

# **IF3070 Dasar Intelegensi Artifisial**

## **Tugas Besar 1**



**Disusun oleh:**

**Kelompok 48**

Rajendra Farras Rayhan / 18222105

Lina Azizah R.H. / 18222107

Gracya Tio Damena S. / 18222110

M. Kasyfil Aziz / 18222127

**Program Studi Sistem dan Teknologi Informasi**

**Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung**

**Jl. Ganesha 10, Bandung 40132**

# Daftar Isi

<b>Daftar Isi</b>	<b>1</b>
<b>1. Deskripsi Persoalan</b>	<b>2</b>
<b>2. Pembahasan</b>	<b>3</b>
2.1. Pemilihan Objective Function	3
2.2. Implementasi Algoritma Local Search	3
2.2.2. Algoritma Steepest Ascent Hill-climbing	12
2.2.3. Algoritma Hill-climbing with Sideways Move	16
2.2.4. Algoritma Random Restart Hill-climbing	22
2.2.5. Algoritma Stochastic Hill-climbing	31
2.2.6. Algoritma Simulated Annealing	36
2.2.7. Algoritma Genetic	46
2.3. Hasil Eksperimen dan Analisis	66
2.3.1. Algoritma Steepest Ascent Hill-climbing	66
2.3.2. Algoritma Hill-climbing with Sideways Move	72
2.3.3. Algoritma Random Restart Hill-climbing	78
2.3.4. Algoritma Stochastic Hill-climbing	85
2.3.5. Algoritma Simulated Annealing	90
2.3.6. Algoritma Genetic	100
2.3.6.1. Jumlah populasi sebagai kontrol	101
2.3.6.2. Jumlah iterasi sebagai kontrol	118
<b>3. Kesimpulan dan Saran</b>	<b>137</b>
<b>4. Pembagian Tugas</b>	<b>138</b>
<b>5. Referensi</b>	<b>139</b>



## 2. Pembahasan

### 2.1. Pemilihan *Objective Function*

*Objective function* adalah sebuah fungsi yang akan digunakan untuk menyelesaikan permasalahan optimasi. Hasil yang didapatkan dengan algoritma pencarian nantinya akan dibandingkan dengan fungsi objektif untuk menilai efektivitas solusi algoritma *local search* yang digunakan. *Objective function* dalam memecahkan permasalahan *Diagonal Magic Cube* melibatkan syarat-syarat yang telah disebutkan pada deskripsi permasalahan.

Setiap syarat-syarat yang telah disebutkan di deskripsi permasalahan akan dikurangi dengan *magic number* yang telah ada, sehingga jika semakin mendekati 0 maka *error* dimensi tersebut semakin kecil. Maka, hasil dengan nilai 0 adalah hasil terbaik untuk perhitungan ini. Berikut adalah rumus *objective function* yang digunakan

$$f(x) = \sum_{i=1}^n |Sum(baris_i) - M| + \sum_{j=1}^n |Sum(kolom_j) - M| + \sum_{k=1}^n |Sum(tiang_k) - M| + \sum_{d=1}^D |Sum(diagonal_d) - M|$$

**Keterangan:**

$n$  = Ukuran *magic cube*

$d$  = Jumlah diagonal

$i, j, k$  = Indeks,  $0 < i, j, k \leq n$

$a_{i,j,k}$  = Posisi elemen di baris ke- $i$ , kolom ke- $j$ , tiang ke- $k$

$M$  = *Magic number*

### 2.2. Implementasi Algoritma *Local Search*

#### 2.2.1. Implementasi *Magic Cube*

Sebelum mengimplementasikan algoritma *local search*, diperlukan algoritma untuk membuat *diagonal magic cube* yang akan digunakan sebagai penguji efektivitas algoritma *local search*. Terdapat tiga komponen yang diperlukan, yakni *magic\_cube*, *neighbor\_state*, dan *objective\_function*.

Berikut merupakan *source code* dari implementasi komponen-komponen implementasi *magic cube*

```
#magic_cube.py
import numpy as np
import random

class MagicCube:
    def __init__(self, size=5):
        self.size = size
        self.magic_number = 315
        self.cube = self.initialize_cube()

    def initialize_cube(self):
        numbers = list(range(1, self.size**3 + 1))
        random.shuffle(numbers)
        return np.array(numbers).reshape((self.size, self.size,
self.size))

    def display(self):
        print(self.cube)
```

```
#neighbor_state.py
import numpy as np
import random
from cube.magic_cube import MagicCube

class NeighborState:
    def __init__(self, magic_cube):
        self.magic_cube = magic_cube

    # Memilih dua posisi acak berbeda
```

```

def select_random_positions(self):
    # Mencari random posisi
    i1, j1, k1 = random.randint(0, self.magic_cube.size - 1),
random.randint(0, self.magic_cube.size - 1), random.randint(0,
self.magic_cube.size - 1)
    i2, j2, k2 = random.randint(0, self.magic_cube.size - 1),
random.randint(0, self.magic_cube.size - 1), random.randint(0,
self.magic_cube.size - 1)

    # Memastikan posisi 1 dan posisi 2 tidak sama
    while (i1, j1, k1) == (i2, j2, k2):
        i2, j2, k2 = random.randint(0, self.magic_cube.size -
1), random.randint(0, self.magic_cube.size - 1),
random.randint(0, self.magic_cube.size - 1)

    return i1, j1, k1, i2, j2, k2

# Melakukan pencarian neighbor random
def generate_neighbor(self):
    neighbor = MagicCube(size=self.magic_cube.size)
    neighbor.cube = self.magic_cube.cube.copy()

    i1, j1, k1, i2, j2, k2 = self.select_random_positions()

    # Menukar neighbor
    neighbor.cube[i1, j1, k1], neighbor.cube[i2, j2, k2] =
neighbor.cube[i2, j2, k2], neighbor.cube[i1, j1, k1]

    return neighbor

```

```

#objective_function.py
import numpy as np

```

```

class ObjectiveFunction:
    def __init__(self, magic_cube):
        self.magic_cube = magic_cube
        self.size = magic_cube.size
        self.magic_number = magic_cube.magic_number

    def calculate(self):
        cube = self.magic_cube.cube
        total_error = 0

        for i in range(self.size):
            for j in range(self.size):
                # Jumlah error di baris tiap layer
                total_error += abs(np.sum(cube[i, j, :]) -
self.magic_number)

                # Jumlah error di kolom tiap layer
                total_error += abs(np.sum(cube[i, :, j]) -
self.magic_number)

                # Jumlah error di tiang tiap layer
                total_error += abs(np.sum(cube[:, i, j]) -
self.magic_number)

            # Diagonal bidang pada layer horizontal
            for i in range(self.size):
                horizontal_diag_1 = np.sum(np.diag(cube[i]))
                horizontal_diag_2 = np.sum(np.diag(np.fliplr(cube[i])))
                total_error += abs(horizontal_diag_1 -
self.magic_number)
                total_error += abs(horizontal_diag_2 -
self.magic_number)

```

```

        # Diagonal menyilang horizontal dari depan ke belakang di
        # tiap baris
        for j in range(self.size):
            front_back_diag_1 = np.sum([cube[i, j, i] for i in
            range(self.size)])
            front_back_diag_2 = np.sum([cube[i, j, self.size - 1 -
            i]

            for i in range(self.size)])
            total_error += abs(front_back_diag_1 -
            self.magic_number)
            total_error += abs(front_back_diag_2 -
            self.magic_number)

        # Diagonal menyilang vertikal dari sisi kiri ke kanan di
        # tiap kolom
        for k in range(self.size):
            left_right_diag_1 = np.sum([cube[i, i, k] for i in
            range(self.size)])
            left_right_diag_2 = np.sum([cube[i, self.size - 1 - i,
            k] for i in range(self.size)])
            total_error += abs(left_right_diag_1 -
            self.magic_number)
            total_error += abs(left_right_diag_2 -
            self.magic_number)

        # Jumlah error diagonal ruang kubus
        total_error += abs(np.sum([cube[i, i, i] for i in
            range(self.size)]) - self.magic_number)
        total_error += abs(np.sum([cube[i, i, self.size - 1 - i]
            for i in range(self.size)]) - self.magic_number)
        total_error += abs(np.sum([cube[i, self.size - 1 - i, i]

```



```

for i in range(self.size)]:) - self.magic_number)
    total_error += abs(np.sum([cube[self.size - 1 - i, i, i]
for i in range(self.size)]:) - self.magic_number)

    return total_error

```

Selain komponen-komponen *magic cube*, terdapat *main.py* yang digunakan untuk memanggil algoritma-algoritma *local search*.

```

from algorithm.genetic_algorithm.GeneticAlgorithm import GeneticAlgorithm
from algorithm.hill_climbing.SteepestAscent import SteepestAscent
from algorithm.hill_climbing.RandomRestart import RandomRestart
from algorithm.simulated_annealing.SimulatedAnnealing import
SimulatedAnnealing
from algorithm.hill_climbing.Sideway import SidewayHillClimbing
from algorithm.hill_climbing.Stochastic import StochasticHillClimbing
from ascii import print_dual_color_ascii, print_loading_animation

def main_menu():

print("=====START=====
=====")

    ascii_header = """
    SELAMAT
    DATANG
    """

    print_dual_color_ascii(ascii_header, 3)
    while True:

```

```
print("=====  
===")  
  
# Tampilkan menu opsi  
print("""  
+-----+  
|           PILIH ALGORITMA YANG           |  
|               INGIN DICoba                |  
+-----+  
""")  
# print("\nPilih algoritma yang ingin dicoba:")  
print("1. Steepest Ascent Hill-Climbing")  
print("2. Hill-Climbing with Sideways Move")  
print("3. Random Restart Hill-Climbing")  
print("4. Stochastic Hill-Climbing")  
print("5. Simulated Annealing")  
print("6. Genetic Algorithm")  
print("7. Keluar dari program")  
  
# Menerima input opsi dari pengguna  
pilihan = input("Masukkan nomor opsi (1-7): ")  
  
cube = ""  
  
          .....  
        .....:---:  
.....:-::--::~:  
.+#*+=--::--::-+*#=.  
.+#####*+==-::-=**#####=  
.+#####*+=---=+**#####=  
.+#####-.  
.=####@%#####%###-.  
.=####%#####%#####-.  
.=####%#####%#####-.  
.=####%#####%#####*:..  
.=####%#####%#####%###*:..  
.=####%@#####%#####%#@###*:..  
...=####%#####%#####%#####*:..  
...=####%#####*:*..  
.....:-+#####*=~:..  
.....::=*#####*=~:..  
.....:-+#####+-:~:..  
          ..:-++=:..  
""")  
  
# Menentukan aksi berdasarkan input  
if (pilihan == '1'):  
    print("\nAnda memilih Steepest Ascent Hill-Climbing.")  
  
    print_dual_color_ascii(cube, 2)  
    print_loading_animation("Proses solving magic cube")
```

```

        sa = SteepestAscent()
        sa.evaluateNeighbor()

    elif (pilihan == '2'):
        print("\nAnda memilih Hill-Climbing with Sideways Move")
        try:
            ms = int(input("Masukkan variasi maksimal sideways moves:
"))
            print(f"\nMenjalankan pengujian dengan berbagai variasi
jumlah maksimal {ms} sideways moves:")

            print_dual_color_ascii(cube, 2)
            print_loading_animation("Proses solving magic cube")

            shc = SidewayHillClimbing(max_sideways_moves=ms)
            shc.evaluateNeighbor()

        except ValueError:
            print("Input harus berupa angka yang valid!")
    elif (pilihan == '3'):
        print("\nAnda memilih Random Restart Hill-Climbing.")
        try:
            print("\nMasukkan jumlah maksimal restart (misalnya:
10):")
            max_restart = int(input("Jumlah maksimal restart: "))

            print(f"\nMenjalankan Random Restart Hill Climbing dengan
maksimal restart {max_restart}")
            print_dual_color_ascii(cube, 2)
            print_loading_animation("Proses solving magic cube")

            rr = RandomRestart(max_restarts=max_restart)
            rr.randomRestart()

        except ValueError:
            print("Input harus berupa angka yang valid!")

    elif (pilihan == '4'):
        print("\nAnda memilih Algoritma Stochastic Hill-Climbing")
        try:
            maximum_iteration = int(input("Jumlah maksimal iterasi:
"))
            print(f"\nPengujian dengan maksimal {maximum_iteration}
iterasi:")

            print_dual_color_ascii(cube, 2)
            print_loading_animation("Proses solving magic cube")

```

```

        shc =
StochasticHillClimbing(max_iteration=maximum_iteration)
        shc.evaluateNeighbor()

    except ValueError:
        print("Input harus berupa angka yang valid!")
    elif (pilihan == '5'):
        print("\nAnda memilih Simulated Annealing")
        try:
            # Menerima input nilai yang dibutuhkan
            starting_tem_input = int(input("Masukkan nilai temperatur
awal (misalnya: 1000) : "))
            cooling_rate_input = float(input("Masukkan nilai cooling
rate (misalnya: 0.95) : "))
            minimum_tem_input = float(input("Masukkan nilai
temperatur minimal (misalnya: 0.1) : "))

            # Menjalankan algoritma sesuai dengan input
            print_dual_color_ascii(cube, 2)
            print_loading_animation("Proses solving magic cube")
            sa = SimulatedAnnealing(starting_tem_input,
cooling_rate_input, minimum_tem_input)
            sa.simulatedannealing()

        except ValueError:
            print("Input harus berupa angka yang valid!")

    elif (pilihan == '6'):
        print("\nAnda memilih Genetic Algorithm.")

        print("\n===Pengujian dengan variasi jumlah populasi dan
iterasi tetap===")
        populations = GeneticAlgorithm.get_valid_input("\nMasukkan 3
variasi jumlah populasi (misalnya: 30 50 100): ", min_value=2)
        iteration_fixed = GeneticAlgorithm.get_valid_input("Masukkan
jumlah iterasi tetap (contoh: 100): ", min_value=1)[0]

        GeneticAlgorithm.run_multiple_tests(populations,
[iteration_fixed], is_fixed_iteration=True)

        print("\n===Pengujian dengan variasi jumlah iterasi dan
populasi tetap===")
        iterations = GeneticAlgorithm.get_valid_input("\nMasukkan 3
variasi jumlah iterasi (contoh: 50 100 150): ", min_value=1)
        population_fixed = GeneticAlgorithm.get_valid_input("Masukkan
jumlah populasi tetap (contoh: 20): ", min_value=2)[0]

        GeneticAlgorithm.run_multiple_tests([population_fixed],
iterations, is_fixed_iteration=False)

```



Berikut merupakan *source code* dari implementasi algoritma *steepest-ascent hill-climbing*

```
import matplotlib.pyplot as plt
from cube.magic_cube import MagicCube
from cube.objective_function import ObjectiveFunction
from cube.neighbor_state import NeighborState

class SteepestAscent:
    def __init__(self):
        self.current_state = MagicCube()
        self.current_value = 999999
        self.objective_values = []

    def searchbestNeighbor(self):
        best_neighbor = None
        best_neighbor_value = 999999

        for i in range(100):
            neighbor =
NeighborState(self.current_state).generate_neighbor()
            neighbor_value = ObjectiveFunction(neighbor).calculate()

            if neighbor_value < best_neighbor_value:
                best_neighbor_value = neighbor_value
                best_neighbor = neighbor

        return best_neighbor, best_neighbor_value

    def evaluateNeighbor(self):
        total_iteration = 0

        while True:
            best_neighbor, best_neighbor_value = self.searchbestNeighbor()
```

```

    #comparing
    if best_neighbor_value < self.current_value:
        self.current_state = best_neighbor
        self.current_value = best_neighbor_value
        total_iteration += 1
        self.objective_values.append(self.current_value)
    else:
        break

    return total_iteration

@staticmethod
def plot_progression(objective_values, title="Objective Function
Value Progression"):
    plt.figure(figsize=(10, 6))
    plt.plot(objective_values, label='Objective Value', color='blue')
    plt.title(title)
    plt.xlabel('Iteration')
    plt.ylabel('Objective Function Value')
    plt.grid(True, linestyle='--', linewidth=0.5)
    plt.legend()
    plt.tight_layout()
    plt.show()

```

<b>Nama Fungsi</b>	searchbestNeighbor(self)
<b>Deskripsi</b>	Sesuai dengan konsep <i>steepest ascent hill-climbing</i> yang menggunakan <i>highest-valued successor of current</i> sebagai <i>neighbor</i> -nya, fungsi ini melakukan pencarian <i>neighbor</i> yang di-generate dari <i>current state</i> kubus dan <i>neighbor</i> memiliki nilai mendekati <i>global maximum</i> atau mendekati 0. Pada iterasi pencarian <i>best neighbor</i> ini, diberikan maksimal iterasi untuk memastikan bahwa eksplorasi <i>neighbor</i> -nya dilakukan secara lebih luas,

	namun tetap dengan batasan untuk mencegah <i>infinite loop</i> dan/atau <i>run time</i> yang terlalu lama.
<b>Output</b>	best_neighbor: MagicCube, best_neighbor_value: int
<b>Source Code</b>	<pre> def searchbestNeighbor(self):     best_neighbor = None     best_neighbor_value = 999999      for i in range(100):         neighbor = NeighborState(self.current_state).generate_neighbor()         neighbor_value = ObjectiveFunction(neighbor).calculate()          if neighbor_value &lt; best_neighbor_value:             best_neighbor_value = neighbor_value             best_neighbor = neighbor      return best_neighbor, best_neighbor_value </pre>

<b>Nama Fungsi</b>	evaluateNeighbor(self)
<b>Deskripsi</b>	<p>Fungsi ini merepresentasikan proses pencarian <i>hill-climbing</i>. Fungsi ini akan menjalankan pemeriksaan <i>neighbor</i> sesuai dengan konsep <i>steepest ascent hill-climbing</i>. Fungsi ini membandingkan <i>current value</i> dengan <i>neighbor value</i>.</p> <p>Jika <i>neighbor</i> memiliki nilai yang lebih optimal daripada <i>current state</i>, <i>neighbor</i> tersebut akan menjadi <i>current state</i> baru. Pemeriksaan <i>neighbor</i> dilakukan hingga ditemukan <i>neighbor</i> yang nilainya sama dengan atau tidak lebih baik dari nilai <i>current state</i>.</p>
<b>Output</b>	total_iteration: int



<b>Source Code</b>	<pre> def evaluateNeighbor(self):     total_iteration = 0      while True:         best_neighbor, best_neighbor_value = self.searchbestNeighbor()          #comparing         if best_neighbor_value &lt; self.current_value:             self.current_state = best_neighbor             self.current_value = best_neighbor_value             total_iteration += 1  self.objective_values.append(self.current_value)         else:             break      return total_iteration </pre>
--------------------	---

### 2.2.3. Algoritma *Hill-climbing with Sideways Move*

*Hill-climbing with sideways move* merupakan algoritma yang efektif untuk menghadapi *shoulder* atau *flat local maximum (plateau)*. Pergerakan pencarian solusi akan bergerak ke samping, *neighbor* dengan nilai yang sama. Hal ini dilakukan untuk keluar dari *shoulder* dan mendapatkan solusi terbaik. Variasi ini memiliki probabilitas menemukan *global optimum* lebih tinggi daripada *steepest hill-climbing*.

Berikut merupakan *source code* dari implementasi algoritma *hill-climbing with sideways move*

```

from cube.magic_cube import MagicCube
from cube.objective_function import ObjectiveFunction

```

```

from cube.neighbor_state import NeighborState
import matplotlib.pyplot as plt
import time

class SidewayHillClimbing:
    def __init__(self, max_sideways_moves):
        self.current_state = MagicCube()
        self.current_value =
        ObjectiveFunction(self.current_state).calculate()
        self.max_sideways_moves = max_sideways_moves
        self.iterations = 0
        self.objective_values = []

    def searchbestNeighbor(self):
        best_neighbor = None
        best_neighbor_value = 99999

        for _ in range(100):
            neighbor =
NeighborState(self.current_state).generate_neighbor()
            neighbor_value = ObjectiveFunction(neighbor).calculate()

            if neighbor_value < best_neighbor_value:
                best_neighbor = neighbor
                best_neighbor_value = neighbor_value

        return best_neighbor

    def evaluateNeighbor(self):
        print("\nState Awal:")
        self.current_state.display()
        print(f"Nilai Initial Objective Function: {self.current_value}")

```

```

sideways_moves = 0
start_time = time.time()

while True:
    best_neighbor = self.searchbestNeighbor()
    best_neighbor_value =
ObjectiveFunction(best_neighbor).calculate()
    self.iterations += 1

    self.objective_values.append(self.current_value)

    if best_neighbor_value < self.current_value:
        self.current_state = best_neighbor
        self.current_value = best_neighbor_value
        sideways_moves = 0 # RESET

    elif best_neighbor_value == self.current_value and
sideways_moves < self.max_sideways_moves:
        self.current_state = best_neighbor
        sideways_moves += 1

    else:
        break

self.objective_values.append(self.current_value)

end_time = time.time()
total_duration = end_time - start_time

print(f"State Akhir: ")
self.current_state.display()
print(f"Nilai Final Objective Function: {self.current_value}")

```

```

print(f"Jumlah Iterasi: {self.iterations}")
print(f"Total search duration: {total_duration:.2f} seconds")

results = []

results.append((self.objective_values, f"Jumlah Sideway
Moves: {sideways_moves}", total_duration))
SidewayHillClimbing.plot_multiple_runs(results,
title="Perbandingan objective function terhadap banyak iterasi
yang telah dilewati menggunakan Sideways Hill-Climbing")

@staticmethod
def plot_multiple_runs(results, title="Perbandingan
Objective Function dan Iterasi"):
    fig, axes = plt.subplots(1, len(results), figsize=(18, 6),
sharey=True)

    if len(results) == 1:
        axes = [axes]

    fig.suptitle(title)

    for idx, (objective_values, run_label, total_duration) in
enumerate(results):
        ax = axes[idx]
        iterations = range(len(objective_values))

        ax.plot(iterations, objective_values, label='Objective
Value', color='blue')
        ax.set_title(run_label)
        ax.set_xlabel('Iteration')
        if idx == 0:
            ax.set_ylabel('Objective Function Value')

```

```

        ax.grid(True, linestyle='--', linewidth=0.5)
        ax.legend()

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

```

<b>Nama Fungsi</b>	searchbestNeighbor(self)
<b>Deskripsi</b>	Fungsi ini digunakan untuk melakukan pencarian <i>neighbor</i> dengan <i>value</i> terbaik berdasarkan <i>objective value</i> . Fungsi ini akan melakukan pencarian terhadap beberapa <i>neighbor</i> secara sekaligus. Jika <i>neighbor</i> tersebut memiliki nilai yang lebih rendah dari <i>current_state</i> , maka <i>neighbor</i> tersebut menjadi <i>best_neighbor</i> .
<b>Output</b>	best_neighbor: MagicCube
<b>Source Code</b>	<pre> def searchbestNeighbor(self):     best_neighbor = None     best_neighbor_value = 99999      for _ in range(100):         neighbor = NeighborState(self.current_state).generate_neighb or()         neighbor_value = ObjectiveFunction(neighbor).calculate()          if neighbor_value &lt; best_neighbor_value:             best_neighbor = neighbor             best_neighbor_value = neighbor_value </pre>

	<b>return</b> best_neighbor
--	-----------------------------

<b>Nama Prosedur</b>	evaluateNeighbor(self)
<b>Deskripsi</b>	Prosedur ini akan menjalankan algoritma <i>hill-climbing with sideways move</i> . Prosedur ini akan melakukan pencarian <i>neighbor</i> terbaik menggunakan <i>searchbestNeighbor()</i> . Jika nilai <i>best_neighbor</i> lebih rendah, <i>neighbor</i> tersebut akan menjadi <i>current_state</i> baru dan jumlah <i>sideways_move</i> = 0. Namun, jika nilai <i>best_neighbor</i> sama dengan <i>current_state</i> , fungsi akan bergerak ke samping dengan menerima <i>best_neighbor</i> tersebut sebagai <i>current_state</i> dan menambah jumlah <i>sideways_move</i> jika jumlah gerakan ke samping belum melebihi jumlah maksimal <i>sideways_move</i> .
<b>Output</b>	
<b>Source Code</b>	<pre> def evaluateNeighbor(self):     sideways_moves = 0     print("\nState Awal:")     self.current_state.display()     print(f"Nilai Initial Objective Function: {self.current_value}")     start_time = time.time()      while True:         best_neighbor = self.searchbestNeighbor()         best_neighbor_value = ObjectiveFunction(best_neighbor).calculate()         self.iterations += 1          self.objective_values.append(self.current_value ) </pre>

```

        if best_neighbor_value < self.current_value:
            self.current_state = best_neighbor
            self.current_value = best_neighbor_value
            sideways_moves = 0 # RESET

        elif best_neighbor_value ==
self.current_value and sideways_moves <
self.max_sideways_moves:
            self.current_state = best_neighbor
            sideways_moves += 1

        else:
            break

        self.objective_values.append(self.current_value
)

        end_time = time.time()
        total_duration = end_time - start_time

        print(f"State Akhir: ")
            self.current_state.display()
            print(f"Nilai Final Objective Function:
{self.current_value}")

        print(f"Jumlah Iterasi: {self.iterations}")
        print(f"Total search duration:
{total_duration:.2f} seconds")

        results = []

        results.append((self.objective_values, f"Jumlah
Sideway Moves: {sideways_moves}", total_duration))

```

```
SidewayHillClimbing.plot_multiple_runs(results,  
title="Perbandingan objective function terhadap  
banyak iterasi yang telah dilewati menggunakan  
Sideways Hill-Climbing")
```

#### 2.2.4. Algoritma *Random Restart Hill-climbing*

*Random restart hill-climbing* merupakan algoritma *hill-climbing* yang akan melakukan pencarian baru ketika berada di *local optimum*. Pencarian baru tersebut dilakukan secara acak dalam beberapa kali sampai mendapatkan solusi terbaik.

Berikut merupakan *source code* dari implementasi algoritma *random restart hill-climbing*

```
from cube.magic_cube import MagicCube  
from cube.objective_function import ObjectiveFunction  
from cube.neighbor_state import NeighborState  
import matplotlib.pyplot as plt  
import time  
  
class RandomRestart:  
    def __init__(self, max_restarts):  
        self.max_restarts = max_restarts  
        self.best_value = 999999  
        self.best_state = None  
        self.objective_values = []  
  
    def searchbestNeighbor(self, current_state):  
        best_neighbor_value = 999999  
        best_neighbor = None  
  
        for i in range(100):
```



```

        neighbor = NeighborState(current_state).generate_neighbor()
        #generate neighbor
        neighbor_value = ObjectiveFunction(neighbor).calculate()
        #check neighbor value

        if neighbor_value < best_neighbor_value:
            best_neighbor_value = neighbor_value
            best_neighbor = neighbor

    return best_neighbor, best_neighbor_value

def evaluateNeighbor(self, initial_state):
    current_state = initial_state
    current_value =
ObjectiveFunction(current_state).calculate()
    iterations = 0

    while True:
        best_neighbor, best_neighbor_value =
self.searchbestNeighbor(current_state)

        #comparing
        if best_neighbor_value < current_value:
            current_state = best_neighbor
            current_value = best_neighbor_value
            iterations += 1
        else:
            break
        self.objective_values.append(current_value)
    return current_state, current_value, iterations

def randomRestart(self):
    iteration_per_start = []

```

```

final_value_per_start = []

#hill climb pertama
initial_state = MagicCube()
print(f"\nState Awal:")
initial_state.display()

total_iterations = 0

start_time = time.time()

final_state, final_value, iterations =
self.evaluateNeighbor(initial_state)
iteration_per_start.append(iterations)
final_value_per_start.append(final_value)
total_iterations += iterations

if final_value < self.best_value:
    self.best_value = final_value
    self.best_state = final_state

for restart in range(self.max_restarts): #restart
    initial_state = MagicCube()

    final_state, final_value, iterations =
self.evaluateNeighbor(initial_state)
    total_iterations += iterations
    iteration_per_start.append(iterations)
    final_value_per_start.append(final_value)

    if final_value < self.best_value:
        self.best_value = final_value
        self.best_state = final_state

```

```

        execute_time = time.time() - start_time

        #print information
        for step in range(len(iteration_per_start)):
            if step == 0:
                print(f"\nLangkah pertama - Total Iterations:
{iteration_per_start[step]}, Iterations Value:
{final_value_per_start[step]}")
            else:
                print(f"Restart ke-{step} - Total Iterations:
{iteration_per_start[step]}, Iterations Value:
{final_value_per_start[step]}")

        print(f"\nState Akhir:")
        self.best_state.display()
        print(f"\nNilai objective akhir: {self.best_value}")
        print(f"Waktu yang dibutuhkan: {execute_time:.2f} detik\n")

        self.show_plot(
            objective_values=self.objective_values,
            max_restarts=len(self.objective_values),
            title=f'Perkembangan Nilai Objective Function\nMaksimal
Restart: {self.max_restarts}\nWaktu: {execute_time:.2f} detik'
        )

        @staticmethod
        def show_plot(objective_values, max_restarts,
            title="Perkembangan Nilai Objective Function"):
            plt.figure(figsize=(10, 6))
            plt.plot(objective_values, label='Objective Value',
                color='blue')
            plt.title(title)

```

```
plt.xlabel('Iteration')
plt.ylabel('Objective Function Value')
plt.grid(True, linestyle='--', linewidth=0.5)
plt.legend()
plt.tight_layout()
plt.show()
```

<b>Nama Fungsi</b>	searchbestNeighbor(self, current_state)
<b>Deskripsi</b>	Pencarian neighbor pada Random Restart pada intinya sama dengan konsep <i>steepest ascent hill-climbing</i> yang menggunakan <i>highest-valued successor of current</i> sebagai <i>neighbor</i> -nya. Fungsi ini melakukan pencarian <i>neighbor</i> yang di-generate dari <i>current state</i> kubus dan <i>neighbor</i> memiliki nilai mendekati <i>global maximum</i> atau mendekati 0. Pada iterasi pencarian <i>best neighbor</i> ini, diberikan maksimal iterasi untuk memastikan bahwa eksplorasi <i>neighbor</i> -nya dilakukan secara lebih luas, namun tetap dengan batasan untuk mencegah <i>infinite loop</i> dan/atau <i>run time</i> yang terlalu lama.
<b>Output</b>	best_neighbor: MagicCube, best_neighbor_value: int
<b>Source Code</b>	<pre>def searchbestNeighbor(self, current_state):     best_neighbor_value = 999999     best_neighbor = None      for i in range(100):         neighbor = NeighborState(current_state).generate_neighbor() #generate neighbor         neighbor_value = ObjectiveFunction(neighbor).calculate() #check neighbor value</pre>

	<pre>         if neighbor_value &lt; best_neighbor_value:             best_neighbor_value = neighbor_value             best_neighbor = neighbor      return best_neighbor, best_neighbor_value </pre>
--	---

<b>Nama Fungsi</b>	evaluateNeighbor(self, initial_state)
<b>Deskripsi</b>	Fungsi ini merepresentasikan proses <i>hill-climbing</i> . Fungsi ini akan melakukan perbandingan antara nilai <i>neighbor</i> dengan nilai <i>current state</i> . Apabila <i>neighbor</i> memiliki nilai yang lebih baik dibandingkan nilai <i>current state</i> , dalam kasus ini nilai yang lebih baik adalah nilai yang mendekati 0, maka <i>neighbor</i> akan menggantikan <i>current state</i> saat ini. Iterasi pemeriksaan <i>neighbor</i> ini dilakukan hingga ditemukan <i>neighbor</i> yang nilainya sama dengan atau tidak lebih baik dari nilai <i>current state</i> .
<b>Output</b>	current_state: MagicCube, current_value: int, iterations: int
<b>Source Code</b>	<pre> def evaluateNeighbor(self, initial_state):     current_state = initial_state     current_value = ObjectiveFunction(current_state).calculate()     iterations = 0     while True:         best_neighbor, best_neighbor_value = self.searchbestNeighbor(current_state)          #comparing         if best_neighbor_value &lt; </pre>

	<pre> current_value:     current_state = best_neighbor     current_value = best_neighbor_value     iterations += 1     else:         break  self.objective_values.append(current_value)     return current_state, current_value, iterations </pre>
--	--

<b>Nama Fungsi</b>	randomRestart(self)
<b>Deskripsi</b>	<p>Prosedur ini akan menjalankan pencarian dengan konsep <i>random restart hill-climbing</i>. Prosedur ini dimulai dengan memanggil fungsi <i>evaluateNeighbor</i>. Setelah proses evaluasi neighbor (hill-climbing) pertama selesai, dilakukan penyimpanan <i>best state</i> dan <i>best value</i> sementara. Hill-climbing selanjutnya akan dimulai kembali dengan men-<i>generate initial state</i> baru secara acak. Setelah kembali menemukan hasil <i>state</i> dan <i>value</i> yang baru, dibandingkan dengan <i>best state</i> dan <i>best value</i> yang sebelumnya telah disimpan. Apabila hasil yang baru memiliki nilai lebih baik dari yang disimpan pada <i>hill-climbing</i> sebelumnya, hasil yang baru ini akan menjadi <i>best state</i> dan <i>best value</i> sementara. Proses ini akan terus berulang hingga mencapai maksimal pengulangan (<i>restart</i>).</p>
<b>Output</b>	-
<b>Source Code</b>	<pre> def randomRestart(self):     iteration_per_start = []     final_value_per_start = []      #hill climb pertama </pre>

```

        initial_state = MagicCube()
        print(f"\nState Awal:")
        initial_state.display()

        total_iterations = 0

        start_time = time.time()

        final_state, final_value, iterations =
self.evaluateNeighbor(initial_state)
        iteration_per_start.append(iterations)
        final_value_per_start.append(final_value)
        total_iterations += iterations

        if final_value < self.best_value:
            self.best_value = final_value
            self.best_state = final_state

        for restart in range(self.max_restarts):
#restart
            initial_state = MagicCube()

            final_state, final_value, iterations
= self.evaluateNeighbor(initial_state)
            total_iterations += iterations

iteration_per_start.append(iterations)

final_value_per_start.append(final_value)

            if final_value < self.best_value:
                self.best_value = final_value
                self.best_state = final_state

```

```

        execute_time = time.time() - start_time

        #print information
        for step in
range(len(iteration_per_start)):
            if step == 0:
                print(f"\nLangkah pertama - Total
Iterations: {iteration_per_start[step]},
Iterations Value: {final_value_per_start[step]}")
            else:
                print(f"Restart ke-{step} - Total
Iterations: {iteration_per_start[step]},
Iterations Value: {final_value_per_start[step]}")

        print(f"\nState Akhir:")
        self.best_state.display()
        print(f"\nNilai objective akhir:
{self.best_value}")
        print(f"Waktu yang dibutuhkan:
{execute_time:.2f} detik\n")

        self.show_plot(

objective_values=self.objective_values,

max_restarts=len(self.objective_values),
                title=f'Perkembangan Nilai
Objective Function\nMaksimal Restart:
{self.max_restarts}\nWaktu: {execute_time:.2f}
detik'
        )

```



### 2.2.5. Algoritma Stochastic Hill-climbing

*Stochastic hill-climbing* merupakan algoritma yang akan memilih *neighbor state* secara acak. Kemudian, node tersebut dibandingkan dengan *initial state* dan dibandingkan dengan setiap *neighbor*. Jika lebih mendekati dengan solusi, *neighbor* tersebut menjadi *current state*. Namun, jika tidak lebih mendekati solusi, akan memeriksa *node* lain secara acak.

