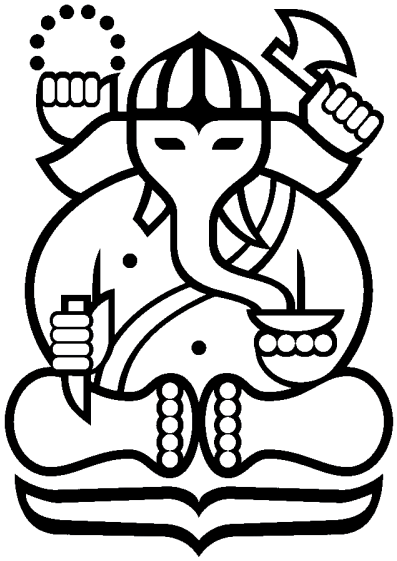
TUGAS BESAR IF3070 DASAR INTELEGENSI ARTIFISIAL
Pencarian Solusi *Diagonal Magic Cube* dengan *Local Search*



Kelompok 36
Anggota Kelompok :

K01 Bihurin Salsabila Firdaus	18222015@std.stei.itb.ac.id
K01 Dinda Thalia Fahira	18222055@std.stei.itb.ac.id
K01 Muhammad Nurul Hakim	18222097@std.stei.itb.ac.id
K01 Dahayu Ramaniya Aurasindu	18222099@std.stei.itb.ac.id

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

Deskripsi Persoalan

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

- 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

Pada tugas besar ini, diharapkan untuk menyelesaikan permasalahan Diagonal Magic Cube dengan ukuran 5x5x5 yang angka-angkanya masih tersusun secara acak dengan menggunakan algoritma *local search*. Angka-angka yang tertera pada kubus adalah angka 1 hingga 5^3. Langkah yang boleh dilakukan untuk tiap iterasi algoritma *local search* adalah menukar posisi dari 2 angka pada kubus tersebut.

Pembahasan

Pemilihan Objective Function

$$f(C) = -\sum_{f \in features} \left| \left(\sum_{x \in f} x \right) - 315 \right|$$

dengan:
C = konfigurasi angka pada kubus
f = feature
x = elemen feature

Fungsi objektif yang dipilih adalah **negatif** dari total selisih absolut antara jumlah elemen-elemen pada feature dengan target sum.

Feature dapat berupa:
- Baris
- Kolom
- Tiang
- Diagonal Ruang
- Diagonal Bidang

Sementara itu, target sum 315 didapatkan dari hasil penelitian oleh (Walter Trump dan Christian Boyer, 2003) yang menemukan solusi Diagonal Magic Cube 5x5x5.

Trump and Boyer's order 5 perfect magic cube

25	16	80	104	90
115	98	4	1	97
42	111	85	2	75
66	72	27	102	48
67	18	119	106	5

91	77	71	6	70
52	64	117	69	13
30	118	21	123	23
26	39	92	44	114
116	17	14	73	95

47	61	45	76	86
107	43	38	33	94
89	68	63	58	37
32	93	88	83	19
40	50	81	65	79

31	53	112	109	10
12	82	34	87	100
103	3	105	8	96
113	57	9	62	74
56	120	55	49	35

121	108	7	20	59
29	28	122	125	11
51	15	41	124	84
78	54	99	24	60
36	110	46	22	101

Penjelasan Implementasi Algoritma Local Search

Terdapat beberapa class yang dibuat untuk mendukung implementasi, yaitu State dan MagicCube. State dibuat dalam bentuk matriks 5x5x5 yang merepresentasikan konfigurasi elemen kubus. Kubus memiliki level yang didapatkan dengan cara melakukan pemotongan secara horizontal. Oleh karena itu, setiap matriks 5x5 yang disimpan pada state dapat dibayangkan sebagai permukaan kubus jika dilihat dari atas.

Gambar Kode Class State

```python
import numpy as np

class State():
    def __init__(self, state=None):
        if state is not None:
            self.state = state.reshape(5, 5, 5)
        else:
            self.state = self.generate_state()
        self.value = self.evaluate()

    def generate_state(self):
        state = np.random.choice(np.arange(1, 126), size=125, replace=False).reshape((5, 5, 5))
        return state

    def evaluate(self):
        # pillars (25)
        pillar_sum = np.sum(self.state, axis=0).flatten()

        # rows (25)
        row_sum = np.sum(self.state, axis=2).flatten()

        # collumns (25)
        column_sum = np.sum(self.state, axis=1).flatten()

        # space diagonals (4)
        space_diagonals = []
        space_diagonals.append([self.state[i, i, i] for i in range(5)])
        space_diagonals.append([self.state[i, i, 4 - i] for i in range(5)])
        space_diagonals.append([self.state[i, 4 - i, i] for i in range(5)])
        space_diagonals.append([self.state[4 - i, i, i] for i in range(5)])
        space_diagonal_sum = np.sum(space_diagonals, axis=1)

        # diagonals (30)
        diagonals = []
        for n in range(5):
            # plane xy
            diagonals.append([self.state[i, i, n] for i in range(5)])
            diagonals.append([self.state[i, 4-i, n] for i in range(5)])
            # plane xz
            diagonals.append([self.state[i, n, i] for i in range(5)])
            diagonals.append([self.state[i, n, 4-i] for i in range(5)])
            # plane yz
            diagonals.append([self.state[n, i, i] for i in range(5)])
            diagonals.append([self.state[n, i, 4-i] for i in range(5)])
        diagonal_sum = np.sum(diagonals, axis=1)

        # collect all features
        feature_sum = np.concatenate((pillar_sum, row_sum, column_sum, space_diagonal_sum, diagonal_sum))

        # calculate state value
        self.value = 0
        for x in feature_sum:
            self.value -= abs(x-315)

        return self.value

    def is_goal_state(self):
        return self.value == 0

    def __str__(self):
        return np.array2string(self.state, separator=' ')
```

Class State

Representasi state pada algoritma local search.

Attributes

- state
  Menyimpan kumpulan angka yang merepresentasikan konfigurasi elemen kubus
- value
  Menyimpan state value yang merupakan hasil perhitungan objective function

Methods

- __init__
  Inisialisasi state dengan method generate state dan mengisi value dengan method evaluate
- generate_state
  Menghasilkan sebuah numpy array berisi 125 elemen
- is_goal_state
  Mengecek apakah state ini adalah goal state
- evaluate
  Menghitung state value menggunakan objective function
- __str__
  Mencetak state dalam bentuk matriks 5x5x5

Gambar Kode Class MagicCube

```python
import time
import matplotlib.pyplot as plt
from SteepestAscentHC import SteepestAscentHC as SAHC
from HCWithSidewaysMove import HCWithSidewaysMove as HCSM
from StochasticHC import StochasticHC as SHC
from RandomRestartHC import RandomRestartHC as RRHC
from SimulatedAnnealing import SimulatedAnnealing as SA
from GeneticAlgorithm import GeneticAlgorithm as GA

# Hill Climbing With Sideways Move
MAX_SIDEWAYS_MOVE = 10

# Stochastic Hill Climbing
SHC_MAX_ITERATION = 100

# Random Restart Hill Climbing
MAX_RESTART = 2

# Simulated Annealing
INITIAL_TEMPERATURE = 100.0
COOLING_RATE = 1.0

# Genetic Algorithm
POPULATION_SIZE = 50
GA_MAX_ITERATION = 100
MUTATION_RATE = 0.2

class MagicCube():
    def __init__(self, alg):
        if alg == 'sahc':
            self.alg = alg
            self.algorithm = SAHC()
        elif alg == 'hcsm':
            self.alg = alg
            self.algorithm = HCSM(MAX_SIDEWAYS_MOVE)
        elif alg == 'shc':
            self.alg = alg
            self.algorithm = SHC(SHC_MAX_ITERATION)
        elif alg == 'rrhc':
            self.alg = alg
            self.algorithm = RRHC(MAX_RESTART)
        elif alg == 'sa':
            self.alg = alg
            self.algorithm = SA(INITIAL_TEMPERATURE, COOLING_RATE)
        elif alg == 'ga':
            self.alg = alg
            self.algorithm = GA(POPULATION_SIZE, GA_MAX_ITERATION, MUTATION_RATE)
        else:
            print('Invalid algorithm!')

    def solve(self):
        assert self.algorithm is not None, "No algorithm selected!"
        start_time = time.time()
        result = self.algorithm.search()
        end_time = time.time()
        self.final_state = result[0]
        values = result[1]
        print(self.final_state)
        print(self.final_state.value)
        print(f'duration: {end_time-start_time} seconds')

        # Plot
        if self.alg == 'rrhc':
            plt.title("State Value Plot")
            plt.xlabel("Iteration")
            plt.ylabel("Value")
            for key, value in values.items():
                plt.plot(value, label=str(key))
            plt.legend()
            plt.show()
        else:
            if self.alg == 'sa':
                probs = result[2]
                plt.title("Probabilities Plot")
                plt.xlabel("Iteration")
                plt.ylabel("Probabilities")
                plt.plot(probs)
                plt.show()
            plt.title("State Value Plot")
            plt.xlabel("Iteration")
            plt.ylabel("Value")
            plt.plot(values)
            plt.show()
```

Class MagicCube
Kelas yang menyimpan seluruh algoritma local search
Attributes
- alg
  menyimpan singkatan nama algoritma local search
- algorithm
  menyimpan instance salah satu kelas algoritma local search

Methods
- __init__
  inisialisasi algoritma local search
- solve
  menyelesaikan kubus dengan algoritma local search yang dipilih dan menampilkan grafik hasil eksperimen

1. Hill Climbing
   Hill Climbing diimplementasikan dengan membuat sebuah parent class bernama Hill Climbing dan menurunkan empat buah child class, yaitu SteepestAscentHC, HCWithSidewaysMove, StochasticHC, dan RandomRestartHC. Successor dari current state didapatkan dengan cara mempertukarkan dua buah elemen pada current state. Pembangkitan successor menghasilkan sejumlah 7.750 state yang dapat dihitung dengan C_2^{125} dengan 125 merupakan jumlah elemen pada state dan 2 merupakan jumlah elemen yang mengalami pertukaran.

Gambar Kode Class HillClimbing

```python
from state import State
import numpy as np

class HillClimbing():
    def __init__(self):
        self.current_state = State()

    def generate_successors(self):
        indices = [(i, j, k) for i in range(5) for j in range(5) for k in range(5)]
        successors = []
        for i in range(len(indices)):
            for j in range(i + 1, len(indices)):  # Pastikan tidak menukar elemen yang sama
                index_1 = indices[i]
                index_2 = indices[j]
                successor = State()
                successor.state = np.copy(self.current_state.state)
                successor.state[index_1], successor.state[index_2] = self.current_state.state[index_2], self.current_state.state[index_1]
                successor.value = successor.evaluate()
                successors.append(successor)
        successors = np.array(successors)
        return successors
```

Class HillClimbing
Kelas yang menjadi dasar bagi berbagai variasi algoritma Hill Climbing

Attributes

- current_state
  Menyimpan instance kelas State sebagai representasi state terkini

Methods

- __init__
  Inisialisasi current state dengan sebuah state yang di-generate secara acak
- generate_successors
  Menghasilkan array of state elemen yang merupakan kumpulan suksesor dari current state

a. Steepest Ascent Hill Climbing
   Implementasi algoritma Steepest Ascent Hill Climbing dibuat sesuai dengan algoritma pada buku Russell. Neighbor didapatkan dari highest value successor. Kondisi terminasi terjadi ketika tidak ada lagi neighbor yang lebih baik daripada current state.

Gambar Kode Class SteepestAscentHC

```python
from HillClimbing import HillClimbing

class SteepestAscentHC(HillClimbing):
    def __init__(self):
        super().__init__()

    def select_highest_value_successor(self, successors):
        best_successor_index = 0
        best_value = successors[0].value
        for i in range(1, len(successors)):
            next_value = successors[i].value
            if (next_value > best_value):
                best_successor_index = i
                best_value = next_value
        neighbor = successors[best_successor_index]
        return neighbor

    def search(self):
        print(self.current_state)
        print(f'initial state value: {self.current_state.value}')
        values = [self.current_state.value]
        iteration_count = 0
        while not self.current_state.is_goal_state():
            successors = self.generate_successors() # Hasilkan tetangga dari current state
            neighbor = self.select_highest_value_successor(successors) # Pilih tetangga terbaik
            iteration_count += 1
            if neighbor.value > self.current_state.value:
                self.current_state = neighbor # Update current state
                print(f'current state value: {self.current_state.value}')
                values.append(self.current_state.value)
            else:
                break
        print('terminate')
        print(f'final state value: {self.current_state.value}')
        print(f'iteration count: {iteration_count}')
        return self.current_state, values
```

Class SteepestAscentHC

Kelas yang menyimpan implementasi algoritma Steepest Ascent Hill Climbing

Attributes

- current_state
  Menyimpan instance kelas State sebagai representasi state terkini

Methods

- __init__
  Inisialisasi current state dengan sebuah state yang di-generate secara acak
- select_highest_value_successor
  Memilih suksesor dengan state value terbesar sebagai neighbor
- search
  Melakukan pencarian dengan algoritma Steepest Ascent Hill Climbing

b. Hill Climbing With Sideways Move

Implementasi algoritma Hill Climbing With Sideways Move dibuat sesuai dengan algoritma pada buku Russell. Neighbor didapatkan dari highest value successor. Kondisi terminasi terjadi ketika tidak ada lagi neighbor yang lebih baik/setara dengan current state atau sudah mencapai maksimal sideways move yang diperbolehkan.

Gambar Kode Class HCWithSidewaysMove

```python
from HillClimbing import HillClimbing

class HCWithSidewaysMove(HillClimbing):
    def __init__(self, max_sideways_move):
        super().__init__()
        self.max_sideways_move = max_sideways_move

    def select_highest_value_successor(self, successors):
        best_successor_index = 0
        best_value = successors[0].value
        for i in range(1, len(successors)):
            next_value = successors[i].value
            if (next_value > best_value):
                best_successor_index = i
                best_value = next_value
        neighbor = successors[best_successor_index]
        return neighbor

    def search(self):
        print(self.current_state)
        print(f'initial state value: {self.current_state.value}')
        values = [self.current_state.value]
        iteration_count = 0
        sideways_move = 0
        while not self.current_state.is_goal_state() and sideways_move < self.max_sideways_move:
            successors = self.generate_successors() # Hasilkan tetangga dari current state
            neighbor = self.select_highest_value_successor(successors) # Pilih tetangga terbaik
            iteration_count += 1
            if neighbor.value > self.current_state.value:
                self.current_state = neighbor # Update current state
                print(f'current state value: {self.current_state.value}')
                values.append(self.current_state.value)
            elif neighbor.value == self.current_state.value and sideways_move < self.max_sideways_move:
                self.current_state = neighbor # Update current state
                print(f'current state value: {self.current_state.value}')
                values.append(self.current_state.value)
                sideways_move += 1
                print(f'sideways move remaining: {self.max_sideways_move-sideways_move}')
            else:
                break  # Jika tidak ada perbaikan, hentikan
        print('terminate')
        print(f'final state value: {self.current_state.value}')
        print(f'iteration count: {iteration_count}')
        return self.current_state, values
```

Class HCWithSidewaysMove

Kelas yang menyimpan implementasi algoritma Hill Climbing With Sideways Move
Attributes

- current_state
  Menyimpan instance kelas State sebagai representasi state terkini
- max_sideways_move
  Menyimpan angka maksimal sideways move yang diperbolehkan

Methods

- __init__
  Inisialisasi current state dengan sebuah state yang di-generate secara acak
- select_highest_value_successor
  Memilih suksesor dengan state value terbesar sebagai neighbor
- search
  Melakukan pencarian dengan algoritma Hill Climbing With Sideways Move

c. Stochastic Hill Climbing
   Implementasi algoritma Stochastic Hill Climbing dibuat sesuai dengan algoritma pada buku Russell. Neighbor didapatkan dari random successor. Kondisi terminasi terjadi ketika mencapai maximum iteration.

Gambar Kode Class StochasticHC

```python
import random
from HillClimbing import HillClimbing

class StochasticHC(HillClimbing):
    def __init__(self, max_iteration):
        super().__init__()
        self.max_iteration = max_iteration

    def select_random_successor(self, successors):
        random_index = random.randrange(len(successors))
        random_successor = successors[random_index]
        return random_successor

    def search(self):
        print(self.current_state)
        print(f'initial state value: {self.current_state.value}')
        values = [self.current_state.value]
        for _ in range(self.max_iteration):
            successors = self.generate_successors() # Hasilkan tetangga dari current state
            neighbor = self.select_random_successor(successors) # Pilih tetangga terbaik
            if neighbor.value > self.current_state.value:
                self.current_state = neighbor # Update current state
            print(f'current state value: {self.current_state.value}')
            values.append(self.current_state.value)

        print('terminate')
        print(f'final state value: {self.current_state.value}')
        return self.current_state, values
```

Class StochasticHC

Kelas yang menyimpan implementasi algoritma Stochastic Hill Climbing

Attributes

- current_state
  Menyimpan instance kelas State sebagai representasi state terkini
- max_iteration
  Menyimpan angka maksimal iterasi yang diperbolehkan

Methods

- __init__
  Inisialisasi current state dengan sebuah state yang di-generate secara acak
- select_random_successor
  Memilih suksesor secara acak sebagai neighbor
- search
  Melakukan pencarian dengan algoritma Stochastic Hill Climbing

d. Random Restart Hill Climbing

Implementasi algoritma Random Restart Hill Climbing dibuat sesuai dengan algoritma pada buku Russell. Algoritma ini merupakan variasi dari Steepest Ascent Hill Climbing yang diulang dari awal ketika terjebak di local optimum. Kondisi terminasi terjadi ketika maximum restart tercapai atau goal state sudah ditemukan.

