi menggunakan fungsi fitness, yang menghitung jumlah komponen dalam magic cube (seperti baris, kolom, diagonal bidang, dan diagonal ruang) yang memiliki jumlah angka sebesar 315.

Proses seleksi *parent* dilakukan melalui metode *random wheel*, di mana peluang terpilihnya suatu state sebagai *parent* sebanding dengan nilai *fitness*-nya. Dengan cara ini, state yang memiliki nilai *fitness* lebih tinggi memiliki kemungkinan lebih besar untuk dipilih sebagai *parent*. Setelah itu, dua *parents* dipilih secara acak dan diterapkan operasi *crossover*, yaitu menukar sebagian angka dari masing-masing orang tua dari posisi tertentu hingga akhir untuk menghasilkan child state(s). Selanjutnya, mutasi diterapkan pada beberapa *child* dengan metode *random swap* untuk menukar posisi angka secara acak, yang bertujuan memperkenalkan variasi dalam populasi.

Setelah proses *crossover* dan mutasi, *child* yang dihasilkan membentuk populasi baru yang dievaluasi kembali di iterasi berikutnya. Pada setiap generasi, algoritma mencatat fungsi tujuan terbaik (*best objective function*) yang ditemukan. Algoritma akan terus berulang hingga ditemukan individu yang memenuhi kriteria solusi optimal berdasarkan fungsi fitness, atau hingga batas maksimum iterasi (*max_iter*) tercapai.

```python
import numpy as np
from algorithms.base_algorithm import BaseAlgorithm
from utils.data_processing import init_cube, random_swap, get_rows, get_cols, get_pillars, get_side_diagonals, get_space_diagonals
import random
import time

class GeneticAlgorithm(BaseAlgorithm):
    def __init__(self, cube: np.ndarray, max_iter: int, population_size: int, mutation_rate: float):
        """ Initialize genetic algorithm """
        super().__init__(cube, max_iter)
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.population = self.init_population()

    def init_population(self) -> list:
        """ Generate initial population """
        return [init_cube(self.cube.shape[0]) for _ in range(self.population_size)]

    def fitness_function(self, cube: np.ndarray) -> int:
        """ Fitness function to evaluate individual """
        n = cube.shape[0]
        magic_number = 1/2 * n * (n**3 + 1)
        fitness = 0

        rows = get_rows(cube)
        cols = get_cols(cube)
        pills = get_pillars(cube)
        side_diags = get_side_diagonals(cube)
        space_diags = get_space_diagonals(cube)

        for component in [rows, cols, pills, side_diags, space_diags]:
            for c in component:
                if np.sum(c) == magic_number:
                    fitness += 1

        return fitness

    def is_solution(self, individual: np.ndarray) -> bool:
        """ Check if individual is solution """
        n = individual.shape[0]
        return self.fitness_function(individual) == (3*(n**2) + 6*n + 4)

    def evaluate_population(self) -> list:
        """ Evaluate all individuals in population """
        return [self.fitness_function(individual) for individual in self.population]

    def select_parents(self) -> tuple:
        """ Select parents based on fitness value """
        fitness_val = self.evaluate_population()
        total_fitness = sum(fitness_val)
        prob = [val / total_fitness for val in fitness_val]
        parents = np.random.choice(self.population_size, size=2, p=prob, replace=False)
        return self.population[parents[0]], self.population[parents[1]]

```

Gambar 3.2.8.1. Source Code Genetic Algorithm (1)

```python

    def crossover(self, parent1, parent2) -> np.ndarray:
        """ Crossover between two parents """
        point = np.random.randint(parent1.size)
        child = parent1.copy()
        child.flat[point:] = parent2.flat[point:]
        return child.reshape(parent1.shape)

    def mutate(self, individual: np.ndarray) -> np.ndarray:
        """ Mutate individual with random swap """
        if random.random() < self.mutation_rate:
            return random_swap(individual)
        return individual

    def run(self):
        """ Run genetic algorithm """
        self.start_time  = time.time()
        self.iter = 0
        best_individual = None
        best_fitness = 0
        found_sol = False

        while self.iter < self.max_iter and not found_sol:
            new_population = []
            fitness_vals = self.evaluate_population()
            pop_best_fitness = max(fitness_vals)
            if pop_best_fitness > best_fitness:
                best_fitness = pop_best_fitness
                best_individual = self.population[np.argmax(fitness_vals)]
                self.cube = best_individual
            # Check if solution is found
            if self.is_solution(best_individual):
                found_sol = True
                break

            self.track_value()
            # Generate new population
            for _ in range(self.population_size // 2):
                parent1, parent2 = self.select_parents()
                child1 = self.crossover(parent1, parent2)
                child2 = self.crossover(parent2, parent1)
                mutate_child1 = self.mutate(child1)
                mutate_child2 = self.mutate(child2)
                new_population.extend([mutate_child1, mutate_child2])
            self.population = new_population
            self.iter += 1

        self.cube = best_individual
        self.end_time = time.time()

    def get_results(self) -> dict:
        """ Get results of genetic algorithm """
        results = super().get_results()
        results.update({
            "population_size": self.population_size,
        })
        return results

```

Gambar 3.2.8.2. Source Code Genetic Algorithm (2)

3.2.9. Algoritma Visualisasi

Algoritma ini menyediakan fungsi untuk menampilkan dan memvisualisasikan proses dan hasil eksperimen dari algoritma optimasi pada kubus (*state tracking*). Fungsi `display_cube` menampilkan tiap layer dari kubus pada setiap state, sedangkan `display_all_states` mencetak semua state yang dilacak selama iterasi. Dua fungsi plotting, `plot_obj_vs_iter` dan `plot_prob_vs_iter`, digunakan untuk menggambarkan grafik nilai fungsi objektif dan probabilitas (untuk *Simulated Annealing*) terhadap iterasi. Fungsi `visualize_experiment` menggabungkan semua hasil eksperimen, menampilkan informasi state awal dan akhir dari kubus, serta hasil analisis statistik seperti nilai fungsi objektif awal/akhir, jumlah iterasi, dan durasi eksperimen.

```python
import matplotlib.pyplot as plt
import numpy as np

def display_cube(cube: np.ndarray, title: str = "State"):
    """ Display the cube """
    print(title)
    print("----------")
    for layer in cube:
        print(layer)
        print()

def display_all_states(state_tracker: list):
    """ Display all states in the state tracker """
    print("Displaying all states:")
    for i, state in enumerate(state_tracker):
        display_cube(state, title=f"State at Iteration {i+1}")
        print()

def plot_obj_vs_iter(tracker: list, title: str = "Objective Function vs Iterations"):
    """ Plot the objective function vs iterations """
    plt.figure()
    plt.plot(range(len(tracker)), tracker, marker="o", linestyle="-", color="b")
    plt.xlabel("Iterations")
    plt.ylabel("Objective Function Value")
    plt.title(title)
    # plt.show()

def plot_prob_vs_iter(prob_values: list, title="Probability (e^ΔE/T) vs Iterations"):
    """ Plot the probability values vs iterations """
    plt.figure()
    plt.plot(range(len(prob_values)), prob_values, marker="x", linestyle="--", color="g")
    plt.xlabel("Iterations")
    plt.ylabel("Probability (e^ΔE/T)")
    plt.title(title)
    # plt.show()

def visualize_experiment(results, algo_name="Algorithm"):
    """ Visualize the results of the experiment """
    print(f"\nVisualization for {algo_name}\n")
    # Display the initial and final state of the cube
    display_cube(results["initial_state"], title="Initial State")
    display_cube(results["final_state"], title="Final State")
    # Plot the objective function vs iterations
    plot_obj_vs_iter(results["tracker"], title=f"{algo_name} - Objective Function vs Iterations")
    if "state_tracker" in results:
        display_all_states(results["state_tracker"])
    # Print the results
    print(f"Initial Objective Value: {results["initial_value"]}")
    print(f"Final Objective Value: {results["final_value"]}")
    print(f"Iterations: {results["iterations"]}")
    print(f"Duration: {results["duration"]}")
    # Print additional results of Simulated Annealing Algorithm
    if "prob_values" in results and "stuck_freq" in results:
        plot_prob_vs_iter(results["prob_values"], title=f"{algo_name} - Probability (e^ΔE/T) vs Iterations")
        print(f"Stuck Frequency (Local Optima): {results["stuck_freq"]}")
    # Print additional results of Genetic Algorithm
    if "population_size" in results:
        print(f"Population Size: {results["population_size"]}")
    # Print additional results of Random Restart Algorithm
    if "restarts" in results and "iter_per_restart" in results:
        print(f"Restarts: {results["restarts"]}")
        print(f"Iterations per Restart: {results["iter_per_restart"]}")
    plt.show()
```

Gambar 3.2.9. Source Code Algoritma Visualisasi

BAB IV
HASIL EKSPERIMEN DAN ANALISIS

4.1. Hasil Eksperimen Algoritma Local Search

Seluruh eksperimen dilakukan dengan *initial state* kubus dengan kondisi yang sama dan komputer yang sama untuk mendapatkan perbandingan data yang baik.

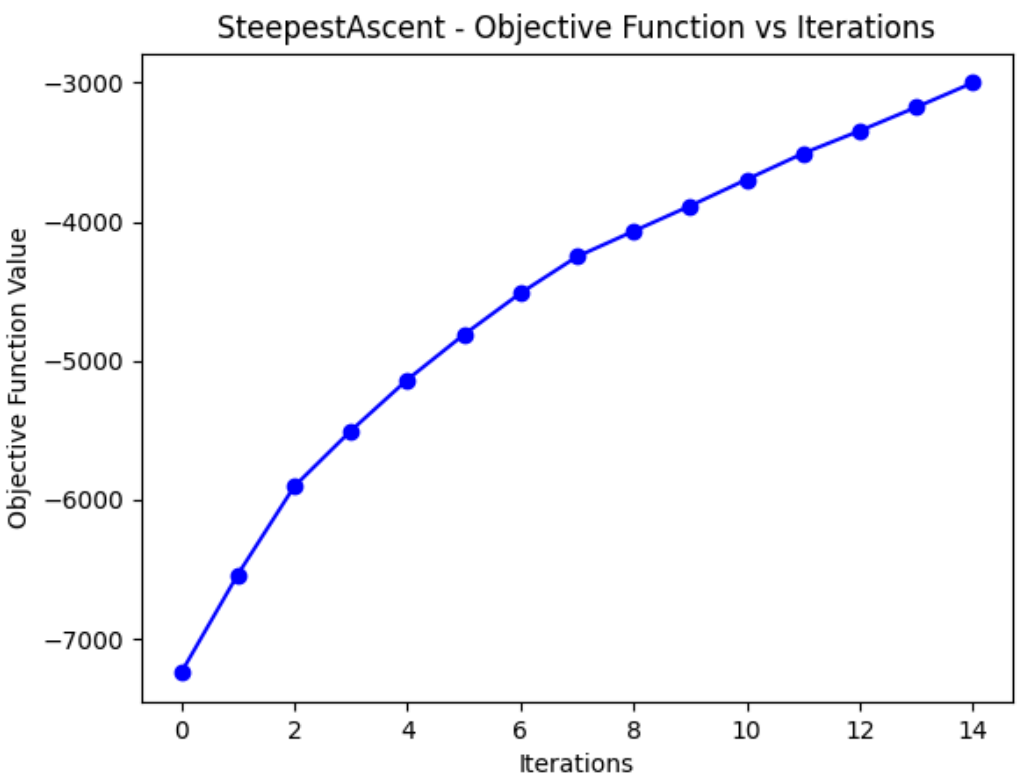
4.1.1. *Steepest Ascent Hill-climbing*

Eksperimen 1 - *Steepest Ascent Hill-climbing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[17 49 25 115 83] [85 103 29 37 61] [51 14 96 86 67] [90 71 120 1 44] [13 69 26 77 106]] [[122 74 80 8 48] [93 15 6 52 39] [47 87 50 64 62] [38 94 45 88 111] [40 5 110 117 33]] [[30 46 79 124 105] [10 121 58 7 109] [123 75 60 34 28] [76 16 73 100 68] [118 56 43 72 20]] [[57 22 21 95 78] [42 41 89 112 35] [125 59 82 3 65] [18 99 101 24 55] [66 98 23 119 70]] [[92 53 104 9 36] [84 19 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 11 91]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-2840
Iterations	15
Duration	60.265127420425415 seconds

Grafik:

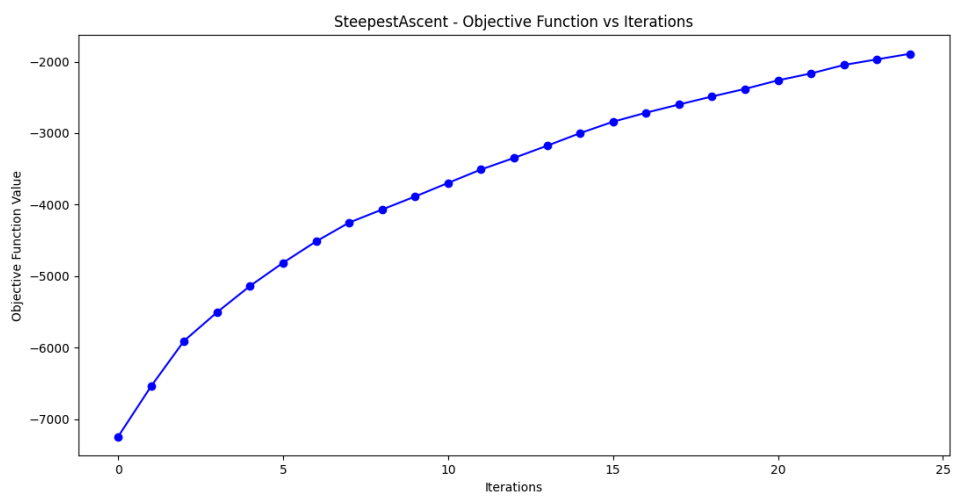


Eksperimen 2 - *Steepest Ascent Hill-climbing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[44 49 25 115 70] [85 103 29 37 61] [51 14 96 86 67] [112 71 120 1 17] [38 69 26 77 106]] [[122 74 80 8 48] [93 15 45 52 39] [47 87 50 64 62] [13 94 6 88 111] [40 30 110 117 33]] [[5 46 58 104 105] [10 121 79 7 109] [123 75 60 34 28] [76 16 73 100 68] [118 56 43 72 20]] [[57 98 21 95 55] [42 41 89 90 35] [125 59 82 3 65] [18 99 101 24 78] [66 22 23 119 83]] [[92 53 124 9 36] [84 19 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 11 91]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1829
Iterations	25
Duration	107.32665252685547 seconds

Grafik:

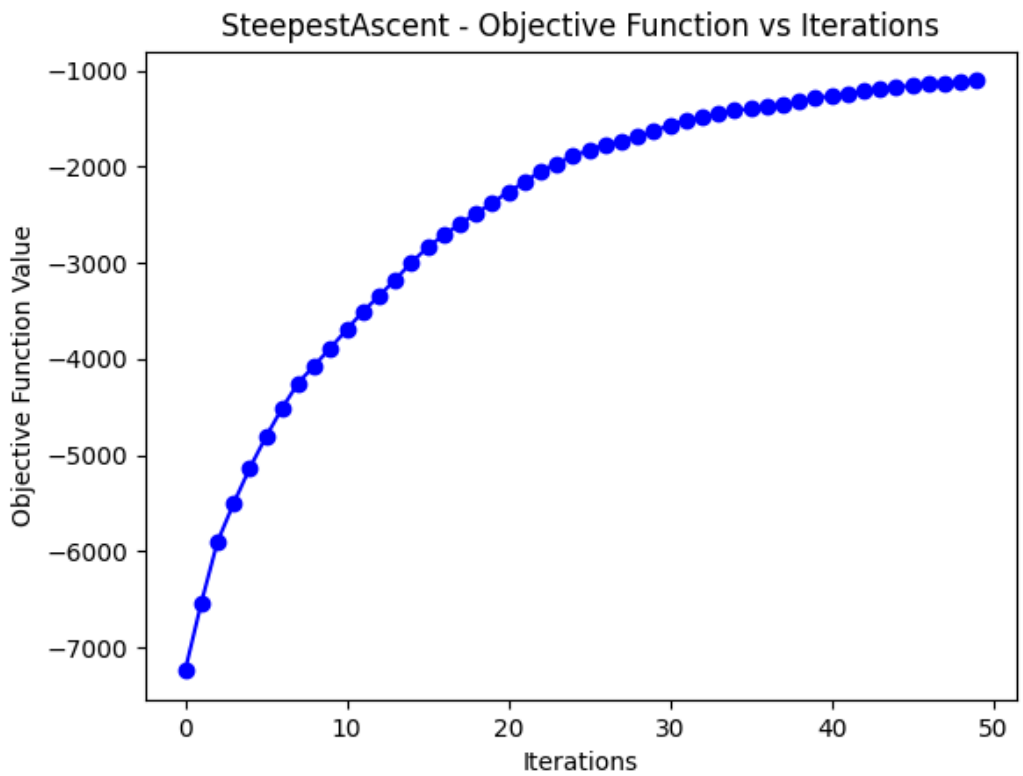


Eksperimen 3 - *Steepest Ascent Hill-climbing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[44 49 25 122 70] [85 103 29 37 61] [31 14 109 86 67] [112 71 120 1 12] [28 76 26 77 106]] [[115 74 80 8 39] [93 33 65 68 48] [34 87 50 79 62] [13 94 6 88 116] [63 30 110 96 15]] [[5 42 58 104 105] [10 121 64 7 117] [123 56 72 40 22] [69 16 73 100 52] [118 75 43 60 20]] [[57 98 21 81 55] [46 41 101 90 35] [125 59 82 3 45] [18 78 89 24 99] [66 38 23 114 83]] [[92 53 124 9 36] [84 19 54 113 47] [4 97 2 107 119] [102 51 27 111 17] [32 95 108 11 91]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1097
Iterations	50
Duration	219.64201498031616 seconds

Grafik:



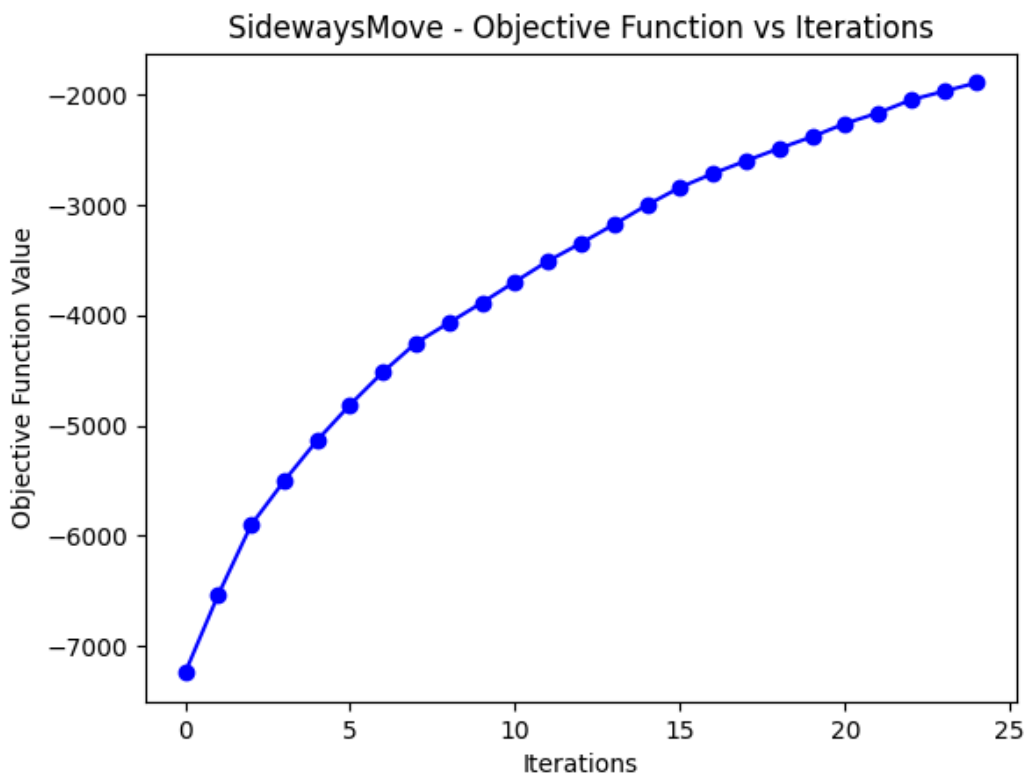
4.1.2. *Hill-climbing with Sideways Move*

Eksperimen 1 - *Hill-climbing with Sideways Move*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[44 49 25 115 70] [85 103 29 37 61] [51 14 96 86 67] [112 71 120 1 17] [38 69 26 77 106]] [[122 74 80 8 48] [93 15 45 52 39] [47 87 50 64 62] [13 94 6 88 111] [40 30 110 117 33]] [[5 46 58 104 105] [10 121 79 7 109] [123 75 60 34 28] [76 16 73 100 68] [118 56 43 72 20]] [[57 98 21 95 55] [42 41 89 90 35] [125 59 82 3 65] [18 99 101 24 78] [66 22 23 119 83]] [[92 53 124 9 36] [84 19 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 11 91]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1829
Iterations	25
Duration	112.66550850868225 seconds

Grafik:

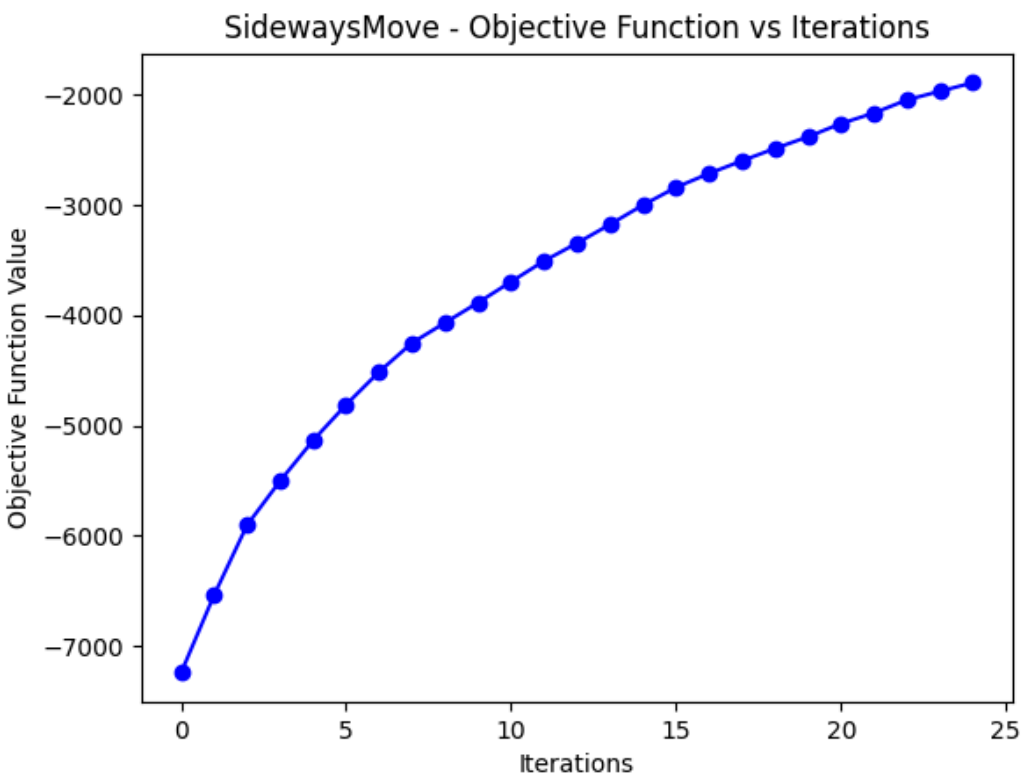


Eksperimen 2 - *Hill-climbing with Sideways Move*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[44 49 25 115 70] [85 103 29 37 61] [51 14 96 86 67] [112 71 120 1 17] [38 69 26 77 106]] [[122 74 80 8 48] [93 15 45 52 39] [47 87 50 64 62] [13 94 6 88 111] [40 30 110 117 33]] [[5 46 58 104 105] [10 121 79 7 109] [123 75 60 34 28] [76 16 73 100 68] [118 56 43 72 20]] [[57 98 21 95 55] [42 41 89 90 35] [125 59 82 3 65] [18 99 101 24 78] [66 22 23 119 83]] [[92 53 124 9 36] [84 19 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 11 91]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1829
Iterations	25
Duration	111.94457077980042 seconds

Grafik:

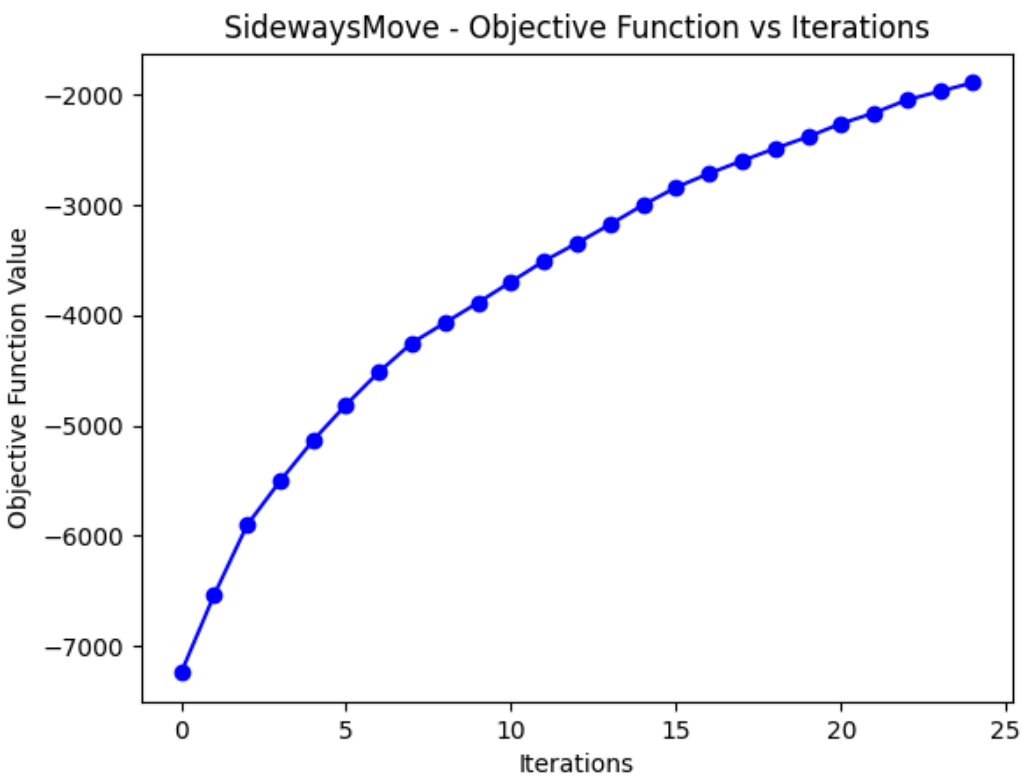


Eksperimen 3 - *Hill-climbing with Sideways Move*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[44 49 25 115 70] [85 103 29 37 61] [51 14 96 86 67] [112 71 120 1 17] [38 69 26 77 106]] [[122 74 80 8 48] [93 15 45 52 39] [47 87 50 64 62] [13 94 6 88 111] [40 30 110 117 33]] [[5 46 58 104 105] [10 121 79 7 109] [123 75 60 34 28] [76 16 73 100 68] [118 56 43 72 20]] [[57 98 21 95 55] [42 41 89 90 35] [125 59 82 3 65] [18 99 101 24 78] [66 22 23 119 83]] [[92 53 124 9 36] [84 19 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 11 91]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1829
Iterations	25
Duration	108.29212045669556 seconds

Grafik:



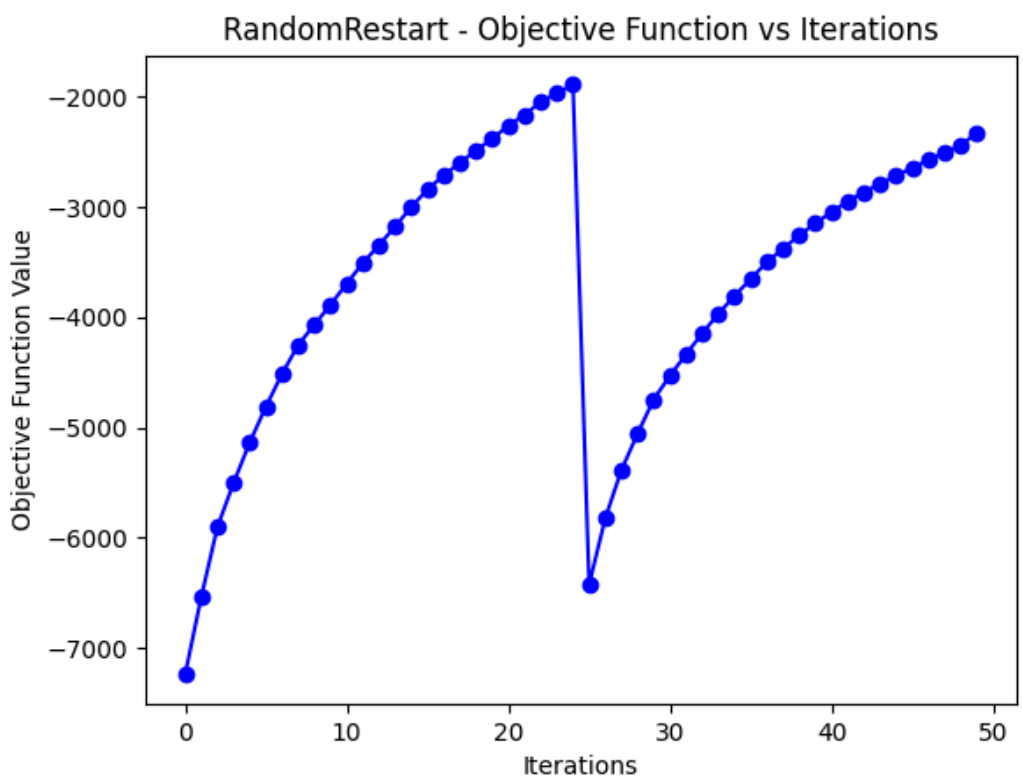
4.1.3. *Random Restart Hill-climbing*

Eksperimen 1 - *Random Restart Hill-climbing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[44 49 25 115 70] [85 103 29 37 61] [51 14 96 86 67] [112 71 120 1 17] [38 69 26 77 106]] [[122 74 80 8 48] [93 15 45 52 39] [47 87 50 64 62] [13 94 6 88 111] [40 30 110 117 33]] [[5 46 58 104 105] [10 121 79 7 109] [123 75 60 34 28] [76 16 73 100 68] [118 56 43 72 20]] [[57 98 21 95 55] [42 41 89 90 35] [125 59 82 3 65] [18 99 101 24 78] [66 22 23 119 83]] [[92 53 124 9 36] [84 19 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 11 91]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1829
Iterations	50
Duration	205.75057816505432 seconds
Restarts	2
Iteration per Restarts	25

Grafik:

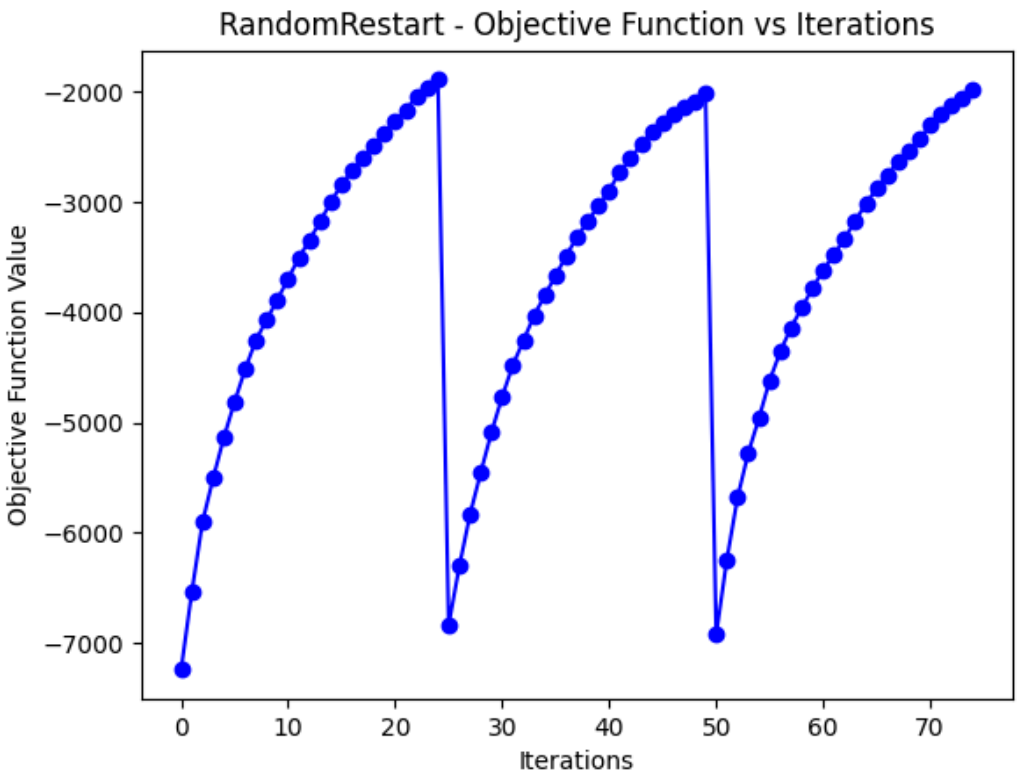


Eksperimen 2 - *Random Restart Hill-climbing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[44 49 25 115 70] [85 103 29 37 61] [51 14 96 86 67] [112 71 120 1 17] [38 69 26 77 106]] [[122 74 80 8 48] [93 15 45 52 39] [47 87 50 64 62] [13 94 6 88 111] [40 30 110 117 33]] [[5 46 58 104 105] [10 121 79 7 109] [123 75 60 34 28] [76 16 73 100 68] [118 56 43 72 20]] [[57 98 21 95 55] [42 41 89 90 35] [125 59 82 3 65] [18 99 101 24 78] [66 22 23 119 83]] [[92 53 124 9 36] [84 19 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 11 91]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1829
Iterations	75
Duration	469.6200747489929 seconds
Restarts	3
Iteration per Restarts	25

Grafik:

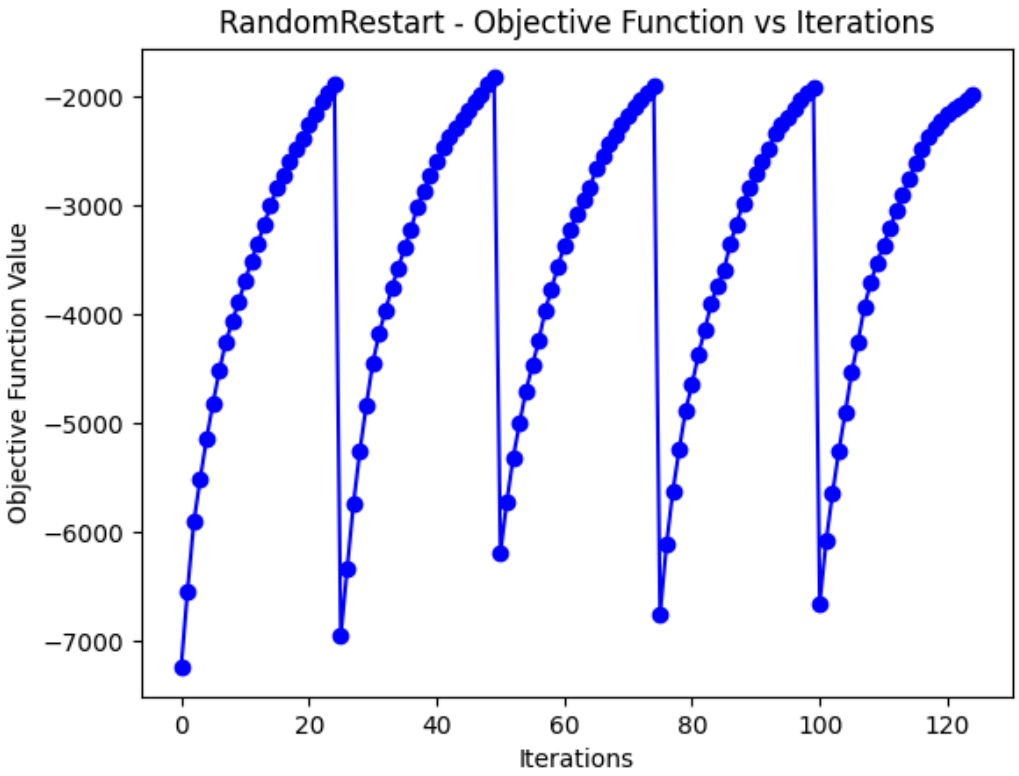


Eksperimen 3 - *Random Restart Hill-climbing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[84 81 99 2 98] [61 107 118 31 4] [25 65 22 117 70] [80 55 64 38 79] [69 24 7 110 72]] [[88 13 95 101 5] [59 56 28 51 125] [3 67 100 27 114] [50 87 23 112 42] [122 89 73 11 18]] [[33 91 76 85 8] [47 35 43 105 92] [115 48 82 53 26] [103 74 12 29 109] [14 78 113 60 102]] [[41 30 6 106 121] [16 75 90 77 52] [120 124 1 49 15] [21 46 94 86 36] [83 32 119 9 93]] [[71 97 66 17 62] [123 45 34 68 44] [20 19 108 57 111] [63 54 96 37 58] [40 104 10 116 39]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1829
Iterations	125
Duration	512.2493891716003 seconds
Restarts	5
Iteration per Restarts	25

Grafik:



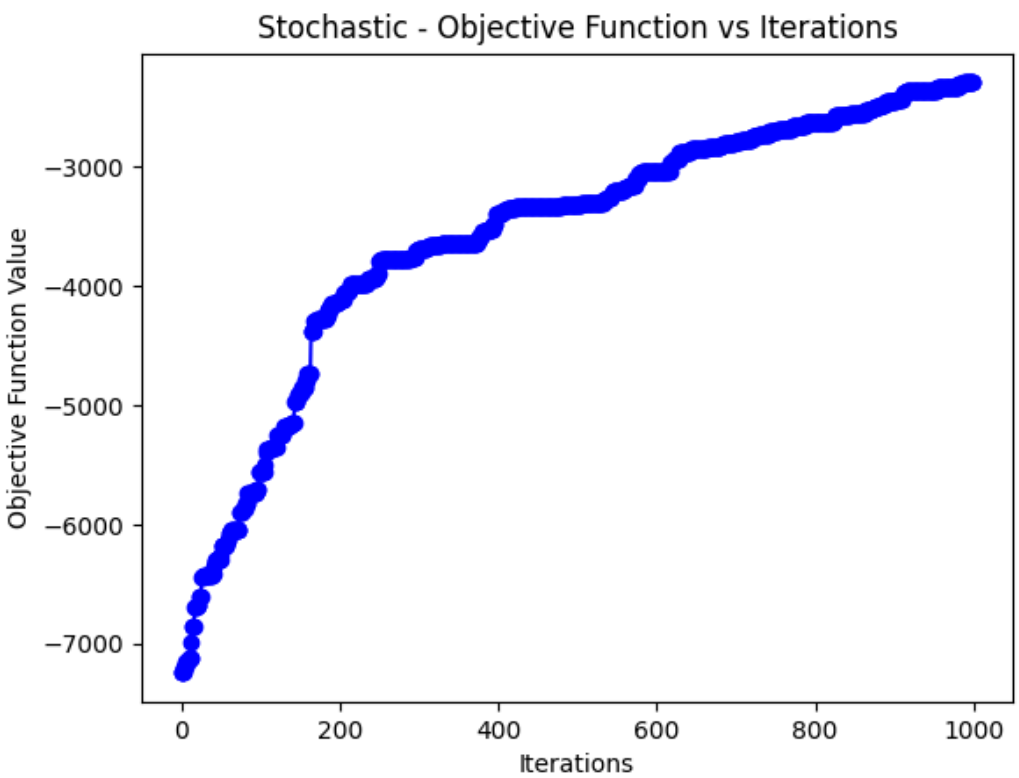
4.1.4. *Stochastic Hill-climbing*

Eksperimen 1 - *Stochastic Hill-climbing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[17 76 95 40 85] [54 103 41 84 61] [123 56 97 20 25] [74 27 32 79 107] [26 111 57 73 22]] [[122 90 13 12 72] [63 50 102 9 83] [8 15 71 116 77] [91 87 5 67 106] [35 78 110 104 2]] [[1 33 93 124 70] [29 121 58 96 10] [119 98 60 16 28] [82 14 69 24 118] [80 39 48 52 115]] [[113 94 21 59 18] [44 7 114 109 105] [53 112 37 34 55] [30 64 99 62 42] [66 47 23 49 86]] [[89 31 88 75 36] [120 51 11 81 45] [4 68 3 38 125] [6 108 117 65 19] [100 46 92 43 101]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-2291
Iterations	1000
Duration	1.1045794486999512 seconds

Grafik:

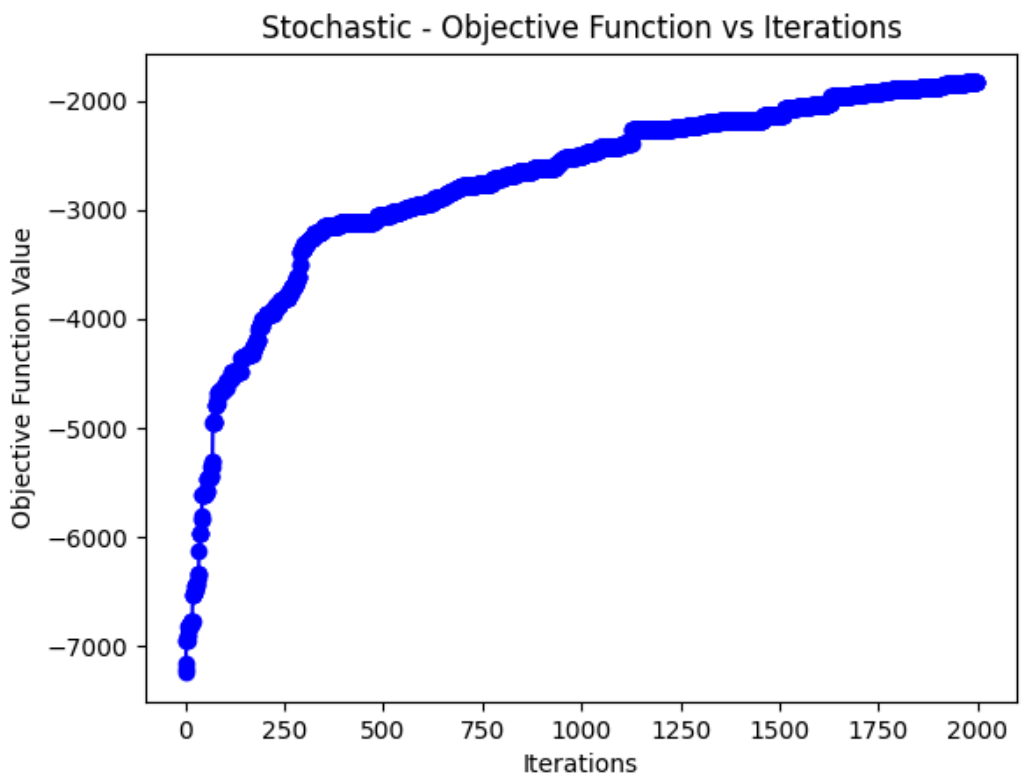


Eksperimen 2 - *Stochastic Hill-climbing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[13 93 109 9 103] [65 99 6 50 108] [116 38 42 86 4] [79 21 122 76 22] [43 92 16 84 78]] [[123 83 61 2 100] [112 36 28 34 39] [8 97 63 90 67] [5 77 44 91 114] [68 10 118 107 7]] [[27 37 71 124 48] [1 119 51 72 115] [120 49 85 24 31] [110 14 53 35 106] [94 104 52 58 26]] [[57 41 75 111 20] [69 17 105 70 33] [18 74 82 3 113] [87 81 32 66 45] [73 80 23 46 95]] [[96 62 30 59 40] [101 11 117 88 25] [56 47 19 89 102] [54 125 55 64 29] [12 60 98 15 121]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1833
Iterations	2000
Duration	2.1587934494018555 seconds

Grafik:

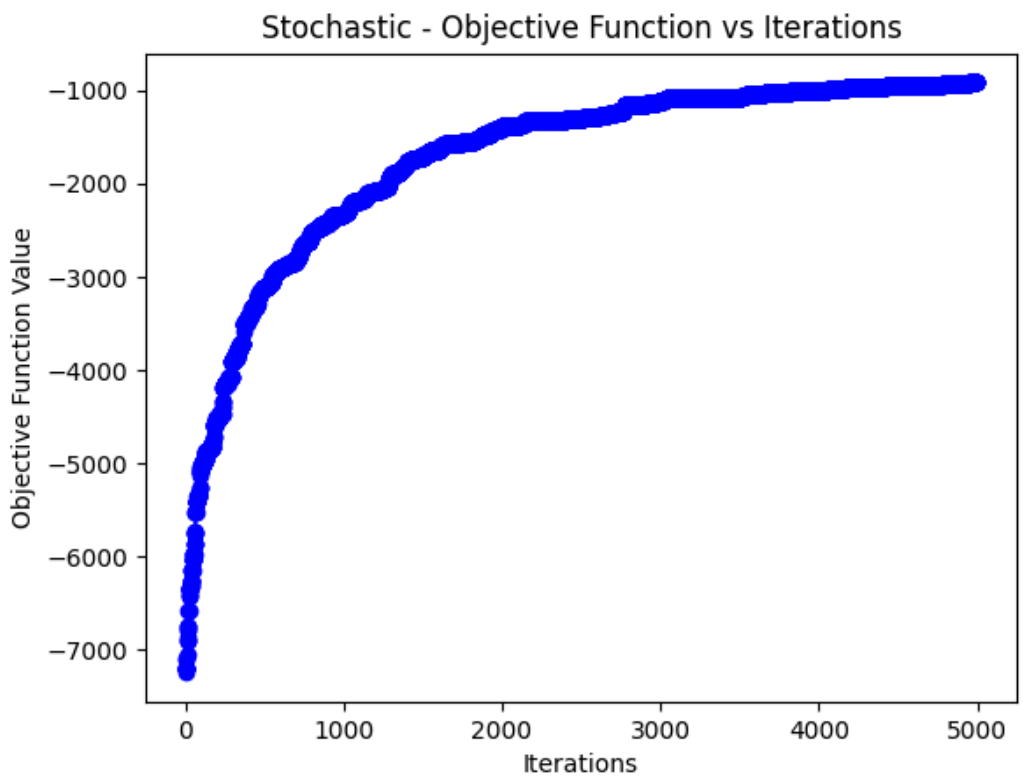


Eksperimen 3 - *Stochastic Hill-climbing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[27 104 81 54 49] [100 61 65 86 4] [75 55 117 7 88] [64 22 28 60 123] [47 80 21 112 52]] [[122 25 93 39 36] [19 111 6 99 105] [58 78 23 89 74] [42 85 124 3 48] [77 10 69 83 53]] [[16 92 91 115 8] [116 2 37 32 118] [119 9 73 66 46] [5 120 41 114 31] [56 110 72 1 107]] [[76 59 14 62 113] [24 106 98 11 70] [29 40 67 79 101] [103 45 97 57 13] [90 63 34 102 18]] [[82 33 30 50 109] [68 38 108 87 15] [26 125 35 96 12] [94 43 20 71 95] [44 51 121 17 84]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-916
Iterations	5000
Duration	5.11855673789978 seconds

Grafik:



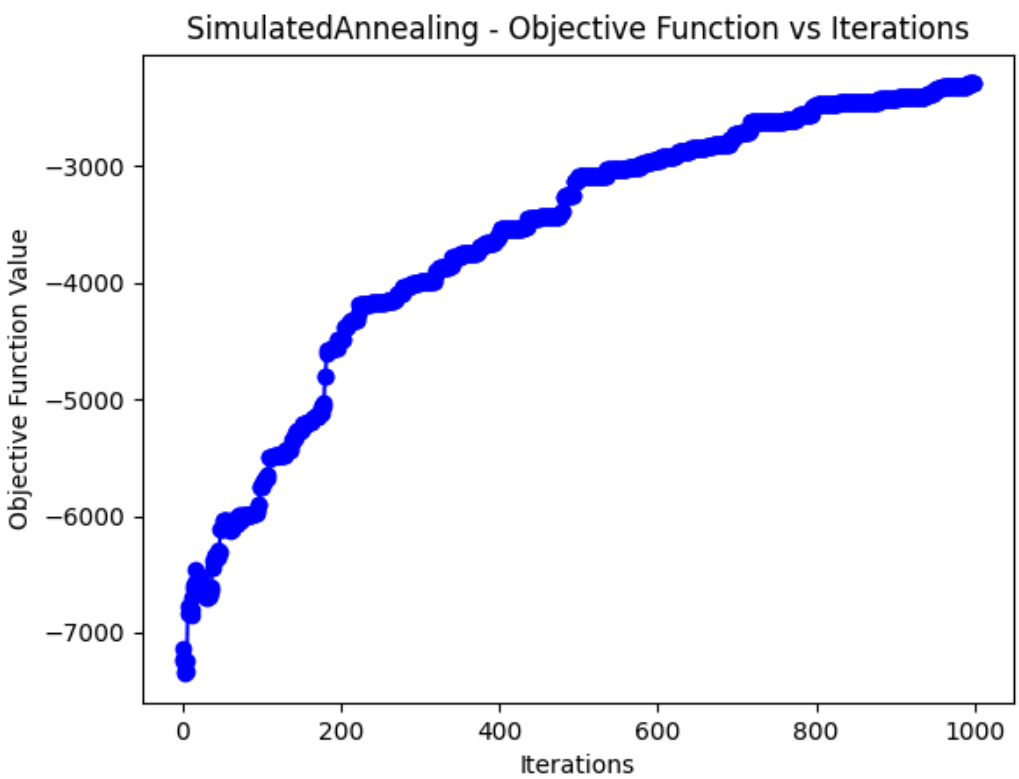
4.1.5. *Simulated Annealing*

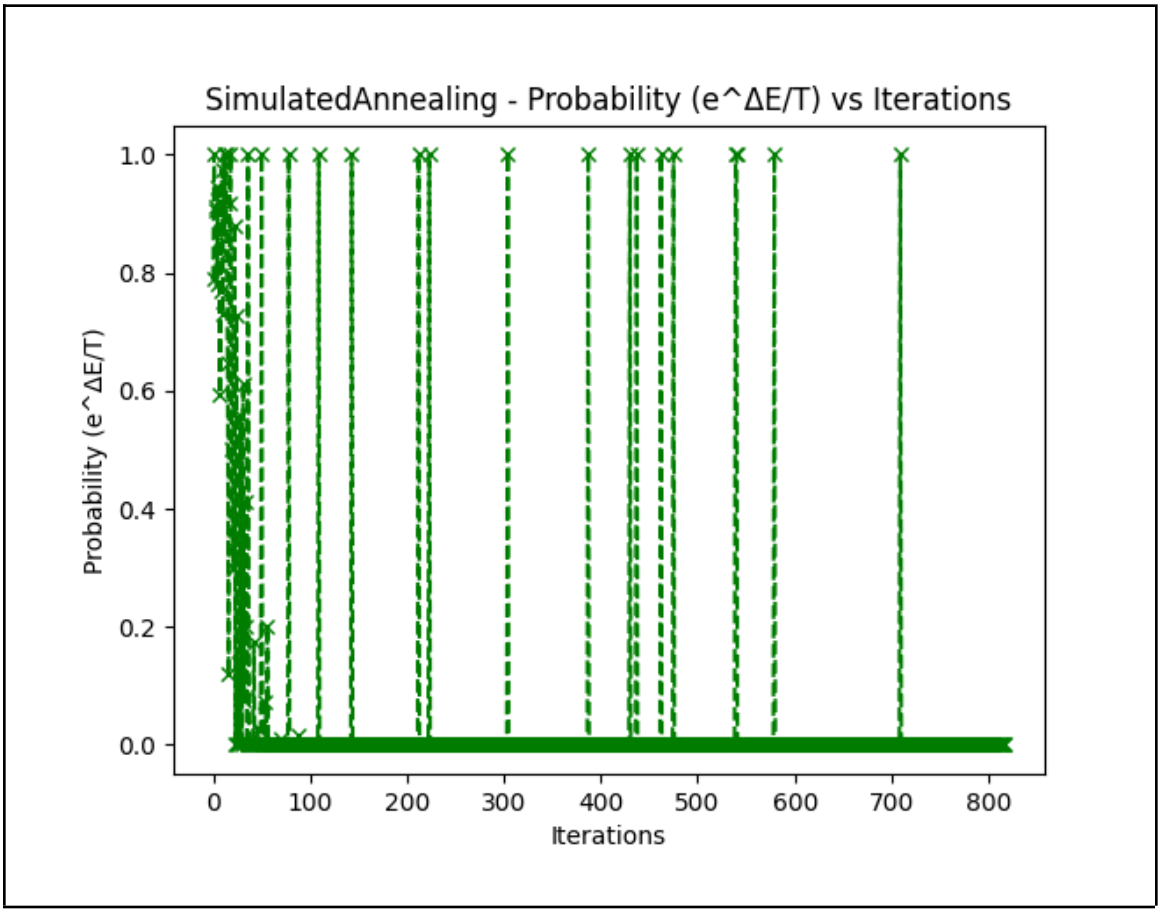
Eksperimen 1 - *Simulated Annealing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[4 69 101 112 31] [67 120 18 50 59] [12 57 100 62 94] [121 14 40 22 73] [115 30 34 75 65]] [[122 71 56 47 117] [64 95 13 60 104] [82 45 7 80 93] [6 52 119 87 2] [68 86 105 21 33]] [[27 10 113 123 23] [74 76 109 84 28] [54 107 58 20 91] [61 97 89 17 48] [51 25 11 90 124]] [[63 41 1 79 125] [29 24 114 42 102] [88 55 98 83 16] [110 77 39 81 8] [5 111 72 32 49]] [[92 106 35 38 36] [99 19 103 78 9] [53 43 46 66 108] [3 85 15 118 96] [70 37 116 26 44]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-2291
Iterations	1000
Duration	1.5647015571594238 seconds
Stuck Frequency (Local Optima)	36

Grafik:



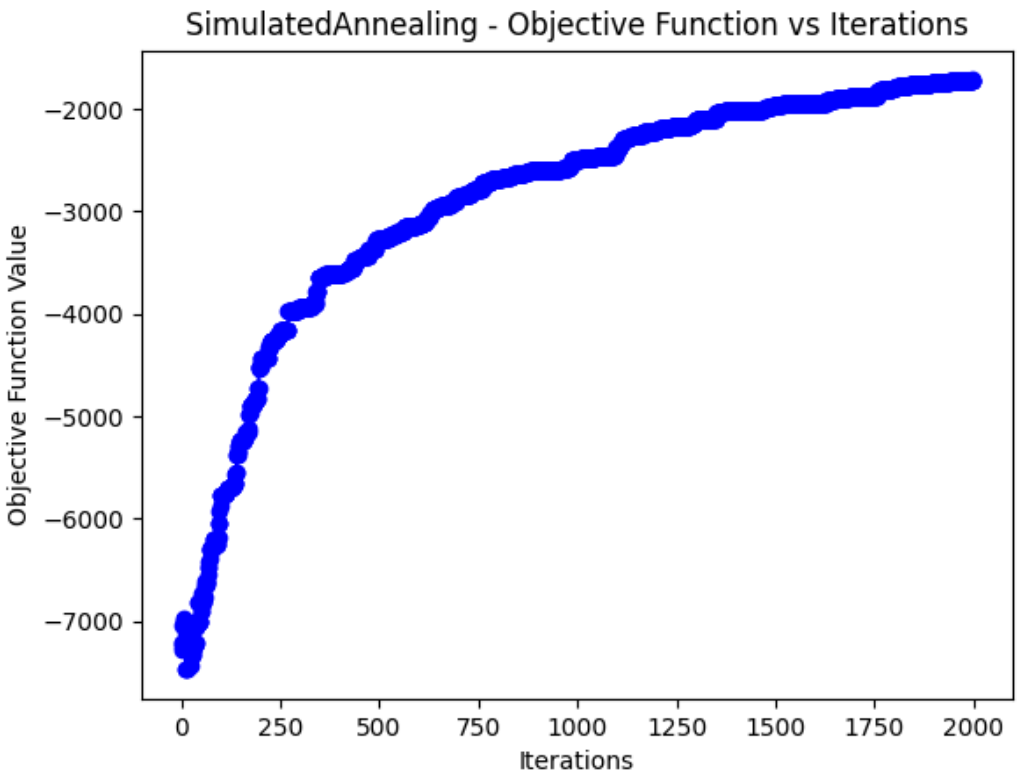

SimulatedAnnealing - Probability (e^ΔE/T) vs Iterations
Probability (e^ΔE/T)
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
600
700
800
Iterations

