

TUGAS BESAR 1
IF3170 Inteligensi Artifisial
Pencarian Solusi Diagonal Magic Cube dengan Local Search



Disusun Oleh:

- | | |
|----------|--------------------------------|
| 13522061 | Maximilian Sulistiyo |
| 13522075 | Marvel Pangondian |
| 13522083 | Evelyn Yosiana |
| 13522103 | Steven Tjhia |
| 13521154 | Naufal Baldemar Ardanni |

Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

DAFTAR ISI

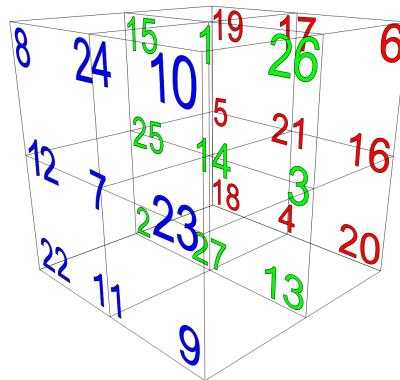
DAFTAR ISI.....	2
BAB 1.....	3
BAB 2.....	6
2.1. Pemilihan Objective Function.....	6
2.2. Implementasi Class Cube.....	6
2.3. Implementasi Algoritma Local Search.....	10
2.2.1. Steepest Ascent Hill-climbing.....	10
2.2.2. Hill-climbing with Sideways Move.....	16
2.2.3. Random Restart Hill-climbing.....	21
2.2.4. Stochastic Hill-climbing.....	42
2.2.5. Simulated Annealing.....	48
2.2.6. Genetic Algorithm.....	56
BAB 3.....	106
BAB 4.....	108
BAB 5.....	109
LAMPIRAN.....	110
DAFTAR PUSTAKA.....	111

BAB 1

Deskripsi Persoalan

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

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



- Terdapat 9 potongan bidang, yaitu:



- Diagonal yang dimaksud adalah yang dilingkari warna merah saja
- Ilustrasi dan penjelasan lebih detail bisa anda lihat di link berikut: [Features of the magic cube - Magisch vierkant](#)

Pada tugas ini, peserta kuliah akan menyelesaikan permasalahan Diagonal Magic Cube berukuran 5x5x5. Initial state dari suatu kubus adalah susunan angka 1 hingga 5^3 secara acak. Kemudian, tiap iterasi pada algoritma local search, langkah yang boleh dilakukan adalah menukar posisi dari 2 angka pada kubus tersebut (2 angka yang ditukar tidak harus bersebelahan). Khusus untuk genetic algorithm, boleh dilakukan penukaran posisi lebih dari 2 angka sekaligus dalam satu iterasi (tetapi hanya menukar posisi 2 angka saja juga diperbolehkan).

Anda akan diminta untuk mengimplementasikan rencana yang telah Anda buat pada Tugas Kecil 1. Berikut merupakan hal-hal yang perlu dilakukan oleh setiap kelompok:

- Implementasikan 3 algoritma local search dengan rincian sebagai berikut:
 - Salah satu algoritma hill-climbing
 - Simulated Annealing
 - Genetic Algorithm
- Lakukan eksperimen dengan skema sebagai berikut:
 - Jalankan setiap algoritma sebanyak **3 kali**, kemudian catat beberapa hal berikut:
 - Berlaku untuk semua algoritma
 - State awal dan akhir dari kubus
 - Nilai objective function akhir yang dicapai
 - Plot nilai objective function terhadap banyak iterasi yang telah dilewati
 - Durasi proses pencarian
 - Berlaku hanya untuk Steepest Ascent Hill-Climbing dan Stochastic Hill-Climbing
 - Banyak iterasi hingga proses pencarian berhenti
 - Berlaku hanya untuk Hill-Climbing with Sideways Move
 - Banyak iterasi hingga proses pencarian berhenti
 - Note: Tambahkan parameter maximum sideways move, dimana ketika banyak sideways move yang dilakukan sudah mencapai maksimum, pencarian dihentikan
 - Berlaku hanya untuk Random Restart Hill-Climbing
 - Banyak restart
 - Banyak iterasi per restart
 - Note: Tambahkan parameter maximum restart, dimana ketika banyak restart sudah mencapai maksimum, pencarian dihentikan.
 - Berlaku hanya untuk Simulated Annealing
 - Plot $e^{\frac{\Delta E}{T}}$ terhadap banyak iterasi yang telah dilewati
 - Frekuensi ‘stuck’ di local optima
 - Khusus untuk Genetic Algorithm, lakukan beberapa hal berikut:

- Terdapat 2 parameter yang dapat diubah, yaitu **jumlah populasi** dan **banyak iterasi**.
- Jadikan jumlah populasi sebagai kontrol, kemudian pilih **3 variasi** banyak iterasi yang berbeda. Jalankan program sebanyak masing-masing **3 kali** untuk setiap konfigurasi parameter.
- Jadikan banyak iterasi sebagai kontrol, kemudian pilih **3 variasi** jumlah populasi yang berbeda. Jalankan program sebanyak masing-masing **3 kali** untuk setiap konfigurasi parameter.
- Untuk setiap eksperimen, catat beberapa hal berikut:
 - State awal dan akhir dari kubus
 - Nilai objective function akhir yang dicapai
 - Plot nilai objective function terhadap banyak iterasi yang telah dilewati (Cukup plot nilai objective function maksimum dan rata-rata dari populasi terhadap banyak iterasi yang telah dilewati. Jika sudah terlanjur membuat plot untuk tiap individu pada populasi tidak menjadi masalah, silahkan diberikan keterangan saja maksud plotnya apa)
 - Jumlah populasi
 - Banyak iterasi
 - Durasi proses pencarian
- Lakukan analisis terhadap hasil eksperimen, berikut merupakan beberapa pertanyaan yang dapat menjadi acuan untuk analisis yang Anda lakukan (Anda boleh menambahkan beberapa pertanyaan tambahan jika dirasa perlu untuk dijelaskan):
 - Seberapa dekat tiap-tiap algoritma bisa mendekati global optima dan mengapa hasilnya demikian?
 - Bagaimana perbandingan hasil pencarian tiap-tiap algoritma dengan algoritma local search yang lain?
 - Bagaimana perbandingan durasi proses pencarian tiap algoritma relatif terhadap algoritma lainnya?
 - Seberapa konsisten hasil akhir yang didapatkan dari tiap-tiap eksperimen yang dilakukan?
 - Bagaimana pengaruh banyak iterasi dan jumlah populasi terhadap hasil akhir pencarian pada Genetic Algorithm?
 - dst...
- Program yang dibuat harus bisa **memvisualisasikan** state awal kubus, state akhir kubus, dan juga hasil eksperimentnya (sesuaikan informasi hasil eksperimen yang ditampilkan dengan ketentuan yang telah dijelaskan di poin sebelum ini).
- Cara visualisasi dibebaskan kepada kelompok masing-masing selama seluruh angka pada kubus dan juga hasil eksperimen terlihat dengan jelas.
- Dibebaskan menggunakan bahasa pemrograman apapun.
- Diperbolehkan untuk menggunakan heuristik yang Anda buat sendiri atau dari referensi lain untuk optimasi pencarian solusi, asalkan masih dalam lingkup local search. Jangan lupa jelaskan heuristik yang Anda pakai di laporan.