Gambar Kode Class RandomRestartHC

```python
from state import State
from HillClimbing import HillClimbing

class RandomRestartHC(HillClimbing):
    def __init__(self, max_restart):
        self.max_restart = max_restart

    def select_highest_value_successor(self, successors):
        best_successor_index = 0
        best_value = successors[0].value
        for i in range(1, len(successors)):
            next_value = successors[i].value
            if (next_value > best_value):
                best_successor_index = i
                best_value = next_value
        neighbor = successors[best_successor_index]
        return neighbor

    def search(self):
        values_per_restart = {}
        for restart in range(1, self.max_restart+1):
            self.current_state = State()
            print(self.current_state)
            print(f'initial state value: {self.current_state.value}')
            values = [self.current_state.value]
            iteration_count = 0
            while not self.current_state.is_goal_state():
                successors = self.generate_successors() # Hasilkan tetangga dari current state
                neighbor = self.select_highest_value_successor(successors) # Pilih tetangga terbaik
                iteration_count += 1
                if neighbor.value > self.current_state.value:
                    self.current_state = neighbor # Update current state
                    print(f'current state value: {self.current_state.value}')
                    values.append(self.current_state.value)
                else:
                    print(f'restart remaining: {self.max_restart-restart}')
                    break
            values_per_restart[restart] = values
            print(f'final state value: {self.current_state.value}')
            print(f'iteration in this restart: {iteration_count}')
            if self.current_state.is_goal_state():
                break
        print('terminate')
        return self.current_state, values_per_restart
```

Class RandomRestartHC

Kelas yang menyimpan implementasi algoritma Random Restart Hill Climbing

Attributes

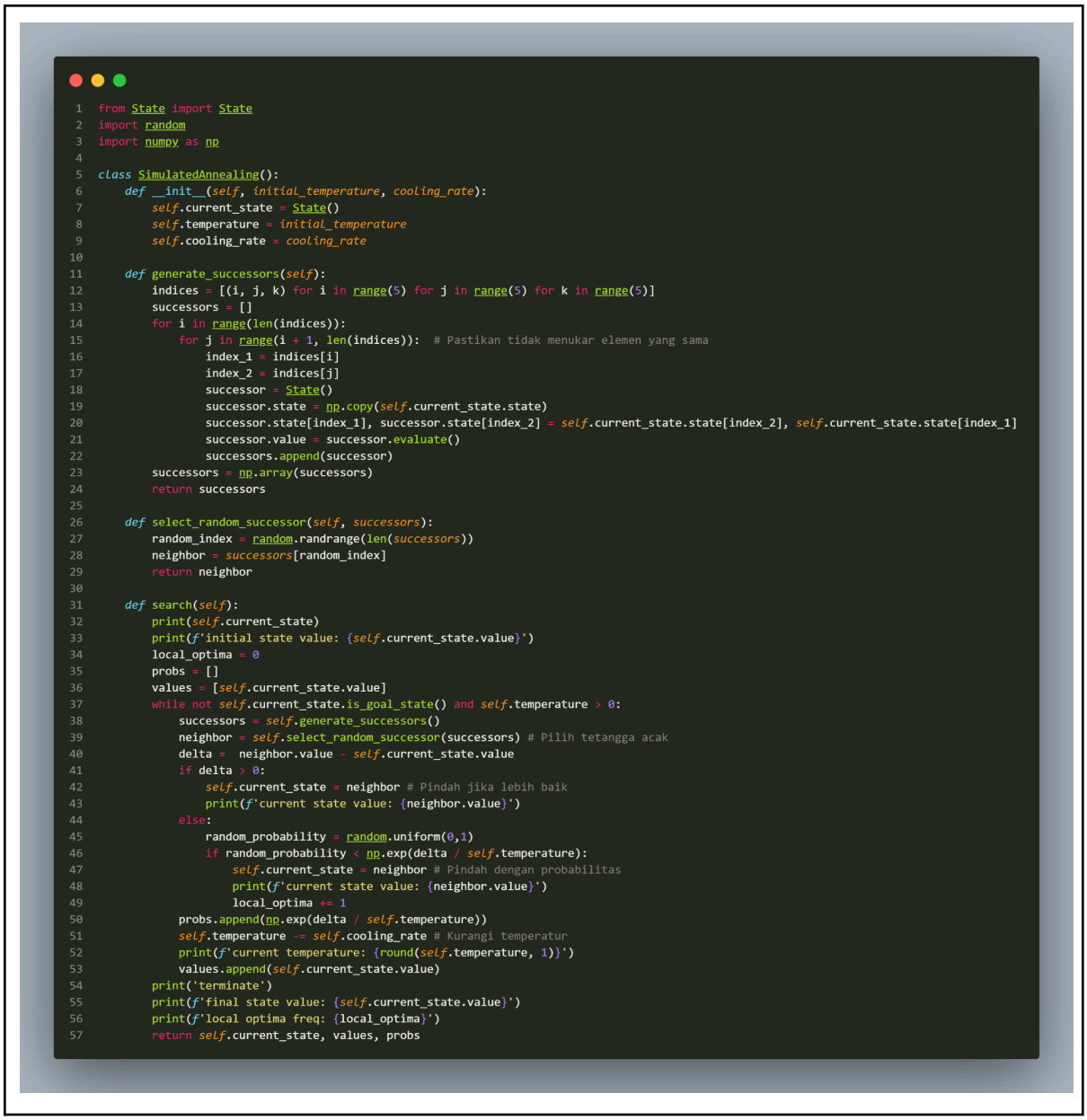
- current_state
  Menyimpan instance kelas State sebagai representasi state terkini
- max_restart
  Menyimpan angka maksimal restart yang diperbolehkan

Methods

- __init__
  Inisialisasi current state dengan sebuah state yang di-generate secara acak
- select_highest_value_successor
  Memilih suksesor dengan state value terbesar sebagai neighbor
- search
  Melakukan pencarian dengan algoritma Random Restart Hill Climbing

2. Simulated Annealing
   Simulated Annealing diimplementasikan dengan membuat sebuah class bernama SimulatedAnnealing. Implementasi dibuat sesuai dengan yang tertulis pada buku Russell. Terdapat sebuah temperature yang akan berkurang seiring waktu dengan faktor cooling rate. Neighbor didapatkan dari random successor. Apabila neighbor value tidak lebih baik daripada current state value, masih ada kesempatan untuk berpindah state dengan probabilitas $e^{\frac{\Delta E}{T}}$. Probabilitas ini dibandingkan dengan sebuah probabilitas lain yang dibangkitkan secara acak. Nilai temperatur yang semakin kecil juga mempersempit kesempatan untuk berpindah state. Kondisi terminasi terjadi ketika goal state sudah ditemukan atau temperature sama dengan nol.

Gambar Kode Class SimulatedAnnealing



```python
from State import State
import random
import numpy as np

class SimulatedAnnealing():
    def __init__(self, initial_temperature, cooling_rate):
        self.current_state = State()
        self.temperature = initial_temperature
        self.cooling_rate = cooling_rate

    def generate_successors(self):
        indices = [(i, j, k) for i in range(5) for j in range(5) for k in range(5)]
        successors = []
        for i in range(len(indices)):
            for j in range(i + 1, len(indices)):  # Pastikan tidak menukar elemen yang sama
                index_1 = indices[i]
                index_2 = indices[j]
                successor = State()
                successor.state = np.copy(self.current_state.state)
                successor.state[index_1], successor.state[index_2] = self.current_state.state[index_2], self.current_state.state[index_1]
                successor.value = successor.evaluate()
                successors.append(successor)
        successors = np.array(successors)
        return successors

    def select_random_successor(self, successors):
        random_index = random.randrange(len(successors))
        neighbor = successors[random_index]
        return neighbor

    def search(self):
        print(self.current_state)
        print(f'initial state value: {self.current_state.value}')
        local_optima = 0
        probs = []
        values = [self.current_state.value]
        while not self.current_state.is_goal_state() and self.temperature > 0:
            successors = self.generate_successors()
            neighbor = self.select_random_successor(successors) # Pilih tetangga acak
            delta =  neighbor.value - self.current_state.value
            if delta > 0:
                self.current_state = neighbor # Pindah jika lebih baik
                print(f'current state value: {neighbor.value}')
            else:
                random_probability = random.uniform(0,1)
                if random_probability < np.exp(delta / self.temperature):
                    self.current_state = neighbor # Pindah dengan probabilitas
                    print(f'current state value: {neighbor.value}')
                    local_optima += 1
            probs.append(np.exp(delta / self.temperature))
            self.temperature -= self.cooling_rate # Kurangi temperatur
            print(f'current temperature: {round(self.temperature, 1)}')
            values.append(self.current_state.value)
        print('terminate')
        print(f'final state value: {self.current_state.value}')
        print(f'local optima freq: {local_optima}')
        return self.current_state, values, probs
```

Class SimulatedAnnealing

Kelas yang menyimpan implementasi algoritma Simulated Annealing

Attributes

- current_state
  Menyimpan instance kelas State sebagai representasi state terkini
- temperature
  Menyimpan temperatur yang akan berkurang seiring waktu
- cooling_rate
  Menyimpan faktor pengurang bagi temperatur

Methods

- __init__

Inisialisasi current state dengan sebuah state yang di-generate secara acak, inisialisasi temperature dan cooling rate

- generate_successors
  Menghasilkan array of state yang merupakan kumpulan suksesor dari current state
- select_random_successor
  Memilih suksesor secara acak sebagai neighbor
- search
  Melakukan pencarian dengan algoritma Simulated Annealing

3. Genetic Algorithm
   Genetic Algorithm diimplementasikan dengan membuat sebuah class bernama GeneticAlgorithm. Implementasi dibuat sesuai dengan yang tertulis pada buku Russell. Pencarian diawali dengan pembangkitan sejumlah individu sebagai satu populasi. Kemudian, dilakukan iterasi untuk memperbarui populasi. Iterasi diawali dengan memilih dua buah parent secara random, tetapi dengan probabilitas yang tidak uniform. Individu yang memiliki nilai fitness function lebih besar memiliki probabilitas yang juga lebih besar untuk terpilih menjadi parent. Fitness function yang dipilih adalah jumlah feature yang memiliki jumlah elemen-elemen sama dengan 315. Selanjutnya, dilakukan single-point crossover terhadap dua buah parent tersebut. Crossover point didapatkan secara acak. Tahap ini menghasilkan sebuah child. Kemudian, dibangkitkan sebuah probabilitas acak. Apabila probabilitas ini lebih besar daripada mutation rate, maka dilakukan mutasi terhadap child dengan cara menukarkan dua buah gen yang dipilih secara acak. Penukaran gen dilakukan dengan alasan setiap elemen dalam kubus harus unik sehingga mutasi terhadap satu gen saja tidak dapat dilakukan. Child yang dihasilkan ini kemudian menjadi penghuni populasi baru. Tahapan ini dilakukan berulang hingga child memenuhi populasi baru. Dilakukan pemilihan individu terbaik pada setiap iterasi. Individu terbaik adalah individu yang memiliki nilai objective function terbesar. Objective function dalam hal ini sama dengan objective function pada algoritma local search sebelumnya. Kondisi terminasi terjadi ketika populasi saat ini mengandung individu yang merupakan goal state atau maximum iteration telah tercapai.

Gambar Kode Class GeneticAlgorithm

```python
import numpy as np
import random
from state import State

GENES = 125

class GeneticAlgorithm():
    def __init__(self, population_size, max_iteration, mutation_rate):
        self.population_size = population_size
        self.max_iteration = max_iteration
        self.mutation_rate = mutation_rate
        self.population = np.array([np.random.choice(np.arange(1, 126), size=125, replace=False) for _ in range(self.population_size)])
        self.current_state = State()

    def random_selection(self):
        fitnesses = np.array([self.fitness(individual) for individual in self.population])
        probabilities = fitnesses / fitnesses.sum()
        parent_index = np.random.choice(np.arange(self.population_size), p=probabilities)
        parent = self.population[parent_index]
        return parent

    # Crossover function (single-point crossover)
    def reproduce(self, parent1, parent2):
        point = random.randint(1, GENES-1)
        child = np.concatenate((parent1[:point], parent2[point:]))
        return child

    def mutate(self, individual):
        point1 = random.randrange(GENES)
        point2 = random.randrange(GENES)
        temp = individual[point1]
        individual[point1] = individual[point2]
        individual[point1] = temp
        return individual

    def get_feature_sum(self, individual):
        state = np.copy(individual).reshape(5, 5, 5)

        # pillars (25)
        pillar_sum = np.sum(state, axis=0).flatten()

        # rows (25)
        row_sum = np.sum(state, axis=2).flatten()

        # collumns (25)
        column_sum = np.sum(state, axis=1).flatten()

        # space diagonals (4)
        space_diagonals = []
        space_diagonals.append([state[i, i, i] for i in range(5)])
        space_diagonals.append([state[i, i, 4 - i] for i in range(5)])
        space_diagonals.append([state[i, 4 - i, i] for i in range(5)])
        space_diagonals.append([state[4 - i, i, i] for i in range(5)])
        space_diagonal_sum = np.sum(space_diagonals, axis=1)

        # diagonals (30)
        diagonals = []
        for n in range(5):
            # plane xy
            diagonals.append([state[i, i, n] for i in range(5)])
            diagonals.append([state[i, 4-i, n] for i in range(5)])
            # plane xz
            diagonals.append([state[i, n, i] for i in range(5)])
            diagonals.append([state[i, n, 4-i] for i in range(5)])
            # plane yz
            diagonals.append([state[n, i, i] for i in range(5)])
            diagonals.append([state[n, i, 4-i] for i in range(5)])
        diagonal_sum = np.sum(diagonals, axis=1)

        # collect all features
        feature_sum = np.concatenate((pillar_sum, row_sum, column_sum, space_diagonal_sum, diagonal_sum))

        return feature_sum

    def fitness(self, individual):
        feature_sum = self.get_feature_sum(individual)

        # calculate fitness
        value = 0
        for x in feature_sum:
            value -= abs(x-315)

        return 10000 + value

    def evaluate(self, individual):
        feature_sum = self.get_feature_sum(individual)

        # calculate state value
        value = 0
        for x in feature_sum:
            value += abs(x-315)

        return value

    def is_goal_state_in_population(self):
        values = np.array([self.evaluate(individual) for individual in self.population])
        return np.any(values == 0)

    def get_best_individual(self):
        values = np.array([self.evaluate(individual) for individual in self.population])
        best_index = np.argmax(values)
        best_individual = self.population[best_index]
        return best_individual

    def search(self):
        print(self.current_state)
        iteration = 1
        values = [self.evaluate(self.get_best_individual())]
        while not self.is_goal_state_in_population() and iteration <= self.max_iteration:
            print(f'iteration: {iteration}')
            offspring = []
            for _ in range(self.population_size):
                # Selection
                parent1 = self.random_selection()
                parent2 = self.random_selection()

                # Crossover / Reproduce
                child = self.reproduce(parent1, parent2) # Crossover antara dua parent

                # Mutation
                random_probability = random.random()
                if random_probability < self.mutation_rate:
                    child = self.mutate(child)

                offspring.append(child)
            self.population = offspring
            best_individual = self.get_best_individual()
            best_individual_value = self.evaluate(best_individual)
            print(f'best individual value: {best_individual_value}')
            values.append(best_individual_value)
            iteration += 1
        final_state = State(best_individual)
        return final_state, values
```

Class GeneticAlgorithm

Kelas yang menyimpan implementasi algoritma Genetic Algorithm

Attributes

- population_size
  Menyimpan angka yang merupakan ukuran populasi
- max_iteration
  Menyimpan angka maksimal iterasi yang diperbolehkan
- mutation_rate
  Menyimpan angka probabilitas diperbolehkannya mutasi
- population
  Menyimpan populasi yang merupakan kumpulan individu

Methods

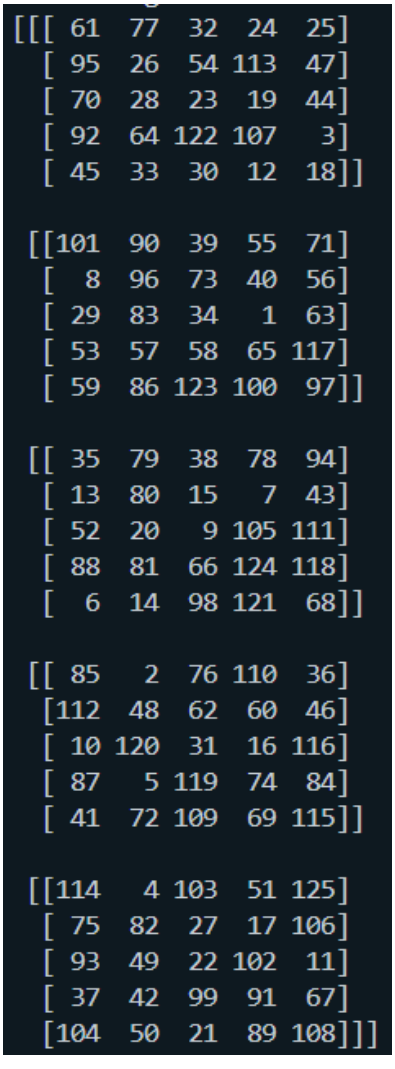
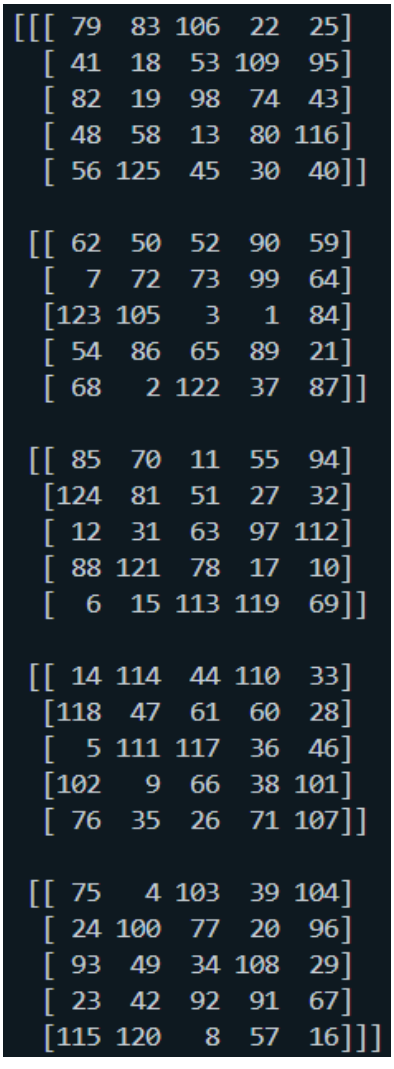
- __init__
  Inisialisasi population size, maximum iteration, mutation rate, dan population
- random_selection
  Memilih dua buah parent dari populasi terkini berdasarkan probabilitas yang didapatkan dari perhitungan fitness function
- reproduce
  Melakukan single-point crossover untuk menghasilkan sebuah child
- mutate
  Melakukan mutasi dengan cara menukarkan dua buah gen yang berada pada indeks yang ditentukan secara acak
- get_feature_sum
  Menghasilkan jumlah elemen-elemen pada feature
- fitness
  Menghitung nilai fitness function pada setiap individu
- evaluate
  Menghitung nilai objective function pada setiap individu
- is_goal_state_in_population
  Mengecek apakah populasi mengandung individu yang merupakan goal state
- get_best_individual
  Menghasilkan individu dengan nilai objective function terbesar
- search
  Melakukan pencarian dengan algoritma Genetic Algorithm

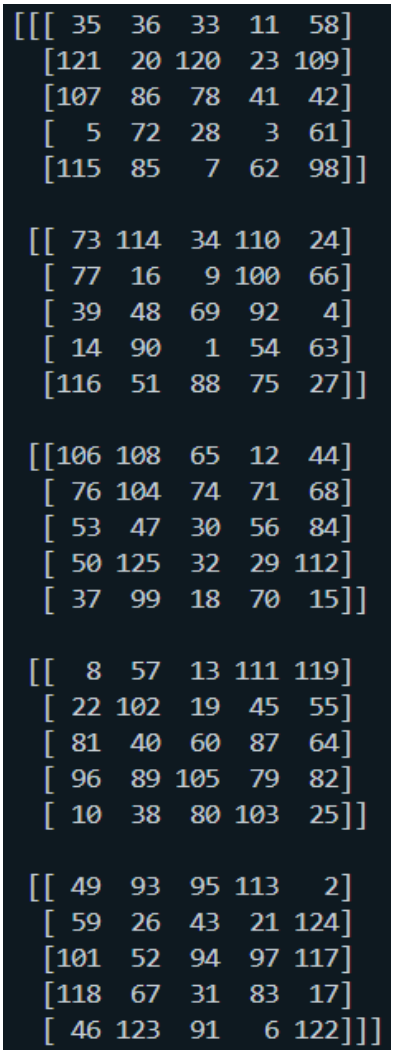
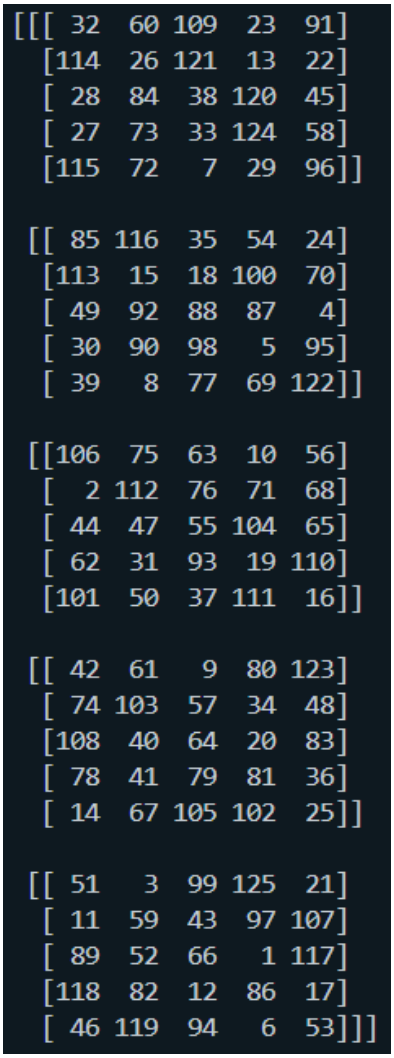
Hasil Eksperimen dan Analisis