Berikut merupakan *source code* dari implementasi algoritma *stochastic hill-climbing*

```
from cube.magic_cube import MagicCube
from cube.objective_function import ObjectiveFunction
from cube.neighbor_state import NeighborState
import matplotlib.pyplot as plt
import time

class StochasticHillClimbing:
    def __init__(self, max_iteration):
        self.current_state = MagicCube()
        self.current_value =
ObjectiveFunction(self.current_state).calculate()
        self.max_iteration = max_iteration
        self.objective_values = []

    def searchbestNeighbor(self):
        best_neighbor_value = 999999
        best_neighbor = None

        for _ in range (self.max_iteration):
            neighbor =
NeighborState(self.current_state).generate_neighbor()
            neighbor_value = ObjectiveFunction(neighbor).calculate()

            if neighbor_value < best_neighbor_value:
```

```

        best_neighbor = neighbor
        best_neighbor_value = neighbor_value

    return best_neighbor

def evaluateNeighbor(self):
    print("\nState Awal:")
    self.current_state.display()
    print(f"Nilai Initial Objective Function: {self.current_value}")
    start_time = time.time()

    for _ in range(self.max_iteration):
        best_neighbor = self.searchbestNeighbor()
        best_neighbor_value =
ObjectiveFunction(best_neighbor).calculate()

        if best_neighbor_value < self.current_value:
            self.current_state = best_neighbor
            self.current_value = best_neighbor_value
        else:
            break

    self.objective_values.append(self.current_value)

    end_time = time.time()
    total_duration = end_time - start_time

    print(f"State Akhir: ")
    self.current_state.display()
    print(f"Nilai Final Objective Function: {self.current_value}")

    print(f"Jumlah Iterasi: {self.iterations}")
    print(f"Total search duration: {total_duration:.2f} seconds")

```

```

        StochasticHillClimbing.plot_multiple_runs(
            [(self.objective_values, f'Percobaan dengan
{self.max_iteration} iterasi', total_duration)],
max_iteration=self.max_iteration, title=f'Perbandingan
objective function terhadap banyak iterasi yang telah dilewati
menggunakan Stochastic Hill-Climbing'
        )

    @staticmethod
    def plot_multiple_runs(results, max_iteration,
title="Perkembangan Nilai Objective Function"):
        iteration = range(max_iteration)
        num_runs = len(results)

        fig, axes = plt.subplots(1, num_runs, figsize=(18, 6),
sharey=True)

        if num_runs == 1:
            axes = [axes]

        fig.suptitle(title)

        for idx, (objective_values, run_label, total_duration)
in enumerate(results):
            ax = axes[idx]
            ax.plot(range(len(objective_values)),
objective_values, label='Nilai Objective', color='blue')
            ax.set_title(run_label)
            ax.set_xlabel('Iterasi')
            if idx == 0:
                ax.set_ylabel('Nilai Objective Function')
            ax.grid(True, linestyle='--', linewidth=0.5)

```

```

ax.legend()

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

```

<b>Nama Fungsi</b>	searchbestNeighbor(self)
<b>Deskripsi</b>	Fungsi ini digunakan untuk melakukan pencarian <i>neighbor</i> dengan <i>value</i> terbaik berdasarkan <i>objective value</i> . Fungsi ini akan melakukan pencarian terhadap beberapa <i>neighbor</i> secara sekaligus. Jika <i>neighbor</i> tersebut memiliki nilai yang lebih rendah dari <i>current_state</i> , maka <i>neighbor</i> tersebut menjadi <i>best_neighbor</i> .
<b>Output</b>	best_neighbor: MagicCube
<b>Source Code</b>	<pre> def searchbestNeighbor(self):     best_neighbor_value = 999999     best_neighbor = None      for _ in range (self.max_iteration):         neighbor = NeighborState(self.current_state).generate_neighb or()         neighbor_value = ObjectiveFunction(neighbor).calculate()          if neighbor_value &lt; best_neighbor_value:             best_neighbor = neighbor             best_neighbor_value = neighbor_value      return best_neighbor </pre>

<b>Nama Prosedur</b>	evaluateNeighbor(self)
----------------------	------------------------

<b>Deskripsi</b>	Prosedur ini akan menjalankan algoritma <i>hill-climbing with sideways move</i> . Prosedur ini akan melakukan pencarian <i>neighbor</i> terbaik menggunakan <i>searchbestNeighbor()</i> sebanyak jumlah <i>max_iteration</i> yang diinput. Jika nilai <i>best_neighbor</i> lebih rendah, <i>neighbor</i> tersebut akan menjadi <i>current_state</i> baru.
<b>Output</b>	
<b>Source Code</b>	<pre> def evaluateNeighbor(self):     print("\nState Awal:")     self.current_state.display()     print(f"Nilai Initial Objective Function: {self.current_value}")     start_time = time.time()      for _ in range(self.max_iteration):         best_neighbor = self.searchbestNeighbor()         best_neighbor_value = ObjectiveFunction(best_neighbor).calculate()          if best_neighbor_value &lt; self.current_value:             self.current_state = best_neighbor             self.current_value = best_neighbor_value         else:             break      self.objective_values.append(self.current_value)      end_time = time.time()     total_duration = end_time - start_time     print(f"State Akhir: ")     self.current_state.display()     print(f"Nilai Final Objective Function: {self.current_value}") </pre>

```

print(f"Jumlah Iterasi: {self.iterations}")
print(f"Total search duration:
{total_duration:.2f} seconds")

results = []
StochasticHillClimbing.plot_multiple_runs(
    [(self.objective_values, f'Percobaan dengan
{self.max_iteration} iterasi', total_duration)],
    max_iteration=self.max_iteration,
    title=f'Perbandingan objective function
terhadap banyak iterasi yang telah dilewati
menggunakan Stochastic Hill-Climbing'
)

```

#### 2.2.6. Algoritma *Simulated Annealing*

*Simulated annealing* merupakan algoritma yang memungkinkan untuk mencoba kemungkinan solusi terburuk dalam menghindari *local optima*. Probabilitas menggunakan solusi terburuk tersebut akan berkurang seiring bertambahnya percobaan solusi.

Berikut merupakan *source code* dari implementasi algoritma *simulated annealing*

```

import math
import random
import time
import matplotlib.pyplot as plt
from cube.magic_cube import MagicCube
from cube.objective_function import ObjectiveFunction
from cube.neighbor_state import NeighborState

```

```

class SimulatedAnnealing:
    def __init__(self, starting_tem, cooling_rate,
minimum_tem):
        self.starting_tem = starting_tem
        self.cooling_rate = cooling_rate
        self.minimum_tem = minimum_tem
        self.magic_cube = MagicCube()

    def initial_state(self):
        # Display state awal kubus
        print("State Awal: ")
        self.magic_cube.display()
        awal_score =
ObjectiveFunction(self.magic_cube).calculate()
        print("Nilai Initial Objective Function: ",
awal_score)

    def accept_neighbor(self, current_score, neighbor_score,
tem):
        # Penerimaan neighbor
        if neighbor_score < current_score:
            return True, 1.0 # Probabilitas penuh jika lebih
baik
            probability = math.exp((current_score -
neighbor_score) / tem)
            return random.random() < probability, probability

    def tracker_data(self, score_seluruh, prob_seluruh,
current_score, tem, probability):
        # Update data tracker untuk plotting
        score_seluruh.append(current_score)
        # tem_seluruh.append(tem)
        prob_seluruh.append(probability)

```

```

def simulatedannealing(self):
    neighbor_generator = NeighborState(self.magic_cube)
    objective_function =
ObjectiveFunction(self.magic_cube)

    # Initial state kubus
    current_state = self.magic_cube
    current_score = objective_function.calculate()
    tem = self.starting_tem

    # Display data initial state
    self.initial_state()

    # Tracker data
    score_seluruh = [current_score]
    tem_seluruh = [tem]
    prob_seluruh = []
    iterations = 0
    stuck_count = 0

    start_time = time.time()

    while tem > self.minimum_tem:
        # Pemanggilan algoritma neighbor
        neighbor = neighbor_generator.generate_neighbor()

        # Penyimpan nilai neighbor
        objective_function.magic_cube = neighbor
        neighbor_score = objective_function.calculate()

        # Pemanggilan algoritma penerimaan neighbor
        accepted, probability =

```



```

self.accept_neighbor(current_score, neighbor_score, tem)
    if accepted:
        current_state = neighbor
        current_score = neighbor_score
        stuck_count = 0
        iterations += 1
    else:
        stuck_count += 1

    # Update tracker data
    self.tracker_data(score_seluruh, prob_seluruh,
current_score, tem, probability)

    # Update state cube
    self.magic_cube.cube = current_state.cube

    # Cooling system dan penghitungan iterasi
    tem *= self.cooling_rate

    # Penghitungan durasi
    end_time = time.time()
    duration = end_time - start_time

    # Pemanggilan algoritma print
    self.final_state(current_score, iterations, duration,
stuck_count)
    self.plot_results(score_seluruh, prob_seluruh)

    return self.magic_cube.cube

    def final_state(self, current_score, iterations, duration,
stuck_count):
        # Print data final state cube

```

```

        print("\nState Akhir: ")
        print(self.magic_cube.cube)
        print("Nilai Final Objective Function: ",
current_score)
        print("Total Iterasi: ", iterations)
        print(f"Durasi: {duration:.2f}")
        print("Jumlah Stuck: ", stuck_count)

    def plot_results(self, scores_seluruh, prob_seluruh):
        plt.figure(figsize=(12, 6))

        # Plot objective function terhadap iterations
        plt.subplot(1, 2, 1)
        plt.plot(scores_seluruh, label='Objective Function')
        plt.xlabel('Iterations')
        plt.ylabel('Objective Function')
        plt.title('Objective Function terhadap Iterasi')
        plt.legend()

        # Plot probability terhadap iterations
        plt.subplot(1, 2, 2)
        plt.plot(prob_seluruh, label='Probability',
color='green')
        plt.xlabel('Iterations')
        plt.ylabel('Acceptance Probability')
        plt.title('Probability terhadap Iterasi')
        plt.legend()

        plt.tight_layout()
        plt.show()

```

**Nama  
Prosedur**

initial\_state(self)

<b>Deskripsi</b>	Prosedur ini digunakan untuk melakukan printing state awal <i>magic cube</i> yang terdiri dari tampilan <i>cube</i> itu sendiri lalu nilai <i>objective function</i> dari state awal itu sendiri.
<b>Output</b>	Menampilkan state awal <i>magic cube</i> dan informasinya
<b>Source Code</b>	<pre>def initial_state(self):     # Display state awal kubus     print("State Awal: ")     self.magic_cube.display()     awal_score = ObjectiveFunction(self.magic_cube).calculate()     print("Nilai Initial Objective Function: ",     awal_score)</pre>

<b>Nama Fungsi</b>	accept_neighbor(self, current_score, neighbor_score, tem)
<b>Deskripsi</b>	Fungsi ini digunakan untuk melakukan pengecekan apakah <i>neighbor</i> akan diterima atau tidak, terdapat juga variabel probability untuk membantu algoritma mengambil keputusan penerimaan <i>neighbor</i> . Fungsi ini akan mengembalikan boolean berdasarkan apakah <i>neighbor</i> diterima atau tidak
<b>Output</b>	Boolean
<b>Source Code</b>	<pre>def accept_neighbor(self, current_score, neighbor_score, tem):     # Penerimaan neighbor     if neighbor_score &lt; current_score:         return True, 1.0 # Probabilitas penuh jika lebih baik     probability = math.exp((current_score - neighbor_score) / tem)     return random.random() &lt; probability,</pre>

	probability
--	-------------

<b>Nama Fungsi</b>	<code>simulatedannealing(self)</code>
<b>Deskripsi</b>	<p>Fungsi ini akan menjalankan algoritma <i>simulated annealing</i>. Fungsi ini akan melakukan pencarian <i>neighbor</i> dengan menggunakan fungsi <code>NeighborState()</code> lalu melakukan pengecekan penerimaan <i>neighbor</i> tersebut dengan fungsi <code>accept_neighbor()</code>. Ketika <i>neighbor</i> memiliki nilai <i>objective function</i> yang lebih baik maka <i>neighbor</i> tersebut akan dijadikan <i>state magic cube</i> selanjutnya, namun jika tidak maka algoritma akan mempertimbangkan untuk menerima <i>state</i> tersebut dengan menggunakan perbandingan angka random dengan rumus <math>P = e^{\frac{-\Delta E}{T}}</math>. Setelah itu algoritma akan terus berjalan hingga mencapai temperatur minimal yang dimasukkan. Banyaknya iterasi dan lamanya program berjalan akan bergantung pada masukan cooling rate, temperatur awal, dan temperatur minimal yang dapat dimodifikasi saat program dijalankan.</p>
<b>Output</b>	magic_cube: MagicCube
<b>Source Code</b>	<pre>def simulatedannealing(self):     neighbor_generator = NeighborState(self.magic_cube)     objective_function = ObjectiveFunction(self.magic_cube)      # Initial state kubus     current_state = self.magic_cube     current_score = objective_function.calculate()     tem = self.starting_tem</pre>

```

# Display data initial state
self.initial_state()

# Tracker data
score_seluruh = [current_score]
tem_seluruh = [tem]
prob_seluruh = []
iterations = 0
stuck_count = 0

start_time = time.time()

while tem > self.minimum_tem:
    # Pemanggilan algoritma neighbor
    neighbor =
neighbor_generator.generate_neighbor()

    # Penyimpan nilai neighbor
    objective_function.magic_cube = neighbor
    neighbor_score =
objective_function.calculate()

    # Pemanggilan algoritma penerimaan
neighbor
    accepted, probability =
self.accept_neighbor(current_score,
neighbor_score, tem)
    if accepted:
        current_state = neighbor
        current_score = neighbor_score
        stuck_count = 0
        iterations += 1

```

	<pre> else:     stuck_count += 1      # Update tracker data     self.tracker_data(score_seluruh, prob_seluruh, current_score, tem, probability)      # Update state cube     self.magic_cube.cube = current_state.cube      # Cooling system dan penghitungan iterasi     tem *= self.cooling_rate  # Penghitungan durasi end_time = time.time() duration = end_time - start_time  # Pemanggilan algoritma print self.final_state(current_score, iterations, duration, stuck_count) self.plot_results(score_seluruh, prob_seluruh)  return self.magic_cube.cube </pre>
--	--

<b>Nama Prosedur</b>	final_state(self, current_score, iterations, duration, stuck_count)
<b>Deskripsi</b>	Prosedur ini digunakan untuk melakukan printing state final <i>magic cube</i> yang terdiri dari tampilan <i>cube</i> itu sendiri lalu nilai <i>objective function</i> , total iterasi, durasi, dan jumlah stuck pada local optima.
<b>Output</b>	Menampilkan state akhir <i>magic cube</i> dan informasinya

<b>Source Code</b>	<pre>def final_state(self, current_score, iterations, duration, stuck_count):     # Print data final state cube     print("\nState Akhir: ")     print(self.magic_cube.cube)     print("Nilai Final Objective Function: ", current_score)     print("Total Iterasi: ", iterations)     print(f"Durasi: {duration:.2f}")     print("Jumlah Stuck: ", stuck_count)</pre>
--------------------	--

<b>Nama Prosedur</b>	plot_results(self, scores_seluruh, prob_seluruh)
<b>Deskripsi</b>	<p>Prosedur ini digunakan untuk melakukan printing nilai plot yang dibutuhkan, pada algoritma ini terdapat 2 plot yang akan di print yaitu nilai objective function terhadap banyak iterasi yang telah dilewati dan <math>e^{\frac{\Delta E}{T}}</math> terhadap banyak iterasi yang telah dilewati.</p>
<b>Output</b>	Menampilkan nilai plot <i>magic cube</i>
<b>Source Code</b>	<pre>def plot_results(self, scores_seluruh, prob_seluruh):     plt.figure(figsize=(12, 6))      # Plot objective function terhadap iterations     plt.subplot(1, 2, 1)     plt.plot(scores_seluruh, label='Objective Function')     plt.xlabel('Iterations')     plt.ylabel('Objective Function')     plt.title('Objective Function terhadap Iterasi')     plt.legend()</pre>

```

# Plot probability terhadap iterations
plt.subplot(1, 2, 2)
plt.plot(prob_seluruh, label='Probability',
color='green')
plt.xlabel('Iterations')
plt.ylabel('Acceptance Probability')
plt.title('Probability terhadap Iterasi')
plt.legend()

plt.tight_layout()
plt.show()

```

### 2.2.7. Algoritma Genetic

*Genetic algorithm* merupakan algoritma *local search* yang menggunakan populasi dari kemungkinan solusi dari permasalahan. Algoritma ini dipengaruhi oleh beberapa komponen, seperti *population*, *mutation rate*, *parent*, dan keturunan (*child*). *Population* adalah kandidat-kandidat solusi yang mungkin. *Parent* adalah individu yang dipilih dari *population* berdasarkan metode seleksi yang kemudian dilakukan *crossover* menjadi *child*. Setelah *child* dihasilkan oleh dua *parent* melalui *crossover*, akan dilakukan *mutation* dengan *mutation rate* yang telah ditentukan. Selanjutnya, kumpulan *child* tadi akan menggantikan populasi sebelumnya (generasi *parent* dari *child*).

Berikut merupakan *source code* dari implementasi algoritma *genetic*

```

import matplotlib.pyplot as plt
import numpy as np
import random
import time
from cube.magic_cube import MagicCube
from cube.objective_function import ObjectiveFunction

```



```

from cube.neighbor_state import NeighborState

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 = [MagicCube() for _ in
range(population_size)]
        self.fitness_scores = []

    def get_num_elite(self, iteration):
        ''' Mendapatkan jumlah individu elite di populasi '''
        if (self.population_size <= 5):
            return 0
        elif (self.population_size <= 10):
            return 1
        elif self.population_size <= 20:
            if (iteration < self.max_iteration * 0.4):
                return 1
            else:
                return 2
        else:
            if (iteration < self.max_iteration * 0.4):
                return 1
            elif (iteration < self.max_iteration * 0.8):
                return 2
            else:
                return 3

    def evaluate_population(self):
        ''' Prosedur untuk mengevaluasi seluruh populasi dan

```

```

di-sorting menaik '''
    self.fitness_scores = []

    for cube in self.population:
        fitness_score = ObjectiveFunction(cube).calculate()
        self.fitness_scores.append((cube, fitness_score))

    self.fitness_scores.sort(key=lambda x: x[1])

    def tournament_selection(self, tournament_size):
        ''' Konsep seleksi menggunakan tournament selection, yaitu
        mengambil sampel sebanyak tournament size secara random untuk
        diambil 2 individu terbaik sebagai parent'''
        participants = random.sample(self.fitness_scores,
        tournament_size)
        # Mengurutkan peserta berdasarkan fitness secara ascending
        sorted_participants = sorted(participants, key=lambda x:
        x[1])
        # Mengembalikan dua individu terbaik
        return sorted_participants[0][0],
        sorted_participants[1][0]

    def selection(self):
        ''' Seleksi individu untuk dijadikan parent dan memastikan
        parent bukanlah individu yang sama '''
        tournament_size = 2 if self.population_size < 20 else 5
        parent1, parent2 =
        self.tournament_selection(tournament_size=tournament_size)
        while parent1 == parent2:
            parent2 = self.tournament_selection(tournament_size=5)

        return parent1, parent2

```

```

def partial_layer_crossover(self, parent1, parent2):
    ''' Teknik crossover yang digunakan adalah menggabungkan
    konsep layer preservation dengan unique element filling '''
    # Membuat matriks kosong untuk child
    child1_cube = np.zeros_like(parent1.cube)
    child2_cube = np.zeros_like(parent2.cube)

    # Isi 3 layer pertama child 1 dengan elemen dari parent 1
    child1_cube[:3] = parent1.cube[:3]

    # Isi 3 layer terakhir child 2 dengan elemen dari parent 2
    child2_cube[-3:] = parent2.cube[-3:]

    # Buat set elemen yang sudah digunakan di child 1 dan
    child 2 serta menghilangkan bilangan nol di dalam set
    used_elements_1 = set(child1_cube.flatten()) - {0}
    used_elements_2 = set(child2_cube.flatten()) - {0}

    # Isi layer yang tersisa di child 1 dengan elemen unik
    dari parent 2
    for i in range(parent2.size):
        if (len(used_elements_1) == parent1.size**3):
            break
        for j in range(parent2.size):
            if (len(used_elements_1) == parent1.size**3):
                break
            for k in range(parent2.size):
                if (len(used_elements_1) == parent1.size**3):
                    break
                element = parent2.cube[i, j, k]
                if element not in used_elements_1:
                    for x in range(3, parent1.size):
                        for y in range(parent1.size):

```

```

        for z in range(parent1.size):
            if child1_cube[x, y, z] == 0:
                child1_cube[x, y, z] = element
                used_elements_1.add(element)
                break
            if element in used_elements_1:
                break
        if element in used_elements_1:
            break

# Isi layer yang tersisa di child 2 dengan elemen unik
dari parent 1
for i in range(parent1.size):
    if (len(used_elements_2) == parent2.size**3):
        break
    for j in range(parent1.size):
        if (len(used_elements_2) == parent2.size**3):
            break
        for k in range(parent1.size):
            if (len(used_elements_2) == parent2.size**3):
                break
            element = parent1.cube[i, j, k]
            if element not in used_elements_2:
                for x in range(parent2.size - 3):
                    for y in range(parent2.size):
                        for z in range(parent2.size):
                            if child2_cube[x, y, z] == 0:
                                child2_cube[x, y, z] = element
                                used_elements_2.add(element)
                                break
                        if element in used_elements_2:
                            break
                    if element in used_elements_2:
                        break
                if element in used_elements_2:
                    break

```

```

        break

    # Membuat objek MagicCube baru untuk child
    child1 = MagicCube(size=parent1.size)
    child2 = MagicCube(size=parent2.size)
    child1.cube = child1_cube
    child2.cube = child2_cube
    return child1, child2

def swap_mutation(self, magic_cube):
    ''' Swap mutation dilakukan dengan cara yang sama seperti
    mencari neighbor '''
    neighbor_state = NeighborState(magic_cube)
    return neighbor_state.generate_neighbor()

def scramble_mutation(self, magic_cube):
    ''' Scramble mutation dilakukan dengan cara mencari index
    layer cube secara acak lalu bilangan-bilangan pada layer
    tersebut akan dishuffle '''
    layer_index = random.randint(0, magic_cube.size - 1)
    flat_layer = magic_cube.cube[layer_index].flatten()
    np.random.shuffle(flat_layer)
    magic_cube.cube[layer_index] =
flat_layer.reshape(magic_cube.size, magic_cube.size)
    return magic_cube

def adaptive_mutation(self, magic_cube, iteration,
max_iteration):
    ''' Adaptive mutation akan memilih antara scramble mutation
    atau swap mutation berdasarkan progress iteration '''
    iteration_progress = iteration / max_iteration
    # Atur mutation rate adaptif berdasarkan progres generasi
    if iteration_progress < 0.5:

```

```

        # Di awal hingga pertengahan, mutation rate akan
        ditingkatkan hingga 50% dengan tujuan eksplorasi yang kuat
        if (self.mutation_rate < 0.5):
            self.mutation_rate = self.mutation_rate * 1.1
        else: # 0.5 <= iteration_progress <= 1
            # Di pertengahan hingga akhir, turunkan mutation rate
            untuk mengeksploitasi solusi terbaik
            if(self.mutation_rate > 0.1):
                self.mutation_rate = self.mutation_rate * 0.99

        # Terdapat probabilitas mutation rate yang jika tidak
        dipenuhi maka tidak akan dilakukan mutation
        if (random.random() >= self.mutation_rate):
            return magic_cube

        # Untuk 30% dari total iterasi, akan dilakukan scramble
        mutation
        if (iteration < (max_iteration * 0.3)):
            mutated_cube = self.scramble_mutation(magic_cube)
        else: # 70% sisanya akan dilakukan swap mutation
            mutated_cube = self.swap_mutation(magic_cube)

        # Memastikan bahwa jika ada yang duplikat fitnessnya
        maka akan dilakukan swap mutation hingga semua fitness pada
        populasi bersifat unik
        while self.is_duplicate_fitness(mutated_cube):
            mutated_cube = self.swap_mutation(mutated_cube)

        return mutated_cube

    def is_duplicate_fitness(self, cube):
        ''' Fungsi untuk memeriksa apakah ada duplikat fitness
        pada populasi '''
        # Menghitung fitness score dari individu yang di-check

```

```

        fitness_score = ObjectiveFunction(cube).calculate()

        # Memeriksa apakah fitness score ini sudah ada di populasi
        for _, existing_fitness in self.fitness_scores:
            if existing_fitness == fitness_score:
                return True
        return False

    @staticmethod
    def get_valid_input(prompt, min_value=1,
value_type="integer"):
        ''' Fungsi untuk memastikan input dari pengguna itu harus
        berupa integer dan harus memiliki nilai lebih dari sama dengan
        min_value '''
        while True:
            try:
                user_input = input(prompt).split()

                if (value_type == "integer"):
                    if not all(x.lstrip('-').isdigit() for x in
user_input):
                        raise TypeError("Semua input harus berupa angka
bulat (integer).")

                    values = [int(x) for x in user_input]

                    if any(value < min_value for value in values):
                        raise ValueError(f"Nilai minimal harus {min_value}
atau lebih besar.")

                    return values

            except (ValueError, TypeError) as e:

```

```

        print(f"Error: {e}\nSilakan masukkan input yang
valid.")

    @staticmethod
    def run(population_size, max_iteration, mutation_rate=0.3):
        ''' Fungsi untuk menjalankan Genetic Algorithm, GA berikut
menggunakan konsep elitisme, yaitu menyimpan elemen terbaik
pada suatu iterasi untuk diletakkan pada generasi
berikutnya'''
        ga = GeneticAlgorithm(population_size, max_iteration,
mutation_rate)
        # Memulai timer
        start_time = time.time()
        # Menyiapkan array untuk nilai maksimum dan rata-rata
score per iterasi
        max_scores_per_iteration = []
        avg_scores_per_iteration = []
        # Mengevaluasi populasi
        ga.evaluate_population()
        # Menampilkan state awal populasi (best fitness)
        print("\nState awal populasi (menampilkan individu
pertama):")
        ga.population[0].display()

        for iteration in range(max_iteration):
            # Populasi baru dikosongkan ulang setiap iterasi baru
            new_population = []
            # Mendapatkan jumlah elemen elite
            num_elite = ga.get_num_elite(iteration)
            # Memasukkan elemen elite ke dalam populasi baru
            new_population = [ga.fitness_scores[i][0] for i in
range(num_elite)]

```



```

while len(new_population) < ga.population_size:
    # Melakukan seleksi, crossover, mutation dan
    dimasukkan ke dalam array new population
    parent1, parent2 = ga.selection()
    child1, child2 = ga.partial_layer_crossover(parent1,
parent2)
    child1 = ga.adaptive_mutation(child1, iteration,
ga.max_iteration)
    child2 = ga.adaptive_mutation(child2, iteration,
ga.max_iteration)
    new_population.extend([child1, child2])

    # Memotong kalo kelebihan populasi (karena jumlah
    populasi yang harus dimasukkan ke new population itu ganjil)
    if len(new_population) > ga.population_size:
        new_population = new_population[:ga.population_size]

    # Mengganti populasi lama dengan populasi baru
    ga.population = new_population

    # Evaluasi populasi baru
    ga.evaluate_population()

    scores = [score[1] for score in ga.fitness_scores]
    max_score = min(scores) # Nilai minimum dianggap
    terbaik
    avg_score = sum(scores) / len(scores)
    # Menyimpan score maksimum dan rata-rata ke dalam array
    max_scores_per_iteration.append(max_score)
    avg_scores_per_iteration.append(avg_score)

    # Mengakhiri timer
    end_time = time.time()

```

```

# Menghitung durasi
duration = end_time - start_time

# Menampilkan state akhir populasi (best fitness)
print("\nState akhir populasi (menampilkan individu
terbaik):")

ga.fitness_scores[0][0].display()
best_fitness_score = ga.fitness_scores[0][1]

# Menampilkan Objective Function akhir, populasi, iterasi,
dan durasi pencarian
print(f"\nNilai objective function akhir yang dicapai:
{best_fitness_score}")

print(f"Jumlah populasi: {population_size}")
print(f"Banyak iterasi: {max_iteration}")
print(f"Durasi proses pencarian: {duration:.2f} detik")

# Menjalankan fungsi plotting hasil iterasi
GeneticAlgorithm.plot_result(max_scores_per_iteration,
avg_scores_per_iteration, max_iteration, population_size)

@staticmethod
def plot_result(max_scores, avg_scores, max_iteration,
population_size):
    ''' Fungsi untuk melakukan plotting hasil pencarian
    genetik '''
    plt.figure(figsize=(10, 6))
    plt.plot(range(max_iteration), max_scores, label='Nilai
Maksimum', linestyle='--', color='red')
    plt.plot(range(max_iteration), avg_scores, label='Nilai
Rata-rata', color='blue')
    plt.xlabel('Iterasi')
    plt.ylabel('Nilai Objective Function')
    plt.title(f'Perkembangan Nilai Objective
Function\nPopulasi: {population_size} dan Iterasi:
{max_iteration}')

```

```
plt.legend()
plt.grid(True)
plt.show()
```

<b>Nama Fungsi</b>	get_num_elite(self, iteration)
<b>Deskripsi</b>	Fungsi ini digunakan untuk menghitung jumlah elemen individu <i>elite</i> di populasi berdasarkan ukuran populasi dan berdasarkan progres iterasi.
<b>Output</b>	Me-return banyaknya elemen <i>elite</i> dengan tipe <i>integer</i>
<b>Source Code</b>	<pre>def get_num_elite(self, iteration):     ''' Mendapatkan jumlah individu elite di     populasi '''     if (self.population_size &lt;= 5):         return 0     elif (self.population_size &lt;= 10):         return 1     elif self.population_size &lt;= 20:         if (iteration &lt; self.max_iteration * 0.4):             return 1         else:             return 2     else:         if (iteration &lt; self.max_iteration * 0.4):             return 1         elif (iteration &lt; self.max_iteration * 0.8):             return 2         else:             return 3</pre>

<b>Nama Prosedur</b>	evaluate_population(self)
----------------------	---------------------------

<b>Deskripsi</b>	Prosedur ini digunakan untuk mengevaluasi dan mengurutkan populasi secara ascending berdasarkan fitness score.
<b>Output</b>	Mengurutkan list fitness scores yang berisi seluruh fitness score dari populasi
<b>Source Code</b>	<pre>def evaluate_population(self):     ''' Prosedur untuk mengevaluasi seluruh     populasi dan di-sorting menaik '''     self.fitness_scores = []      for cube in self.population:         fitness_score = ObjectiveFunction(cube).calculate()         self.fitness_scores.append((cube, fitness_score))      self.fitness_scores.sort(key=lambda x: x[1])</pre>

<b>Nama Fungsi</b>	tournament_selection(self, tournament_size)
<b>Deskripsi</b>	Fungsi ini digunakan untuk memilih <i>parent</i> dengan metode seleksi turnamen dengan cara mengambil sampel sebanyak <i>tournament size</i> secara random lalu diambil 2 individu terbaik sebagai <i>parent</i> .
<b>Output</b>	Me-return dua individu terpilih dengan tipe class MagicCube
<b>Source Code</b>	<pre>def tournament_selection(self, tournament_size):     ''' Konsep seleksi menggunakan tournament     selection, yaitu mengambil sampel sebanyak     tournament size secara random untuk diambil 2     individu terbaik sebagai parent'''     participants = random.sample(self.fitness_scores, tournament_size)     # Mengurutkan peserta berdasarkan fitness</pre>

	secara ascending sorted_participants = <b>sorted</b> (participants, key=lambda x: x[1]) # Mengembalikan dua individu terbaik <b>return</b> sorted_participants[0][0], sorted_participants[1][0]
--	--

