

LAPORAN TUGAS BESAR 01

IF3170 Inteligensi Artifisial

Pencarian Solusi Diagonal Magic Cube dengan Local Search



Disusun Oleh:

Ivan Hendrawan Tan 13522111

William Glory Henderson 13522113

Naufal Adnan 13522116

Mesach Harmasendro 13522117

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

2024

DAFTAR ISI

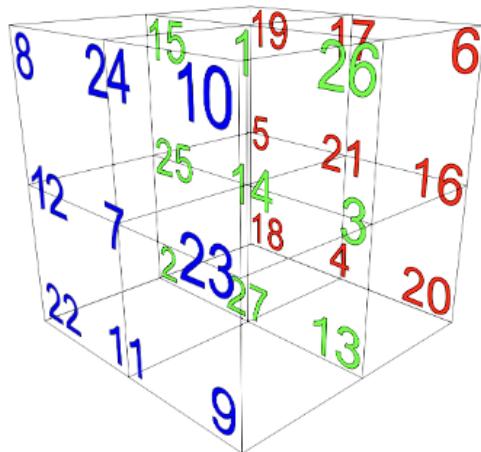
DAFTAR ISI.....	1
BAB I DESKRIPSI PERMASALAHAN.....	2
BAB II PEMBAHASAN.....	4
A. Fungsi Objektif.....	4
B. Implementasi Local Search.....	8
C. Hasil Percobaan dan Analisis.....	25
BAB III KESIMPULAN DAN SARAN.....	111
PEMBAGIAN TUGAS.....	115
LAMPIRAN.....	116
REFERENSI.....	117

BAB I

DESKRIPSI PERMASALAHAN

Diagonal magic cube adalah sebuah kubus berukuran $n \times n \times n$ yang berisi angka 1 hingga n^3 tanpa pengulangan dengan n adalah panjang sisi pada kubus tersebut. Angka-angka tersebut tersusun dengan properti-properti berikut:

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



Gambar 1.1 Ilustrasi potongan bidang dari kubus berukuran 3

Pada persoalan ini, diberikan diagonal magic cube yang berukuran $5 \times 5 \times 5$. Susunan angka 1 hingga 5^3 (125) secara acak akan menjadi initial state dari diagonal magic cube tersebut. Permasalahan dari diagonal magic cube ini perlu diselesaikan dengan 3 buah algoritma local search yaitu algoritma hill-climbing, simulated annealing, dan genetic algorithm. Eksperimen akan dilakukan menggunakan masing-masing algoritma tersebut

dengan berbagai parameter dan kondisi awal untuk menganalisis performanya. Analisis ini akan memberikan wawasan tentang efektivitas dari setiap algoritma dalam menyelesaikan permasalahan diagonal magic cube ini.

BAB II

PEMBAHASAN

A. Fungsi Objektif

Objective function adalah fungsi yang digunakan untuk mengevaluasi seberapa baik suatu solusi dalam mencapai tujuan tertentu. Dalam konteks diagonal magic cube, objective function yang dibuat akan menghitung seberapa jauh susunan angka saat ini dari mencapai kondisi "magic". Untuk membuat objective function, perlu dicari terlebih dahulu nilai konstan magic number. Jika terdapat sebuah kubus, maka nilai magic number dari magic cube tersebut adalah

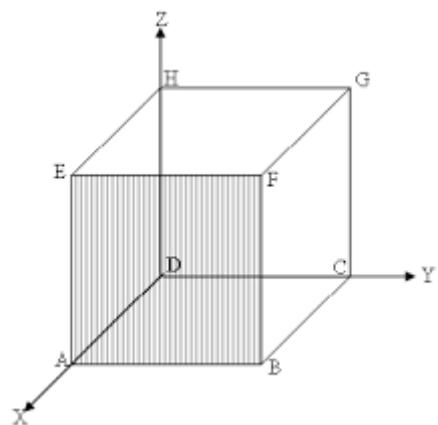
$$M(n) = n \left(\frac{\frac{n^3(n^3+1)}{2}}{n^3} \right)$$

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

Untuk persoalan diagonal magic cube 5x5x5, maka memiliki magic number

$$M(n) = \frac{5(5^3+1)}{2} = \frac{5(125+1)}{2} = \frac{5(126)}{2} = \frac{630}{2} = 315$$

Untuk diagonal magic cube ukuran 5x5x5, maka jumlah baris, kolom, tiang, dan diagonalnya adalah sebagai berikut.



Gambar 2.1 Ilustrasi Sebuah Kubus

- Baris

Misalkan baris adalah bidang xy, maka terdapat 5 layer dengan setiap layernya memiliki 5 baris. Jadi, total baris adalah $5 \times 5 = 25$.

- Kolom

Misalkan kolom adalah bidang xz, maka terdapat 5 layer dengan setiap layernya memiliki 5 kolom. Jadi, total kolom adalah $5 \times 5 = 25$.

- Tiang

Misalkan tiang adalah bidang yz, maka terdapat 5 layer dengan setiap layernya memiliki 5 tiang. Jadi, total tiang adalah $5 \times 5 = 25$.

- Diagonal Bidang

Setiap bidang $n \times n$ memiliki 2 diagonal yaitu diagonal utama dan sekunder. Ada 3 orientasi bidang dalam kubus yaitu bidang xy, yz, dan xz. Setiap orientasi memiliki 5 bidang maka total seluruh diagonal bidang adalah: $2 \times 3 \times 5 = 30$. Berikut perhitungannya:

- Pada Bidang XY

$$\text{Diagonal utama: } \sum_{i=1}^5 D_{i,i,k}, \text{ Diagonal sekunder: } \sum_{i=1}^5 D_{i,6-i,k}$$

Di sini, nilai elemen diagonal utama diambil dari titik $D_{i,i,k}$ di mana i bergerak dari 1 sampai 5 (untuk elemen sepanjang diagonal) dan k merupakan indeks tetap yang menunjukkan posisi pada sumbu Z (dari 1 hingga 5).

- Pada Bidang YZ

$$\text{Diagonal utama: } \sum_{i=1}^5 D_{k,i,i}, \text{ Diagonal sekunder: } \sum_{i=1}^5 D_{k,i,6-i}$$

Di sini, nilai elemen diagonal utama diambil dari titik $D_{k,i,i}$ di mana i bergerak dari 1 sampai 5 (untuk elemen sepanjang diagonal) dan k merupakan indeks tetap yang menunjukkan posisi pada sumbu X (dari 1 hingga 5).

- Pada Bidang XZ

$$\text{Diagonal utama: } \sum_{i=1}^5 D_{i,k,i}, \text{ Diagonal sekunder: } \sum_{i=1}^5 D_{6-i,k,i}$$

Di sini, nilai elemen diagonal utama diambil dari titik $D_{i,k,i}$ di mana i bergerak dari 1 sampai 5 (untuk elemen sepanjang diagonal) dan k merupakan indeks tetap yang menunjukkan posisi pada sumbu Y (dari 1 hingga 5).

- Diagonal Ruang

Dalam kubus, terdapat 4 diagonal ruang yang menghubungkan sudut-sudut berlawanan melalui pusat kubus. Total diagonal ruang = 4

Pada persoalan diagonal magic cube, terdapat 2 pendekatan objective function. Yang pertama, kita dapat menghitung banyaknya baris, kolom, tiang, dan diagonal yang jumlah angkanya 315. Karena kita mengetahui jumlah dari semua baris, kolom, tiang, dan diagonal jumlahnya 109, maka nilai dari objective function yang paling mendekati 109 adalah yang paling bagus. Yang kedua, kita dapat menjumlahkan semua baris, kolom, tiang, dan seluruh diagonal menjadi sama dengan magic number. Secara intuitif, langkah termudah untuk mengukur seberapa jauh konfigurasi penempatan angka pada kubus saat ini dari mencapai kondisi magic adalah dengan menghitung total selisih absolut antara semua baris, semua kolom, semua tiang, dan semua diagonal dengan magic numbernya.

Kami memilih pendekatan kedua yaitu menghitung total selisih absolut antara semua baris, semua kolom, semua tiang, dan semua diagonal dengan magic numbernya karena pendekatan ini lebih efektif dibandingkan pendekatan pertama. Dari ide pendekatan kedua diperoleh rumus untuk objective function adalah

$$f(x) = \sum_{i=1}^{25} |s_{row,i} - M| + \sum_{j=1}^{25} |s_{col,j} - M| + \sum_{k=1}^{25} |s_{tower,k} - M| + \sum_{l=1}^{30} |s_{diag-plane,l} - M| + \sum_{m=1}^4 |s_{diag-space,m} - M|$$

Di mana:

$s_{row,i}$ adalah jumlah angka pada baris ke- i ,

$s_{col,j}$ adalah jumlah angka pada kolom ke- j ,

$s_{tower,k}$ adalah jumlah angka pada tiang ke- k ,

$s_{diag-plane,l}$ adalah jumlah angka pada diagonal bidang ke- l

$s_{diag-space,m}$ adalah jumlah angka pada diagonal bidang ke-m

M adalah magic number.

Rumus tersebut dituliskan sebagai gambaran dari ide cara perhitungan objective function. Nilai mutlak digunakan untuk menciptakan konsistensi dalam penilaian karena semua elemen (baris, kolom, tiang, dan diagonal) yang tidak memenuhi syarat akan dihitung sebagai "kesalahan" dalam bentuk selisih positif, tanpa membingungkan arah kesalahan. Dengan begitu, Fungsi $f(x)$ akan bernilai $f(x) \geq 0$. Dari fungsi tersebut, maka sebuah kubus akan memenuhi magic cube ketika memiliki nilai $f(x) = 0$ yang artinya semua baris, semua kolom, semua tiang, dan seluruh diagonal memenuhi syarat dengan jumlah yang sama persis dengan magic number (M). Sementara itu, nilai $f(x) > 0$ menunjukkan adanya ketidaksesuaian. Nilai $f(x)$ yang semakin besar menunjukkan seberapa besar kesalahan dalam konfigurasi saat ini, menandakan seberapa jauh solusi saat ini dari magic cube yang valid. Peningkatan nilai $f(x)$ mendekati nilai 0 menjadi tantangan dalam menemukan solusi yang lebih baik, yang harus diatasi oleh algoritma pencarian yang digunakan.

Alasan $f(x)$ dipilih sebagai objective function yang dipilih untuk $5 \times 5 \times 5$ diagonal karena fungsi ini secara efektif mengukur kualitas solusi dengan menghitung total defisit dari jumlah setiap baris, kolom, tiang, dan diagonal terhadap magic number (M). Pemilihan fungsi ini didasarkan pada kemampuannya untuk menilai seberapa baik konfigurasi angka dalam kubus memenuhi syarat magic cube, di mana nilai $f(x)$ yang lebih rendah menunjukkan bahwa lebih banyak elemen mendekati jumlah yang diinginkan. Dengan memfokuskan pada konsistensi, fungsi ini mendorong penyusunan angka yang selaras, sekaligus memungkinkan algoritma pencarian untuk mengidentifikasi langkah-langkah yang paling berpotensi menghasilkan peningkatan. Selain itu, objective function ini bersifat umum, dapat diterapkan pada berbagai ukuran kubus, sehingga memudahkan perluasan dan pengembangan lebih lanjut. Dengan mempertimbangkan seluruh struktur kubus, fungsi ini juga mengurangi kemungkinan terjebak di lokal optima, memungkinkan pencarian solusi yang lebih menyeluruh. Secara keseluruhan, pemilihan objective function ini bertujuan untuk

menciptakan pendekatan yang sistematis dan efisien dalam menyelesaikan masalah diagonal magic cube.

B. Implementasi Local Search

Implementasi dilakukan menggunakan bahasa pemrograman python dengan tkinter sebagai GUI tampilannya serta menggunakan matplotlib untuk membantu dalam pembuatan grafik plotnya. Selain menggunakan library di atas juga digunakan library numpy untuk mempercepat dan mempermudah proses perhitungan yang ada pada program terutama pada perhitungan nilai objective. Untuk memperjelas state suatu kubus digunakan juga visualisasi 3D dengan menggunakan library VTK.

Dalam tugas ini kami menggunakan pendekatan OOP untuk membuat programnya sehingga bisa lebih terstruktur dan jelas. Dalam program ini terdapat beberapa class dasar yang digunakan untuk mempermudah proses local search. Class dasar yang paling utama adalah Class Cube yang berisi segala code yang berhubungan dengan cube seperti generate random cube, perhitungan nilai objective, generate random atau best successor, dan lain-lain. Berikut adalah source code dari Class Cube

```

● ○ ●

1 import numpy as np
2 import random
3 import itertools
4 import copy
5 from concurrent.futures import ThreadPoolExecutor
6
7 class Cube:
8     def __init__(self, n):
9         self.size = n
10        self.magic_number = n * (n**3 + 1) // 2
11        self.cube = np.zeros((n, n, n), dtype=int)
12
13        self.row_sums = np.zeros((n, n), dtype=int)
14        self.col_sums = np.zeros((n, n), dtype=int)
15        self.pillar_sums = np.zeros((n, n), dtype=int)
16        self.main_diag_sums = np.zeros(4, dtype=int)
17        self.plane_diag_sums = np.zeros(12, dtype=int)
18
19
20    def generate_cube(self):
21        numbers = np.arange(1, self.size ** 3 + 1)
22        np.random.shuffle(numbers)
23        self.cube = numbers.reshape((self.size, self.size, self.size))
24        self.initialize_sums()
25
26
27    def swap(self, idx1, idx2):
28        i1, j1, k1 = idx1
29        i2, j2, k2 = idx2
30
31        self.cube[i1, j1, k1], self.cube[i2, j2, k2] = self.cube[i2, j2, k2], self.cube[i1, j1, k1]
32
33
34
35    def evaluate_objective_function(self):
36        n = self.size
37        magic_number = self.magic_number

```

```

38
39     row_sums = np.sum(self.cube, axis=2)
40     col_sums = np.sum(self.cube, axis=1)
41     pillar_sums = np.sum(self.cube, axis=0)
42
43
44     total_diff = np.sum(np.abs(row_sums - magic_number))
45     total_diff += np.sum(np.abs(col_sums - magic_number))
46     total_diff += np.sum(np.abs(pillar_sums - magic_number))
47
48     main_diag_1 = np.sum([self.cube[i, i, i] for i in range(n)])
49     main_diag_2 = np.sum([self.cube[i, i, n-1-i] for i in range(n)])
50     main_diag_3 = np.sum([self.cube[i, n-1-i, i] for i in range(n)])
51     main_diag_4 = np.sum([self.cube[i, n-1-i, n-1-i] for i in range(n)])
52
53
54     total_diff += np.abs(main_diag_1 - magic_number)
55     total_diff += np.abs(main_diag_2 - magic_number)
56     total_diff += np.abs(main_diag_3 - magic_number)
57     total_diff += np.abs(main_diag_4 - magic_number)
58
59
60     plane_diag_1 = np.sum([self.cube[i, i, 0] for i in range(n)])
61     plane_diag_2 = np.sum([self.cube[i, n-1-i, 0] for i in range(n)])
62     plane_diag_3 = np.sum([self.cube[i, i, n-1] for i in range(n)]) #tt
63     plane_diag_4 = np.sum([self.cube[i, n-1-i, n-1] for i in range(n)])
64
65     plane_diag_5 = np.sum([self.cube[i, 0, i] for i in range(n)])
66     plane_diag_6 = np.sum([self.cube[i, 0, n-1-i] for i in range(n)])
67     plane_diag_7 = np.sum([self.cube[i, n-1, i] for i in range(n)])
68     plane_diag_8 = np.sum([self.cube[i, n-1, n-1-i] for i in range(n)])
69

```

```

70     plane_diag_9 = np.sum([self.cube[0, i, i] for i in range(n)])
71     plane_diag_10 = np.sum([self.cube[0, i, n-1-i] for i in range(n)]) #tt
72     plane_diag_11 = np.sum([self.cube[n-1, i, i] for i in range(n)])
73     plane_diag_12 = np.sum([self.cube[n-1, i, n-1-i] for i in range(n)])
74
75
76     total_diff += np.abs(plane_diag_1 - magic_number)
77     total_diff += np.abs(plane_diag_2 - magic_number)
78     total_diff += np.abs(plane_diag_3 - magic_number)
79     total_diff += np.abs(plane_diag_4 - magic_number)
80     total_diff += np.abs(plane_diag_5 - magic_number)
81     total_diff += np.abs(plane_diag_6 - magic_number)
82     total_diff += np.abs(plane_diag_7 - magic_number)
83     total_diff += np.abs(plane_diag_8 - magic_number)
84     total_diff += np.abs(plane_diag_9 - magic_number)
85     total_diff += np.abs(plane_diag_10 - magic_number)
86     total_diff += np.abs(plane_diag_11 - magic_number)
87     total_diff += np.abs(plane_diag_12 - magic_number)
88
89
90     return total_diff

```

```

91
92     def generate_all_successors(self):
93
94         indices = [(i, j, k) for i in range(self.size) for j in range(self.size) for k in range(self.size)]
95         index_pairs = itertools.combinations(indices, 2)
96
97         for idx1, idx2 in index_pairs:
98             self.swap(idx1, idx2)
99             yield self
100            self.swap(idx1, idx2)
101
102
103    def generate_random_successor(self):
104
105        indices = [(i, j, k) for i in range(self.size) for j in range(self.size) for k in range(self.size)]
106        idx1, idx2 = random.sample(indices, 2)
107
108        self.swap(idx1, idx2)
109        successor = copy.deepcopy(self)
110        self.swap(idx1, idx2)
111
112        return successor

```

```

113
114    def get_best_successor(self) -> 'Cube':
115
116        best_cube = None
117        best_score = float('inf')
118
119        for successor in self.generate_all_successors():
120            score = successor.evaluate_objective_function()
121            if score < best_score:
122                best_cube = copy.deepcopy(successor)
123                best_score = score
124
125        return best_cube
126
127    def get_random_successor(self):
128
129        return self.generate_random_successor()
130
131    def to_flat_list(self):
132        return self.cube.flatten().tolist()
133
134    def from_flat_list(flat_list):
135        size = int(round(len(flat_list)**(1/3)))
136        cube = Cube(size)
137        cube.cube = np.array(flat_list).reshape((size, size, size))
138        return cube

```

Berikut adalah penjelasan fungsi-fungsi pada Class Cube

- generate_cube(): berfungsi untuk menginisiasi data pada Class cube dengan angka random mulai dari 1 dan tentu saja tidak ada angka yang terduplicasi
- swap(idx1, idx2): berfungsi untuk menukar angka dengan index idx1 pada Cube dengan angka pada index idx2
- evaluate_objective_function(): berfungsi untuk menghitung dan mengembalikan nilai objective berdasarkan state Cube saat ini

- generate_all_successor(): berfungsi untuk menghasilkan semua successor yang mungkin dari state Cube saat ini
- generate_random_succesor(): berfungsi untuk menghasilkan successor random dari state Cube saat ini
- get_best_successor(): berfungsi untuk menghasilkan successor dengan nilai objective terbaik diantara semua successor yang ada pada state Cube saat ini
- get_random_successor(): berfungsi untuk mendapatkan successor random dari state Cube saat ini
- to_flat_list(): berfungsi untuk mengubah cube dari bentuk array 3d menjadi array 1d
- from_flat_list(flat_list): berfungsi untuk mengembalikan cube dalam bentuk array 1d menjadi array 3d

Selain Class Cube ada juga Class dasar lain yaitu BaseLocalSearchAlgorithm yang merupakan abstract class dari semua class local search lainnya. Class ini hanya berisi satu abstract class yaitu run yang digunakan untuk memastikan semua class local search nantinya akan memiliki fungsi untuk menjalankan algoritmanya. Class ini juga hanya terdiri dari satu atribut yaitu Cube yang pastinya akan selalu digunakan pada Class algoritma lainnya. Untuk semua Class yang mewakili masing-masing algoritma nantinya akan menjadi turunan dari Class ini. Berikut adalah source code dari Class BaseLocalSearchAlgorithm



```

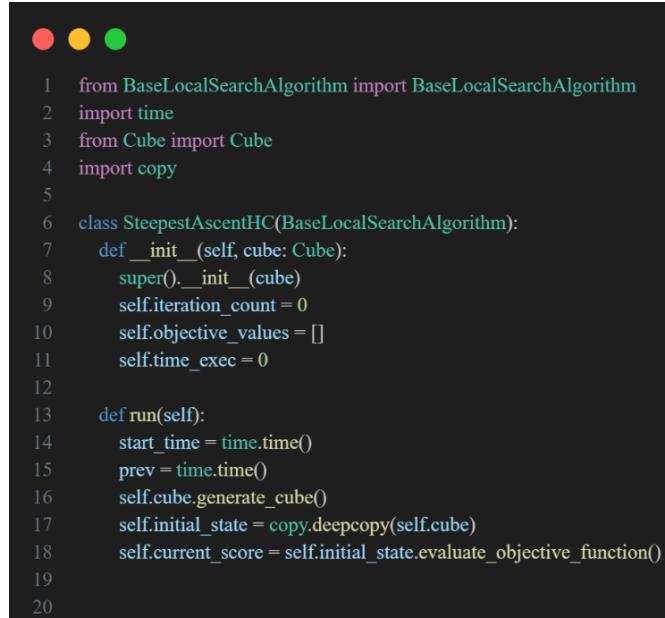
1  from abc import ABC, abstractmethod
2  from Cube import Cube
3  from typing import Type
4
5  class BaseLocalSearchAlgorithm(ABC):
6      def __init__(self, cube: Cube):
7          self.cube = cube
8
9      @abstractmethod
10     def run(self):
11         pass
12
13

```

Dengan menggunakan Class dasar tersebut Class lain untuk masing algoritma sudah bisa dibuat dengan baik. Berikut adalah masing-masing Class yang mewakili masing-masing algoritma:

1. Class SteepestAscentHC

Class ini adalah Class yang merepresentasikan algoritma Steepest Ascent Hill Climbing. Class ini hanya berisi satu fungsi yaitu implementasi dari fungsi abstract run. Fungsi tersebut adalah fungsi yang berfungsi untuk melakukan pencarian solusi dari magic cube dengan menerapkan algoritma Steepest Ascent Hill Climbing. Fungsi ini nantinya akan mengembalikan sebuah dictionary yang berisi data yang memang diperlukan untuk ditampilkan sesuai dengan ketentuan pada spesifikasi. Berikut adalah source code dari algoritma ini



```
 1  from BaseLocalSearchAlgorithm import BaseLocalSearchAlgorithm
 2  import time
 3  from Cube import Cube
 4  import copy
 5
 6  class SteepestAscentHC(BaseLocalSearchAlgorithm):
 7      def __init__(self, cube: Cube):
 8          super().__init__(cube)
 9          self.iteration_count = 0
10          self.objective_values = []
11          self.time_exec = 0
12
13      def run(self):
14          start_time = time.time()
15          prev = time.time()
16          self.cube.generate_cube()
17          self.initial_state = copy.deepcopy(self.cube)
18          self.current_score = self.initial_state.evaluate_objective_function()
19
20
```

```

21     while (True):
22         self.iteration_count += 1
23         successor = self.cube.get_best_successor()
24
25         successor_value = successor.evaluate_objective_function()
26         self.objective_values.append(successor_value)
27
28         if (successor_value < self.current_score):
29             self.cube = successor
30             self.current_score = successor_value
31         else:
32             end_time = time.time()
33             self.time_exec = end_time-start_time
34             print(self.current_score)
35             print(successor_value)
36             print()
37             break;
38
39         if self.iteration_count % 2 == 0:
40             print("count", self.iteration_count)
41             print("current_value", self.current_score)
42             print(time.time() - prev)
43             prev = time.time()
44             print()
45             print()
46
47         # self.initial_state.display_cube()
48
49     return {
50         "initial_state": self.initial_state,
51         "final_state": self.cube,
52         "final_objective": self.current_score,
53         "iterations": self.iteration_count,
54         "duration": self.time_exec,
55         "objective_progress": self.objective_values,
56
57     }
58
59

```

2. Class HCSidewaysMove

Sesuai namanya, class ini berisi algoritma Hill Climbing with Sideways Move. Di dalam class ini, terdapat hanya satu buah fungsi yaitu run untuk menjalankan algoritma ini. Fungsi ini juga akan mengembalikan sebuah dictionary yang berisi data-data untuk ditampilkan sesuai dengan spek tugas. Berikut adalah foto source code dari class ini.



```
1 from BaseLocalSearchAlgorithm import BaseLocalSearchAlgorithm
2 import time
3 from Cube import Cube
4 import copy
5
6 class HCSidewaysMove(BaseLocalSearchAlgorithm):
7     def __init__(self, cube: Cube, max_sideways):
8         super().__init__(cube)
9         self.iteration_count = 0
10        self.objective_values = []
11        self.time_exec = 0
12        self.max_sideways = max_sideways
13        self.sideways_count = 0
14
15    def run(self):
16        start_time = time.time()
17        prev = time.time()
18        self.cube.generate_cube()
19        self.initial_state = copy.deepcopy(self.cube)
20        self.current_score = self.initial_state.evaluate_objective_function()
21
22        while (True):
23            self.iteration_count += 1
24            successor = self.cube.get_best_successor()
25
26            successor_value = successor.evaluate_objective_function()
27            self.objective_values.append(successor_value)
28
```

```

29     if (successor_value < self.current_score):
30         self(cube = successor
31         self.current_score = successor_value
32     elif (successor_value == self.current_score):
33         if self.sideways_count >= self.max_sideways:
34             end_time = time.time()
35             self.time_exec = end_time - start_time
36             break
37         self.sideways_count += 1
38         self(cube = successor
39
40     else:
41         end_time = time.time()
42         self.time_exec = end_time - start_time
43         break
44
45     if self.iteration_count % 1 == 0:
46         print("count", self.iteration_count)
47         print("current_value", self.current_score)
48         print(time.time() - prev)
49         prev = time.time()
50         print()
51         print()
52
53     return {
54         "initial_state": self.initial_state,
55         "final_state": self(cube,
56         "final_objective": self.current_score,
57         "iterations": self.iteration_count,
58         "duration": self.time_exec,
59         "objective_progress": self.objective_values,
60         "sideways_moves": self.sideways_count
61     }