Hasil Eksperimen

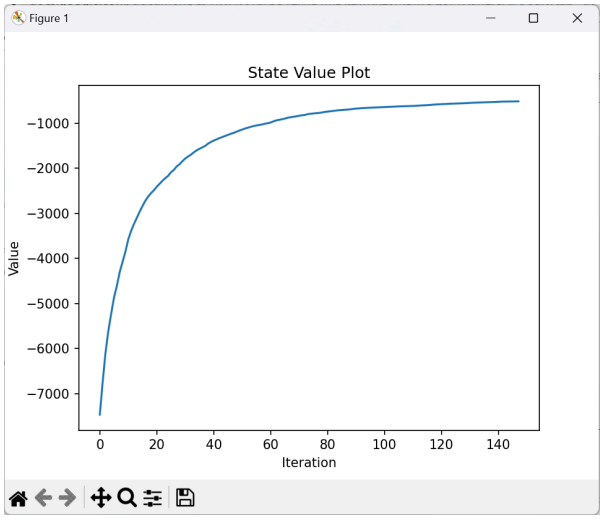
1. Steepest Ascent Hill Climbing

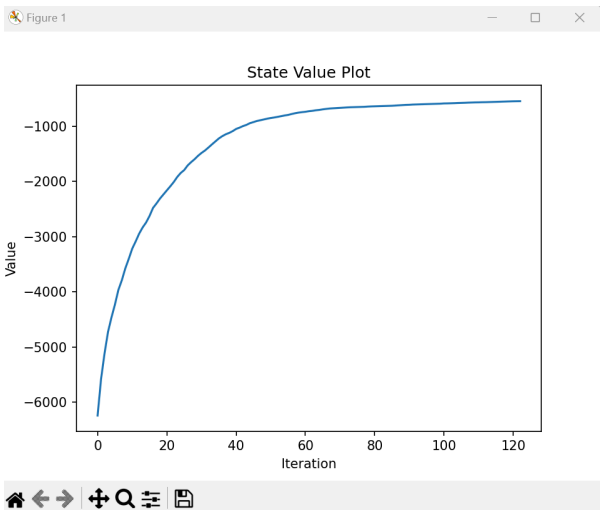
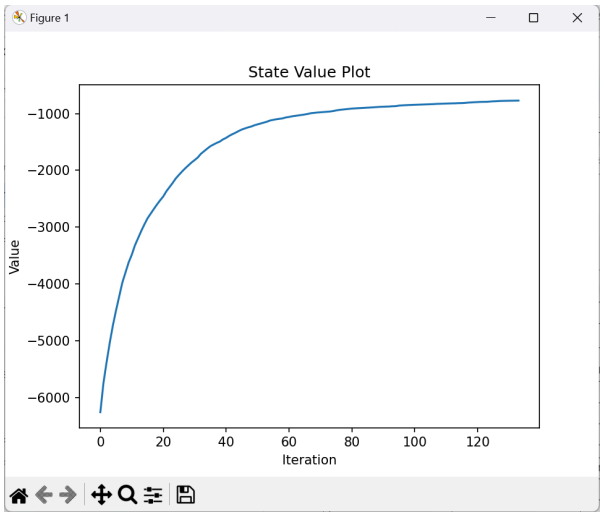
Percobaan	Durasi Pencarian	Banyak Iterasi	Nilai Objective Function Akhir
1	257.2 detik	148	-515
2	241.4 detik	123	-545
3	667.1 detik	134	-772

Percobaan	State Awal	State Akhir
1		

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

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

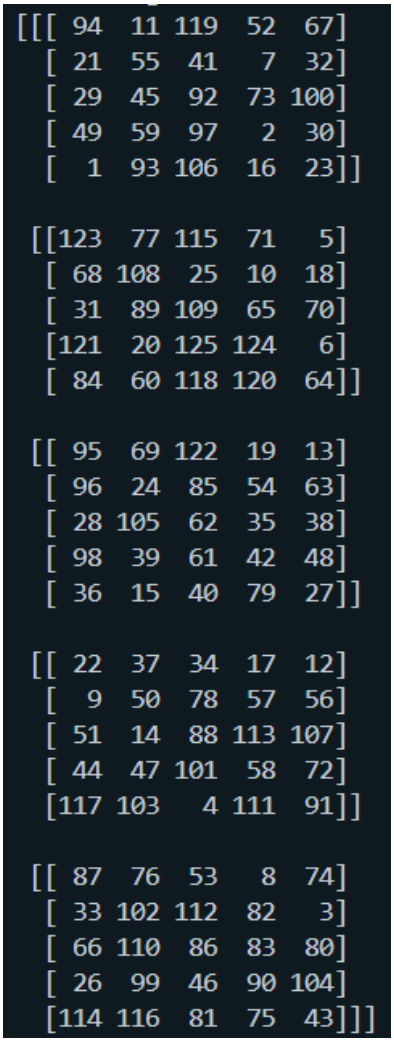
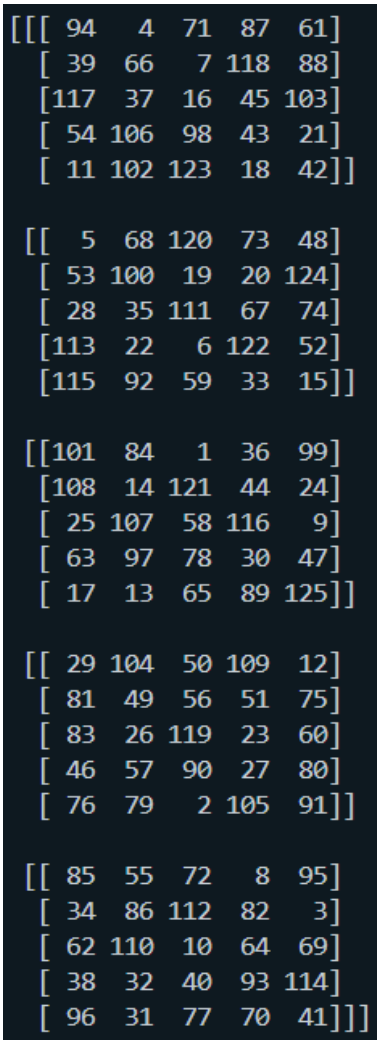
Plot Percobaan 1	

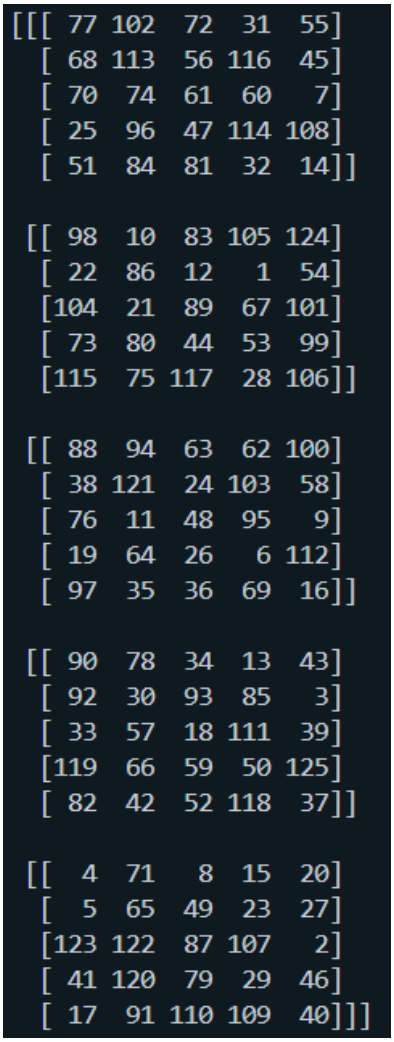
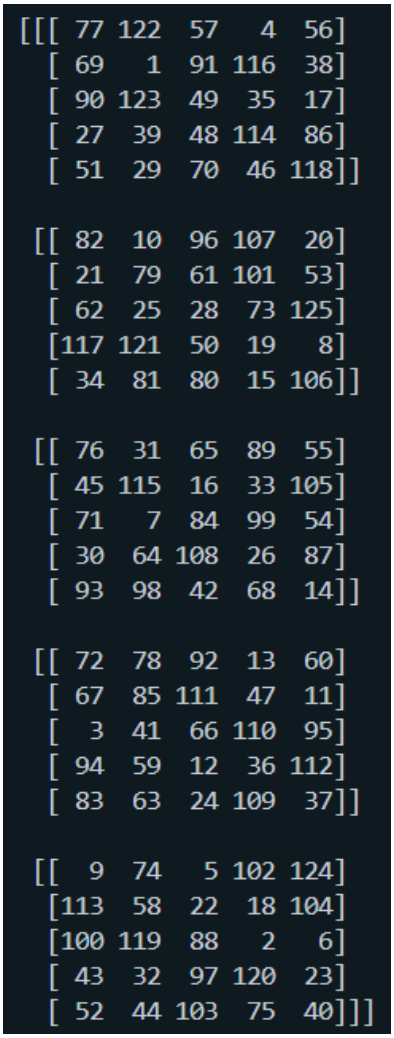
Plot Percobaan 2	
Plot Percobaan 3	

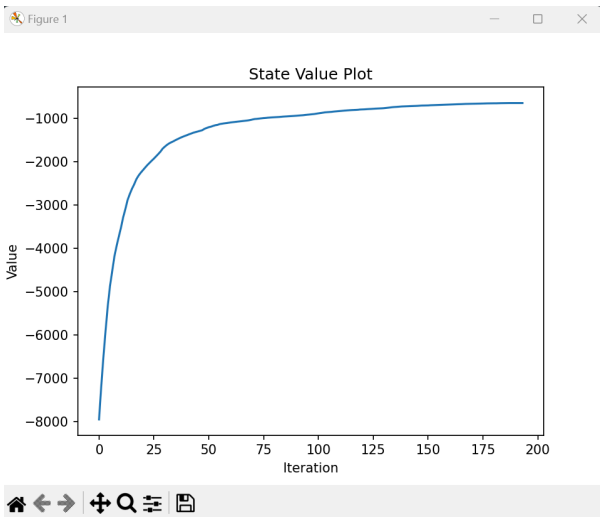
2. Hill Climbing With Sideways Move

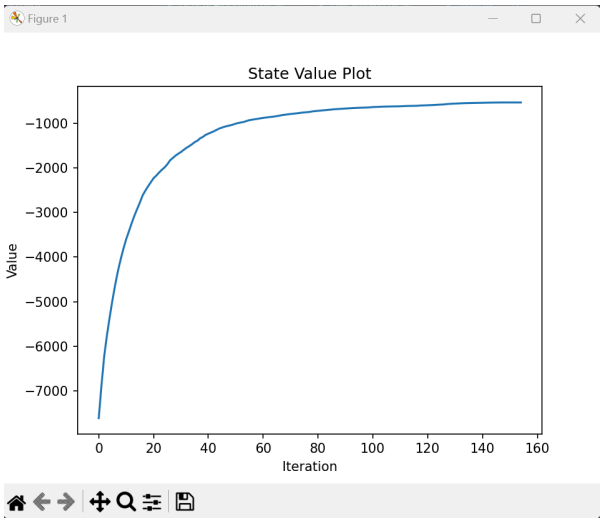
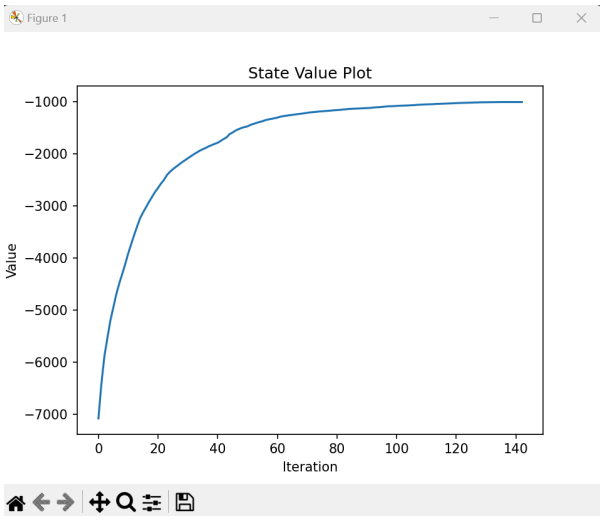
Percobaan	Durasi Pencarian	Banyak Iterasi	Maximum Sideways Move	Nilai Objective Function Akhir
1	916.637 detik	193	10	-643
2	295.098 detik	154	10	-529
3	272.028 detik	142	10	-1010

Percobaan	State Awal	State Akhir

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

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

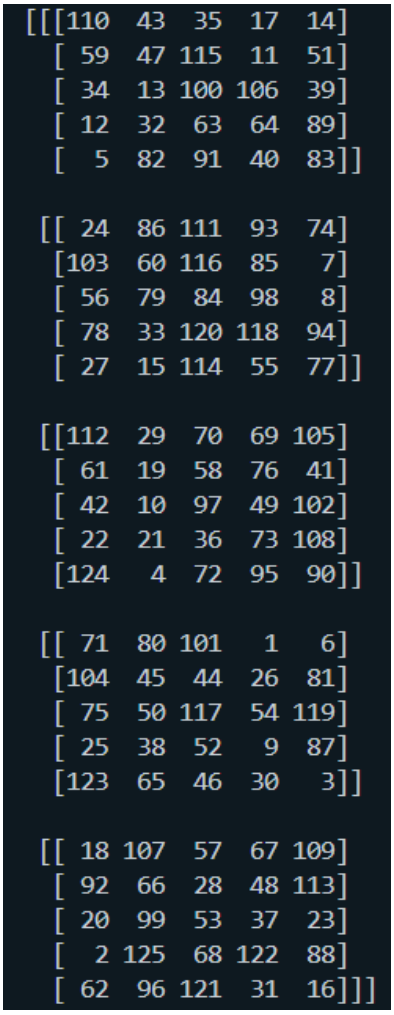
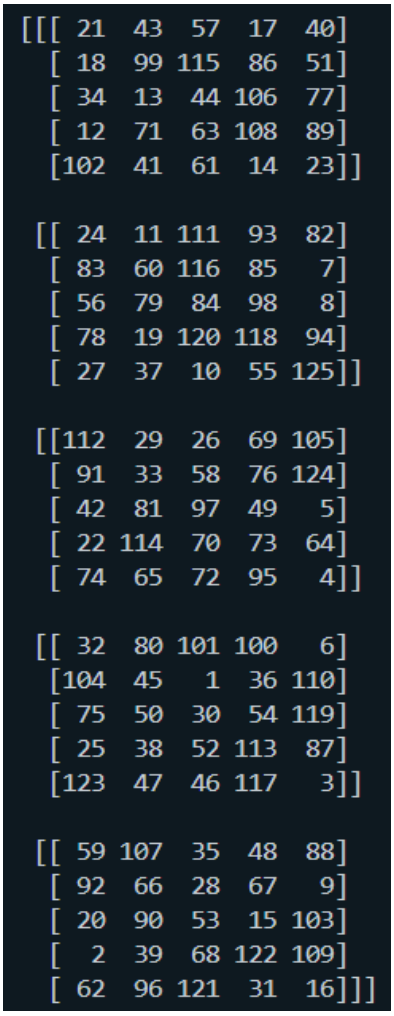
3	[[[122 39 24 19 99] [101 68 57 107 58] [106 92 87 76 105] [45 123 15 65 40] [96 75 71 9 114]] [[29 47 102 50 64] [67 118 103 111 12] [109 20 33 14 116] [11 61 90 46 72] [23 28 49 25 74]] [[66 79 91 80 104] [21 41 6 125 2] [119 120 60 51 86] [124 5 18 69 112] [113 13 26 44 88]] [[93 17 52 81 27] [98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

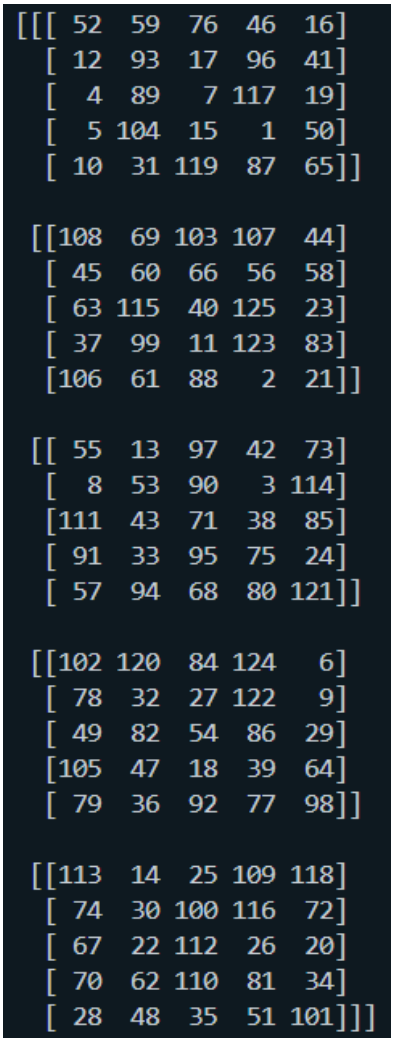
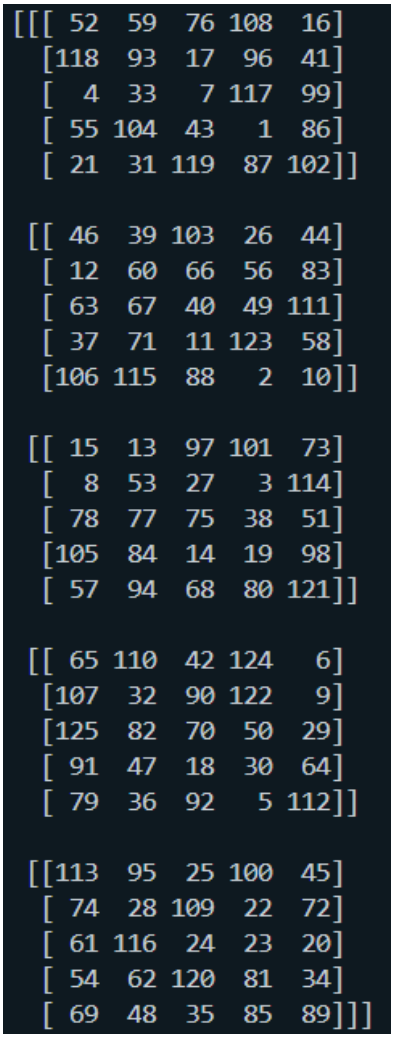
Plot Percobaan 2	
Plot Percobaan 3	

3. Stochastic Hill Climbing

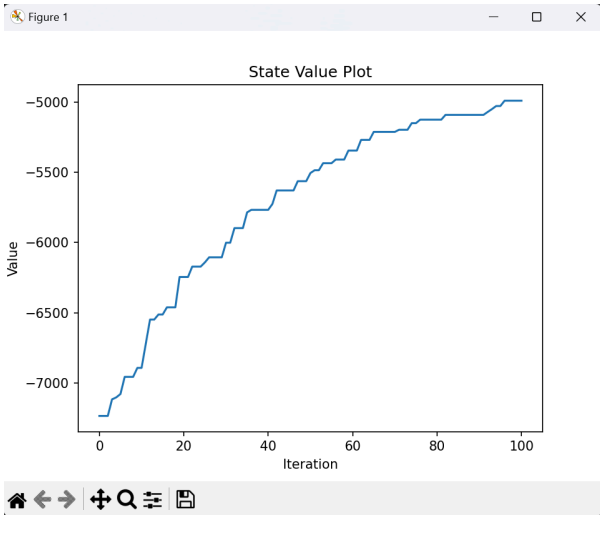
Percobaan	Durasi Pencarian	Banyak Iterasi	Nilai Objective Function Akhir
1	475.475 detik	100	-4990
2	490.573 detik	100	-4695
3	625.028 detik	100	-5059