<b>Nama Fungsi</b>	selection(self)
<b>Deskripsi</b>	Fungsi ini digunakan untuk menyeleksi dua individu yang akan dijadikan <i>parent</i> dengan <i>tournament selection</i> dan memastikan pasangan <i>parent</i> tersebut adalah individu yang berbeda.
<b>Output</b>	Me-return dua parent terpilih dengan tipe class MagicCube
<b>Source Code</b>	<pre> def selection(self):     ''' Seleksi individu untuk dijadikan parent     dan memastikan parent bukanlah individu yang sama     '''     tournament_size = 2 if self.population_size &lt; 20 else 5     parent1, parent2 = self.tournament_selection(tournament_size=tournam ent_size)     while parent1 == parent2:         parent2 = self.tournament_selection(tournament_size=5)      return parent1, parent2 </pre>

<b>Nama Fungsi</b>	partial_layer_crossover(self, parent1, parent2)
<b>Deskripsi</b>	Fungsi ini digunakan untuk menyilangkan dua parent agar menghasilkan dua anak baru dengan cara menyimpan beberapa

	layer dari parent lalu mengiterasi beberapa elemen dari parent lain untuk dimasukkan ke dalam suatu anak sambil memastikan setiap bilangan yang ada di cube anak bersifat <i>unique</i> .
<b>Output</b>	Me-return dua child hasil layer crossover secara parsial dengan tipe class MagicCube
<b>Source Code</b>	<pre> def partial_layer_crossover(self, parent1, parent2):     ''' Teknik crossover yang digunakan adalah menggabungkan konsep layer preservation dengan unique element filling '''     # Membuat matriks kosong untuk child     child1_cube = np.zeros_like(parent1.cube)     child2_cube = np.zeros_like(parent2.cube)      # Isi 3 layer pertama child 1 dengan elemen dari parent 1     child1_cube[:3] = parent1.cube[:3]      # Isi 3 layer terakhir child 2 dengan elemen dari parent 2     child2_cube[-3:] = parent2.cube[-3:]      # Buat set elemen yang sudah digunakan di child 1 dan child 2 serta menghilangkan bilangan nol di dalam set     used_elements_1 = set(child1_cube.flatten()) - {0}     used_elements_2 = set(child2_cube.flatten()) - {0}      # Isi layer yang tersisa di child 1 dengan elemen unik dari parent 2     for i in range(parent2.size): </pre>

```

        if (len(used_elements_1) ==
parent1.size**3):
            break
        for j in range(parent2.size):
            if (len(used_elements_1) ==
parent1.size**3):
                break
            for k in range(parent2.size):
                if (len(used_elements_1) ==
parent1.size**3):
                    break
                element = parent2.cube[i, j, k]
                if element not in used_elements_1:
                    for x in range(3, parent1.size):
                        for y in range(parent1.size):
                            for z in range(parent1.size):
                                if child1_cube[x, y, z] == 0:
                                    child1_cube[x, y, z] =
element
                                    used_elements_1.add(element)
                                    break
                                if element in used_elements_1:
                                    break
                                if element in used_elements_1:
                                    break

# Isi layer yang tersisa di child 2 dengan
elemen unik dari parent 1
        for i in range(parent1.size):
            if (len(used_elements_2) ==
parent2.size**3):
                break
            for j in range(parent1.size):

```

```

        if (len(used_elements_2) ==
parent2.size**3):
            break
        for k in range(parent1.size):
            if (len(used_elements_2) ==
parent2.size**3):
                break
            element = parent1.cube[i, j, k]
            if element not in used_elements_2:
                for x in range(parent2.size - 3):
                    for y in range(parent2.size):
                        for z in range(parent2.size):
                            if child2_cube[x, y, z] == 0:

child2_cube[x, y, z] = element

used_elements_2.add(element)
                    break
                if element in used_elements_2:
                    break
            if element in used_elements_2:
                break

# Membuat objek MagicCube baru untuk child
child1 = MagicCube(size=parent1.size)
child2 = MagicCube(size=parent2.size)
child1.cube = child1_cube
child2.cube = child2_cube
return child1, child2

```

<b>Nama Fungsi</b>	swap_mutation(self, magic_cube)
<b>Deskripsi</b>	Fungsi ini digunakan untuk mutasi sebuah <i>magic cube</i> dengan cara



	<i>swap</i> dua titik tertentu secara <i>random</i> .
<b>Output</b>	Me- <i>return magic cube</i> baru yang berasal dari hasil <i>swap</i> 2 titik pada suatu individu
<b>Source Code</b>	<pre>def swap_mutation(self, magic_cube):     ''' Swap mutation dilakukan dengan cara yang sama seperti mencari neighbor '''     neighbor_state = NeighborState(magic_cube)     return neighbor_state.generate_neighbor()</pre>

<b>Nama Fungsi</b>	<code>scramble_mutation(self, magic_cube)</code>
<b>Deskripsi</b>	Fungsi ini digunakan untuk mutasi sebuah <i>magic cube</i> dengan cara mengacak suatu lapisan <i>magic cube</i> tersebut.
<b>Output</b>	Me- <i>return magic cube</i> baru yang berasal dari hasil <i>scramble</i> pada suatu lapisan.
<b>Source Code</b>	<pre>def scramble_mutation(self, magic_cube):     ''' Scramble mutation dilakukan dengan cara mencari index layer cube secara acak lalu bilangan-bilangan pada layer tersebut akan dishuffle '''     layer_index = random.randint(0, magic_cube.size - 1)     flat_layer = magic_cube.cube[layer_index].flatten()     np.random.shuffle(flat_layer)     magic_cube.cube[layer_index] = flat_layer.reshape(magic_cube.size, magic_cube.size)     return magic_cube</pre>

<b>Nama Fungsi</b>	<code>adaptive_mutation(self, magic_cube, iteration, max_iteration)</code>
--------------------	--

<b>Deskripsi</b>	Fungsi ini digunakan untuk mutasi sebuah <i>magic cube</i> secara adaptif dan mengubah mutation rate berdasarkan progres iterasi, serta memastikan bahwa tidak ada nilai <i>fitness</i> yang duplikat dengan cara mutasi lagi.
<b>Output</b>	Me-return <i>magic cube</i> yang sudah dimutasi atau tidak dimutasi akibat probabilitas laju mutasi.
<b>Source Code</b>	<pre> def adaptive_mutation(self, magic_cube, iteration, max_iteration):     ''' Adaptive mutation akan memilih antara     scramble mutation atau swap mutation berdasarkan     progress iteration '''     iteration_progress = iteration / max_iteration     # Atur mutation rate adaptif berdasarkan     progres generasi     if iteration_progress &lt; 0.5:         # Di awal hingga pertengahan, mutation rate         akan ditingkatkan hingga 50% dengan tujuan         eksplorasi yang kuat         if (self.mutation_rate &lt; 0.5):             self.mutation_rate = self.mutation_rate             * 1.1         else: # 0.5 &lt;= iteration_progress &lt;= 1             # Di pertengahan hingga akhir, turunkan             mutation rate untuk mengeksploitasi solusi             terbaik             if(self.mutation_rate &gt; 0.1):                 self.mutation_rate =                 self.mutation_rate * 0.99          # Terdapat probabilitas mutation rate yang         jika tidak dipenuhi maka tidak akan dilakukan         mutation          if (random.random() &gt;= </pre>

	<pre> self.mutation_rate):     return magic_cube     # Untuk 30% dari total iterasi, akan     dilakukan scramble mutation     if (iteration &lt; (max_iteration * 0.3)):         mutated_cube = self.scramble_mutation(magic_cube)     else: # 70% sisanya akan dilakukan swap     mutation         mutated_cube = self.swap_mutation(magic_cube)      # Memastikan bahwa jika ada yang duplikat     fitnessnya maka akan dilakukan swap mutation     hingga semua fitness pada populasi bersifat unik     while self.is_duplicate_fitness(mutated_cube):         mutated_cube = self.swap_mutation(mutated_cube)      return mutated_cube </pre>
--	---

<b>Nama Fungsi</b>	is_duplicate_fitness(self, cube)
<b>Deskripsi</b>	Fungsi ini digunakan untuk memeriksa apakah ada nilai fitness yang duplikat pada suatu populasi.
<b>Output</b>	Me-return <i>boolean</i> bahwa ada atau tidaknya fitness score yang sama.
<b>Source Code</b>	<pre> def is_duplicate_fitness(self, cube):     ''' Fungsi untuk memeriksa apakah ada     duplikat fitness pada populasi '''     # Menghitung fitness score dari individu yang     di-check </pre>

	<pre>         fitness_score = ObjectiveFunction(cube).calculate()          # Memeriksa apakah fitness score ini sudah ada di populasi         for _, existing_fitness in self.fitness_scores:             if existing_fitness == fitness_score:                 return True         return False </pre>
--	---

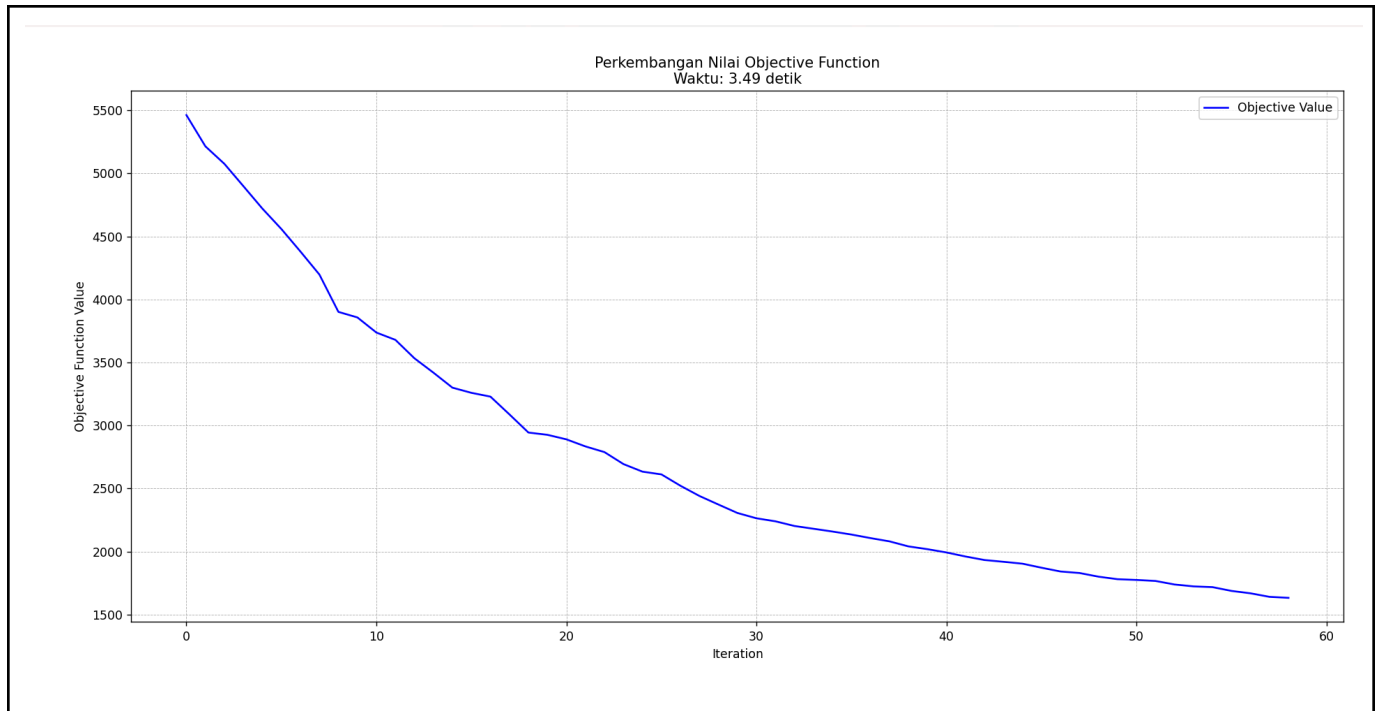
## 2.3. Hasil Eksperimen dan Analisis

### 2.3.1. Algoritma *Steepest Ascent Hill-climbing*

Berikut merupakan hasil eksperimen yang dilakukan sebanyak tiga kali menggunakan algoritma *steepest ascent hill-climbing*.

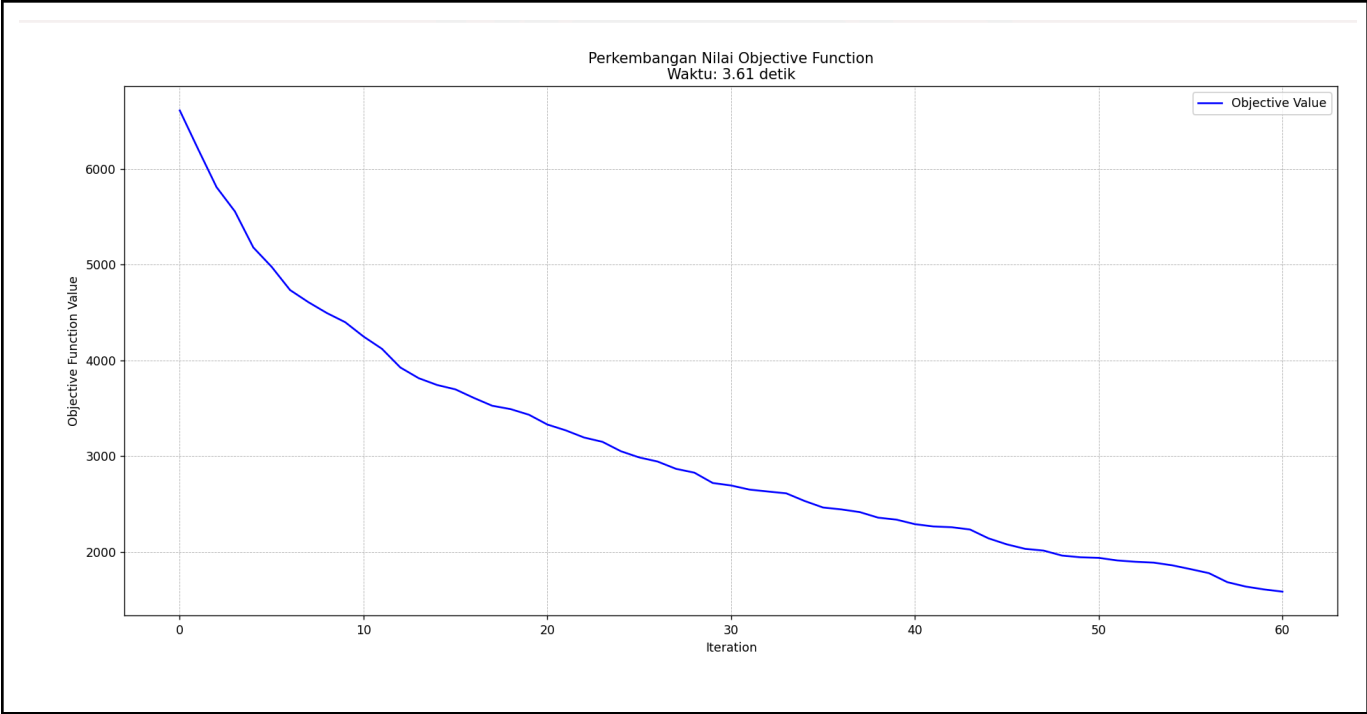
Eksperimen 1			
<b>State awal</b>	<pre> [[[ 60 81  4 118  7]  [ 78 10 79 42 116]  [ 69 46 21 34 72]  [  6 95 19 120 31]  [112 105 53 92 59]]  [[115 29 35  8 39]  [ 13 123 26 98 96]  [ 80  3 61 70 44]  [ 74 11 83  5 76]  [ 54 55 68 107 16]]  [[ 1 14 93 90 17]  [56 122  2 30 63] </pre>	<b>State akhir</b>	<pre> [[[ 53 81 72 95  7]  [ 78 10 79 42 106]  [ 82 108 54 30 47]  [  3 118 19 120 46]  [ 99 29 60 28 88]]  [[115 52 45 12 67]  [ 23 96 26 98 59]  [ 80 22 68 70 86]  [ 74 77 71 14 76]  [ 27 55 112 107 16]]  [[ 1  5 116 103 100]  [123 122  2 34 31] </pre>

	[ 77 100 85 124 71] [ 25 119 114 9 87] [ 36 84 47 32 108]]  [[ 33 106 104 75 97] [ 82 23 91 15 89] [ 51 27 58 64 110] [ 65 125 50 94 40] [ 73 86 28 117 48]]  [[ 67 52 66 45 22] [ 20 12 99 113 101] [102 111 41 18 57] [ 24 121 88 38 103] [ 62 43 109 49 37]]]]		[ 11 17 85 124 83] [ 92 9 114 35 36] [ 87 110 4 21 94]]  [[ 6 66 89 64 105] [ 69 93 102 15 38] [ 51 32 58 75 84] [125 65 50 43 49] [ 73 56 25 117 48]]  [[119 97 13 44 33] [ 20 8 121 113 101] [ 91 111 41 18 57] [ 24 37 61 104 90] [ 62 63 109 40 39]]]]
Nilai <i>objective function</i> yang dicapai			1633
Durasi proses pencarian			3.49 detik
Banyak iterasi hingga proses pencarian berhenti			59
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Eksperimen 2			
<b>State awal</b>	<div>[[[123 40 39 113 25]</div> <div>[ 30 49 67 59 36]</div> <div>[ 54 18 97 26 71]</div> <div>[102 7 80 91 45]</div> <div>[ 27 58 43 2 87]]</div> <div>[[ 55 74 86 95 11]</div> <div>[ 37 115 62 89 114]</div> <div>[ 93 82 47 16 99]</div> <div>[103 53 34 119 106]</div> <div>[104 75 68 88 35]]</div> <div>[[ 44 66 90 22 73]</div> <div>[121 124 6 83 72]</div> <div>[ 64 98 41 105 17]</div> <div>[100 3 21 31 42]</div>	<b>State akhir</b>	<div>[[[ 93 72 1 121 25]</div> <div>[ 63 27 88 92 46]</div> <div>[ 37 41 97 26 110]</div> <div>[102 91 80 5 45]</div> <div>[ 8 100 43 77 87]]</div> <div>[[ 40 74 90 107 2]</div> <div>[ 6 50 62 89 94]</div> <div>[123 24 47 16 99]</div> <div>[ 19 66 34 119 85]</div> <div>[111 75 68 30 35]]</div> <div>[[117 31 18 22 120]</div> <div>[ 44 84 54 9 124]</div> <div>[ 64 61 79 105 17]</div> <div>[ 86 73 60 67 42]</div>

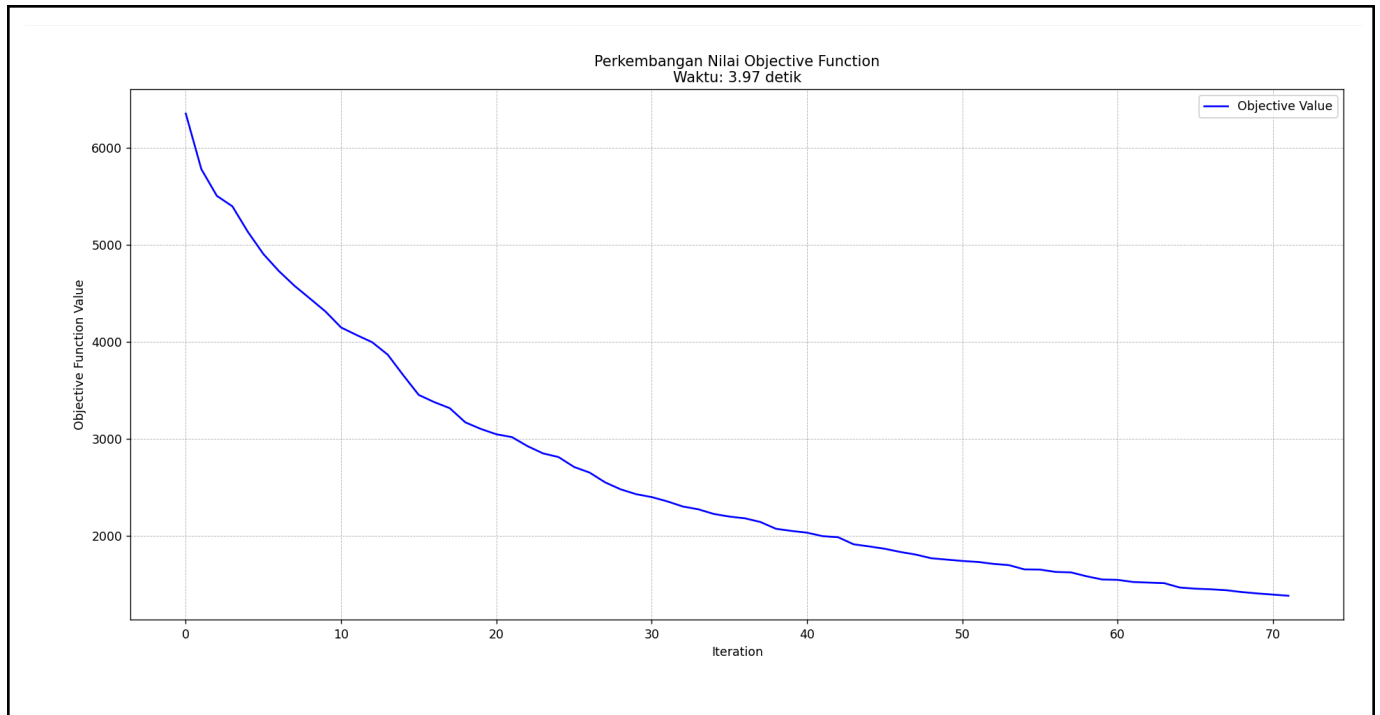
	[ 32 20 1 50 4]]  [[ 46 94 13 60 12] [ 52 111 8 117 79] [ 5 61 65 122 15] [107 69 96 14 109] [108 70 29 63 116]]  [[120 33 9 56 112] [118 48 57 125 23] [ 84 110 38 51 78] [ 85 92 28 101 19] [ 81 76 24 10 77]]]]		[ 32 52 103 115 4]]  [[ 69 114 101 15 12] [106 48 39 113 14] [ 7 98 65 122 21] [ 71 36 96 11 109] [ 83 20 29 58 116]]  [[ 3 33 108 56 112] [118 104 57 13 23] [ 55 95 38 78 51] [ 53 59 28 125 49] [ 81 76 82 10 70]]]]
Nilai <i>objective function</i> yang dicapai			1583
Durasi proses pencarian			3.61 detik
Banyak iterasi hingga proses pencarian berhenti			61
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Eksperimen 3			
State awal	<div>[[[ 39 120 29 117 4]</div> <div>[101 43 15 11 32]</div> <div>[ 16 35 94 38 93]</div> <div>[ 26 9 25 118 19]</div> <div>[ 34 31 96 106 37]]</div> <div><div>[[ 7 91 59 63 55]</div><div>[ 21 44 12 30 83]</div><div>[ 78 119 113 68 28]</div><div>[ 8 121 81 23 110]</div><div>[ 14 56 2 47 10]]</div></div> <div><div>[[ 53 41 116 86 73]</div><div>[ 70 71 5 66 115]</div><div>[ 98 67 24 36 79]</div><div>[108 1 74 111 75]</div></div>	State akhir	<div>[[[ 62 120 37 104 5]</div> <div>[101 43 38 11 123]</div> <div>[ 16 35 124 46 93]</div> <div>[ 89 83 25 50 69]</div> <div>[ 34 31 105 106 29]]</div> <div><div>[[ 90 55 14 122 49]</div><div>[ 67 15 109 30 91]</div><div>[ 76 118 113 4 6]</div><div>[ 8 65 81 40 110]</div><div>[ 72 73 2 119 61]]</div></div> <div><div>[[ 7 39 116 53 112]</div><div>[ 77 108 10 94 19]</div><div>[114 42 52 36 70]</div><div>[ 71 1 58 111 75]</div></div>



	[ 57 95 84 20 54]]  [[ 46 52 125 105 72] [122 92 112 123 76] [ 51 80 109 100 102] [ 87 88 45 60 42] [103 64 90 40 50]]  [[ 69 77 17 27 48] [ 62 107 49 124 114] [ 85 13 3 97 82] [ 65 18 6 89 22] [ 61 33 58 99 104]]]]		[ 57 117 78 20 41]]  [[ 44 21 125 13 47] [ 63 80 56 79 51] [ 18 92 12 100 86] [ 87 99 45 88 54] [103 24 64 23 82]]  [[115 95 17 27 68] [ 9 66 107 102 32] [ 85 28 3 97 96] [ 59 84 121 26 22] [ 48 33 74 60 98]]]]
Nilai <i>objective function</i> yang dicapai			1381
Durasi proses pencarian			3.97 detik
Banyak iterasi hingga proses pencarian berhenti			72
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Melalui ketiga eksperimen yang dilakukan dengan algoritma *steepest ascent hill-climbing* didapatkan hasil *objective function value* yang jauh lebih baik dibandingkan dengan *objective function value* inisiasinya. *Steepest ascent hill-climbing* memiliki kekurangan yaitu cepat untuk terminasi dan memiliki kemungkinan yang besar bahwa akan berhenti di *local maxima*. Hal ini disebabkan karena dalam proses *hill-climb*-nya, *steepest ascent hill-climb* akan menyelesaikan pencarian bertemu dengan *neighbor* yang punya nilai *objective value* yang sama dengan atau tidak lebih baik dari *current state*. Namun, hasil akhir *objective function value* bisa dioptimalkan dengan memperluas eksplorasi pencarian *successor* yang akan dijadikan *neighbor* sehingga mencegah terlalu cepatnya terminasi proses *hill-climbing*-nya.

### 2.3.2. Algoritma *Hill-climbing with Sideways Move*

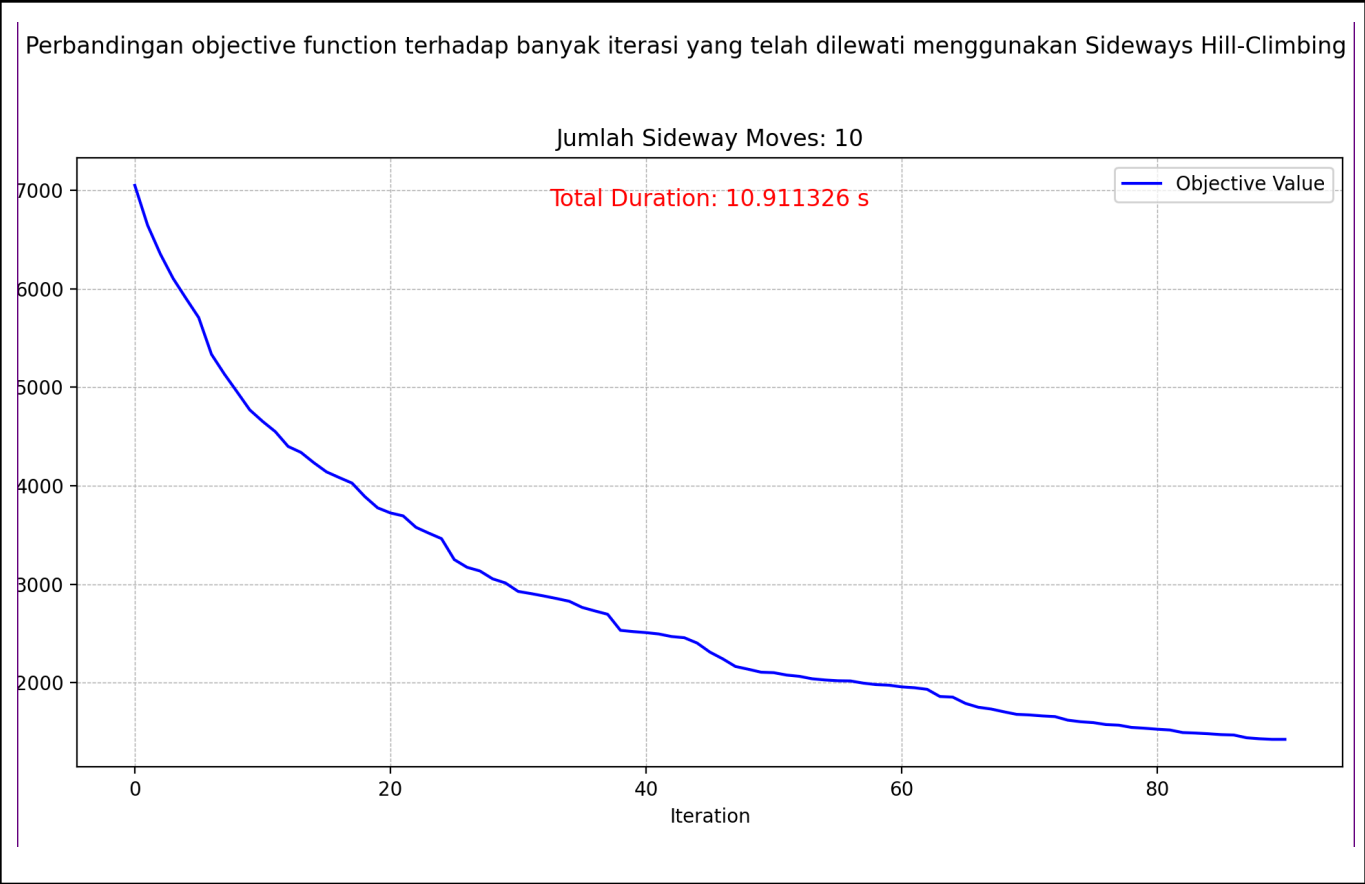
Berikut merupakan hasil eksperimen yang dilakukan sebanyak tiga kali menggunakan algoritma *steepest hill-climbing with sideways move*.