BAB 2

Pembahasan

2.1. Pemilihan Objective Function

Objective Function yang ditetapkan oleh penulis pada setiap *state* adalah fungsi yang menghitung nilai dari setiap *state* berdasarkan jumlah angka pada setiap baris, kolom, tiang, diagonal ruang, dan diagonal pada suatu potongan bidang yang memiliki nilai 315. Nilai 315 ini didasarkan pada rumus magic number (S), yaitu:

$$S = m \cdot (m^3 + 1)/2$$

Objective Function dimulai dengan mencari jumlah dari setiap baris, kolom, tiang, diagonal ruang, dan diagonal irisan bidang. Kemudian, untuk setiap jumlah tersebut akan dihitung banyaknya yang bernilai 315. Jumlah dari semua nilai yang bernilai 315 akan menjadi nilai dari *state* tersebut. Alasan penulis memilih *objective function* adalah untuk mencerminkan tujuan dari *magic cube*, yaitu memastikan bahwa setiap baris, kolom, tiang, diagonal ruang, dan diagonal pada suatu potongan bidang memiliki jumlah yang sama dengan *magic number*.

2.2. Implementasi Class Cube

Class ini merupakan representasi dari Cube. Cube disini merupakan matrix 3 dimensi yang direpresentasikan menggunakan array 1D. Pemilihan ini digunakan untuk mempermudah proses komputasi karena komputasi array 1D lebih mudah dibandingkan proses array of array of array. Representasi ini pun digunakan untuk mempermudah *Genetic Algorithm* yang menggunakan representasi ini untuk melakukan crossover.

Pada class ini pun terdapat beberapa fungsi penting seperti

- `toIndex` : Menghitung index dari nilai posisi 3D
- `Insert_value` : Memasukkan suatu nilai ke dalam posisi 3D
- `Calculate_state_value` : Menghitung nilai state dari suatu cube dan memasukkannya ke dalam atribut cube
- `get_value` : Mendapatkan nilai dari suatu posisi
- `Delete_value` : Menghapus nilai dari suatu posisi
- `print_cube` : Mencetak cube
- `objective_cube` : Menghitung nilai state dari suatu cube
- `generate_successor` : Membangkitkan semua successor dari cube

Source Code

```
import random
import copy
import time

class Cube:
    def __init__(self, x_size, y_size, z_size, random=False, array=None):
        self.x_size = x_size
        self.y_size = y_size
        self.z_size = z_size
        self.array = [None] * (self.x_size * self.y_size * self.z_size)
        self.magic_number = (
            x_size * (x_size**3 + 1) // 2
            if x_size == y_size and y_size == z_size
            else None
        )
        if random:
            self.random_cube()
            self.state_value = objective_function(self)
        if array:
            self.array = array
            self.state_value = objective_function(self)

    def random_cube(self):
        numbers = list(range(1, self.x_size**3 + 1))
        random.shuffle(numbers)
        self.array = numbers

    def to_index(self, x, y, z):
        return x * (self.y_size * self.z_size) + y * (self.z_size) + z

    def insert_value(self, x, y, z, value):
        self.array[self.to_index(x, y, z)] = value

    def calculate_state_value(self):
        self.state_value = objective_function(self)

    def get_value(self, x, y, z):
        return self.array[self.to_index(x, y, z)]

    def delete_value(self, x, y, z):
        index = self.to_index(x, y, z)
        value = self.array[index]
```

```

        self.array[index] = None
        return value

    def print_cube(self):
        for x in range(self.x_size):
            print(f"Layer {x}:")
            for y in range(self.y_size):
                row = [self.array[self.to_index(x, y, z)] for z in range(self.z_size)]
                print(row)
            print()

    def copy(self):
        new_array = copy.deepcopy(self.array)
        new_cube = Cube(self.x_size, self.y_size, self.z_size, array=new_array)
        new_cube.state_value = self.state_value
        return new_cube

    def check_rows(cube):
        size = cube.x_size
        arr = []
        for x in range(size):
            for y in range(size):
                row_sum = sum(cube.get_value(x, y, z) for z in range(size))
                arr.append(row_sum)
        return arr

    def check_columns(cube):
        size = cube.x_size
        arr = []
        for x in range(size):
            for z in range(size):
                col_sum = sum(cube.get_value(x, y, z) for y in range(size))
                arr.append(col_sum)
        return arr

    def check_pillars(cube):
        size = cube.x_size
        arr = []
        for y in range(size):
            for z in range(size):
                pillar_sum = sum(cube.get_value(x, y, z) for x in range(size))
                arr.append(pillar_sum)
        return arr