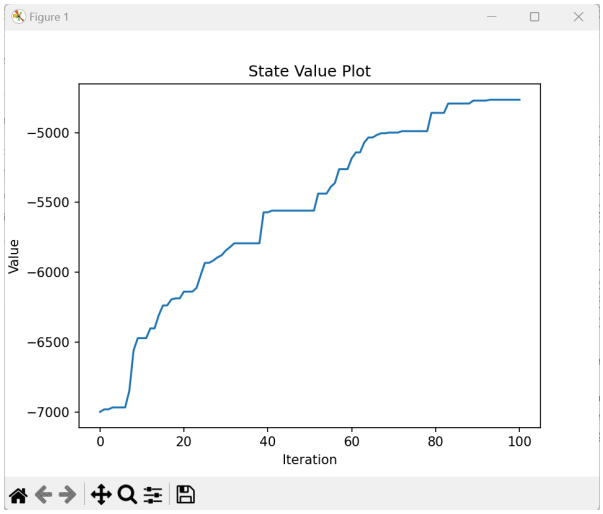
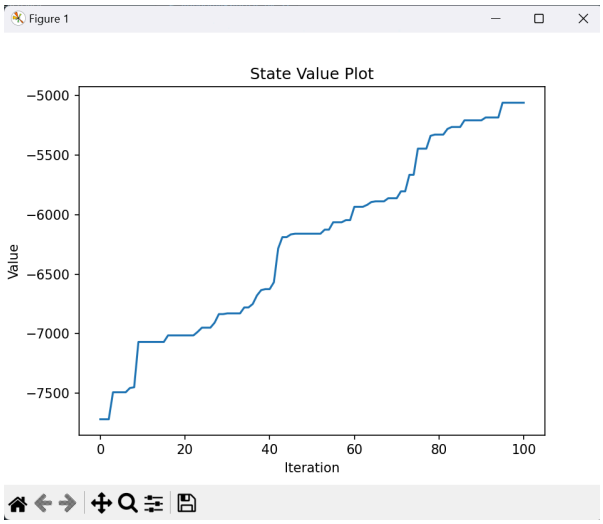
Percobaan	State Awal	State Akhir

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

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

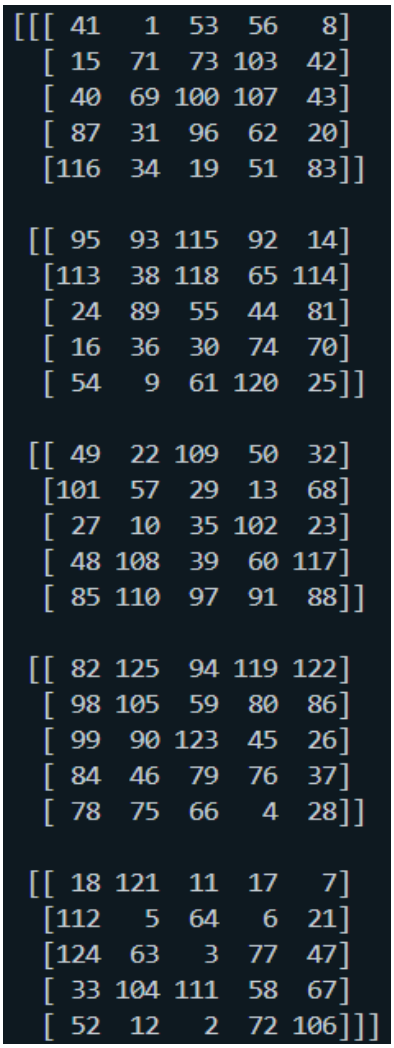
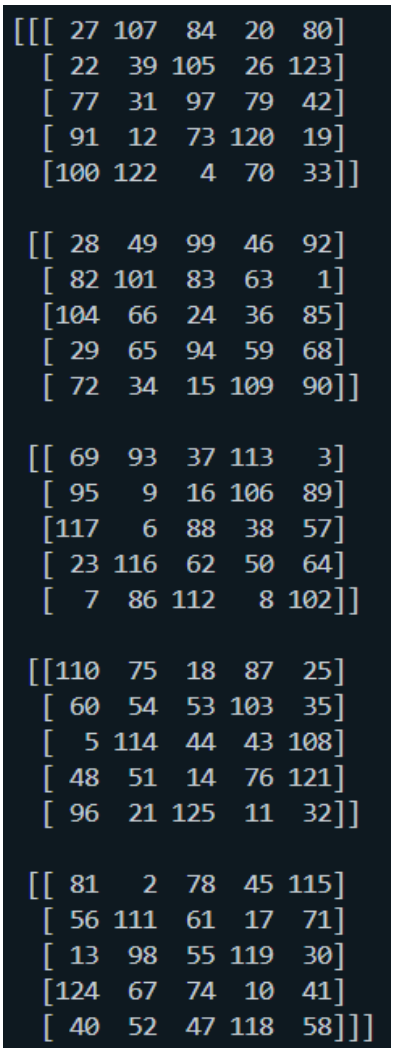
3	[[[114 121 123 105 72] [81 50 120 111 87] [106 26 100 66 40] [47 62 97 116 27] [109 10 122 118 58]] [[104 39 110 43 67] [93 23 35 74 86] [102 11 83 33 17] [71 80 32 22 36] [13 2 108 21 64]] [[37 16 84 56 14] [101 88 89 57 45] [18 6 95 24 9] [31 19 8 55 91] [59 82 15 7 76]] [[46 12 60 98 96] [49 85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

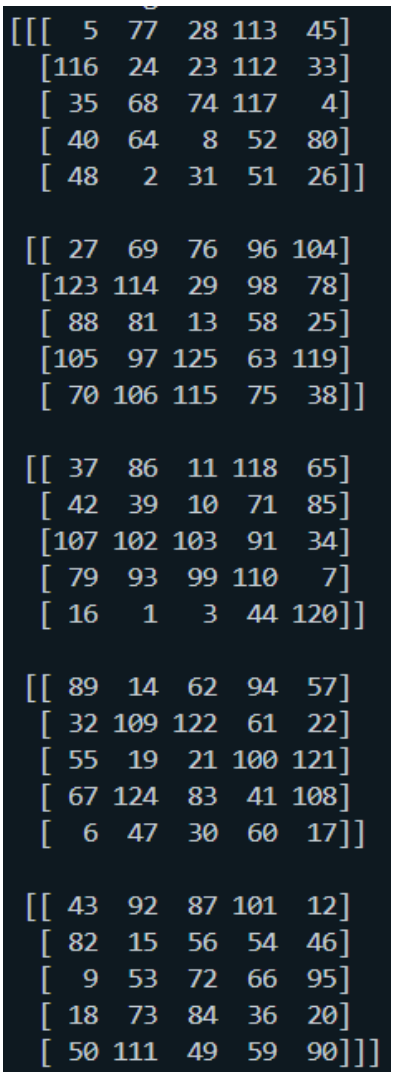
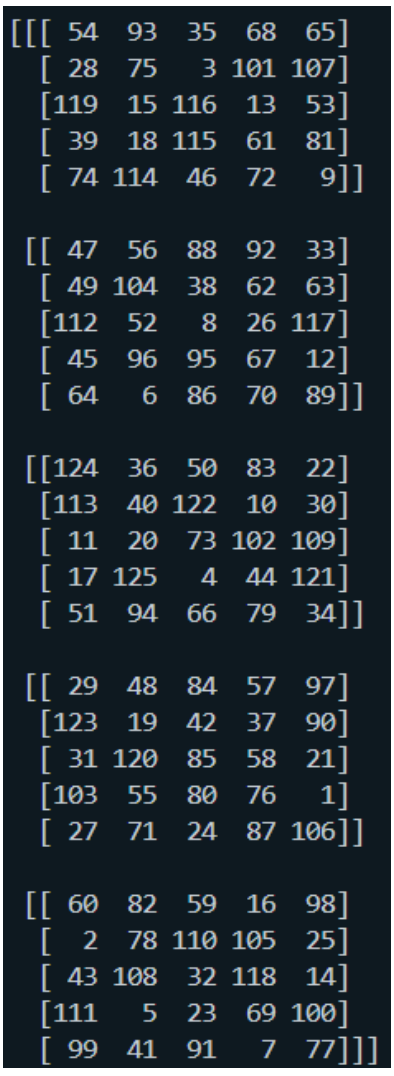
Plot Percobaan 1	

Plot Percobaan 2	
Plot Percobaan 3	

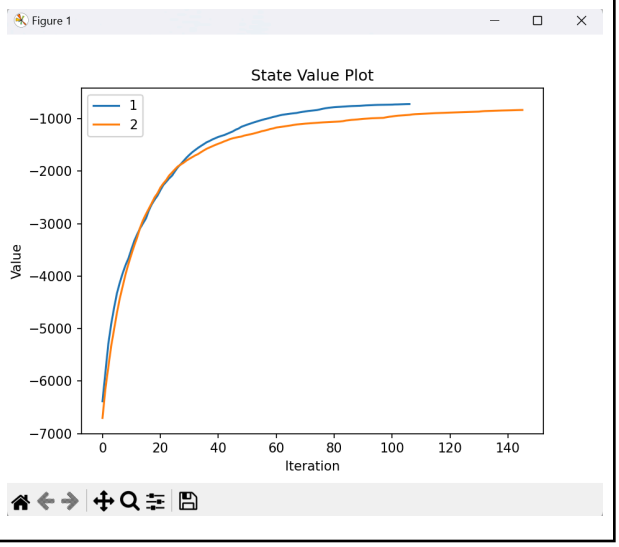
4. Random Restart Hill Climbing

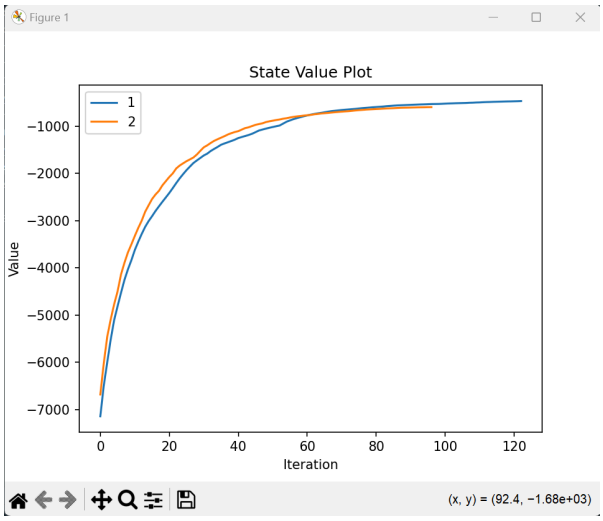
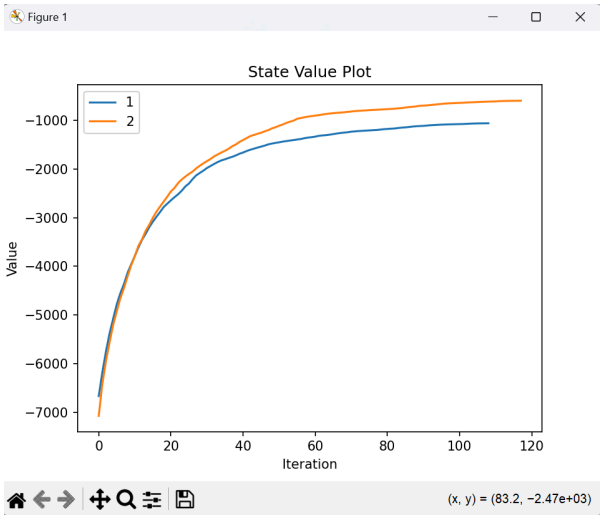
Percobaan	Durasi Pencarian	Restart	Banyak Iterasi	Nilai Objective Function Akhir
1	1169.237 detik	1	174	-723
		2	157	-836
2	694.786 detik	1	123	-468
		2	97	-596
3	487.886 detik	1	109	-1062
		2	118	-598

Percobaan	State Awal	State Akhir
1		

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

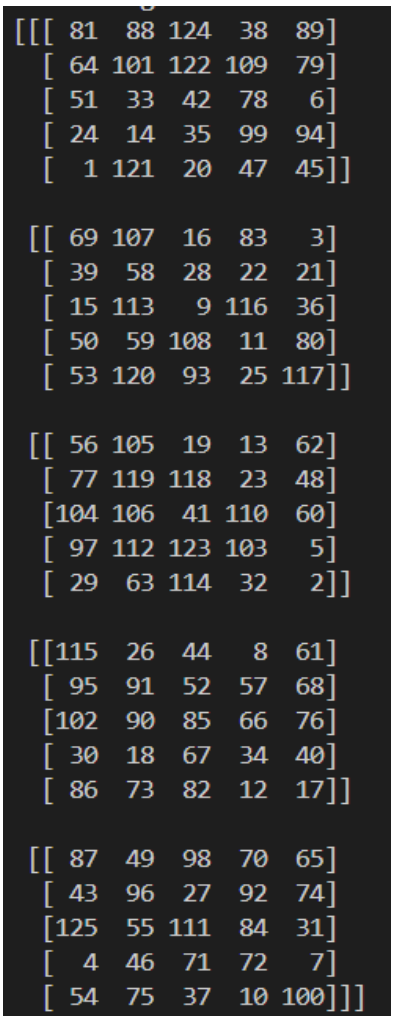
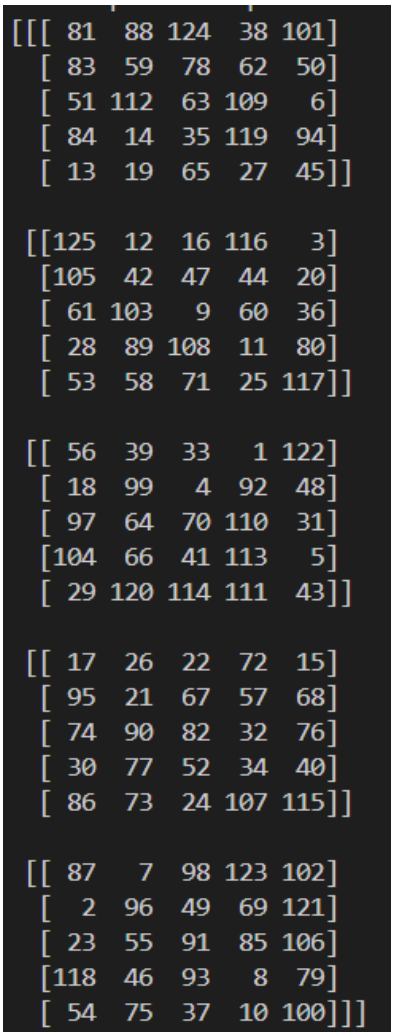
3	[[[[112 67 100 62 22]
	  [125 6 45 14 17]
	  [63 4 111 53 30]
	  [66 120 75 29 21]
	  [70 7 118 58 83]]
	
	  [[12 86 25 11 115]
	  [92 37 19 104 101]
	  [96 103 9 114 2]
	  [90 110 69 55 47]
	  [32 80 77 108 122]]
	
	  [[27 76 73 68 34]
	  [102 15 95 57 61]
	  [105 56 28 65 39]
	  [50 113 46 40 97]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

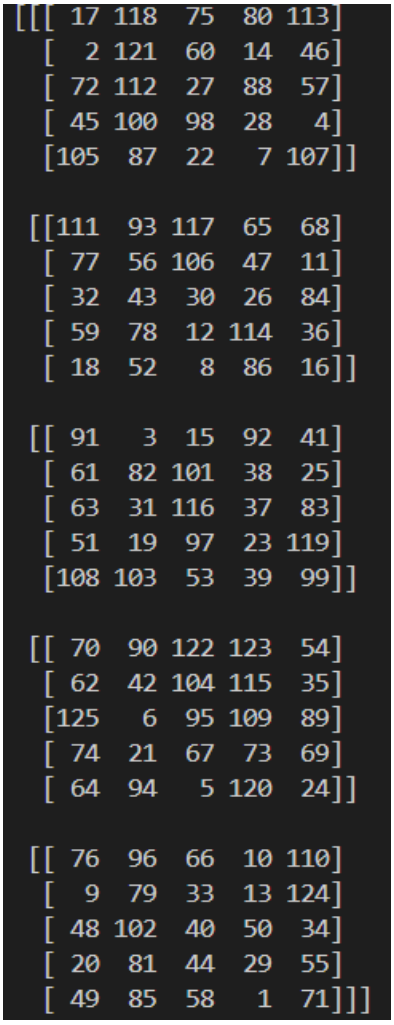
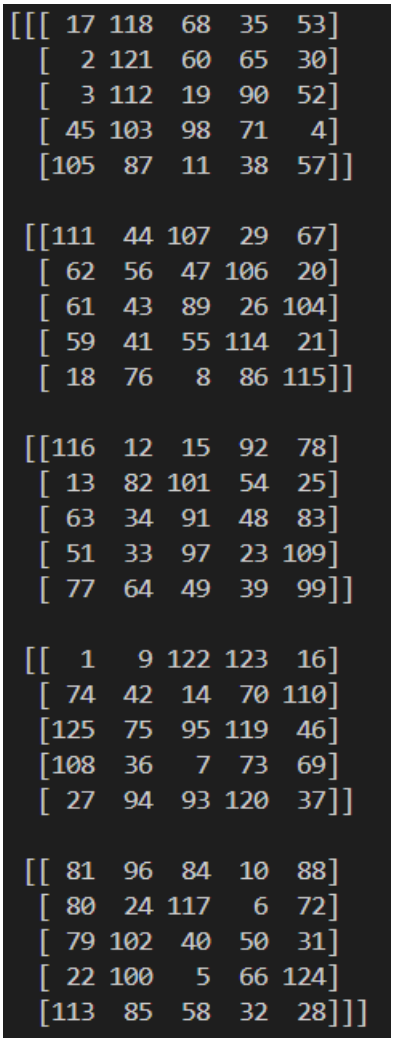
Plot Percobaan 1		

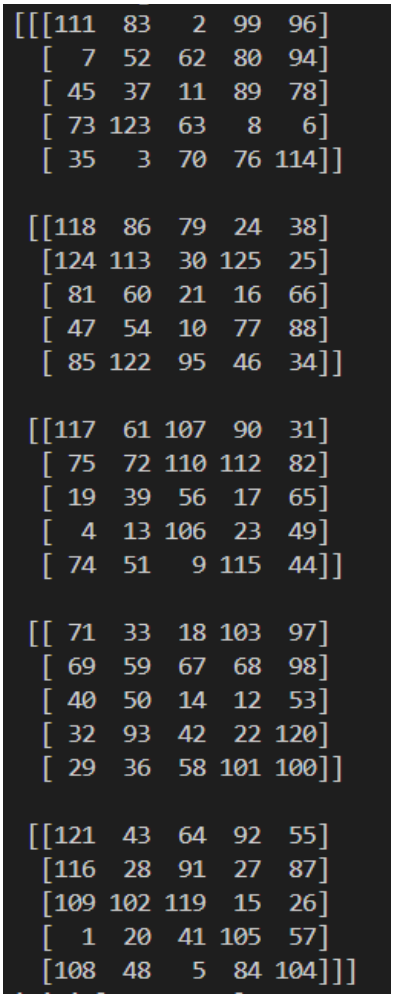
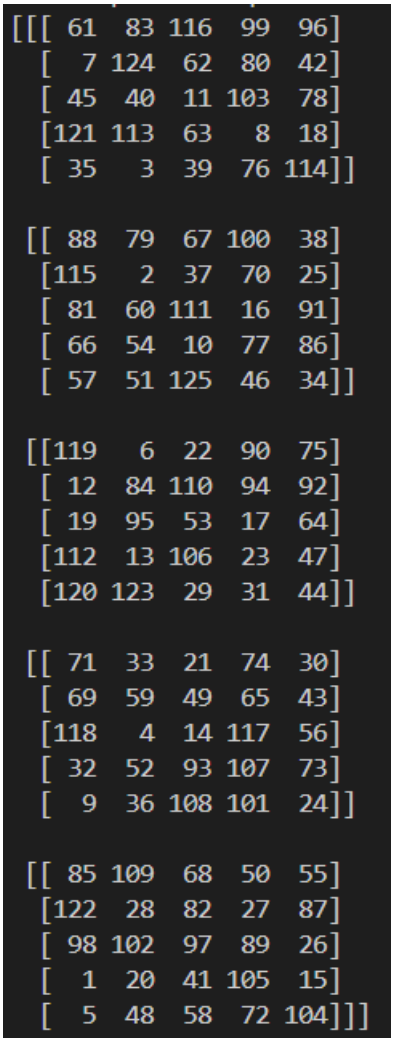
Plot Percobaan 2	
Plot Percobaan 3	