**Eksperimen 1** (*Maximum sideways move* = 5)

<b>State awal</b>	<p>[[[ 55 57 78 24 19] [ 48 56 21 123 18] [114 60 77 117 68] [ 54 51 45 14 62] [110 8 90 93 49]]</p> <p>[[ 80 94 92 100 96] [101 97 26 104 118] [106 30 107 50 65] [ 5 66 53 86 40] [ 73 122 83 102 23]]</p> <p>[[ 42 58 38 7 85] [120 46 52 13 10] [ 17 11 41 74 64] [ 95 124 76 108 103] [ 16 111 116 15 3]]</p> <p>[[109 27 9 79 61] [119 91 89 32 22] [ 75 70 1 25 33] [ 4 115 29 112 99] [ 69 43 84 39 125]]</p> <p>[[105 31 20 59 12] [ 63 44 98 36 47] [ 6 2 82 67 28] [ 35 72 88 71 121] [113 87 34 37 81]]]</p> <p>Nilai <i>Objective</i> Awal: 7050</p>	<b>State akhir</b>	<p>[[[ 61 57 114 70 15] [ 18 115 22 123 48] [ 67 55 77 19 103] [124 51 17 12 105] [ 45 36 82 93 49]]</p> <p>[[ 9 32 83 100 96] [ 85 97 10 24 98] [106 39 119 27 8] [ 20 66 91 64 75] [ 94 73 13 102 29]]</p> <p>[[ 42 69 76 7 122] [109 46 80 72 14] [ 23 87 56 74 86] [121 25 1 110 62] [ 16 111 101 54 33]]</p> <p>[[ 78 52 37 79 60] [ 71 11 89 65 90] [118 125 26 43 3] [ 4 92 88 95 53] [ 58 35 84 30 108]]</p> <p>[[116 113 5 59 21] [ 47 44 112 31 63] [ 2 6 68 120 117] [ 41 99 107 40 28] [104 38 34 50 81]]]</p> <p>Nilai <i>Objective</i> Akhir: 1424</p>
-------------------	---	--------------------	--

Nilai <i>objective function</i> yang dicapai	1424
Durasi proses pencarian	10.91 detik
Banyak iterasi hingga proses pencarian berhenti	90

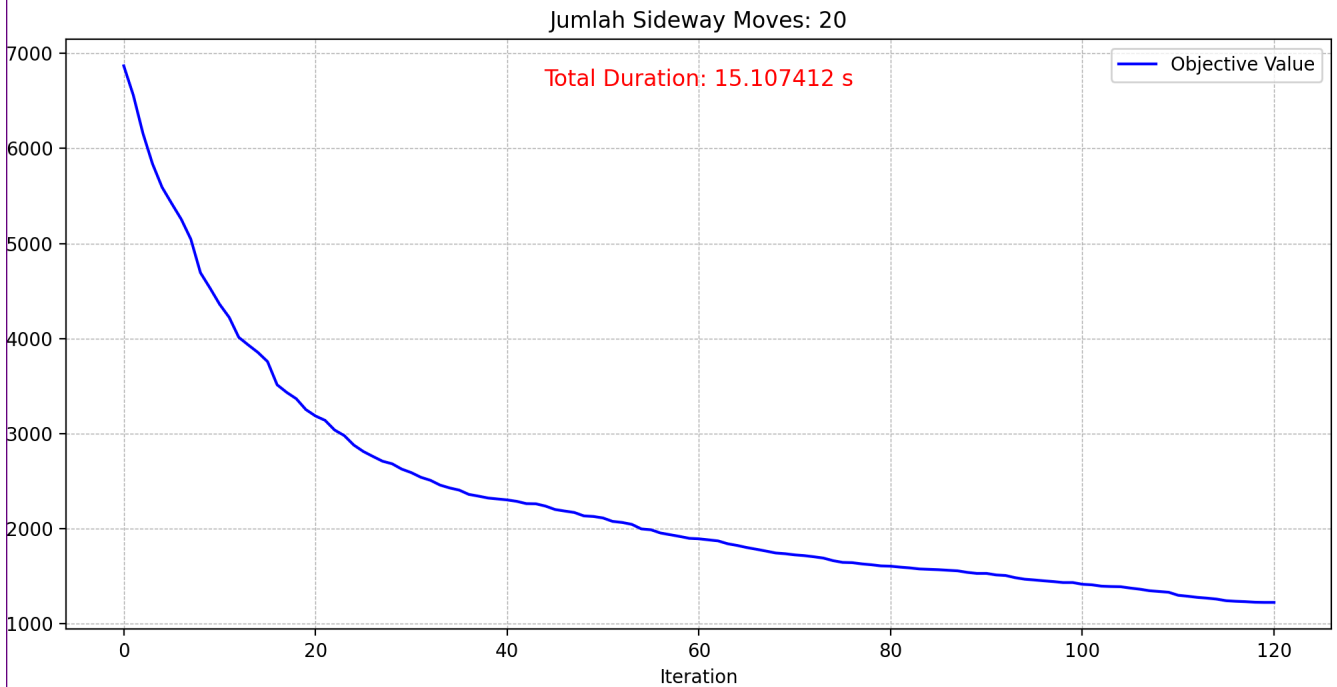
Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati



Eksperimen 2 ( <i>Maximum sideways move</i> = 20)			
<b>State awal</b>	[[[ 42 60 105 92 99] [ 83 108 33 44 80] [ 71 107 28 100 18] [ 57 101 78 46 38] [ 66 8 96 116 104]]	<b>State akhir</b>	[[[ 95 60 104 22 37] [ 75 74 5 44 116] [ 35 39 28 110 106] [ 29 124 103 57 8] [ 79 18 69 80 51]]

	[[ 82 76 45 81 124] [ 75 21 70 89 24] [ 88 14 1 27 79] [ 22 53 62 64 118] [ 72 9 50 32 51]]  [[ 74 95 11 120 41] [119 48 91 63 98] [ 69 39 4 26 93] [113 58 47 2 117] [ 67 52 125 86 85]]  [[ 17 94 16 40 55] [ 43 122 123 20 36] [114 15 102 73 25] [ 87 84 68 35 109] [112 111 49 90 12]]  [[ 59 106 37 30 103] [ 19 97 34 56 29] [ 65 121 6 61 13] [ 7 10 5 110 3] [ 23 115 54 77 31]]]		[[ 78 65 49 84 38] [ 88 24 62 89 41] [ 10 113 31 67 93] [ 87 83 68 64 21] [ 58 23 98 12 122]]  [[ 53 102 9 120 32] [119 7 91 107 1] [ 71 76 70 26 82] [ 52 94 50 6 115] [ 11 48 96 61 92]]  [[ 33 72 54 46 109] [ 19 97 118 20 55] [125 3 81 73 25] [ 17 34 47 90 112] [121 111 4 86 14]]  [[ 59 15 108 43 99] [ 16 101 30 56 105] [ 66 85 114 42 13] [123 2 27 100 63] [ 45 117 40 77 36]]]
	Nilai <i>Objective</i> Awal: 6869		Nilai <i>Objective</i> Akhir: 1225
Nilai <i>objective function</i> yang dicapai			1225
Durasi proses pencarian			15.11 detik
Banyak iterasi hingga proses pencarian berhenti			120
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			

Perbandingan objective function terhadap banyak iterasi yang telah dilewati menggunakan Sideways Hill-Climbing



### Eksperimen 3 (*Maximum sideways move = 30*)

State awal	State akhir
[[[101 11 109 31 63] [ 74 99 107 100 28] [ 97 87 47 110 26] [105 35 14 95 12] [ 88 44 113 48 46]]	[[[ 90 12 80 31 112] [ 24 110 15 70 96] [ 97 29 20 99 32] [ 40 79 94 95 11] [ 64 56 108 28 62]]
[[ 76 96 49 120 22] [ 89 18 52 118 59] [ 50 9 4 77 10] [ 80 7 25 58 33] [ 64 124 55 119 82]]	[[ 76 123 21 27 68] [105 18 92 46 53] [ 44 74 116 77 10] [ 73 72 4 47 125] [ 17 25 86 119 57]]
[[ 69 66 20 122 27]	[[ 69 3 43 122 102]

	[ 36 23 121 1 114] [ 70 125 21 5 117] [ 54 62 71 108 75] [ 2 115 32 57 68]]  [[ 30 106 65 42 16] [102 41 103 8 3] [ 19 123 86 51 73] [111 92 67 53 85] [ 98 90 37 56 6]]  [[ 38 45 40 83 24] [104 81 60 15 93] [ 79 29 17 61 34] [ 91 39 84 13 72] [ 43 94 78 116 112]]]  Nilai <i>Objective</i> Awal: 7038		[ 36 33 120 2 114] [ 60 106 42 22 89] [ 54 58 66 118 6] [101 115 50 51 5]]  [[ 75 109 87 30 13] [111 41 55 117 9] [ 19 83 16 65 124] [ 8 71 93 61 85] [ 98 35 63 37 88]]  [[ 14 59 104 103 26] [ 38 113 34 81 45] [ 91 23 107 52 48] [121 39 67 1 84] [ 49 82 7 78 100]]]  Nilai <i>Objective</i> Akhir: 1190
Nilai <i>objective function</i> yang dicapai			1190
Durasi proses pencarian			12.28 detik
Banyak iterasi hingga proses pencarian berhenti			114
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Untuk menguji implementasi algoritma *hill-climbing with sideways move*, dilakukan eksperimen sebanyak tiga kali dengan tiga nilai *maximum sideways move* yang berbeda-beda, yakni 10, 20, dan 30. Berdasarkan eksperimen tersebut, dapat disimpulkan bahwa nilai *objective function* yang dicapai bergantung dengan jumlah *maximum sideways* yang digunakan. Jika jumlah *maximum sideways* semakin besar, hasil *objective function* cenderung akan semakin mendekati nilai *global maximum*, yakni nol. Namun, ketika bertambahnya jumlah *maximum sideways* berpeluang untuk terlalu banyak berfokus pada *plateau* sehingga hasil yang ditemukan belum tentu membaik. durasi yang dibutuhkan untuk proses pencarian juga akan semakin bertambah ketika jumlah *maximum sideways* ditingkatkan.

### 2.3.3. Algoritma *Random Restart Hill-climbing*

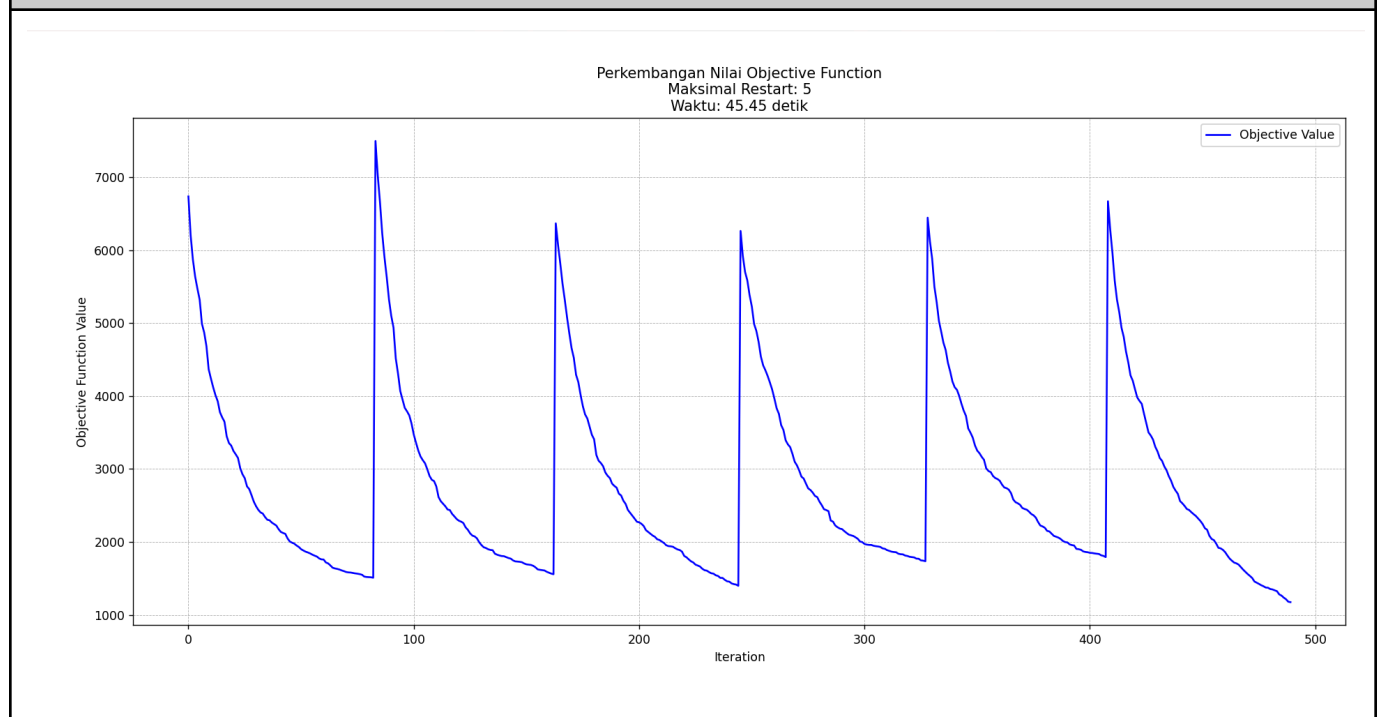
Berikut merupakan hasil eksperimen yang dilakukan sebanyak tiga kali menggunakan algoritma *random restart hill-climbing*.



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

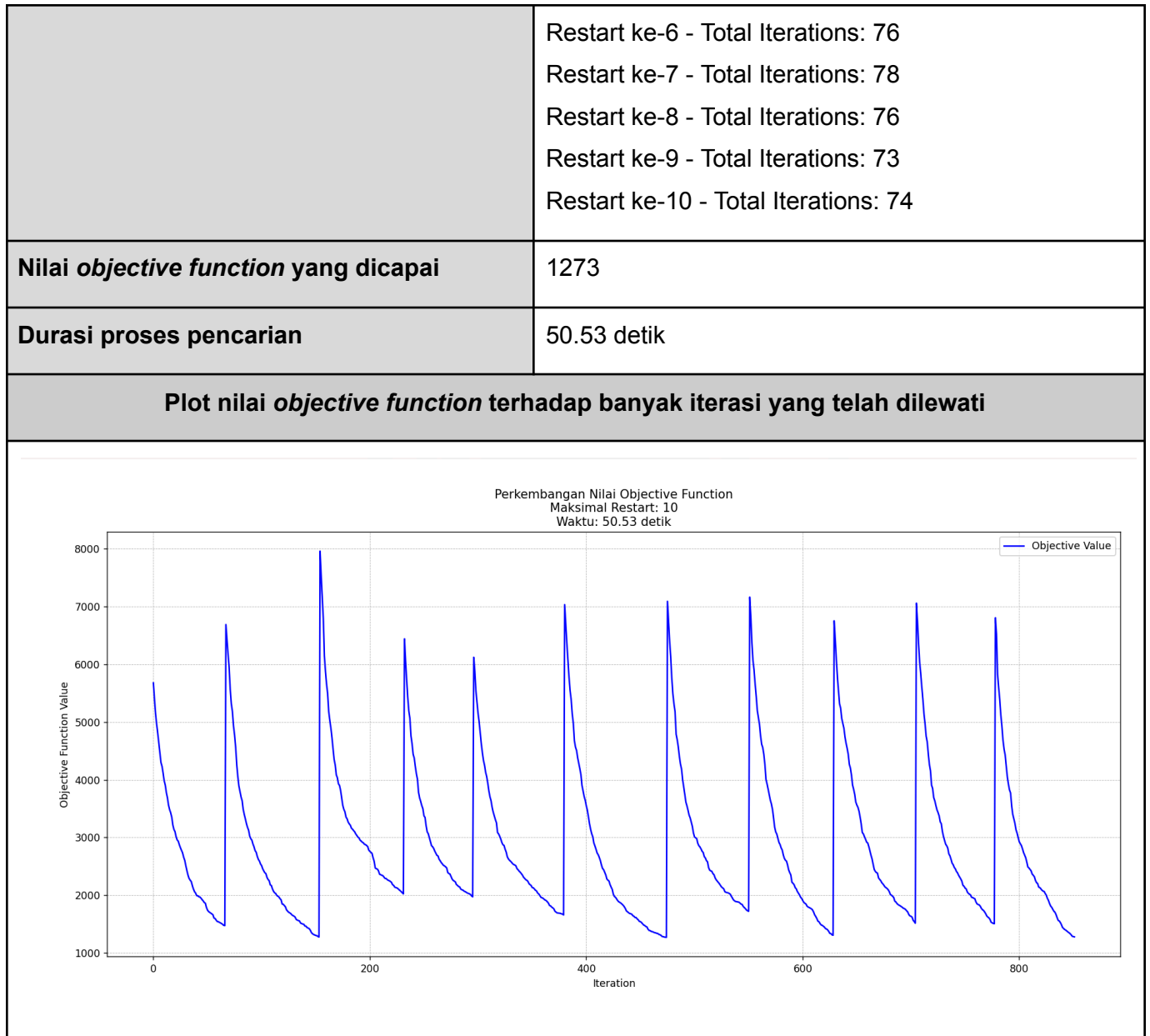
<b>Banyak iterasi per <i>restart</i></b>	Langkah pertama - Total Iterations: 83 Restart ke-1 - Total Iterations: 80 Restart ke-2 - Total Iterations: 82 Restart ke-3 - Total Iterations: 83 Restart ke-4 - Total Iterations: 80 Restart ke-5 - Total Iterations: 82
<b>Nilai <i>objective function</i> yang dicapai</b>	1175
<b>Durasi proses pencarian</b>	45.45 detik

**Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati**



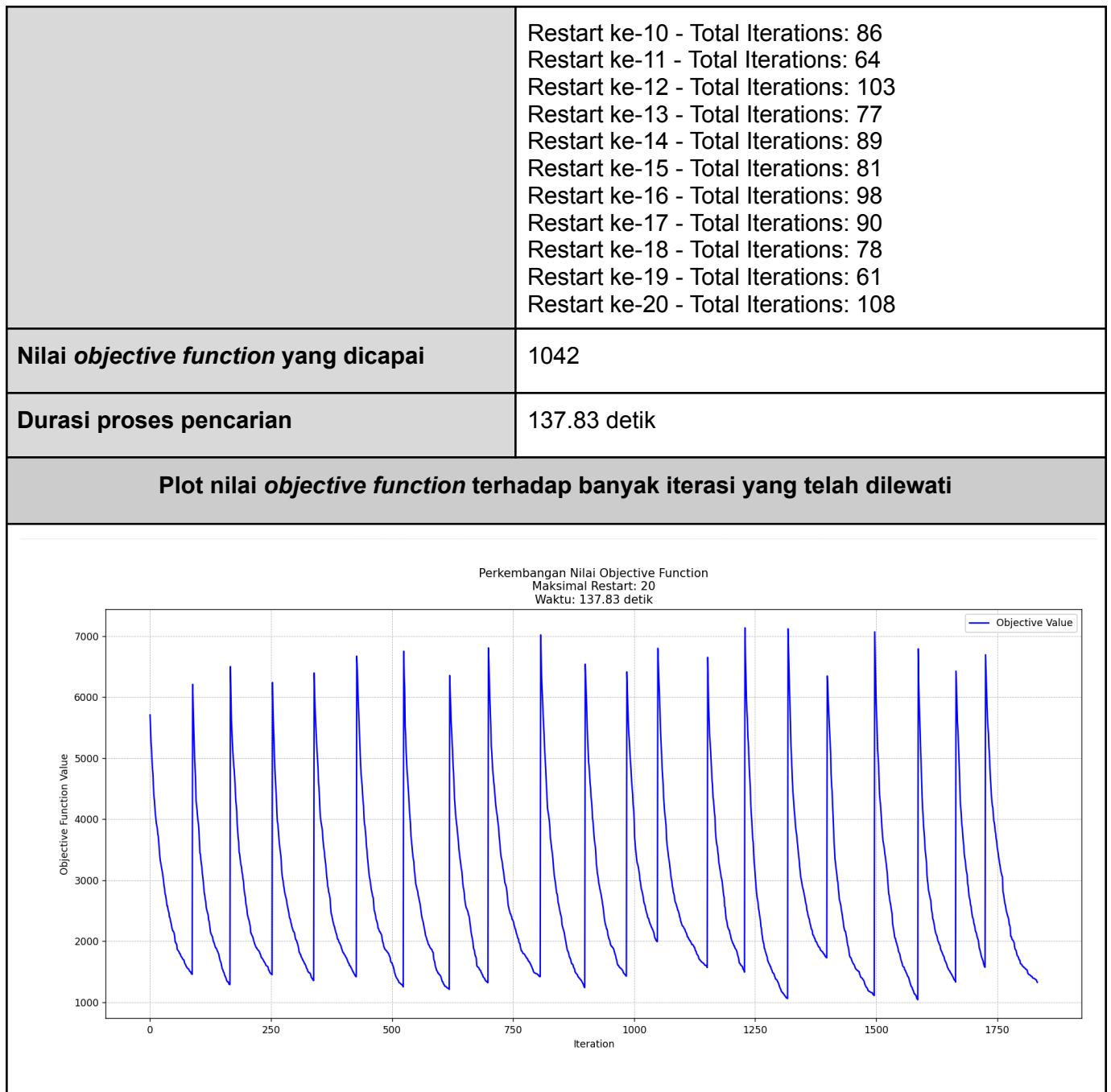
<b>Eksperimen 2</b>			
<b>State awal</b>	[[[ 86 3 56 96 18] [ 71 51 123 28 117] [ 38 99 122 2 36] [ 80 33 72 84 29] [ 48 42 74 110 55]]]	<b>State akhir</b>	[[[ 33 19 119 45 100] [ 79 112 49 42 30] [ 67 10 40 122 91] [ 115 104 84 11 1] [ 31 73 26 68 117]]]

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



Eksperimen 3			
State awal	[[[ 56 59 110 2 92] [ 37 20 7 96 27] [107 119 60 28 112] [ 95 66 45 104 121] [ 23 108 99 6 109]]	State akhir	[[[ 30 82 8 96 91] [ 46 44 97 84 53] [ 83 118 78 14 19] [109 21 9 102 56] [ 41 49 123 16 94]]

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



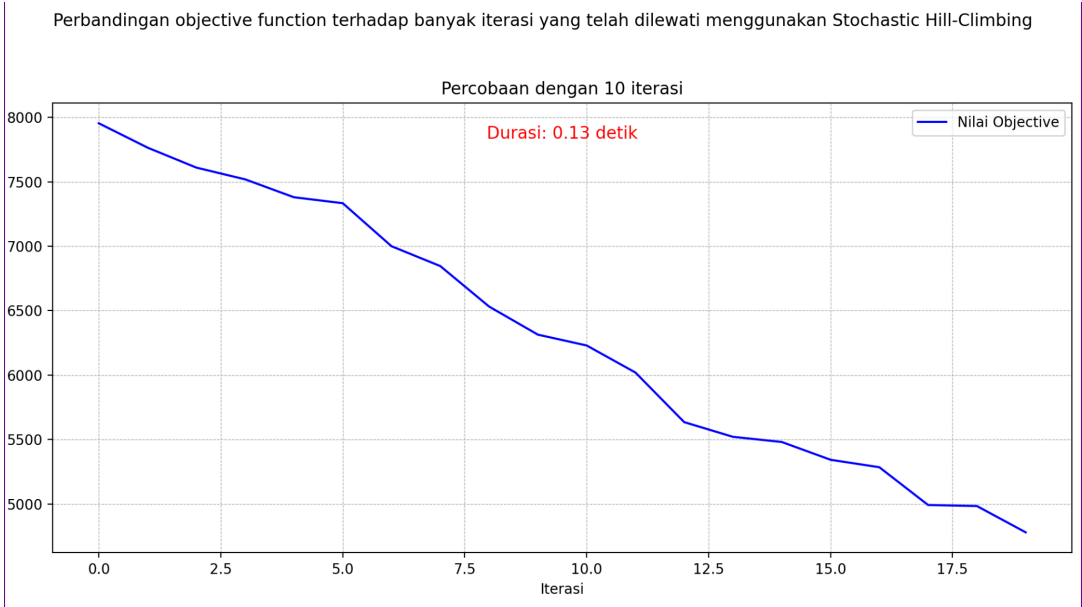
Untuk menguji implementasi algoritma *random restart hill-climbing*, dilakukan eksperimen sebanyak tiga kali dengan tiga nilai *maximum restart* yang berbeda-beda, yakni 5, 10, dan 20. Berdasarkan hasil eksperimen tersebut, dapat disimpulkan bahwa hasil pencarian akan semakin mendekati *global maximum* atau semakin optimal jika jumlah *maximum restart* ditingkatkan. Selain dipengaruhi oleh maksimal iterasi

dalam eksplorasi *successor*, durasi pencarian juga akan semakin meningkat seiring meningkatnya nilai *maximum restart*. Selain itu juga, pada hasil plot *objective function value* dan iterasinya, dapat dilihat bahwa terjadi lonjakan-lonjakan *objective function value*. Hal inilah yang menunjukkan karakteristik algoritma *random restart hill-climbing* yang melakukan pencarian ulang dengan inisiasi awal yang acak.

#### 2.3.4. Algoritma *Stochastic Hill-climbing*

Berikut merupakan hasil eksperimen yang dilakukan sebanyak tiga kali menggunakan algoritma *stochastic hill-climbing*.

Eksperimen 1 (Maksimal iterasi = 10)			
State awal		State akhir	
	[[[ 96 51 8 68 7] [ 66 22 69 87 75] [ 48 25 108 31 89] [115 125 24 110 82] [116 123 99 76 42]]		[[[ 96 51 8 68 7] [ 66 22 69 87 75] [ 48 25 108 31 89] [ 21 125 24 110 82] [ 63 123 99 76 42]]
	[[120 67 40 23 38] [ 26 74 88 18 27] [101 13 10 73 98] [ 92 84 43 32 19] [ 28 2 107 46 111]]		[[120 67 40 23 121] [ 26 74 88 70 27] [101 41 10 73 98] [ 92 84 43 32 19] [ 28 2 107 119 111]]
	[[ 97 16 5 20 112] [113 44 52 50 104] [ 35 55 119 56 54] [ 77 29 105 100 14] [ 95 79 17 49 33]]		[[ 97 16 5 20 112] [113 44 52 50 104] [ 35 55 46 56 54] [ 65 29 105 100 14] [ 95 106 17 86 33]]
	[[118 61 94 83 11] [ 81 109 117 58 103]		[[ 36 61 94 83 11] [ 81 109 117 58 103]

	<p>[ 21 59 53 60 15] [ 34 102 64 6 114] [ 3 70 106 36 85]]</p> <p>[[ 4 1 41 63 47] [ 45 57 86 30 65] [ 91 72 39 71 80] [ 62 121 124 78 9] [ 90 37 93 12 122]]]</p> <p>Nilai <i>Objective</i> Awal: 7914</p>		<p>[115 59 53 60 15] [ 34 102 64 6 114] [ 3 18 79 118 85]]</p> <p>[[ 4 1 13 116 47] [ 45 57 49 30 77] [ 91 72 39 71 80] [ 62 38 124 78 9] [ 90 37 93 12 122]]]</p> <p>Nilai <i>Objective</i> Akhir: 4780</p>
Nilai <i>objective function</i> yang dicapai			4780
Durasi proses pencarian			0.13 detik
Banyak iterasi hingga proses pencarian berhenti			10
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			
<p>Perbandingan objective function terhadap banyak iterasi yang telah dilewati menggunakan Stochastic Hill-Climbing</p> <p>Percobaan dengan 10 iterasi</p>  <p>Durasi: 0.13 detik</p> <p>Nilai Objective</p>			



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

	6321		3428
Nilai <i>objective function</i> yang dicapai			3428
Durasi proses pencarian			0.42 detik
Banyak iterasi hingga proses pencarian berhenti			20

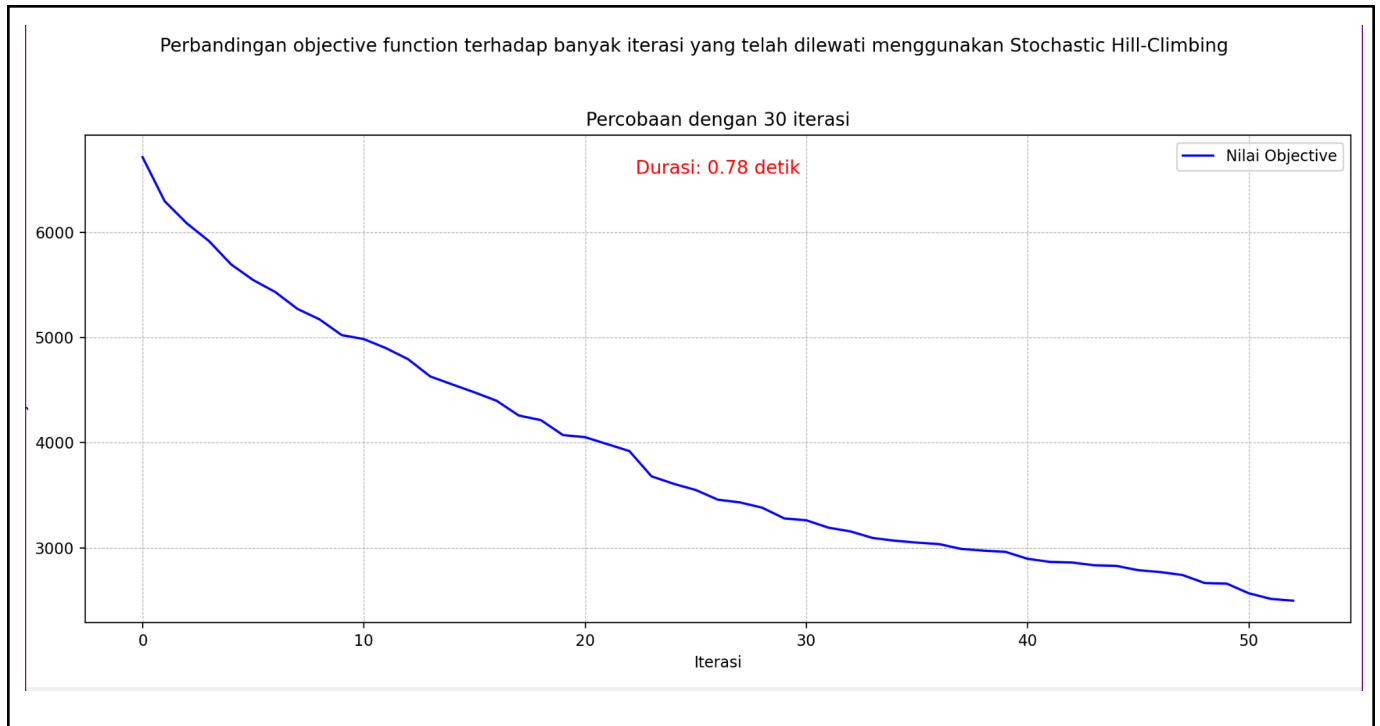
**Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati**



**Eksperimen 3 (Maksimal iterasi = 30)**

<b>State awal</b>	<pre>[[[112 96 77 27 4]  [ 53 2 108 113 91]  [ 30 81 56 86 61]  [ 93 72 19 39 97]  [ 60 54 79 24 109]]   [[ 9 38 107 95 52]  [ 17 116 31 55 32]]</pre>	<b>State akhir</b>	<pre>[[[ 87 96 72 27 4]  [ 53 2 108 113 91]  [ 30 81 56 86 106]  [ 85 77 19 39 97]  [ 60 57 79 9 109]]   [[ 24 74 88 95 17]  [ 44 116 31 66 32]]</pre>
-------------------	--	--------------------	--

	[ 75 49 58 8 124] [ 29 64 125 68 90] [117 7 13 102 20]]  [[ 92 44 36 80 78] [ 88 1 47 74 114] [ 66 121 34 48 11] [ 3 100 57 110 40] [ 50 76 123 5 69]]  [[ 41 118 46 12 122] [ 87 59 16 37 33] [ 42 106 115 98 26] [103 6 105 67 70] [ 21 119 35 101 84]]  [[ 71 18 63 111 82] [ 25 120 94 14 28] [104 45 51 83 23] [ 99 85 10 62 43] [ 89 22 65 73 15]]]  Nilai <i>Objective</i> Awal: 6713		[ 75 49 58 8 124] [ 25 64 125 68 50] [117 7 13 102 62]]  [[ 92 28 36 80 78] [107 1 47 48 114] [ 55 110 34 90 11] [ 3 100 54 121 40] [ 38 76 123 5 69]]  [[ 41 118 46 12 122] [112 59 16 37 33] [ 42 61 115 98 26] [ 82 6 105 67 70] [ 21 119 35 111 29]]  [[ 71 18 63 101 103] [ 43 120 94 14 52] [104 23 51 83 45] [ 99 93 10 20 84] [ 89 22 65 73 15]]]  Nilai <i>Objective</i> Akhir: 2498
Nilai <i>objective function</i> yang dicapai			2498
Durasi proses pencarian			0.78 detik
Banyak iterasi hingga proses pencarian berhenti			30
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Untuk menguji implementasi algoritma *stochastic hill-climbing*, dilakukan eksperimen sebanyak tiga kali dengan tiga nilai *maximum iterations* yang berbeda-beda, yakni 10, 20, dan 30. Berdasarkan hasil eksperimen tersebut, dapat disimpulkan bahwa hasil pencarian akan semakin mendekati *global maximum* atau semakin optimal jika jumlah *maximum iterations* ditingkatkan. Durasi pencarian juga akan semakin meningkat seiring meningkatnya nilai *maximum iterations*.

### 2.3.5. Algoritma *Simulated Annealing*

Berikut merupakan hasil eksperimen yang dilakukan sebanyak tiga kali menggunakan algoritma *simulated annealing*.