```

```

        arr.append(pillar_sum)
    return arr

def check_space_diagonals(cube):
    size = cube.x_size
    diag1 = sum(cube.get_value(i, i, i) for i in range(size))
    diag2 = sum(cube.get_value(i, i, size - i - 1) for i in range(size))
    diag3 = sum(cube.get_value(i, size - i - 1, i) for i in range(size))
    diag4 = sum(cube.get_value(i, size - i - 1, size - i - 1) for i in range(size))
    return [diag1, diag2, diag3, diag4]

def check_plane_diagonals(cube):
    arr = []
    size = cube.x_size
    for x in range(size):
        diag1 = sum(cube.get_value(x, i, i) for i in range(size))
        diag2 = sum(cube.get_value(x, i, size - i - 1) for i in range(size))
        arr.append(diag1)
        arr.append(diag2)

    for y in range(size):
        diag1 = sum(cube.get_value(i, y, i) for i in range(size))
        diag2 = sum(cube.get_value(size - i - 1, y, i) for i in range(size))
        arr.append(diag1)
        arr.append(diag2)

    for z in range(size):
        diag1 = sum(cube.get_value(i, i, z) for i in range(size))
        diag2 = sum(cube.get_value(size - i - 1, i, z) for i in range(size))
        arr.append(diag1)
        arr.append(diag2)

    return arr

def objective_function_view(cube):
    arr = []
    arr += (
        check_rows(cube)
        + check_columns(cube)
        + check_pillars(cube)
        + check_plane_diagonals(cube)
    )

```

```

        + check_space_diagonals(cube)
    )
print(arr)

def generate_successor(curr_cube_arr):
    successors = []
    n = len(curr_cube_arr)
    for i in range(n):
        for j in range(i + 1, n):
            new_arr = curr_cube_arr[:]
            new_arr[i], new_arr[j] = new_arr[j], new_arr[i]
            successors.append(Cube(5, 5, 5, False, new_arr))
    return successors

def random_1d_array(n):
    numbers = list(range(1, n**3 + 1))
    random.shuffle(numbers)
    return numbers

def objective_function(cube):
    arr = []
    arr += (
        check_rows(cube)
        + check_columns(cube)
        + check_pillars(cube)
        + check_plane_diagonals(cube)
        + check_space_diagonals(cube)
    )
    return sum(1 for el in arr if el == cube.magic_number)

```

2.3. Implementasi Algoritma Local Search

2.2.1. Steepest Ascent Hill-climbing

Langkah-langkah pada algoritma ini :

1. Membuat sebuah cube random dan dihitung state valuenya, setiap cube dan state valuenya disimpan pada array cubes dan values. Dihitung juga banyak iterasi yang dilakukan
2. Melakukan loop dimulai dengan pembangkitan seluruh successor dari current state

3. Menghitung seluruh state value dari successor dan mencari yang terbaik
4. Melakukan komparasi dari nilai neighbour terbaik saat ini dan nilai state sekarang. Jika lebih besar maka simpan neighbour dan valuenya ke current state, Jika tidak maka return current state

Source Code

```
def SteepestAscentHillClimbingCube(init_cube):  
    current_cube = Cube(5, 5, 5, False, init_cube)  
    current_value = current_cube.state_value  
    cubes = [current_cube]  
    values = [current_value]  
    count_iter = 0  
  
    while True:  
        print("Current val: ", current_value)  
        count_iter += 1  
        neighbors = generate_successor(current_cube.array)  
        neighbors_value = [cube.state_value for cube in neighbors]  
        best_neighbor = neighbors[np.argmax(neighbors_value)]  
        best_value = max(neighbors_value)  
  
        if best_value > current_value:  
            current_cube = best_neighbor  
            current_value = best_value  
        else:  
            return cubes, values, count_iter  
  
    cubes.append(current_cube)  
    values.append(current_value)
```

2.2.1.1. Percobaan 1

State Awal

3D Cube Matrix Layers

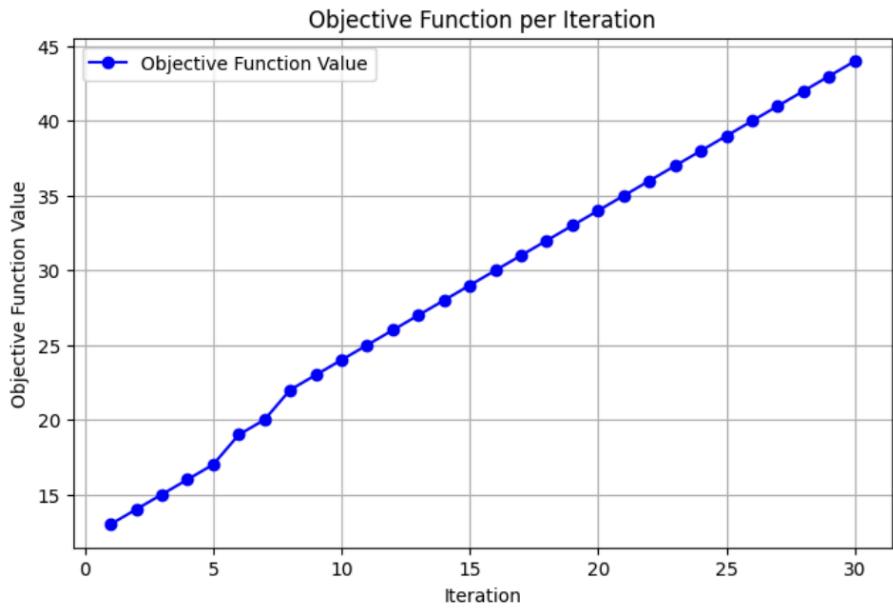
Matrix Level 1					Matrix Level 2					Matrix Level 3				
1	2	3	4	5	26	27	28	29	30	51	52	53	54	55
6	7	8	9	10	31	32	33	34	35	56	57	58	59	60
11	12	13	14	15	36	37	38	39	40	61	62	63	64	65
16	17	18	19	20	41	42	43	44	45	66	67	68	69	70
21	22	23	24	25	46	47	48	49	50	71	72	73	74	75
Matrix Level 4					Matrix Level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