5. Simulated Annealing

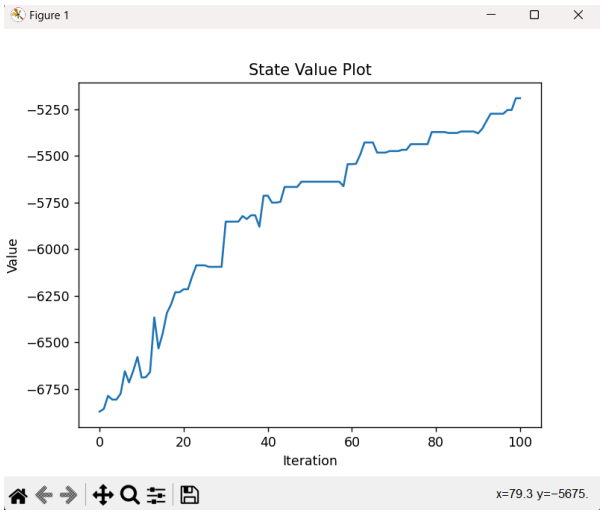
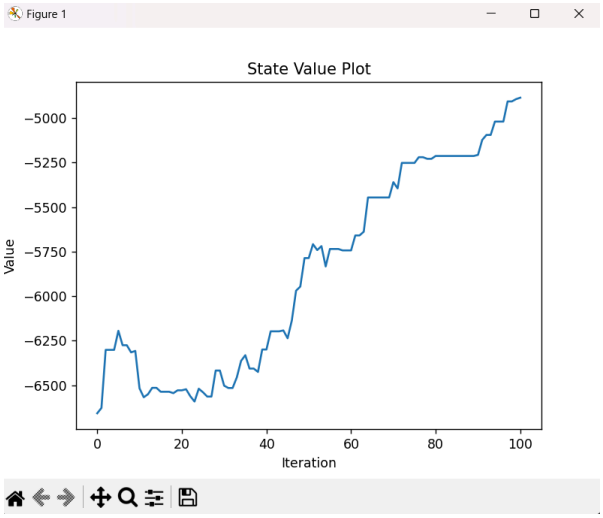
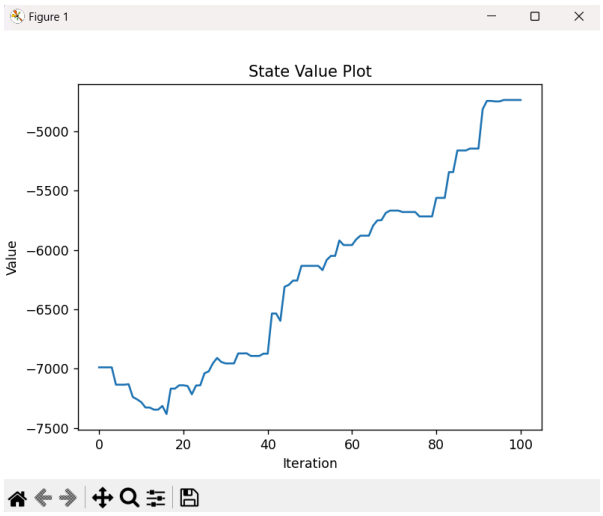
Percobaan	State Awal	State Akhir

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

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

3		

Percobaan	Durasi Pencarian	Banyak Iterasi	Frekuensi Stuck di Local Optima	Nilai Objective Function Akhir
1	229.996135473 25134 detik	100	17	-5192
2	197.135483741 76025 detik	100	24	-4886
3	213.706839084 62524 detik	100	24	-4738

Plot Percobaan 1	
Plot Percobaan 2	
Plot Percobaan 3	

6. Genetic Algorithm

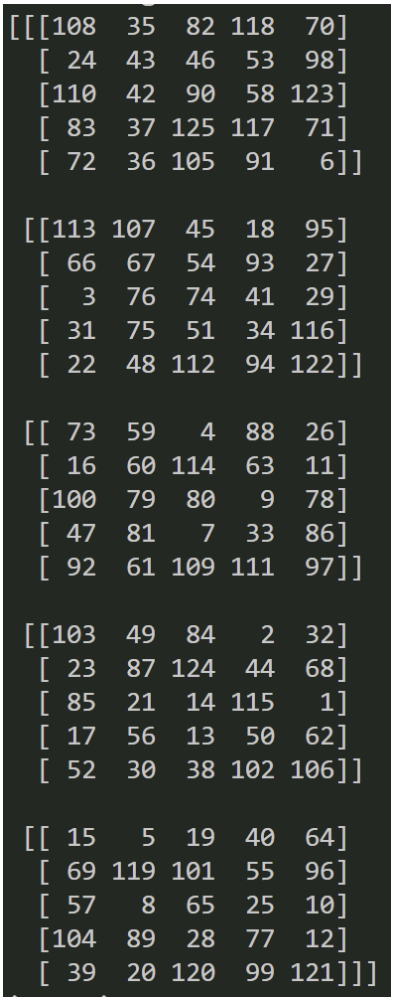
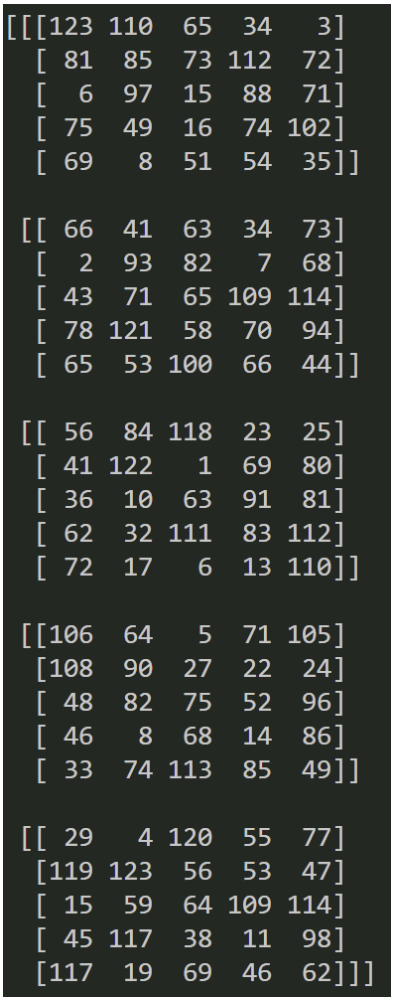
Jumlah Populasi	Banyak Iterasi	Percobaan	Durasi Pencarian	Nilai Objective Function Akhir
50	10	1	3.8598687648773193 detik	-5659
		2	3.8096349239349365 detik	-5598
		3	5.8885817527771 detik	-4958
	50	4	20.725285530090332 detik	-5507
		5	27.228473424 9115 detik	-4591
		6	21.109693527 22168 detik	-4730
	100	7	54.542582750320435 detik	-4465
		8	91.863746643 0664 detik	-5109
		9	82.016458034 51538 detik	-4680

Jumlah Populasi	Banyak Iterasi	Percobaan	Durasi Pencarian	Nilai Objective Function Akhir
10	50	10	3.2556943893432617 detik	-6336
		11	3.3057277202 6062 detik	-6128
		12	3.1898396015167236 detik	-5662
50		13	72.713326215 74402 detik	-5377

		14	73.1917963027 9541 detik	-4837
		15	73.9279170036 3159 detik	-4695
100		16	183.325494527 81677 detik	-4643
		17	184.330958604 81262 detik	-5031
		18	91.4061980247 4976 detik	-5101

Percobaan	Best Individual Awal	Best Individual Akhir
1	-5647	-5659
2	-6132	-5598
3	-5297	-4958
4	-5635	-5507
5	-6035	-4591
6	-5859	-4730
7	-5940	-4465
8	-6119	-51090
9	-5922	-4680
10	-6412	-6336
11	-6355	-6128
12	-5914	-5662
13	-5854	-5377
14	-5564	-4837
15	-5936	-4695

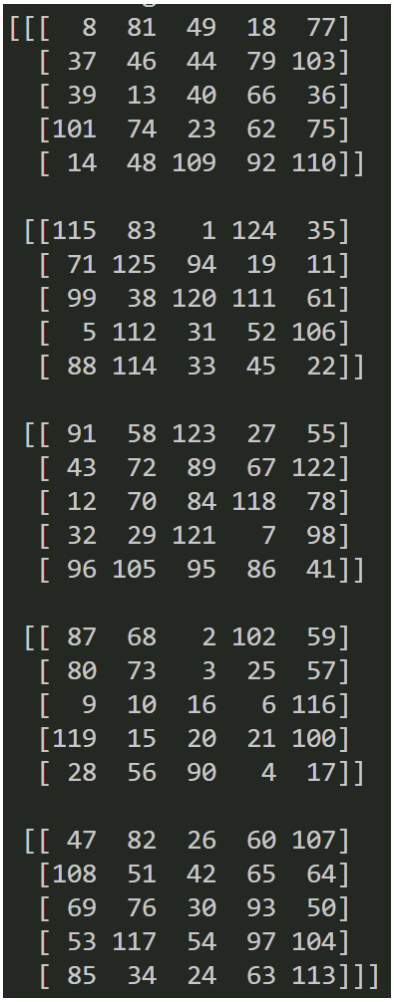
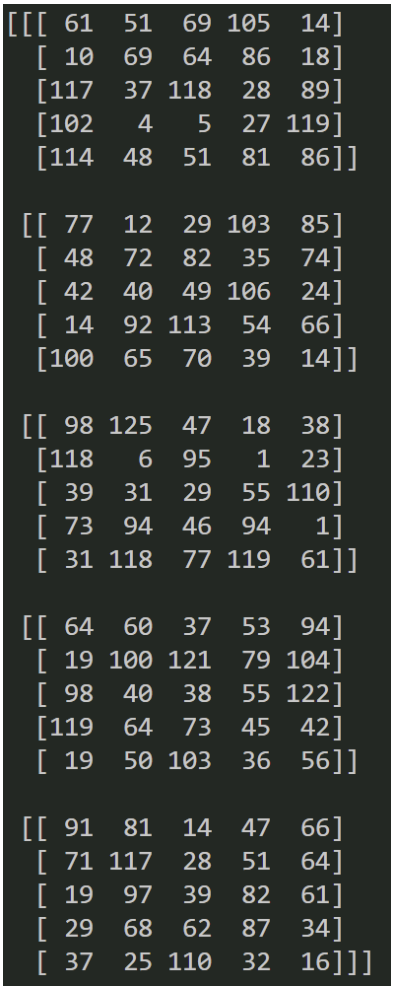
16	-5961	-4643
17	-6080	-5031
18	-5703	-5101

Percobaan	State Awal	State Akhir
1		

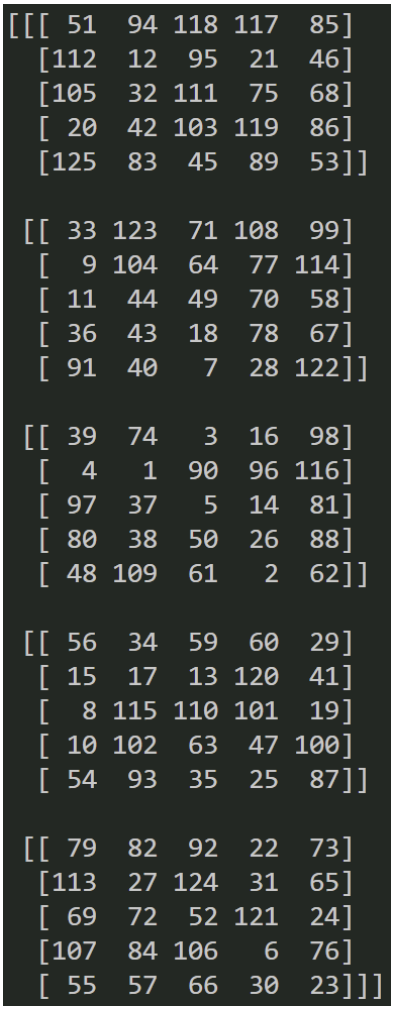
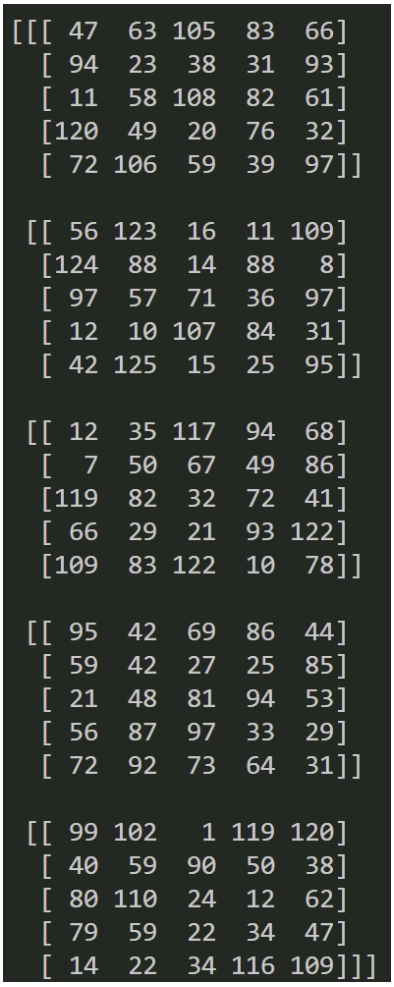
2	[[[68 121 23 116 108]
	  [111 113 89 3 106]
	  [97 43 19 7 53]
	  [18 36 67 2 91]
	  [83 79 59 51 24]]
	
	 [[31 65 44 93 86]
	  [124 105 72 12 38]
	  [48 75 71 56 100]
	  [90 77 13 112 104]
	  [20 81 88 37 95]]
	
	 [[60 14 45 61 118]
	  [41 11 117 29 25]
	  [9 57 114 76 15]
	  [63 102 69 26 73]
	 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

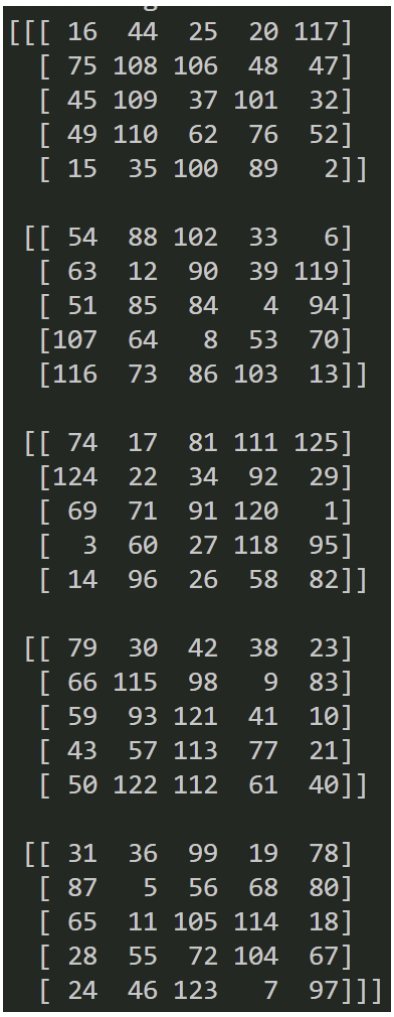
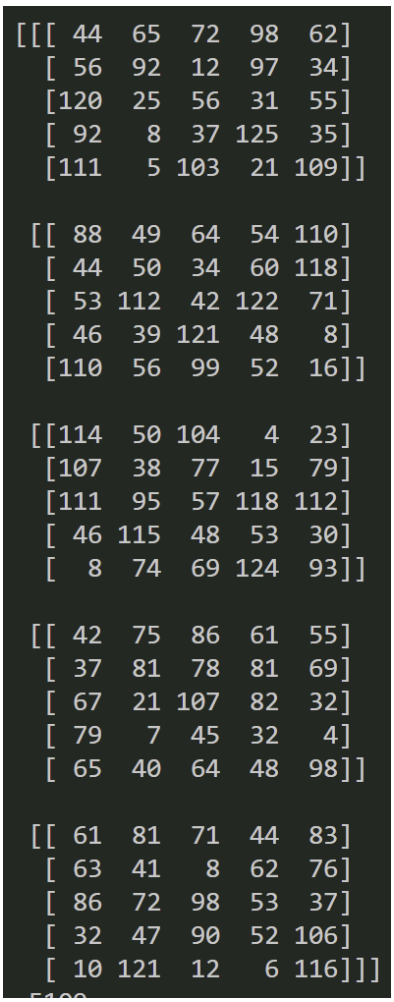
3	[[[27 114 40 53 91]
	  [7 52 48 41 101]
	  [117 104 66 24 68]
	  [33 122 12 47 32]
	  [69 70 57 20 84]]
	
	 [[50 99 72 115 110]
	  [80 42 76 2 21]
	  [89 11 112 22 73]
	  [35 95 77 16 116]
	  [10 119 65 5 26]]
	
	 [[3 29 36 121 98]
	  [4 37 61 90 60]
	  [86 87 88 45 100]
	  [74 83 6 8 97]
	  [28 123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

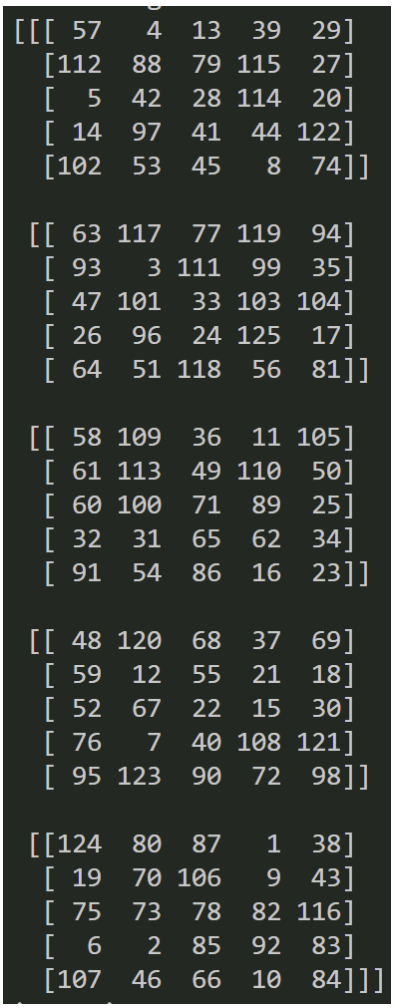
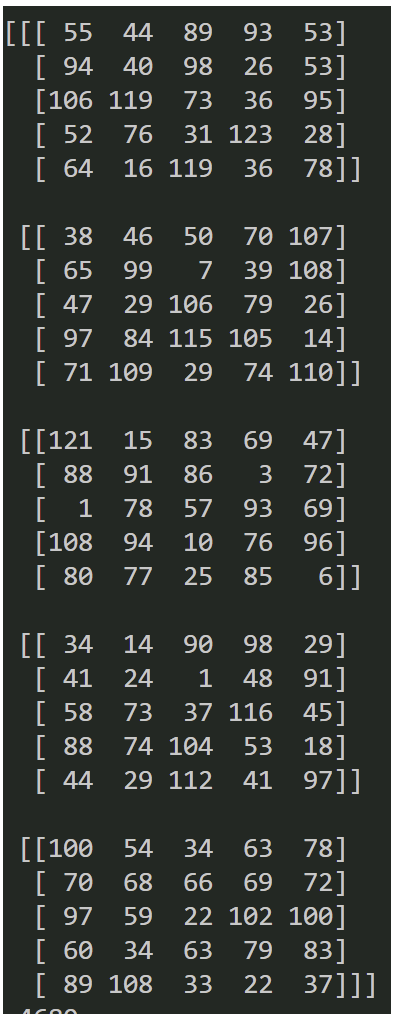
4	[[[51 100 49 81 75]
	  [53 18 23 59 118]
	  [56 16 32 91 4]
	  [115 103 63 34 109]
	  [85 62 101 122 97]]
	