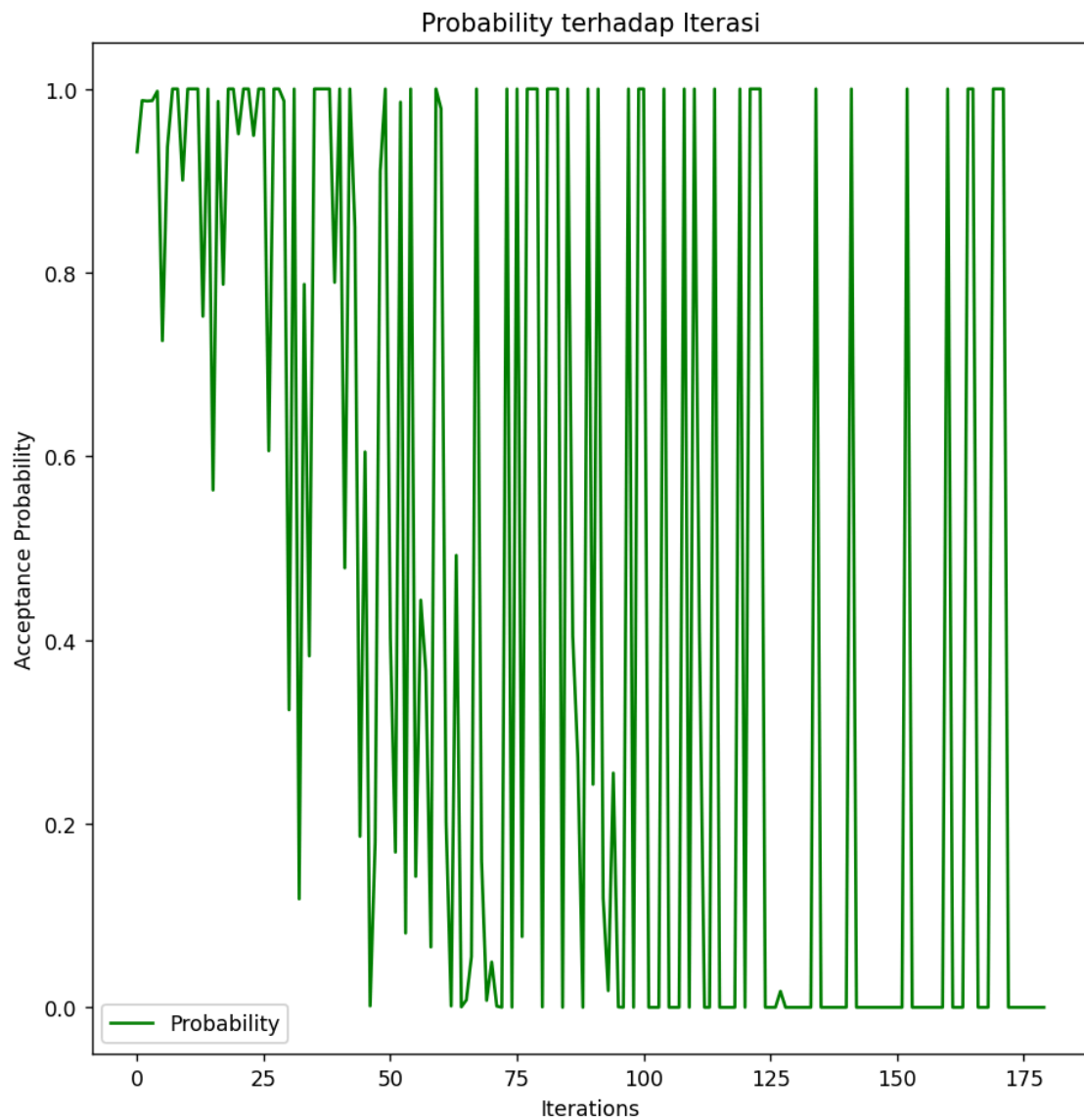
Eksperimen 1 (Temperatur awal = 1000, <i>Cooling rate</i> = 0.95, Temperatur minimal = 0.1)			
<b>State awal</b>	[[[ 30 107 73 113 7] [101 114 106 89 79] [ 45 42 26 18 12] [117 70 23 14 8]	<b>State akhir</b>	[[[ 30 20 23 104 85] [ 15 12 106 108 67] [ 45 121 26 52 80] [117 70 73 63 8]

	<p>[ 99 94 123 5 118]]</p> <p>[[ 33 54 104 20 122] [ 13 100 49 41 3] [ 90 28 44 29 95] [ 51 50 34 71 40] [ 65 38 116 63 91]]</p> <p>[[ 88 69 57 48 27] [ 82 66 87 56 17] [124 64 19 77 109] [ 92 24 11 6 111] [ 68 39 110 35 47]]</p> <p>[[ 9 4 32 112 108] [ 93 31 16 102 75] [ 37 120 15 59 25] [125 78 98 36 10] [ 84 74 80 2 103]]</p> <p>[[ 21 119 58 121 61] [ 67 53 1 76 105] [ 96 85 62 60 46] [ 83 81 97 86 43] [ 55 52 22 115 72]]]</p> <p>Nilai Initial Objective Function: 6764</p>		<p>[ 99 105 96 28 6]]</p> <p>[[ 5 66 18 124 49] [ 53 97 122 41 24] [ 55 33 95 29 75] [ 51 46 34 71 40] [123 61 19 83 43]]</p> <p>[[ 88 69 84 48 27] [ 82 90 7 36 17] [ 64 2 107 38 120] [ 44 114 22 102 31] [ 42 39 92 76 86]]</p> <p>[[ 77 32 113 111 1] [ 3 112 16 110 118] [ 87 68 101 109 25] [125 59 98 10 89] [ 50 74 93 4 103]]</p> <p>[[ 21 119 58 94 9] [ 79 57 78 14 116] [ 65 91 62 60 47] [ 81 13 100 35 37] [ 54 56 11 115 72]]]</p> <p>Nilai Final Objective Function: 4143</p>
Frekuensi ' <i>stuck</i> ' di <i>local optima</i>			8
Nilai <i>objective function</i> yang dicapai			4143
Durasi proses pencarian			0.08

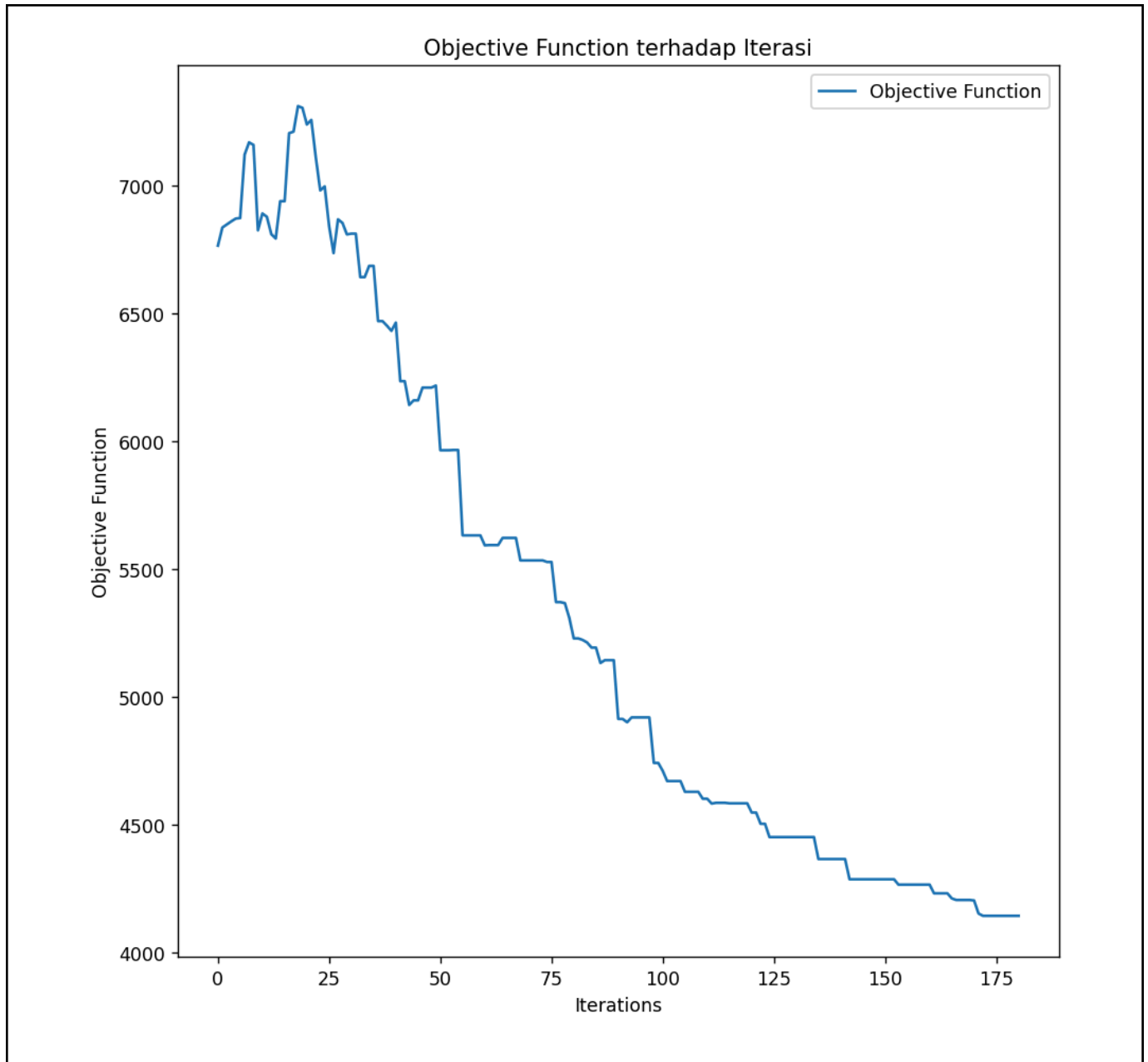
Banyak iterasi hingga proses pencarian berhenti

83

Plot  $e^{\frac{\Delta E}{T}}$  terhadap banyak iterasi yang telah dilewati



Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati



Eksperimen 2 (Temperatur awal = 10000, <i>Cooling rate</i> = 0.99, Temperatur minimal = 0.01)			
<b>State awal</b>	<div>[[[ 78 71 98 6 101]</div> <div>[114 59 76 105 77]</div> <div>[ 33 44 120 69 79]</div> <div>[ 54 28 61 64 125]</div>	<b>State akhir</b>	<div>[[[ 17 11 97 100 106]</div> <div>[113 32 29 45 34]</div> <div>[ 73 119 33 2 79]</div> <div>[ 46 51 104 125 1]</div>

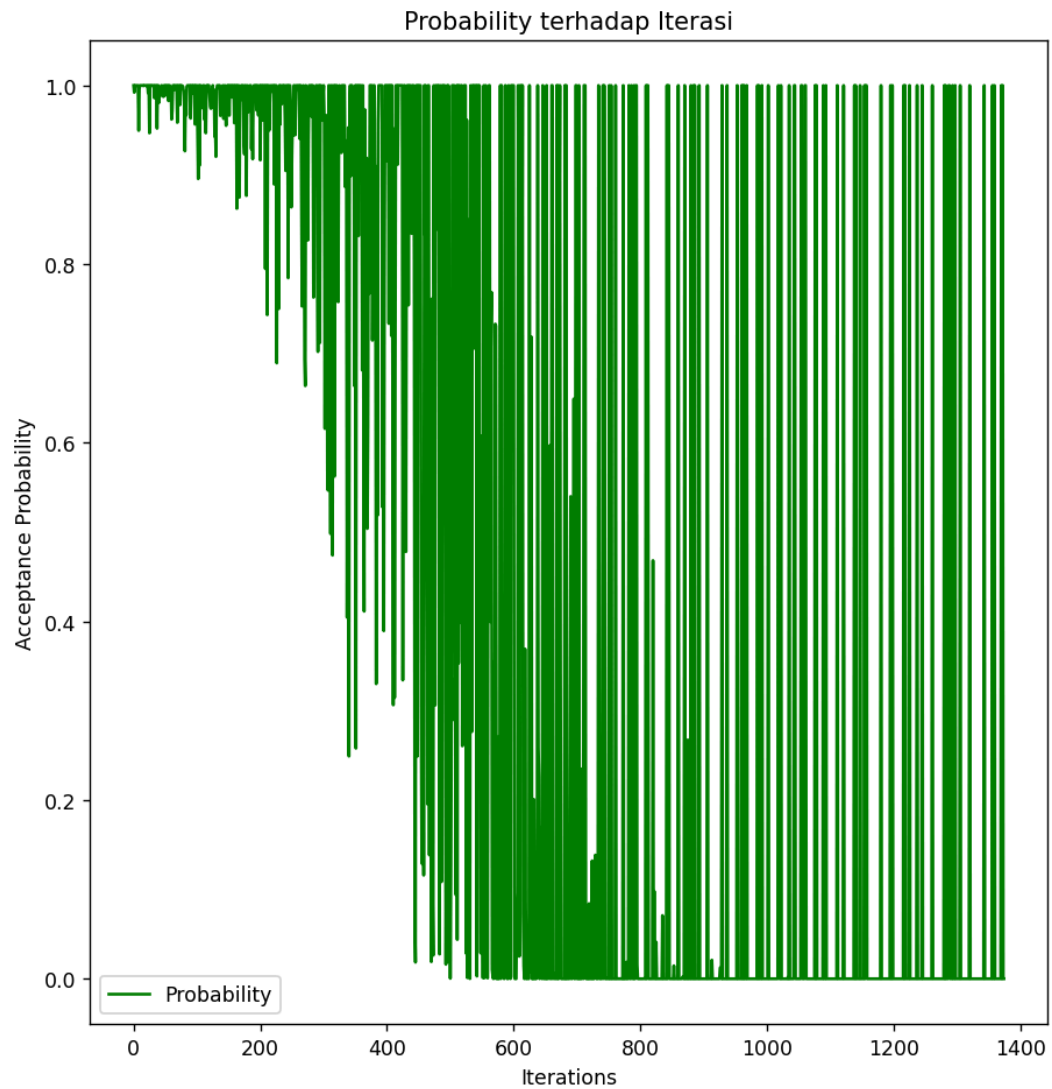
	[ 1 60 38 46 30]]  [[ 12 27 40 103 2] [ 67 109 89 74 35] [ 26 123 111 22 23] [ 37 31 116 34 25] [100 93 15 106 108]]  [[110 8 84 32 118] [ 4 55 82 3 43] [ 19 48 21 95 97] [124 7 87 99 104] [107 42 20 11 62]]  [[ 86 112 92 81 70] [121 57 91 13 63] [117 94 115 29 122] [ 53 83 56 10 47] [ 9 80 16 50 72]]  [[ 90 49 14 88 66] [ 65 73 17 24 96] [ 75 85 41 18 68] [119 39 58 52 51] [ 36 113 5 102 45]]]  Nilai Initial Objective Function: 6920		[ 81 103 31 8 86]]  [[ 91 115 111 15 20] [ 21 67 99 70 84] [ 55 22 35 80 108] [ 58 59 36 63 69] [ 82 62 27 107 38]]  [[ 94 96 23 7 110] [ 48 47 19 120 72] [124 89 75 68 4] [ 49 14 98 37 93] [ 5 71 102 114 40]]  [[ 87 117 25 83 3] [ 10 90 76 42 95] [ 12 9 52 122 118] [109 101 54 6 56] [121 16 88 30 61]]  [[ 28 13 60 123 66] [105 85 92 43 44] [ 50 78 116 41 24] [ 39 74 26 53 112] [ 18 57 65 64 77]]]  Nilai Final Objective Function: 1905
Frekuensi ' <i>stuck</i> ' di <i>local optima</i>			2
Nilai <i>objective function</i> yang dicapai			1905
Durasi proses pencarian			0.57



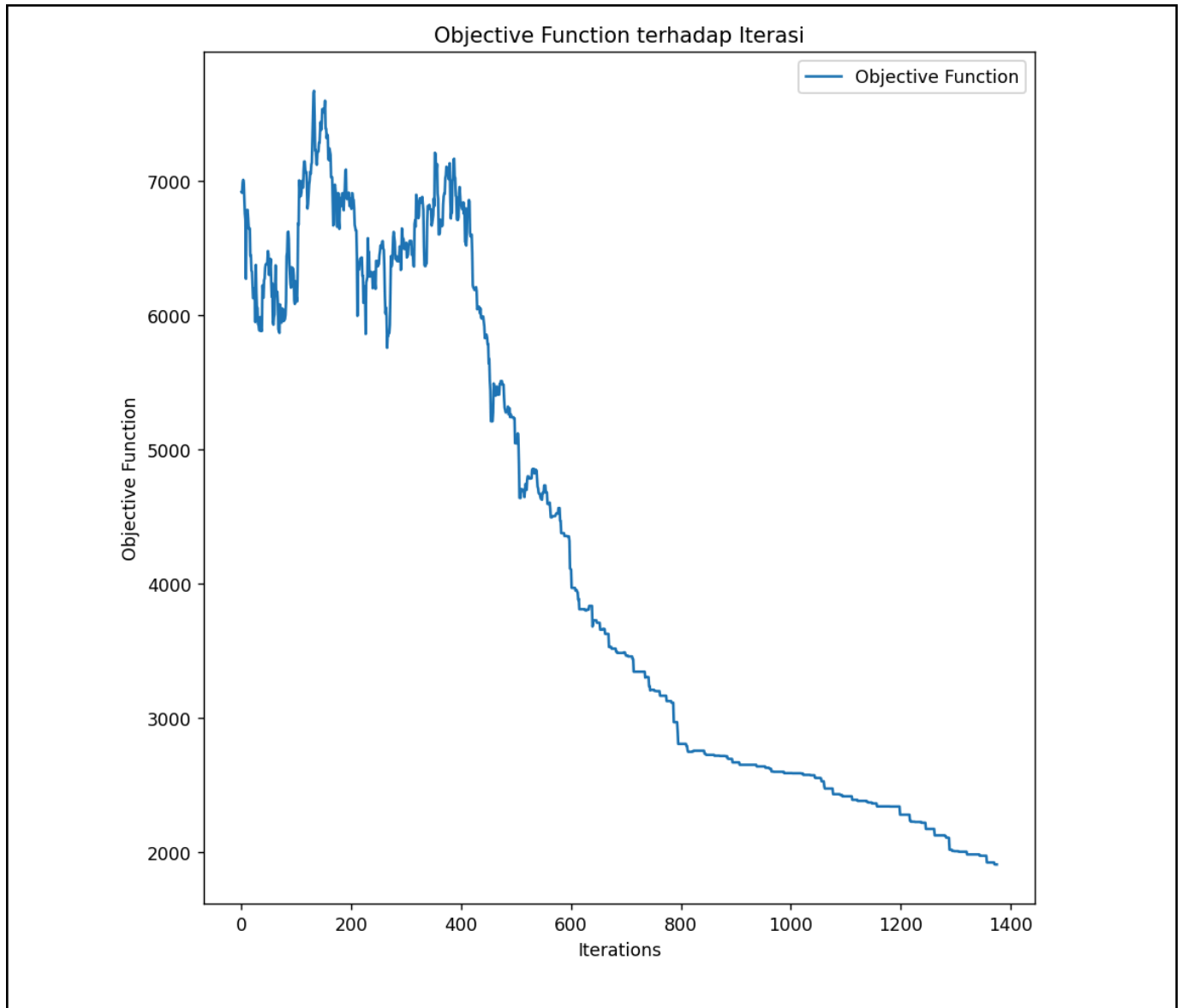
Banyak iterasi hingga proses pencarian berhenti

615

Plot  $e^{\frac{\Delta E}{T}}$  terhadap banyak iterasi yang telah dilewati



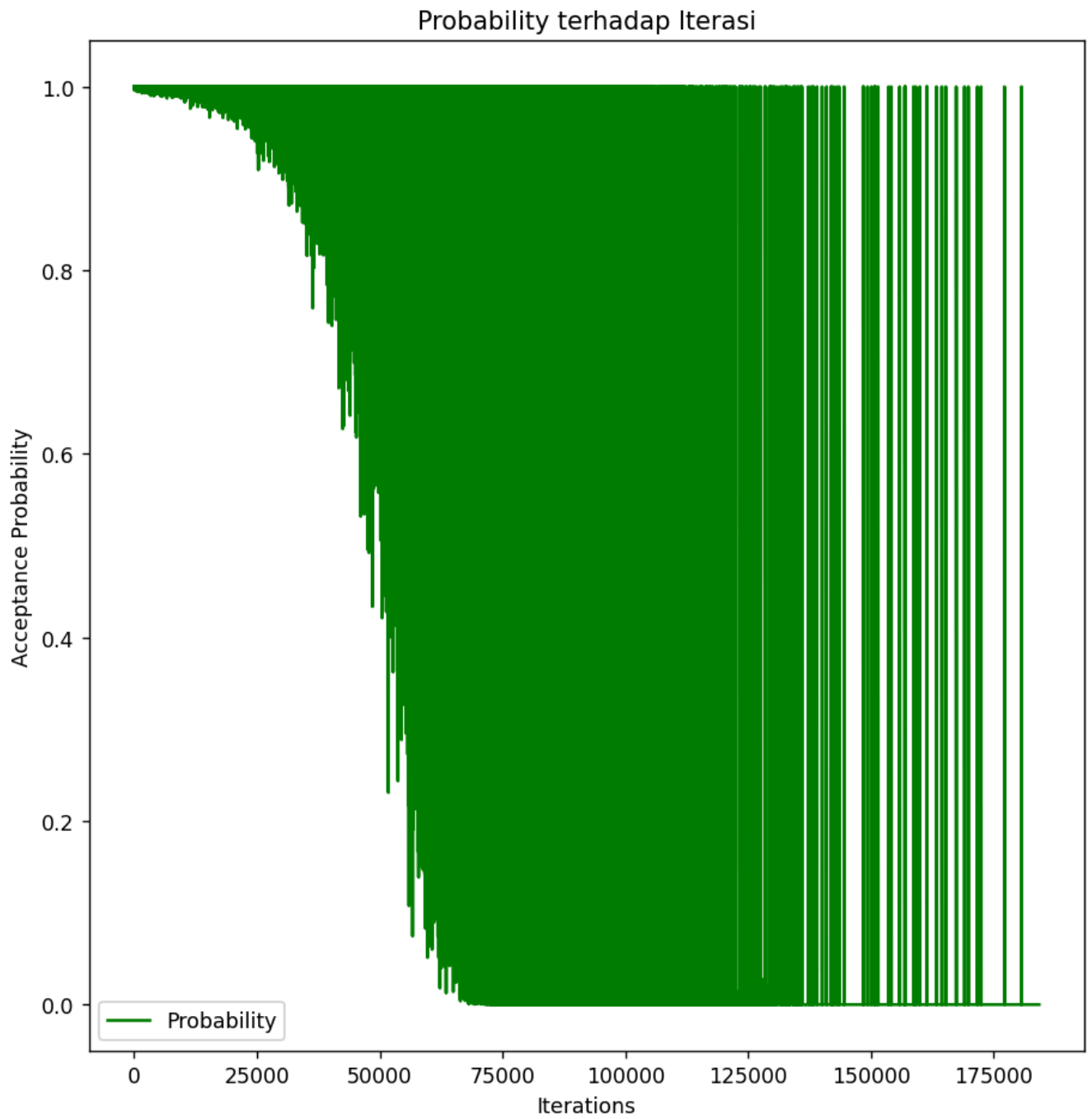
Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati



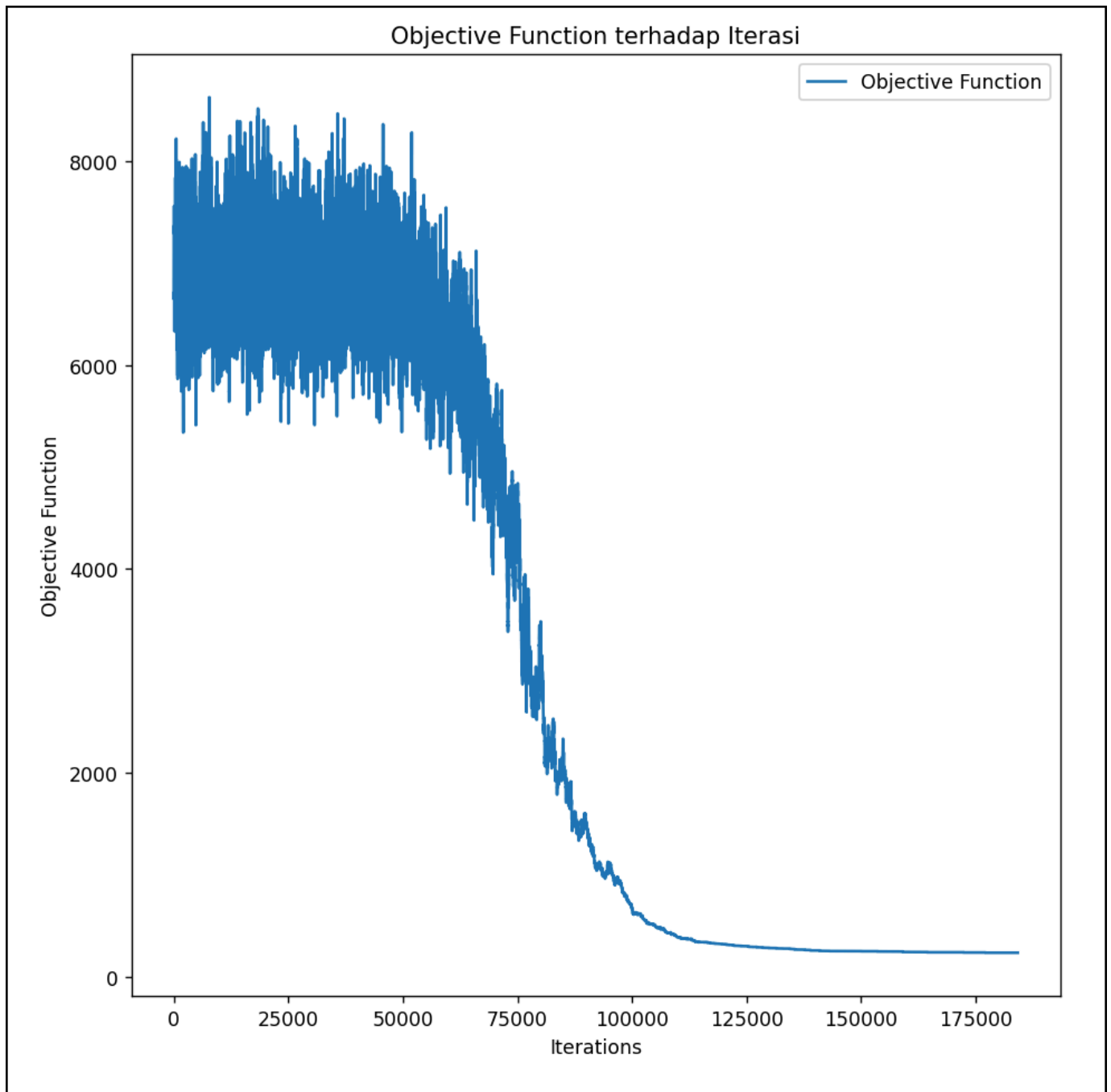
Eksperimen 3 (Temperatur awal = 100000, Cooling rate = 0.9999, Temperatur minimal = 0.001)			
<b>State awal</b>	<pre>[[[ 92  51  56  50  43]   [ 13  64  26  68  11]   [ 32  27 116  84 105]   [ 94  14  62  28  54]   [122  79  12  95  72]]  [[ 38  19  47 109  21]</pre>	<b>State akhir</b>	<pre>[[[ 76  34  89  48  72]   [ 12  57 125  60  56]   [ 73  55   6 119  61]   [ 74  98  77  58   8]   [ 80  71  18  30 118]]  [[ 59  29 109  21  96]</pre>

	[ 22 7 123 8 82] [111 78 48 112 106] [ 23 101 4 90 121] [ 98 63 46 6 83]]  [[ 85 93 125 81 49] [ 10 41 113 57 86] [ 35 65 80 15 97] [ 5 77 75 44 91] [108 55 3 67 18]]  [[ 25 33 100 66 124] [ 30 61 69 99 36] [ 74 1 89 16 29] [ 37 52 31 120 96] [ 59 88 87 24 60]]  [[ 2 53 17 114 9] [ 71 70 110 76 40] [ 45 42 20 103 39] [102 107 115 117 73] [ 34 118 58 104 119]]]  Nilai Initial Objective Function: 6708		[ 79 85 17 52 82] [ 15 70 113 88 31] [117 9 67 39 83] [ 45 122 10 115 23]]  [[ 24 75 64 53 99] [108 19 43 66 78] [110 87 62 42 14] [ 22 41 95 121 35] [ 47 93 51 33 91]]  [[106 94 11 101 4] [ 1 50 123 111 36] [100 90 20 3 102] [ 5 81 37 68 120] [103 2 124 32 54]]  [[ 46 84 44 92 49] [116 104 7 26 63] [ 16 13 114 65 107] [ 97 86 38 25 69] [ 40 28 112 105 27]]]  Nilai Final Objective Function: 235
Frekuensi ' <i>stuck</i> ' di <i>local optima</i>			3582
Nilai <i>objective function</i> yang dicapai			235
Durasi proses pencarian			78.47
Banyak iterasi hingga proses pencarian berhenti			73188

Plot  $e^{\frac{\Delta E}{T}}$  terhadap banyak iterasi yang telah dilewati



Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati



Pada percobaan eksperimen algoritma *Simulated Annealing*, dalam setiap eksperimen memiliki temperatur awal, *cooling rate*, temperatur minimal yang berbeda-beda yang pada setiap eksperimen nilai-nilai ini akan dimulai dari terkecil pada eksperimen 1 dan terbesar pada eksperimen 3. Berdasarkan eksperimen yang dilakukan, nilai yang paling mendekati *objective function* adalah eksperimen ke-3 dimana nilai

tersebut adalah 235 dan didapatkan dengan memasukkan nilai-nilai input yang cukup besar, hal ini dapat didapatkan dikarenakan algoritma ini dapat melakukan eksplorasi pada awal algoritma dimulai dengan memungkinkan pengambilan langkah yang lebih buruk dari nilai saat ini. Hal ini dapat dilihat dari plot nilai setiap eksperimen dimana nilai *objective function* cenderung naik turun dibandingkan pada saat pertengahan jalannya algoritma. Selain itu, pada plot nilai probabilitas menunjukkan bahwa algoritma akan lebih berhati-hati dalam mengambil langkah pada saat algoritma berjalan cukup lama. Hal inilah yang menyebabkan algoritma dapat mendekati nilai optimal karena semakin lama algoritma berjalan, hasil akan semakin mendekati *objective function*. Hasil algoritma ini merupakan hasil terbaik dibandingkan dengan algoritma lain dikarenakan algoritma ini memungkinkan eksplorasi yang lebih pada *magic cube* dibandingkan algoritma lainnya. Durasi algoritma ini akan bergantung dengan nilai inputnya, sehingga algoritma ini dapat menjadi lebih cepat atau lebih lambat dari algoritma lain, namun untuk mendapatkan hasil yang optimal akan memakan waktu yang cukup lama sehingga durasi algoritma ini akan menjadi yang terlama secara *general*. Hasil akhir yang didapatkan pada tiap eksperimen menunjukkan hasil yang cukup konsisten disebabkan semakin besar nilai input hasil akan semakin baik serta durasi algoritma akan semakin lama. Selain itu plot nilai juga menunjukkan pola yang cukup sama secara keseluruhan. Untuk saat ini algoritma cukup konsisten dalam menghasilkan nilai *objective function*, namun terkadang algoritma masih memerlukan sedikit keberuntungan untuk mencapai nilai yang lebih baik dikarenakan masih menggunakan nilai *random* pada penerimaan langkah, sehingga algoritma ini masih bisa ditingkatkan agar mencapai nilai yang lebih konsisten dari sekarang.

#### **2.3.6. Algoritma Genetic**

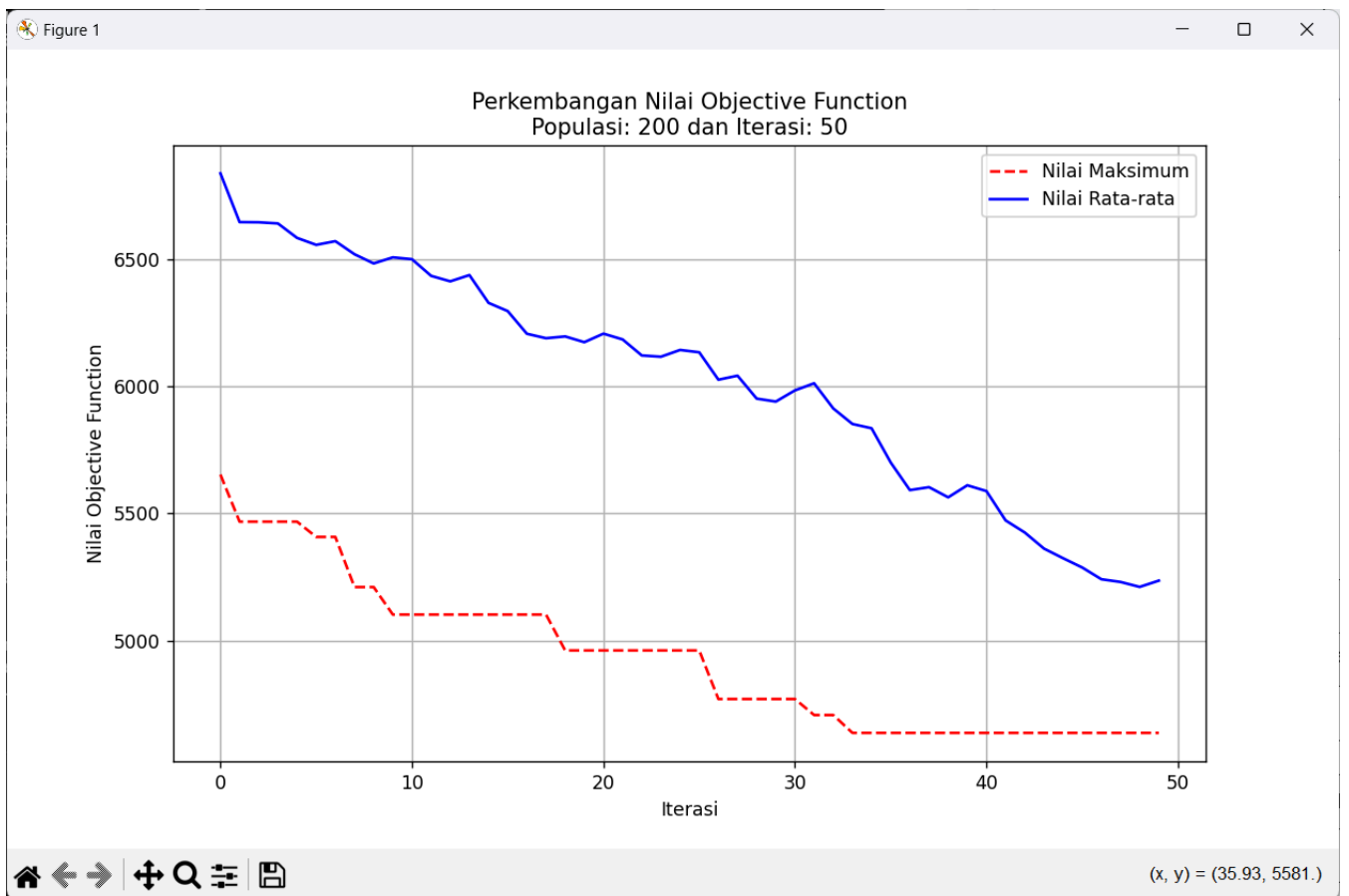
Berikut merupakan hasil eksperimen yang dilakukan sebanyak tiga kali menggunakan algoritma *genetic*.