State Akhir

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

Nilai Objective Function: 44

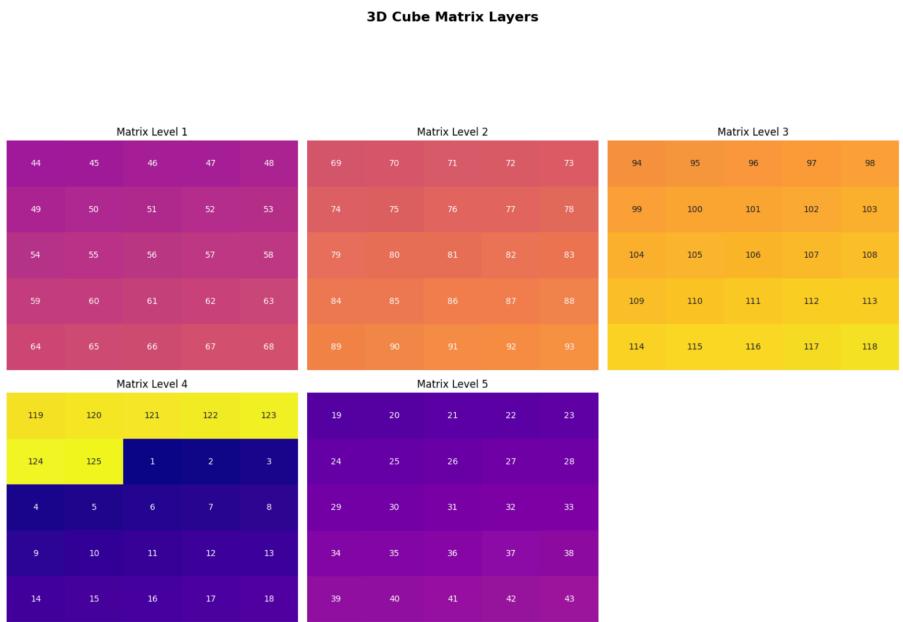
Plot Nilai Objective Function:



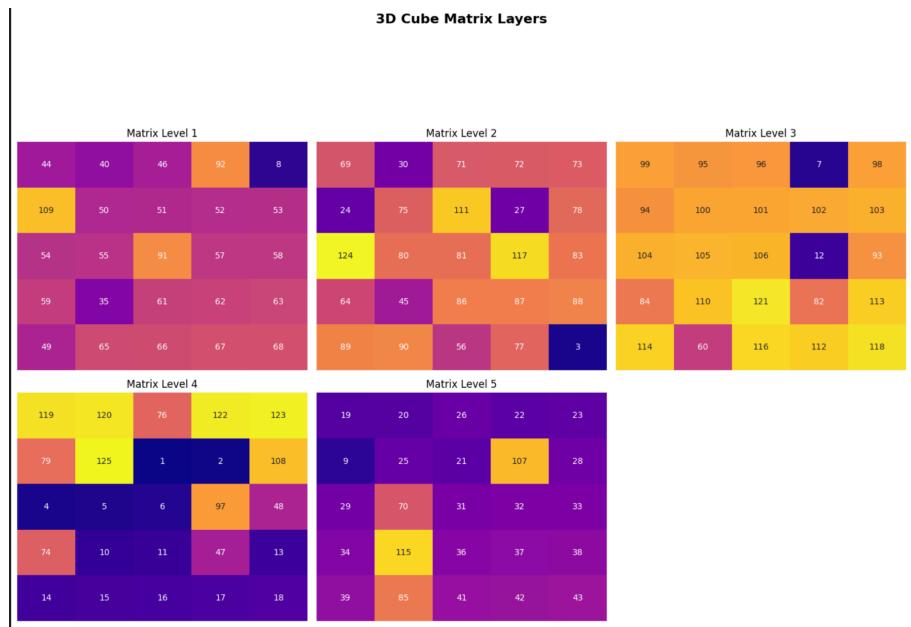
Banyak Iterasi: 30

2.2.1.2. Percobaan 2

State Awal

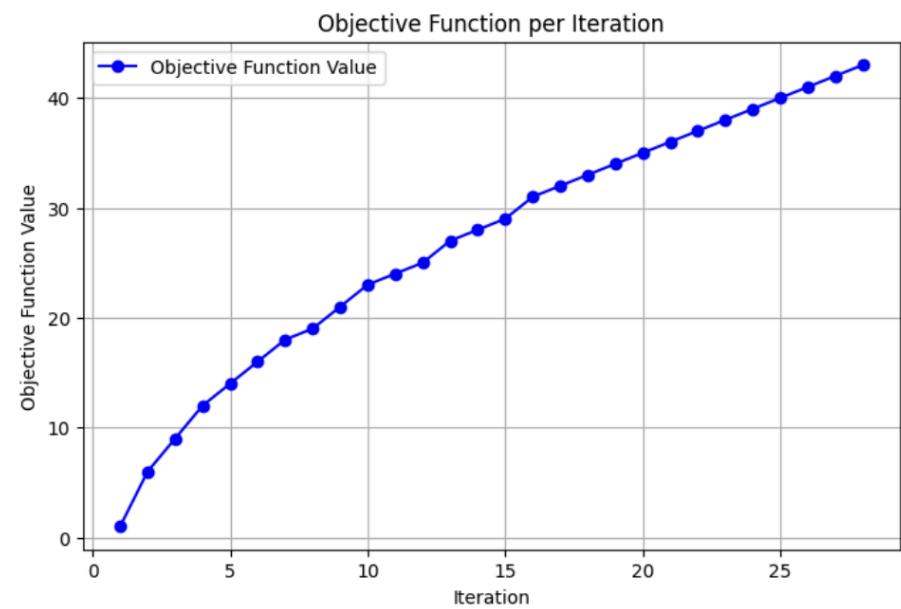


State Akhir



Nilai Objective Function: 43

Plot Nilai Objective Function:



Banyak Iterasi: 28

2.2.1.3. Percobaan 3

State Awal

3D Cube Matrix Layers

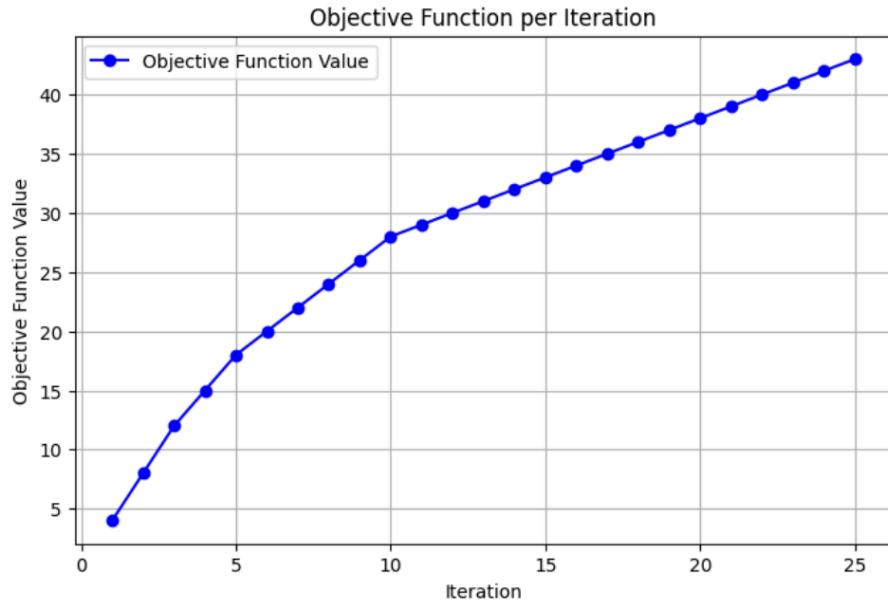
Matrix Level 1					Matrix Level 2					Matrix Level 3				
86	87	88	89	90	111	112	113	114	115	11	12	13	14	15
91	92	93	94	95	116	117	118	119	120	16	17	18	19	20
96	97	98	99	100	121	122	123	124	125	21	22	23	24	25
101	102	103	104	105	1	2	3	4	5	26	27	28	29	30
106	107	108	109	110	6	7	8	9	10	31	32	33	34	35
Matrix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