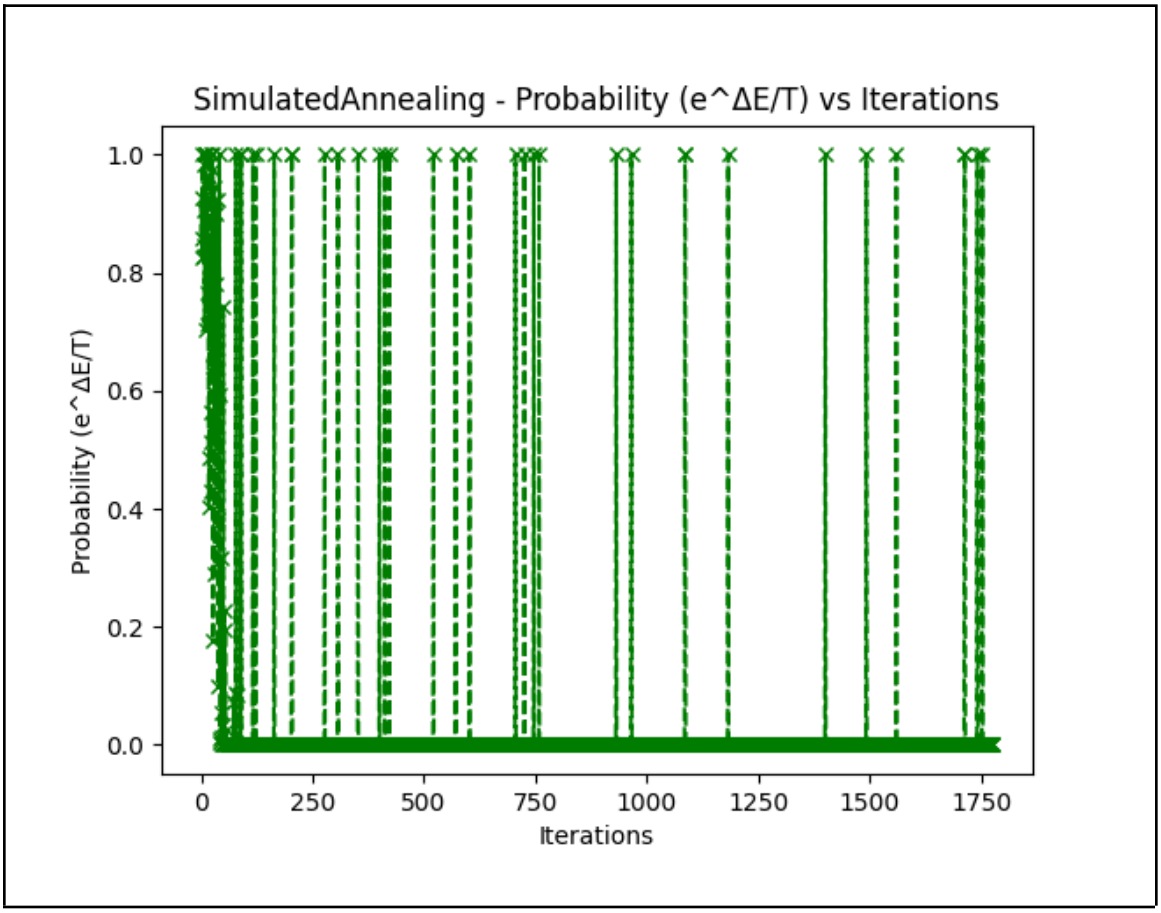
Eksperimen 2 - *Simulated Annealing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[6 98 68 79 64] [85 72 2 15 118] [86 70 112 26 36] [33 28 103 122 73] [94 57 53 74 35]] [[124 24 14 34 113] [63 23 44 108 83] [125 67 16 101 4] [13 93 117 40 52] [1 91 123 30 66]] [[69 3 110 109 20] [45 121 82 32 25] [47 75 60 61 77] [62 96 59 7 84] [89 22 10 116 81]] [[78 76 107 55 12] [114 8 58 106 42] [9 56 120 31 105] [100 65 19 39 54] [17 111 21 71 97]] [[50 87 29 38 92] [5 102 104 48 46] [41 51 18 99 88] [95 37 11 90 80] [119 43 115 27 49]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1715
Iterations	2000
Duration	3.0487987995147705 seconds
Stuck Frequency (Local Optima)	103

Grafik:



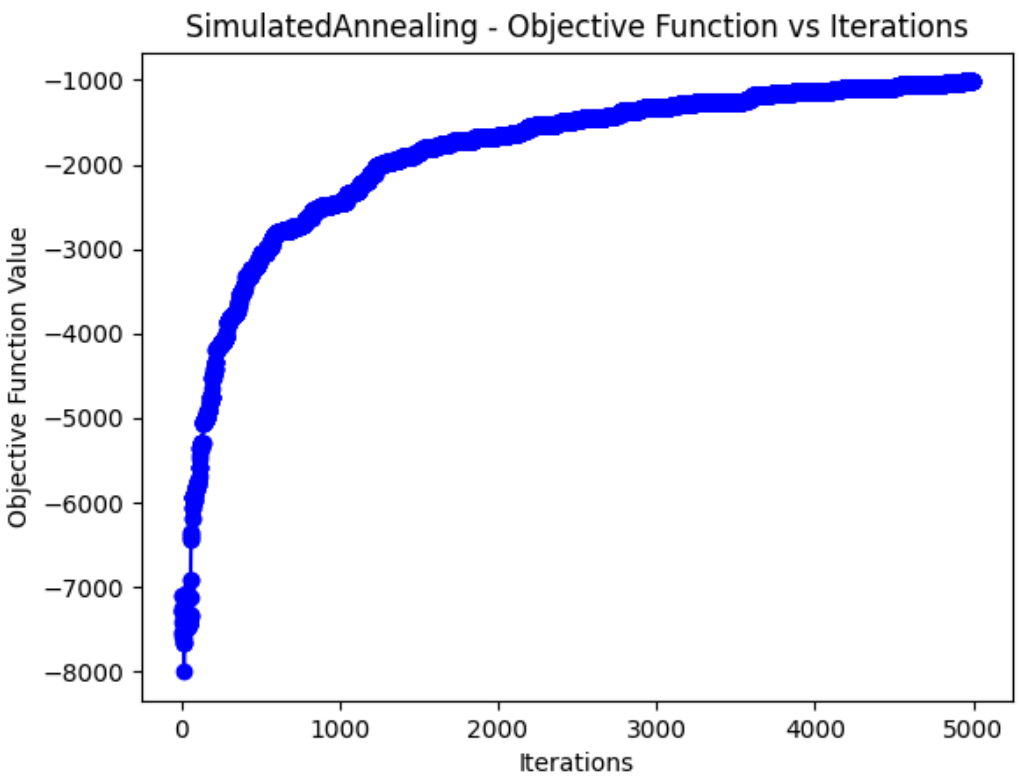

SimulatedAnnealing - Probability (e^ΔE/T) vs Iterations
Probability (e^ΔE/T)
1.0
0.8
0.6
0.4
0.2
0.0
0
250
500
750
1000
1250
1500
1750
Iterations

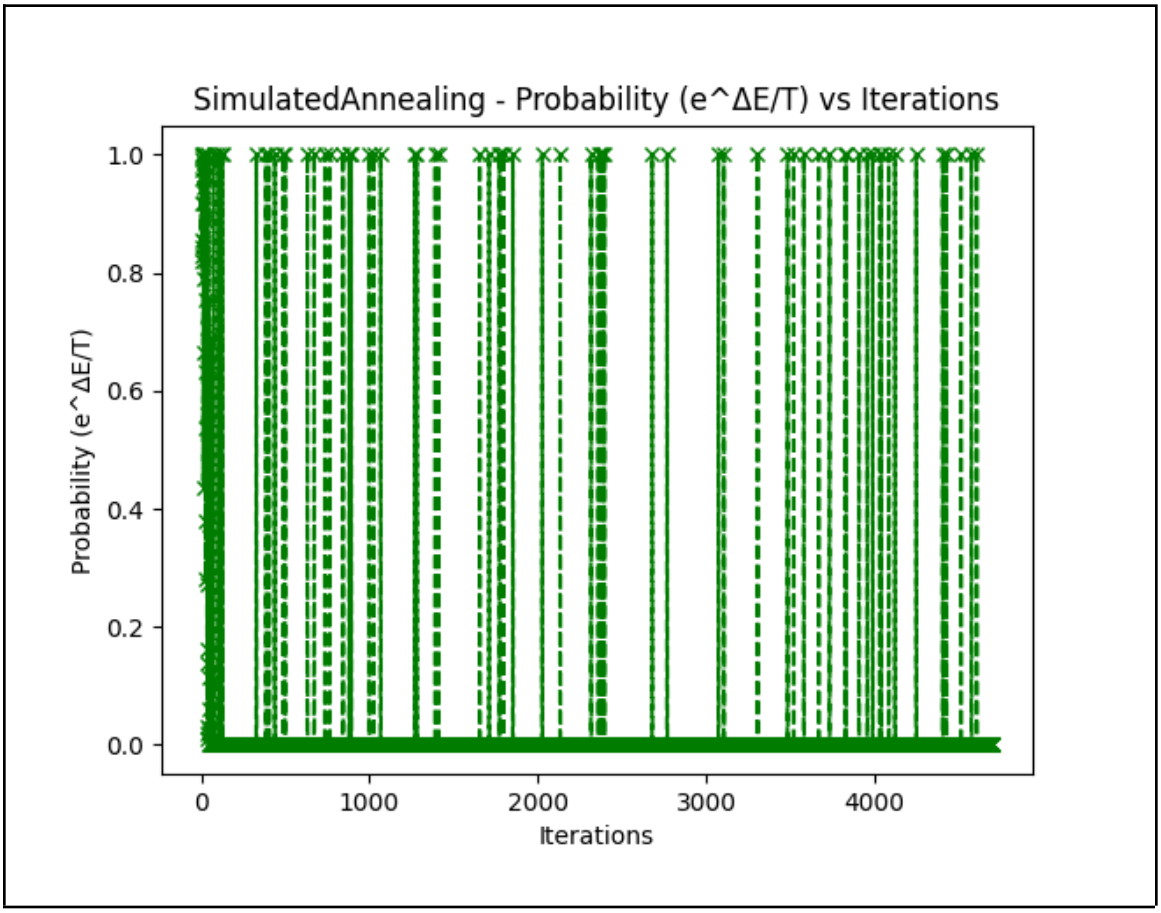
Eksperimen 3 - *Simulated Annealing*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[50 46 81 43 97] [78 112 4 6 114] [69 64 67 98 16] [13 23 123 71 83] [105 70 29 99 18]] [[52 100 95 2 82] [125 36 14 86 8] [22 73 68 56 96] [92 76 19 87 37] [10 17 113 84 90]] [[79 24 40 124 44] [21 117 58 60 85] [12 88 62 35 109] [120 74 48 25 41] [80 11 111 75 38]] [[20 101 9 121 65] [42 31 122 66 54] [119 45 61 30 63] [33 34 115 94 49] [104 107 7 5 91]] [[118 51 89 27 28] [26 3 116 102 72] [108 47 57 93 32] [59 103 1 39 106] [15 110 53 55 77]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-1020
Iterations	5000
Duration	7.615490913391113 seconds
Stuck Frequency (Local Optima)	340

Grafik:




SimulatedAnnealing - Probability (e^ΔE/T) vs Iterations
Probability (e^ΔE/T)
1.0
0.8
0.6
0.4
0.2
0.0
0
1000
2000
3000
4000
Iterations

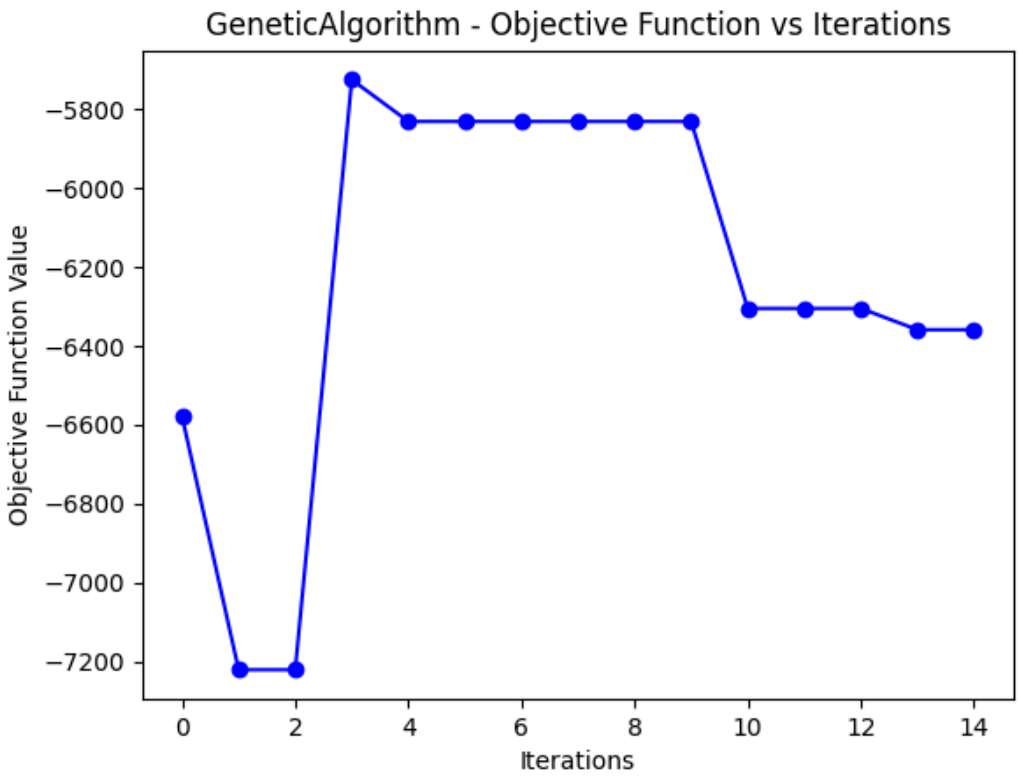
4.1.6. *Genetic Algorithm*

Eksperimen Jumlah Populasi 1-1 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[119 118 63 110 2] [93 3 4 29 31] [24 85 55 104 114] [25 6 68 91 37] [54 112 125 32 106]] [[74 89 11 47 121] [51 66 44 48 39] [41 83 86 20 9] [61 33 53 75 101] [12 27 72 95 105]] [[117 14 43 26 98] [62 35 18 79 88] [109 42 15 115 71] [80 91 78 8 58] [40 102 107 46 108]] [[17 124 45 111 13] [103 60 65 97 36] [90 50 122 84 70] [52 19 64 99 81] [87 67 16 96 49]] [[116 38 123 21 57] [92 7 100 78 38] [51 103 62 57 34] [43 4 49 73 91] [121 55 67 124 33]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6029
Iterations	25
Duration	73.46373295783997 seconds
Population Size	50

Grafik:

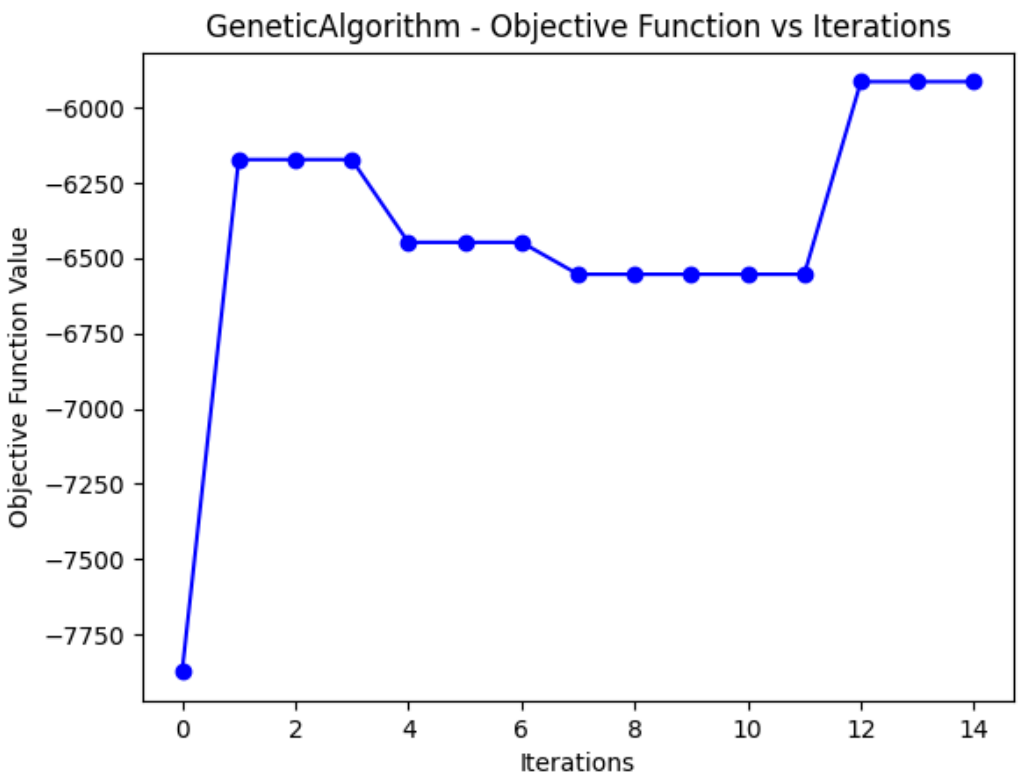


Eksperimen Jumlah Populasi 1-2 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[63 59 26 115 82] [16 12 28 22 95] [76 33 75 107 73] [122 46 113 112 4] [101 84 88 68 53]] [[59 25 30 56 120] [106 40 70 96 47] [19 2 71 17 109] [24 104 82 93 59] [38 87 62 86 78]] [[33 55 113 108 83] [58 1 33 37 112] [118 19 89 27 70] [8 30 58 61 94] [70 15 62 55 40]] [[124 33 115 31 54] [78 65 64 90 98] [42 24 48 79 59] [5 89 37 118 45] [85 50 51 123 47]] [[9 32 31 61 41] [37 73 34 90 104] [113 45 71 103 27] [15 35 56 61 113] [78 108 9 39 81]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-5913
Iterations	15
Duration	28.9536 seconds
Population Size	80

Grafik:

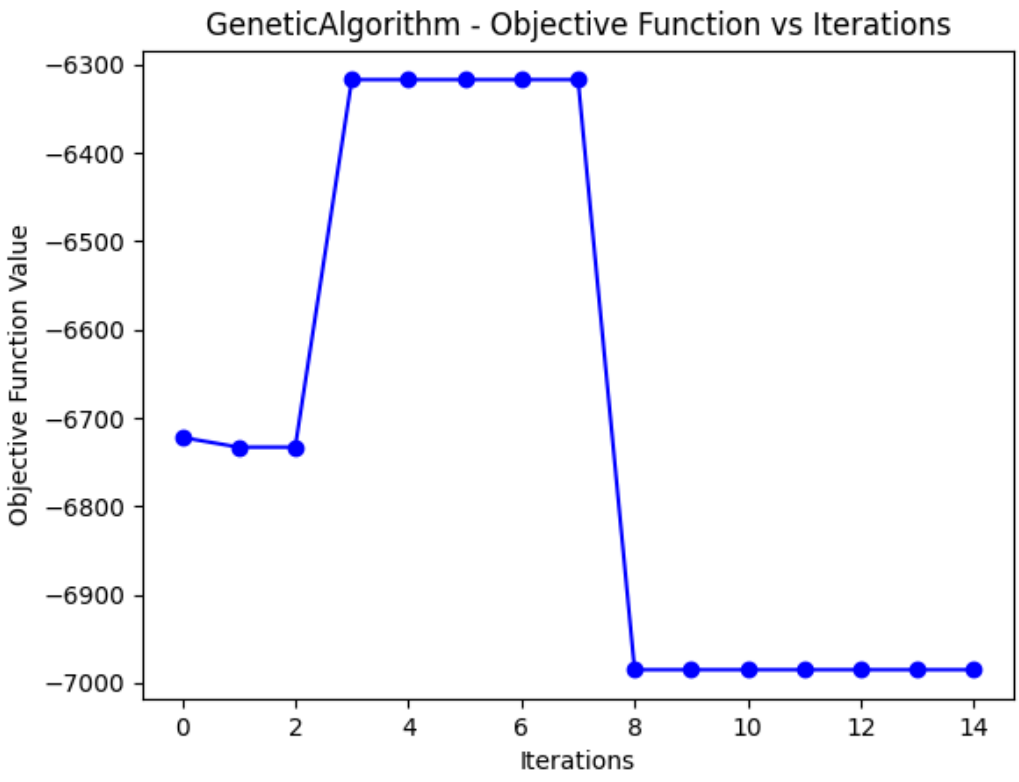


Eksperimen Jumlah Populasi 1-3 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[36 1 112 84 10] [115 73 6 14 86] [3 59 81 92 65] [38 17 103 108 105] [12 26 13 116 118]] [[32 123 11 69 27] [88 2 25 94 110] [57 125 114 19 32] [48 27 84 93 109] [44 122 74 89 37]] [[55 53 76 32 94] [35 15 11 114 48] [58 17 80 54 60] [107 71 7 5 12] [50 121 116 86 8]] [[26 70 68 88 14] [16 47 73 78 62] [30 59 108 98 75] [56 49 118 100 45] [115 90 24 95 34]] [[42 20 65 120 87] [105 64 13 18 102] [22 2 101 46 21] [96 36 3 117 9] [31 81 51 23 110]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6985
Iterations	15
Duration	28.6447 seconds
Population Size	80

Grafik:

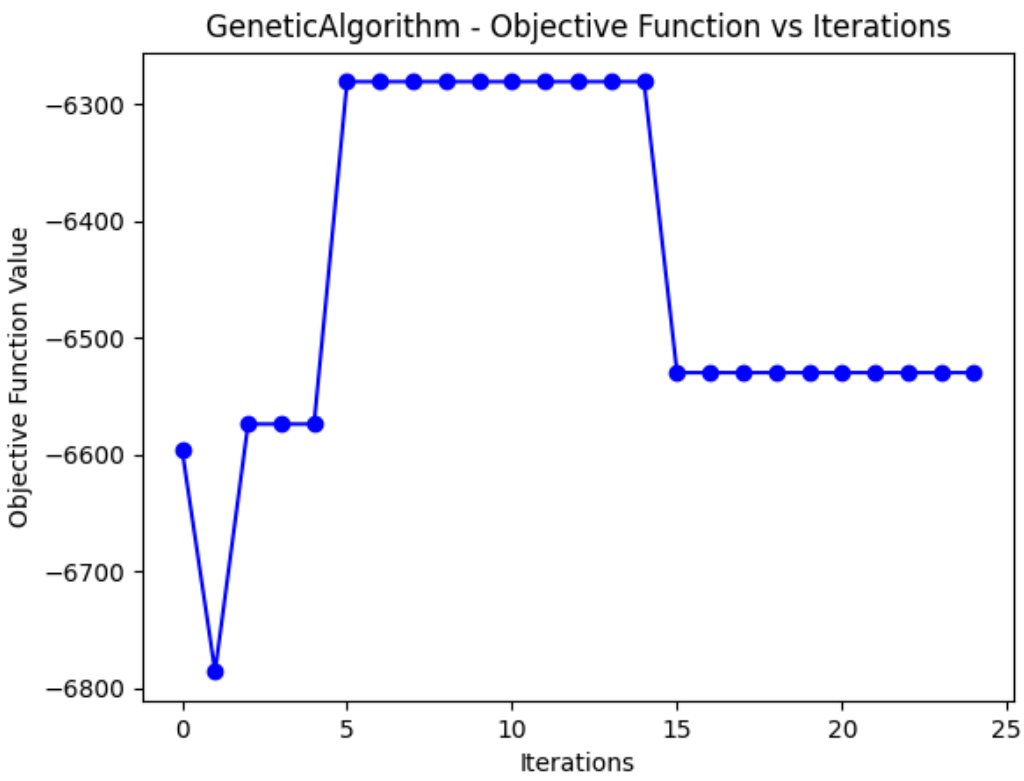


Eksperimen Jumlah Populasi 2-1 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[40 112 90 45 3] [75 24 63 30 123] [46 60 20 47 87] [37 26 78 55 82] [106 69 25 1 33]] [[108 19 15 71 44] [63 121 120 68 113] [17 14 64 6 93] [115 56 9 99 60] [29 105 107 74 101]] [[76 47 103 66 57] [27 59 58 49 23] [32 70 41 62 31] [90 10 120 67 72] [124 66 17 11 121]] [[44 102 56 110 56] [32 64 5 1 63] [100 112 62 22 19] [95 102 60 13 78] [121 30 31 25 69]] [[119 109 51 103 43] [45 9 93 122 28] [2 46 41 97 120] [6 42 75 108 84] [26 78 54 87 59]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6530
Iterations	25
Duration	47.5618 seconds
Population Size	80

Grafik:

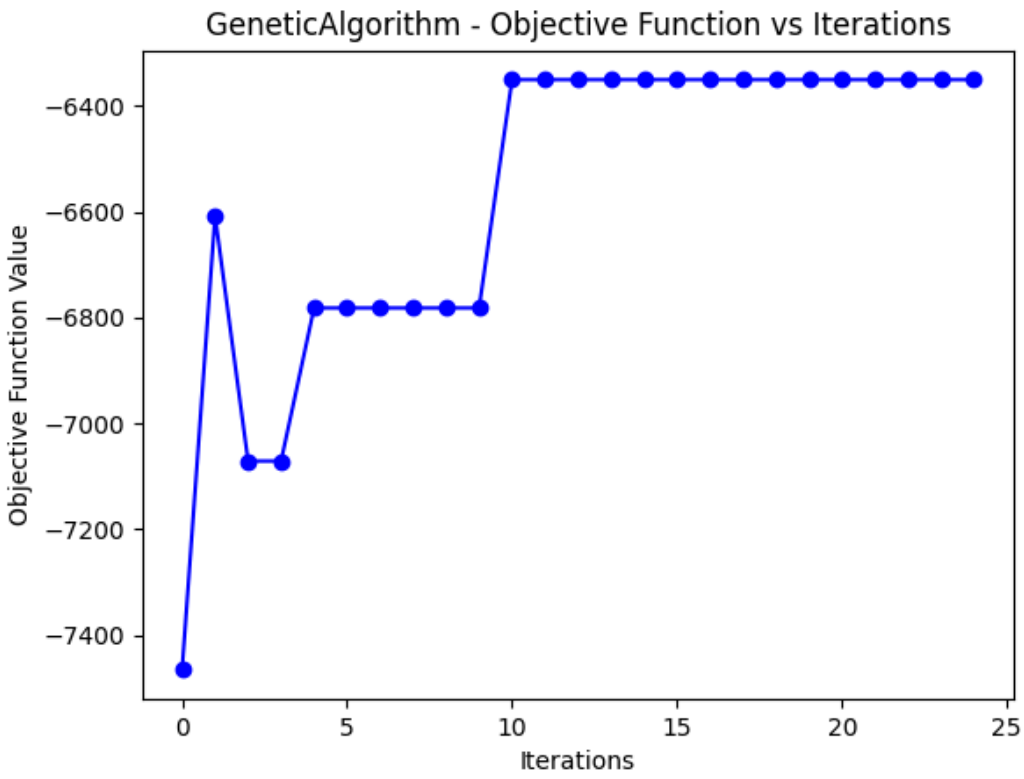


Eksperimen Jumlah Populasi 2-2 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[88 89 39 16 83] [123 45 84 79 42] [20 4 123 96 104] [93 9 75 114 38] [105 21 103 54 4]] [[113 72 86 106 112] [39 98 38 87 17] [121 107 91 59 16] [89 98 13 113 107] [121 92 47 89 32]] [[20 8 44 90 77] [40 31 98 97 55] [117 87 63 50 45] [32 101 125 21 36] [47 21 69 69 98]] [[40 72 74 22 73] [111 77 11 19 110] [56 49 68 57 85] [55 81 82 73 105] [107 87 33 45 28]] [[96 32 72 31 63] [14 35 84 5 104] [88 16 114 61 6] [46 33 23 113 59] [84 77 103 17 19]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6351
Iterations	25
Duration	47.7674 seconds
Population Size	80

Grafik:

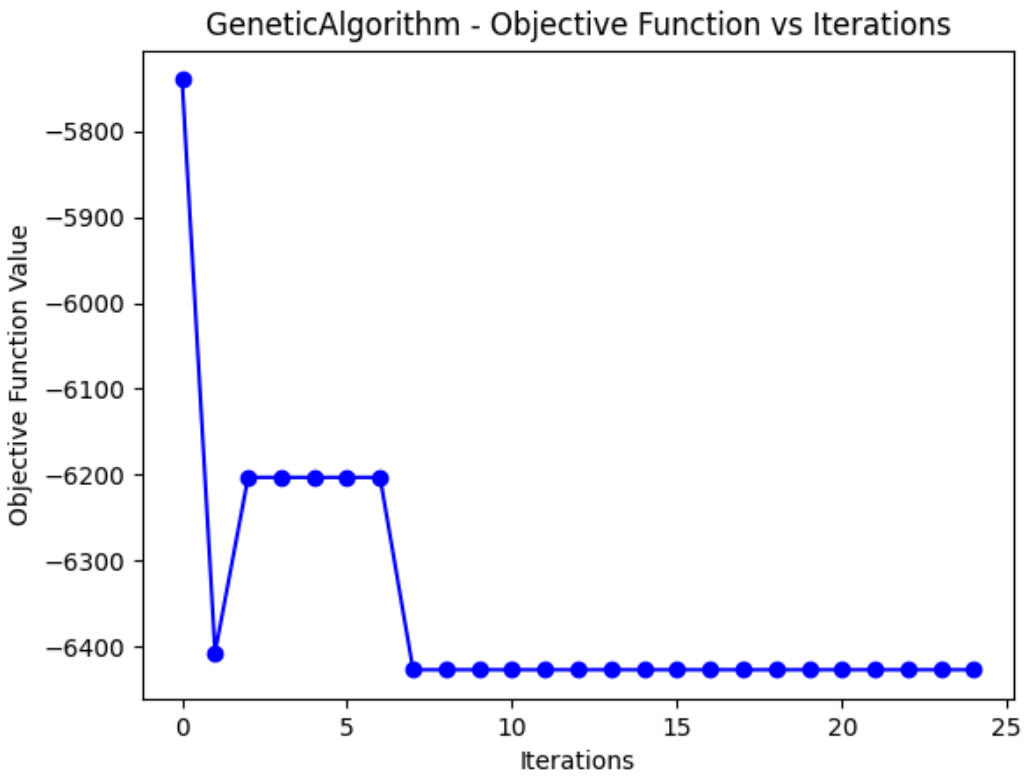


Eksperimen Jumlah Populasi 2-3 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[45 56 92 100 36] [64 52 83 14 101] [103 38 49 13 112] [109 53 91 19 29] [60 42 94 114 73]] [[12 7 41 70 22] [104 9 93 82 43] [1 113 20 119 10] [99 100 112 46 78] [51 118 30 36 24]] [[13 121 101 91 26] [68 11 42 53 72] [65 110 50 29 46] [39 84 69 16 97] [106 92 54 43 20]] [[75 48 102 34 108] [70 19 12 82 14] [109 116 17 14 61] [94 22 35 93 25] [88 32 107 23 21]] [[18 114 10 79 100] [9 56 15 52 85] [45 31 47 122 112] [67 62 74 104 8] [27 111 113 99 76]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6427
Iterations	25
Duration	55.4283 seconds
Population Size	80

Grafik:

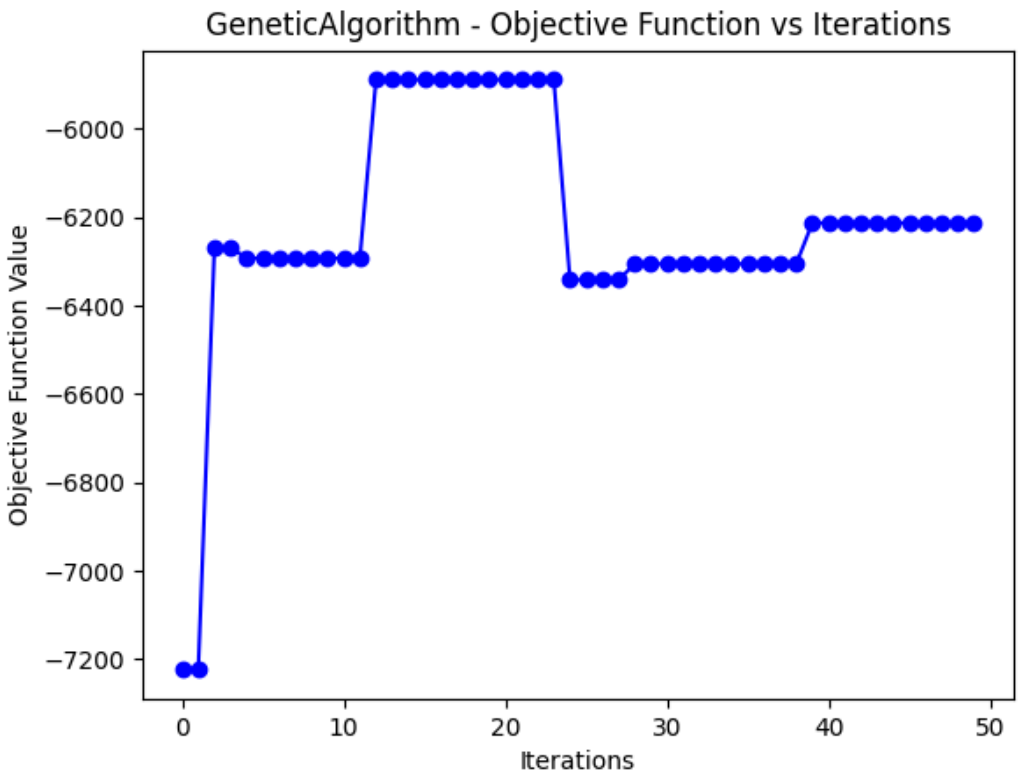


Eksperimen Jumlah Populasi 3-1 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[17 90 85 56 67] [42 73 29 66 22] [97 27 79 83 76] [107 50 55 51 58] [124 38 54 23 76]] [[43 9 6 16 107] [10 94 118 37 86] [83 79 51 57 44] [51 33 47 24 87] [6 71 93 114 31]] [[5 90 22 102 7] [15 16 14 5 79] [42 115 58 7 62] [84 105 85 12 123] [21 101 23 93 43]] [[45 110 71 42 68] [47 91 90 27 51] [98 106 89 125 56] [16 88 44 122 21] [114 1 111 78 50]] [[76 87 83 28 120] [113 77 118 20 59] [26 10 38 39 35] [26 65 27 106 91] [78 60 49 7 64]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6231
Iterations	50
Duration	110.989 seconds
Population Size	80

Grafik:

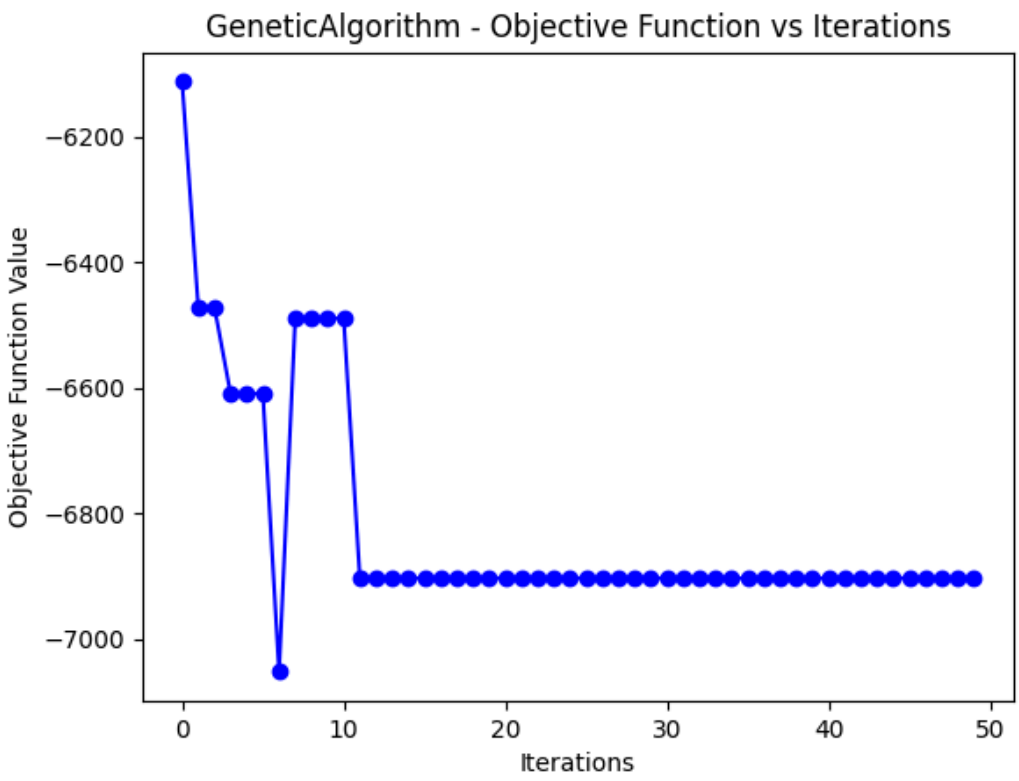


Eksperimen Jumlah Populasi 3-2 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[11 84 21 121 14] [78 55 18 36 45] [40 115 44 1 71] [59 3 104 95 54] [17 79 84 93 114]] [[89 83 110 103 81] [55 100 70 90 65] [59 42 18 125 33] [96 66 108 30 103] [122 82 123 90 113]] [[107 59 91 42 118] [12 23 55 47 27] [49 100 32 84 80] [90 97 38 114 21] [74 9 99 92 68]] [[71 55 5 116 39] [63 119 124 49 76] [79 44 10 64 115] [59 29 56 12 30] [43 112 100 107 103]] [[37 80 44 51 116] [28 74 95 68 50] [43 64 56 61 91] [15 48 84 18 93] [19 33 58 21 17]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6903
Iterations	50
Duration	98.555 seconds
Population Size	80

Grafik:

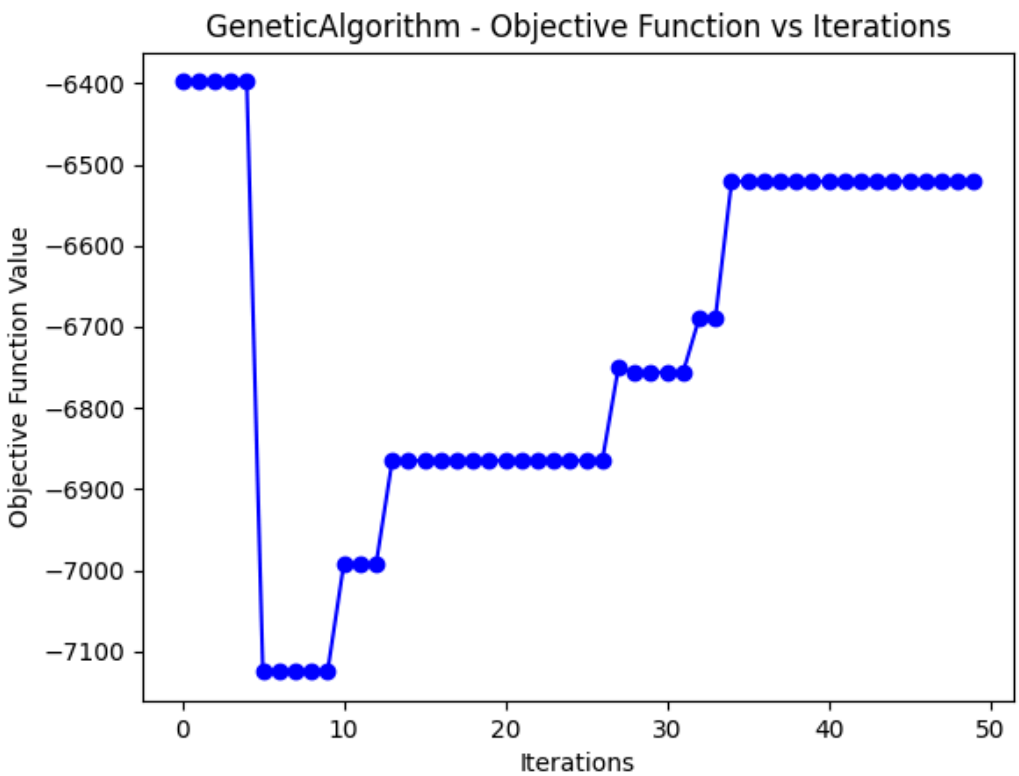


Eksperimen Jumlah Populasi 3-3 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[11 17 50 28 16] [33 6 46 108 83] [24 76 52 36 122] [5 105 17 29 13] [22 37 100 114 81]] [[48 86 53 55 78] [40 114 65 102 34] [112 27 7 95 66] [62 46 37 72 57] [30 39 33 108 105]] [[31 112 36 94 34] [48 6 88 117 60] [55 107 104 36 21] [45 83 111 47 94] [15 7 27 110 2]] [[11 99 85 100 25] [23 103 106 123 59] [125 124 42 99 30] [117 37 21 87 48] [39 13 80 75 103]] [[59 33 86 69 82] [104 125 72 37 7] [102 57 125 21 10] [35 44 61 50 88] [28 70 75 22 120]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6521
Iterations	50
Duration	107.576 seconds
Population Size	80

Grafik:

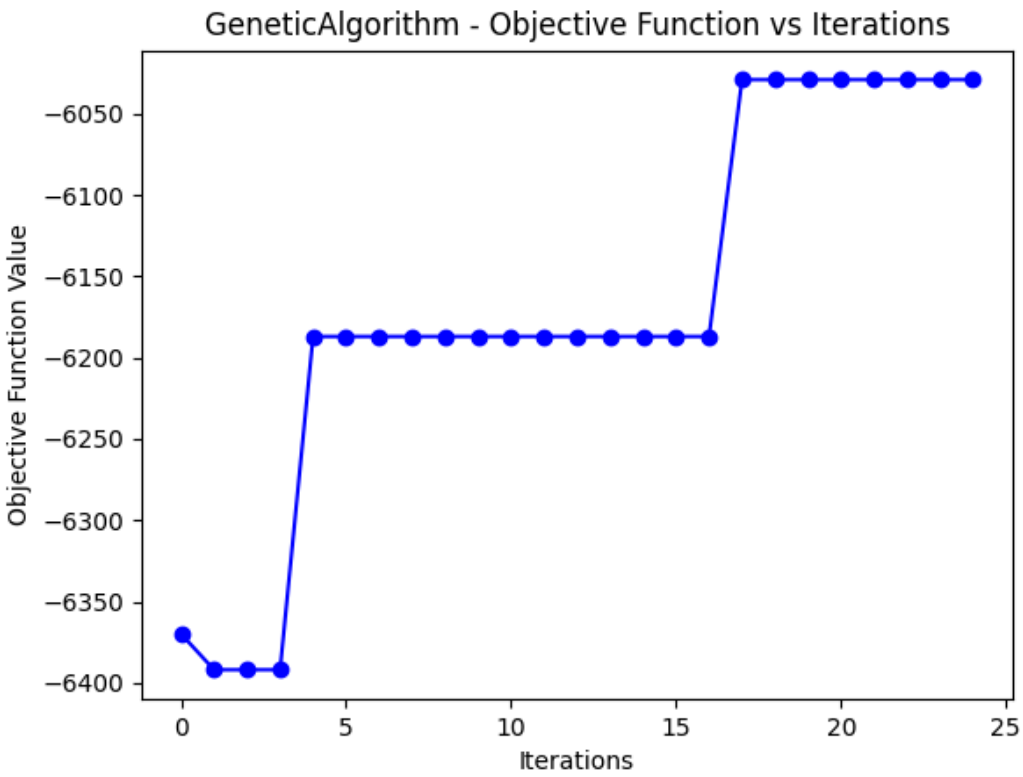


Eksperimen Banyak Iterasi 1-1 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[11 17 50 28 16] [33 6 46 108 83] [24 76 52 36 122] [5 105 17 29 13] [22 37 100 114 81]] [[48 86 53 55 78] [40 114 65 102 34] [112 27 7 95 66] [62 46 37 72 57] [30 39 33 108 105]] [[31 112 36 94 34] [48 6 88 117 60] [55 107 104 36 21] [45 83 111 47 94] [15 7 27 110 2]] [[11 99 85 100 25] [23 103 106 123 59] [125 124 42 99 30] [117 37 21 87 48] [39 13 80 75 103]] [[59 33 86 69 82] [104 125 72 37 7] [102 57 125 21 10] [35 44 61 50 88] [28 70 75 22 120]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6029
Iterations	25
Duration	73.464 seconds
Population Size	50

Grafik:

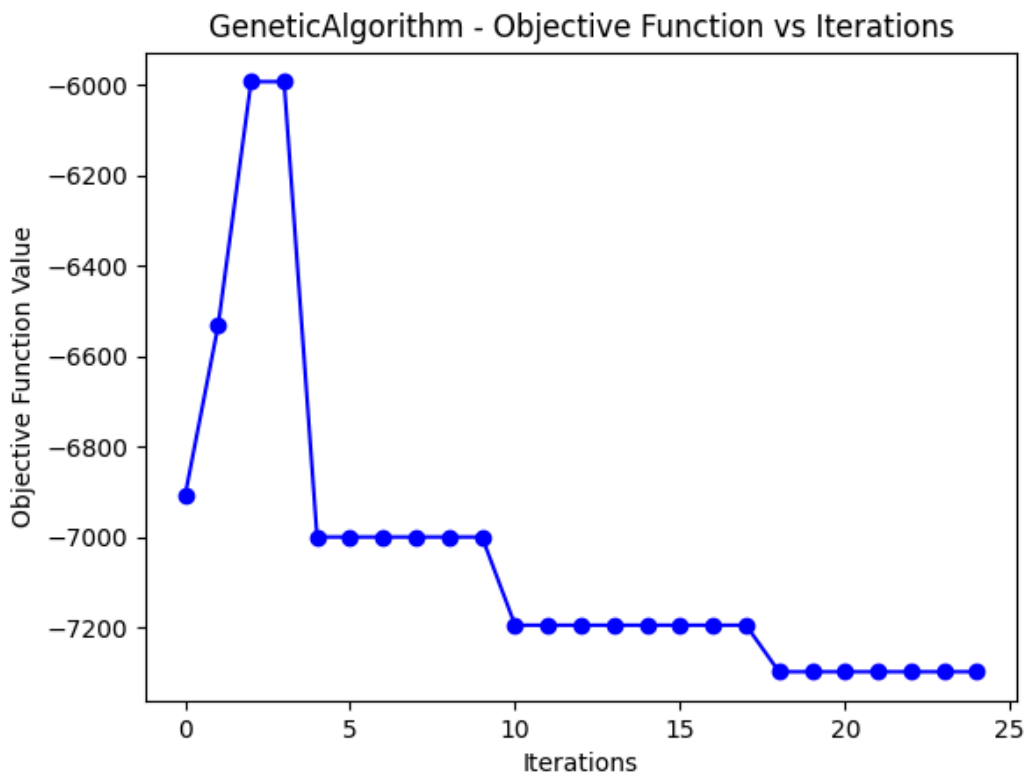


Eksperimen Banyak Iterasi 1-2 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[60 41 94 1 119] [114 90 116 114 55] [39 67 11 90 80] [59 31 48 85 106] [79 61 105 118 72]] [[116 16 28 54 78] [39 48 84 93 7] [67 92 57 106 36] [112 91 124 83 10] [77 74 58 80 11]] [[120 117 2 64 76] [24 51 60 110 70] [32 13 43 9 120] [53 28 75 12 90] [14 66 75 72 118]] [[14 101 2 125 108] [40 93 21 78 121] [63 100 64 9 117] [13 49 68 115 70] [16 6 103 123 109]] [[111 113 36 82 107] [110 102 20 104 5] [45 7 49 47 73] [26 87 32 65 105] [82 110 40 31 121]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-7297
Iterations	25
Duration	20.654 seconds
Population Size	50

Grafik:

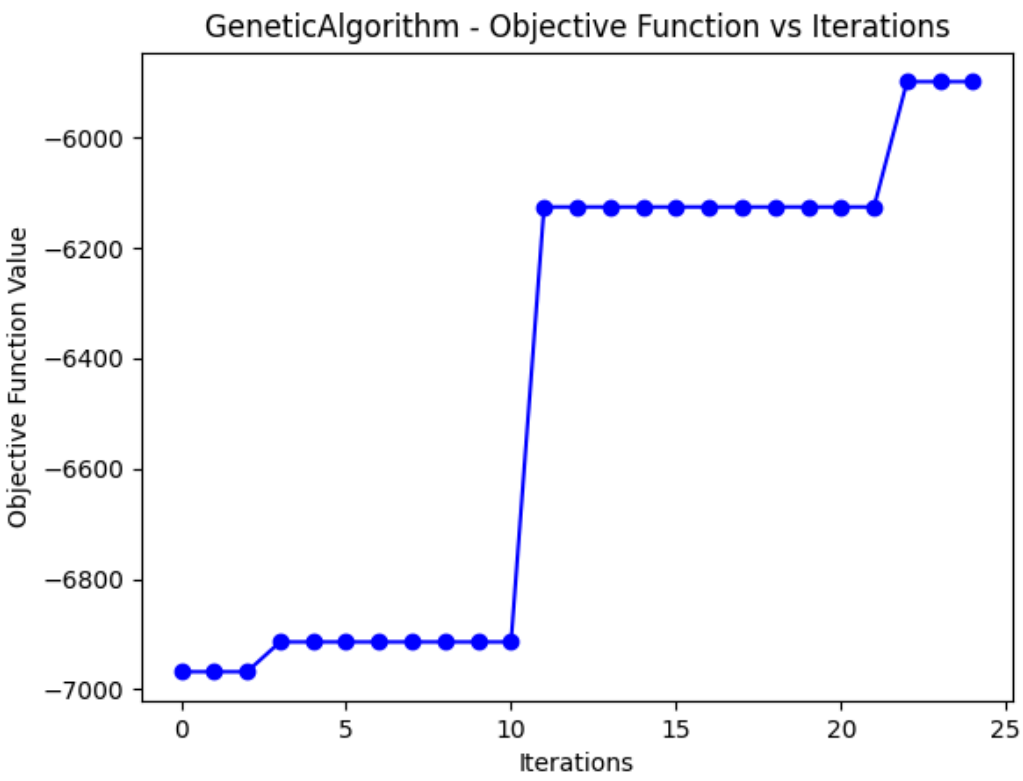


Eksperimen Banyak Iterasi 1-3 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[60 41 94 1 119] [114 90 116 114 55] [39 67 11 90 80] [59 31 48 85 106] [79 61 105 118 72]] [[116 16 28 54 78] [39 48 84 93 7] [67 92 57 106 36] [112 91 124 83 10] [77 74 58 80 11]] [[120 117 2 64 76] [24 51 60 110 70] [32 13 43 9 120] [53 28 75 12 90] [14 66 75 72 118]] [[14 101 2 125 108] [40 93 21 78 121] [63 100 64 9 117] [13 49 68 115 70] [16 6 103 123 109]] [[111 113 36 82 107] [110 102 20 104 5] [45 7 49 47 73] [26 87 32 65 105] [82 110 40 31 121]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-5899
Iterations	25
Duration	19.531 seconds
Population Size	50

Grafik:

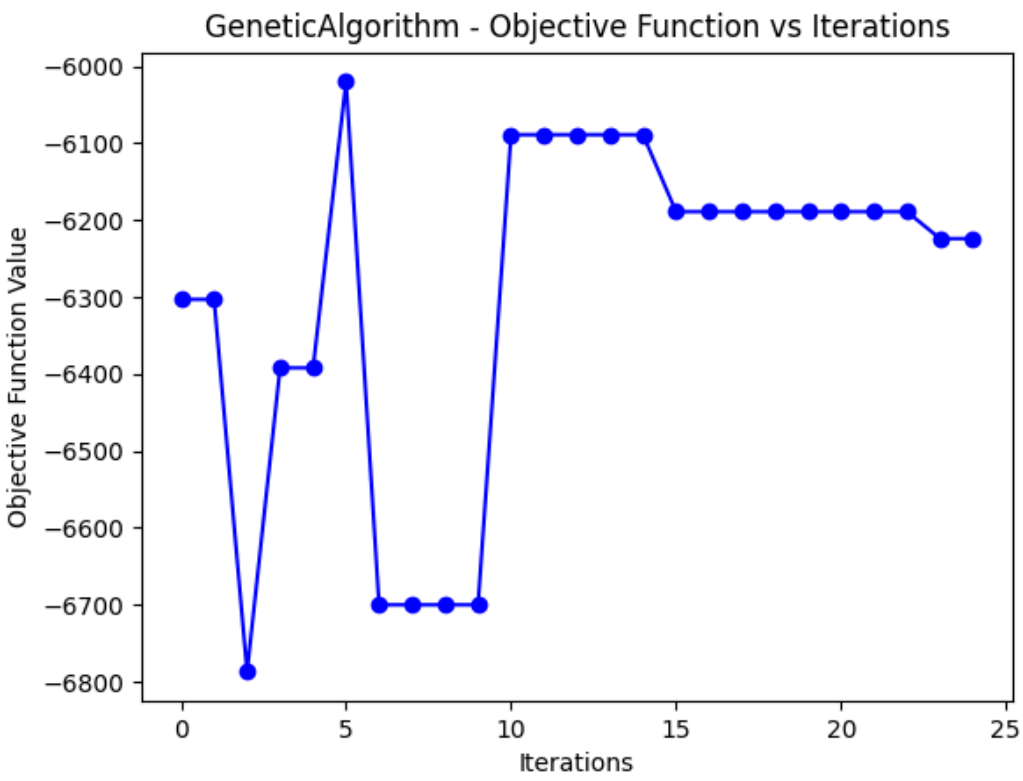


Eksperimen Banyak Iterasi 2-1 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[29 17 10 111 56] [58 95 20 21 34] [19 53 57 103 64] [11 105 41 70 88] [76 74 65 76 24]] [[67 120 102 70 111] [11 38 26 55 8] [49 86 30 90 74] [61 51 46 112 19] [53 115 5 35 103]] [[113 20 117 71 10] [95 116 68 52 69] [121 2 62 47 83] [94 22 60 45 57] [98 110 66 100 37]] [[88 1 21 18 39] [54 119 80 41 72] [85 91 113 76 99] [72 19 30 106 88] [110 15 80 50 3]] [[2 24 37 21 56] [51 75 43 59 110] [61 105 36 48 18] [4 24 20 68 123] [82 111 76 9 37]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6224
Iterations	25
Duration	50.968 seconds
Population Size	50

Grafik:

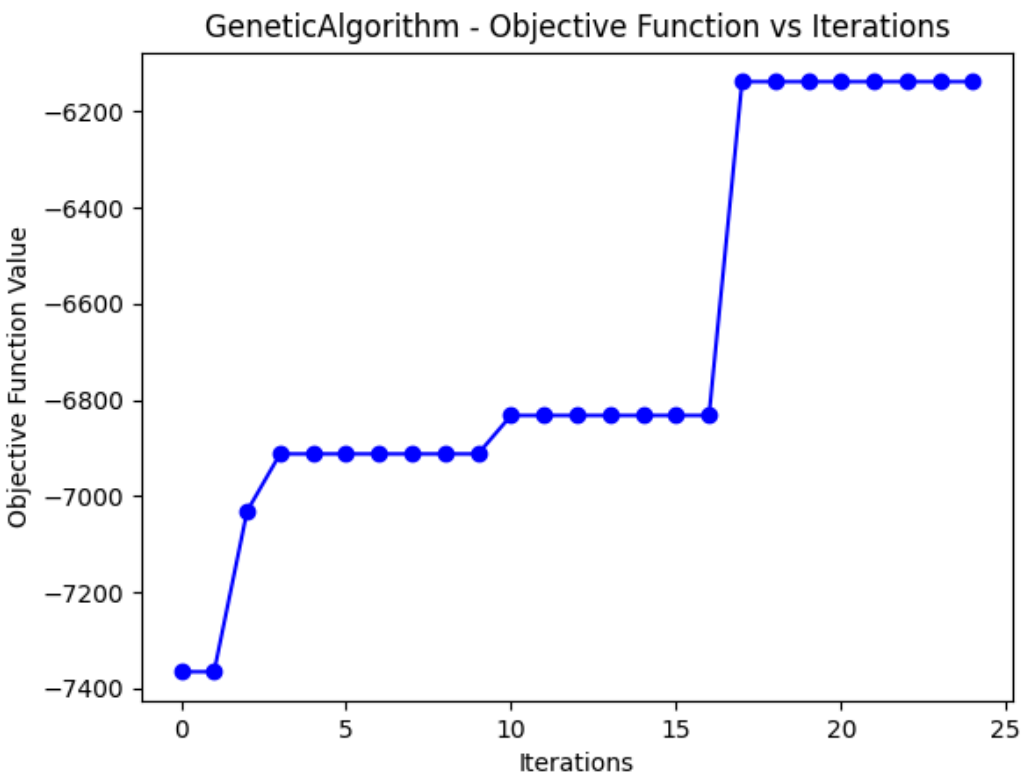


Eksperimen Banyak Iterasi 2-2 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[77 7 122 119 89] [70 34 93 52 71] [56 100 16 33 39] [58 72 78 102 5] [21 96 44 40 92]] [[119 95 51 85 109] [48 59 70 42 69] [2 73 14 104 13] [84 49 116 43 18] [4 21 110 107 80]] [[62 19 22 51 92] [81 37 94 77 99] [24 60 91 45 59] [55 23 7 14 109] [70 13 48 89 5]] [[88 1 21 18 39] [54 119 80 41 72] [85 91 113 76 99] [72 19 30 106 88] [110 15 80 50 3]] [[2 24 37 21 56] [51 75 43 59 110] [61 105 36 48 18] [4 24 20 68 123] [82 111 76 9 37]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6138
Iterations	25
Duration	52.456 seconds
Population Size	80