```

3. Class Random Restart Hill Climbing

Class ini adalah class yang mengimplementasikan algoritma dari Random Restart Hill Climbing. Class ini berisi satu fungsi yaitu fungsi run untuk menjalankan algoritma Random Restart Hill Climbing untuk mencari solusi dari diagonal magic cube. Fungsi ini akan mengembalikan sebuah dictionary yang berisi value untuk data yang akan ditampilkan sebagai luaran dari spesifikasi tugas. Berikut adalah foto source code untuk algoritma ini.



```
1  from BaseLocalSearchAlgorithm import BaseLocalSearchAlgorithm
2  import time
3  from Cube import Cube
4  import copy
5
6  class RandomRestartHC(BaseLocalSearchAlgorithm):
7      def __init__(self, cube: Cube, max_restarts: int = 100):
8          super().__init__(cube)
9          self.max_restarts = max_restarts
10         # self.no_improvement_restarts = no_improvement_restarts
11         self.stagnant_restarts = 0
12         self.iteration_count = 0
13         self.iteration_per_restart = []
14         self.total_restarts = 0
15         self.objective_values = []
16         self.best_score = float('inf')
17         self.best_state = None
18         self.time_exec = 0
19
20     def run(self):
21         start_time = time.time()
22
23         while self.total_restarts < self.max_restarts:
24             # if self.stagnant_restarts >= self.no_improvement_restarts:
25
26                 # break;
27
28             self.total_restarts += 1
29             self.iteration_count = 0
30             prev = time.time()
31             self.cube.generate_cube()
32             self.initial_state = copy.deepcopy(self.cube)
33             self.current_score = self.initial_state.evaluate_objective_function()
34             self.objective_values.append(self.current_score)
35
36
37
38             while (True):
39                 self.iteration_count += 1
40                 successor = self.cube.get_best_successor()
41
42                 successor_value = successor.evaluate_objective_function()
43
```

```

44     if (successor_value < self.current_score):
45         self(cube = successor
46         self.current_score = successor_value
47         self.objective_values.append(successor_value)
48
49     else:
50         self.objective_values.append(self.current_score)
51
52         self.iteration_per_restart.append(self.iteration_count)
53
54     if self.current_score < self.best_score:
55         self.best_score = self.current_score
56         self.best_state = copy.deepcopy(self(cube))
57         # self.stagnant_restarts = 0
58
59     # else:
60     # self.stagnant_restarts += 1
61     print("restart: ", self.total_restarts)
62     print(self.iteration_per_restart)
63     print(self.best_score)
64     print()
65     break
66
67     if self.iteration_count % 10 == 0:
68         print("count", self.iteration_count)
69         print("current_value", self.current_score)
70         print(time.time() - prev)
71         print()
72         print()
73
74     end_time = time.time()
75     self.time_exec = end_time - start_time
76
77     return {
78         "initial_state": self.initial_state,
79         "final_state": self.best_state,
80         "final_objective": self.best_score,
81         "max_restart": self.max_restarts,
82         "total_restart": self.total_restarts,
83         "iteration per restart": self.iteration_per_restart,
84         "duration": self.time_exec,
85         "objective_progress": self.objective_values,
86     }

```

4. Class Stochastic Hill Climbing

Class ini merupakan class yang berisi algoritma dari Stochastic Hill Climbing. Class ini berisi satu fungsi yaitu fungsi run untuk menjalankan algoritma Stochastic Hill Climbing untuk mencari solusi dari diagonal magic cube. Fungsi ini akan mengembalikan sebuah dictionary yang berisi value untuk data yang akan ditampilkan sebagai luaran dari spesifikasi tugas. Berikut adalah foto source code untuk algoritma ini.

```

1 import time
2 import copy
3 from Cube import Cube
4 from BaseLocalSearchAlgorithm import BaseLocalSearchAlgorithm
5
6 class StochasticHillClimbing(BaseLocalSearchAlgorithm):
7     def __init__(self, cube: Cube):
8         super().__init__(cube)
9         self.iteration_count = 0
10        self.objective_values = []
11        self.time_exec = 0
12        self.initial_state = 0
13        self.current_score = 999999
14
15    def run(self):
16        start_time = time.time()
17        prev = time.time()
18        self.cube.generate_cube()
19        self.initial_state = copy.deepcopy(self.cube)
20        self.current_score = self.initial_state.evaluate_objective_function()
21
22        while True:
23            self.iteration_count += 1
24            successor = self.cube.generate_random_successor()
25            successor_value = successor.evaluate_objective_function()
26
27            if (successor_value < self.current_score):
28                self.cube = successor
29                self.current_score = successor_value
30                self.objective_values.append(successor_value)
31            else:
32                end_time = time.time()
33                self.time_exec = end_time - start_time
34                print(f"Current score: {self.current_score}")
35                print(f"Successor value: {successor_value}")
36                print(f"Time execution: {self.time_exec:.8f} seconds\n\n")
37                break
38
39        print(f"Count: {self.iteration_count}")
40        print(f"Current_value: {self.current_score}")
41        print(f"One iteration time: {(time.time() - prev):.12f}\n\n")
42        prev = time.time()
43
44        self.initial_state.display_cube()
45        return {
46            "initial_state": self.initial_state,
47            "final_state": self.cube,
48            "final_objective": self.current_score,
49            "iterations": self.iteration_count,
50            "duration": self.time_exec,
51            "objective_progress": self.objective_values,
52        }
53

```

5. Class SimulatedAnnealing

Class ini adalah Class yang merepresentasikan algoritma Simulated Annealing. Class ini hanya berisi satu fungsi yaitu implementasi dari fungsi abstract run. Fungsi tersebut adalah fungsi yang berfungsi untuk melakukan

pencarian solusi dari magic cube dengan menerapkan algoritma Simulated Annealing. Fungsi ini nantinya akan mengembalikan sebuah dictionary yang berisi data yang memang diperlukan untuk ditampilkan sesuai dengan ketentuan pada spesifikasi. Berikut adalah source code dari algoritma ini.



```
1 from BaseLocalSearchAlgorithm import BaseLocalSearchAlgorithm
2 import time
3 from Cube import Cube
4 import copy
5 import math
6 import random
7
8 class SimulatedAnnealing(BaseLocalSearchAlgorithm):
9     def __init__(self, cube: Cube, initial_temperature):
10         super().__init__(cube)
11         self.iteration_count = 0
12         self.objective_values = []
13         self.time_exec = 0
14         self.temperature = initial_temperature
15         self.cooling_rate = 0.995
16         self.probability_values = []
17         self.stuck_frequency = 0
18
19     def run(self):
20         start_time = time.time()
21         prev_time = time.time()
22         self(cube).generate_cube()
23         self.initial_state = copy.deepcopy(self(cube))
24         self.current_score = self.initial_state.evaluate_objective_function()
25
26         while True:
```

```

27     self.iteration_count += 1
28     self.temperature = self.temperature * self.cooling_rate
29
30     if (self.temperature <= 1e-6):
31         break
32
33     successor = self(cube).get_random_successor()
34     successor_value = successor.evaluate_objective_function()
35     self.objective_values.append(successor_value)
36
37     delta_e = successor_value - self.current_score
38
39     if delta_e < 0:
40         self(cube) = successor
41         self.current_score = successor_value
42         self.probability_values.append(1)
43     else:
44         probability = math.exp(-delta_e / self.temperature)
45         self.probability_values.append(probability)
46         # print(probability)
47         self.stuck_frequency += 1
48         if random.uniform(0, 1) < probability:
49             self(cube) = successor
50             self.current_score = successor_value
51
52
53     if self.iteration_count % 500 == 0:
54         print("Iteration count:", self.iteration_count)
55         print("Current objective value:", self.current_score)
56         print("Temperature:", self.temperature)
57         print(time.time() - prev_time)
58

```

```

59     end_time = time.time()
60     self.time_exec = end_time - start_time
61     print("Iteration count:", self.iteration_count)
62     print("Current objective value:", self.current_score)
63     print("Temperature:", self.temperature)
64     print(self.current_score)
65     print(successor_value)
66     print()
67
68     return {
69         "initial_state": self.initial_state,
70         "final_state": self(cube),
71         "final_objective": self.current_score,
72         "iterations": self.iteration_count,
73         "duration": self.time_exec,
74         "objective_progress": self.objective_values,
75         "probability_progress": self.probability_values,
76         "final_temperature": self.temperature,
77         "stuck_frequency": self.stuck_frequency
78     }
79

```

6. Class Genetic Algorithm

Class ini adalah Class yang merepresentasikan Genetic Algorithm. Class ini hanya berisi enam fungsi yaitu fungsi fitness_function, initialize_population, select_parents, order_crossover, mutate, dan run. Fungsi fitness_function digunakan untuk mengevaluasi kualitas dari sebuah individu. Fungsi initialize_population berguna untuk membangkitkan sebuah populasi. Fungsi select_parents digunakan untuk memilih parents dari populasi yang

dibangkitkan sebelumnya. Fungsi `order_crossover` adalah fungsi untuk menggabungkan dua individu dari populasi agar menghasilkan sebuah individu baru dengan karakteristik gabungan dari kedua orang tuanya. Fungsi `mutate` merupakan fungsi untuk mengubah satu atau lebih angka pada cube. Fungsi `run` digunakan untuk menjalankan algoritma genetik ini. Fungsi `run` akan mengembalikan sebuah dictionary yang berisi data yang memang diperlukan untuk ditampilkan sesuai dengan ketentuan pada spesifikasi. Berikut adalah source code dari algoritma ini.

```
● ● ●
1 from BaseLocalSearchAlgorithm import BaseLocalSearchAlgorithm
2 import time
3 from Cube import Cube
4 import copy
5 import random
6 import numpy as np
7
8 class GeneticAlgorithm(BaseLocalSearchAlgorithm):
9     def __init__(self, cube: Cube, population_size, max_iterations):
10         super().__init__(cube)
11         self.iteration_count = 0
12         self.objective_values = []
13         self.avg_objective_values = []
14         self.time_exec = 0
15         self.population_size = population_size
16         self.max_iterations = max_iterations
17
18     def fitness_function(self, cube):
19         objective_value = cube.evaluate_objective_function()
20         return (1 / (objective_value + 1e-5), objective_value)
21
22     def initialize_population(self):
23         population = []
24         for _ in range(self.population_size):
25             individual = copy.deepcopy(self.cube)
26             individual.generate_cube()
27             population.append(individual)
28
29         return population
```

```

29
30     def select_parents(self, population):
31         fitness_values = [self.fitness_function(individu)[0] for individu in population]
32
33         total_fitness = np.sum(fitness_values)
34
35         selection_probabilities = [fitness / total_fitness for fitness in fitness_values]
36
37         parents = random.choices(population, weights=selection_probabilities, k=2)
38         return parents
39
40     def order_crossover(self, parent1, parent2):
41         size = 125
42         start, end = sorted(random.sample(range(size), 2))
43
44         crossover_flat = [None] * size
45         parent1_flat = parent1.to_flat_list()
46         parent2_flat = parent2.to_flat_list()
47
48         crossover_flat[start:end] = parent1_flat[start:end]
49
50         parent2_remaining = [x for x in parent2_flat if x not in crossover_flat[start:end]]
51         current_index = end
52
53         for value in parent2_remaining:
54             if current_index >= size:
55                 current_index = 0
56             while crossover_flat[current_index] is not None:
57                 current_index += 1
58                 if current_index >= size:
59                     current_index = 0
60                 crossover_flat[current_index] = value
61
62         crossover = Cube.from_flat_list(crossover_flat)
63         return crossover
64
65     def mutate(self, cube):
66         if random.uniform(0, 1) <= 0.05:
67             flat_list = cube.to_flat_list()
68             idx1, idx2 = random.sample(range(len(flat_list)), 2)
69             flat_list[idx1], flat_list[idx2] = flat_list[idx2], flat_list[idx1]
70             mutated_cube = Cube.from_flat_list(flat_list)
71             return mutated_cube
72         return cube
73
74     def run(self):
75         start_time = time.time()
76         population = self.initialize_population()
77         fitness_objective_values = [(individual, self.fitness_function(individual)) for individual in population]
78
79         curr_best_individual, (curr_best_fitness, curr_best_objective) = max(fitness_objective_values, key=lambda x: x[1][0])
80         self.objective_values.append(curr_best_objective)
81         best_individual = curr_best_individual
82         best_fitness = curr_best_fitness
83         obj_val = curr_best_objective
84         avg_objective_value = sum(obj_val[1] for _, obj_val in fitness_objective_values) / len(population)
85         self.avg_objective_values.append(avg_objective_value)
86         initial_state = copy.deepcopy(best_individual)
87
88
89         for iteration in range(self.max_iterations):
90             new_population = []
91             for _ in range(self.population_size):
92                 parent1, parent2 = self.select_parents(population)
93                 crossover = self.order_crossover(parent1, parent2)
94                 mutation = self.mutate(crossover)
95                 new_population.append(mutation)

```

```

96     population = new_population
97
98     fitness_objective_values = [(individual, self.fitness_function(individual)) for individual in population]
99
100    curr_best_individual, (curr_best_fitness, curr_best_objective) = max(fitness_objective_values, key=lambda x: x[1][0])
101    self.objective_values.append(curr_best_objective)
102    best_individual = curr_best_individual
103    best_fitness = curr_best_fitness
104    obj_val = curr_best_objective
105
106    avg_objective_value = sum(obj_val[1] for _, obj_val in fitness_objective_values) / len(population)
107    self.avg_objective_values.append(avg_objective_value)
108
109    if iteration % 400 == 0:
110        print(f"Iteration {iteration}: Best Objective Value = {obj_val}")
111
112    end_time = time.time()
113    self.time_exec = end_time - start_time
114
115
116    return {
117        "initial_state": initial_state,
118        "final_state": best_individual,
119        "final_objective": obj_val,
120        "final_fitness": best_fitness,
121        "iterations": self.max_iterations,
122        "duration": self.time_exec,
123        "objective_progress": self.objective_values,
124        "avg_objective_progress": self.avg_objective_values
125    }

```

C. Hasil Percobaan dan Analisis

1. Steepest Ascent Hill Climbing

a. Eksperimen

- Eksperimen 1

TUBES ARTIFICIAL INTELLIGENCE

Steepest Ascent Algorithm

Initial State

Show 3D Initial State

Layer 1

1	85	77	59	37
125	60	28	80	107
71	103	48	46	72
12	91	19	54	13
44	86	34	55	119

Layer 2

10	2	17	24	49
26	21	78	94	51
121	99	62	53	20
82	109	42	63	31
83	22	47	14	118

Layer 3

6	32	102	15	5
58	66	98	8	90
84	115	81	110	38
67	120	114	3	30
39	124	68	65	101

Layer 4

52	4	111	112	95
92	25	113	23	89
64	123	18	33	87
96	122	29	100	70
105	69	117	61	50

Layer 5

Layer 5

27	73	76	74	108
36	16	104	57	40
56	116	35	41	88
11	97	106	45	9
43	93	79	75	7

Final State

[Show 3D Final State](#)

Layer 1

8	94	119	58	37
125	42	16	25	107
57	6	121	53	78
76	87	19	99	34
45	86	41	81	59

Layer 2

33	101	12	120	49
60	28	79	97	51
104	55	72	64	20
63	109	35	21	84
54	22	117	13	111

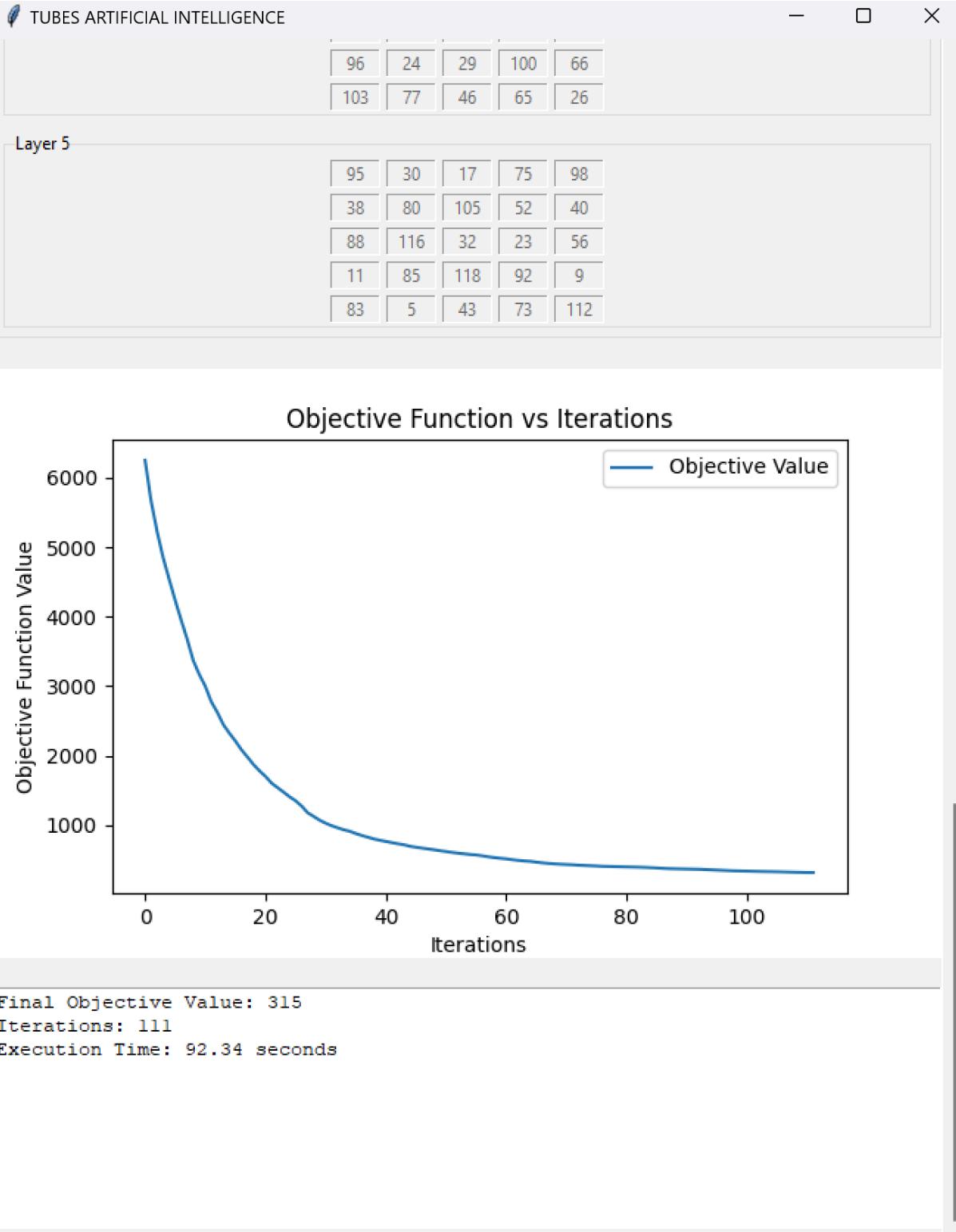
Layer 3

110	93	61	15	36
48	74	1	102	90
68	14	70	113	50
67	10	114	3	122
31	124	71	82	7

Layer 4

69	2	106	47	91
44	89	115	39	27
4	123	18	62	108
96	24	29	100	66
103	77	46	65	26

Layer 5



- Eksperimen 2

TUBES ARTIFICIAL INTELLIGENCE

Steepest Ascent Algorithm

[Start](#)

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

52	45	122	4	24
96	44	105	38	49
63	31	103	80	29
25	46	41	47	7
115	86	37	69	43

Layer 2

8	51	91	84	111
64	73	62	120	102
125	9	92	82	72
106	48	22	35	61
89	10	30	112	87

Layer 3

42	123	95	75	12
14	117	18	104	124
58	34	97	81	68
94	17	70	98	36
66	55	53	13	26

Layer 4

90	1	99	33	79
6	101	19	50	59
76	109	32	93	65
16	57	88	60	116
21	100	83	5	54

Layer 5



Layer 5

114	20	2	119	71
78	56	85	15	113
107	74	108	3	23
40	27	11	67	77
39	121	28	118	110

Final State

[Show 3D Final State](#)

Layer 1

90	98	2	72	60
96	40	125	45	10
29	28	95	82	81
48	62	30	47	122
52	88	63	69	43

Layer 2

9	58	83	84	80
68	77	56	110	4
44	11	92	78	91
106	116	22	38	27
89	49	61	5	113

Layer 3

42	123	104	33	12
14	46	26	103	124
51	97	15	57	99
94	25	70	109	18
114	24	100	13	64

Layer 4

115	1	119	6	73
65	121	19	50	59
79	105	16	93	20
35	36	86	55	111
21	53	75	112	54

Layer 5

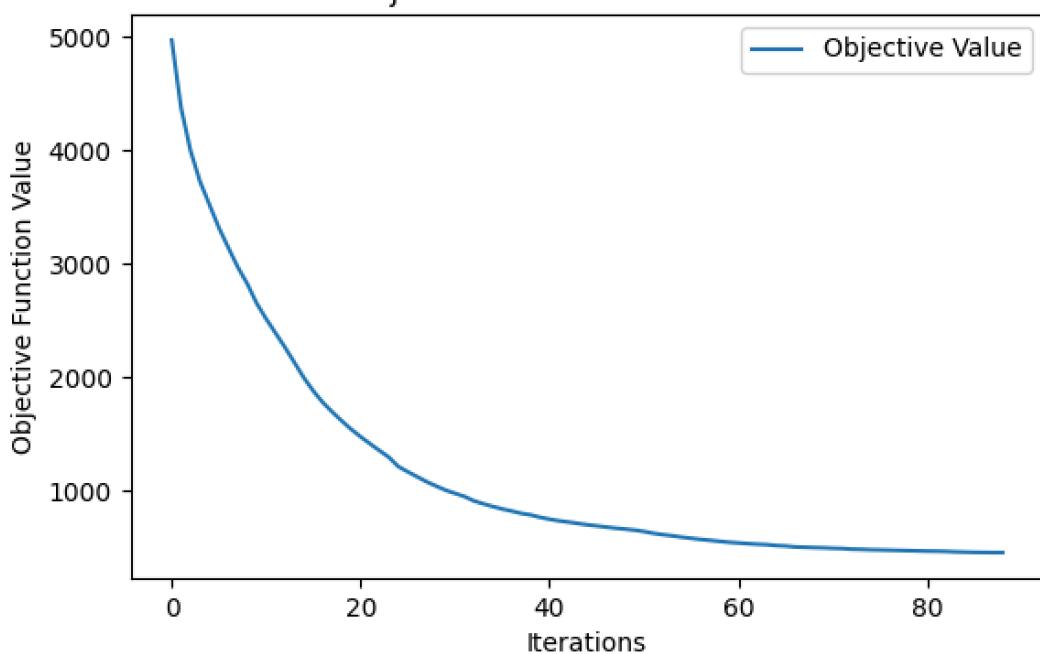


35	36	86	55	111
21	53	75	112	54

Layer 5

66	34	7	120	87
71	31	85	8	118
108	74	107	3	23
32	76	102	67	37
39	101	17	117	41

Objective Function vs Iterations



Final Objective Value: 454
Iterations: 88
Execution Time: 76.62 seconds

- Eksperimen 3

TUBES ARTIFICIAL INTELLIGENCE

Steepest Ascent Algorithm

[Start](#)

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

102	93	55	54	10
71	26	36	58	90
73	113	77	40	21
59	20	30	43	85
98	37	9	44	48

Layer 2

17	112	101	106	51
39	82	60	23	105
84	28	117	16	57
41	18	47	45	64
76	75	65	13	80

Layer 3

63	107	3	94	124
125	89	69	35	33
50	12	56	66	110
91	81	11	38	49
8	25	86	19	99

Layer 4

96	70	88	108	78
87	83	24	7	116
111	53	114	61	29
32	100	62	74	67
34	109	115	2	14

Layer 5



Layer 5

22	118	92	122	46
6	5	121	42	1
123	15	119	79	31
4	95	120	103	52
97	27	104	72	68

Final State

[Show 3D Final State](#)

Layer 1

86	90	78	46	15
53	26	40	106	88
22	117	82	19	75
59	18	110	43	85
95	64	7	102	52

Layer 2

84	8	24	125	57
45	66	54	44	105
17	118	116	28	36
114	20	47	97	37
55	92	74	13	80

Layer 3

10	104	70	11	120
124	89	69	29	4
62	16	56	122	58
109	81	61	34	32
9	25	60	119	101

Layer 4

96	3	49	108	76
6	71	31	94	113
115	51	41	73	35
33	100	67	38	77
65	107	123	2	14

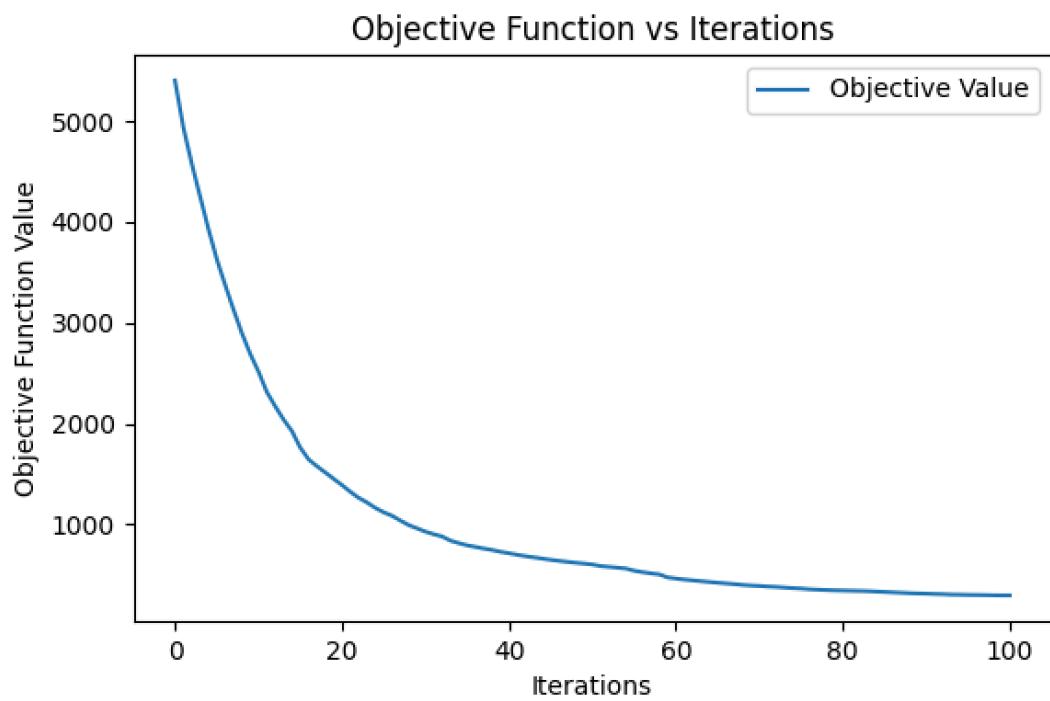
Layer 5

TUBES ARTIFICIAL INTELLIGENCE

33	100	67	38	77
65	107	123	2	14

Layer 5

39	112	93	23	48
87	63	121	42	5
99	12	21	72	111
1	98	30	103	83
91	27	50	79	68



Final Objective Value: 293
Iterations: 100
Execution Time: 84.95 seconds

b. Analisis

Dalam eksperimen yang dilakukan, algoritma ini menunjukkan bahwa ia dapat mendekati global optima, yang ditandai dengan nilai objective mendekati nol. Meskipun tidak selalu mencapai nol secara sempurna, nilai objective akhir biasanya berkisar antara 200 hingga 400, yang menandakan perbaikan signifikan dari kondisi awal di mana nilai objective bisa mencapai lebih dari 4000. Hal ini menunjukkan bahwa algoritma Steepest Ascent cukup efektif dalam memperbaiki solusi dari iterasi ke iterasi meskipun cenderung terkunci pada local optima akibat strategi pergerakannya yang tidak mempertimbangkan nilai yang sama atau lebih buruk dari nilai saat ini.