### 2.3.6.1. Jumlah populasi sebagai kontrol

Eksperimen 1: Populasi 200 Iterasi 50			
State awal		State akhir	
	[[[ 54 16 103 65 1] [ 27 116 63 19 98] [ 6 25 109 43 121] [ 36 51 20 11 28] [ 4 48 32 31 118]]		[[[ 44 36 78 90 19] [ 56 12 89 65 97] [ 99 96 21 3 109] [ 14 33 115 119 50] [105 101 22 7 110]]
	[[ 9 78 89 23 95] [ 76 125 115 101 86] [ 53 12 29 100 99] [ 5 41 70 42 3] [ 35 62 106 47 40]]		[[107 20 112 13 82] [ 30 118 2 18 92] [ 37 55 24 124 47] [ 46 45 54 69 111] [ 28 86 73 83 26]]
	[[ 37 91 66 52 26] [ 73 108 105 7 49] [104 14 123 38 45] [112 68 57 67 44] [ 39 46 71 96 97]]		[[ 11 114 53 49 81] [ 25 15 62 102 59] [ 60 40 76 72 68] [ 74 116 31 108 64] [ 93 42 61 1 104]]
	[[111 119 56 69 117] [ 92 17 13 55 122] [ 60 93 80 21 77] [ 22 79 94 24 114] [ 88 113 82 81 107]]		[[ 5 94 67 113 8] [117 66 103 52 106] [ 39 6 23 48 121] [ 95 125 57 34 9] [ 58 38 17 80 91]]
	[[ 61 74 102 120 18] [ 90 124 110 15 75] [ 2 72 8 84 87] [ 64 34 30 10 50] [ 85 33 58 59 83]]		[[ 4 87 75 10 29] [ 70 35 71 79 77] [100 120 63 98 16] [ 41 32 51 43 122] [ 88 27 123 84 85]]

Jumlah populasi	200
Banyak iterasi	50
Nilai <i>objective function</i> yang dicapai	4638
Durasi proses pencarian	6.84 detik

**Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati**



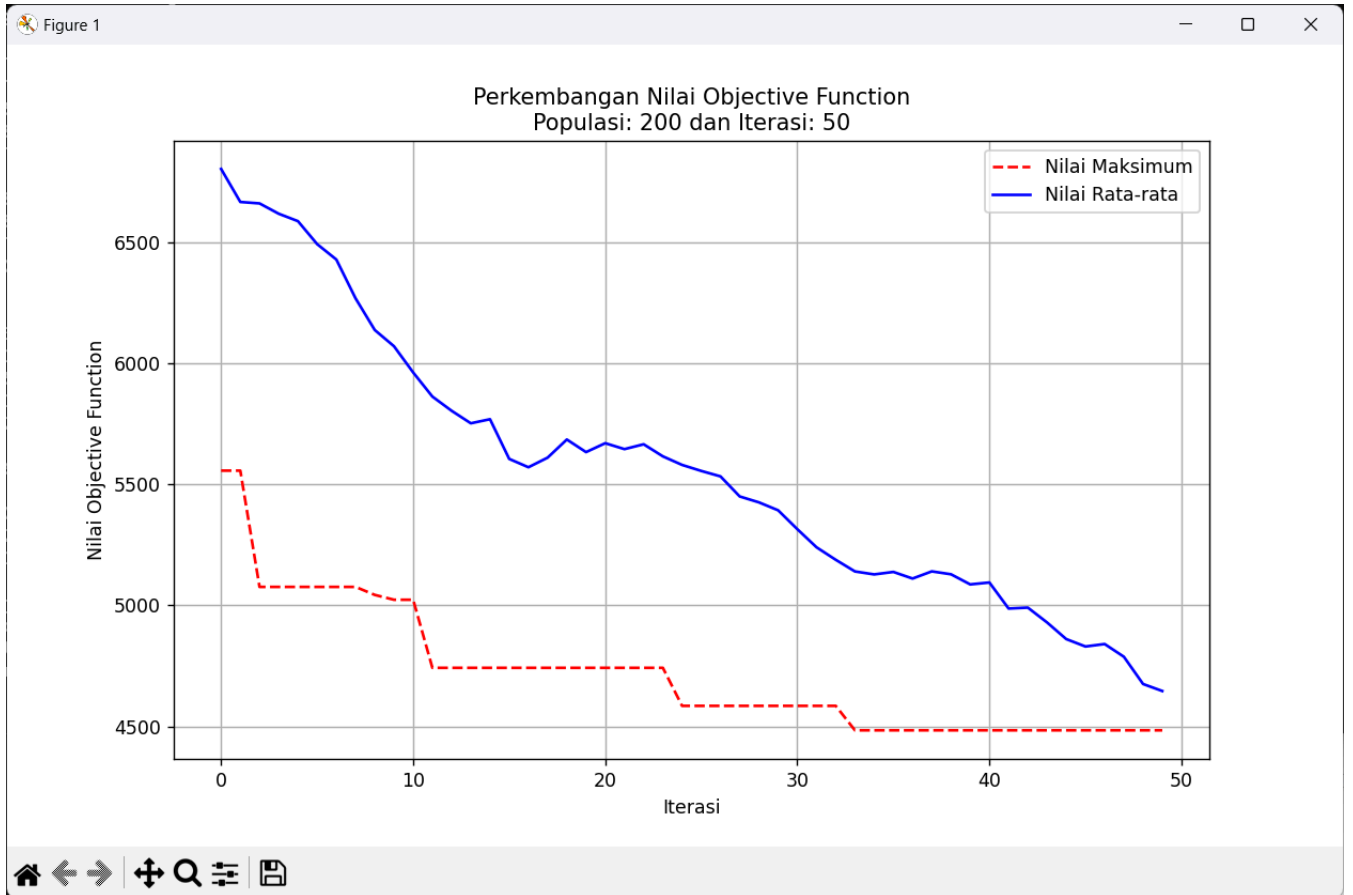
### Eksperimen 2: Populasi 200 Iterasi 50

<b>State awal</b>	[[[120 48 85 92 96] [ 95 41 80 121 39] [ 60 24 98 73 74]	<b>State akhir</b>	[[[101 38 115 49 123] [ 75 62 77 94 56] [114 87 16 36 45]
-------------------	--	--------------------	---



	[ 44 34 84 67 37] [ 4 112 122 27 81]]  [[ 86 59 1 2 19] [ 33 55 79 40 57] [115 118 76 87 20] [ 68 109 38 105 31] [ 16 23 123 36 110]]  [[ 22 11 69 97 15] [ 62 107 117 119 113] [125 30 17 88 9] [ 93 102 75 50 13] [ 71 6 14 65 54]]  [[ 89 90 21 3 82] [ 25 72 114 64 78] [ 77 63 10 94 106] [ 32 108 8 111 46] [116 47 56 104 101]]  [[ 42 124 28 43 29] [ 26 91 45 100 99] [ 12 18 52 70 7] [ 58 61 49 103 53] [ 35 51 5 66 83]]]]		[117 29 84 66 15] [ 5 96 59 8 88]]  [[112 76 95 34 54] [ 57 71 44 73 85] [ 30 52 37 103 91] [ 23 82 72 61 65] [110 79 28 24 42]]  [[ 35 83 14 70 55] [ 68 86 17 20 125] [ 97 74 43 51 21] [ 11 106 120 31 1] [ 67 7 99 116 113]]  [[118 32 9 93 80] [ 12 19 102 100 58] [ 25 50 27 4 121] [ 40 111 81 22 26] [ 18 124 104 90 33]]  [[ 89 60 41 107 3] [ 98 2 64 53 63] [105 108 48 39 109] [ 6 122 46 78 92] [ 13 10 47 119 69]]]]
Jumlah populasi			200
Banyak iterasi			50
Nilai <i>objective function</i> yang dicapai			4484
Durasi proses pencarian			6.82 detik

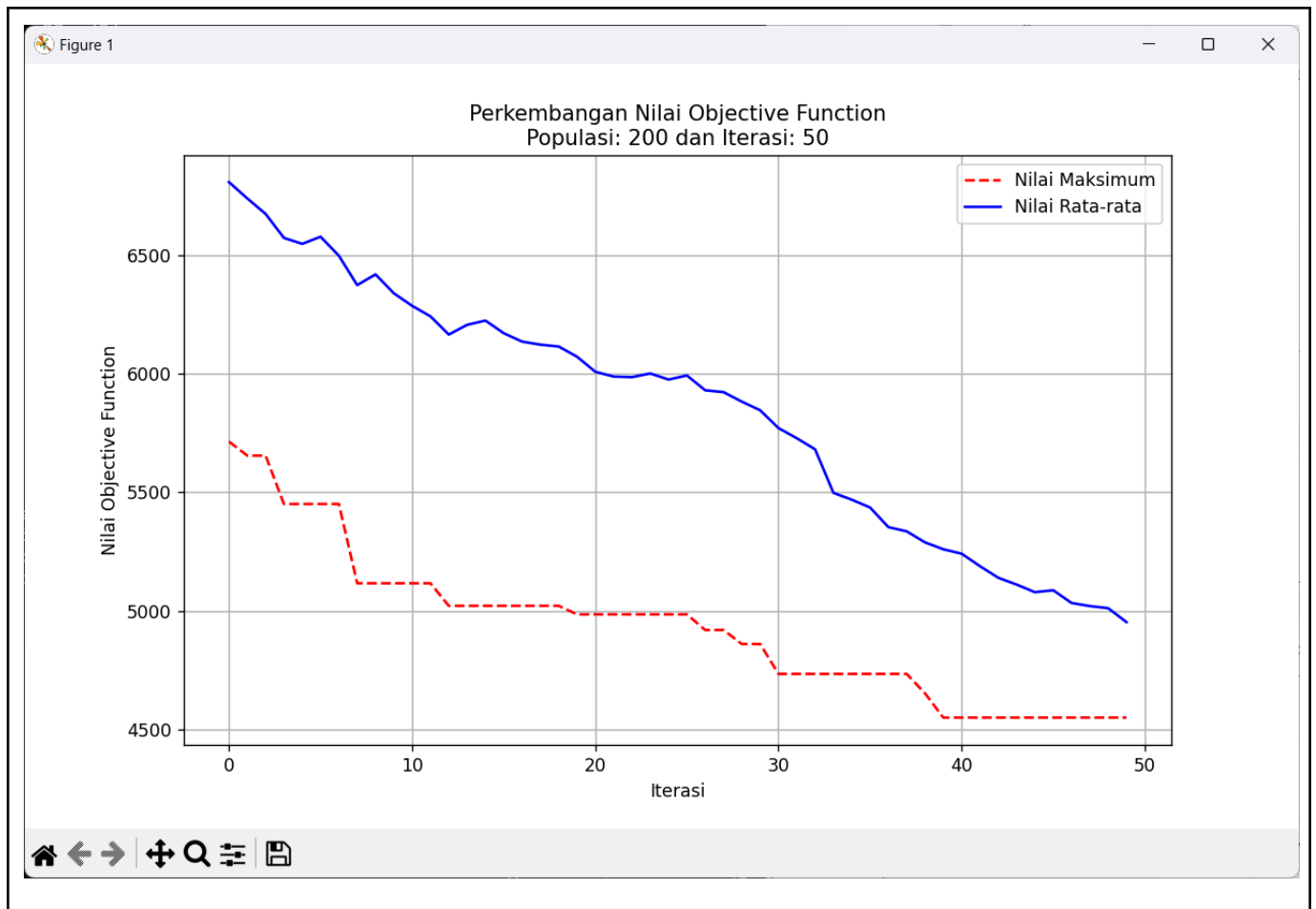
### Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati



### Eksperimen 3: Populasi 200 Iterasi 50

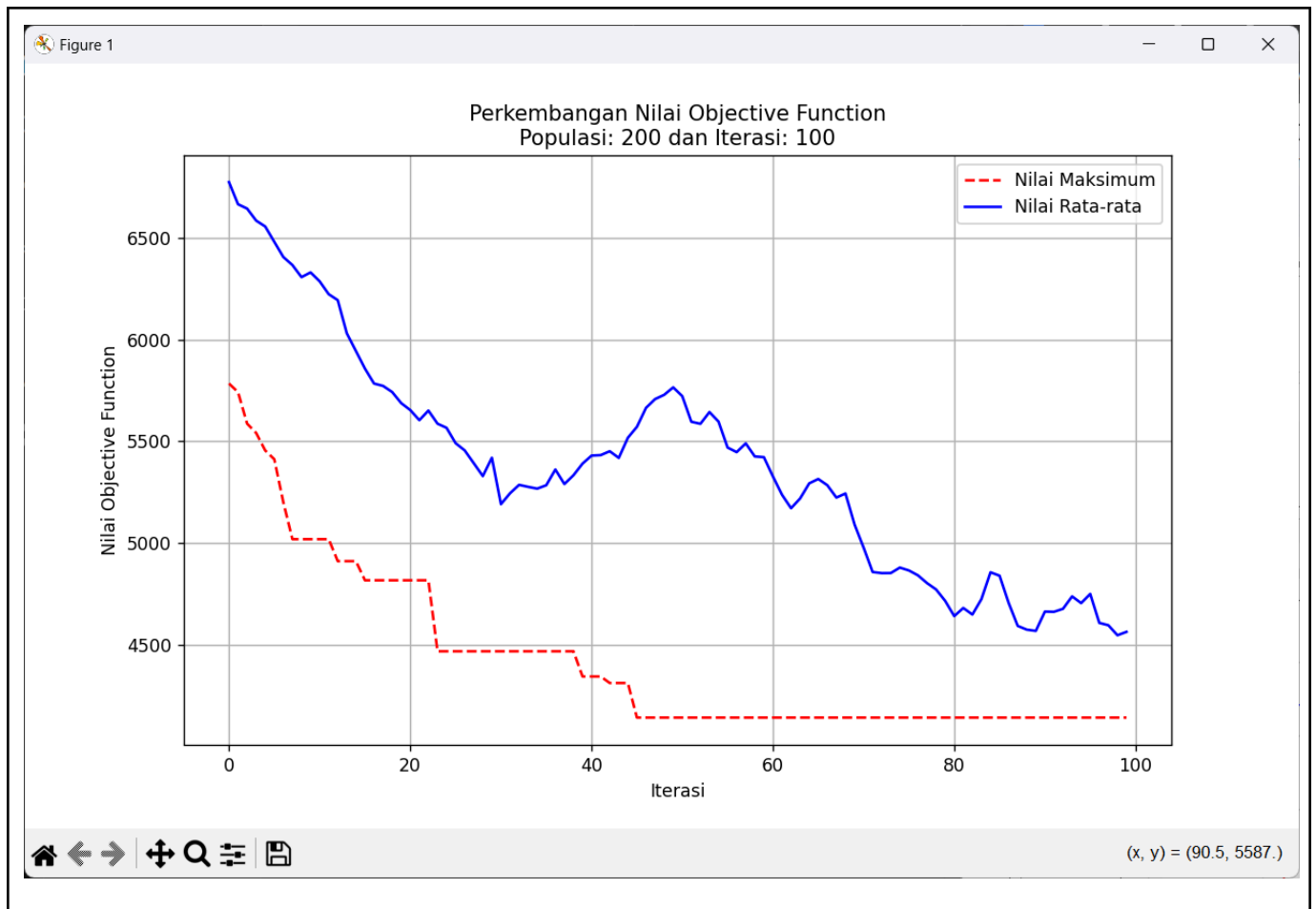
State awal		State akhir	
	[[[ 97 77 14 61 55]		[[[ 23 123 111 39 30]
	[ 19 111 93 112 105]		[ 21 84 97 50 75]
	[119 20 53 26 118]		[ 16 60 83 121 29]
	[ 36 86 17 100 56]		[105 82 69 15 77]
	[ 63 8 83 69 15]]		[ 64 42 80 90 44]]
	[[ 9 124 22 107 71]		[[124 3 38 62 68]
	[ 85 116 102 48 115]		[ 47 63 74 86 46]
	[ 10 110 62 113 98]		[ 49 87 28 102 101]
	[ 82 95 42 109 65]		[ 9 66 104 24 31]

	[ 91 13 81 120 108]]  [[ 3 52 18 45 84] [ 88 27 59 51 76] [ 94 46 24 104 68] [ 50 34 79 101 60] [ 58 11 123 43 33]]  [[ 67 106 66 99 29] [ 39 117 103 87 78] [ 57 74 70 32 89] [125 6 40 92 1] [ 4 5 54 30 21]]  [[ 28 12 122 41 96] [121 75 73 2 90] [ 38 64 72 35 49] [ 25 23 31 47 37] [ 7 16 80 114 44]]]]		[ 99 45 35 41 106]]  [[ 48 20 32 115 120] [108 6 94 10 5] [113 2 88 27 58] [ 11 25 91 89 118] [ 92 110 8 37 57]]  [[ 72 36 122 73 78] [114 81 4 61 96] [ 7 67 119 79 70] [ 33 93 34 76 40] [ 95 54 56 109 17]]  [[ 59 85 12 107 19] [ 13 65 51 53 125] [ 26 112 22 98 71] [ 14 103 43 117 1] [ 55 18 116 52 100]]]]
Jumlah populasi			200
Banyak iterasi			50
Nilai <i>objective function</i> yang dicapai			4551
Durasi proses pencarian			7.01 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



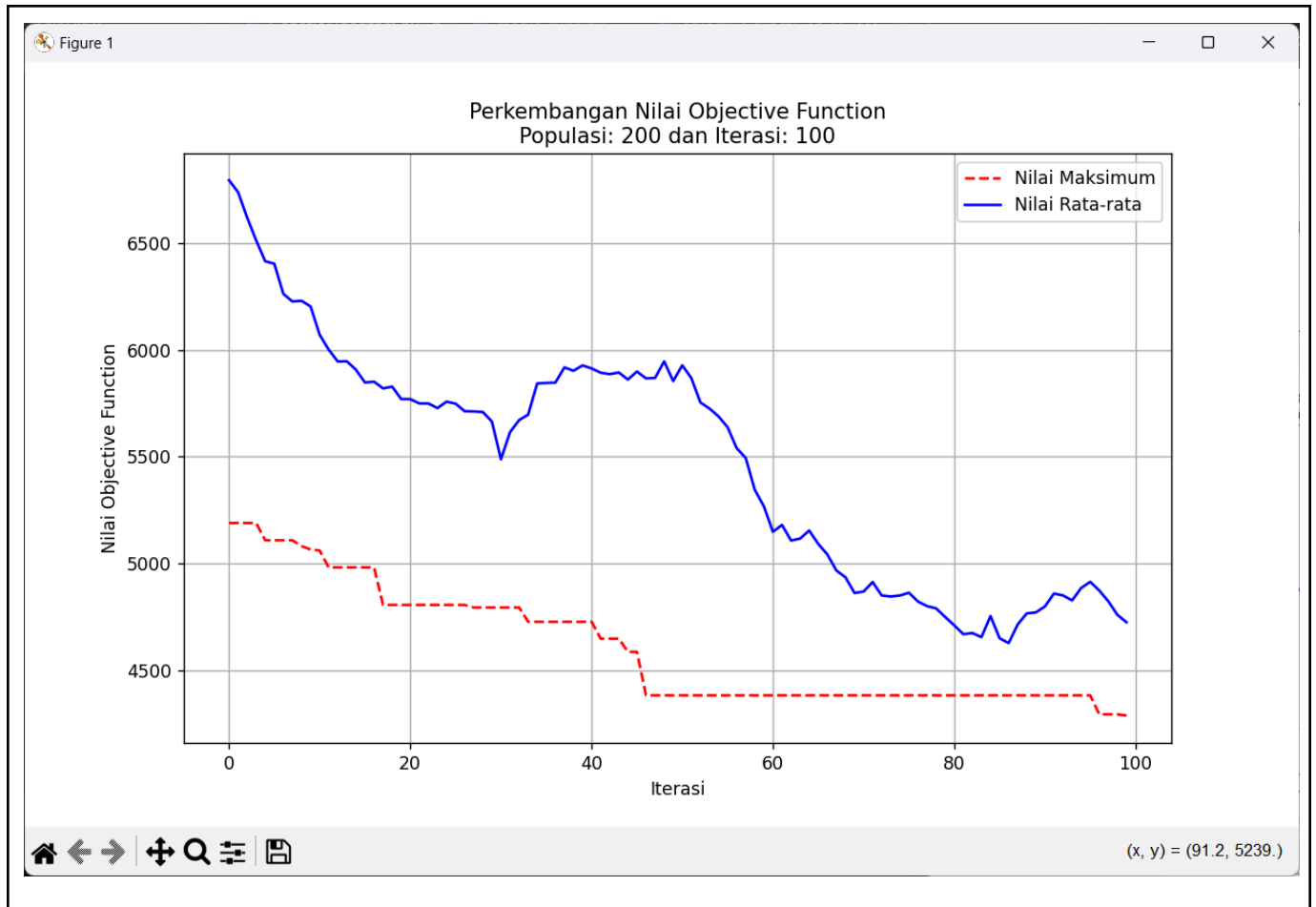
Eksperimen 4: Populasi 200 Iterasi 100			
<b>State awal</b>	<div>[[[ 57 93 20 67 42]</div> <div>[ 54 5 35 84 99]</div> <div>[ 38 49 72 123 26]</div> <div>[ 88 16 52 65 3]</div> <div>[ 66 61 112 11 121]]</div> <div>[[124 50 122 25 40]</div> <div>[ 9 74 17 27 108]</div> <div>[114 110 69 36 85]</div> <div>[116 48 117 87 46]</div> <div>[ 30 91 71 12 45]]</div>	<b>State akhir</b>	<div>[[[ 25 103 5 40 113]</div> <div>[123 50 125 24 16]</div> <div>[ 26 81 84 120 68]</div> <div>[ 93 33 86 63 41]</div> <div>[ 19 60 83 73 23]]</div> <div>[[121 58 119 29 87]</div> <div>[ 31 38 71 62 55]</div> <div>[111 66 11 47 89]</div> <div>[ 18 94 61 118 82]</div> <div>[ 59 110 72 56 2]]</div>

	[[ 15 32 102 70 104] [ 24 43 79 76 111] [ 37 83 62 63 18] [ 77 103 73 75 28] [ 81 1 13 106 101]]  [[119 125 14 47 23] [ 10 80 68 113 22] [120 82 21 95 92] [ 97 51 41 94 44] [ 60 33 31 64 115]]  [[ 89 53 34 6 105] [ 58 19 118 59 96] [ 4 107 78 98 55] [109 56 39 90 7] [ 29 100 86 8 2]]]		[[ 75 8 108 100 28] [ 37 106 30 49 88] [ 51 65 98 104 3] [ 52 97 15 13 122] [101 14 107 7 76]]  [[ 20 6 21 99 32] [124 69 9 17 105] [112 67 79 54 10] [115 53 85 95 12] [ 70 64 57 45 96]]  [[ 34 77 39 27 74] [ 91 43 90 22 109] [ 36 78 114 1 42] [102 4 92 44 35] [ 48 116 46 117 80]]]
Jumlah populasi			200
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4142
Durasi proses pencarian			13.80 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Eksperimen 5: Populasi 200 Iterasi 100			
<b>State awal</b>	<pre> [[[ 54 75 50 97 86]  [109 116 103 105 56]  [120 41 69 81 66]  [ 99 37 121 101 18]  [ 84 48 47 102 4]]  [[ 13 88 49 23 57]  [ 40 82 72 10 92]  [ 74  5 76 61 93]  [117 45 60 53 62]  [124 25 36 11 38]] </pre>	<b>State akhir</b>	<pre> [[[ 68 100 59 23 39]  [101 34 81 117 37]  [ 2 95 67 15 124]  [ 42 71 29 73 75]  [ 27  5 77 88 45]]  [[ 4 46 112 84 90]  [ 18 115 66 89 36]  [108 55 14 10 11]  [113 40 20 43 109]  [ 91 41  8 122 114]] </pre>

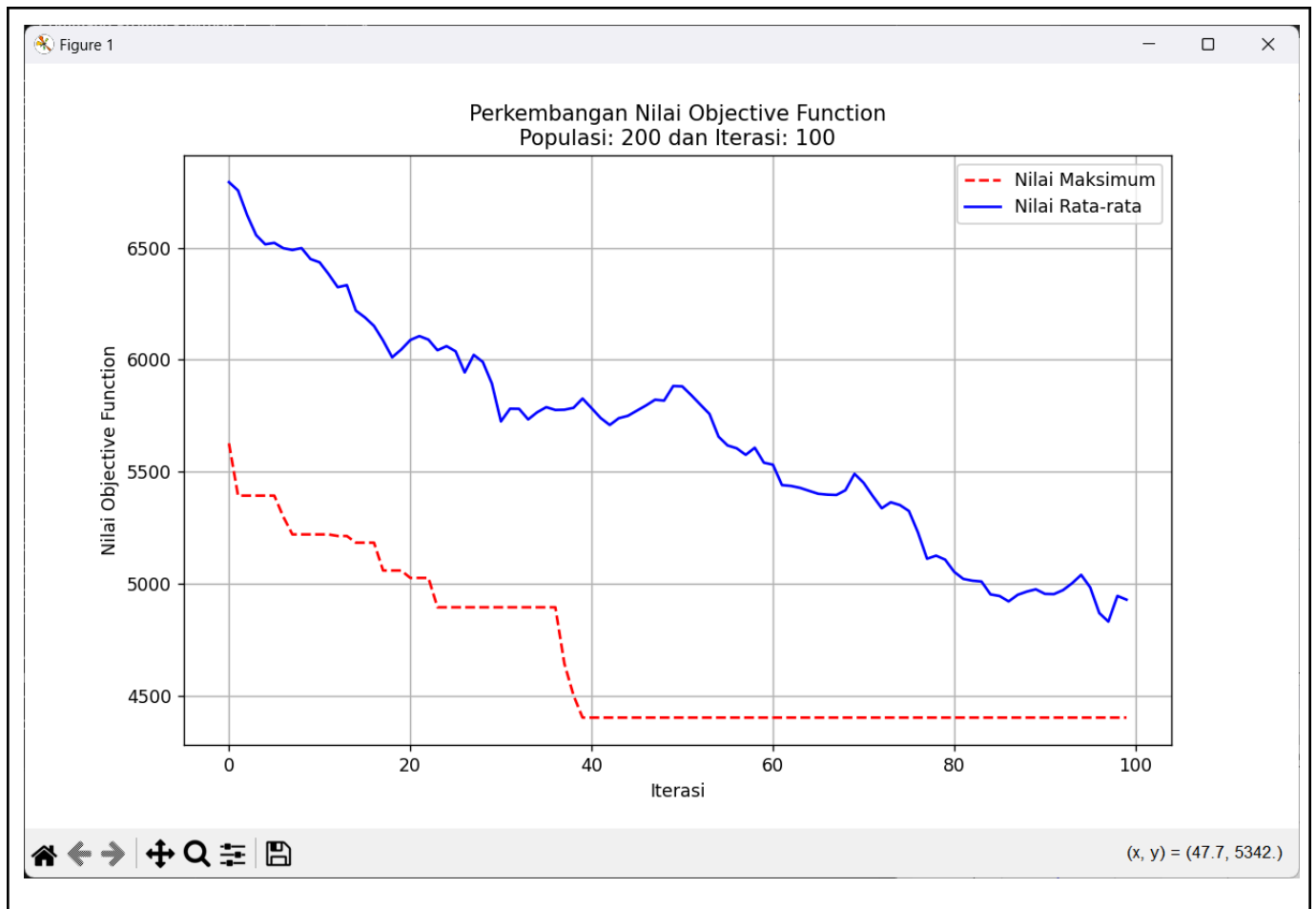
	[[ 7 19 67 108 32] [112 111 100 55 31] [114 44 51 115 39] [ 35 17 12 107 14] [ 24 71 78 73 106]]  [[ 30 85 22 119 52] [ 77 8 89 21 9] [ 46 104 20 65 59] [ 79 91 90 70 123] [ 95 42 1 118 43]]  [[ 98 125 27 26 63] [ 16 3 2 58 6] [ 33 110 94 29 80] [ 15 96 34 28 113] [ 64 68 122 87 83]]]		[[121 9 50 106 51] [ 56 76 24 7 107] [ 44 32 82 111 53] [ 22 120 12 97 65] [ 54 93 118 6 3]]  [[ 52 60 19 72 110] [ 64 98 31 30 33] [ 21 86 87 116 58] [105 74 92 63 104] [103 80 79 69 13]]  [[ 35 85 70 25 57] [123 96 1 26 16] [ 83 48 102 62 17] [ 99 38 125 49 94] [ 47 78 61 28 119]]]
Jumlah populasi			200
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4286
Durasi proses pencarian			13.70 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Eksperimen 6: Populasi 200 Iterasi 100			
<b>State awal</b>	<pre> [[[ 64 15 108 5 20]  [102 58 44 121 21]  [ 30 72 32 117 109]  [ 8 35 89 48 78]  [ 51 81 100 104 105]]  [[ 14 98 77 124 49]  [ 63 94 115 37 56]  [ 46 6 61 118 79]  [103 68 123 74 107]  [ 65 2 19 125 116]] </pre>	<b>State akhir</b>	<pre> [[[ 81 93 39 9 74]  [ 91 122 4 71 41]  [ 92 19 116 106 12]  [ 96 3 110 118 65]  [ 37 97 61 32 21]]  [[ 94 123 43 5 44]  [ 11 26 90 125 75]  [121 63 34 115 72]  [ 27 68 101 109 1]  [ 56 47 49 119 112]] </pre>

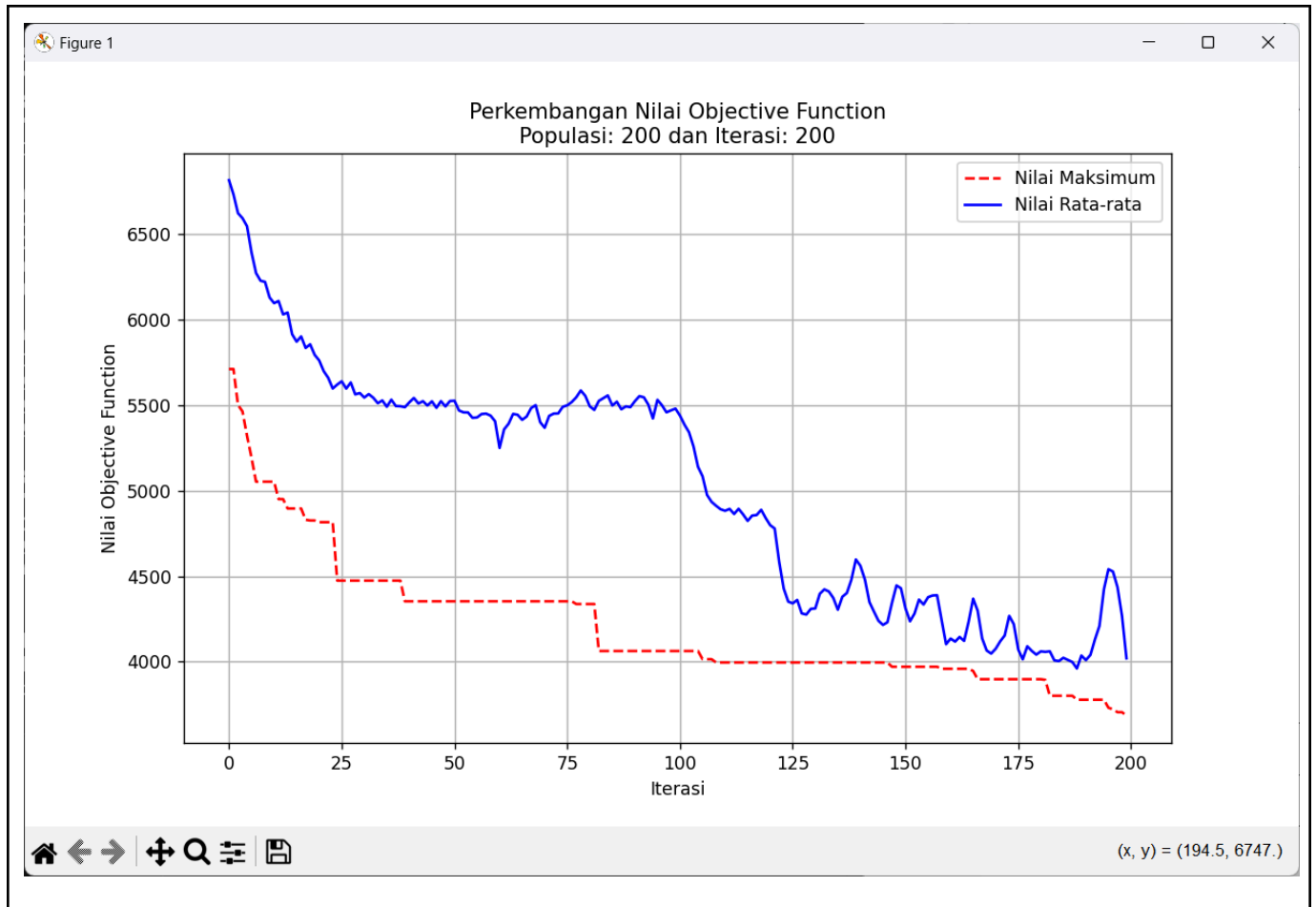


	[[ 55 86 91 95 47] [ 25 80 50 27 4] [112 101 83 53 28] [ 3 26 41 120 10] [ 93 70 67 119 33]]  [[ 43 106 52 11 85] [ 40 97 69 96 38] [ 75 88 92 36 66] [113 110 73 18 122] [ 13 42 12 17 59]]  [[ 29 60 114 34 82] [ 24 62 16 45 54] [ 90 9 22 31 39] [ 84 7 111 57 23] [ 87 71 76 99 1]]]		[[ 45 55 104 46 69] [120 53 54 88 10] [ 15 83 76 6 108] [ 67 82 24 42 38] [ 59 60 57 84 58]]  [[ 25 14 117 8 98] [103 33 107 40 80] [ 86 77 114 28 23] [ 87 30 100 105 2] [ 31 36 18 113 35]]  [[ 85 16 52 124 66] [ 51 17 89 50 111] [ 7 73 70 22 64] [ 29 95 48 99 62] [ 78 79 102 13 20]]]
Jumlah populasi			200
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4404
Durasi proses pencarian			13.91 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



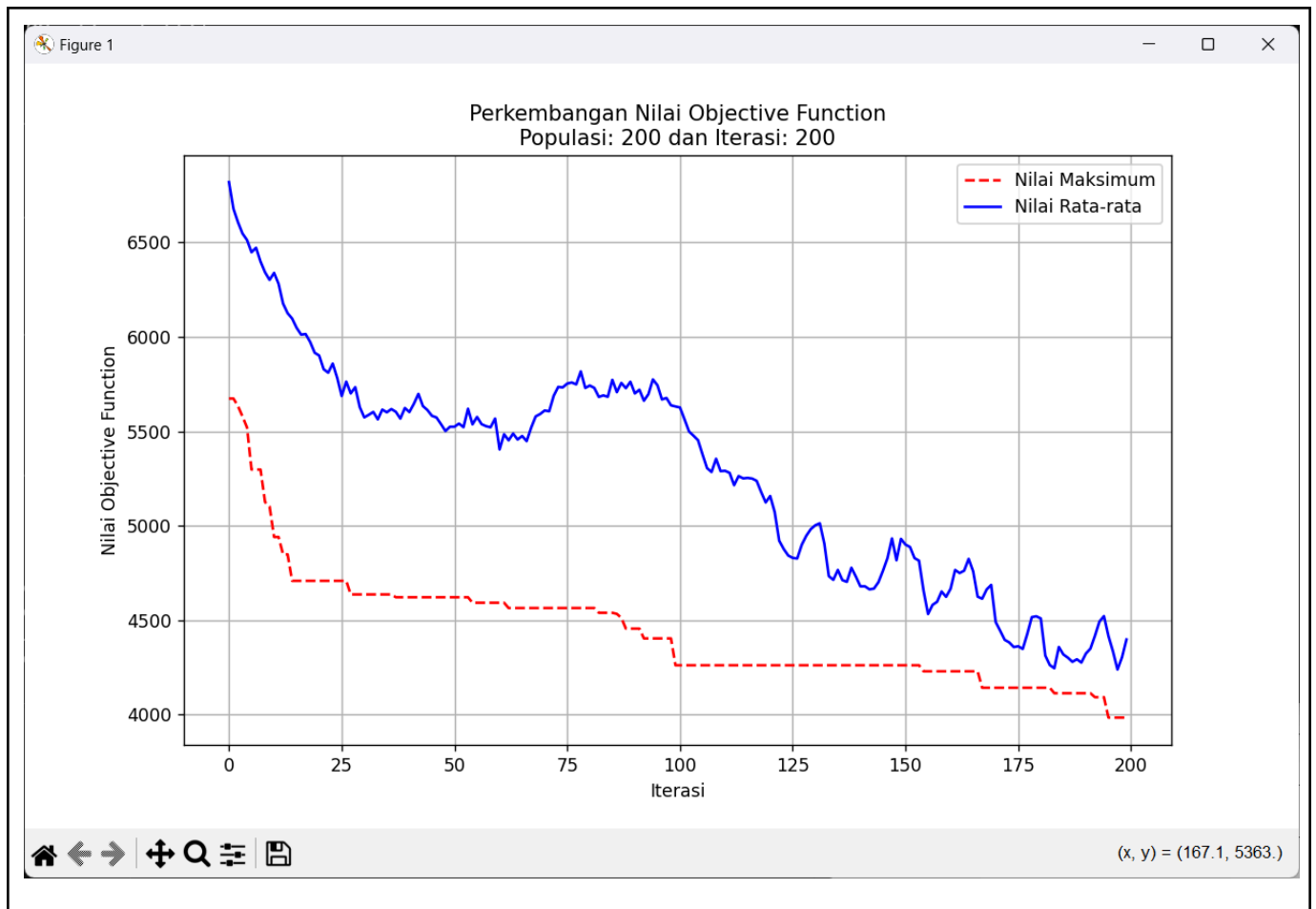
Eksperimen 7: Populasi 200 Iterasi 200			
<b>State awal</b>	<pre>[[[124 87 109 99 48]  [ 19 50 70 30 98]  [ 64 68 39 104 77]  [ 73 100 37 55 43]  [103 123 91 41 59]]   [[ 88  3 25 66 108]  [ 23 102 85 22 101]  [ 15 115 34 56 105]  [112 17 82 60 11]  [121 18 119  5 93]]</pre>	<b>State akhir</b>	<pre>[[[ 50 97 118 29 13]  [ 84 93  6 114 10]  [ 46 44 123  1 94]  [ 90 15 70  8 102]  [ 20 81 23 112 95]]   [[ 85 51 60 47 53]  [ 27 32 61 124 63]  [ 38 101 35 80 71]  [ 77 41 115 34 21]  [ 91 87 45 14 121]]</pre>