Grafik:

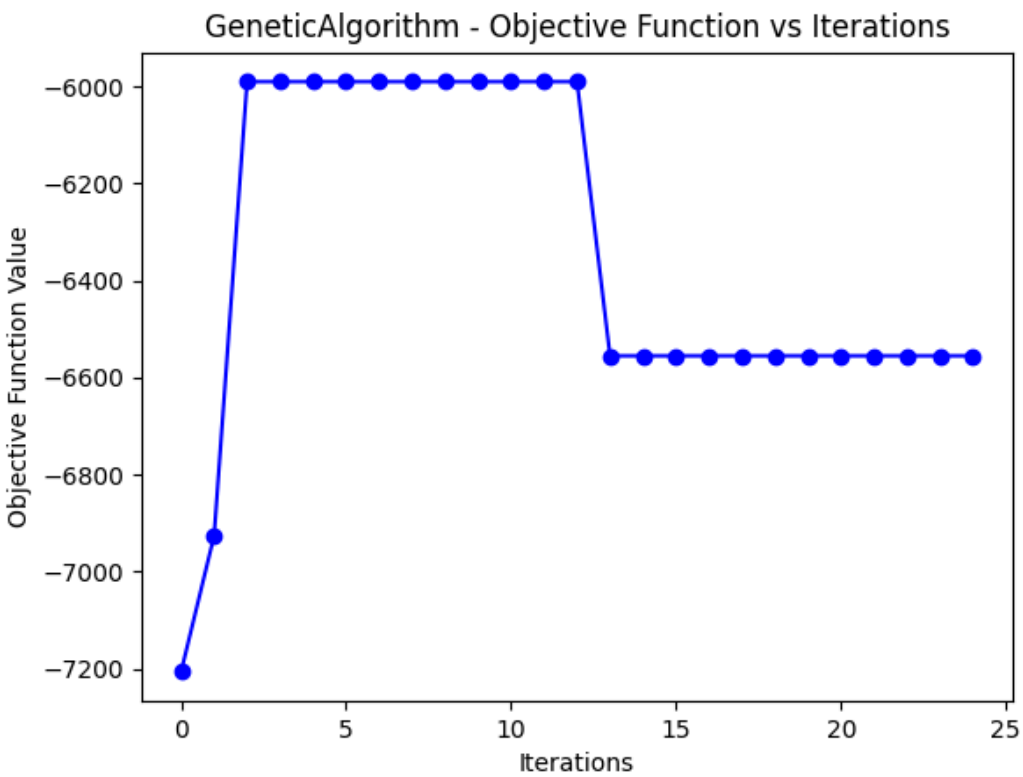


Eksperimen Banyak Iterasi 2-3 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[121 113 75 71 22] [42 28 48 98 99] [97 36 82 101 23] [95 43 62 9 106] [87 100 24 67 15]] [[47 86 92 3 114] [82 101 106 8 111] [53 119 23 68 12] [20 70 11 120 14] [105 35 2 78 77]] [[62 19 22 51 92] [81 37 94 77 99] [24 60 91 45 59] [55 23 7 14 109] [70 13 48 89 5]] [[88 1 21 18 39] [54 119 80 41 72] [85 91 113 76 99] [72 19 30 106 88] [110 15 80 50 3]] [[2 24 37 21 56] [51 75 43 59 110] [61 105 36 48 18] [4 24 20 68 123] [82 111 76 9 37]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-6555
Iterations	25
Duration	51.105 seconds
Population Size	80

Grafik:

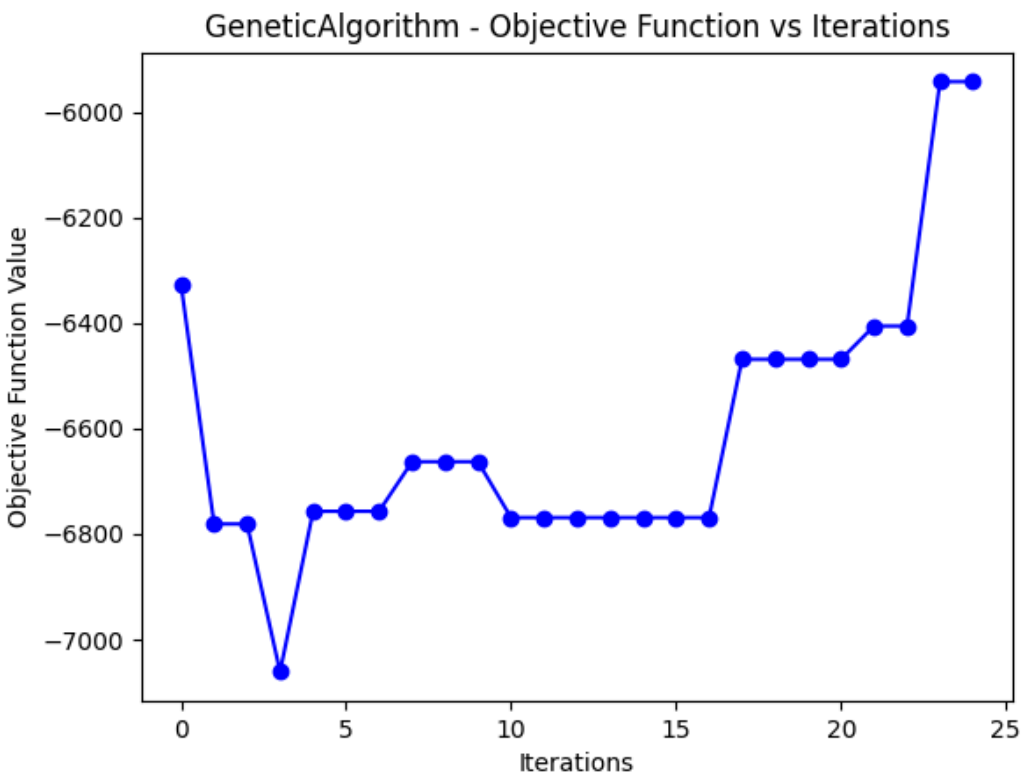


Eksperimen Banyak Iterasi 3-1 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[68 56 10 55 11] [81 44 121 62 76] [101 54 46 89 108] [94 4 78 43 96] [64 17 22 74 13]] [[67 3 86 95 64] [30 116 118 113 68] [85 20 32 9 79] [73 48 12 100 45] [58 51 38 23 47]] [[80 18 98 59 8] [41 79 34 91 114] [57 84 56 52 27] [40 110 125 87 63] [123 53 2 26 21]] [[107 14 39 75 33] [103 105 55 71 117] [9 65 77 115 88] [19 37 28 5 39] [52 94 15 99 53]] [[23 61 122 119 71] [36 115 6 2 83] [7 92 72 124 20] [21 15 75 9 29] [82 22 86 60 79]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-5943
Iterations	25
Duration	166.446 seconds
Population Size	150

Grafik:

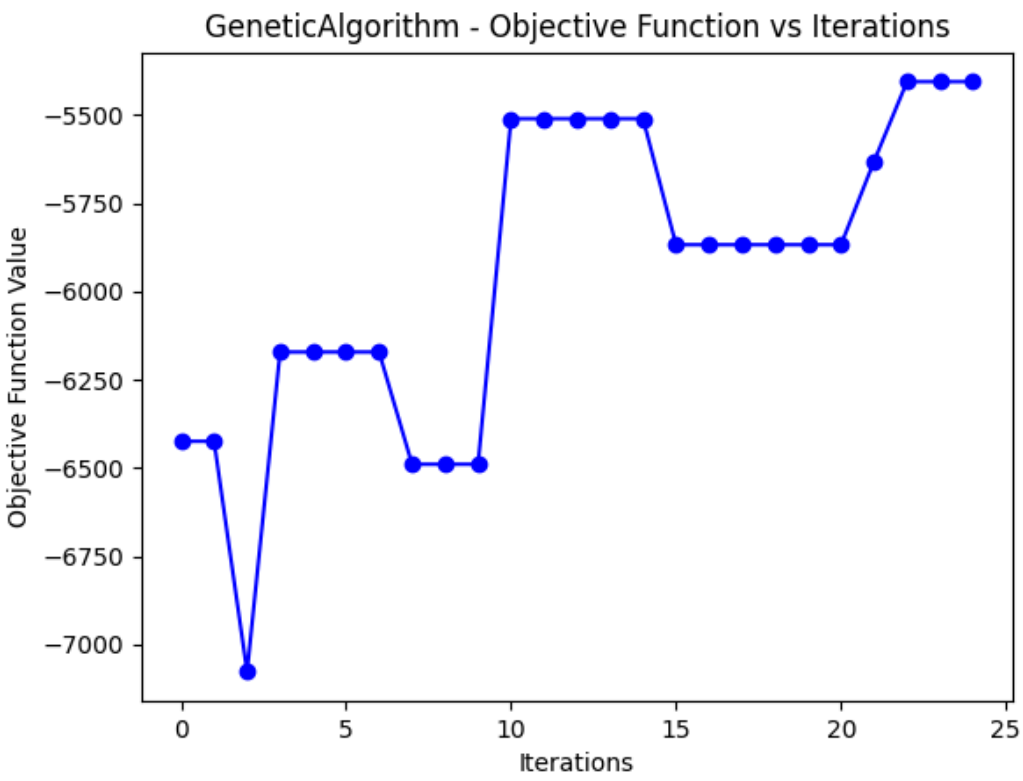


Eksperimen Banyak Iterasi 3-2 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[42 71 60 96 83] [70 67 56 40 52] [44 25 60 104 76] [65 110 86 34 20] [82 67 124 63 69]] [[47 115 74 38 41] [45 122 11 58 50] [31 102 91 25 66] [61 49 79 68 38] [40 74 108 44 30]] [[99 111 108 100 1] [101 37 47 73 111] [102 52 14 80 10] [79 45 25 58 49] [125 32 17 34 107]] [[108 51 122 98 73] [27 61 101 9 89] [109 116 103 54 79] [80 18 106 84 33] [107 69 48 62 41]] [[20 84 4 44 109] [97 82 64 57 114] [83 87 39 22 84] [23 76 93 122 90] [34 8 116 48 25]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-5405
Iterations	25
Duration	165.599 seconds
Population Size	150

Grafik:

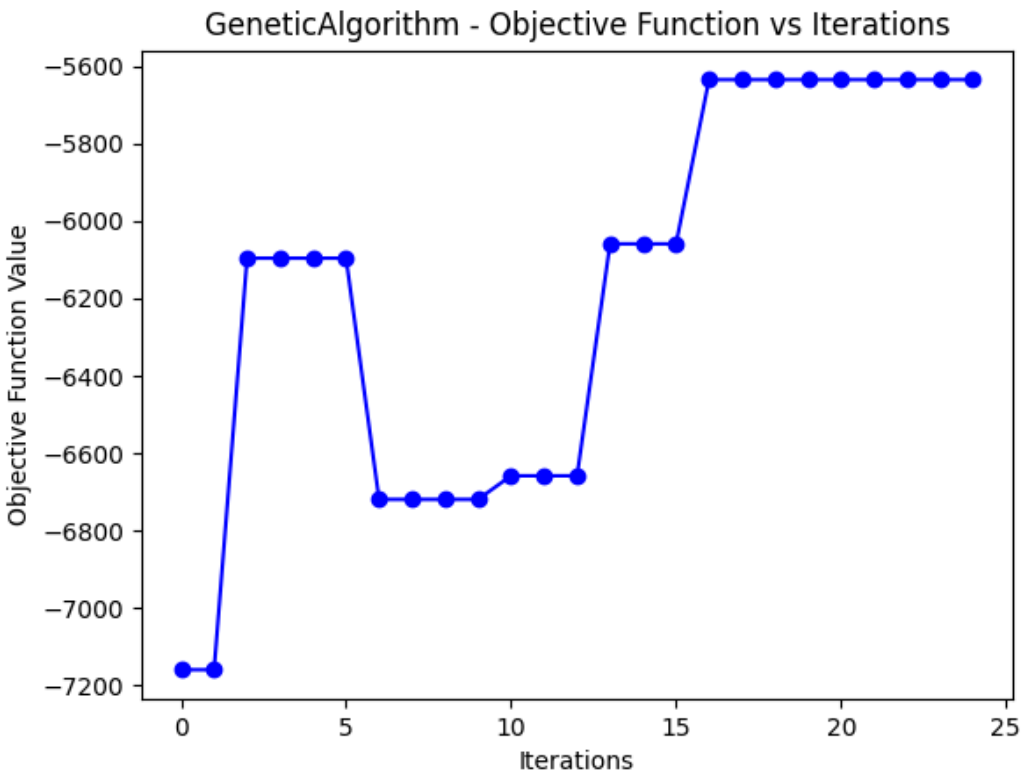


Eksperimen Banyak Iterasi 3-3 - *Genetic Algorithm*	
Initial State ---------- [[17 49 115 25 83] [85 103 29 37 61] [123 71 96 86 67] [90 14 120 1 44] [13 69 26 77 3]] [[122 74 80 52 100] [9 118 6 8 39] [47 87 50 64 62] [38 94 45 88 5] [40 111 110 117 33]] [[30 46 79 124 7] [41 121 58 105 75] [51 109 60 34 28] [76 16 73 48 68] [15 56 11 72 20]] [[57 22 21 95 78] [42 10 89 112 35] [18 59 82 106 65] [125 99 101 24 55] [66 98 23 119 70]] [[92 53 19 93 36] [84 104 54 113 63] [4 97 2 116 114] [102 31 27 107 12] [32 81 108 43 91]]	Final State ---------- [[25 4 29 48 45] [35 112 84 24 1] [83 16 99 115 13] [39 82 58 17 48] [124 99 17 29 62]] [[61 32 91 52 20] [119 105 63 36 12] [14 51 8 76 118] [18 111 110 95 4] [103 108 85 94 61]] [[110 56 47 27 75] [53 67 59 65 95] [31 100 89 107 72] [113 101 39 3 108] [17 21 125 109 43]] [[63 111 89 107 80] [60 104 76 54 11] [90 83 81 19 42] [60 44 49 55 107] [75 34 47 87 51]] [[37 74 35 115 54] [30 2 20 40 28] [15 77 38 67 112] [63 1 109 14 5] [24 62 41 83 122]]

Keterangan:

Initial Objective Value	-7244
Final Objective Value	-5635
Iterations	25
Duration	288.958 seconds
Population Size	150

Grafik:



4.2. Perbandingan Waktu Algoritma Local Search

Berikut adalah perbandingan waktu eksekusi algoritma *Steepest Ascent Hill-climbing, Hill-climbing with Sideways Move, Random Restart Hill-climbing, Stochastic Hill-climbing* dan *Simulated Annealing*:

Algoritma	Waktu Eksekusi Eksperimen 1	Waktu Eksekusi Eksperimen 2	Waktu Eksekusi Eksperimen 3	Waktu Rata-rata Eksekusi
Steepest Ascent Hill-climbing	Iterasi: 15 60.26 s	Iterasi: 25 107.33 s	Iterasi: 50 219.64 s	129.08 s
Hill-climbing with Sideways Move	Iterasi: 25 Max sideways move: 5 112.66 s	Iterasi: 25 Max sideways move: 10 111.94 s	Iterasi: 25 Max sideways move: 25 108.29 s	110.96 s
Random Restart Hill-climbing	Restarts: 2 Iterasi per restart: 25 205.75 s	Restarts: 3 Iterasi per restart: 25 496.62 s	Restarts: 5 Iterasi per restart: 25 512.25 s	404.87 s
Stochastic Hill-climbing	Iterasi: 1000 1.10 s	Iterasi: 2000 2.16 s	Iterasi: 5000 5.12 s	8.38 s
Simulated Annealing	Iterasi: 1000 Frek. stuck: 36 1.56 s	Iterasi: 2000 Frek. stuck: 103 3.05 s	Iterasi: 5000 Frek. stuck: 340 5.12 s	3.24 s

4.3. Perbandingan Hasil Solusi Algoritma Local Search

Berikut adalah perbandingan nilai objektif yang dihasilkan algoritma *Steepest Ascent Hill-climbing, Hill-climbing with Sideways Move, Random Restart Hill-climbing, Stochastic Hill-climbing* dan *Simulated Annealing*:

Algoritma	Selisih dan Nilai Akhir *Objective Function* Eksperimen 1	Selisih dan Nilai Akhir *Objective Function* Eksperimen 2	Selisih dan Nilai Akhir *Objective Function* Eksperimen 3	Rata-rata Selisih dan Nilai Akhir *Objective Function*
Steepest Ascent	S: 4404	S: 5415	S: 6147	S: 5322

Hill-climbing	NA: -2840	NA: -1829	NA: -1097	NA: -1922
Hill-climbing with Sideways Move	S: 5415 NA: -1829	S: 5415 NA: -1829	S: 5415 NA: -1829	S: 5415 NA: -1829
Random Restart Hill-climbing	S: 5415 NA: -1829	S: 5415 NA: -1829	S: 5415 NA: -1829	S: 5415 NA: -1829
Stochastic Hill-climbing	S: 4953 NA: -2291	S: 5411 NA: -1833	S: 6328 NA: -916	S: 5564 NA: -1680
Simulated Annealing	S: 4953 NA: -2291	S: 5529 NA: -1715	S: 6244 NA: -1020	S: 5568.67 NA: -1675.34

4.4. Perbandingan Hasil Genetic Algorithm

Pencarian *Genetic Algorithm* dengan ukuran populasi sebanyak 80 dan variasi iterasi sebagai berikut:

Eksperimen	Iterasi	Waktu Eksekusi	Nilai Akhir
1.1	15	28.53	-6359
1.2	15	28.95	-5913
1.3	15	28.64	-6985
Rata-rata		**28.71**	**-6419**
2.1	25	47.56	-6530
2.2	25	47.77	-6351
2.3	25	55.43	-6427
Rata-rata		**50.25**	**-6436**
3.1	50	110.99	-6213
3.2	50	98.55	-6903
3.3	50	107.57	-6521
Rata-rata		**105.7**	**-6545**

Pencarian *Genetic Algorithm* dengan banyak iterasi sebanyak 25 dan variasi populasi sebagai berikut:

Eksperimen	Populasi	Waktu Eksekusi	Nilai Akhir
1.1	50	73.46	-6029
1.2	50	20.65	-7297
1.3	50	19.53	-5899
Rata-rata		**37.88**	**-6408.3**
2.1	80	50.97	-6224
2.2	80	52.45	-6138
2.3	80	51.10	-6555
Rata-rata		**51.51**	**-6305.7**
3.1	150	166.47	-5943
3.2	150	165.60	-5405
3.3	150	288.96	-5635
Rata-rata		**207.01**	**-5661**

4.5. Analisis Algoritma Local Search

Seberapa dekat tiap-tiap algoritma bisa mendekati global optima dan mengapa hasilnya demikian?