Algoritma Steepest Ascent dapat mendekati global optima dalam banyak percobaan, tetapi sering kali terhenti di local optima. Kelemahan ini disebabkan oleh pendekatannya yang hanya memilih successor terbaik pada setiap langkah tanpa mempertimbangkan kemungkinan menerima solusi yang sedikit lebih buruk untuk eksplorasi lebih lanjut. Walaupun demikian, hasil yang dicapai oleh algoritma ini sudah cukup dekat dengan solusi optimal, dengan nilai objective akhir yang signifikan lebih rendah dibandingkan kondisi awal.

Dari segi durasi, algoritma Steepest Ascent cenderung membutuhkan waktu lebih lama dibandingkan beberapa algoritma lain. Hal ini disebabkan oleh proses pemilihan successor terbaik di antara semua kemungkinan yang ada, yang jumlahnya sangat banyak. Kompleksitas perhitungan nilai objektif yang cukup rumit juga mempengaruhi kecepatan eksekusi algoritma ini, membuatnya lebih lambat dibandingkan algoritma yang menggunakan pendekatan seleksi acak atau probabilistik.

Hasil eksperimen menunjukkan bahwa algoritma ini cukup konsisten, dengan hasil yang selalu berada dalam rentang yang tidak jauh berbeda. Namun, jika algoritma dijalankan lebih banyak lagi dengan

kondisi awal yang berbeda, konsistensi tersebut mungkin menjadi kurang stabil. Kondisi awal Cube sangat mempengaruhi hasil akhir, karena jalur perbaikan yang diambil oleh algoritma bergantung pada keadaan awal Cube tersebut.

Pada masalah magic cube, perbaikan nilai objective function cenderung semakin kecil seiring bertambahnya iterasi karena pada tahap awal, perbaikan signifikan mudah ditemukan ketika state masih jauh dari solusi optimal. Namun, saat algoritma Steepest Ascent mendekati solusi, perbaikan menjadi lebih sulit ditemukan karena sifat greedy algoritma ini hanya memilih langkah terbaik di setiap iterasi tanpa mempertimbangkan eksplorasi yang lebih luas. Selain itu, ruang solusi yang tersedia untuk perbaikan semakin terbatas menyebabkan perbaikan yang ditemukan semakin kecil. Akibatnya, algoritma cenderung terjebak di local optima, dimana perbaikan tambahan hanya sedikit atau tidak ada sama sekali. Berdasarkan penjelasan dan eksperimen yang telah dilakukan bisa dikatakan bahwa memang algoritma ini kekurangan kemampuan eksplorasi pada tahap awal iterasi.

2. Hill Climbing with Sideways Move

a. Eksperimen

- Eksperimen 1

TUBES ARTIFICIAL INTELLIGENCE

Sideways Move HC Algorithm

Max Sideways Move:

Start

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

70	55	67	111	23
83	90	80	103	93
28	114	124	26	101
51	52	58	56	37
9	64	123	4	72

Layer 2

94	43	10	21	75
110	33	8	2	59
125	106	40	34	20
5	54	95	32	74
108	39	45	79	73

Layer 3

119	76	1	49	35
86	92	60	44	48
63	89	22	109	68
29	62	78	91	38
47	113	3	46	13

Layer 4

16	25	81	36	53
57	105	97	102	12
122	69	24	7	18
47	30	66	44	15

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

16	25	81	36	53
57	105	97	102	12
122	69	24	7	18
17	30	66	115	15
120	100	117	41	98

Layer 5

82	31	50	116	6
42	88	99	65	84
14	107	71	19	96
104	61	11	121	77
87	85	27	112	118

Final State

Show 3D Final State

Layer 1

69	110	7	108	22
79	15	11	112	97
25	35	114	34	103
117	53	58	56	32
26	102	123	4	61

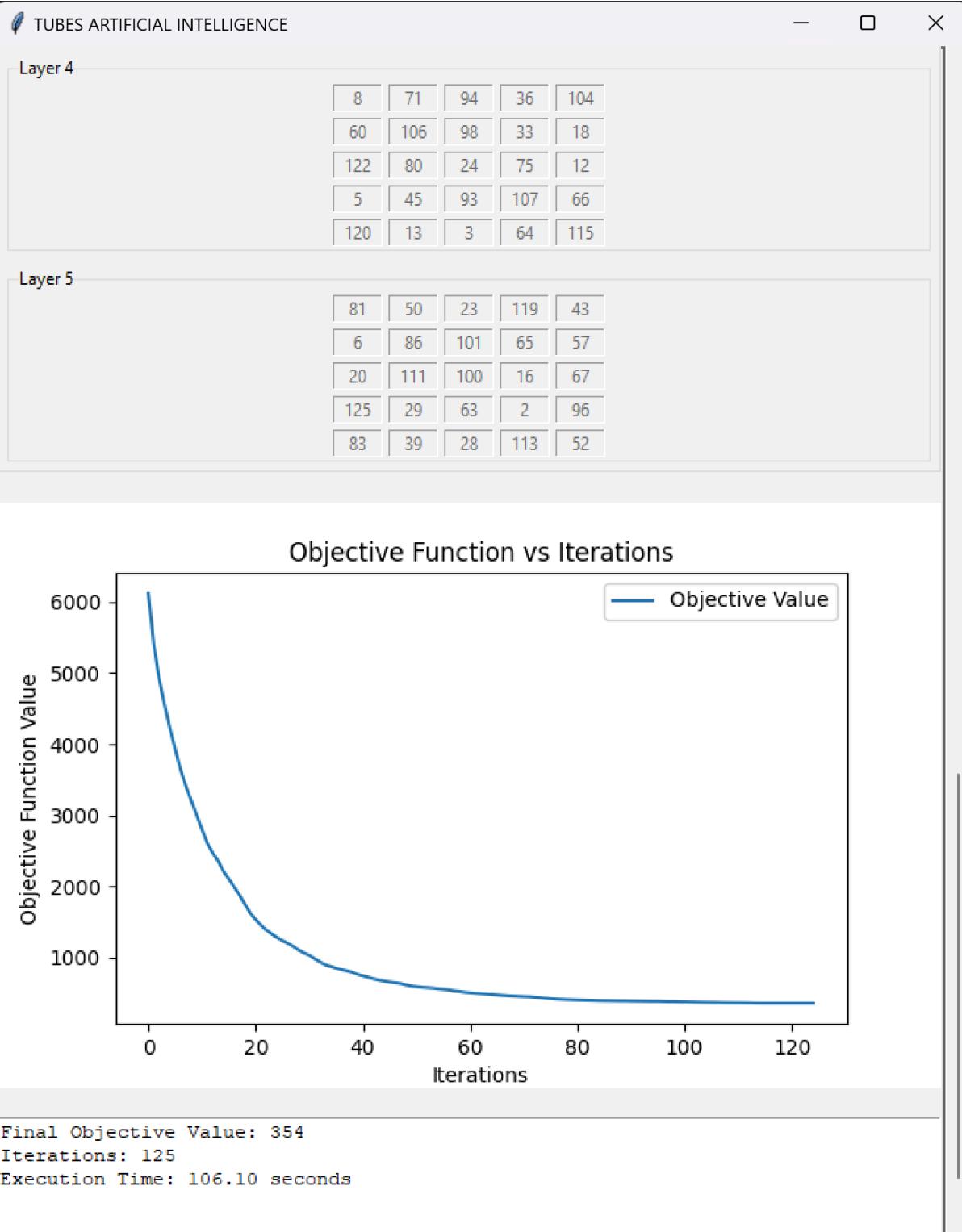
Layer 2

90	42	70	21	92
82	31	51	95	59
76	89	40	85	17
19	109	91	27	74
48	44	62	87	73

Layer 3

68	41	121	30	55
88	77	54	10	84
72	1	37	105	116
49	78	9	124	47
38	118	99	46	14

Layer 4



- Eksperimen 2

TUBES ARTIFICIAL INTELLIGENCE

Sideways Move HC Algorithm

Max Sideways Move:

Start

Back to Main Menu

Initial State

Show 3D Initial State

Layer 1

106	70	57	9	101
30	56	72	29	103
121	18	60	54	75
69	125	100	50	63
12	26	123	71	49

Layer 2

108	4	82	46	8
73	124	19	116	112
89	36	14	22	37
102	79	62	55	76
39	114	23	28	88

Layer 3

27	81	95	109	16
15	38	115	5	74
86	66	93	47	80
58	13	24	120	45
107	44	34	119	105

Layer 4

111	41	77	48	99
31	97	64	122	68
43	67	42	32	94
117	11	11	11	11

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

111	41	77	48	99
31	97	64	122	68
43	67	42	32	94
117	92	52	10	40
78	83	11	98	91

Layer 5

53	1	85	84	90
118	65	51	59	61
113	35	21	17	6
2	104	7	33	96
87	110	25	3	20

Final State

Show 3D Final State

Layer 1

105	112	2	11	84
30	60	70	52	103
121	18	42	73	61
40	119	87	56	13
19	7	115	123	53

Layer 2

95	57	83	66	9
6	94	22	120	71
101	23	100	54	37
110	12	62	55	75
3	118	49	21	124

Layer 3

26	106	39	109	35
79	65	116	5	50
28	67	88	47	85
59	63	24	125	44
122	14	46	29	104

Layer 4



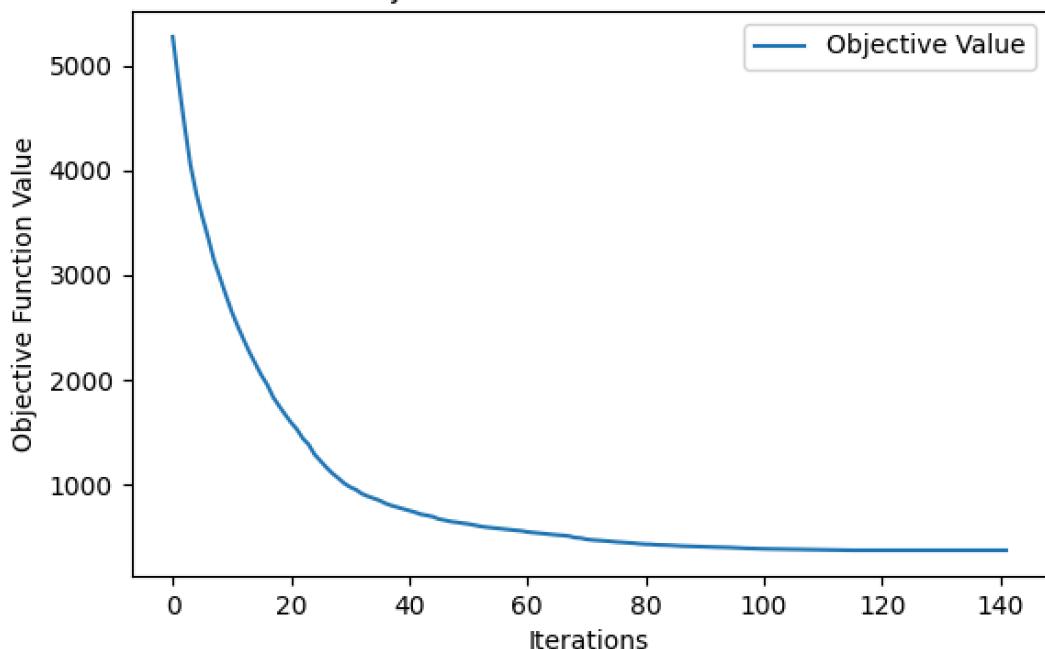
Layer 4

17	41	111	48	98
86	45	64	107	15
34	92	58	33	96
102	38	74	10	90
78	99	8	117	16

Layer 5

72	1	80	82	89
114	51	43	31	76
32	113	27	108	36
4	81	68	69	93
91	77	97	25	20

Objective Function vs Iterations



Final Objective Value: 374

Iterations: 142

Execution Time: 126.76 seconds

- Eksperimen 3

TUBES ARTIFICIAL INTELLIGENCE

Sideways Move HC Algorithm

Max Sideways Move:

Start

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

25	69	63	18	39
115	19	37	46	53
11	82	87	15	28
104	91	103	30	121
80	3	51	21	43

Layer 2

48	88	102	47	50
105	58	14	2	10
112	79	71	5	1
23	107	123	55	35
6	4	94	62	45

Layer 3

74	33	85	60	13
76	114	56	111	120
34	41	97	26	75
42	40	38	125	9
108	70	68	54	99

Layer 4

119	8	96	90	86
61	122	57	113	36
66	27	77	110	59
20	81	21	52	66

Layer 4

119	8	96	90	86
61	122	57	113	36
66	27	77	110	59
20	81	31	52	65
22	95	116	64	93

Layer 5

92	106	89	29	7
109	16	117	73	100
84	44	49	12	24
101	72	78	83	17
124	32	98	67	118

Final State

[Show 3D Final State](#)

Layer 1

25	83	39	114	54
115	19	33	45	104
9	78	111	107	10
80	12	94	30	98
86	123	38	22	49

Layer 2

8	89	102	65	50
105	58	36	110	6
112	59	43	5	96
21	109	35	29	121
69	4	95	106	42

Layer 3

73	27	85	31	99
11	113	55	16	120
60	41	62	77	75
101	40	44	124	7
70	103	63	66	13

Layer 4

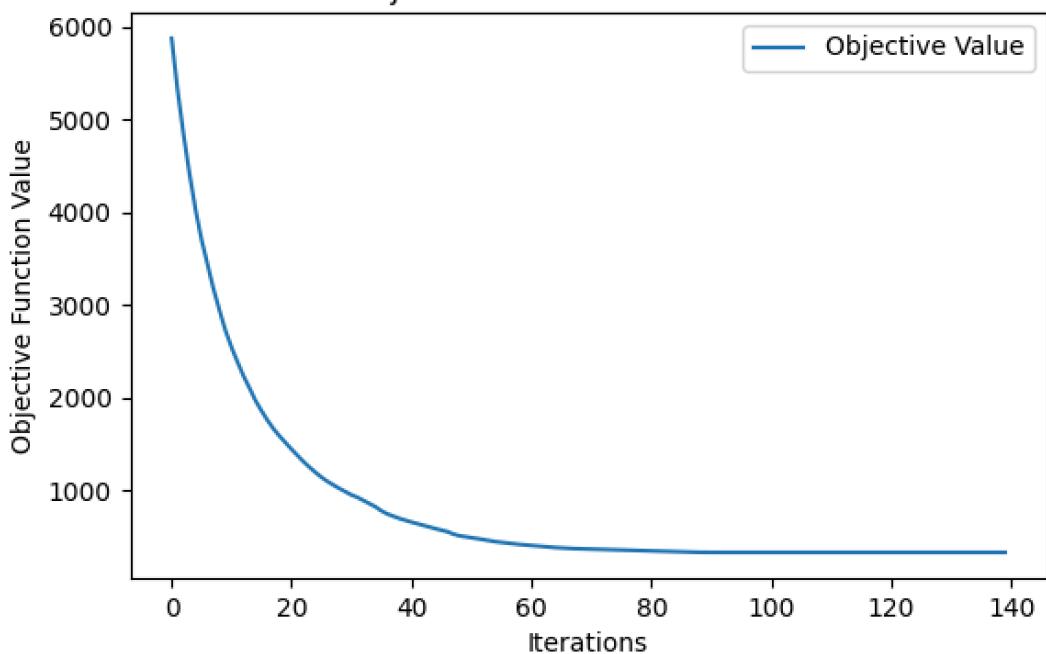
Layer 4

117	20	1	90	87
68	122	71	18	37
51	46	76	108	34
56	82	61	52	64
23	32	116	47	93

Layer 5

92	97	88	14	28
15	2	119	125	48
84	91	24	17	100
57	72	81	79	26
67	53	3	74	118

Objective Function vs Iterations



Final Objective Value: 329
Iterations: 140
Execution Time: 114.17 seconds

b. Analisis

Dalam eksperimen yang dilakukan, maksimum sideways move yang dieksperimenkan adalah 10, 30, dan 50. Meskipun jarang mencapai nol secara sempurna, nilai objektif akhir berkisar antara 300 hingga 400, yang menandakan perbaikan signifikan dari kondisi awal di mana nilai objektif dapat mencapai lebih dari 5000. Algoritma ini menunjukkan bahwa ia dapat mendekati global optima dengan lebih baik dibandingkan Steepest Ascent, dengan mengizinkan eksplorasi untuk dilanjutkan ketika nilai tetangganya masih sama dengan nilai current state. Hal ini menunjukkan bahwa algoritma Hill Climbing with Sideways Move cukup efektif dalam memperbaiki solusi dari iterasi ke iterasi, dengan kemampuan untuk menghindari terkuncinya pergerakan pada local optima yang sering kali menjadi kelemahan Steepest Ascent.

Pada algoritma ini, kemampuan untuk menerima solusi yang nilainya sama dengan nilai saat ini memungkinkan eksplorasi yang lebih luas ketika algoritma terjebak di local optima. Namun, kelemahan dari pendekatan ini adalah meningkatnya risiko algoritma menghabiskan terlalu banyak waktu dalam pencarian di sekitar satu local optima tanpa perbaikan signifikan, terutama jika jumlah gerakan sideways tidak dibatasi. Untuk mengatasi hal ini, diterapkan batasan jumlah sideways move, sehingga algoritma akan mencari solusi yang lebih menjanjikan setelah melakukan sejumlah gerakan sideways yang diperbolehkan.

Durasi eksekusi algoritma Hill Climbing with Sideways Move cenderung lebih lama dan jumlah iterasinya lebih banyak dibandingkan Steepest Ascent. Hal ini karena adanya tambahan langkah sideways yang memungkinkan eksplorasi lebih lanjut dalam ruang solusi. Kompleksitas perhitungan nilai objektif dan pilihan sideways yang dilakukan berkontribusi pada waktu eksekusi yang lebih besar dibandingkan algoritma dengan pendekatan seleksi acak atau

probabilistik. Namun, waktu eksekusi tambahan ini seringkali memberikan perbaikan yang signifikan pada nilai objektif akhir.

Hasil eksperimen menunjukkan bahwa algoritma ini cukup konsisten, dengan nilai objektif akhir yang relatif stabil dan berkisar dalam rentang tertentu. Namun, jika algoritma dijalankan lebih banyak dengan kondisi awal yang berbeda, konsistensi tersebut dapat berubah karena adanya variasi hasil tergantung pada jumlah gerakan sideways yang dilakukan. Kondisi awal Cube juga tetap mempengaruhi hasil akhir, terutama karena jalur perbaikan yang diambil algoritma bergantung pada keadaan awal tersebut.

Pada masalah *magic cube*, perbaikan nilai objektif cenderung semakin kecil seiring bertambahnya iterasi karena di awal, perbaikan signifikan mudah ditemukan ketika state masih jauh dari solusi optimal. Akan tetapi, saat algoritma mendekati solusi, perbaikan menjadi lebih sulit ditemukan karena semakin terbatasnya ruang solusi. Meskipun algoritma *Hill Climbing with Sideways Move* memiliki kemampuan eksplorasi tambahan, tetap ada kecenderungan terjebak di sekitar local optima setelah batasan sideways move tercapai. Berdasarkan eksperimen yang dilakukan, dapat disimpulkan bahwa meskipun algoritma ini memiliki kemampuan eksplorasi yang lebih baik dibandingkan *Steepest Ascent*, keterbatasan sideways move masih menjadi faktor yang perlu diperhatikan agar algoritma tidak terus-menerus terjebak pada local optima.

3. Random Restart Hill Climbing

a. Eksperimen

- Eksperimen 1

TUBES ARTIFICIAL INTELLIGENCE

Random Restart HC Algorithm

Max Restarts:

Start

[Back to Main Menu](#)

Initial State

Show 3D Initial State

Layer 1

101	40	84	118	18
109	13	49	1	91
26	52	78	93	42
106	6	29	94	72
77	9	23	64	87

Layer 2

115	24	58	98	100
62	32	105	96	31
25	122	30	111	82
10	70	90	88	83
104	89	66	86	47

Layer 3

123	28	80	119	45
110	69	3	37	48
27	4	35	67	59
50	16	71	11	74
17	112	60	114	54

Layer 4

14	56	33	46	15
117	2	19	95	113
39	124	38	121	51

Layer 4

14	56	33	46	15
117	2	19	95	113
39	124	38	121	51
34	55	44	125	92
116	68	57	75	53

Layer 5

36	65	41	79	108
8	97	102	22	20
73	43	103	85	5
120	21	107	7	76
61	63	81	12	99

Final State

[Show 3D Final State](#)

Layer 1

25	35	84	118	53
120	86	45	7	57
26	52	80	71	87
103	123	11	36	42
40	19	95	82	79

Layer 2

109	58	24	16	108
62	17	104	101	31
28	88	30	59	111
14	63	91	124	22
102	89	66	15	43

Layer 3

121	27	77	46	44
33	112	55	37	78
117	4	49	90	54
32	100	70	29	83
6	72	64	114	56

Layer 4

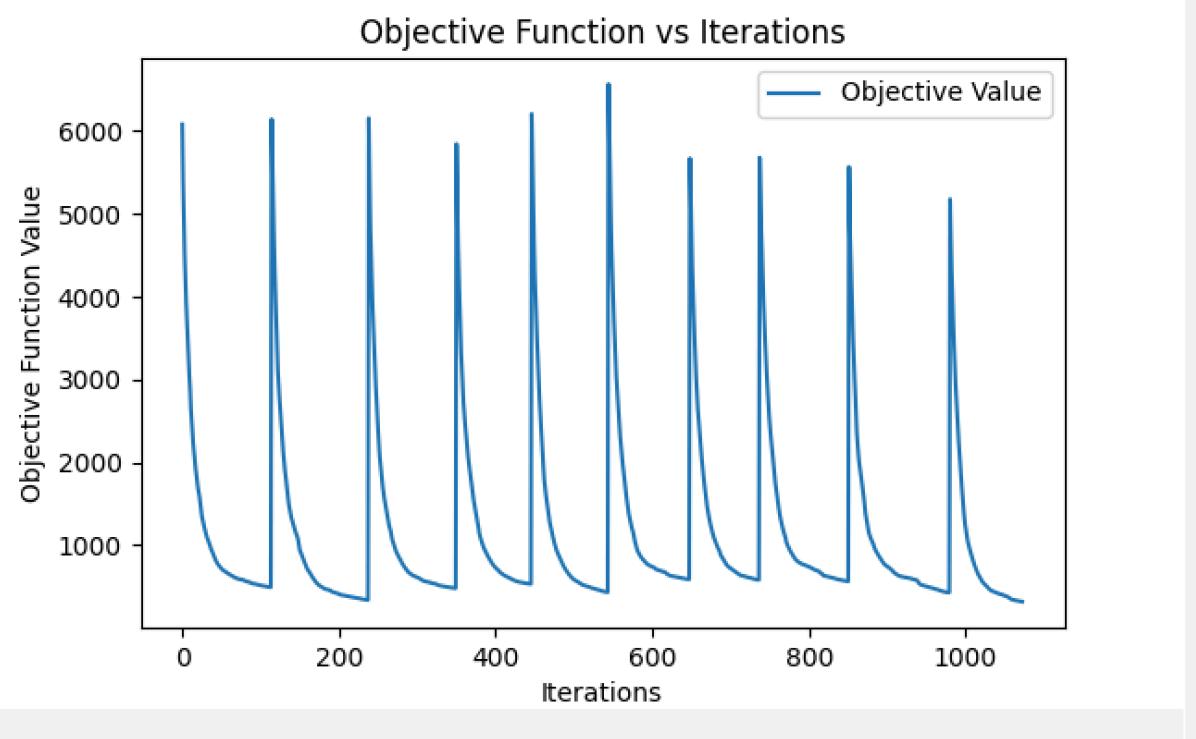
TUBES ARTIFICIAL INTELLIGENCE

Layer 4

12	119	110	60	18
96	2	5	97	113
38	122	94	10	51
50	8	39	125	93
115	69	67	23	41

Layer 5

48	76	20	75	92
3	98	107	73	34
106	47	68	85	9
116	21	105	1	74
61	65	13	81	99



```

Final Objective Value: 318
Many Restart: 10
Execution Time: 919.94 seconds
Iterations per Restart:
  Restart 1: 113 iterations
  Restart 2: 123 iterations

```

```
Restart 2: 123 iterations
Restart 3: 111 iterations
Restart 4: 95 iterations
Restart 5: 97 iterations
Restart 6: 103 iterations
Restart 7: 88 iterations
Restart 8: 113 iterations
Restart 9: 128 iterations
Restart 10: 92 iterations
```

- Eksperimen 2

TUBES ARTIFICIAL INTELLIGENCE

Random Restart HC Algorithm

Max Restarts:

Start

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

10	33	41	26	103
42	20	11	4	92
95	123	61	65	59
78	24	71	111	73
83	66	48	34	112

Layer 2

57	6	76	81	17
7	99	109	60	121
8	55	108	84	93
44	51	101	54	28
91	102	70	117	29

Layer 3

46	122	13	85	39
53	116	47	67	69
94	113	62	58	124
12	52	104	22	110
100	88	21	64	3

Layer 4

50	25	119	105	125
90	40	98	120	2
1	72	45	80	68

Layer 4

50	25	119	105	125
90	40	98	120	2
1	72	45	80	68
79	89	35	37	75
9	15	49	77	115

Layer 5

97	43	82	118	56
107	31	36	106	23
19	5	74	86	63
87	114	14	27	96
16	38	30	32	18

Final State

[Show 3D Final State](#)