	[[ 9 33 40 4 92] [ 21 75 6 12 2] [ 61 125 53 24 76] [113 35 1 96 89] [111 16 63 110 120]]  [[ 79 58 65 97 26] [ 54 14 62 8 81] [117 94 38 29 13] [ 36 69 90 7 95] [122 106 31 67 72]]  [[ 46 74 118 44 32] [ 27 20 78 28 42] [107 71 84 10 57] [ 47 80 83 52 49] [ 86 51 45 116 114]]]		[[ 54 43 96 31 100] [ 42 68 62 26 116] [ 92 73 58 49 78] [106 99 2 117 11] [ 30 22 119 113 16]]  [[ 64 75 25 98 7] [111 24 36 48 103] [ 89 52 86 66 40] [ 19 17 59 82 120] [109 122 3 56 28]]  [[ 88 74 9 33 125] [ 57 69 108 18 107] [ 72 105 37 76 12] [ 79 83 5 67 65] [104 55 39 110 4]]]
Jumlah populasi			200
Banyak iterasi			200
Nilai <i>objective function</i> yang dicapai			3684
Durasi proses pencarian			27.71 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



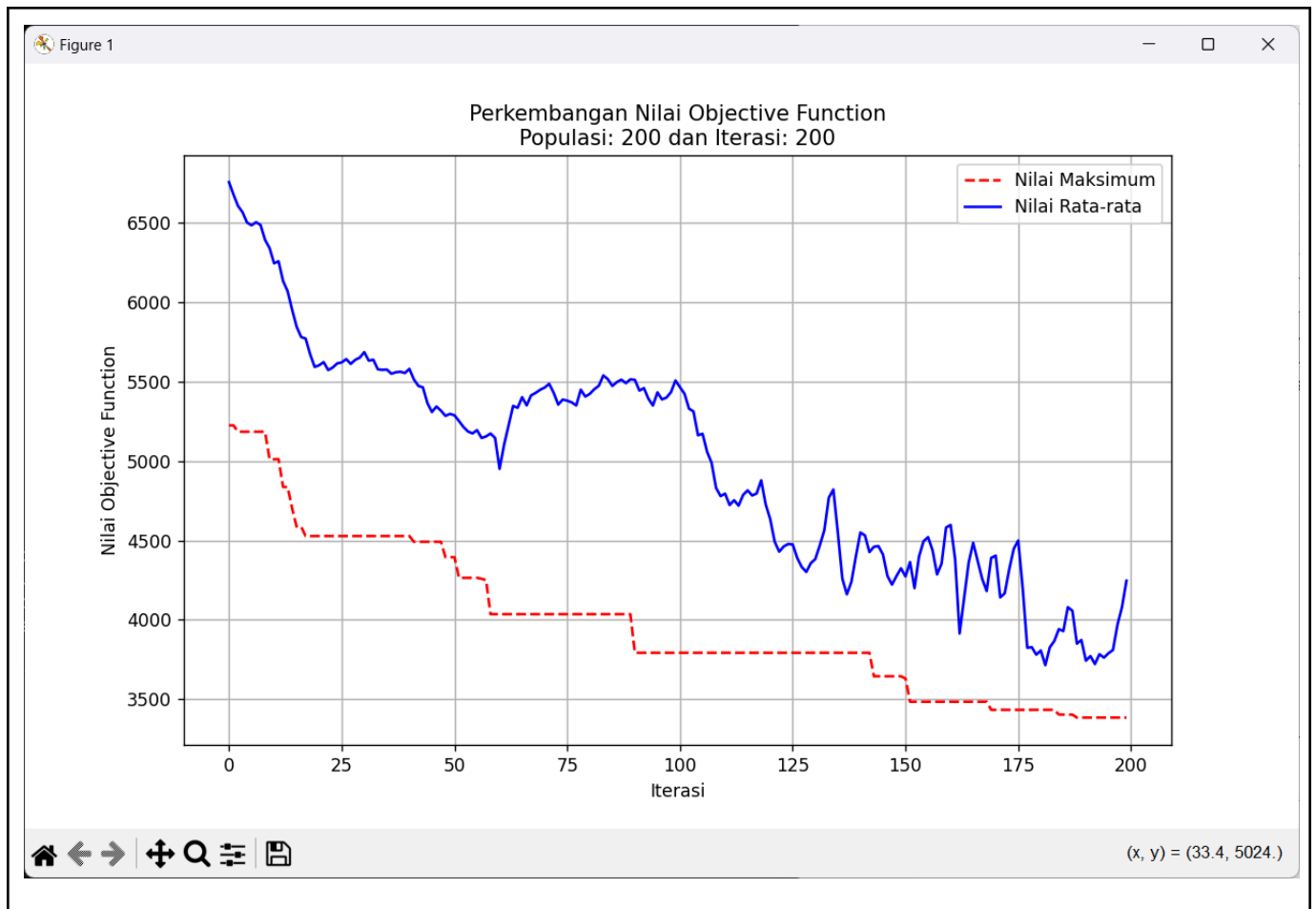
Eksperimen 8: Populasi 200 Iterasi 200			
<b>State awal</b>	<pre>[[[ 80 46 124 111 102] [122 15 107 60 49] [ 7 27 56 38 114] [101 17 23 120 14] [ 47 115 86 62 51]]  [[109 32 100 104 83] [116 77 61 96 26] [ 72 91 8 20 36] [103 70 50 6 93] [119 66 108 39 67]]</pre>	<b>State akhir</b>	<pre>[[[123 5 90 39 57] [ 73 69 14 121 36] [ 80 107 65 55 16] [ 2 75 93 64 96] [ 63 48 61 34 112]]  [[114 86 17 1 117] [ 49 29 72 81 62] [ 84 33 27 110 70] [ 28 67 99 32 60] [ 40 122 104 22 21]]</pre>

	[[ 35 63 69 44 106] [ 84 59 76 25 12] [ 52 79 95 68 110] [ 37 1 88 123 9] [ 29 54 3 71 64]]  [[ 65 75 18 31 57] [ 11 90 121 82 73] [ 24 43 112 42 118] [ 4 113 105 28 98] [ 55 87 41 33 10]]  [[ 16 125 92 13 2] [ 81 21 5 22 89] [ 48 30 74 78 58] [ 85 40 117 97 34] [ 94 45 19 53 99]]]		[[ 46 10 95 124 30] [ 38 42 100 4 118] [ 19 119 56 68 50] [120 88 11 66 78] [125 18 23 53 94]]  [[ 9 45 41 102 24] [111 106 82 74 47] [ 79 6 97 7 108] [ 77 26 76 31 89] [ 35 98 113 85 92]]  [[ 43 13 52 105 83] [ 8 109 71 37 25] [ 58 51 12 44 115] [103 59 116 91 20] [ 54 3 15 101 87]]]
Jumlah populasi			200
Banyak iterasi			200
Nilai <i>objective function</i> yang dicapai			3985
Durasi proses pencarian			27.89 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Eksperimen 9: Populasi 200 Iterasi 200			
<b>State awal</b>	<pre>[[[ 80 46 124 111 102] [122 15 107 60 49] [ 7 27 56 38 114] [101 17 23 120 14] [ 47 115 86 62 51]]  [[109 32 100 104 83] [116 77 61 96 26] [ 72 91 8 20 36] [103 70 50 6 93] [119 66 108 39 67]]</pre>	<b>State akhir</b>	<pre>[[[ 46 59 93 14 112] [107 13 118 31 63] [ 65 100 66 58 44] [ 86 67 11 123 33] [ 23 87 38 74 90]]  [[ 10 110 116 91 4] [ 29 98 30 122 45] [125 35 40 1 106] [120 7 51 95 39] [ 27 83 70 2 117]]</pre>

	[[ 35 63 69 44 106] [ 84 59 76 25 12] [ 52 79 95 68 110] [ 37 1 88 123 9] [ 29 54 3 71 64]]  [[ 65 75 18 31 57] [ 11 90 121 82 73] [ 24 43 112 42 118] [ 4 113 105 28 98] [ 55 87 41 33 10]]  [[ 16 125 92 13 2] [ 81 21 5 22 89] [ 48 30 74 78 58] [ 85 40 117 97 34] [ 94 45 19 53 99]]]		[[115 97 28 60 34] [ 78 73 16 17 114] [ 3 72 89 113 42] [ 20 103 109 19 124] [ 56 9 94 105 18]]  [[ 79 24 15 52 96] [104 49 99 76 37] [ 69 36 68 82 80] [ 54 48 64 55 71] [ 62 102 111 101 12]]  [[ 81 77 25 108 53] [ 61 84 50 85 43] [ 41 6 32 92 47] [ 26 75 121 5 88] [119 22 21 8 57]]]
Jumlah populasi			200
Banyak iterasi			200
Nilai <i>objective function</i> yang dicapai			3385
Durasi proses pencarian			28.27 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			

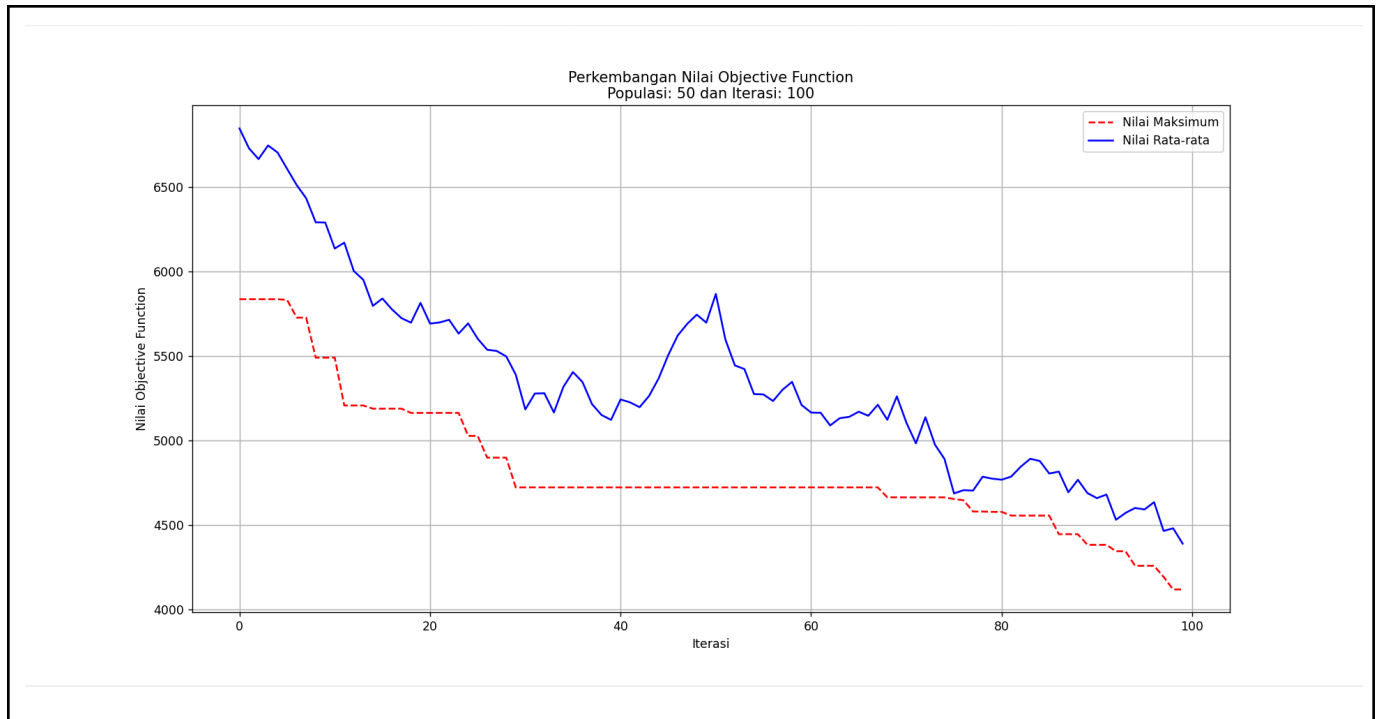


### 2.3.6.2. Jumlah iterasi sebagai kontrol

Eksperimen 1 dengan Jumlah Populasi 50 Jumlah Iterasi 100			
<b>State awal</b>	<div>[[[108 99 122 51 69]</div> <div>[ 26 53 94 2 93]</div> <div>[ 14 79 109 75 85]</div> <div>[ 70 104 15 59 100]</div> <div>[ 42 90 73 32 39]]</div> <div>[[123 13 98 8 92]</div> <div>[121 45 81 33 64]</div> <div>[ 91 105 89 56 40]</div>	<b>State akhir</b>	<div>[[[ 64 118 82 15 59]</div> <div>[114 55 53 4 51]</div> <div>[ 12 113 71 94 57]</div> <div>[ 21 85 96 124 84]</div> <div>[ 83 56 19 50 69]]</div> <div>[[ 45 33 112 80 60]</div> <div>[ 34 92 90 68 62]</div> <div>[ 49 44 72 54 106]</div>

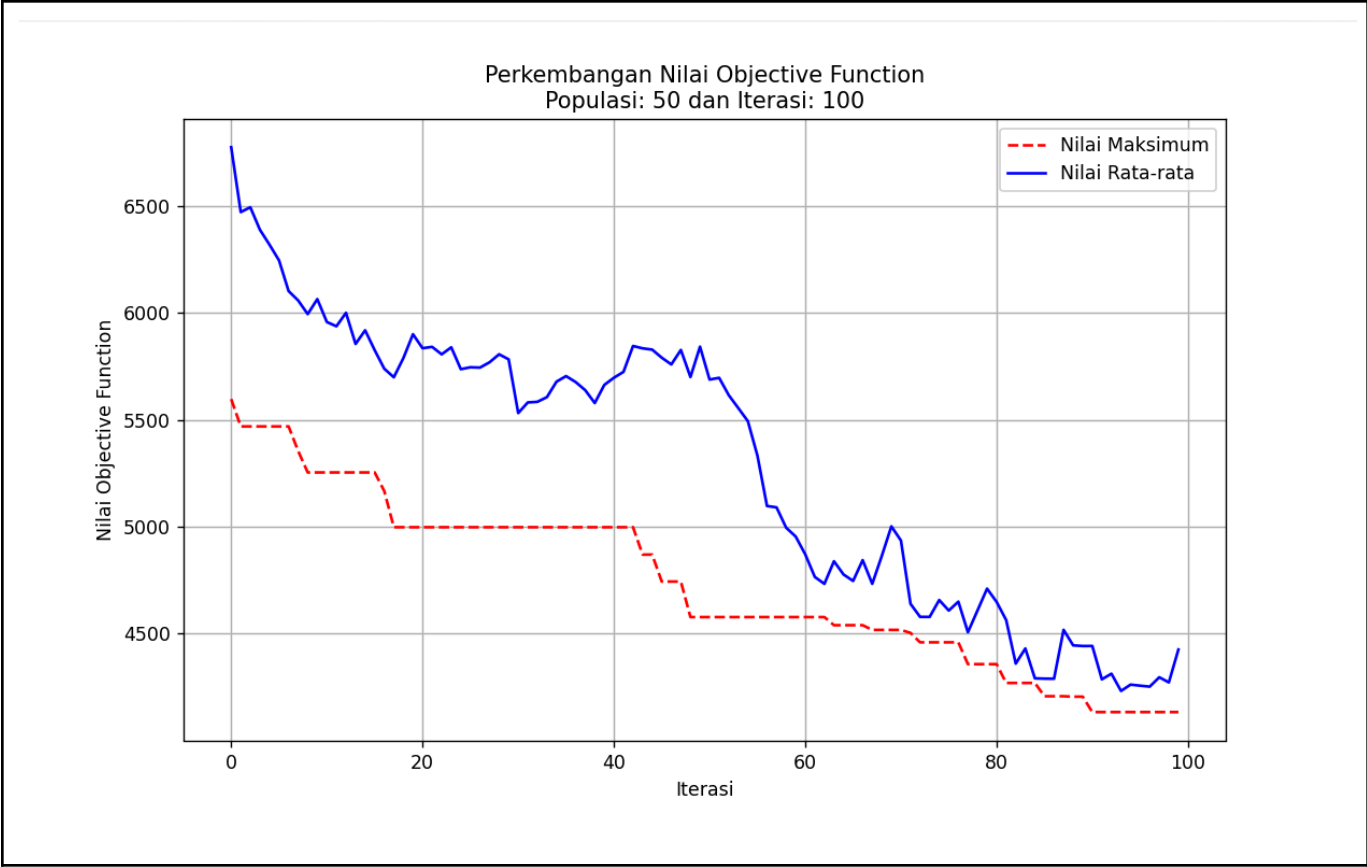


	[ 63 11 65 76 48] [106 67 24 47 82]]  [[ 86 50 120 111 36] [ 38 124 5 46 27] [102 119 71 57 112] [ 87 6 4 17 58] [ 44 68 52 66 74]]  [[115 43 21 41 116] [ 20 101 61 10 3] [ 23 35 80 31 1] [ 37 83 72 88 62] [110 117 103 84 34]]  [[107 95 19 113 49] [ 55 125 29 96 30] [ 28 77 7 114 78] [ 97 16 118 12 54] [ 25 18 60 22 9]]]		[ 81 20 91 76 14] [104 97 6 27 41]]  [[105 18 31 110 58] [ 17 38 70 116 42] [ 63 28 79 103 30] [ 39 86 74 13 87] [ 2 120 48 23 75]]  [[ 9 73 88 40 108] [ 65 36 16 10 111] [ 95 7 100 122 52] [115 66 3 101 35] [ 37 109 117 89 32]]  [[ 24 61 102 46 22] [119 93 1 107 43] [ 98 8 47 5 123] [121 78 26 11 99] [ 67 25 125 77 29]]]
Jumlah populasi			50
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4118
Durasi proses pencarian			8.16 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



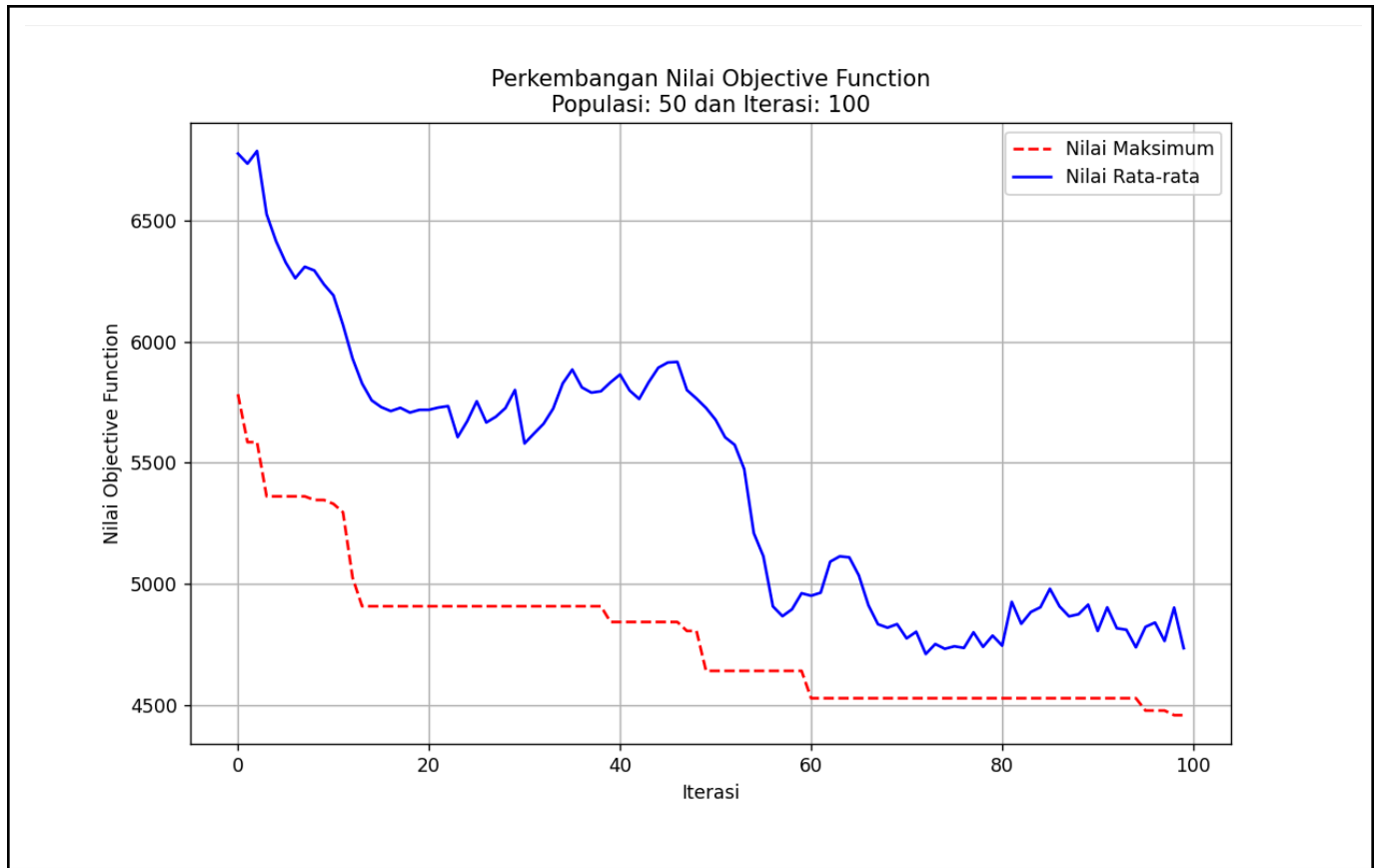
Eksperimen 2 dengan Jumlah Populasi 50 Jumlah Iterasi 100			
<b>State awal</b>	<div>[[[ 55 14 29 3 111]</div> <div>[ 93 121 64 90 84]</div> <div>[ 15 96 114 4 70]</div> <div>[ 16 60 45 2 98]</div> <div>[ 71 82 109 7 88]]</div> <div>[[124 13 59 103 68]</div> <div>[112 80 48 30 105]</div> <div>[102 83 17 76 81]</div> <div>[ 20 123 122 19 72]</div> <div>[119 6 39 51 26]]</div> <div>[[ 58 24 44 10 63]</div> <div>[ 73 49 35 32 85]</div> <div>[ 92 50 57 77 46]</div> <div>[ 87 33 108 100 91]</div>	<b>State akhir</b>	<div>[[[ 86 69 113 19 45]</div> <div>[119 97 89 10 56]</div> <div>[ 22 103 20 73 100]</div> <div>[ 30 37 111 77 82]</div> <div>[ 98 12 11 104 65]]</div> <div>[[ 26 79 32 121 43]</div> <div>[ 68 63 14 124 41]</div> <div>[ 96 106 115 34 13]</div> <div>[ 6 85 71 3 123]</div> <div>[ 58 15 91 21 120]]</div> <div>[[ 42 93 33 67 92]</div> <div>[ 49 36 88 46 84]</div> <div>[ 61 25 59 16 8]</div> <div>[ 99 18 24 112 125]</div>

	[ 34 110 106 28 54]]  [[ 56 38 12 99 113] [ 89 23 116 115 97] [ 1 9 22 62 47] [104 75 65 8 36] [ 31 21 11 117 67]]  [[120 78 66 74 95] [ 61 25 94 37 101] [107 125 41 18 42] [ 79 40 118 5 52] [ 86 27 69 43 53]]]		[ 35 122 108 87 5]]  [[ 38 1 114 101 50] [ 47 57 51 17 118] [ 52 2 94 117 48] [ 72 110 29 55 75] [ 4 54 31 23 90]]  [[105 27 28 44 83] [ 80 9 95 81 40] [107 76 53 60 64] [109 70 116 74 7] [ 39 66 62 102 78]]]
Jumlah populasi			50
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4134
Durasi proses pencarian			8.86 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



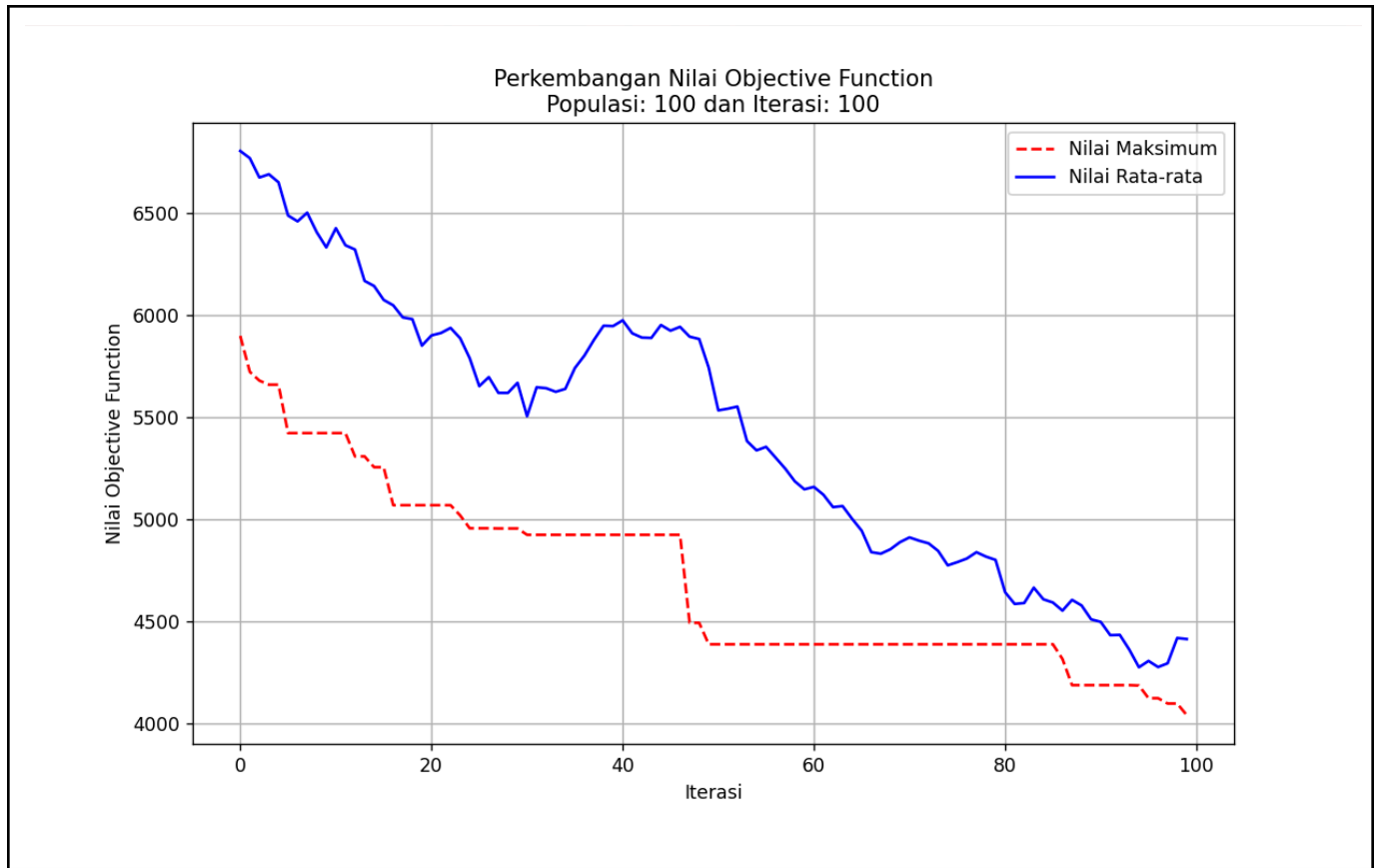
Eksperimen 3 dengan Jumlah Populasi 50 Jumlah Iterasi 100			
State awal	[[[103 115 106 75 13] [ 24 51 119 92 87] [125 16 20 39 21] [ 41 19 29 99 116] [ 47 94 63 67 27]]	State akhir	[[[103 60 12 97 38] [ 11 17 49 13 120] [ 18 118 99 37 63] [ 62 48 122 57 40] [ 95 107 6 58 39]]
	[[[108 77 34 74 102] [ 91 71 101 110 107] [ 10 33 89 6 23] [123 7 73 37 93] [ 98 80 105 32 79]]		[[[ 16 20 14 89 78] [112 106 47 3 56] [ 1 74 10 91 65] [101 35 113 108 42] [123 59 26 15 88]]
	[[ 49 52 60 59 113]		[[125 29 114 36 119]

	[120 4 68 53 38] [ 3 88 22 72 57] [76 70 64 86 81] [ 14 54 40 35 44]]  [[ 17 9 66 48 78] [ 58 25 100 95 69] [ 90 65 61 31 114] [ 43 124 82 111 85] [ 96 62 15 2 104]]  [[118 112 8 36 26] [121 45 97 18 84] [117 11 5 55 83] [ 28 1 122 42 50] [ 56 46 12 30 109]]]		[ 41 66 69 116 34] [ 32 82 85 8 109] [ 81 43 23 54 83] [ 31 96 87 93 22]]  [[ 7 80 73 2 105] [ 46 50 102 33 71] [121 21 9 64 25] [ 75 52 79 55 19] [ 67 90 68 72 92]]  [[ 5 117 110 104 45] [ 84 76 53 115 86] [111 27 24 100 51] [ 30 94 44 70 61] [ 98 28 77 4 124]]]
Jumlah populasi			50
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4457
Durasi proses pencarian			8.57 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Eksperimen 1 dengan Jumlah Populasi 100 Jumlah Iterasi 100			
<b>State awal</b>	<div>[[[ 30 119 31 9 125]</div> <div>[ 32 51 64 102 36]</div> <div>[ 23 84 22 20 61]</div> <div>[ 48 19 3 87 78]</div> <div>[108 55 16 33 1]]</div> <div>[[ 90 89 107 17 29]</div> <div>[ 86 37 53 66 77]</div> <div>[124 100 47 121 2]</div> <div>[ 44 34 118 54 104]</div> <div>[ 50 58 113 57 41]]</div>	<b>State akhir</b>	<div>[[[ 98 49 5 77 125]</div> <div>[ 84 122 39 20 23]</div> <div>[ 41 93 68 106 72]</div> <div>[ 83 4 109 9 12]</div> <div>[ 19 62 70 104 59]]</div> <div>[[ 2 11 116 73 112]</div> <div>[ 66 25 99 65 78]</div> <div>[ 32 58 53 17 108]</div> <div>[ 87 89 1 120 7]</div> <div>[ 30 67 118 8 103]]</div>