Algoritma Steepest Ascent Hill-climbing menunjukkan variasi yang cukup besar dalam nilai akhir fungsi objektifnya, dengan rata-rata selisih sebesar 5322 dan nilai akhir -1922. Selisih yang cukup tinggi ini menunjukkan bahwa algoritma mampu mencapai perbaikan nilai fungsi objektif yang signifikan, tetapi hasil akhirnya tetap jauh dari global optimum. Hal ini disebabkan oleh sifat deterministik dan pendekatan greedy yang membuat algoritma ini cenderung terjebak pada optimum lokal tanpa mekanisme yang cukup untuk keluar dari jebakan tersebut. Setiap iterasi menghasilkan nilai akhir yang berbeda, tergantung pada posisi

awal di ruang pencarian, tetapi semuanya tidak berhasil mendekati nilai optimum yang lebih tinggi.

Hill-climbing dengan Sideways Move menunjukkan konsistensi yang sangat tinggi di semua eksperimen dengan selisih rata-rata sebesar 5415 dan nilai akhir -1829. Dengan adanya sideways move, algoritma mampu mengatasi jebakan lokal yang dangkal, tetapi tidak cukup kuat untuk keluar dari jebakan yang lebih dalam. Hasil ini menunjukkan bahwa meskipun sideways move memberikan manfaat tambahan, efektivitas algoritma ini dalam mendekati global optimum tetap terbatas, terutama ketika terdapat jebakan local optimum yang lebih kompleks.

Random Restart Hill-climbing menghasilkan nilai akhir yang identik dengan Hill-climbing with Sideways Move, dengan rata-rata selisih 5415 dan nilai akhir -1829 di semua eksperimen. Strategi restart memungkinkan algoritma untuk keluar dari jebakan lokal dengan memulai ulang pencarian dari titik awal yang berbeda. Namun, hasil yang sama di setiap eksperimen menunjukkan bahwa titik awal yang berbeda tersebut tidak cukup untuk mendekati global optimum yang lebih tinggi. Dalam konteks ini, meskipun random restart efektif untuk memperbaiki kekurangan Hill-climbing biasa, hasil akhir menunjukkan bahwa algoritma ini masih belum mampu mencapai solusi yang benar-benar optimal tanpa jumlah restart yang jauh lebih tinggi atau penambahan mekanisme eksplorasi lebih lanjut.

Algoritma Stochastic Hill-climbing menghasilkan rata-rata selisih sebesar 5564 dan nilai akhir -1680. Algoritma ini menunjukkan hasil yang bervariasi di tiap eksperimen, dengan nilai akhir yang berbeda-beda tergantung pada pilihan acak yang diambil selama proses pencarian. Sifat stokastik dalam algoritma ini membantu memperluas eksplorasi dengan mencari solusi di area yang lebih luas di ruang pencarian. Meskipun demikian, fluktuasi hasil menunjukkan bahwa algoritma ini masih cenderung mendekati local optimum di beberapa titik, namun lebih dekat dengan global optimum dibandingkan algoritma Hill-climbing yang deterministik. Keberhasilan algoritma ini tergantung pada kualitas randomisasi dan keputusan probabilistik, yang memungkinkan pencarian di area yang tidak diakses oleh algoritma deterministik.

Simulated Annealing menunjukkan performa terbaik di antara algoritma lainnya, dengan rata-rata selisih sebesar 5568,67 dan nilai akhir -1675,34. Algoritma ini berhasil mendekati

global optimum dengan lebih baik dibandingkan algoritma lainnya karena pendekatan *cooling schedule* yang diterapkan. Pada tahap awal pencarian, eksplorasi dilakukan lebih luas sehingga algoritma dapat keluar dari jebakan local optimum dan mengakses area yang belum pernah dicapai. Seiring waktu, *cooling schedule* ini mempersempit ruang pencarian, sehingga algoritma fokus pada area yang lebih menjanjikan. Simulated Annealing mampu mencapai keseimbangan antara eksplorasi dan eksploitasi, yang memungkinkan algoritma ini mendekati global optimum secara efektif. Nilai akhir yang lebih rendah secara konsisten menunjukkan bahwa algoritma ini adalah yang paling berhasil dalam pendekatan solusi mendekati optimum dibandingkan algoritma lain.

Pada pendekatan pertama, Genetic Algorithm dijalankan dengan ukuran populasi tetap sebanyak 80 individu dan berbagai jumlah iterasi (15, 25, dan 50) untuk melihat efek iterasi pada hasil akhir. Pada iterasi yang terlalu rendah (15), algoritma tidak memiliki cukup waktu untuk melakukan eksplorasi mendalam. Sementara iterasi yang lebih tinggi (25) memberikan sedikit perbaikan, sedangkan iterasi yang terlalu tinggi (50) justru dapat menyebabkan masalah seperti overfitting pada solusi lokal, yang menjauhkan algoritma dari global optimum. Untuk mendekati global optimum, diperlukan kombinasi iterasi yang optimal agar algoritma dapat mempertahankan keragaman genetik dan mengeksplorasi solusi potensial secara efisien. Pada pendekatan kedua, GA dijalankan dengan jumlah iterasi tetap (25) dan variasi ukuran populasi (50, 80, dan 150) untuk memahami pengaruh ukuran populasi terhadap hasil akhir. Genetic Algorithm dengan populasi lebih besar lebih efektif dalam mendekati global optimum daripada hanya meningkatkan jumlah iterasi pada populasi tetap.

Bagaimana perbandingan hasil pencarian tiap-tiap algoritma dengan algoritma local search yang lain?

Berdasarkan eksperimen kali ini, algoritma *Steepest Ascent Hill-Climbing, Hill-climbing with Sideways Move, Random Restart Hill-climbing, Stochastic Hill-Climbing,* dan *Simulated Annealing* dapat mendekati global optimal hingga nilai objektifnya sekitar -1900 hingga -1600. Algoritma yang menghasilkan rata-rata nilai objektif paling mendekati global optimum adalah *Simulated Annealing*. Hal tersebut terjadi karena adanya mekanisme untuk dapat lepas dari jebakan lokal optimum. Sedangkan untuk *Genetic Algorithm* hanya dapat

mencapai nilai objektif sekitar -6000. Hal tersebut terjadi karena *Genetic Algorithm* ketika melakukan penyilangan *parents* dapat menyebabkan nilai objektif yang sangat berbeda.

Bagaimana perbandingan durasi proses pencarian tiap algoritma relatif terhadap algoritma lainnya?

Durasi pencarian algoritma *Steepest Ascent Hill-Climbing, Hill-climbing with Sideways Move,* dan *Random Restart Hill-climbing* terlihat jauh lebih lama dibandingkan algoritma lain. Hal tersebut dikarenakan ketiga algoritma tersebut harus menghitung nilai *objective function* dari seluruh *neighbor current state* pada setiap iterasinya. Bahkan pada *Random Restart Hill-climbing* dilakukan sebanyak jumlah restartnya pula. Sehingga, untuk mencapai nilai *objective function* yang sama dengan cara algoritma lain, dibutuhkan lebih lama meskipun iterasi yang dilakukan lebih sedikit.

Sedangkan untuk algoritma *Stochastic Hill-climbing* dan *Simulated Annealing* berjalan dengan sangat cepat dibandingkan algoritma lain. Hal tersebut karena *state* yang diambil adalah pilihan acak tanpa menghitung nilai *objective function* dari *neighbor* terlebih dahulu. Sehingga, untuk mencapai nilai *objective function* yang sama dengan cara algoritma lain, dibutuhkan lebih sedikit waktu meskipun iterasi yang dilakukan jauh lebih banyak.

Lalu, untuk *Genetic Algorithm* waktu yang dibutuhkan tidak selama *Random Restart Hill-climbing,* tetapi juga tidak secepat *Simulated Annealing*. Hal tersebut karena pada *Genetic Algorithm* terdapat tahap perhitungan *objective function* pada masing-masing *parent* yang cukup membutuhkan waktu meski tidak sebanyak jumlah *neighbor*.

Dari keenam algoritma local search, didapatkan rata-rata waktu pencarian paling cepat dilakukan oleh *Simulated Annealing*, serta waktu pencarian paling lama dilakukan oleh *Genetic Algorithm*. Hal tersebut sesuai dengan cara pemilihan neighbor terpilih yang memang sangat berbeda, perhitungan nilai objektif sangat menyita waktu untuk permasalahan ini.

Seberapa konsisten hasil akhir yang didapatkan dari tiap-tiap eksperimen yang dilakukan?

Konsistensi algoritma *Steepest Ascent Hill-Climbing, Hill-climbing with Sideways Move,* dan *Random Restart Hill-climbing* lebih terjaga, terlihat dari tabel dengan menaikkan jumlah iterasi, hasil akhir yang dihasilkan juga konsisten semakin bagus. Hal tersebut terjadi karena dalam menentukan perpindahan *state* dilakukan pertimbangan perubahan *objective function*-nya.

Sedangkan untuk algoritma *Stochastic Hill-climbing* dan *Simulated Annealing*, meskipun masih cenderung rata untuk setiap nilai objektifnya tetapi tidak dapat dikatakan konsisten, peningkatan iterasi yang signifikan tidak menunjukkan peningkatan *objective function* yang signifikan pula. Hal tersebut terjadi karena meskipun iterasi yang dilakukan lebih banyak tetapi *state* yang diambil adalah acak tanpa mempertimbangkan nilai *objective function* yang dapat diambil.

Untuk algoritma *Genetic Algorithm* konsistensi tidak terjaga, terlihat dari hasil nilai *objective function* yang sangat beragam meskipun memiliki parameter iterasi dan populasi yang sama. Hal tersebut karena adanya tahap mutasi yang dapat sangat menentukan perbedaan langkah selanjutnya.

Bagaimana pengaruh banyak iterasi dan jumlah populasi terhadap hasil akhir pencarian pada Genetic Algorithm?

Berdasarkan eksperimen kali ini, terlihat bahwa semakin banyak iterasi yang dilakukan maka semakin jauh nilai objektifnya terhadap global maksimal, sedangkan semakin banyak populasinya maka semakin dekat nilai objektifnya terhadap global maksimal. Hal ini terjadi karena semakin banyak populasi yang dilibatkan, maka semakin banyak pertimbangan nilai objektif suatu keadaan yang dihitung. Sedangkan semakin banyak iterasi tidak dapat menunjukkan semakin mendekati global optimum karena setiap iterasi tidak menjamin nilai objektif semakin baik. Pada kasus ini yang cukup kompleks, persilangan dua parents yang memiliki nilai objektif baik, tidak dapat memastikan bahkan memiliki probabilitas yang cukup kecil untuk menghasilkan populasi baru dengan nilai objektif yang lebih baik.

BAB V
PENGERJAAN BONUS

5.1. Algoritma Player

```python
import tkinter as tk
from tkinter import ttk, filedialog
from utils.file_manager import load_experiment_results
import time
import threading

class ReplayPlayer:
    def __init__(self, root, results=None):
        self.root = root
        self.results = results if results else {}
        self.states = self.results.get("state_tracker", [])
        self.current_index = 0
        self.playing = False
        self.playback_speed = 1.0
        self.total_iterations = len(self.states)
        self.filename = "No file loaded"
        self.setup_ui()

        self.root.protocol("WM_DELETE_WINDOW", self.on_close)

    def setup_ui(self):
        self.frame = tk.Frame(self.root)
        self.frame.pack()

        # Filename label
        self.filename_label = tk.Label(self.frame, text=f"File: {self.filename}", font=("Arial", 12))
        self.filename_label.pack(pady=5)

        # Iteration label
        self.iteration_label = tk.Label(self.frame, text="Iteration: 0", font=("Arial", 11))
        self.iteration_label.pack(pady=5)

        # Canvas for cube
        self.canvas = tk.Canvas(self.frame, width=1000, height=400)
        self.canvas.pack()

        # Progress bar
        self.progress_var = tk.DoubleVar()
        self.progress_bar = ttk.Scale(
            self.frame, from_=0, to=max(0, self.total_iterations - 1),
            orient="horizontal", variable=self.progress_var, command=self.on_seek
        )
        self.progress_bar.pack(fill="x", pady=10)

        # Play/Pause button
        self.play_button = tk.Button(self.frame, text="Play", command=self.toggle_play)
        self.play_button.pack(side="left")

        # Speed slider
        self.speed_label = tk.Label(self.frame, text="Speed:")
        self.speed_label.pack(side="left")
        self.speed_var = tk.DoubleVar(value=1.0)
        self.speed_scale = ttk.Scale(
            self.frame, from_=0.1, to=2.0,
            orient="horizontal", variable=self.speed_var, command=self.on_speed_change
        )
        self.speed_scale.pack(side="left")

        # Load button
        self.load_button = tk.Button(self.frame, text="Load Experiment", command=self.load_experiment)
        self.load_button.pack(side="right")

        # Init thread
        self.replay_thread = threading.Thread(target=self.replay)
        self.replay_thread.daemon = True
```

Gambar 5.1.1. Source Code Bonus (1)

```python
def toggle_play(self):
    if not self.states:
        print("No experiment loaded. Please load an experiment first.")
        return

    self.playing = not self.playing
    if self.playing:
        self.play_button.config(text="Pause")
        if not self.replay_thread.is_alive():
            self.replay_thread = threading.Thread(target=self.replay)
            self.replay_thread.start()
    else:
        self.play_button.config(text="Play")

def replay(self):
    while self.playing and self.current_index < self.total_iterations:
        if not self.root.winfo_exists():
            break
        self.display_state(self.states[self.current_index])
        self.current_index += 1
        self.progress_var.set(self.current_index)
        self.iteration_label.config(text=f"Iteration: {self.current_index}")
        time.sleep(1 / self.playback_speed)
        if self.current_index >= self.total_iterations:
            self.playing = False
            self.play_button.config(text="Play")

def display_state(self, state):
    self.canvas.delete("all")
    layer_spacing_x = 200
    padding_x = 40
    cell_size = 30

    for layer_idx, layer in enumerate(state):
        x_offset = padding_x + layer_idx * layer_spacing_x
        for i in range(len(layer)):
            for j in range(len(layer[i])):
                x = j * cell_size + x_offset
                y = i * cell_size + 50
                self.canvas.create_text(x, y, text=str(layer[i][j]), font=("Arial", 10))

def on_seek(self, value):
    if self.states:
        self.current_index = int(float(value))
        self.display_state(self.states[self.current_index])
        self.iteration_label.config(text=f"Iteration: {self.current_index}")

def on_speed_change(self, value):
    self.playback_speed = float(value)

def load_experiment(self):
    filename = filedialog.askopenfilename(initialdir="assets/results",
        title="Select Experiment File", filetypes=[("JSON files", "*.json")])
    if filename:
        loaded_results = load_experiment_results(filename)
        if loaded_results:
            self.filename = filename.split("/")[-1]
            self.filename_label.config(text=f"Filename: {self.filename}")
            self.results = loaded_results
            self.states = self.results.get('state_tracker', [])
            self.total_iterations = len(self.states)
            self.progress_bar.config(to=max(0, self.total_iterations - 1))
            self.current_index = 0
            self.display_state(self.states[self.current_index])
            self.iteration_label.config(text="Iteration: 0")

def on_close(self):
    self.playing = False
    self.root.destroy()
```

Gambar 5.1.2. Source Code Bonus (2)

5.2. Dokumentasi

Berikut adalah link video contoh replay satu kasus pencarian:

https://youtu.be/KAj8uJs2z6s

BAB VI
PENUTUP

6.1. Kesimpulan

Diagonal magic cube merupakan kubus yang tersusun dari angka 1 hingga n^3 tanpa pengulangan dengan n adalah panjang sisi pada kubus tersebut, serta jumlah angka tiap baris, kolom, tiang, dan diagonalnya sama. Objective function dari permasalahan Diagonal Magic Cube adalah dengan menghitung perbedaan antara jumlah setiap baris, kolom, dan diagonal terhadap nilai target optimal (315). Dengan *objective function* tersebut telah dibuat enam algoritma local search yaitu *Steepest Ascent Hill-Climbing, Hill-climbing with Sideways Move, Random Restart Hill-Climbing, Stochastic Hill-Climbing, Simulated Annealing,* serta *Genetic Algorithm* dalam bahasa python.

Setelah dilakukan eksperimen dengan *initial state* yang sama, didapatkan algoritma yang paling cepat dalam melakukan pencarian adalah algoritma **Simulated Annealing** dengan rata-rata waktu **3.24 detik**. Sedangkan algoritma yang melakukan peningkatan nilai *objective function* paling signifikan adalah algoritma **Simulated Annealing** dengan rata-rata perubahan nilai *objective function* sebesar **5568.67** hingga mendekati global optimum dengan rata-rata nilai objektif **-1675.34**. Konsistensi hasil pencarian terjaga ketika penentuan *neighbor* berdasarkan nilai objektif terbaik yaitu pada algoritma ***Steepest Ascent Hill-Climbing, Hill-climbing with Sideways Move,* dan *Random Restart Hill-climbing.*** Sedangkan untuk algoritma yang kurang cocok untuk studi kasus ini adalah ***Genetic Algorithm*,** karena berdasarkan percobaan menghasilkan rata-rata hasil akhir nilai objektif paling buruk meskipun sudah dilakukan variasi nilai iterasi dan populasi.

6.2. Saran

- Mencoba untuk mengoptimasi cara membuat algoritma agar pencarian dengan persoalan besar seperti Diagonal Magic Cube Number ukuran 5x5x5 dapat berjalan lebih cepat lagi.
- Dapat membuat visualisasi angka dalam kubus dengan lebih menarik.

LAMPIRAN

Tabel pembagian tugas:

NIM	Nama	Tugas
13522069	Nabila Shikoofa Muida	Base algorithm, steepest ascent, sideways move, laporan
13522096	Novelya Putri Ramadhani	Genetic algorithm, visualisasi cube, file manager, main, video player
13522102	Hayya Zuhailii Kinasih	Random restart, stochastic, experiments
13522104	Diana Tri Handayani	Simulated annealing, data processing, analisis dan laporan

Tabel pengerjaan bonus:

Bonus	**Pengerjaan**
Implementasi seluruh algoritma *Hill-climbing*	Terselesaikan
Implementasi visualisasi video	Terselesaikan

DAFTAR PUSTAKA

[1] Magischvierkant.com. *Magic Features of Three-Dimensional Magic Cubes*. Available at: https://www.magischvierkant.com/three-dimensional-eng/magic-features/. Accessed October 2, 2024.

[2] Trump.de. *Magic Cubes - The Magic of the Cube*. Available at: https://www.trump.de/magic-squares/magic-cubes/cubes-1.html. Accessed October 2, 2024.

[3] Wikipedia. *Magic Cube*. Available at: https://en.wikipedia.org/wiki/Magic_cube. Accessed October 2, 2024.