Layer 1

92	37	41	74	72
121	42	26	19	107
15	122	47	77	50
30	109	123	51	2
57	3	78	93	84

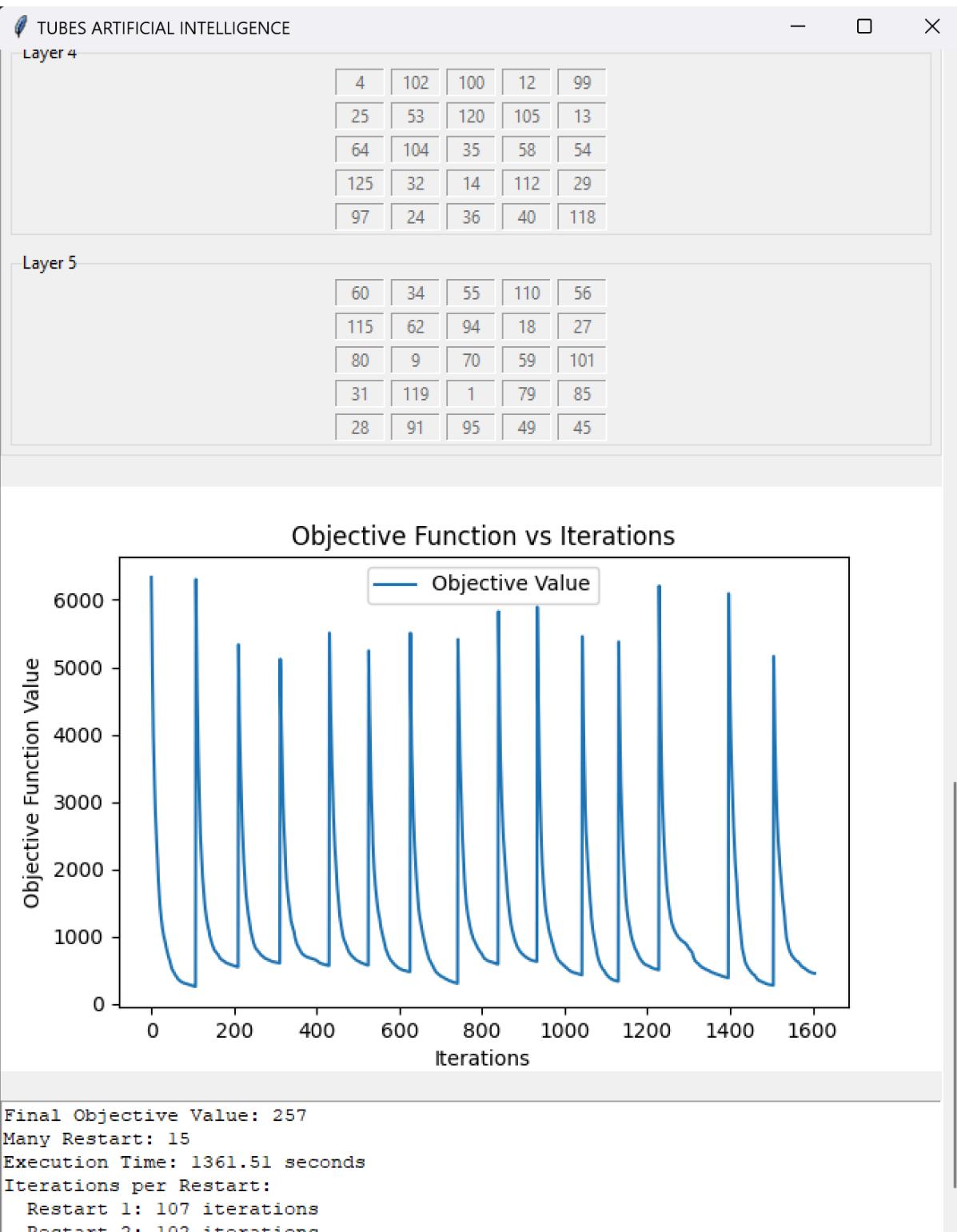
Layer 2

90	83	48	10	81
6	43	68	117	82
89	73	98	33	23
113	20	63	44	75
17	96	39	111	52

Layer 3

69	61	71	108	7
46	106	5	76	86
66	8	65	88	87
16	38	114	22	124
116	103	67	21	11

Layer 4



```
Restart 2: 102 iterations
Restart 3: 100 iterations
Restart 4: 118 iterations
Restart 5: 94 iterations
Restart 6: 100 iterations
Restart 7: 114 iterations
Restart 8: 97 iterations
Restart 9: 93 iterations
Restart 10: 108 iterations
Restart 11: 87 iterations
```

```
Restart 7: 114 iterations
Restart 8: 97 iterations
Restart 9: 93 iterations
Restart 10: 108 iterations
Restart 11: 87 iterations
Restart 12: 97 iterations
Restart 13: 167 iterations
Restart 14: 108 iterations
Restart 15: 99 iterations
```

- Eksperimen 3

TUBES ARTIFICIAL INTELLIGENCE

Random Restart HC Algorithm

Max Restarts:

Start

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

46	105	40	57	96
107	95	29	5	33
65	100	27	50	71
86	9	123	59	67
34	85	2	64	108

Layer 2

69	22	15	75	121
124	122	80	89	32
42	76	16	37	87
39	68	6	66	56
91	13	21	125	7

Layer 3

1	54	117	78	99
114	93	119	90	4
10	44	82	47	26
20	110	63	88	118
97	109	3	62	31

Layer 4

103	55	36	8	101
17	104	74	48	70
77	102	81	60	115

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

103	55	36	8	101
17	104	74	48	70
77	102	81	60	115
45	79	113	73	116
98	112	58	111	14

Layer 5

18	84	12	83	35
49	41	61	38	106
72	52	92	19	120
43	25	51	53	28
24	30	11	23	94

Final State

Show 3D Final State

Layer 1

62	66	111	21	54
92	121	18	58	26
31	13	29	117	124
69	113	51	38	46
61	2	107	80	65

Layer 2

100	16	28	105	63
23	20	118	33	116
35	123	98	4	55
76	41	49	94	56
81	114	22	79	19

Layer 3

101	9	36	77	95
27	85	11	102	90
103	14	122	67	7
6	119	125	24	40
78	88	17	45	87

Layer 4

TUBES ARTIFICIAL INTELLIGENCE

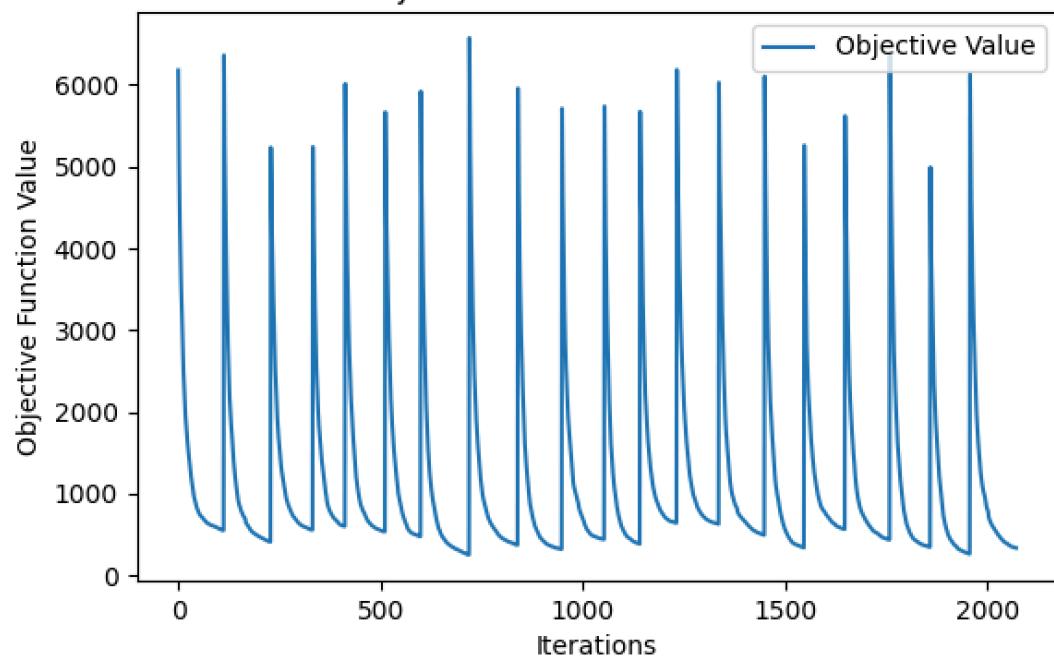
Layer 4

48	115	34	108	10
64	37	96	32	86
104	68	12	84	47
91	25	60	39	99
8	70	112	52	73

Layer 5

5	109	106	3	93
110	53	72	89	1
42	97	50	44	82
75	15	30	120	74
83	43	57	59	71

Objective Function vs Iterations



```

Final Objective Value: 254
Many Restart: 20
Execution Time: 1785.98 seconds
Iterations per Restart:
  Restart 1: 112 iterations
  ----- . . . -----

```

```
Restart 2: 115 iterations
Restart 3: 103 iterations
Restart 4: 79 iterations
Restart 5: 98 iterations
Restart 6: 87 iterations
Restart 7: 119 iterations
Restart 8: 119 iterations
Restart 9: 108 iterations
Restart 10: 104 iterations
Restart 11: 87 iterations
```

```
Restart 12: 90 iterations
Restart 13: 103 iterations
Restart 14: 112 iterations
Restart 15: 97 iterations
Restart 16: 100 iterations
Restart 17: 110 iterations
Restart 18: 99 iterations
Restart 19: 97 iterations
Restart 20: 114 iterations
```

b. Analisis

Dalam eksperimen ini, algoritma *Random Restart Hill Climbing* diujikan dengan maksimum *restart* sebanyak 10, 15, dan 20 untuk mengeksplorasi ruang solusi secara lebih menyeluruh dan menghindari local optima. Setiap *restart* dilakukan hingga tidak ada perbaikan lebih lanjut dalam iterasi atau mencapai batas maksimum iterasi yang telah ditentukan. Hasil dari tiap *restart* dicatat, dan nilai objektif terbaik dipilih sebagai solusi akhir.

Pada eksperimen dengan maksimum *restart* 10, algoritma menghasilkan nilai objektif terbaik sebesar 318. Hal ini menunjukkan perbaikan yang signifikan dari kondisi awal, dengan rata-rata nilai objektif dari setiap *restart* mengalami penurunan yang konsisten. Namun, dengan jumlah *restart* yang terbatas, algoritma masih mungkin terkunci pada local optima yang menghambat pencapaian solusi yang lebih mendekati *global optimum*.

Pada eksperimen dengan maksimum *restart* 15, algoritma berhasil mencapai nilai objektif terbaik sebesar 257. Tambahan lima *restart* memberikan algoritma lebih banyak kesempatan untuk mengeksplorasi

ruang solusi, sehingga mampu menemukan solusi yang lebih baik dibandingkan dengan batas *restart* 10. Peningkatan jumlah *restart* ini memungkinkan algoritma untuk memulai konfigurasi awal kembali dan keluar dari local optima yang sebelumnya menghambat perbaikan, sehingga memberikan hasil yang lebih optimal secara signifikan.

Pada eksperimen dengan maksimum *restart* 20, nilai objektif terbaik yang diperoleh adalah 254. Meskipun hasil ini sedikit lebih baik dibandingkan *restart* 15, perbedaannya tidak signifikan. Hal ini mengindikasikan bahwa setelah sejumlah *restart* tertentu, algoritma mulai mencapai batas efektifitasnya, di mana tambahan *restart* tidak selalu menghasilkan perbaikan yang signifikan pada nilai objektif. Meskipun demikian, batas *restart* yang lebih besar memberikan peluang untuk menemukan solusi yang lebih baik secara konsisten.

Algoritma *Random Restart Hill Climbing* menunjukkan bahwa ia dapat mendekati global optima dengan cukup baik. Algoritma ini melakukan beberapa kali restart dari posisi acak baru setiap kali terjebak di local optima, yang ditandai dengan perbaikan nilai objektif yang terus mendekati nol. Meskipun tidak selalu mencapai nol secara sempurna, nilai objektif akhir biasanya berkisar antara 200 hingga 400, yang menunjukkan perbaikan signifikan dari kondisi awal di mana nilai objektif dapat mencapai lebih dari 4000 pada setiap restartnya. Kemampuan untuk memulai kembali dari posisi baru memungkinkan algoritma ini untuk menghindari local optima yang menjadi kelemahan pada algoritma tanpa restart.

Dengan adanya restart, algoritma ini tidak terjebak terlalu lama di satu local optima, sehingga eksplorasi ruang solusi menjadi lebih efektif. Namun, kelemahan dari pendekatan ini adalah peningkatan jumlah iterasi yang diperlukan untuk menemukan solusi, terutama pada kasus-kasus di mana beberapa restart diperlukan sebelum mencapai solusi yang lebih optimal.

Durasi eksekusi algoritma *Random Restart Hill Climbing* cenderung lebih lama dibandingkan algoritma *Hill Climbing* tanpa restart, tetapi waktu eksekusi ini sering kali memberikan hasil yang lebih baik. Kompleksitas perhitungan nilai objektif dan proses restart mempengaruhi waktu eksekusi, terutama karena algoritma harus mengevaluasi kembali setiap posisi awal baru. Meskipun demikian, keuntungan yang diperoleh dari restart ini adalah peningkatan peluang untuk menemukan solusi yang lebih optimal daripada jika hanya mengandalkan satu pencarian tunggal.

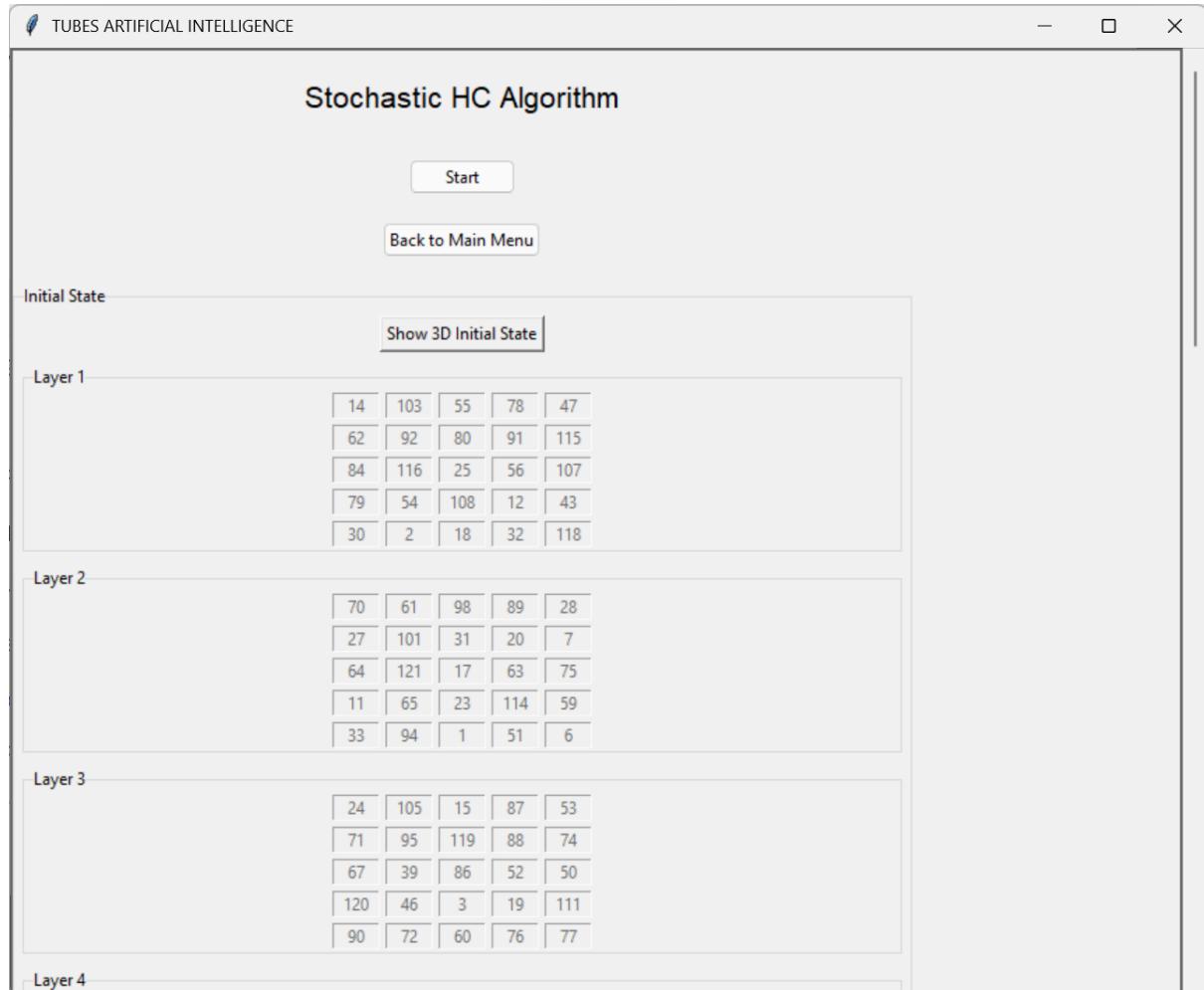
Hasil eksperimen menunjukkan bahwa algoritma ini cukup konsisten, dengan nilai objektif akhir yang berada dalam rentang tertentu pada setiap percobaan. Konsistensi ini dapat berkurang jika algoritma dijalankan lebih banyak dengan kondisi awal yang sangat beragam, tetapi secara umum hasilnya tetap dalam kisaran yang optimal. Jumlah restart dan posisi acak dari setiap restart sangat mempengaruhi hasil akhir, karena jalur perbaikan yang diambil algoritma bergantung pada keadaan awal tersebut.

Namun, efektivitas tambahan *restart* cenderung menurun setelah titik tertentu, yang ditunjukkan oleh sedikitnya perbedaan antara hasil terbaik pada *restart* 15 dan *restart* 20. Berdasarkan hasil ini, batas *restart* yang optimal dapat dipilih sesuai dengan kebutuhan waktu komputasi dan keinginan untuk mencapai solusi yang lebih optimal, dengan rekomendasi batas *restart* antara 15 hingga 20 untuk hasil yang optimal tanpa memperpanjang durasi eksekusi secara signifikan.

4. Stochastic Hill Climbing

a. Eksperimen

- Eksperimen 1



TUBES ARTIFICIAL INTELLIGENCE

Layer 4

73	5	110	42	112
97	117	106	26	13
125	66	22	36	57
8	83	122	99	10
102	40	21	44	4

Layer 5

93	100	81	34	58
68	35	69	45	96
41	85	48	16	113
123	82	104	29	49
109	37	9	38	124

Final State

Show 3D Final State

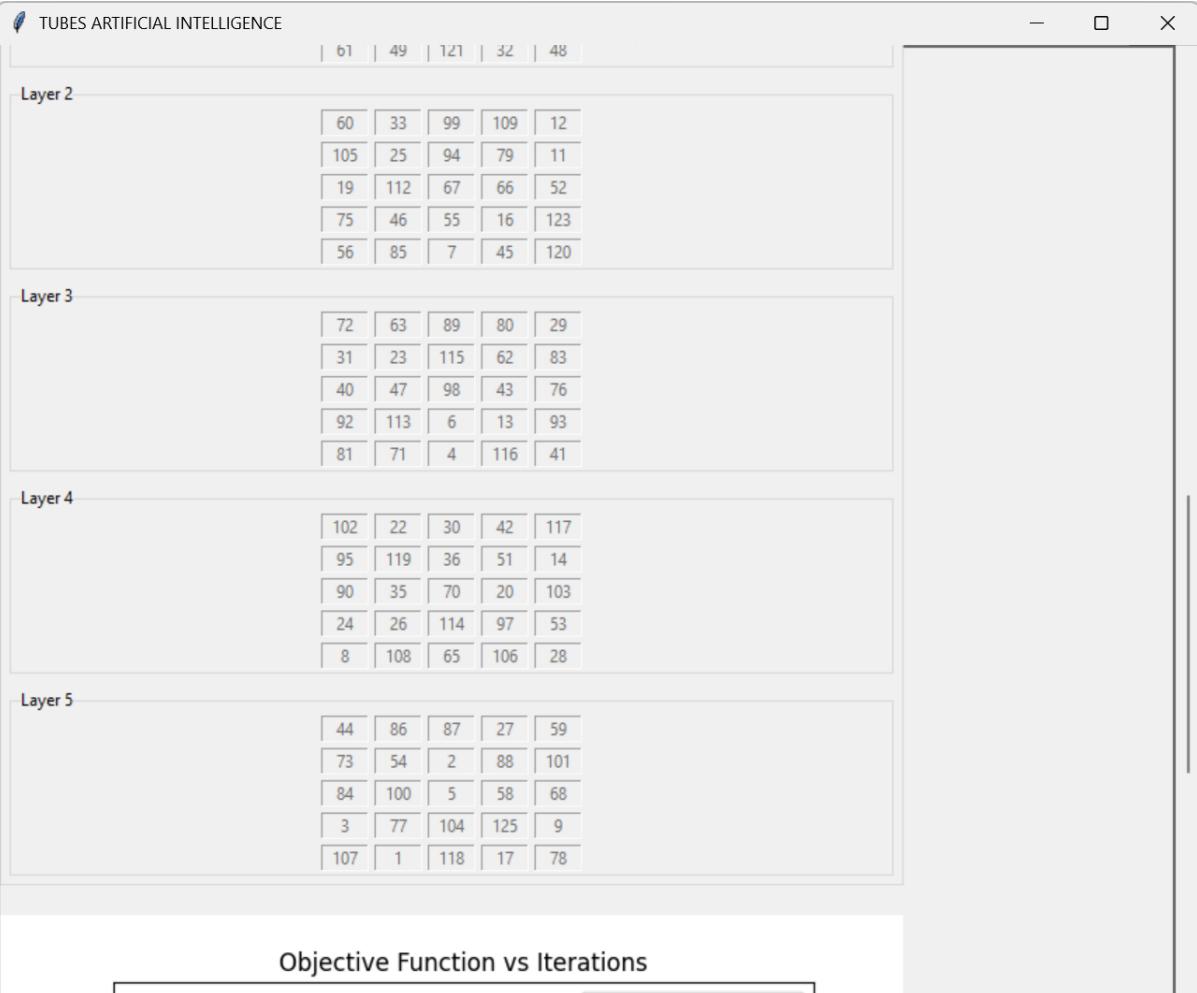
Layer 1

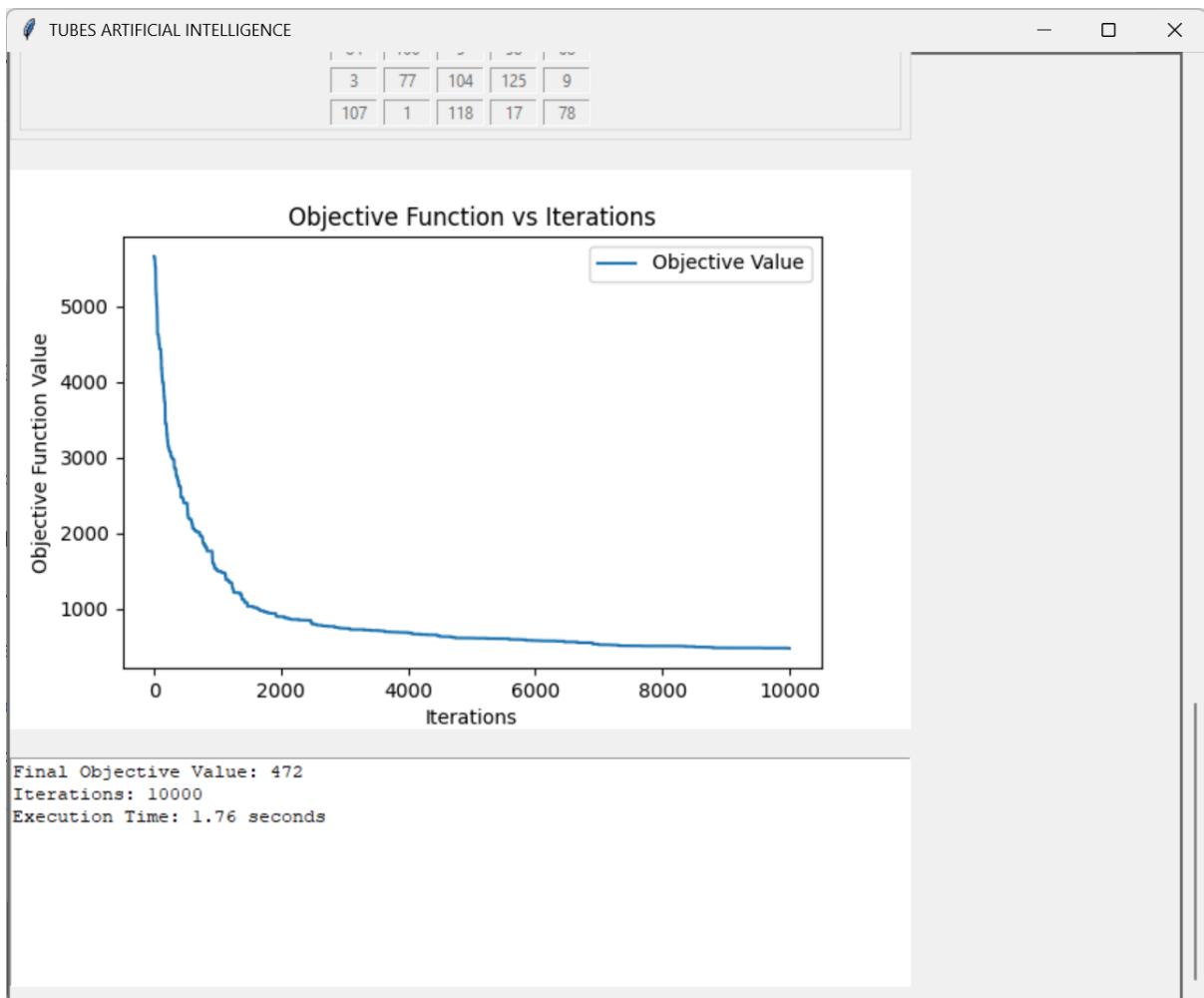
38	110	15	57	96
10	91	69	34	111
82	21	74	122	18
124	50	37	64	39
61	49	121	32	48