State Akhir

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

Nilai Objective Function: 43

Plot Nilai Objective Function:



Banyak Iterasi: 25

2.2.2. Hill-climbing with Sideways Move

Langkah-langkah pada algoritma ini adalah

1. Membuat sebuah cube random dan dihitung state valuenya, setiap cube dan state valuenya disimpan pada array cubes dan values. Dihitung juga banyak iterasi yang dilakukan dan juga inisiasi dari sideways count
2. Melakukan loop dimulai dengan pembangkitan seluruh successor dari current state
3. Menghitung seluruh state value dari successor dan mencari yang terbaik
4. Melakukan komparasi dari nilai neighbour terbaik saat ini dan nilai state sekarang. Jika lebih besar sama dengan maka simpan neighbour dan valuenya ke current state. Jika nilai sama dengan sebelumnya maka pindah ke state tersebut dan increment sideway count. Jika state value neighbour lebih kecil atau sideways count sudah melebihi maksimum maka return current state

Source Code

```
def HillClimbingWithSidewaysMoveCube(init_cube, max_sideways):
    current_cube = Cube(5, 5, 5, False, init_cube)
    current_value = current_cube.state_value
    neighbor_value = current_value

    count_iter = 0
    sideways_count = 0
```

```

        cubes = [current_cube]
        values = [current_value]

        while neighbor_value >= current_value and sideways_count <
max_sideways:
            count_iter += 1
            neighbors = generate_successor(current_cube.array)
            neighbors_value = [cube.state_value for cube in neighbors]
            best_neighbor = neighbors[np.argmax(neighbors_value)]
            neighbor_value = max(neighbors_value)

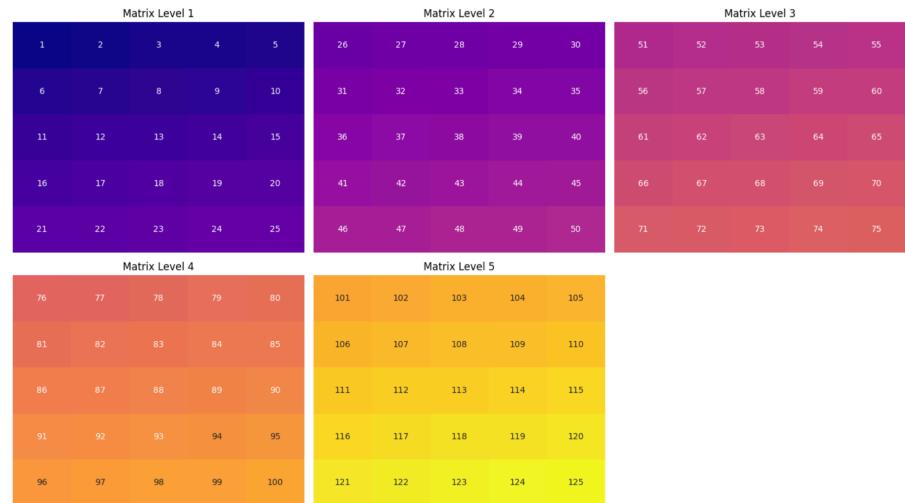
            if neighbor_value >= current_value:
                if neighbor_value > current_value:
                    sideways_count = 0
                elif neighbor_value == current_value:
                    sideways_count += 1
                current_cube = best_neighbor
                current_value = neighbor_value
            print("Current val:", current_value, "| Neighbor Val:",
neighbor_value, "| Sideways Count:", sideways_count)

        return cubes, values, count_iter
    
```

2.2.2.1. Percobaan 1

State Awal

3D Cube Matrix Layers



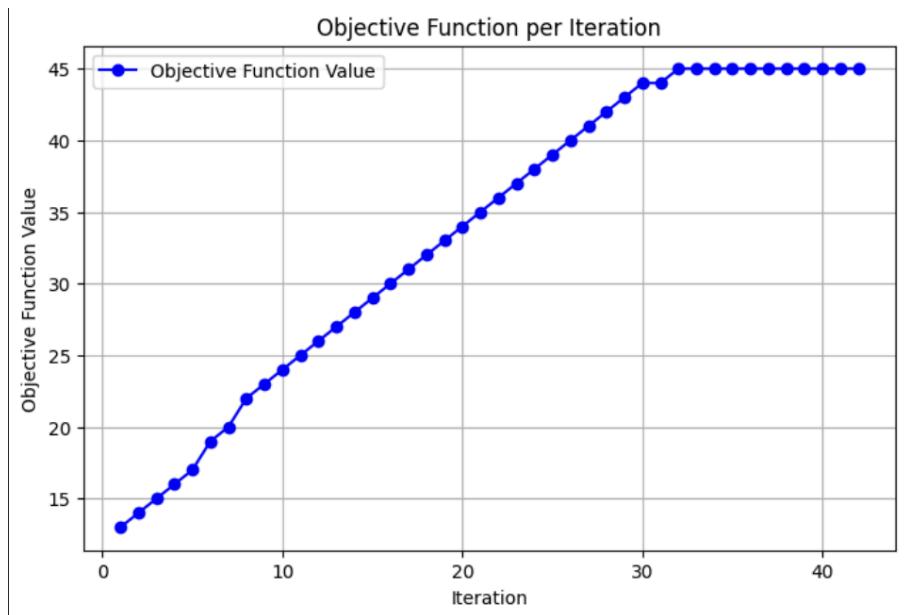
State Akhir

3D Cube Matrix Layers

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

Nilai Objective Function: 45

Plot Nilai Objective Function:



Banyak Iterasi: 41

Max Sideways : 10

2.2.2.2. Percobaan 2

State Awal

3D Cube Matrix Layers

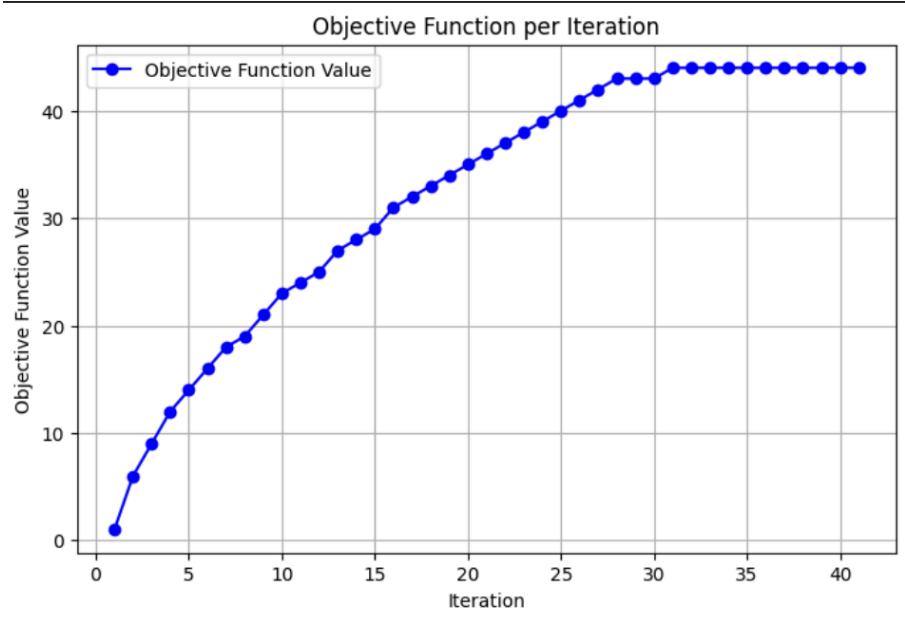
Matrix Level 1					Matrix Level 2					Matrix Level 3				
44	45	46	47	48	69	70	71	72	73	94	95	96	97	98
49	50	51	52	53	74	75	76	77	78	99	100	101	102	103
54	55	56	57	58	79	80	81	82	83	104	105	106	107	108
59	60	61	62	63	84	85	86	87	88	109	110	111	112	113
64	65	66	67	68	89	90	91	92	93	114	115	116	117	118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

State Akhir

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

Nilai Objective Function: 44

Plot Nilai Objective Function



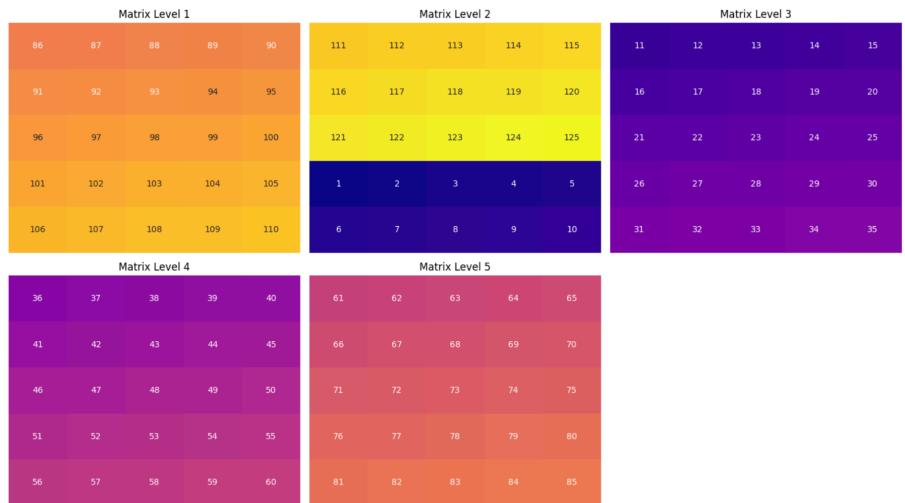
Banyak Iterasi: 40

Max Sideways : 10

2.2.2.3. Percobaan 3

State Awal

3D Cube Matrix Layers



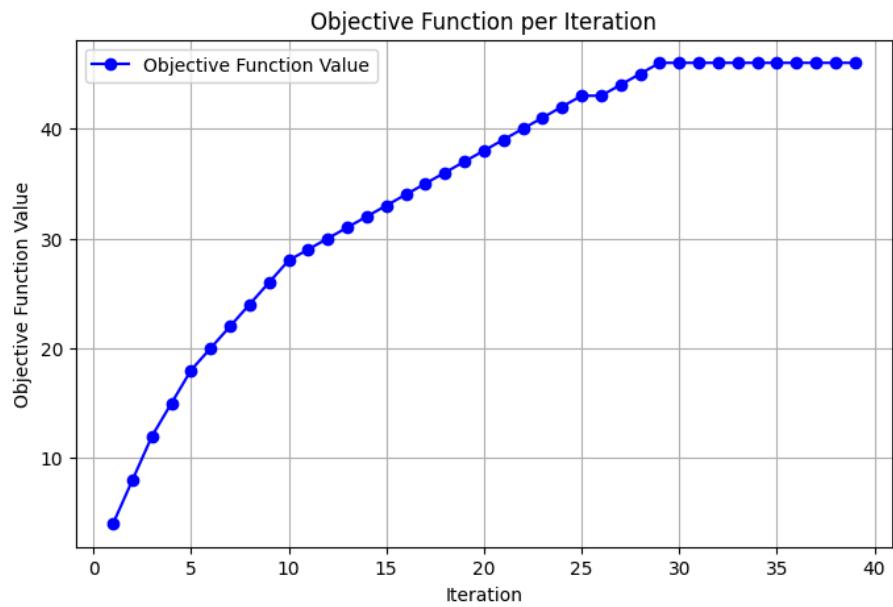
State Akhir

3D Cube Matrix Layers



Nilai Objective Function: 46

Plot Nilai Objective Function:



Banyak Iterasi: 38

Max Sideways : 10

2.2.3. Random Restart Hill-climbing

Langkah-langkah pada algoritma ini:

1. Melakukan local search inisial menggunakan steepest ascent hill climbing untuk menentukan nilai awal
2. Kemudian lakukan restart sebanyak max restart dimana pada setiap iterasi membangkitkan cube baru untuk dilakukan steepest ascent hill climbing
3. Setelah max restart fungsi ini mengembalikan hasil local search per restart, value akhir setiap restart, iterasi per restart, dan jumlah restart yang dilakukan

Source Code

```
def RandomRestartHillClimbingCube(init_cube, max_restart):

    restart = 0
    cubes, values, count_iter =
    SteepestAscentHillClimbingCube(init_cube)

    current_cube = cubes[0]
    current_value = current_cube.state_value

    cubes_per_restart = [cubes]
    values_per_restart = [values]
    iteration_per_restart = [count_iter]

    print(f"Best value at restart {restart} = {values[-1]}")
    if values[-1] < current_value:
        current_value = values[-1]
        current_cube = cubes[-1]

    while restart < max_restart:
        restart += 1
        new_cube = random_1d_array(5) # Generate a new random Cube
instance
        cubes, values, count_iter =
        SteepestAscentHillClimbingCube(new_cube)

        cubes_per_restart.append(cubes)
        values_per_restart.append(values)
        iteration_per_restart.append(count_iter)

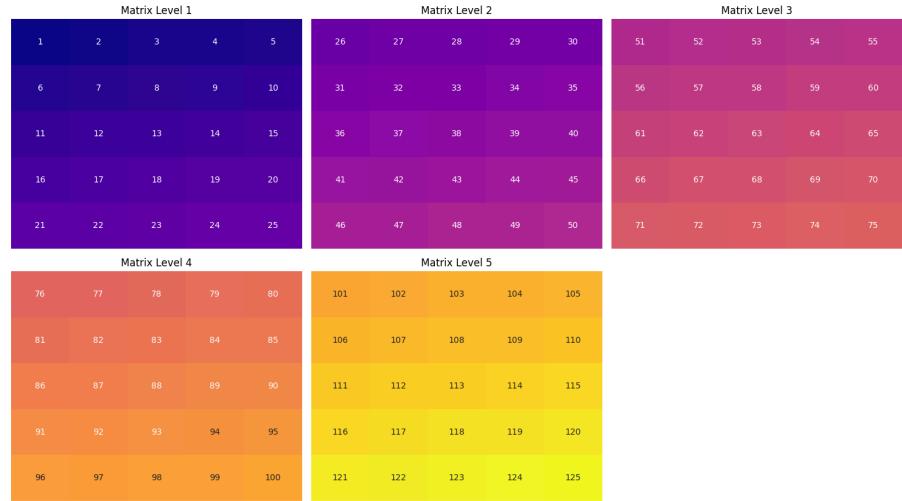
        print(f"Best value at restart {restart} = {values[-1]}")
        if values[-1] < current_value:
            current_value = values[-1]
            current_cube = cubes[-1]
    return cubes_per_restart, values_per_restart,
iteration_per_restart, restart
```