	 [[86 77 55 68 105]
	  [98 39 108 35 76]
	  [78 2 17 25 48]
	  [31 94 71 57 120]
	  [54 95 28 72 80]]
	
	 [[93 36 43 26 107]
	  [1 37 124 27 69]
	  [41 87 9 65 84]
	  [11 52 121 29 123]
	  [114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

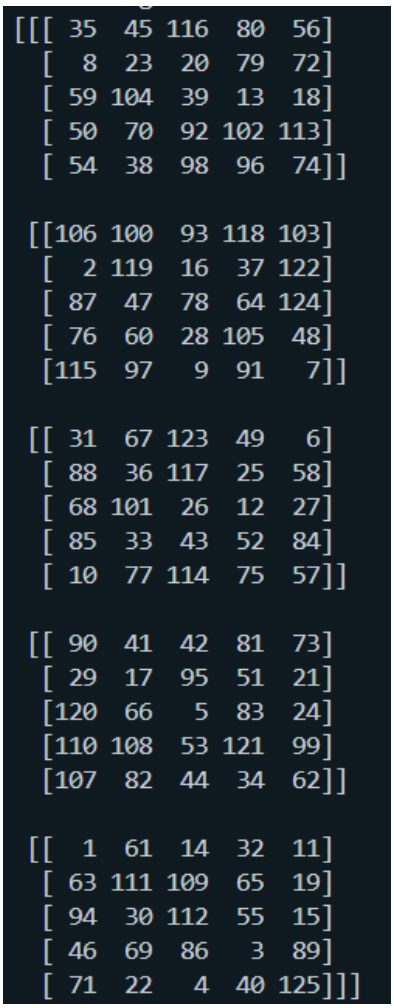
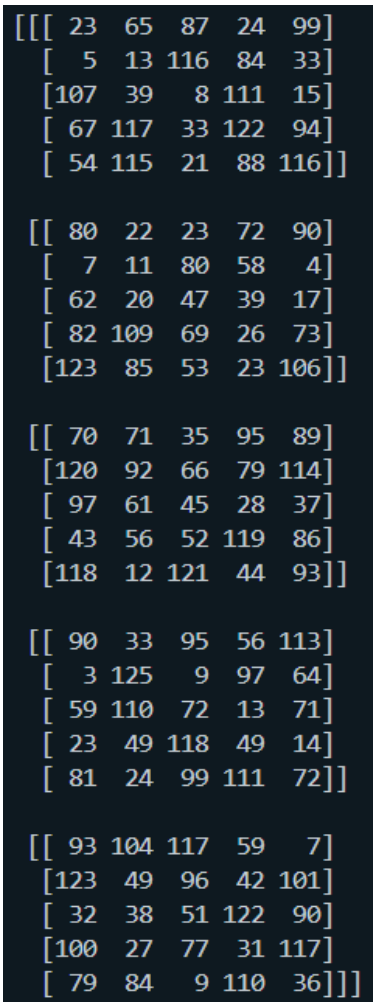
5	 [[[8 81 49 18 77] [37 46 44 79 103] [39 13 40 66 36] [101 74 23 62 75] [14 48 109 92 110]] [[115 83 1 124 35] [71 125 94 19 11] [99 38 120 111 61] [5 112 31 52 106] [88 114 33 45 22]] [[91 58 123 27 55] [43 72 89 67 122] [12 70 84 118 78] [32 29 121 7 98] [96 105 95 86 41]] [[87 68 2 102 59] [80 73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

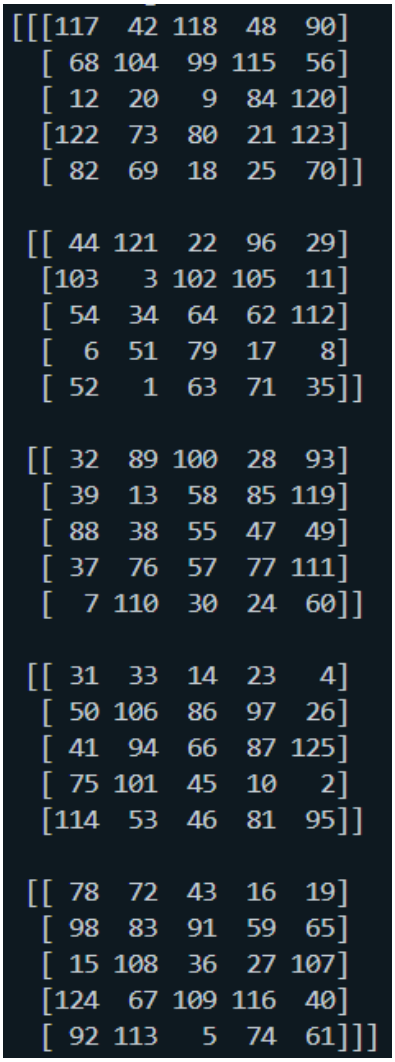
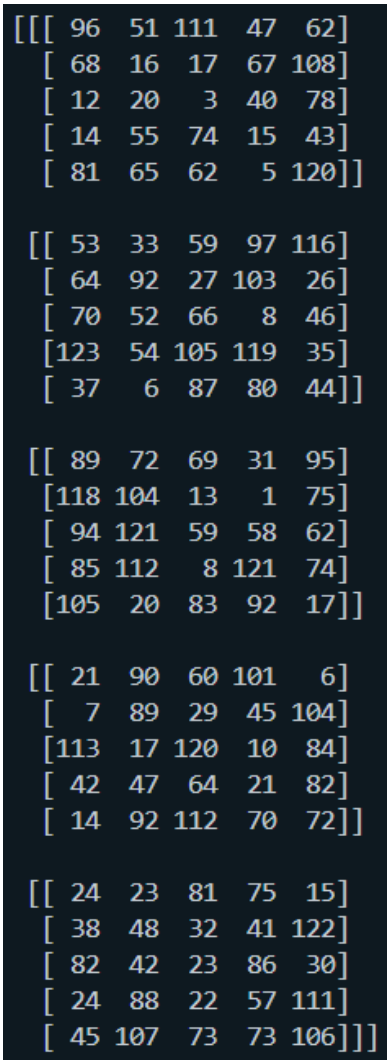
6	[[[88 123 114 34 56]
	  [113 16 84 116 105]
	  [103 59 120 100 122]
	  [107 8 44 19 48]
	  [99 68 118 35 101]]
	
	 [[46 117 39 11 55]
	  [98 43 108 109 62]
	  [57 70 14 9 95]
	  [89 49 63 60 115]
	  [24 29 47 1 12]]
	
	 [[2 50 94 92 73]
	  [64 83 106 37 124]
	  [78 22 28 20 4]
	  [112 32 71 6 97]
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

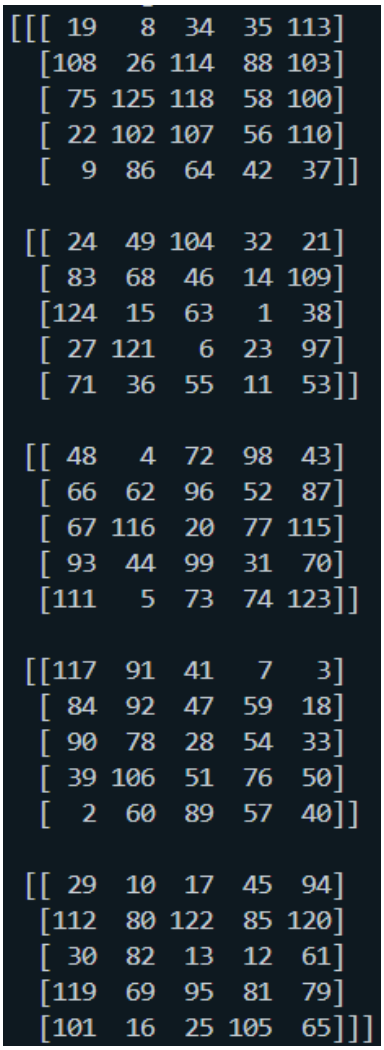
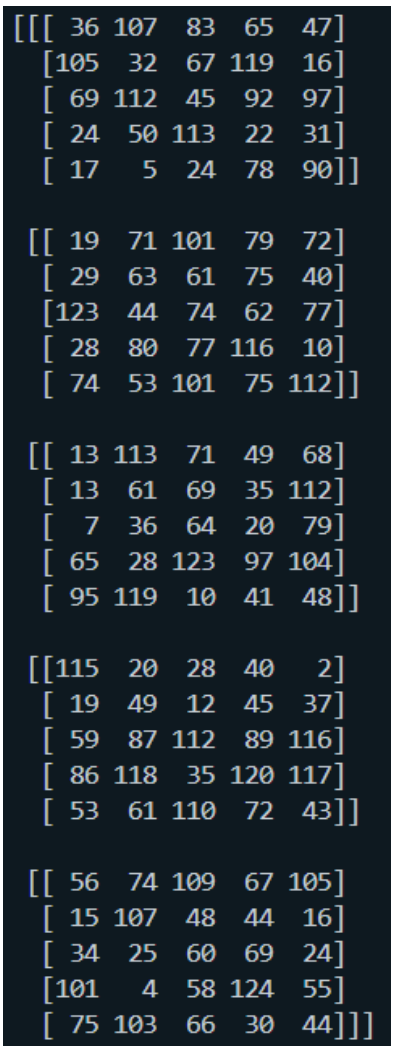
7	 [[[51 94 118 117 85] [112 12 95 21 46] [105 32 111 75 68] [20 42 103 119 86] [125 83 45 89 53]] [[33 123 71 108 99] [9 104 64 77 114] [11 44 49 70 58] [36 43 18 78 67] [91 40 7 28 122]] [[39 74 3 16 98] [4 1 90 96 116] [97 37 5 14 81] [80 38 50 26 88] [48 109 61 2 62]] [[56 34 59 60 29] [15 17 13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

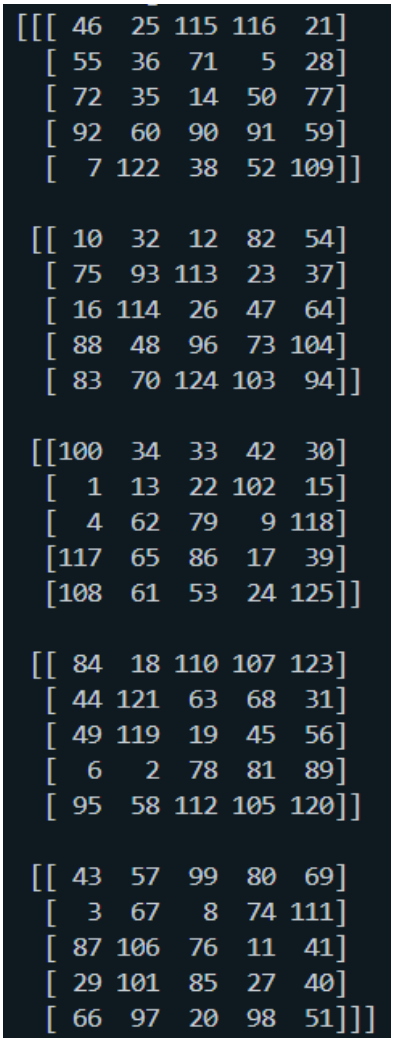
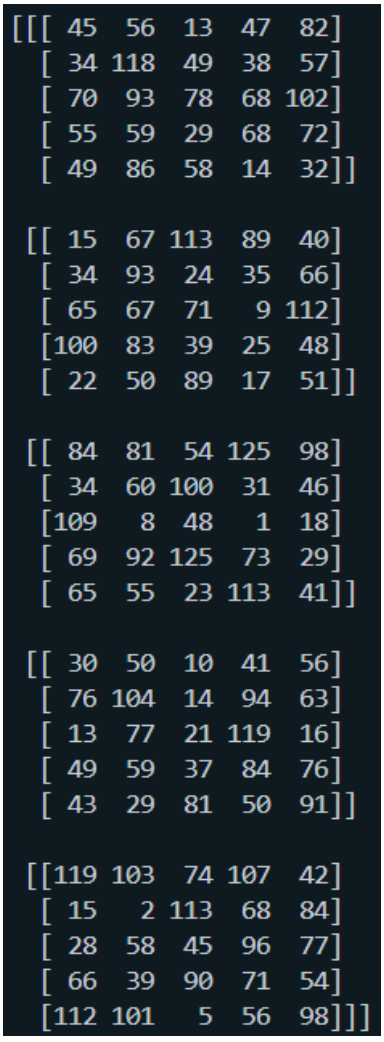
8	 [[[16 44 25 20 117] [75 108 106 48 47] [45 109 37 101 32] [49 110 62 76 52] [15 35 100 89 2]] [[54 88 102 33 6] [63 12 90 39 119] [51 85 84 4 94] [107 64 8 53 70] [116 73 86 103 13]] [[74 17 81 111 125] [124 22 34 92 29] [69 71 91 120 1] [3 60 27 118 95] [14 96 26 58 82]] [[79 30 42 38 23] [66 115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

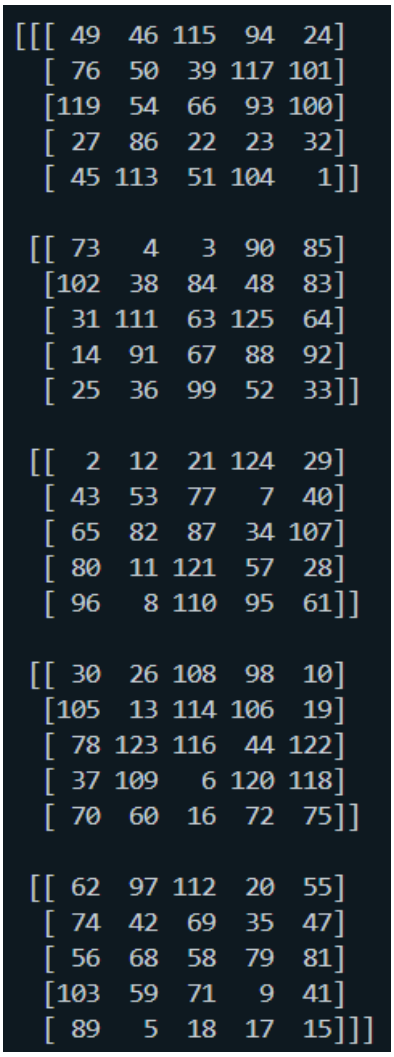
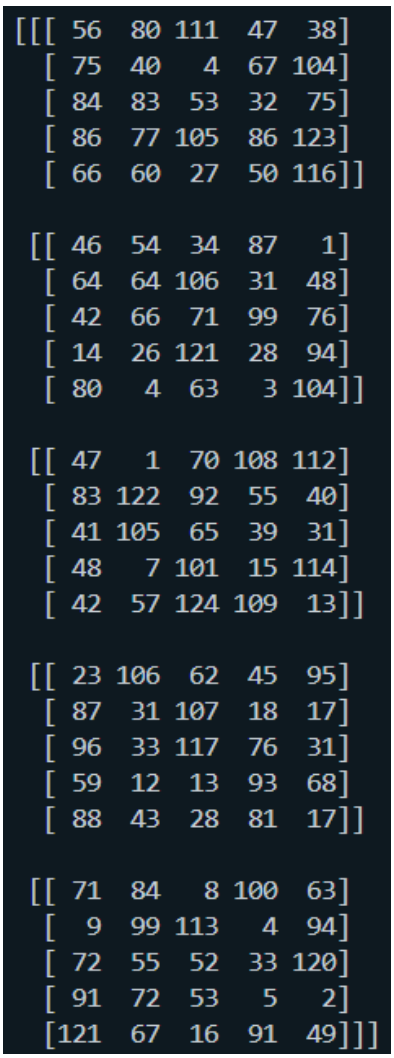
9	 [[[57 4 13 39 29] [112 88 79 115 27] [5 42 28 114 20] [14 97 41 44 122] [102 53 45 8 74]] [[63 117 77 119 94] [93 3 111 99 35] [47 101 33 103 104] [26 96 24 125 17] [64 51 118 56 81]] [[58 109 36 11 105] [61 113 49 110 50] [60 100 71 89 25] [32 31 65 62 34] [91 54 86 16 23]] [[48 120 68 37 69] [59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

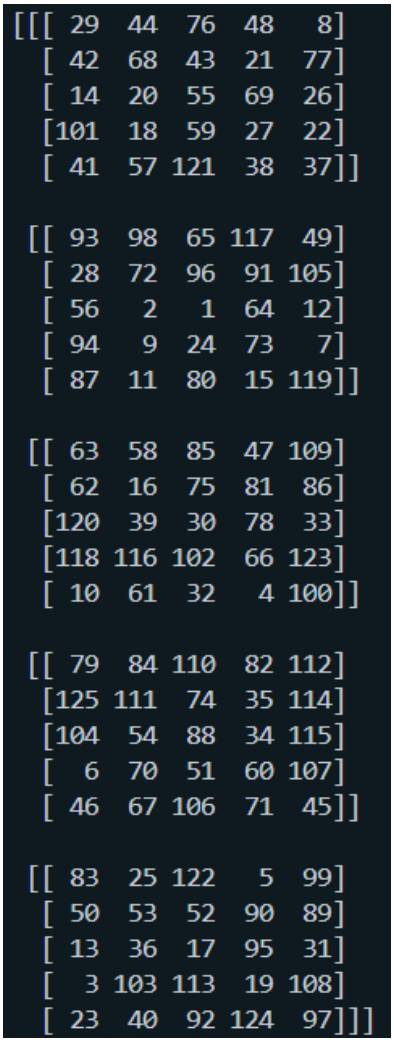
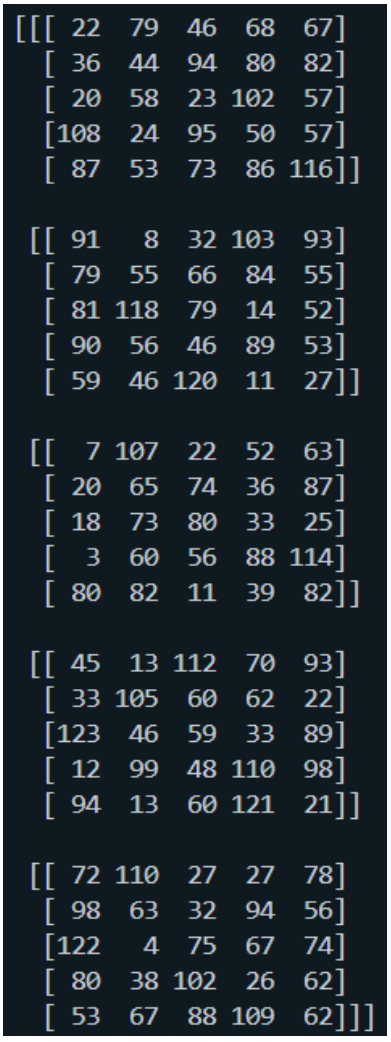
10	 [[[[35 45 116 80 56] [8 23 20 79 72] [59 104 39 13 18] [50 70 92 102 113] [54 38 98 96 74]] [[106 100 93 118 103] [2 119 16 37 122] [87 47 78 64 124] [76 60 28 105 48] [115 97 9 91 7]] [[31 67 123 49 6] [88 36 117 25 58] [68 101 26 12 27] [85 33 43 52 84] [10 77 114 75 57]] [[90 41 42 81 73] [29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