Layer 2

60	33	99	109	12
105	25	94	79	11
19	112	67	66	52
75	46	55	16	123
56	85	7	45	120

Layer 3





- Eksperimen 2



Stochastic HC Algorithm

[Start](#)[Back to Main Menu](#)**Initial State**[Show 3D Initial State](#)**Layer 1**

45	115	117	96	107
76	5	57	77	21
38	78	37	11	116
123	113	4	56	112
108	29	110	98	118

Layer 2

121	85	30	32	2
60	122	9	13	68
90	71	69	81	27
87	47	105	89	39
54	43	15	73	51

Layer 3

94	3	49	12	36
106	80	41	52	42
66	62	44	119	18
59	14	70	1	55
23	92	53	17	84

Layer 4

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

8	82	104	61	19
48	102	24	20	50
111	124	22	64	100
26	79	34	46	25
58	31	93	114	97

Layer 5

99	63	95	65	10
33	7	86	103	6
75	16	40	91	72
67	35	74	88	83
125	101	109	28	120

Final State

Show 3D Final State

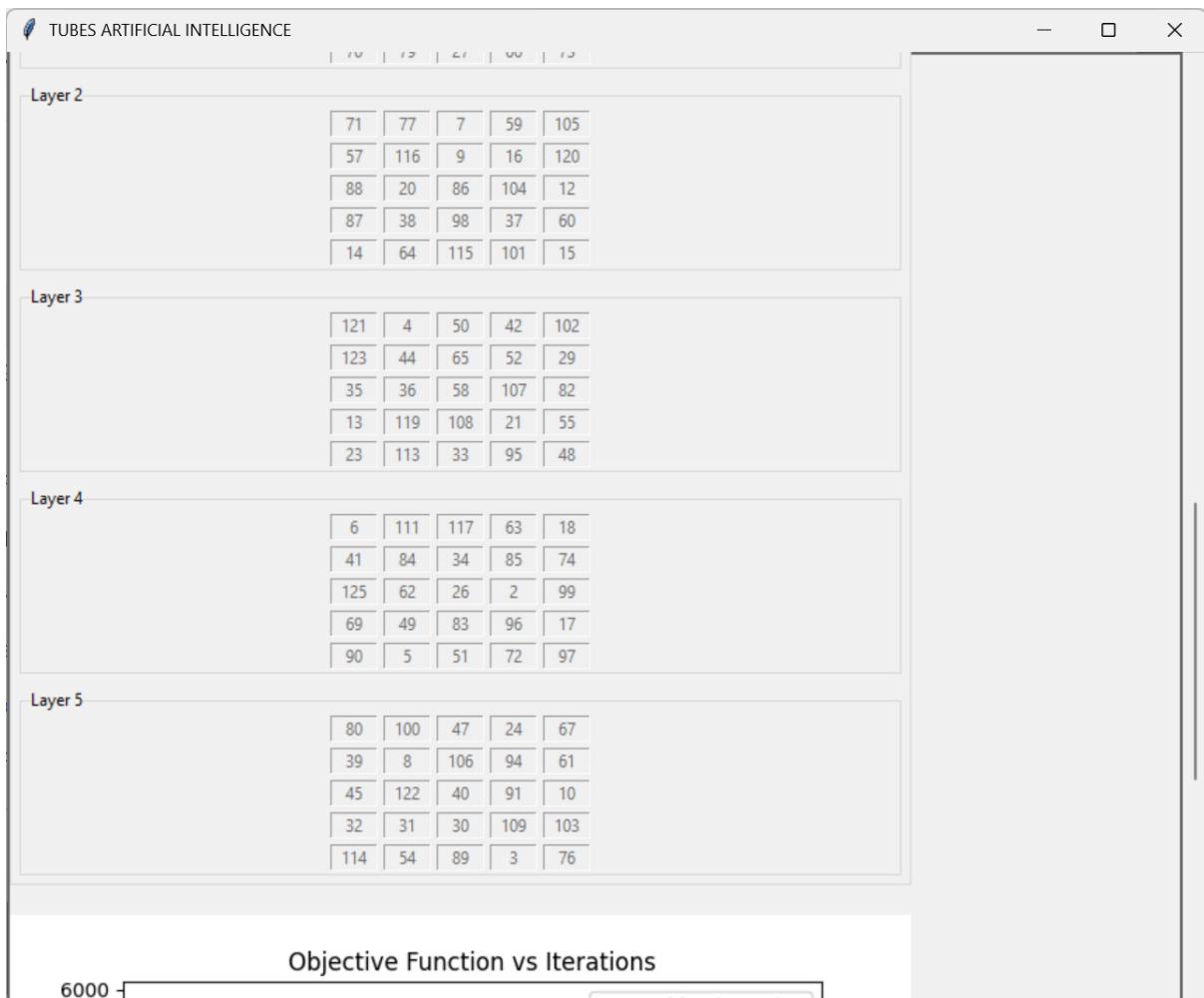
Layer 1

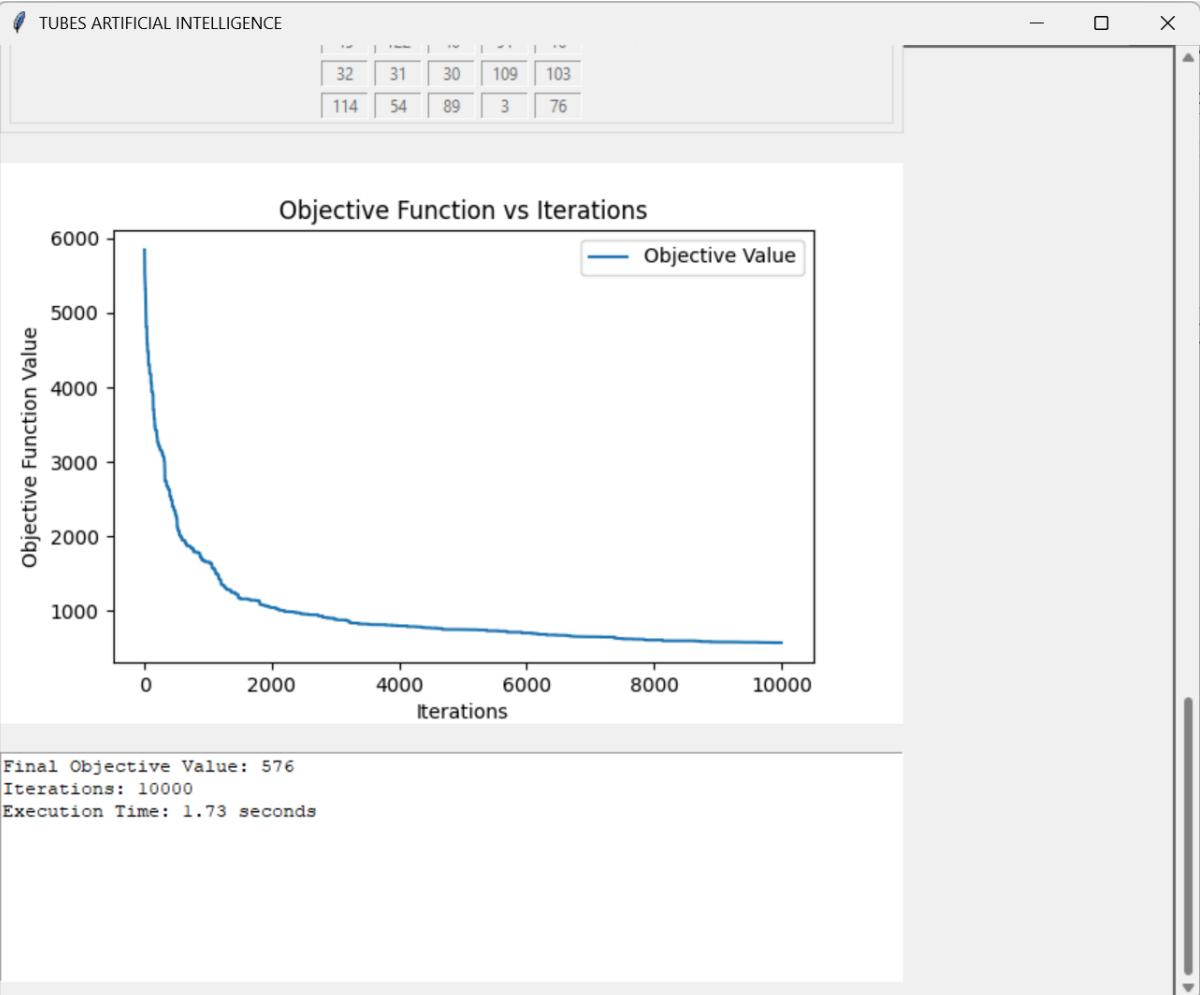
43	25	92	124	19
53	56	110	68	28
22	75	93	11	112
118	78	1	46	81
70	79	27	66	73

Layer 2

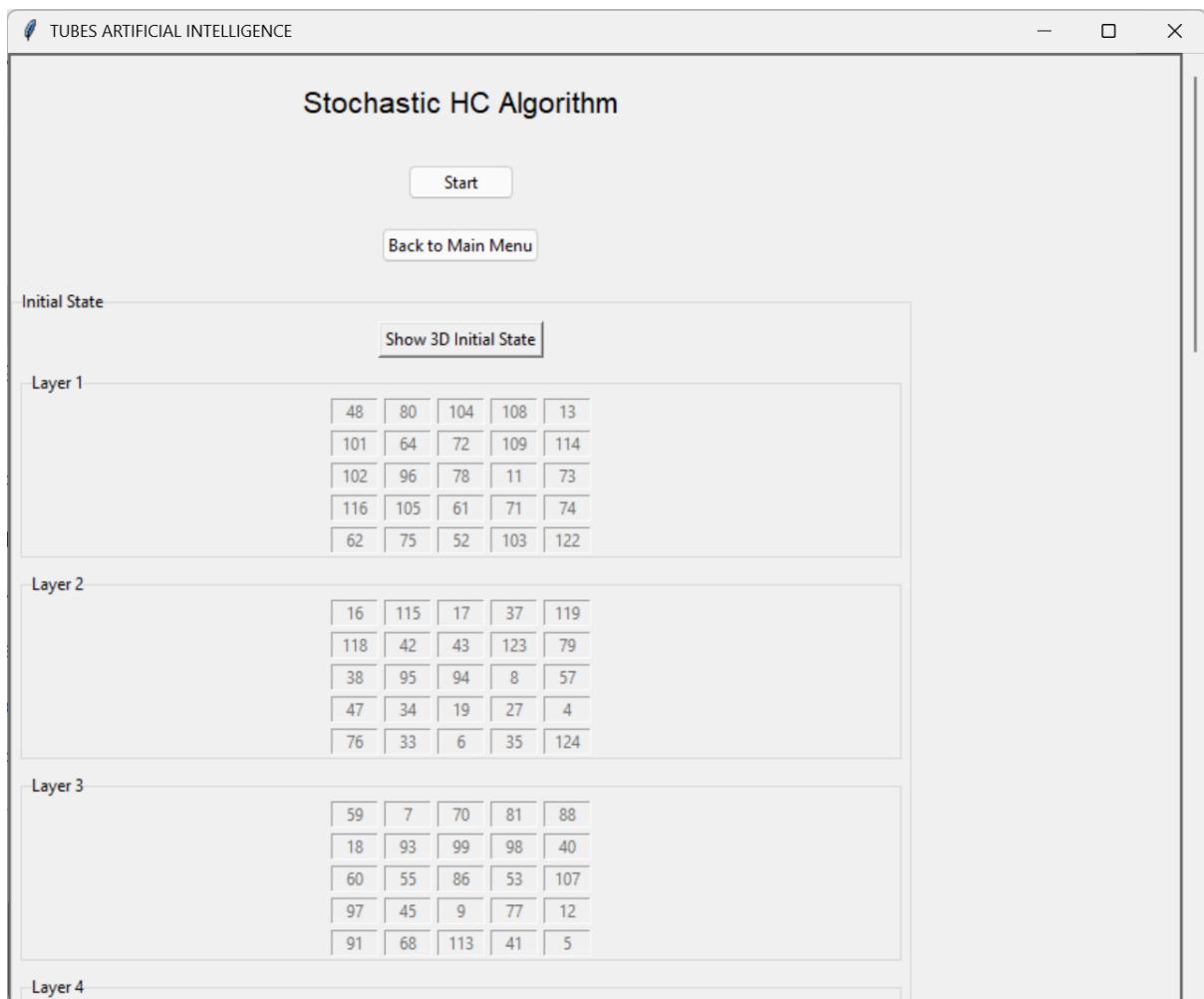
71	77	7	59	105
57	116	9	16	120
88	20	86	104	12
87	38	98	37	60
14	64	115	101	15

Layer 3





- Eksperimen 3



TUBES ARTIFICIAL INTELLIGENCE

91	68	113	41	5
----	----	-----	----	---

Layer 4

31	50	32	117	22
111	58	100	2	125
23	110	36	121	15
46	89	26	69	56
3	66	65	67	82

Layer 5

83	87	21	25	49
112	120	29	24	84
63	1	28	106	85
20	14	51	39	90
54	30	92	10	44

Final State

Show 3D Final State

Layer 1

48	122	116	7	22
61	45	74	109	23
57	53	84	13	108
69	12	17	96	121
81	82	18	88	46

Layer 2

63	70	5	76	110
111	40	10	102	51
66	99	93	27	25
38	95	117	75	1
37	14	94	35	124



81	82	18	88	46
----	----	----	----	----

Layer 2

63	70	5	76	110
111	40	10	102	51
66	99	93	27	25
38	95	117	75	1
37	14	94	35	124

Layer 3

8	30	54	112	104
6	90	98	79	41
52	4	80	60	119
125	100	11	59	19
115	92	72	2	32

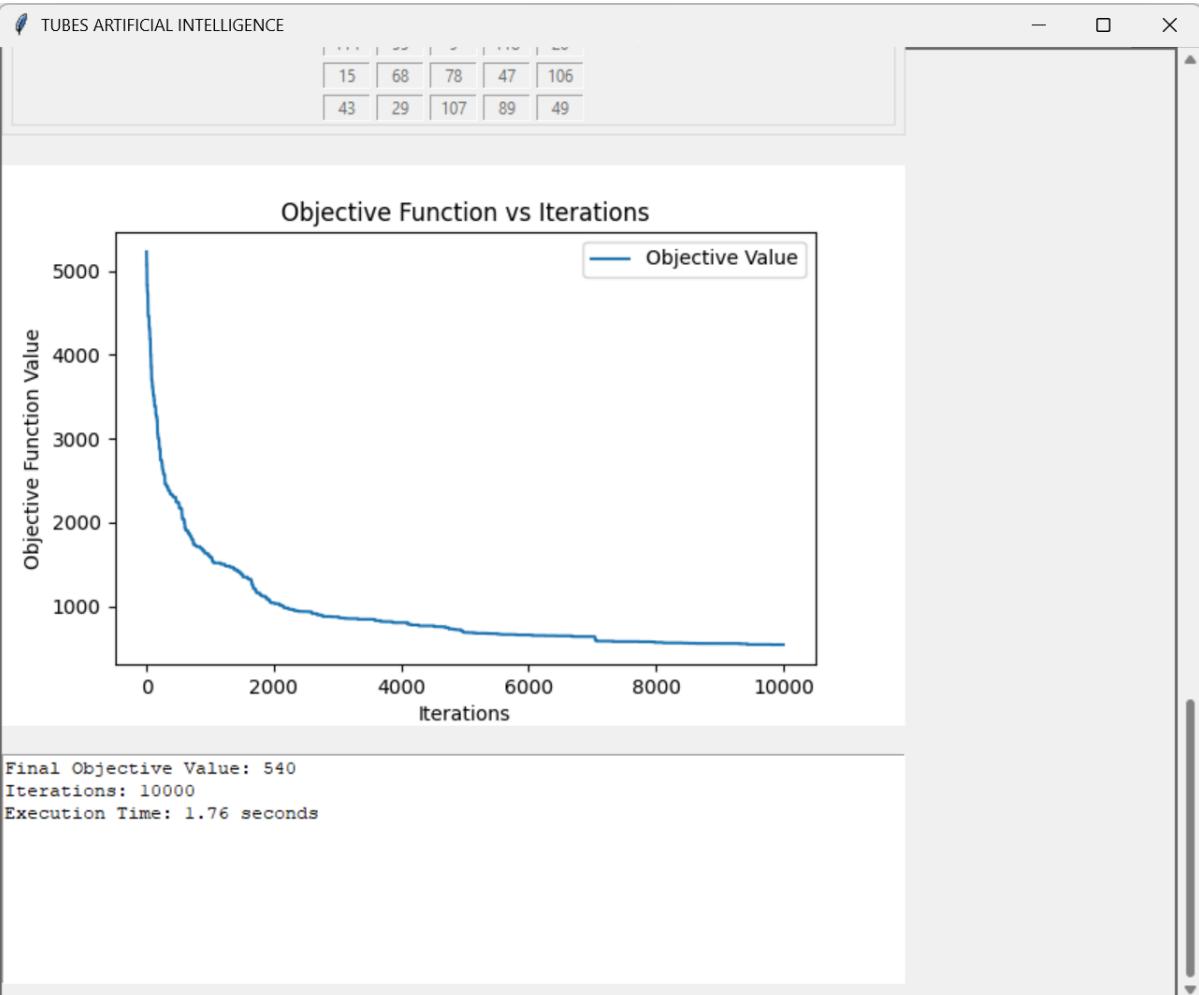
Layer 4

113	50	34	87	31
71	21	123	3	97
26	103	58	86	42
67	62	77	36	73
39	91	24	101	64

Layer 5

83	44	105	33	56
65	120	16	28	85
114	55	9	118	20
15	68	78	47	106
43	29	107	89	49

Objective Function vs Iterations



b. Analisis

Berdasarkan hasil dari eksperimen di atas, algoritma ini mampu mendekati global optima yang ditandai dengan nilai objective function yang perlahan-lahan menuju nol, meskipun tidak mencapai nol dan jumlah iterasi yang jauh lebih besar dibandingkan dengan algoritma lainnya. Besarnya angka iterasi ini terjadi karena pemilihan suksesor dari state saat ini dilakukan secara acak sehingga sering kali terpilih sebuah suksesor yang memiliki nilai objective function lebih buruk dibandingkan nilai objective function saat ini. Proses pencarian solusi berhenti ketika jumlah iterasi telah mencapai batas maksimumnya.

Jika dibandingkan dengan algoritma lainnya, algoritma ini bekerja dengan sangat cepat karena pemilihan suksesor dilakukan secara acak yang mengurangi beban komputasi, berbanding terbalik dengan beberapa algoritma lainnya yang perlu melakukan pencarian suksesor yang lebih baik daripada kondisi saat ini.

Akan tetapi, pemilihan suksesor secara acak ini tidak selalu menghasilkan suksesor yang lebih baik sehingga algoritma ini perlu melakukan iterasi yang sangat besar untuk secara perlahan-lahan mendapatkan sebuah suksesor yang lebih baik dengan cara acak.

Pada pencarian solusi magic cube, tingkat perbaikan nilai objective function terlihat curam pada 2000 iterasi pertama, lama-kelamaan menjadi landai ketika lebih dari 2000 iterasi. Hal ini terjadi karena jumlah suksesor yang lebih baik di atas 2000 iterasi tidak sebanyak di awal 2000 iterasi, sehingga peluang untuk mendapatkan suksesor yang lebih baik juga ikut berkurang karena pemilihan acak ini. Akibatnya, algoritma ini kurang optimal untuk digunakan dalam pencarian solusi dari magic cube ketika batas maksimum iterasinya kecil.

5. Simulated Annealing

a. Eksperimen

- Eksperimen 1

TUBES ARTIFICIAL INTELLIGENCE

Simulated Annealing Algorithm

Initial Temp:

Start

[Back to Main Menu](#)

Initial State

Show 3D Initial State

Layer 1

11	111	88	17	114
83	1	42	43	107
18	28	6	77	22
46	55	60	119	123
105	36	95	74	57

Layer 2

91	50	100	29	37
30	7	33	10	103
79	49	58	92	117
24	81	47	12	59
102	63	16	52	13

Layer 3

72	108	89	21	115
75	101	25	40	54
48	69	4	120	122
87	35	66	2	84
80	32	14	34	27

Layer 4

23	26	8	20	85
98	67	109	31	97
94	121	38	86	62
70	110	125	15	41

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

23	26	8	20	85
98	67	109	31	97
94	121	38	86	62
78	118	125	15	41
64	73	116	3	71

Layer 5

113	112	106	110	104
99	45	19	90	56
68	5	44	51	70
96	53	82	124	76
65	93	9	39	61

Final State

Show 3D Final State

Layer 1

44	99	36	65	73
27	58	115	52	61
123	104	30	37	10
14	53	24	121	108
105	1	124	31	63

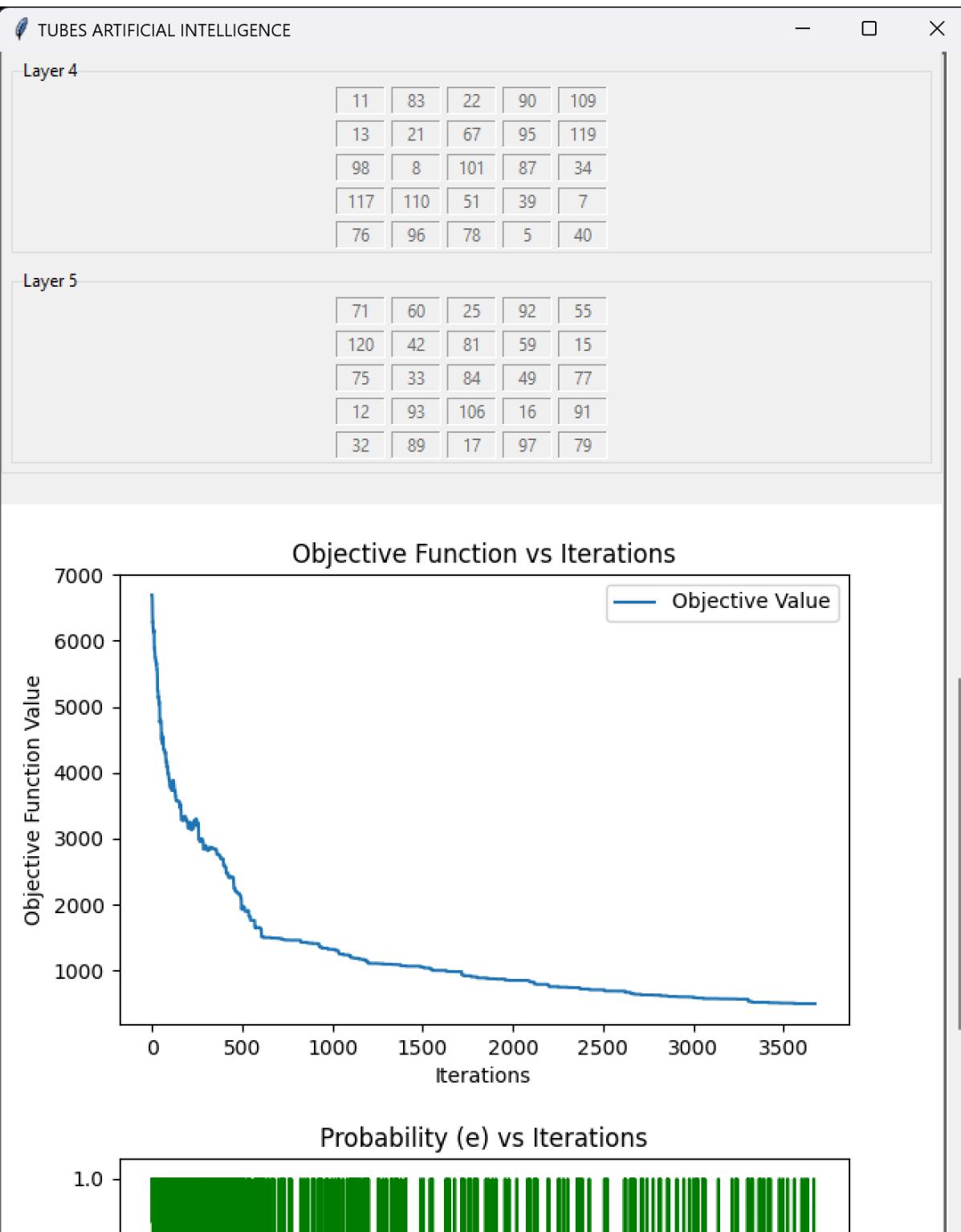
Layer 2

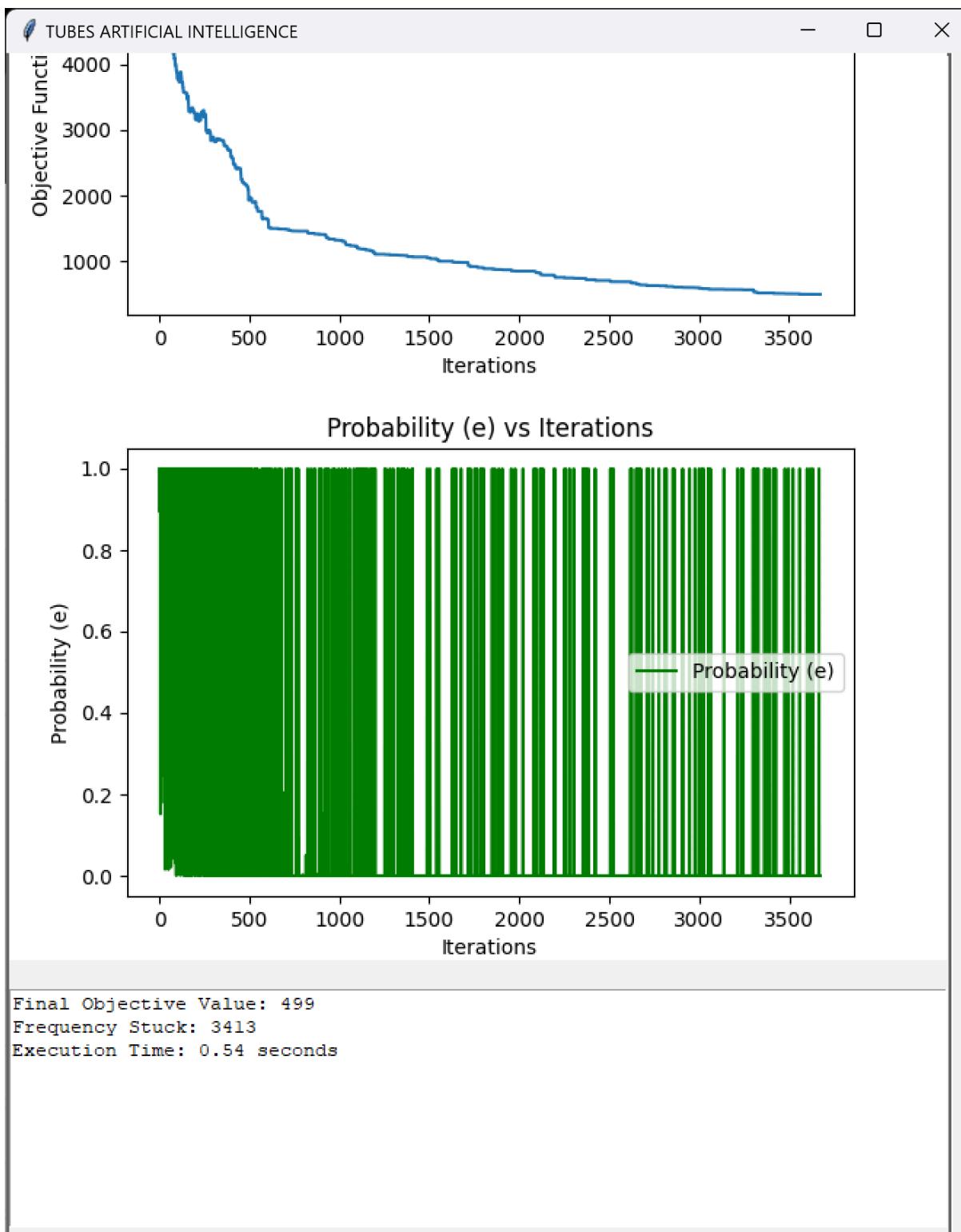
113	28	125	4	48
102	85	6	35	82
2	50	62	88	118
107	47	41	122	3
9	103	70	68	66