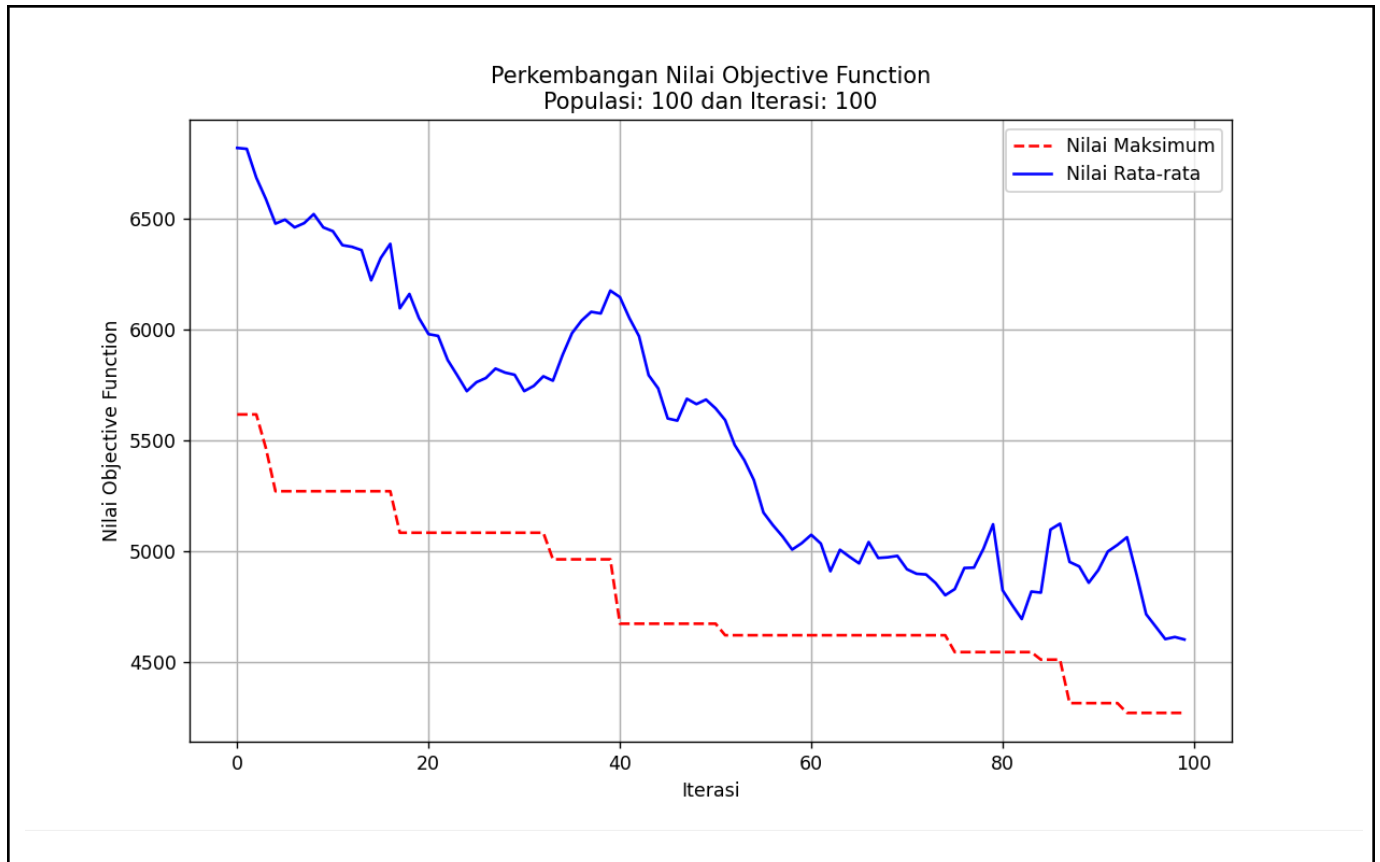
	[[ 83 97 74 114 7] [ 92 24 99 65 72] [ 81 94 39 112 75] [ 63 10 82 68 116] [111 93 6 59 69]]  [[109 46 62 43 38] [120 42 67 15 52] [ 26 25 73 11 8] [ 27 105 21 40 70] [ 12 45 96 85 88]]  [[ 98 115 13 91 80] [ 28 49 123 56 117] [ 95 14 18 122 101] [ 71 79 103 76 110] [ 60 35 4 5 106]]]		[[114 119 40 31 21] [ 15 54 45 97 102] [113 64 75 43 34] [ 35 24 111 28 117] [ 90 82 22 115 3]]  [[107 16 85 74 6] [ 55 71 100 94 13] [ 26 42 44 88 27] [ 57 79 36 47 95] [ 69 56 60 110 61]]  [[ 52 33 10 29 76] [124 46 80 63 96] [ 86 50 81 48 51] [ 37 121 18 123 92] [ 14 101 105 38 91]]]
Jumlah populasi			100
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4043
Durasi proses pencarian			16.83 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Eksperimen 2 dengan Jumlah Populasi 100 Jumlah Iterasi 100			
<b>State awal</b>	<div>[[[ 79 1 40 75 69]</div> <div>[105 50 80 26 8]</div> <div>[ 10 32 86 103 121]</div> <div>[ 94 93 36 52 47]</div> <div>[120 90 31 104 73]]</div> <div>[[ 74 25 44 95 37]</div> <div>[102 38 42 20 34]</div> <div>[109 97 15 17 33]</div> <div>[ 23 107 111 35 4]</div> <div>[ 19 51 43 124 68]]</div>	<b>State akhir</b>	<div>[[[ 67 108 3 88 74]</div> <div>[ 39 85 36 84 44]</div> <div>[116 93 10 7 78]</div> <div>[ 34 90 102 98 109]</div> <div>[ 41 33 114 20 107]]</div> <div>[[ 21 11 119 87 125]</div> <div>[113 123 17 12 59]</div> <div>[110 43 81 47 15]</div> <div>[ 79 105 5 30 73]</div> <div>[ 2 23 122 99 62]]</div>

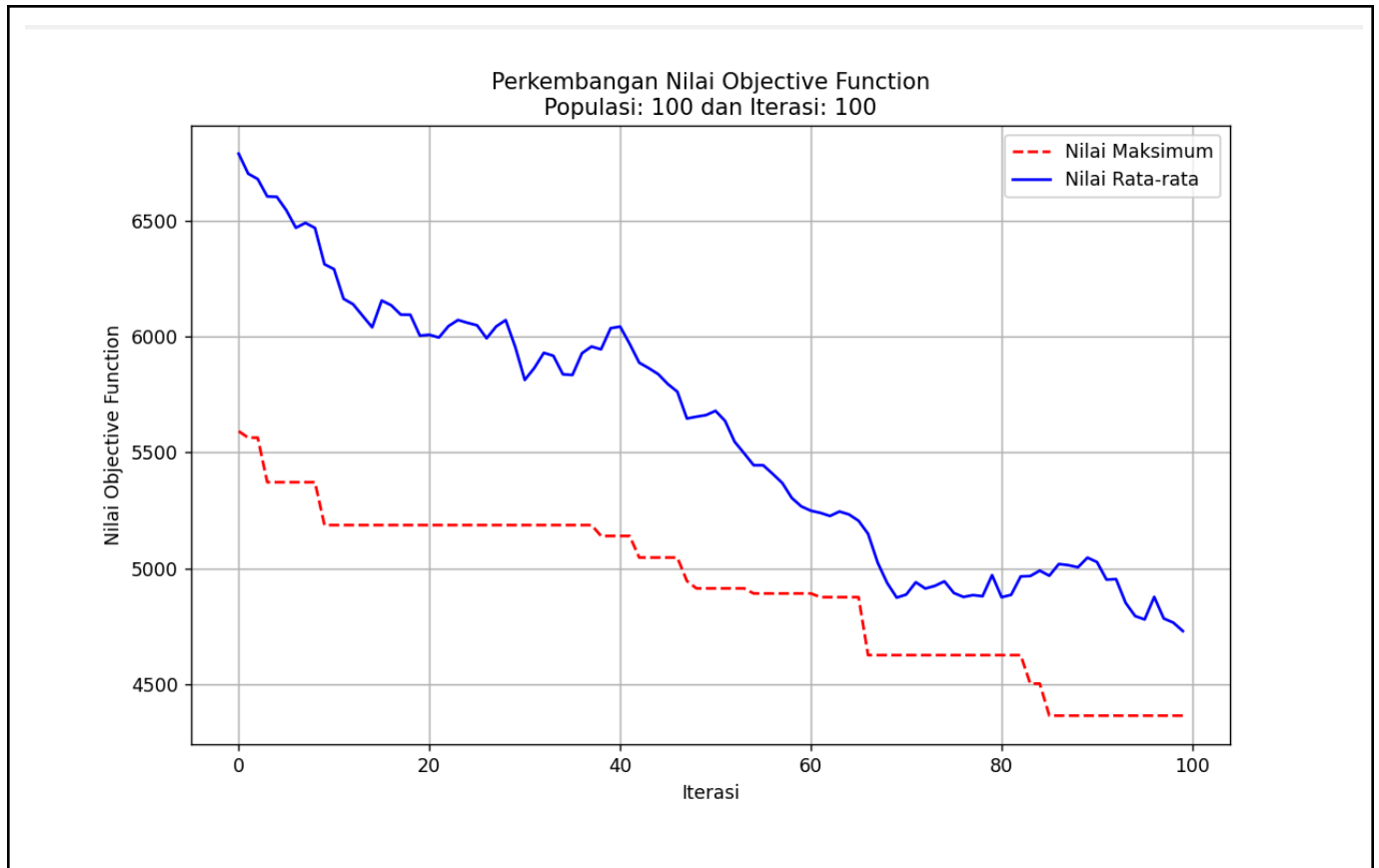


	[[ 55 112 89 114 70] [ 6 72 99 106 59] [ 2 13 18 22 119] [ 87 63 60 61 56] [108 66 49 46 100]]  [[ 58 57 16 3 85] [ 48 110 92 82 122] [ 28 123 96 41 71] [ 77 30 84 54 116] [ 91 81 101 12 29]]  [[ 83 7 64 65 98] [ 5 21 118 9 11] [ 45 27 125 117 88] [ 14 113 76 62 39] [ 53 78 67 115 24]]]		[[ 54 80 115 40 25] [ 13 94 19 49 95] [ 9 63 76 104 35] [118 29 112 16 70] [101 64 6 82 56]]  [[ 89 58 26 111 121] [124 37 96 92 75] [ 42 57 69 28 120] [ 4 1 46 61 24] [106 55 91 31 60]]  [[ 65 77 22 100 8] [ 72 32 71 117 27] [ 50 53 52 86 45] [ 38 66 68 18 97] [ 48 14 83 51 103]]]
Jumlah populasi			100
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4271
Durasi proses pencarian			22.03 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



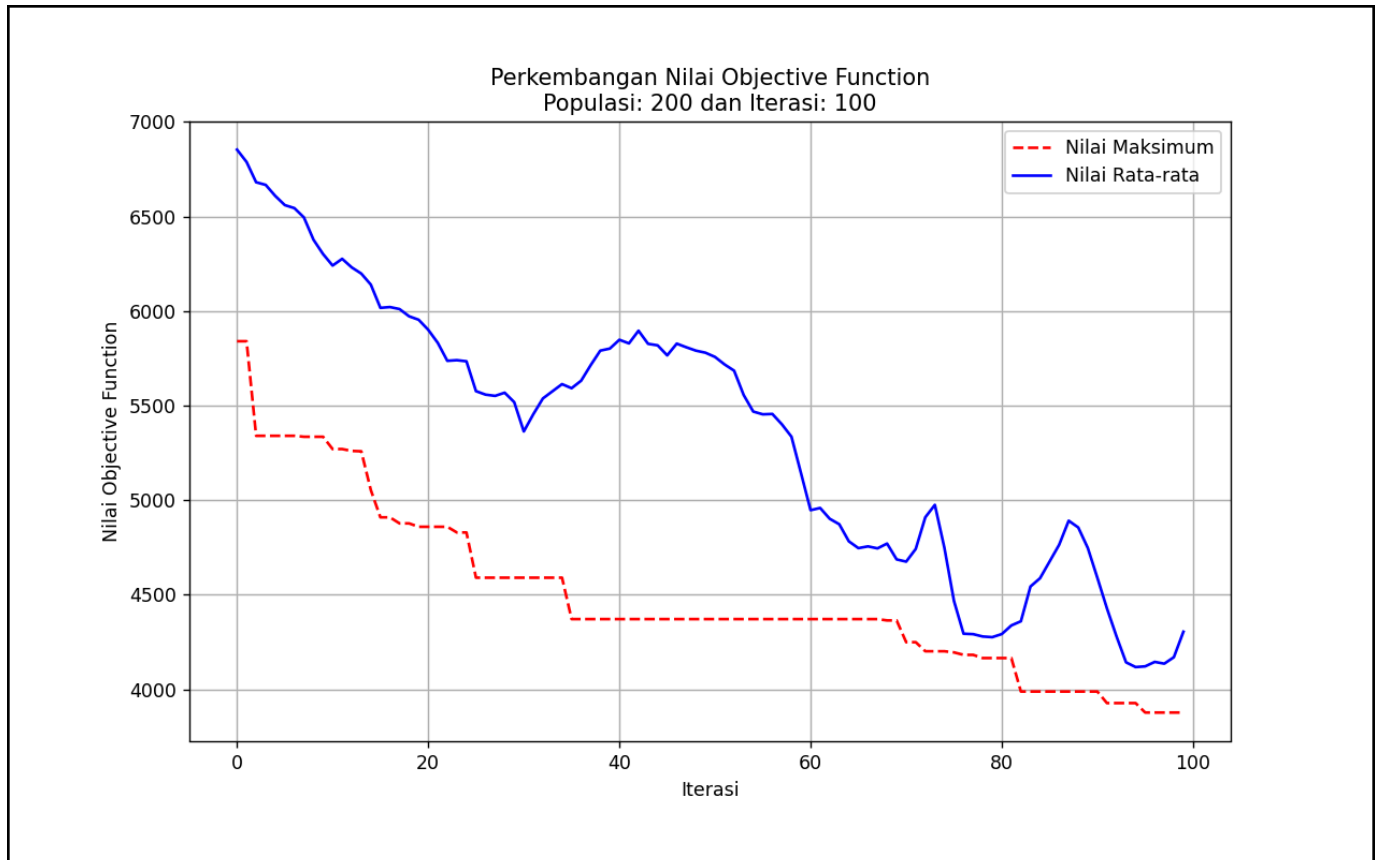
Eksperimen 3 dengan Jumlah Populasi 100 Jumlah Iterasi 100			
<b>State awal</b>	<div>[[[ 73 54 10 35 9]</div> <div>[ 88 90 104 1 68]</div> <div>[114 95 15 47 13]</div> <div>[ 46 27 106 62 110]</div> <div>[ 77 18 22 99 55]]</div> <div>[[ 23 61 74 60 3]</div> <div>[111 112 70 7 67]</div> <div>[ 8 45 100 79 29]</div> <div>[ 57 121 123 32 14]</div> <div>[125 40 108 16 11]]</div>	<b>State akhir</b>	<div>[[[ 5 24 39 99 74]</div> <div>[110 40 2 96 34]</div> <div>[ 29 63 75 48 62]</div> <div>[103 60 104 67 37]</div> <div>[ 59 124 86 20 122]]</div> <div>[[ 80 66 111 36 56]</div> <div>[ 54 121 81 42 76]</div> <div>[ 87 38 94 72 15]</div> <div>[ 18 44 51 70 106]</div> <div>[ 92 35 58 118 9]]</div>

	[[ 48 89 12 37 109] [117 69 85 115 21] [ 98 64 65 103 119] [ 34 105 36 56 96] [ 30 38 97 52 101]]  [[ 71 31 51 86 87] [ 50 116 26 118 75] [ 25 19 53 113 2] [ 63 33 17 4 6] [ 80 124 81 84 91]]  [[ 44 94 92 41 72] [122 42 76 120 102] [ 66 93 49 43 107] [ 39 59 20 78 28] [ 58 24 83 5 82]]]		[[120 101 119 11 46] [ 12 45 47 82 114] [ 65 90 21 84 57] [ 33 64 105 116 25] [107 17 53 115 32]]  [[ 50 112 10 91 4] [ 77 8 7 79 30] [ 68 95 109 31 52] [ 71 43 27 89 117] [123 1 23 28 78]]  [[ 93 26 85 3 102] [ 13 69 108 41 73] [ 55 49 22 125 97] [ 88 100 14 98 16] [ 61 113 83 6 19]]]
Jumlah populasi			100
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4364
Durasi proses pencarian			22.98 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



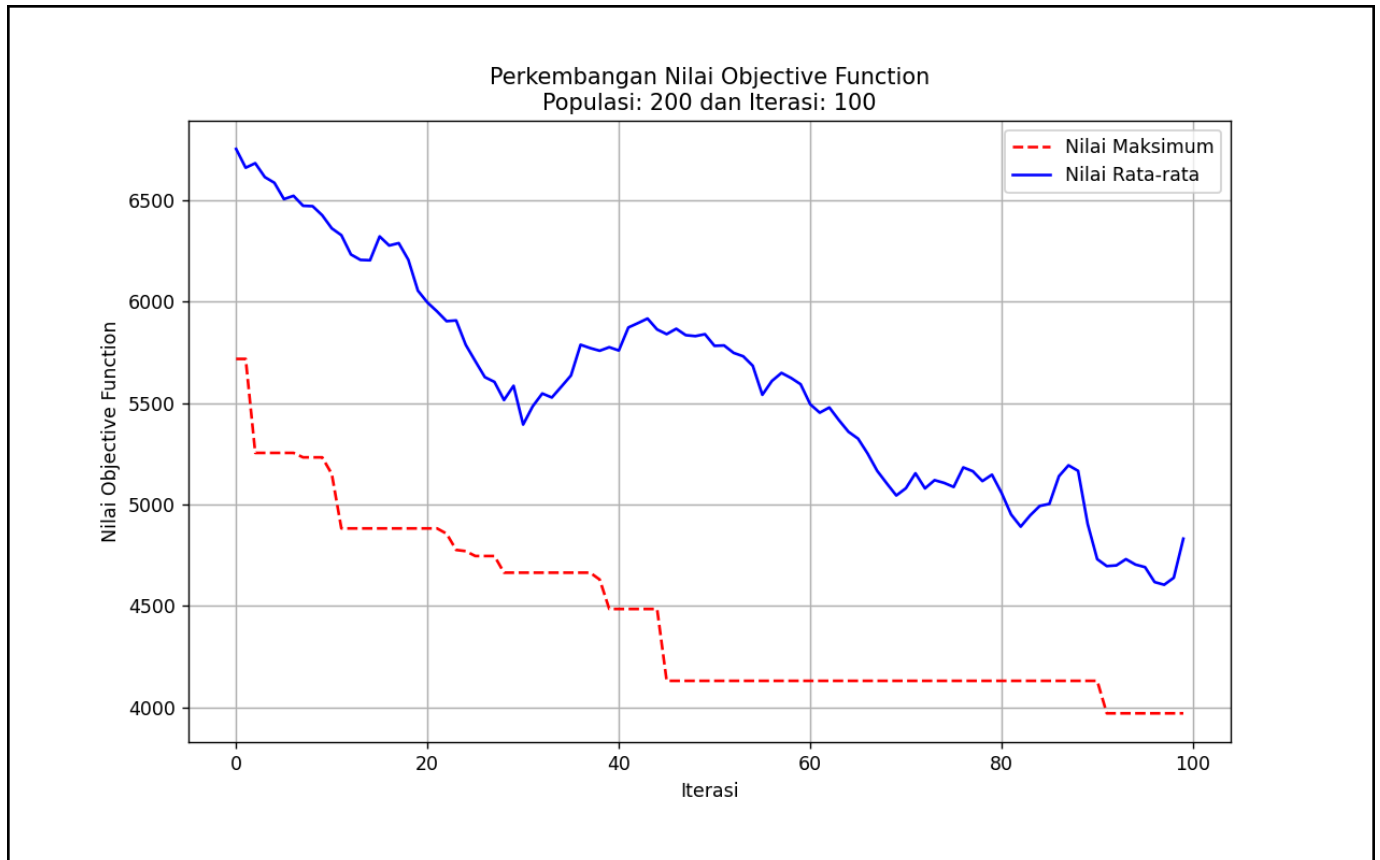
Eksperimen 1 dengan Jumlah Populasi 200 Jumlah Iterasi 100			
<b>State awal</b>	<div>[[[114 34 39 125 107]</div> <div>[ 97 38 19 67 53]</div> <div>[ 52 66 120 30 84]</div> <div>[ 15 23 46 123 33]</div> <div>[104 82 54 70 3]]</div> <div>[[ 44 103 2 117 99]</div> <div>[ 5 81 55 36 105]</div> <div>[ 62 14 51 92 119]</div> <div>[122 80 96 63 37]</div> <div>[ 28 83 17 16 47]]</div>	<b>State akhir</b>	<div>[[[ 48 30 110 51 67]</div> <div>[ 95 77 44 46 100]</div> <div>[123 68 86 4 34]</div> <div>[ 29 81 33 58 61]</div> <div>[ 9 80 41 109 62]]</div> <div>[[ 17 120 89 72 19]</div> <div>[ 70 82 125 99 18]</div> <div>[ 5 59 93 122 32]</div> <div>[124 10 43 22 101]</div> <div>[108 65 35 8 104]]</div>

	[[ 98 71 65 25 29] [ 88 24 95 57 7] [ 72 27 12 11 42] [ 56 91 86 111 76] [ 35 121 31 112 113]]  [[ 20 68 43 75 109] [ 64 22 108 48 18] [102 10 45 77 4] [ 79 41 94 115 74] [ 40 89 21 85 69]]  [[ 13 87 50 61 32] [ 78 100 9 26 59] [ 6 90 116 60 106] [ 58 118 49 124 110] [ 73 101 8 1 93]]]		[[ 66 1 96 39 117] [ 21 118 11 47 50] [ 83 97 40 57 69] [ 92 36 114 102 20] [ 49 2 87 112 28]]  [[105 78 55 7 38] [ 16 90 15 85 63] [ 75 106 37 56 26] [ 13 14 115 111 113] [103 24 25 76 79]]  [[ 23 27 91 98 73] [ 88 64 31 74 54] [ 52 3 42 53 84] [ 60 121 12 107 45] [119 94 116 6 71]]]
Jumlah populasi			200
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			3877
Durasi proses pencarian			33.92 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Eksperimen 2 dengan Jumlah Populasi 200 Jumlah Iterasi 100			
<b>State awal</b>	<div>[[[ 18 57 6 36 46]</div> <div>[ 40 60 111 68 66]</div> <div>[ 89 101 82 33 110]</div> <div>[ 85 62 93 100 99]</div> <div>[ 51 54 114 102 88]]</div> <div>[[112 9 43 121 74]</div> <div>[ 12 47 118 53 122]</div> <div>[ 17 3 72 109 94]</div> <div>[ 90 115 23 20 49]</div> <div>[ 81 95 92 63 4]]</div>	<b>State akhir</b>	<div>[[[ 70 3 94 117 36]</div> <div>[ 46 73 112 55 10]</div> <div>[ 25 62 110 72 38]</div> <div>[ 75 41 8 65 119]</div> <div>[ 89 102 22 23 114]]</div> <div>[[115 120 18 86 6]</div> <div>[ 16 95 1 79 121]</div> <div>[ 74 11 103 69 48]</div> <div>[ 50 29 113 32 122]</div> <div>[ 61 111 96 37 19]]</div>

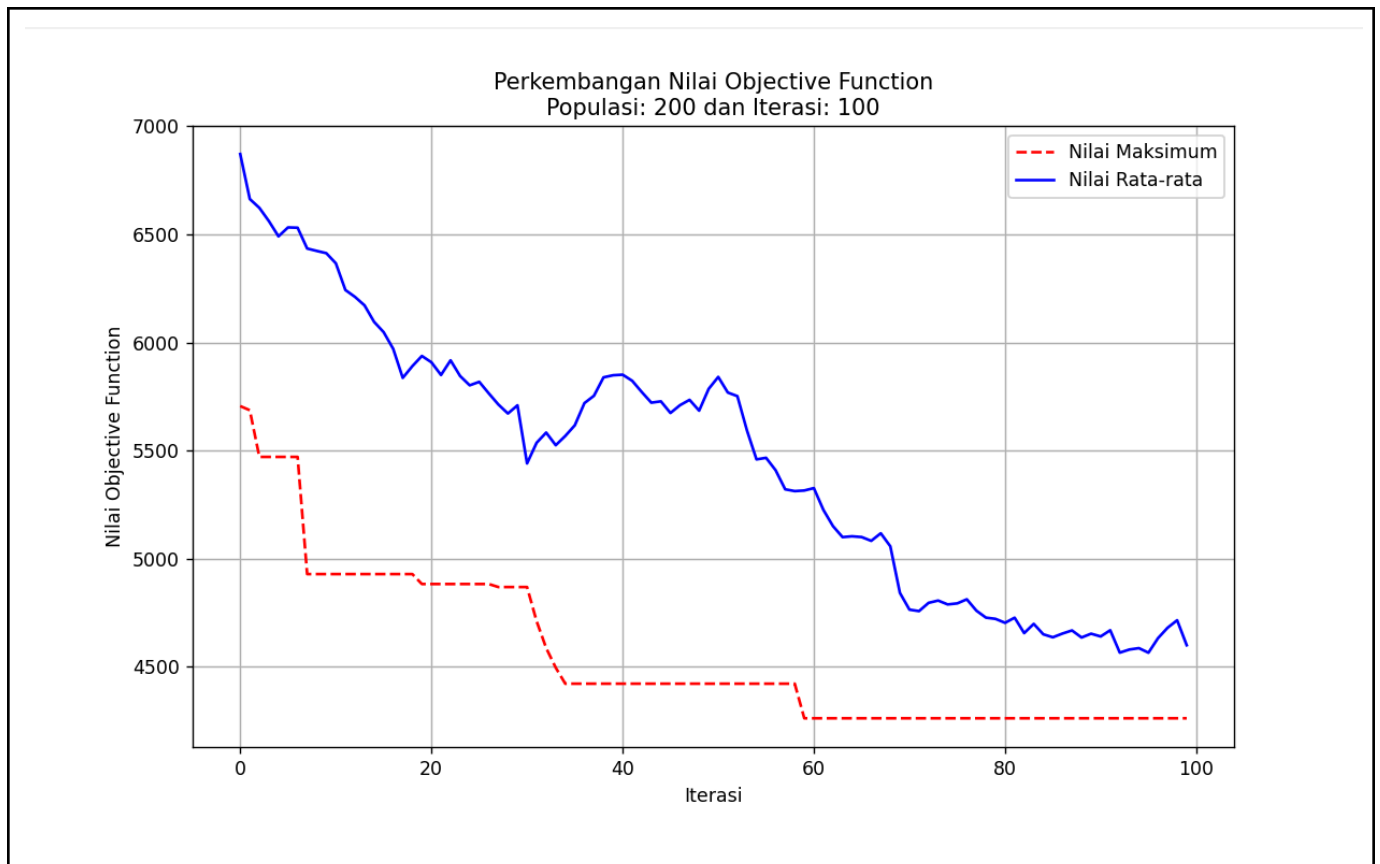
	[[ 67 73 38 124 69] [ 32 125 26 29 116] [106 15 77 103 120] [ 97 56 80 65 78] [ 34 25 11 50 76]]  [[ 79 27 71 104 35] [ 37 70 13 14 84] [ 52 2 105 123 28] [ 42 55 48 98 96] [ 61 117 107 41 59]]  [[ 64 22 45 86 31] [ 44 113 119 19 7] [ 21 91 83 108 10] [ 16 5 87 75 1] [ 30 39 8 58 24]]]		[[ 28 49 43 76 123] [ 47 63 97 51 54] [ 82 99 30 24 53] [ 9 78 88 125 13] [ 81 17 71 68 90]]  [[105 91 40 4 2] [ 56 58 42 124 5] [ 34 92 35 21 104] [ 45 98 116 64 80] [108 14 93 44 66]]  [[ 20 33 109 101 39] [ 57 106 77 15 52] [ 84 26 7 100 118] [ 87 67 60 83 85] [ 27 107 12 59 31]]]
Jumlah populasi			200
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			3971
Durasi proses pencarian			47.72 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Eksperimen 3 dengan Jumlah Populasi 200 Jumlah Iterasi 100			
<b>State awal</b>	<div>[[[ 79 115 107 118 122]</div> <div>[ 34 45 5 42 99]</div> <div>[111 38 8 72 103]</div> <div>[ 55 13 74 80 25]</div> <div>[ 7 101 71 30 26]]</div> <div>[[ 97 24 106 81 40]</div> <div>[117 68 88 94 67]</div> <div>[ 22 10 35 86 57]</div> <div>[ 44 98 47 104 52]</div> <div>[ 1 95 87 125 46]]</div>	<b>State akhir</b>	<div>[[[ 6 100 34 53 56]</div> <div>[ 91 10 97 3 86]</div> <div>[109 12 83 39 71]</div> <div>[ 8 59 45 113 76]</div> <div>[112 88 16 62 32]]</div> <div>[[ 29 123 82 69 63]</div> <div>[ 77 102 74 55 24]</div> <div>[122 64 111 41 17]</div> <div>[ 28 98 60 20 65]</div> <div>[ 9 19 61 116 79]]</div>



	[[ 16 9 20 73 116] [ 62 89 28 91 59] [ 53 93 108 120 83] [123 70 49 58 114] [124 105 51 76 41]]  [[121 6 3 96 75] [ 63 82 113 54 78] [102 60 77 48 66] [ 14 32 27 84 31] [ 85 21 12 19 119]]  [[ 29 92 109 50 4] [ 61 65 90 11 69] [ 37 15 33 110 64] [112 2 17 39 100] [ 56 43 36 23 18]]]		[[119 66 49 40 58] [ 37 117 48 120 103] [ 36 25 54 124 115] [106 7 96 5 107] [101 57 68 21 14]]  [[ 31 51 81 43 75] [ 46 72 44 52 85] [ 84 4 47 94 42] [118 27 95 70 1] [ 99 108 38 92 110]]  [[ 78 50 89 104 30] [ 67 13 22 114 23] [ 18 26 2 87 80] [ 73 125 15 93 105] [ 33 35 90 11 121]]]
Jumlah populasi			200
Banyak iterasi			100
Nilai <i>objective function</i> yang dicapai			4262
Durasi proses pencarian			35.69 detik
Plot nilai <i>objective function</i> terhadap banyak iterasi yang telah dilewati			



Berdasarkan hasil eksperimen tersebut semakin banyak iterasi, semakin banyak kesempatan algoritma *genetic algorithm* ini untuk mengeksplorasi ruang solusi dan mengoptimalkan solusi yang dihasilkan. Semakin banyak populasi, semakin besar pula keragaman (variasi) genetik di dalam populasi yang membantu algoritma untuk mengeksplorasi ruang solusi secara lebih luas. Jika jumlah iterasi terlalu banyak tanpa adanya strategi yang adaptif terhadap eksplorasi dan eksploitasi akan mengalami konvergensi dini. Teknik yang dipakai untuk *Genetic Algorithm* ini adalah *Tournament Selection*, *Partial Layer Crossover*, dan *Adaptive Mutation*.

### 3. Kesimpulan dan Saran

Berdasarkan eksperimen di atas, algoritma *local search* yang paling efektif untuk mendekati *global maximum* adalah *simulated annealing*. Algoritma ini mencapai *objective score* 235. Nilai ini bisa ditingkatkan dengan memperbesar nilai *input*, namun tentunya durasi yang dibutuhkan akan menjadi lebih lama. Untuk mencapai nilai yang lebih baik, dapat dilakukan perubahan beberapa fungsi pada algoritma tersebut, seperti bagaimana algoritma melakukan *cooling* dan penerimaan *neighbor* yang dapat dioptimalkan dengan mengubah parameter *random* menjadi suatu variabel yang dapat memberikan hasil paling mendekati *global maximum*.

Selain itu, algoritma *local search* yang dapat memproses paling cepat adalah *stochastic hill-climbing*. Namun, algoritma ini memiliki hasil *objective score* yang paling tidak optimal. Hal ini dipengaruhi oleh jumlah maksimal iterasi yang digunakan sebagai parameter. Jika ingin menggunakan algoritma ini kedepannya, sebaiknya menggunakan jumlah maksimal iterasi yang lebih besar untuk mendapatkan hasil paling optimal.

## 4. Pembagian Tugas

No.	Nama	NIM	Kegiatan
1.	Rajendra Farras Rayhan	18222105	<ul style="list-style-type: none"><li>- Mengerjakan implementasi <i>genetic algorithm</i></li><li>- Mengerjakan hasil eksperimen dan analisis <i>genetic algorithm</i> pada Laporan Tugas Besar</li></ul>
2.	Lina Azizah R.H.	18222107	<ul style="list-style-type: none"><li>- Mengerjakan implementasi <i>hill-climbing with sideways move</i> dan <i>stochastic hill-climbing</i></li><li>- Mengerjakan hasil eksperimen dan analisis <i>hill-climbing with sideways move</i> serta <i>stochastic hill climbing</i> pada Laporan Tugas Besar</li></ul>
3.	Gracya Tio Damena S.	18222110	<ul style="list-style-type: none"><li>- Mengerjakan implementasi <i>steepest ascent hill-climbing</i> dan <i>random restart hill-climbing</i></li><li>- Mengerjakan hasil eksperimen dan analisis <i>steepest ascent hill-climbing</i> dan <i>random restart hill-climbing</i> pada Laporan Tugas Besar</li></ul>
4.	M. Kasyfil Aziz	18222127	<ul style="list-style-type: none"><li>- Mengerjakan implementasi <i>simulated annealing</i></li><li>- Mengerjakan hasil eksperimen dan analisis <i>simulated annealing</i> pada Laporan Tugas Besar</li></ul>

## 5. Referensi

- Features of the magic cube.* (n.d.). Magisch Vierkant.  
<https://www.magischvierkant.com/three-dimensional-eng/magic-features/>
- GeeksforGeeks. (2024, August 27). *Objective function.* GeeksforGeeks.  
<https://www.geeksforgeeks.org/objective-function/>
- Perfect Magic Cubes.* (n.d.).  
<https://www.trump.de/magic-squares/magic-cubes/cubes-1.html>
- Prestwich, S. D. (2008). The relation between complete and incomplete search. In *Studies in computational intelligence* (pp. 63–83).  
[https://doi.org/10.1007/978-3-540-78295-7\\_3](https://doi.org/10.1007/978-3-540-78295-7_3)
- Russell, S., & Norvig, P. (2019). *Artificial intelligence: A Modern Approach*. Pearson Higher Education.
- Saeidi, S. (2018). Solving the Rubik's Cube using Simulated Annealing and Genetic Algorithm. *International Journal of Education and Management Engineering*, 8(1), 1–10. <https://doi.org/10.5815/ijeme.2018.01.01>
- Wolfram Research, Inc. (n.d.). *Magic Cube -- from Wolfram MathWorld.*  
<https://mathworld.wolfram.com/MagicCube.html>
- Wolfram Research, Inc. (2003, November 18). *MathWorld News: Perfect Magic Cube of Order 5 discovered.* <https://mathworld.wolfram.com/news/2003-11-18/magiccube/>