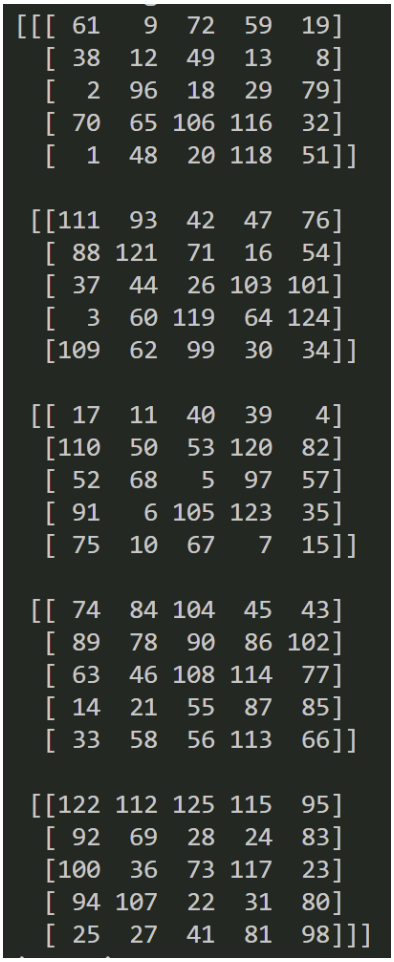
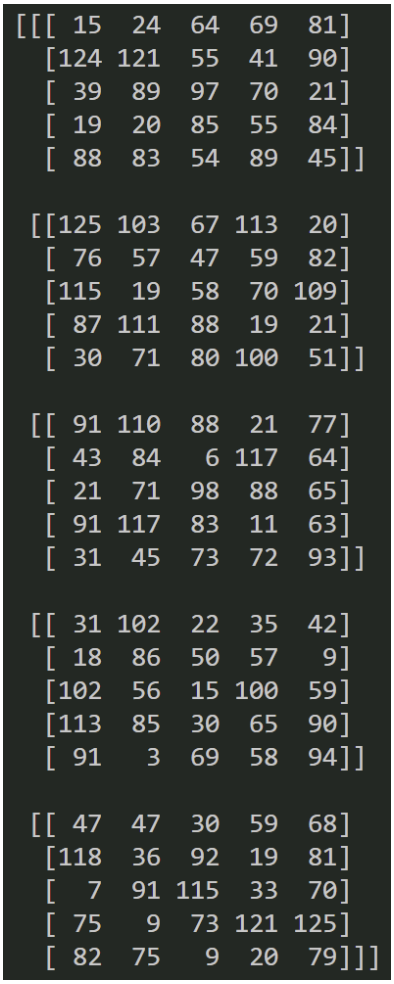
11	 [[[117 42 118 48 90] [68 104 99 115 56] [12 20 9 84 120] [122 73 80 21 123] [82 69 18 25 70]] [[44 121 22 96 29] [103 3 102 105 11] [54 34 64 62 112] [6 51 79 17 8] [52 1 63 71 35]] [[32 89 100 28 93] [39 13 58 85 119] [88 38 55 47 49] [37 76 57 77 111] [7 110 30 24 60]] [[31 33 14 23 4] [50 106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

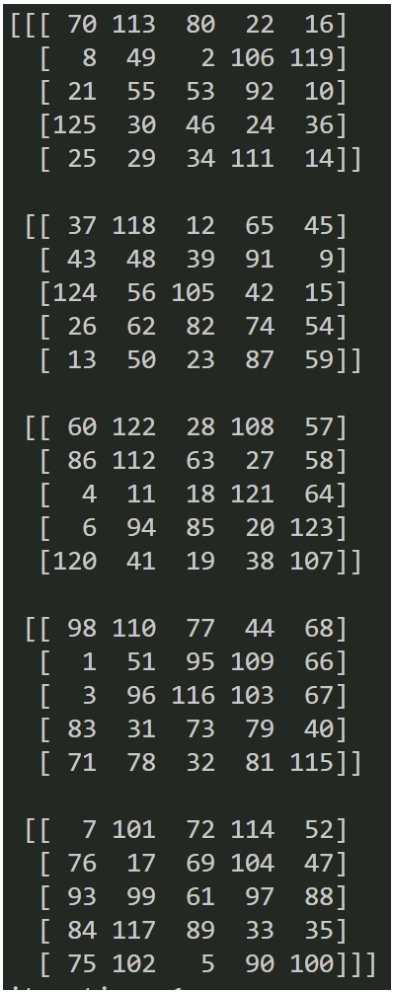
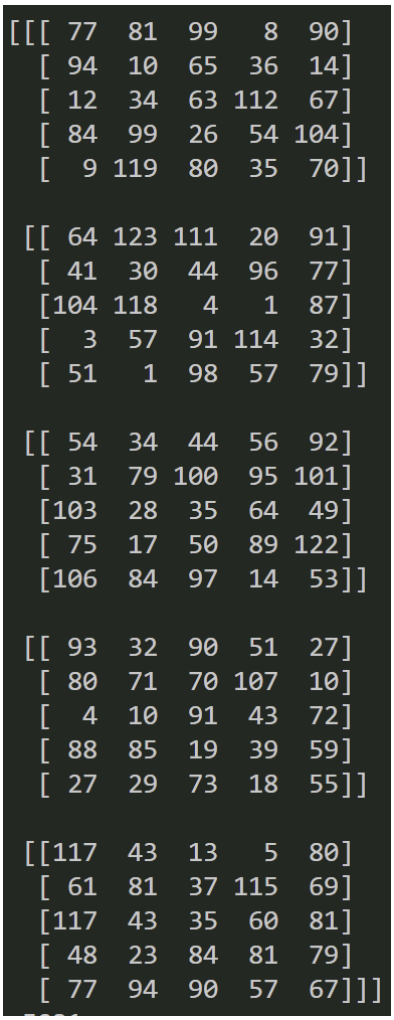
12	 [[[19 8 34 35 113] [108 26 114 88 103] [75 125 118 58 100] [22 102 107 56 110] [9 86 64 42 37]] [[24 49 104 32 21] [83 68 46 14 109] [124 15 63 1 38] [27 121 6 23 97] [71 36 55 11 53]] [[48 4 72 98 43] [66 62 96 52 87] [67 116 20 77 115] [93 44 99 31 70] [111 5 73 74 123]] [[117 91 41 7 3] [84 92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

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

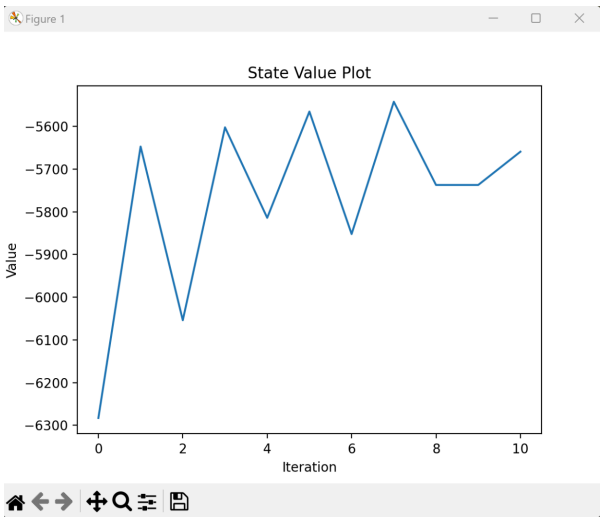
14	 [[[49 46 115 94 24] [76 50 39 117 101] [119 54 66 93 100] [27 86 22 23 32] [45 113 51 104 1]] [[73 4 3 90 85] [102 38 84 48 83] [31 111 63 125 64] [14 91 67 88 92] [25 36 99 52 33]] [[2 12 21 124 29] [43 53 77 7 40] [65 82 87 34 107] [80 11 121 57 28] [96 8 110 95 61]] [[30 26 108 98 10] [105 13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

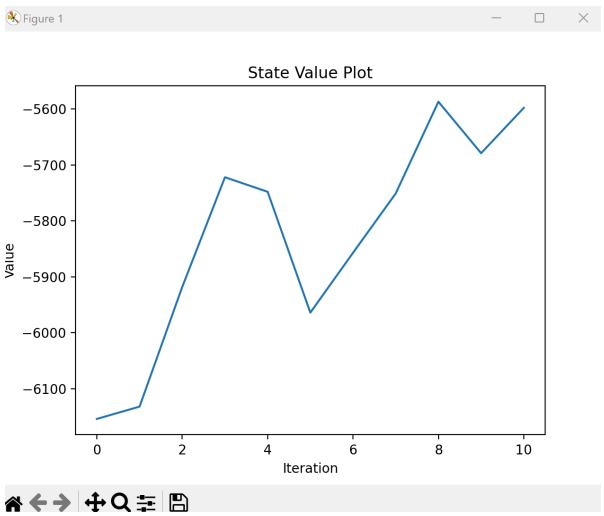
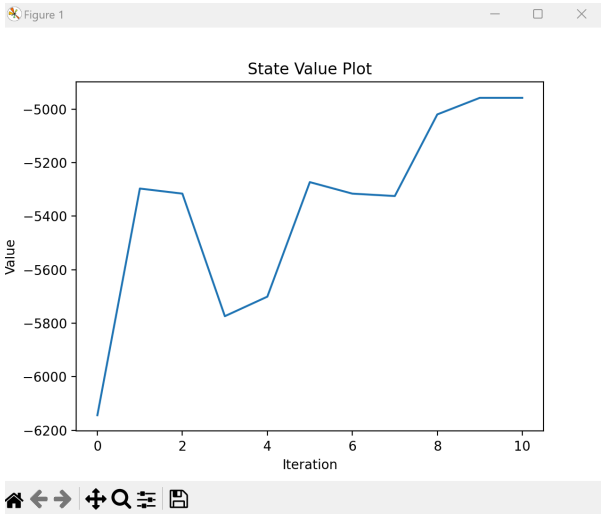
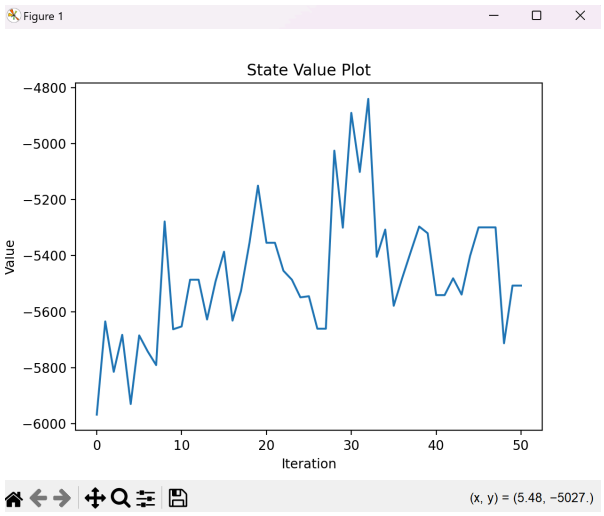
15	 [[[29 44 76 48 8] [42 68 43 21 77] [14 20 55 69 26] [101 18 59 27 22] [41 57 121 38 37]] [[93 98 65 117 49] [28 72 96 91 105] [56 2 1 64 12] [94 9 24 73 7] [87 11 80 15 119]] [[63 58 85 47 109] [62 16 75 81 86] [120 39 30 78 33] [118 116 102 66 123] [10 61 32 4 100]] [[79 84 110 82 112] [125 111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

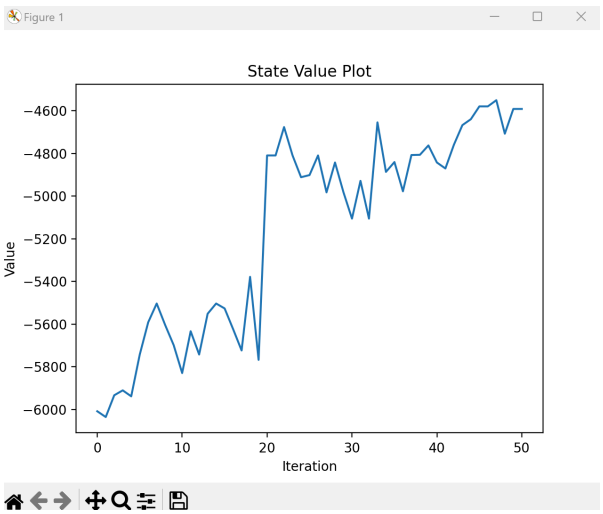
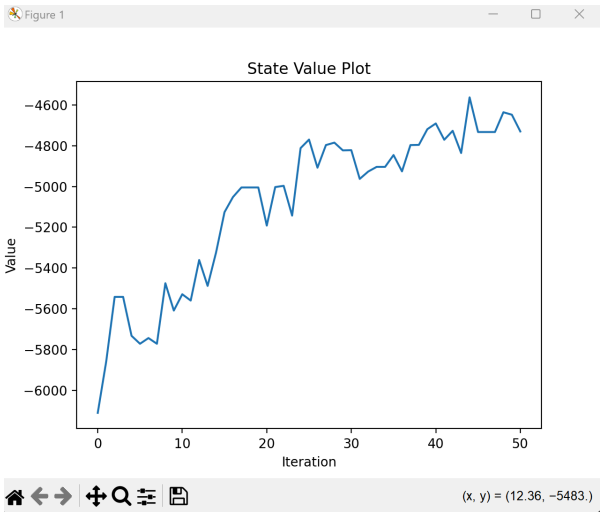
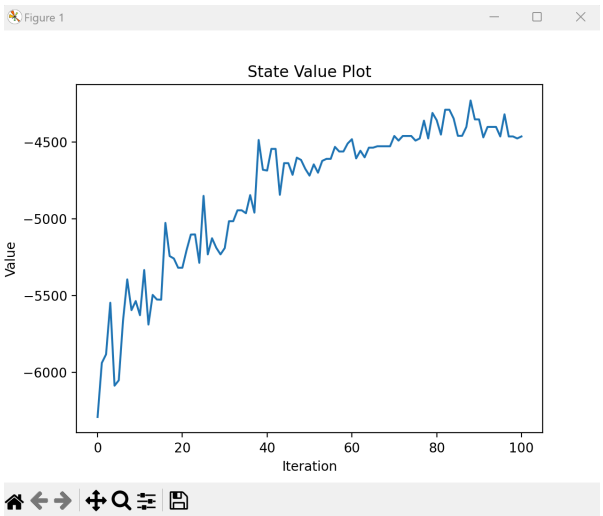
16	 [[[61 9 72 59 19] [38 12 49 13 8] [2 96 18 29 79] [70 65 106 116 32] [1 48 20 118 51]] [[111 93 42 47 76] [88 121 71 16 54] [37 44 26 103 101] [3 60 119 64 124] [109 62 99 30 34]] [[17 11 40 39 4] [110 50 53 120 82] [52 68 5 97 57] [91 6 105 123 35] [75 10 67 7 15]] [[74 84 104 45 43] [89 78 90 86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

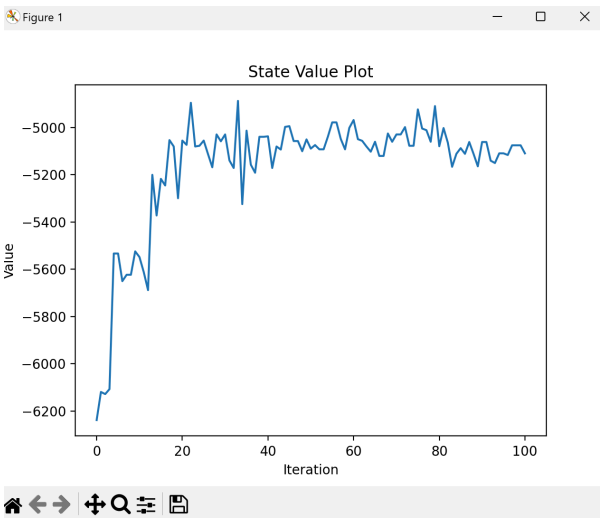
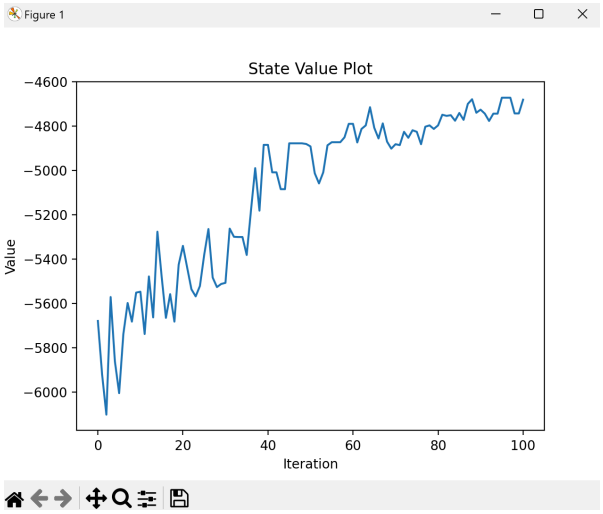
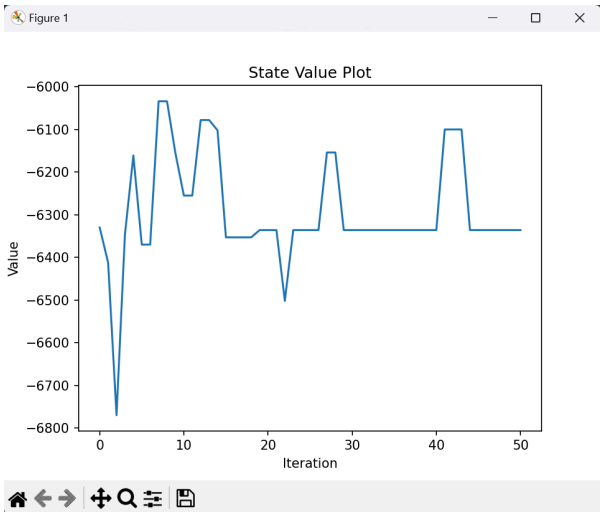
17	 [[[70 113 80 22 16] [8 49 2 106 119] [21 55 53 92 10] [125 30 46 24 36] [25 29 34 111 14]] [[37 118 12 65 45] [43 48 39 91 9] [124 56 105 42 15] [26 62 82 74 54] [13 50 23 87 59]] [[60 122 28 108 57] [86 112 63 27 58] [4 11 18 121 64] [6 94 85 20 123] [120 41 19 38 107]] [[98 110 77 44 68] [1 51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