Layer 3

86	46	100	57	29
56	111	45	69	38
19	116	43	54	74
72	20	94	18	112
80	23	26	114	64

Layer 4





- Eksperimen 2

TUBES ARTIFICIAL INTELLIGENCE

Simulated Annealing Algorithm

Initial Temp:

Start

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

41	72	74	32	61
66	120	48	77	42
68	52	24	33	109
88	43	59	119	49
63	124	100	36	55

Layer 2

96	103	86	97	92
73	105	81	40	80
16	53	83	29	21
30	6	7	67	85
14	91	70	57	122

Layer 3

58	47	111	94	75
113	123	18	104	45
31	37	64	90	34
28	102	20	107	23
15	25	4	93	3

Layer 4

1	62	84	117	101
2	112	89	65	51
82	106	22	125	19

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

1	62	84	117	101
2	112	89	65	51
82	106	22	125	19
60	26	98	9	17
5	27	110	78	99

Layer 5

10	118	11	39	46
69	108	115	50	87
12	56	35	44	71
114	121	76	54	8
38	116	95	79	13

Final State

Show 3D Final State

Layer 1

41	75	102	35	57
125	28	59	3	98
16	50	122	70	53
69	84	5	88	78
52	73	23	119	39

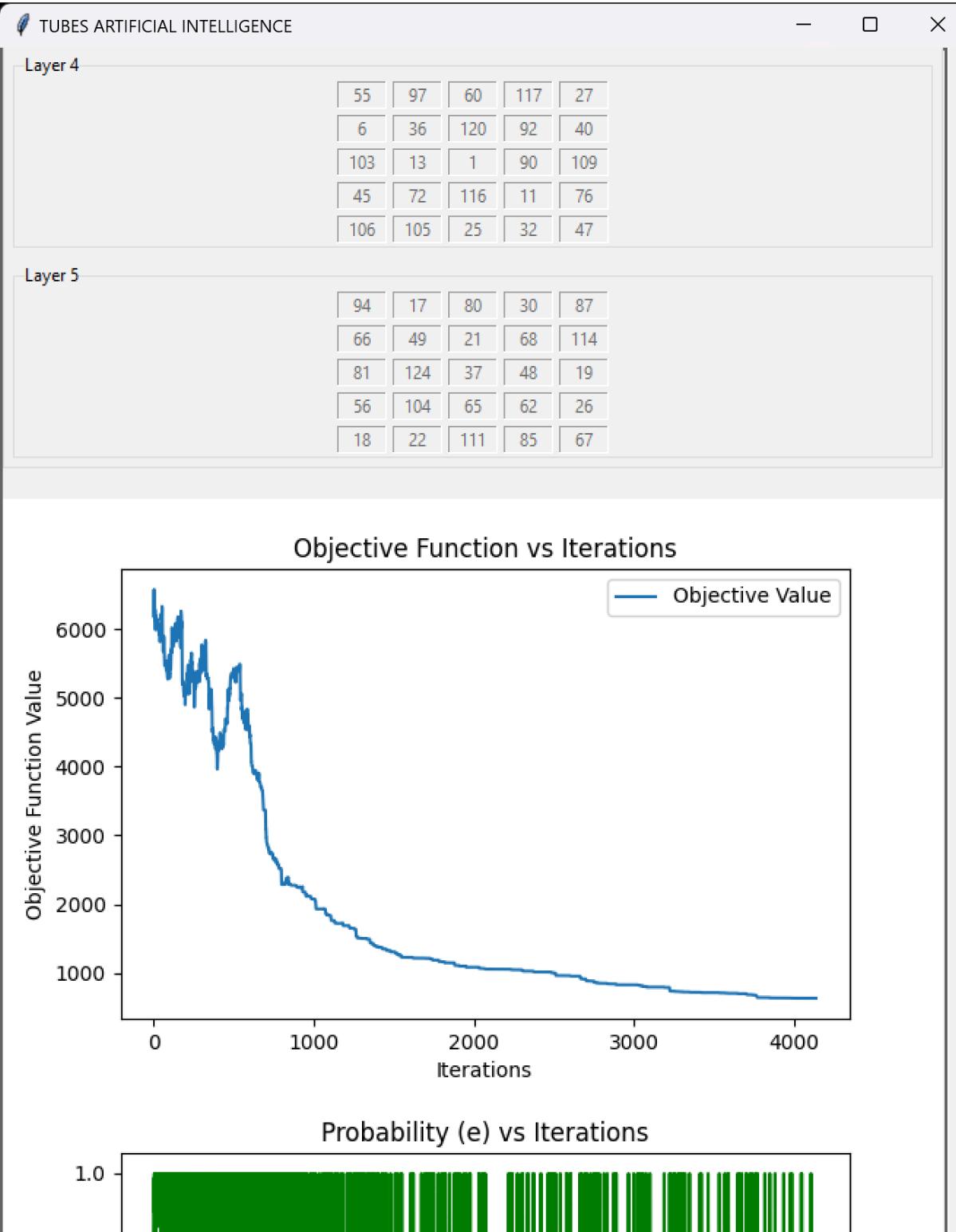
Layer 2

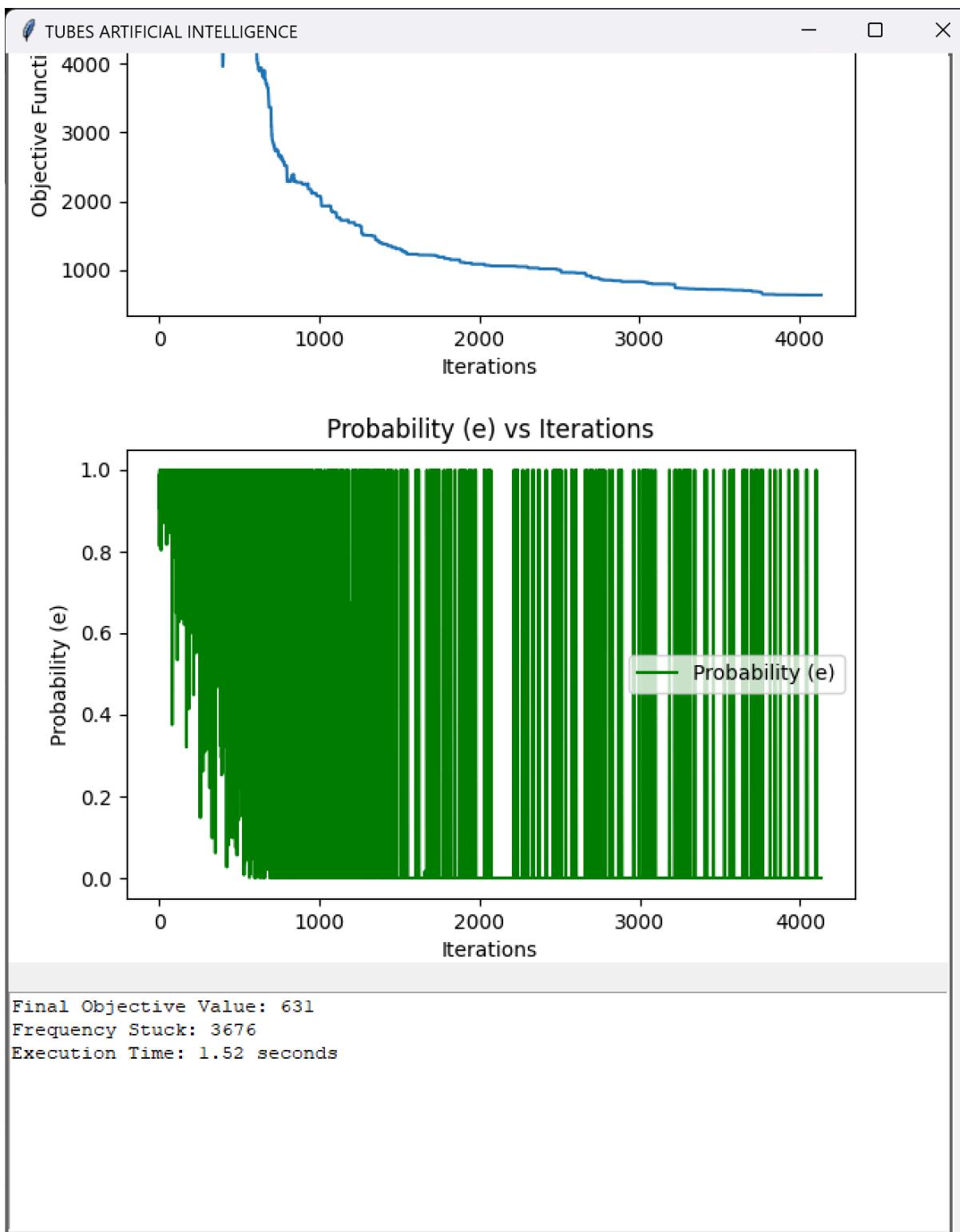
123	29	43	33	91
112	79	4	74	46
15	93	51	99	58
63	12	110	44	77
14	108	95	71	38

Layer 3

7	115	31	96	64
2	121	113	83	9
100	34	89	10	82
86	42	20	101	54
118	8	61	24	107

Layer 4





- Eksperimen 3

TUBES ARTIFICIAL INTELLIGENCE

Simulated Annealing Algorithm

Initial Temp:

Start

Back to Main Menu

Initial State

Show 3D Initial State

Layer 1

76	52	98	37	20
4	50	2	119	109
32	82	121	36	38
28	8	106	19	40
46	91	22	88	124

Layer 2

93	72	74	81	7
89	96	31	17	65
120	67	90	71	12
105	97	11	54	108
23	18	80	77	35

Layer 3

99	83	21	113	51
25	100	104	5	115
26	84	61	49	6
66	101	33	45	15
85	64	39	103	110

Layer 4

87	47	92	69	73
123	60	42	111	125
48	44	79	56	68
21	62	21	107	15

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

87	47	92	69	73
123	60	42	111	125
48	44	79	56	68
34	53	24	107	16
57	59	14	75	10

Layer 5

118	95	9	78	58
1	102	117	13	62
30	27	114	122	63
116	70	112	86	29
43	41	55	94	3

Final State

Show 3D Final State

Layer 1

85	29	125	16	63
48	100	61	91	18
60	56	5	95	102
65	82	118	19	31
66	54	2	97	107

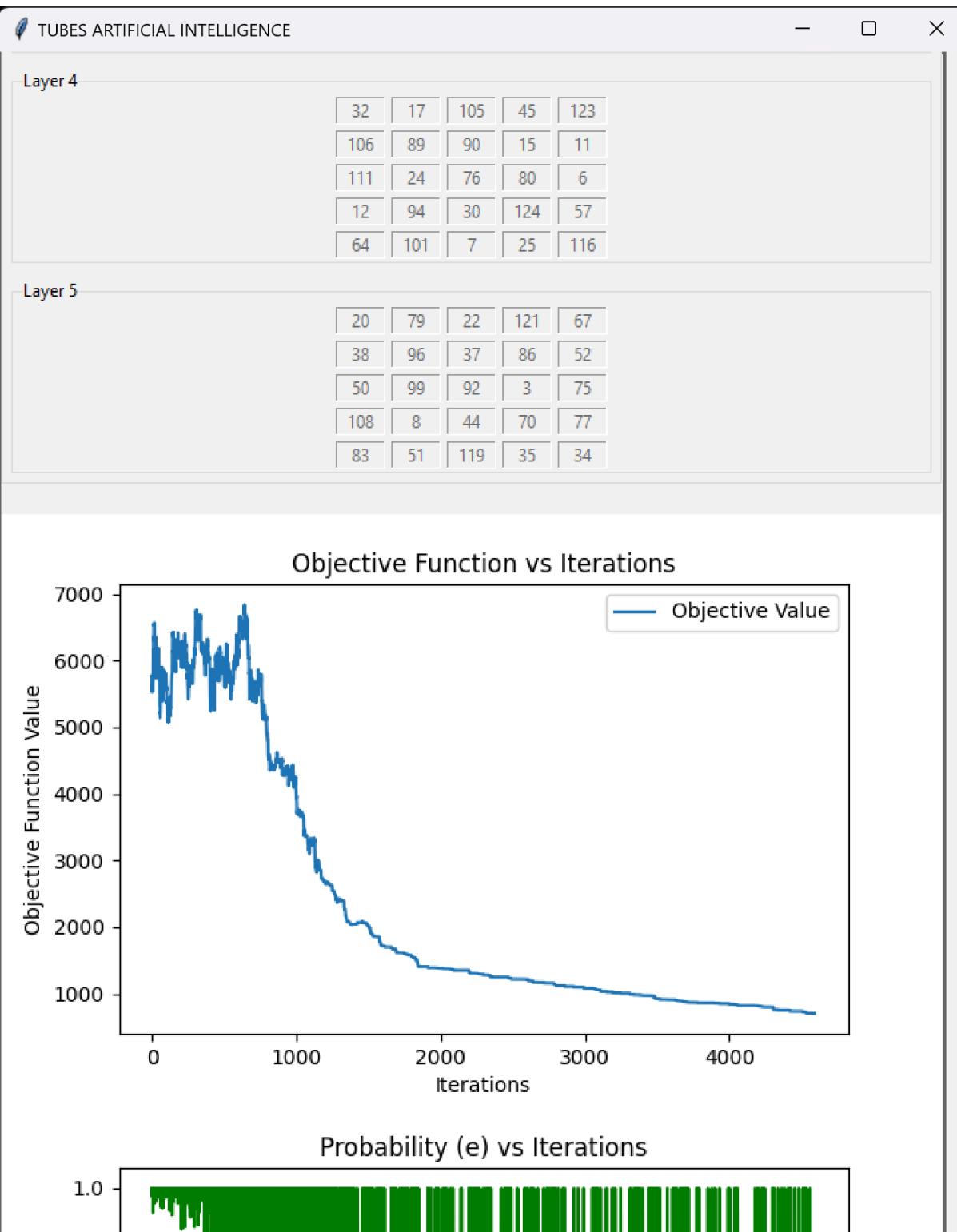
Layer 2

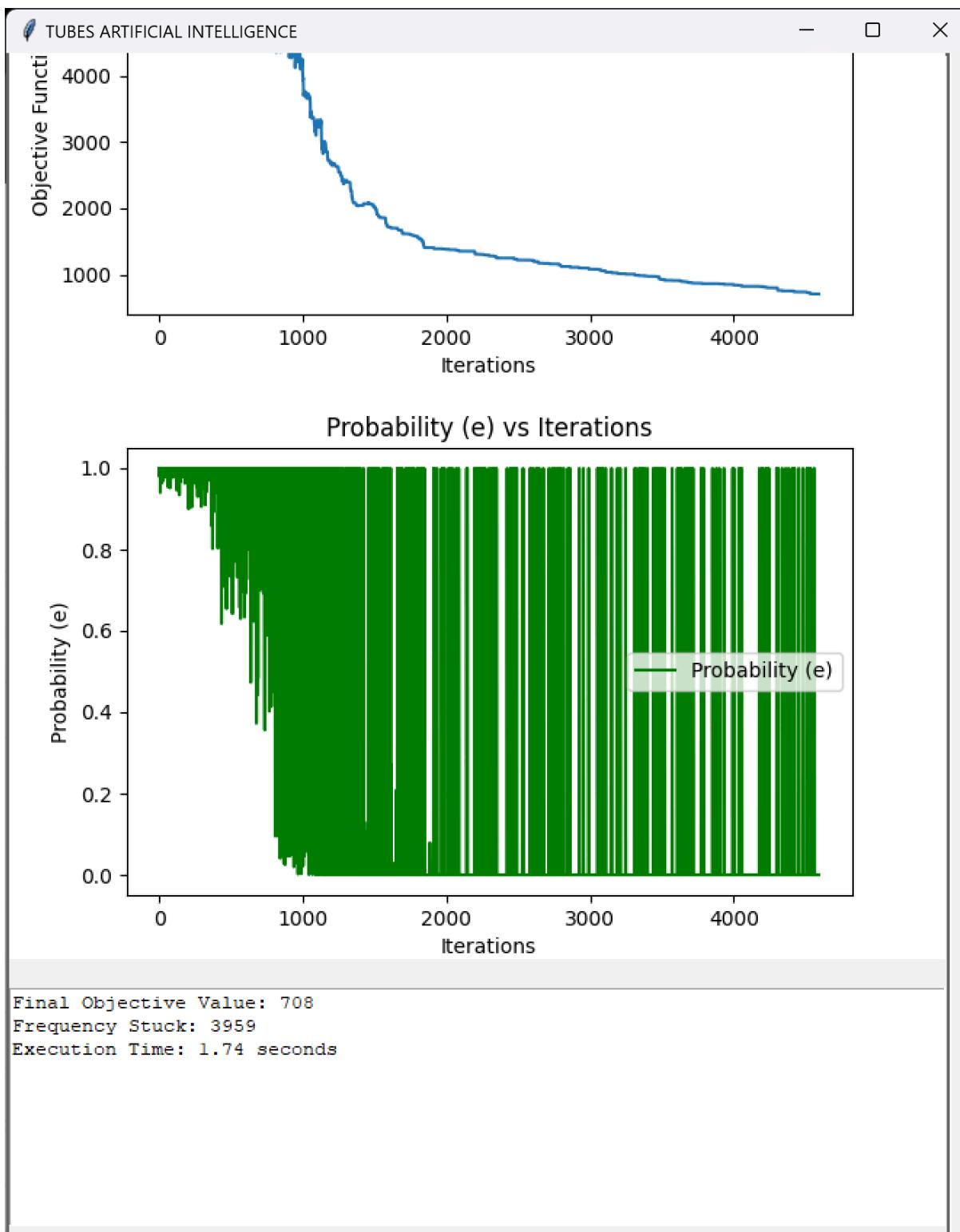
72	62	10	114	42
93	1	73	36	117
46	28	78	115	58
87	103	53	14	59
13	110	104	40	39

Layer 3

112	120	55	9	21
26	47	49	84	113
43	109	71	27	68
41	33	69	74	98
88	4	81	122	23

Layer 4





b. Analisis

Algoritma Simulated Annealing menunjukkan kemampuan yang cukup baik dalam mendekati global optima dalam penyelesaian masalah magic cube. Berdasarkan hasil dari tiga eksperimen dengan initial temperature sebesar 100, 1000, dan 10000, nilai objective function akhir berkisar antara 400-an hingga 700-an. Meskipun hasil ini tidak mencapai global optima (dengan nilai objective function 0), algoritma ini cukup berhasil dalam mendekati solusi optimal. Hal ini disebabkan oleh mekanisme algoritma yang memungkinkan penerimaan solusi yang lebih buruk dalam jumlah tertentu, yang membantu algoritma keluar dari jebakan local optima dan melanjutkan eksplorasi untuk perbaikan lebih lanjut. Seiring penurunan suhu, probabilitas untuk menerima solusi yang lebih buruk juga menurun, sehingga algoritma cenderung mengeksplorasi solusi saat suhu rendah dan perbaikan yang terjadi lebih lambat.

Dibandingkan dengan algoritma local search lainnya, Simulated Annealing memiliki keunggulan dalam eksplorasi ruang solusi. Sebagai contoh, Steepest Ascent Hill HC yang hanya memilih langkah terbaik di setiap iterasi, sehingga lebih rentan terjebak di local optima karena hanya menerima solusi yang lebih baik dari kondisi saat ini. Sebaliknya, Simulated Annealing menggunakan pendekatan probabilistik yang memungkinkan penerimaan solusi yang lebih buruk secara sementara untuk meningkatkan eksplorasi ruang solusi dan menghindari jebakan local optima. Meskipun hasil akhirnya sering serupa dengan algoritma lain, solusi tersebut masih dapat dioptimalkan lebih lanjut. Ini dimungkinkan dengan mengubah nilai initial temperature untuk mempengaruhi tingkat eksplorasi awal. Selain itu penyesuaian nilai konstanta dalam fungsi penurunan temperatur juga masih bisa dicoba untuk dilakukan, misalnya dengan cara memperkecil nilainya untuk memperlambat proses pendinginan. Hal ini memberi algoritma lebih banyak waktu untuk mengeksplorasi solusi yang lebih luas, memungkinkan keseimbangan yang mungkin lebih baik antara

eksplorasi dan eksplorasi serta mungkin menghasilkan solusi yang lebih optimal.

Dari segi durasi proses pencarian, Simulated Annealing relatif cepat jika dilihat pada ketiga eksperimen. Hal ini menunjukkan bahwa algoritma ini dapat bersaing dalam hal efisiensi waktu dibandingkan dengan algoritma lain, terutama Steepest Ascent yang cenderung memerlukan lebih banyak waktu karena harus mengevaluasi semua kemungkinan successor di setiap iterasi. Durasi pencarian yang cepat pada Simulated Annealing disebabkan oleh pendekatannya yang hanya mengevaluasi satu successor per iterasi dan memilih successor secara acak, sehingga tidak perlu mengevaluasi seluruh ruang lingkup successor di setiap langkah.

Hasil eksperimen dengan initial temperature 100, 1000, dan 10000 menunjukkan bahwa algoritma Simulated Annealing mampu mencapai hasil akhir yang konsisten, dengan nilai objective akhir berkisar antara 400 hingga 700. Hal ini menunjukkan bahwa perubahan pada nilai initial temperature tidak memiliki pengaruh signifikan terhadap hasil akhir algoritma ini. Meskipun initial temperature mempengaruhi tingkat eksplorasi awal, hasil eksperimen ini menunjukkan bahwa setelah beberapa iterasi, algoritma cenderung menemukan solusi dalam rentang yang sama. Dengan demikian, dapat disimpulkan bahwa parameter initial temperature lebih berperan dalam menentukan kecepatan kapan fase eksplorasi selesai, tetapi pengaruhnya terhadap kualitas akhir solusi tidak terlalu besar, karena proses penurunan temperatur (fungsi penurunan temperatur termasuk cara dan konstanta yang digunakan di dalamnya) dan penerimaan probabilistik tetap menjadi faktor dominan dalam mengarahkan algoritma menuju konvergensi.

Grafik objective function vs iterations dari setiap eksperimen menunjukkan tren penurunan yang jelas, dimana perbaikan signifikan terjadi di awal iterasi dan kemudian melambat seiring waktu. Pada

awal iterasi nilai objective function sering naik turun karena suhu (temperatur) masih tinggi, yang memungkinkan algoritma menerima solusi yang lebih buruk dengan probabilitas cukup besar. Hal ini bertujuan untuk mengeksplorasi ruang solusi secara luas dan membantu algoritma menghindari jebakan local optima. Dengan menerima langkah yang kurang optimal dalam jangka pendek, algoritma dapat menemukan jalur menuju solusi yang lebih baik di iterasi berikutnya. Seiring berjalannya waktu, suhu menurun, dan probabilitas untuk menerima solusi yang lebih buruk juga berkurang, sehingga algoritma mulai fokus pada eksplorasi solusi terbaik yang ditemukan. Ini menjelaskan fluktuasi besar di awal iterasi dan perbaikan yang lebih stabil di tahap akhir, di mana nilai objective function cenderung menurun menuju solusi yang lebih optimal.

Berdasarkan grafik probability (e) vs iterations, juga terlihat bahwa pada awal iterasi, peluang untuk menerima solusi yang lebih buruk masih cukup tinggi. Hal ini disebabkan oleh suhu yang masih tinggi, memungkinkan algoritma Simulated Annealing untuk melakukan eksplorasi ruang solusi secara luas. Namun, seiring berjalannya iterasi dan suhu yang semakin menurun, peluang tersebut mulai berkurang secara signifikan. Di akhir iterasi, probabilitas untuk menerima solusi buruk semakin mendekati nol, menunjukkan peralihan algoritma dari fase eksplorasi ke fase eksplotatif, di mana algoritma hanya fokus pada menerima solusi yang lebih baik, sehingga perbaikan solusi menjadi lebih stabil dan bertahap menuju optimal.

Secara keseluruhan, Simulated Annealing mampu menghasilkan hasil yang stabil dan cukup dekat dengan solusi optimal dalam beberapa eksperimen, menjadikannya algoritma yang andal dalam penyelesaian masalah magic cube. Durasi eksekusi yang efisien dan konsistensi hasil juga menunjukkan bahwa algoritma ini dapat diandalkan untuk berbagai kondisi awal. Namun, hasil akhir dari eksperimen seringkali stagnan di nilai objective function antara 400-an hingga 700-an. Oleh karena itu, mungkin diperlukan pendekatan tambahan, seperti

penerapan adaptive temperature scheduling, perbaikan fungsi penurunan temperatur (perubahan konstanta atau rumus seluruhnya), restart acak, perbaikan fitness function, atau kombinasi dengan metode lain, untuk meningkatkan kemampuan algoritma dalam mencapai solusi yang lebih optimal dan menghindari terjebak pada kisaran nilai tersebut.

6. Genetic Algorithm

a. Eksperimen

- i. Eksperimen populasi sebagai control (populasi 5, 10, 15, dengan iterasi 5000)

1. Eksperimen 1

TUBES ARTIFICIAL INTELLIGENCE

Genetic Algorithm

Population Size:

Max Iterations:

Initial State

Layer 1

100	78	121	85	14
101	49	92	81	17
50	23	110	117	106
53	55	18	34	1
52	105	67	115	8

Layer 2

90	123	2	122	70
64	60	58	86	36
24	41	7	124	68
109	19	98	45	94
118	40	88	65	84

Layer 3

56	15	26	20	11
39	38	28	48	104
111	59	12	102	96
71	63	114	89	35
66	73	44	79	6

Layer 4

32	76	87	46	5
----	----	----	----	---

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

32	76	87	46	5
33	42	25	125	47
77	74	91	9	108
16	51	95	62	116
54	119	3	43	83

Layer 5

27	61	57	10	4
37	103	82	97	99
21	30	29	120	75
69	107	31	72	113
22	13	93	80	112

Final State

Show 3D Final State

Layer 1

37	75	15	79	30
41	11	96	110	42
43	50	36	95	111
4	66	24	10	123
118	53	81	14	97

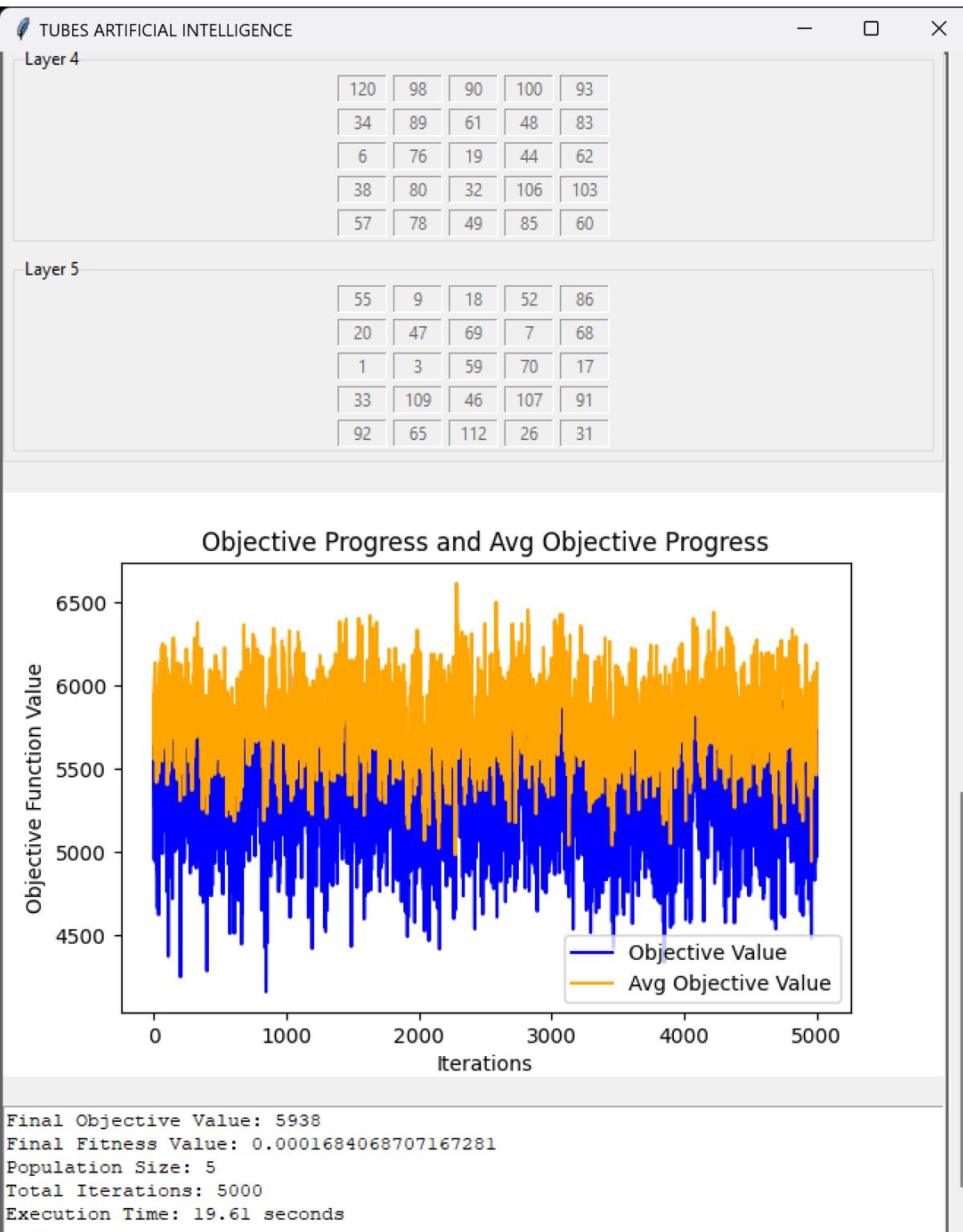
Layer 2

77	56	99	74	51
63	105	104	124	84
64	108	88	16	117
71	94	87	40	113
27	2	72	73	35

Layer 3

116	28	25	54	101
119	13	58	45	12
39	114	22	8	29
115	125	21	82	23
67	102	121	122	5

Layer 4



2. Eksperimen 2

TUBES ARTIFICIAL INTELLIGENCE

Genetic Algorithm

Population Size:

Max Iterations:

Start

Back to Main Menu

Initial State

Show 3D Initial State

Layer 1

59	56	69	17	74
12	112	83	9	51
79	94	120	37	96
10	34	3	107	77
18	52	81	38	76

Layer 2

43	40	124	73	86
15	48	8	106	99
2	46	30	27	44
82	84	100	7	55
92	47	101	88	121

Layer 3

80	31	11	21	32
26	91	85	113	119
71	24	29	116	19
115	67	1	110	57
72	105	108	5	39

Layer 4

13	68	4	78	65
----	----	---	----	----



Layer 4

13	68	4	78	65
103	35	64	49	97
117	104	111	118	90
70	54	114	93	22
25	66	63	102	75

Layer 5

89	87	122	109	58
14	62	6	98	16
28	45	123	23	53
36	42	60	95	125
33	61	41	20	50

Final State

[Show 3D Final State](#)

Layer 1

85	67	20	68	53
106	5	100	75	86
114	47	84	18	97
1	117	16	124	63
45	65	122	93	2

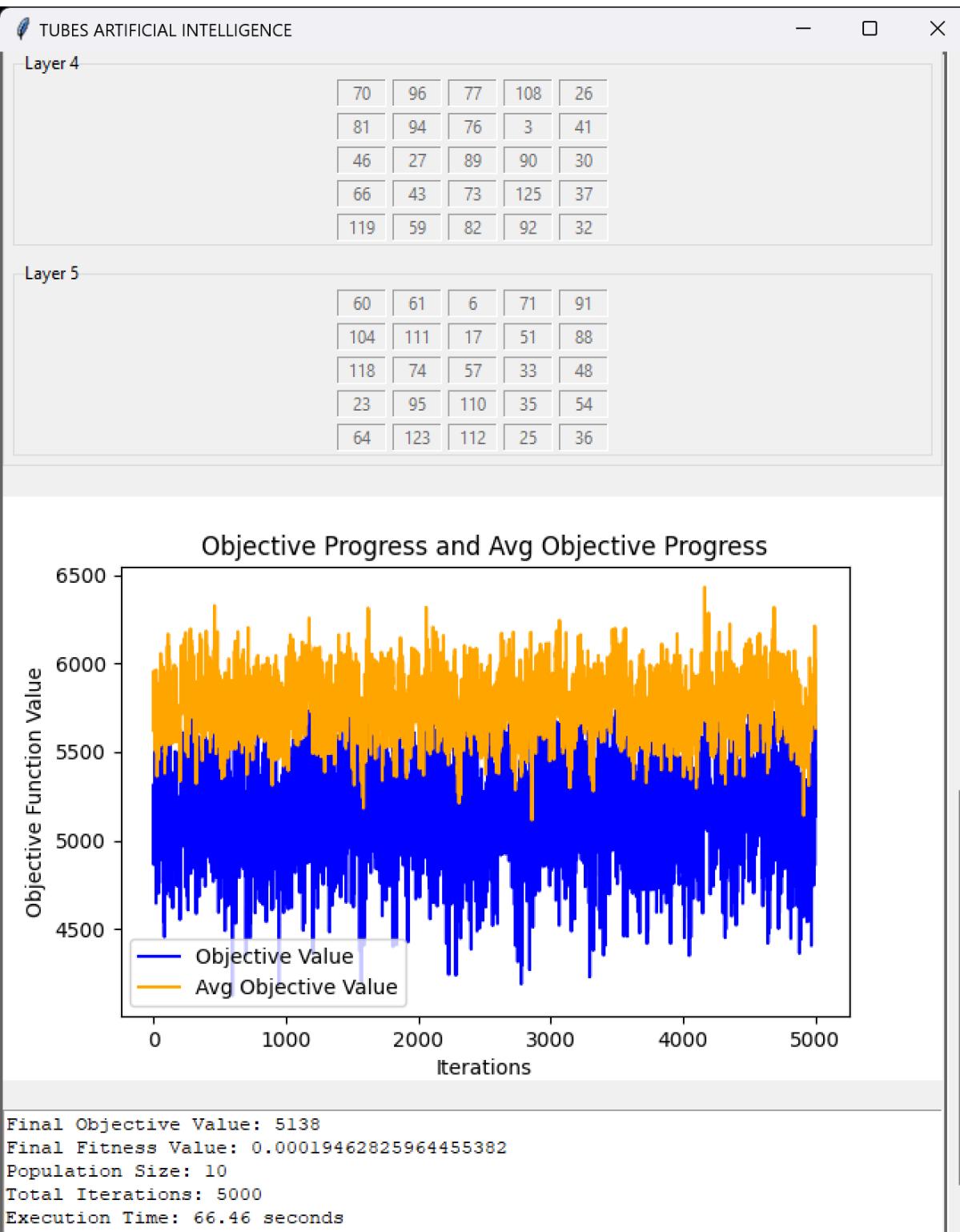
Layer 2

24	34	39	28	87
121	29	58	44	55
103	109	98	107	80
9	38	113	105	42
52	116	11	83	78

Layer 3

49	12	115	4	7
50	120	15	10	101
102	22	13	72	62
40	99	31	8	14
69	19	21	56	79

Layer 4



3. Eksperimen 3

TUBES ARTIFICIAL INTELLIGENCE

Genetic Algorithm

Population Size:

Max Iterations:

Start

Back to Main Menu

Initial State

Show 3D Initial State

Layer 1

14	12	76	55	18
102	94	72	23	56
15	52	6	4	111
44	124	53	2	37
68	75	62	61	110

Layer 2

67	91	1	19	81
40	93	28	20	49
80	36	85	115	3
86	58	82	121	108
66	5	46	22	122

Layer 3

71	120	74	113	34
27	11	98	89	47
63	39	31	60	96
33	25	13	109	97
90	118	16	101	24

Layer 4

123	105	42	30	8
-----	-----	----	----	---

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

123	105	42	30	8
38	107	79	57	77
54	92	117	59	104
65	100	50	51	125
70	78	114	84	29

Layer 5

17	73	95	9	35
116	88	48	43	103
119	106	41	32	64
69	7	87	45	83
21	10	99	26	112

Final State

Show 3D Final State

Layer 1

54	117	88	41	26
101	44	116	63	5
100	59	6	120	28
113	29	103	99	124
97	16	65	49	42

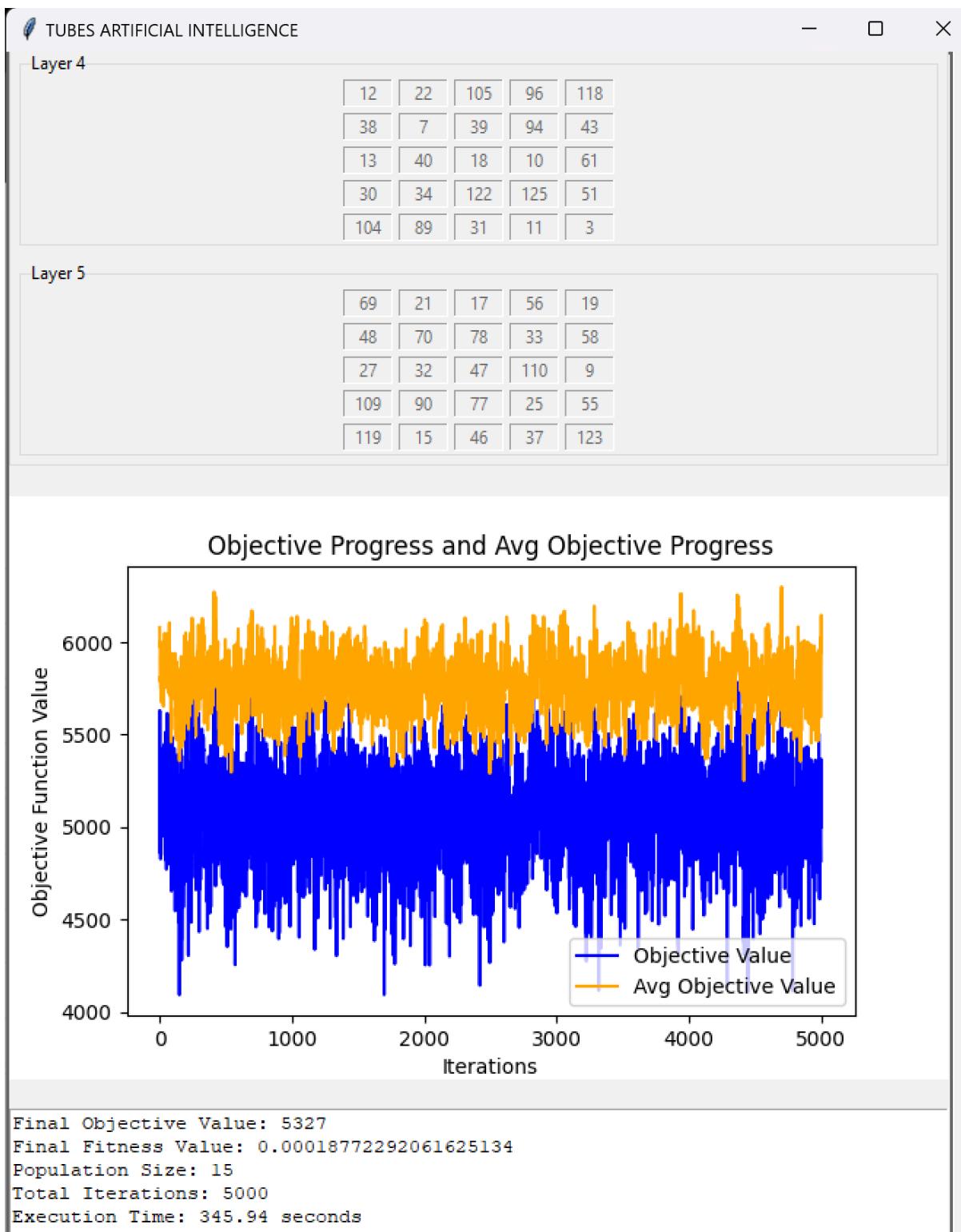
Layer 2

62	76	111	121	35
71	20	102	14	53
82	50	45	106	74
57	72	75	83	23
1	107	24	81	91

Layer 3

68	73	98	2	86
95	85	64	115	108
114	8	52	80	93
66	79	36	60	92
84	67	112	87	4

Layer 4



- ii. Eksperimen max iterasi sebagai control (iterasi 4000 6000, 8000, dengan populasi 10)

1. Eksperimen 1

TUBES ARTIFICIAL INTELLIGENCE

Genetic Algorithm

Population Size:

Max Iterations:

[Start](#)

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

9	108	121	43	95
116	67	58	89	84
40	68	80	99	93
77	63	75	22	76
103	20	41	79	81

Layer 2

119	34	33	48	4
56	91	2	45	55
92	107	30	39	59
120	102	1	57	90
61	113	122	65	111

Layer 3

96	10	42	50	6
123	117	72	11	101
52	46	88	70	85
15	25	69	66	100
27	60	35	26	21

Layer 4

14	115	124	71	62
----	-----	-----	----	----

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

14	115	124	71	62
97	98	23	38	17
105	29	83	49	3
74	24	110	5	82
7	31	8	64	73

Layer 5

87	86	37	112	28
18	44	106	114	12
78	104	47	51	19
16	125	53	54	13
118	36	109	32	94

Final State

Show 3D Final State

Layer 1

89	24	11	95	65
80	116	121	83	86
57	26	49	44	101
61	34	52	105	3
53	39	124	10	76

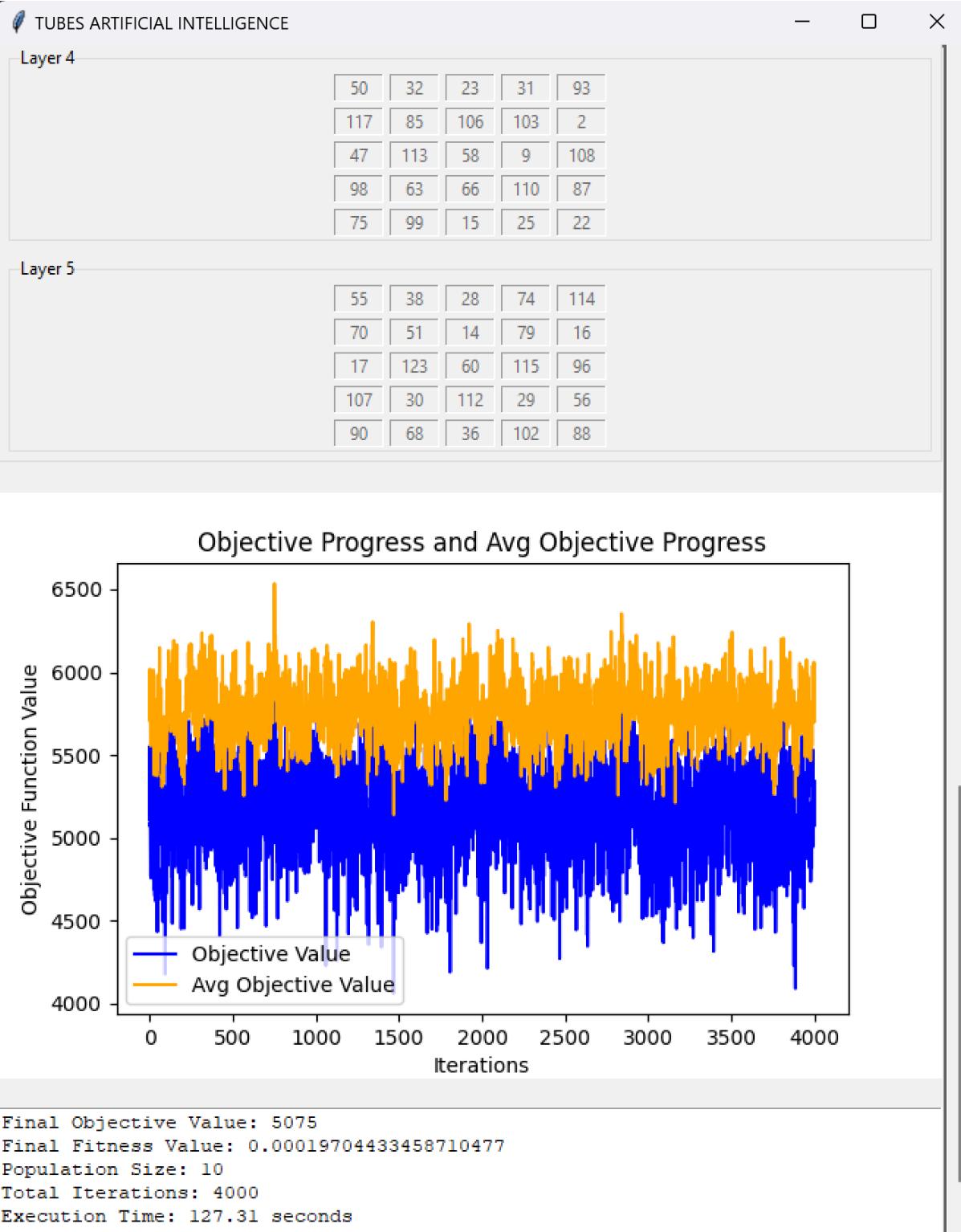
Layer 2

100	33	91	94	46
48	97	43	18	118
71	19	81	5	7
6	120	1	41	62
21	77	4	92	122

Layer 3

119	67	27	54	111
69	78	45	40	8
125	35	104	84	37
20	64	73	72	82
12	109	59	42	13

Layer 4



2. Eksperimen 2

TUBES ARTIFICIAL INTELLIGENCE

Genetic Algorithm

Population Size:

Max Iterations:

Start

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

118	100	115	106	30
104	35	52	69	76
83	7	43	5	92
48	99	75	9	37
120	82	57	90	12

Layer 2

19	26	20	107	13
2	44	95	4	22
51	81	66	53	65
89	88	103	96	84
114	10	28	39	70

Layer 3

79	49	15	59	17
33	6	91	113	18
50	121	38	8	60
11	94	29	123	1
71	101	54	56	124

Layer 4

77	87	108	55	98
----	----	-----	----	----

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

77	87	108	55	98
14	24	46	16	64
31	112	40	102	45
25	109	23	47	93
61	42	119	74	36

Layer 5

116	85	68	73	80
63	78	122	41	32
125	67	21	110	58
111	62	105	27	34
117	3	97	86	72

Final State

Show 3D Final State

Layer 1

56	41	83	48	43
42	89	96	34	87
92	54	58	66	30
93	82	72	3	65
10	8	32	84	111

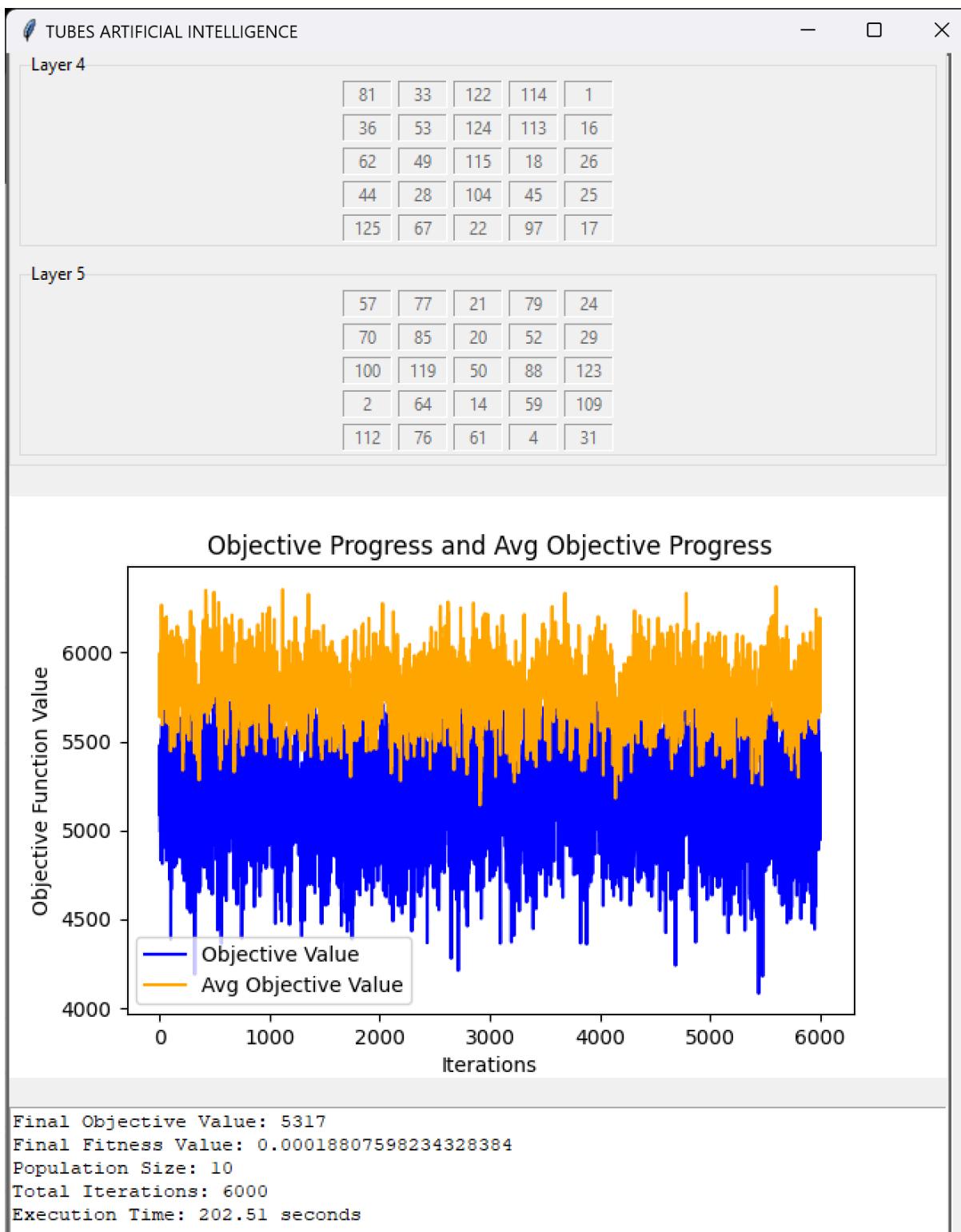
Layer 2

19	12	110	40	120
75	106	68	90	86
5	73	80	99	102
37	91	9	39	38
71	13	121	103	60

Layer 3

6	94	23	101	35
11	95	78	7	47
107	51	69	55	117
108	98	63	118	116
46	74	105	27	15

Layer 4



3. Eksperimen 3

TUBES ARTIFICIAL INTELLIGENCE

Genetic Algorithm

Population Size:

Max Iterations:

Start

[Back to Main Menu](#)

Initial State

[Show 3D Initial State](#)