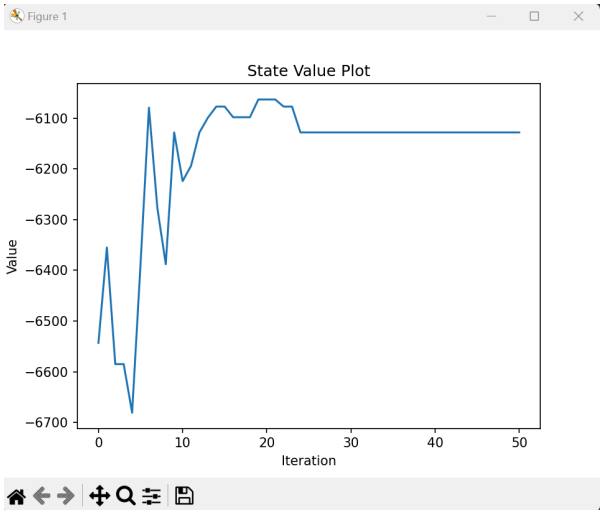
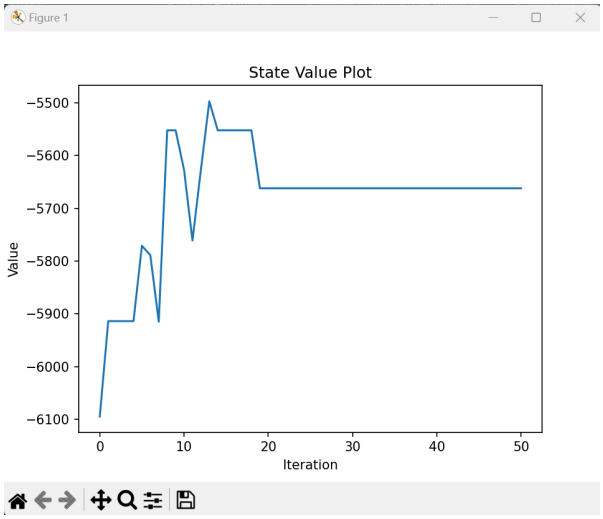
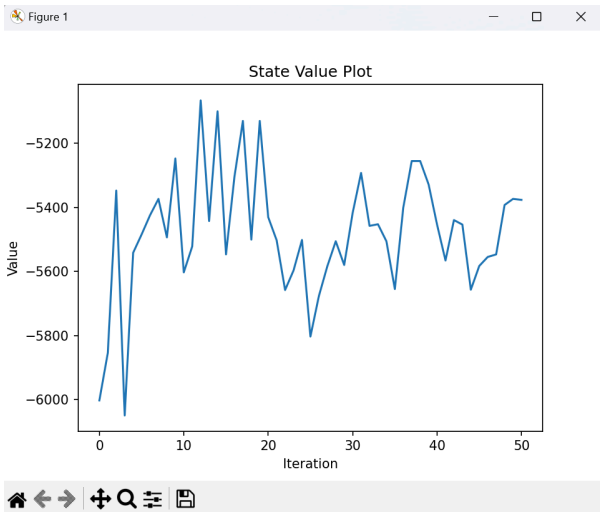
18	<pre>[[[121 40 114 106 70]
	  [41 10 110 64 14]
	  [117 2 96 20 37]
	  [69 30 3 49 51]
	  [9 74 97 17 1]]
	
	 [[29 62 28 79 11]
	  [26 78 55 25 66]
	  [32 113 99 71 46]
	  [59 15 111 65 98]
	  [90 7 13 36 125]]
	
	 [[119 52 100 53 93]
	  [54 88 82 104 19]
	  [21 115 107 38 63]
	  [122 101 43 47 120]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

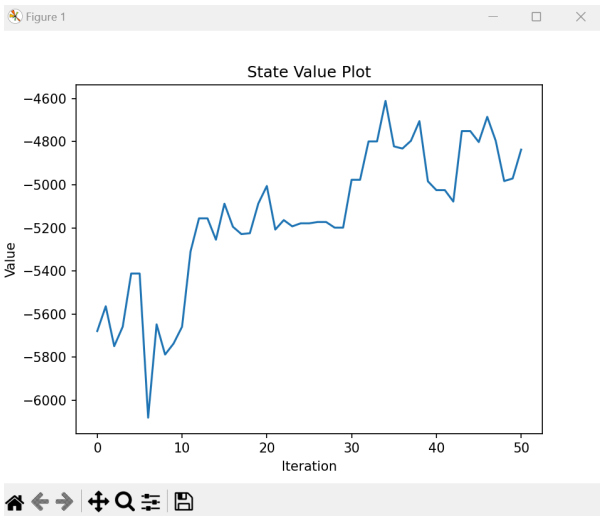
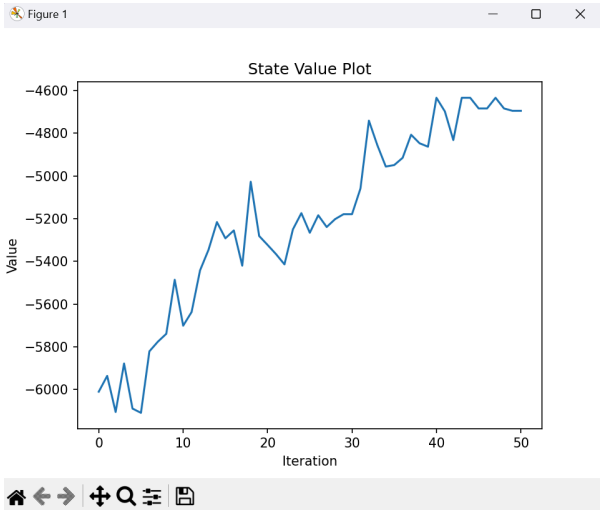
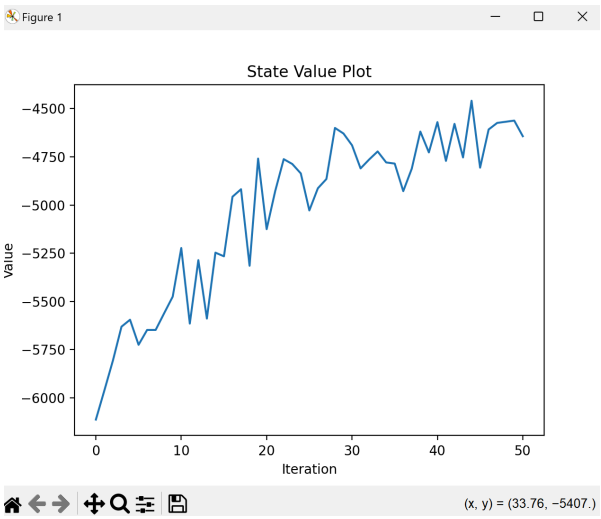
Plot Percobaan 1		

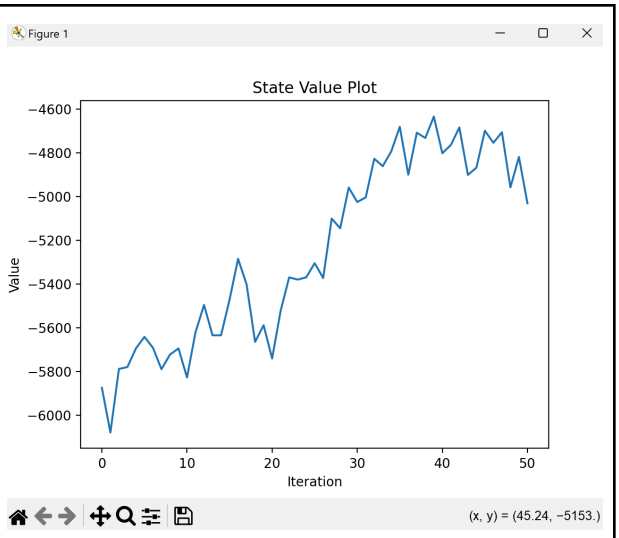
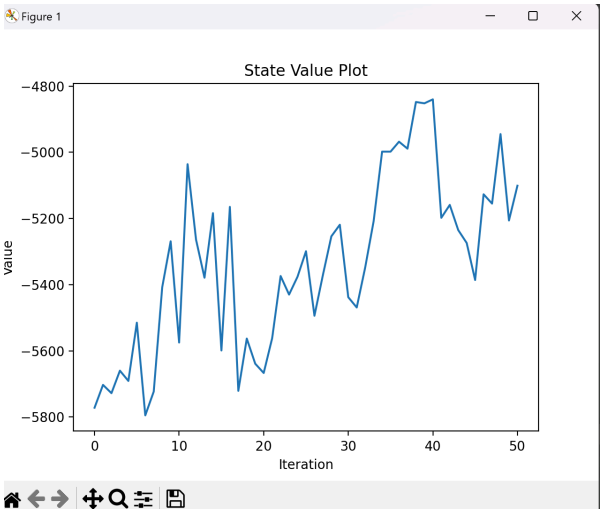
Plot Percobaan 2	
Plot Percobaan 3	
Plot Percobaan 4	

Plot Percobaan 5	
Plot Percobaan 6	
Plot Percobaan 7	

Plot Percobaan 8	
Plot Percobaan 9	
Plot Percobaan 10	

Plot Percobaan 11	
Plot Percobaan 12	
Plot Percobaan 13	

Plot Percobaan 14	
Plot Percobaan 15	
Plot Percobaan 16	

Plot Percobaan 17	
Plot Percobaan 18	

Analisis

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

Steepest Ascent Hill Climbing pada hasil eksperimennya mendapatkan nilai objektif yang cukup besar dan mendekati global optima. Walaupun begitu, besar kemungkinan bahwa solusi yang dihasilkan dari algoritma ini akan terjebak pada *local optimum*.
Hill Climbing with Sideways Move pada hasil eksperimennya mendapatkan nilai objektif yang cukup besar, tetapi secara umum masih lebih rendah daripada Steepest Ascent karena algoritma ini juga mempertimbangkan untuk pindah ke neighbor dengan *value* yang sama. Walaupun begitu, algoritma ini cukup efektif untuk menghindari *plateau*.
Stochastic Hill Climbing pada hasil eksperimennya mendapatkan nilai objektif yang cukup kecil dan jauh dari nilai objektif yang diinginkan. Hal ini karena algoritma ini mencari tetangga secara

random dan iterasi yang dilakukan pun tidak terlalu banyak. Walaupun begitu, algoritma ini akan cukup efektif untuk menghindari *local optimum* karena proses pencarian *neighbor* yang random.
Random Restart Hill Climbing pada hasil eksperimennya mendapatkan nilai objektif yang cukup besar dan cukup mendekati global optima, hasilnya secara umum mirip dengan Steepest Ascent dan Hill Climbing with Sideways Move karena sama-sama mencari *neighbor* terbaik. Algoritma ini juga cukup efektif untuk menghindari *local optimum* karena kemampuannya untuk memulai kembali pencarian dengan *random initial state* apabila tidak menemukan *best neighbor*.
Simulated Annealing pada hasil eksperimennya mendapatkan nilai objektif yang cukup kecil dan jauh dari global optima. Hal ini disebabkan oleh pencarian tetangga yang dilakukan secara random dan rendahnya temperatur yang kami buat dengan pertimbangan waktu yang lama untuk bahasa pemrograman yang kami pakai. Walaupun begitu, apabila disetel dengan temperatur yang cukup tinggi, algoritma ini akan cukup efektif untuk menghindari *local optimum* karena kemampuannya untuk 'turun' apabila probabilitasnya memenuhi.
Genetic Algorithm pada hasil eksperimennya mendapatkan nilai objektif yang cukup kecil dan jauh dari global optima. Hal ini ditentukan oleh jumlah iterasi dan ukuran populasi yang pada awal sudah ditentukan. Ini juga dikarenakan penentuan successor yang random, yaitu penentuan *parent* yang dipilih secara random untuk menciptakan populasi baru. Dan kemudian dengan probabilitas tertentu akan melakukan proses *crossover* ataupun mutasi untuk meningkatkan eksplorasi solusi yang lebih luas. Maka dari itu dari algoritma kami yang melakukan genetic algorithm yang memiliki jumlah iterasi yang sedikit dan populasi yang kurang beragam menjadikan hasil yang didapat kurang optimal dan relatif masih jauh dari global optima.

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

Secara umum, dalam iterasi yang berjalan sebanyak kurang lebih 100 kali, algoritma Hill Climbing cenderung lebih efektif untuk mencari solusi yang mendekati global optima, terutama untuk algoritma Hill Climbing yang mencari *best neighbor* seperti Steepest Ascent Hill Climbing, Hill Climbing with Sideways Move, dan Random Restart Hill Climbing.
Sedangkan algoritma Simulated Annealing dan Stochastic Hill Climbing mendapatkan nilai objektif yang cukup kecil karena pencarian tetangga yang dilakukan secara acak dan iterasi yang tidak terlalu banyak.

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

Pengklasifikasian durasi yang mungkin dapat mewakili pada untuk masing-masing metode yaitu cepat, sedang dan lama. Pada klasifikasi durasi cepat terdapat algoritma Steepest Ascent Hill Climbing dikarenakan proses pencarian yang dilakukan masih belum terlalu luas hanya berfokus pada value dari successor yang terbaik, maka dari itu seringkali terjebak local optima. Lalu pada klasifikasi durasi sedang terdapat algoritma Hill Climbing with Sideways Move dan Stochastic Hill Climbing. Ini dikarenakan proses pencarian sudah mempertimbangkan tindakan

pencegahan untuk menghindari local optima, maka dari itu durasinya relatif bertambah. Kemudian pada klasifikasi lama terdapat algoritma Random Restart Hill Climbing, Simulated Annealing, dan Genetic Algorithm. Ini dikarenakan proses pencarian sudah mempertimbangkan untuk menghindari local optima dan terdapat proses tambahan, seperti pada genetic algorithm ada mutasi dengan probabilitas random untuk proses tersebut akan terjadi, itu yang membuat metode-metode algoritma tersebut relatif lebih lama.

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

Pada Algoritma ini secara garis besar, kekonsistenan ditentukan dengan pemilihan successor yang baik dan tindakan yang dilakukan untuk menghindari local optima. Dalam algoritma variasi Hill Climbing, metode yang mengambil successornya dengan mempertimbangkan successor yang terbaik akan relatif menunjukkan peningkatan yang konsisten dan jika terdapat tindakan yang diambil untuk mencegah local optima maka akan dilakukan dengan menjaga kekonsistenan peningkatan pada value nantinya. Kecuali pada Stochastic Hill Climbing yang memilih successor dengan random, ini membuat konsisten kenaikan dalam valuenya tidak terjamin, tetapi ini akan membuat proses eksplorasi menjadi lebih luas. Sama halnya pada Simulated Annealing dan Genetic Algorithm yang juga menentukan successornya dengan cara random untuk simulated annealing, sedangkan pada genetic algorithm pemilihan *parent* yang diambil untuk menciptakan populasi dipilih secara random. Maka pada algoritma yang menentukan successornya secara random hasil yang didapat akan relatif tidak konsisten, tetapi memperbesar area eksplorasi pada proses pencariannya.

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

Genetic Algorithm merupakan solusi yang efektif untuk mendapatkan global optima yang membuat eksplorasi dapat mendapatkan hasil yang optimal dengan menjaga keragaman dari populasi yang ada, dengan bergantung pada ukuran populasi yang ada dan jumlah iterasinya. Pada Genetic Algorithm, banyak iterasi dan jumlah populasi mempengaruhi solusi dan lamanya durasi pencarian. Semakin banyak iterasi dan jumlah populasi, semakin baik pencarian solusi, karena algoritma memiliki lebih banyak populasi individu dan kesempatan eksplorasi. Namun, durasi pencarian juga akan meningkat seiring dengan meningkatnya iterasi dan populasi.

Simpulan dan Saran

Kesimpulan

Dalam tugas besar ini, beberapa algoritma local search telah diterapkan untuk menyelesaikan persoalan Diagonal Magic Cube, yaitu Steepest Ascent Hill Climbing, Hill Climbing with Sideways Move, Stochastic Hill Climbing, Random Restart Hill Climbing, Simulated Annealing, dan Genetic Algorithm. Hasil eksperimen menunjukkan bahwa:

1. **Steepest Ascent Hill Climbing** cepat mencapai solusi tetapi rentan terhadap local optima, terutama pada konfigurasi yang kompleks.
2. **Hill Climbing with Sideways Move** memperpanjang eksplorasi dan membantu menghindari local optima sementara, namun dengan batasan sideways move yang ketat, masih berisiko tidak menemukan solusi optimal.
3. **Stochastic Hill Climbing** membantu dalam menghindari local optima karena sifatnya yang acak, tetapi memiliki variasi hasil akhir yang cukup besar dan durasi yang relatif lama dibandingkan pendekatan deterministik.
4. **Random Restart Hill Climbing** efektif untuk mencari solusi global dengan memulai ulang pencarian, namun membutuhkan waktu lebih lama dan tidak selalu efisien pada kasus sederhana.
5. **Simulated Annealing** menunjukkan performa yang kurang baik dengan probabilitas pindah state, tetapi membantu menghindari jebakan di local optima, meskipun waktu eksekusinya lama karena proses pendinginan bertahap.
6. **Genetic Algorithm** tidak efektif, tetapi memiliki kemampuan eksplorasi melalui crossover dan mutasi, terutama dalam menjaga variasi populasi yang lebih besar. Namun, pengaturan ukuran populasi dan banyaknya iterasi sangat berpengaruh terhadap performa dan waktu prosesnya.

Secara keseluruhan, pendekatan Hill Climbing sederhana memiliki kinerja lebih baik dalam mendekati global optimum dibandingkan metode Simulated Annealing dan Genetic Algorithm. Pemilihan algoritma sangat tergantung pada trade-off antara waktu komputasi dan kualitas solusi yang diinginkan.

Saran

Pertama, pemilihan algoritma perlu disesuaikan dengan kebutuhan waktu dan ketepatan solusi. Algoritma seperti Simulated Annealing dan Random Restart Hill Climbing dapat menjadi pilihan yang efektif untuk mendapatkan solusi cepat yang mendekati optimal. Simulated Annealing menawarkan kemampuan eksplorasi solusi sementara yang suboptimal, sedangkan Random Restart Hill Climbing mengurangi risiko terjebak pada local optimum. Sementara itu, Genetic Algorithm ideal untuk masalah dengan tingkat kompleksitas tinggi, terutama jika waktu iterasi yang cukup tersedia.

Selanjutnya, pengaturan parameter yang optimal sangat penting dalam memastikan kinerja algoritma. Untuk Simulated Annealing, pengaturan tingkat suhu awal dan cooling rate mempengaruhi kemampuan algoritma dalam menghindari jebakan local optimum dan mencapai solusi lebih baik. Dalam Genetic Algorithm, ukuran populasi dan mutation rate perlu disesuaikan untuk meningkatkan efektivitas pencarian tanpa mengorbankan waktu proses yang berlebihan. Penyesuaian awal melalui eksperimen dapat memberikan panduan dalam menentukan parameter yang paling sesuai untuk tiap algoritma.

Akhirnya, pendekatan hybrid dapat dipertimbangkan sebagai opsi untuk mengatasi keterbatasan masing-masing algoritma. Misalnya, menggunakan Genetic Algorithm untuk membentuk populasi awal yang kuat, kemudian mengoptimalkan solusi lebih lanjut dengan Simulated Annealing, bisa menjadi strategi yang menjanjikan. Eksperimen lebih lanjut perlu dilakukan untuk menguji efektivitas pendekatan hybrid ini dan menilai potensi peningkatan performanya dalam menyelesaikan masalah dengan lebih efisien.

Referensi

GeeksforGeeks. (2024b, March 8). *Genetic algorithms*. https://www.geeksforgeeks.org/genetic-algorithms/

GeeksforGeeks. (2023, April 20). *Introduction to hill climbing: Artificial intelligence*. https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

Local search algorithm. Local Search Algorithm - an overview | ScienceDirect Topics. (n.d.). https://www.sciencedirect.com/topics/computer-science/local-search-algorithm

Search: Joisie (Journal of Information Systems and Informatics Engineering). free website stats program. (n.d.). https://ejournal.pelitaindonesia.ac.id/ojs32/index.php/JOISIE/article/view/2122 References: https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

Simulated annealing algorithm. Simulated Annealing Algorithm - an overview | ScienceDirect Topics. (n.d.). https://www.sciencedirect.com/topics/engineering/simulated-annealing-algorithm

Slide Kuliah

Pembagian Tugas

NIM	Nama	Tugas
18222015	Bihurin Salsabila Firdaus	1. Kode steepest 2. Kode HC with sideways move 3. Pembahasan 4. Kesimpulan dan

		Saran 5. Formatting dokumen
18222055	Dinda Thalia Fahira	1. Kode State 2. Kode MagicCube 3. Kode Main 4. Pembahasan 5. Hasil Eksperimen & Analisis
18222097	Muhammad Nurul Hakim	1. Kode Simulated Annealing 2. Kode Genetic Algorithm 3. Hasil Eksperimen & Analisis
18222099	Dahayu Ramaniya Aurasindu	1. Kode Hill-Climbing 2. Kode Stochastic Hill-Climbing 3. Kode Random Restart Hill-Climbing 4. Deskripsi Persoalan 5. Hasil Eksperimen & Analisis