Layer 1

7	113	103	50	99
79	124	10	59	6
106	24	86	41	32
21	44	116	68	46
122	30	26	81	89

Layer 2

91	98	57	76	34
42	38	73	92	47
5	101	87	39	1
66	27	29	36	3
75	11	84	67	120

Layer 3

100	2	107	22	93
43	111	4	78	112
83	95	65	19	109
31	110	119	20	23
70	114	80	60	88

Layer 4

85	51	55	40	74
----	----	----	----	----

TUBES ARTIFICIAL INTELLIGENCE

Layer 4

85	51	55	40	74
64	118	77	13	45
53	97	15	62	28
102	56	105	71	117
16	61	25	54	94

Layer 5

48	17	121	9	69
108	82	72	37	115
125	63	33	104	90
14	123	58	52	12
96	35	49	8	18

Final State

Show 3D Final State

Layer 1

97	110	7	11	52
21	1	88	22	46
13	30	96	84	108
45	114	99	28	50
85	78	75	111	12

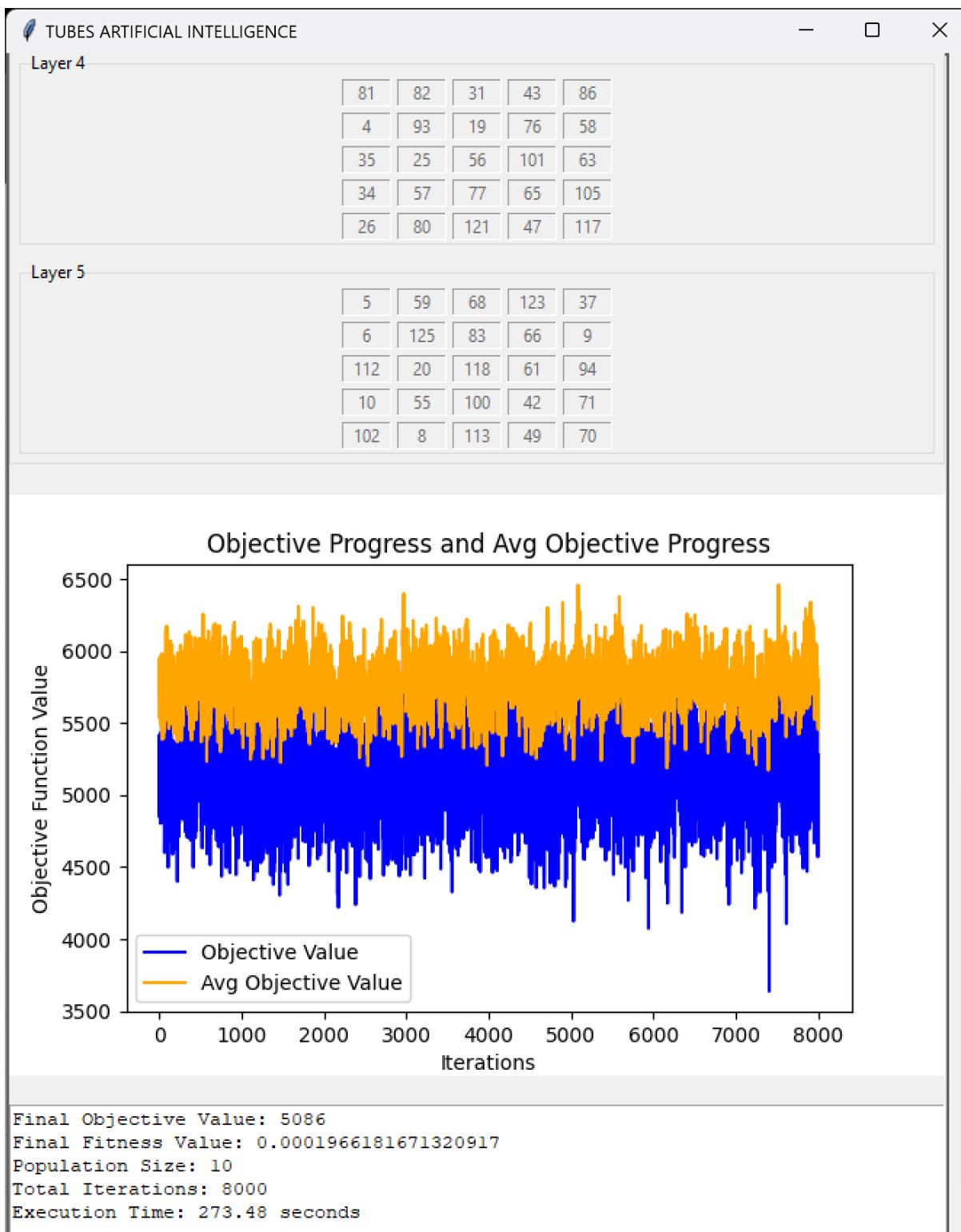
Layer 2

69	3	33	53	36
106	54	44	48	32
16	51	73	98	62
60	14	120	124	79
92	90	95	2	104

Layer 3

74	18	89	67	107
29	40	27	64	41
103	17	72	115	91
109	87	116	39	23
119	122	24	38	15

Layer 4



b. Analisis

Berdasarkan hasil eksperimen, banyaknya iterasi dan jumlah populasi dalam algoritma Genetic Algorithm (GA) ternyata tidak memberikan pengaruh signifikan terhadap hasil akhir pencarian. Meskipun secara teori, peningkatan jumlah iterasi seharusnya memungkinkan algoritma untuk melakukan eksplorasi yang lebih luas dan menemukan solusi yang lebih baik, hasil eksperimen menunjukkan bahwa nilai *objective function* tetap berada dalam rentang yang sama tanpa perbaikan signifikan. Hal ini juga berlaku untuk variasi jumlah populasi, di mana peningkatan ukuran populasi tidak secara konsisten menghasilkan solusi yang lebih baik.

Eksperimen menunjukkan bahwa meskipun populasi yang lebih besar dan iterasi yang lebih panjang seharusnya meningkatkan keragaman dan eksplorasi, hasil akhirnya tetap terjebak pada nilai *objective function* yang tidak jauh berbeda. Grafik menunjukkan bahwa perbaikan nilai *objective* di setiap populasi masih sangat fluktuatif dan tidak memiliki tren yang jelas menuju perbaikan. Hal ini menunjukkan bahwa parameter iterasi dan populasi, dalam skenario ini, tidak cukup untuk mengatasi keterbatasan eksplorasi dan eksplorasi pada algoritma GA yang digunakan. Modifikasi algoritma, seperti penambahan mekanisme untuk mempertahankan individu terbaik atau penggunaan teknik seleksi yang lebih canggih, mungkin diperlukan untuk mencapai hasil yang lebih optimal.

Dibandingkan dengan algoritma local search lainnya seperti Simulated Annealing dan Steepest Ascent, GA memiliki keunggulan dalam mengeksplorasi beberapa solusi secara bersamaan melalui populasi, yang seharusnya meningkatkan peluang menemukan solusi yang lebih baik. Namun, meskipun strategi GA lebih eksploratif dibandingkan algoritma lain, hasil akhirnya justru sering kali tidak lebih baik, bahkan cenderung lebih buruk. Waktu eksekusi algoritma ini juga sulit dibandingkan secara langsung dengan algoritma lain, karena durasi pencarian pada GA sangat dipengaruhi oleh jumlah populasi dan

iterasi. Populasi yang lebih besar dan jumlah iterasi yang lebih banyak memang memungkinkan eksplorasi yang lebih mendalam, tetapi juga meningkatkan durasi eksekusi secara signifikan.

Konsistensi hasil akhir GA cukup baik, dengan perbedaan yang tidak terlalu signifikan meskipun parameter populasi dan iterasi divariasikan. GA menunjukkan stabilitas hasil, meskipun terdapat fluktuasi yang disebabkan oleh variasi crossover dan mutasi. Namun, hasil yang dicapai oleh algoritma ini masih belum memuaskan dan cenderung terjebak dalam rentang yang sama tanpa perbaikan signifikan menuju global optima. Hal ini mungkin disebabkan oleh cara kerja algoritma GA yang digunakan, yang masih dalam bentuk dasar tanpa modifikasi tambahan. Implementasi ini tidak memiliki mekanisme untuk memastikan individu terbaik dipertahankan dalam populasi baru. Peluang untuk mempertahankan individu terbaik hanya bergantung pada proses seleksi parent dan crossover, yang bersifat probabilistik. Dalam kasus magic cube dengan ruang solusi yang sangat luas, banyaknya kemungkinan solusi membuat pendekatan berbasis peluang dan perhitungan acak sederhana menjadi kurang efektif untuk mencapai hasil optimal.

Selain itu, algoritma ini hanya mempertimbangkan individu terbaik dari populasi terakhir sebagai hasil akhir. Hal ini berarti bahwa individu yang lebih baik yang mungkin telah ditemukan di generasi sebelumnya bisa tertimpa oleh individu yang kurang baik di populasi terakhir. Akibatnya, hasil akhir tidak selalu mencerminkan performa terbaik yang dicapai selama proses pencarian. Hal ini juga terlihat pada grafik, di mana nilai objective terbaik di setiap populasi sangat acak dan tidak menunjukkan tren perbaikan yang konsisten, semakin menegaskan bahwa mengandalkan faktor-faktor acak dalam kasus ini tidak efektif.

Secara keseluruhan, hasil eksperimen menunjukkan bahwa Genetic Algorithm dalam bentuk dasarnya memiliki keterbatasan signifikan

dalam mencapai hasil optimal, terutama untuk masalah dengan ruang solusi besar seperti *magic cube*. Penambahan iterasi dan populasi belum cukup untuk mendorong perbaikan signifikan, dengan nilai *objective function* yang cenderung stagnan. Modifikasi seperti mempertahankan individu terbaik, adaptasi parameter, atau teknik seleksi yang lebih canggih diperlukan untuk meningkatkan performa. Tanpa perubahan ini, GA sulit bersaing dengan algoritma local search lain yang lebih efektif, dan cenderung terjebak pada hasil yang kurang optimal dengan waktu eksekusi yang mahal jika iterasi dan populasi terlalu besar.

BAB III

KESIMPULAN DAN SARAN

A. Kesimpulan

Dari semua eksperimen yang telah dilakukan, terlihat bahwa masing-masing algoritma memiliki keunikan dan keunggulannya masing-masing. Selain itu, terlihat jelas bahwa masalah magic cube 5x5x5 ini sangat kompleks, sehingga bahkan semua algoritma local search yang digunakan dalam eksperimen ini belum mampu menyelesaikan permasalahan tersebut sepenuhnya.

Dalam permasalahan ini, dapat dilihat bahwa pemilihan successor secara acak adalah pendekatan yang lebih efektif untuk menangani jumlah successor yang sangat besar, karena mampu mempercepat eksekusi program. Hal ini terlihat dari algoritma seperti Stochastic Hill Climbing dan Simulated Annealing yang memiliki kecepatan eksekusi lebih tinggi dibandingkan dengan algoritma Steepest Ascent.

Fungsi objective yang digunakan dalam kasus ini mungkin masih kurang optimal karena belum mampu membantu algoritma local search menemukan solusi secara efektif. Selain itu, algoritma fungsi objective ini juga terlalu kompleks, yang memperlambat eksekusi program, terutama pada algoritma yang memilih successor terbaik di antara semua successor yang tersedia (Steepest Ascent HC, Random Restart HC, HC With Sideways Move).

Beberapa algoritma cenderung terjebak pada kisaran nilai objective yang sama. Hal ini disebabkan oleh beberapa faktor mendasar. Pertama, ruang solusi untuk masalah magic cube 5x5x5 sangat luas dan kompleks, sehingga algoritma kesulitan menemukan jalur perbaikan yang signifikan setelah mencapai titik tertentu. Faktor eksplorasi pada awal iterasi menjadi sangat penting; ketika eksplorasi awal tidak berjalan dengan baik, algoritma cenderung terjebak dalam local optima. Selain itu, jumlah solusi optimal untuk masalah magic cube ini sangat terbatas, yang semakin mempersulit algoritma dalam menentukan jalur eksplorasi yang efektif dan tepat.

Jika dilihat berdasarkan hasil eksperimen Genetic Algorithm adalah algoritma terburuk untuk menyelesaikan permasalahan ini. Variasi nilai parameter seperti jumlah populasi dan banyak iterasi juga tidak memberikan dampak yang signifikan dan bahkan bisa semakin memperburuk waktu eksekusi algoritma jika ukuran parameternya dibuat

semakin besar. Algoritma ini terlalu banyak mengandalkan probabilitas dan perhitungan acak yang membuatnya sangat tidak sesuai untuk kasus magic cube ini dimana jumlah ruang solusi sangatlah luas.

Stochastic Hill Climbing terbukti menjadi salah satu algoritma yang efektif dalam menyelesaikan kasus magic cube 5x5x5. Dengan memanfaatkan pemilihan successor secara acak, algoritma ini memiliki waktu eksekusi yang cepat dan sebanding dengan Simulated Annealing. Proses pemilihan acak ini memungkinkan algoritma untuk menghindari jebakan local optima, karena jika nilai penerus yang ditemukan lebih buruk atau sama, algoritma ini dapat melakukan pemilihan acak ulang hingga ditemukan nilai yang lebih baik. Keberhasilan algoritma ini sangat bergantung pada batas atas iterasi yang ditentukan, di mana semakin besar nilai batas atas iterasi, semakin besar kemungkinan solusi dapat ditemukan karena algoritma memiliki lebih banyak kesempatan untuk melakukan pemilihan acak ulang. Hasil eksperimen menunjukkan bahwa performa algoritma ini cukup baik jika dibandingkan dengan algoritma lain, dengan solusi yang mendekati global optima.

Steepest Ascent HC termasuk salah satu algoritma dengan waktu eksekusi yang cukup lama, serupa dengan HC with Sideways Move dan Random Restart HC. Algoritma ini membutuhkan waktu yang signifikan dalam proses pencarian successor terbaik dari suatu kondisi Cube. Jika waktu eksekusi diabaikan, algoritma ini sebenarnya cukup baik mengingat rata-rata hasil akhir yang dihasilkan pada eksperimen cukup mendekati global optima. Namun, sama seperti Stochastic HC, sifat greedy dari algoritma ini membuatnya mudah terjebak di local optima, terutama ketika jumlah iterasi sudah banyak dan ukuran ruang solusi yang tersisa semakin terbatas. Algoritma ini juga sangat terpengaruh dengan kondisi awal dari Cube dan hal ini tentu saja sulit untuk diprediksi mengingat kondisi awal dari Cube adalah kondisi yang dibangun secara acak.

HC with Sideways Move adalah algoritma yang mirip dengan Steepest Ascent HC, namun perbedaan kecil dalam strategi pergerakannya memberikan dampak yang cukup signifikan. Algoritma ini memungkinkan pergerakan ke successor dengan nilai objective yang sama, yang dikenal sebagai sideways move. Meskipun waktu eksekusi algoritma ini cukup lama, seperti yang telah dijelaskan sebelumnya, jika aspek waktu diabaikan, HC with Sideways Move mungkin mendekati global optima lebih baik dibandingkan Steepest Ascent HC dengan kondisi awal yang sama. Kemampuan untuk melakukan pergerakan sideways membantu algoritma mengatasi jebakan local optima, terutama ketika ruang pencarian solusi semakin menyempit dan perbaikan signifikan menjadi lebih sulit

ditemukan. Hal ini memungkinkan algoritma untuk melanjutkan eksplorasi dan mendekati solusi optima. Sama seperti Steepest Ascent HC algoritma ini juga sangat bergantung dengan kondisi awal Cube. Jika kondisi awal Cube bisa memberikan jalur pencarian yang baik maka hasil dari algoritma ini juga bisa semakin baik.

Random Restart HC pada dasarnya merupakan pengulangan dari Steepest Ascent HC, dan hal ini dapat memberikan dampak signifikan dalam proses pencarian solusi. Mengulangi proses pencarian dengan keadaan awal yang berbeda memungkinkan algoritma untuk mengeksplorasi jalur pencarian yang lebih beragam, sehingga meningkatkan peluang untuk mendapatkan hasil yang lebih baik dibandingkan dengan pencarian sebelumnya. Peningkatan jumlah restart dapat meningkatkan kemungkinan menemukan solusi yang lebih optimal, meskipun hal ini memiliki trade-off berupa waktu eksekusi yang lebih lama. Berdasarkan hasil eksperimen, algoritma ini menunjukkan rata-rata hasil yang cukup baik dibandingkan dengan beberapa algoritma lainnya. Kelemahan utama dari algoritma ini adalah waktu eksekusinya yang sangat lama, karena prosesnya melibatkan pengulangan penuh dari Steepest Ascent HC.

Simulated Annealing menunjukkan performa yang sangat baik dengan hasil yang cukup mendekati global optima dan waktu eksekusi yang relatif cepat. Mekanisme penerimaan solusi yang lebih buruk membantu algoritma ini melakukan eksplorasi ruang solusi secara efektif, sehingga dapat menghindari jebakan local optima. Sama seperti Stochastic HC, pemilihan successor secara acak memberikan peningkatan performa yang signifikan dibandingkan dengan Steepest Ascent HC dan HC with Sideways Move. Namun, permasalahan yang dihadapi algoritma ini mirip dengan algoritma HC lainnya, di mana kesulitan melakukan perbaikan muncul setelah jumlah iterasi bertambah atau, dalam konteks Simulated Annealing, ketika temperatur mulai menurun. Penggunaan fitness function dalam algoritma ini juga sangat mempengaruhi proses pencarian solusi. Fungsi yang digunakan disini cenderung sederhana dengan rata-rata nilai yang cukup kecil. Hal tersebut mungkin juga berdampak pada performa algoritma ini. Perbaikan pada fitness function bisa saja berpotensi untuk meningkatkan hasil, memungkinkan algoritma untuk mengevaluasi solusi dengan lebih efektif dan mendekati global optima. Algoritma ini juga dipengaruhi dengan nilai temperatur awal. Nilai temperatur yang lebih besar memungkinkan proses eksplorasi di awal iterasi berlangsung lebih lama sehingga bisa memberikan jalur menuju solusi yang lebih baik lagi. Namun hal tersebut juga tidak selalu memberikan hasil akhir yang lebih baik mengingat hasil eksperimen yang masih cenderung konsisten meskipun initial temperatur sudah diubah.

Secara keseluruhan, jika mempertimbangkan waktu eksekusi dan hasil, algoritma yang terbaik untuk permasalahan ini adalah Stochastic Hill Climbing (HC). Selain itu, hasil yang diperoleh dari Stochastic HC pun masih dapat ditingkatkan lagi dengan menaikkan batas atas iterasi, yang semakin menunjukkan bahwa algoritma ini sangat baik untuk menyelesaikan kasus magic cube 5x5x5. Di sisi lain, algoritma Simulated Annealing juga bisa menjadi pilihan yang tepat jika dilakukan beberapa perbaikan untuk meningkatkan keseimbangan antara fase eksplorasi dan eksplorasi. Namun, jika waktu eksekusi dapat diabaikan, algoritma HC lain seperti Steepest Ascent HC, HC with Sideways Move, dan Random Restart HC juga bisa menjadi pilihan yang baik, mengingat rata-rata hasil eksperimen yang mereka capai cukup memuaskan.

B. Saran

Melalui berbagai eksperimen yang sudah dilakukan dengan menggunakan berbagai macam algoritma local search bisa diketahui bahwa algoritma local search biasa masih belum bisa digunakan untuk menyelesaikan permasalahan magic cube ini dan saran yang bisa diberikan adalah perlu dilakukannya modifikasi atau gabungan algoritma untuk menciptakan algoritma pencarian yang lebih baik lagi.

Beberapa hal yang bisa dipertimbangkan untuk meningkatkan performa Genetic Algorithm adalah penerapan strategi seleksi khusus untuk mempertahankan individu terbaik di setiap generasi. Strategi ini dapat membantu algoritma bergerak lebih terarah menuju tujuan optimal. Untuk algoritma Simulated Annealing, perbaikan dalam proses perhitungan temperatur (mengganti konstanta atau mengganti pendekatan perhitungan secara keseluruhan) dapat dilakukan untuk mencapai keseimbangan yang lebih baik antara fase eksplorasi dan fase eksplorasi. Selain itu, penggunaan fungsi fitness yang lebih baik mungkin dapat memberikan hasil yang lebih optimal dan meningkatkan kualitas solusi yang dihasilkan. Penerapan adaptive temperature scheduling, restart acak, serta kombinasi dengan algoritma lain juga bisa dipertimbangkan untuk digunakan pada Simulated Annealing. Untuk random restart HC mungkin bisa ditambahkan Sideways Move untuk meningkatkan peluang mendapatkan solusi yang lebih baik lagi. Untuk Stochastic HC batas atas dari iterasi mungkin bisa ditingkatkan untuk memberikan kesempatan yang lebih banyak dalam proses pencarian successor random yang lebih baik lagi.

Secara keseluruhan permasalahan utama dari lamanya eksekusi program adalah di dalam perhitungan nilai objectivenya yang terlalu kompleks. Mungkin proses perhitungan

ini bisa dioptimalkan lagi atau bahkan bisa dicari fungsi objective lain yang lebih sesuai dan lebih cepat untuk kasus permasalahan ini. Selain itu, penggunaan bahasa pemrograman yang lain juga dapat mempengaruhi waktu eksekusi program.

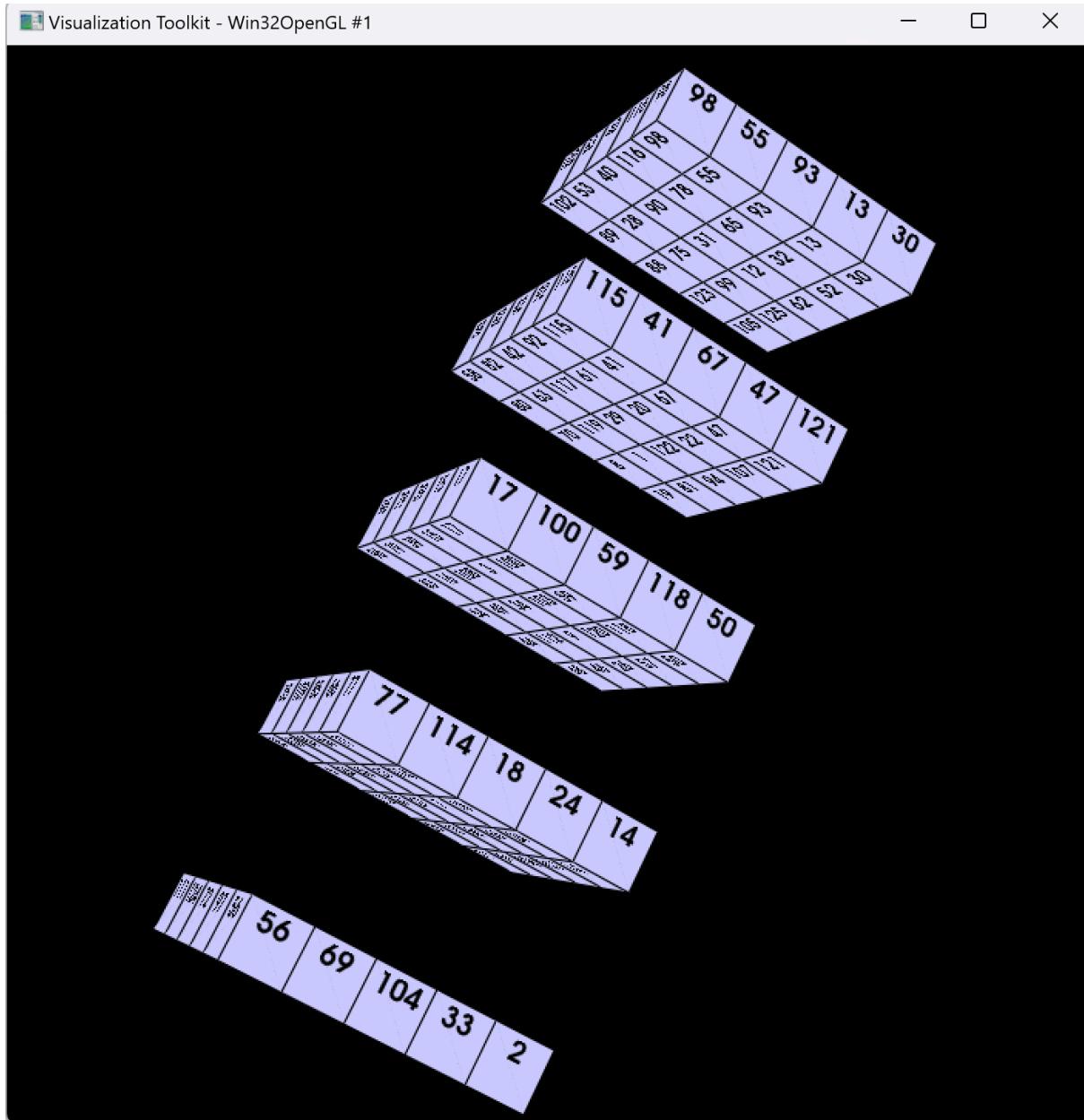
PEMBAGIAN TUGAS

Nama / NIM	Tugas
Ivan Hendrawan Tan / 13522111	<ul style="list-style-type: none">● Stochastic HC● Eksperimen dan Analisis● Laporan
William Glory Henderson / 13522113	<ul style="list-style-type: none">● Genetic Algorithm● Simulated Annealing● HC with Sideways Move● Eksperimen dan Analisis● Laporan
Naufal Adnan / 13522116	<ul style="list-style-type: none">● Random Restart HC● Eksperimen dan Analisis● Laporan
Mesach Harmasendro / 13522117	<ul style="list-style-type: none">● GUI● Steepest Ascent HC● Genetic Algorithm● Eksperimen dan Analisis● Laporan

LAMPIRAN

- Untuk source code yang lebih jelas bisa diakses di repository berikut:
https://github.com/wegeh/IF3170_Tubes1

- 3D visualisasi:



REFERENSI

"Magic Features of Three-Dimensional Magic Squares." *Magischvierkant*. Diakses pada 1 November 2024. <https://www.magischvierkant.com/three-dimensional-eng/magic-features/>

"Magic Cubes." *Trump*. Diakses pada 1 November 2024.
<https://www.trump.de/magic-squares/magic-cubes/cubes-1.html>

Stuart Russell dan Peter Norvig. *Artificial Intelligence: A Modern Approach*. Edisi ke-3, Pearson, 2010.

Trenkler, M. "A Construction of Magic Cubes." *Math. Gaz.* 84, 36-41, 2000.

Wolfram MathWorld. "Magic Cube." Diakses pada 1 November 2024.
<https://mathworld.wolfram.com/MagicCube.html>