2.2.3.1. Percobaan 1

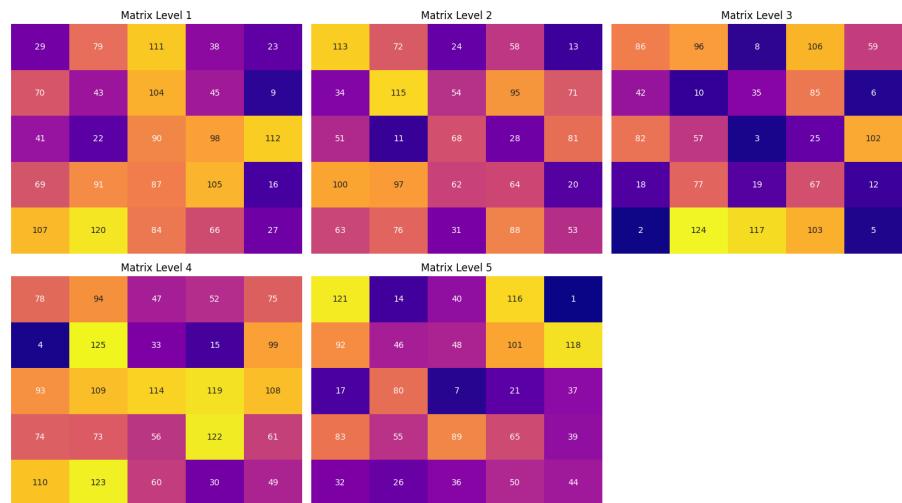
State Awal

Initial state restart - 0

Fitness Value: 13

**Initial state restart - 1**

Fitness Value: 0



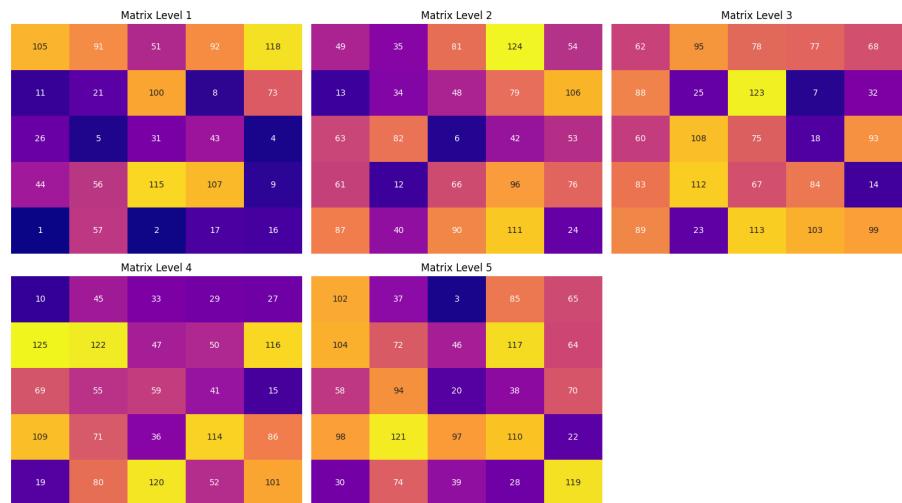
Initial state restart - 2

Fitness Value: 0



Initial state restart - 3

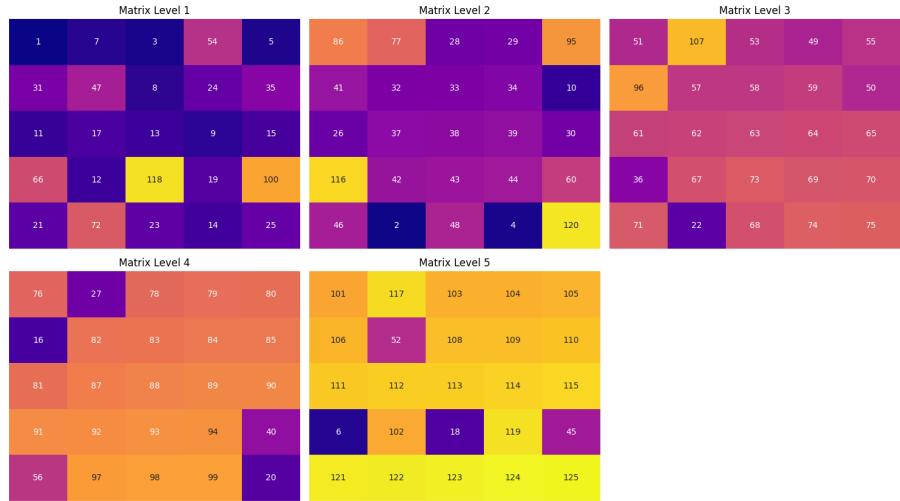
Fitness Value: 0



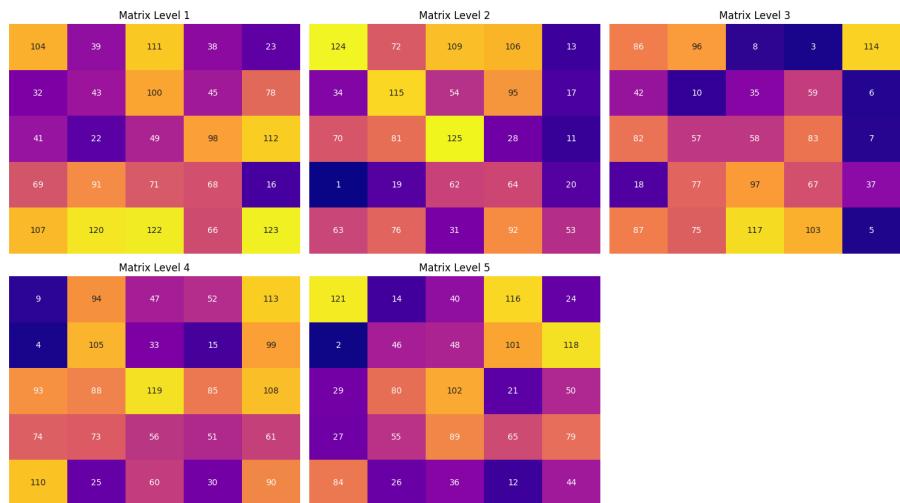
State Akhir

Final state restart - 0

Fitness Value: 44

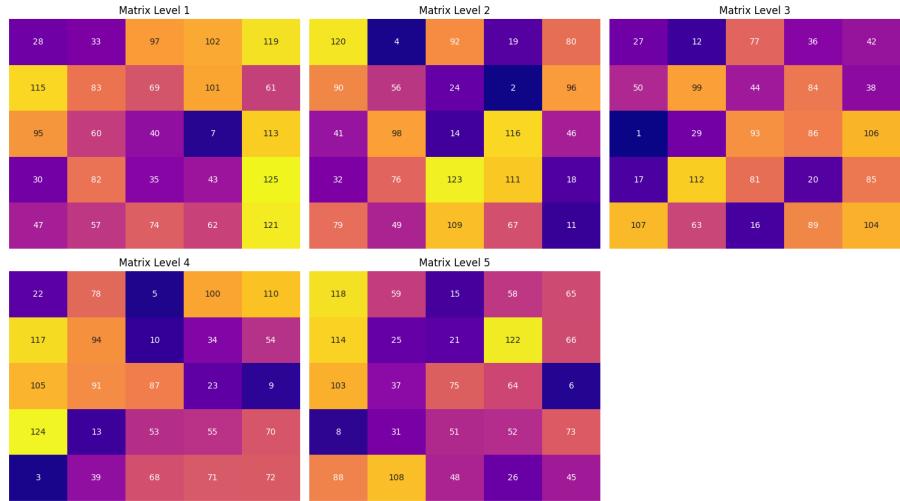
**Final state restart - 1**

Fitness Value: 41



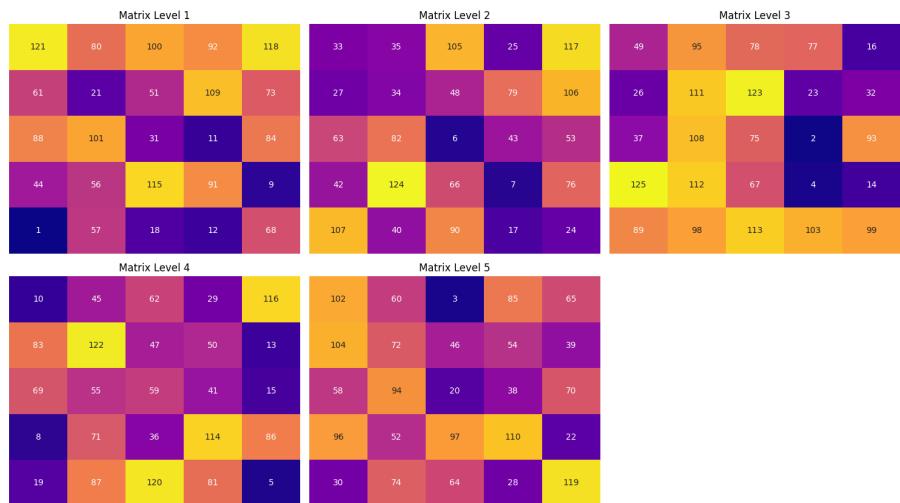
Final state restart - 2

Fitness Value: 40



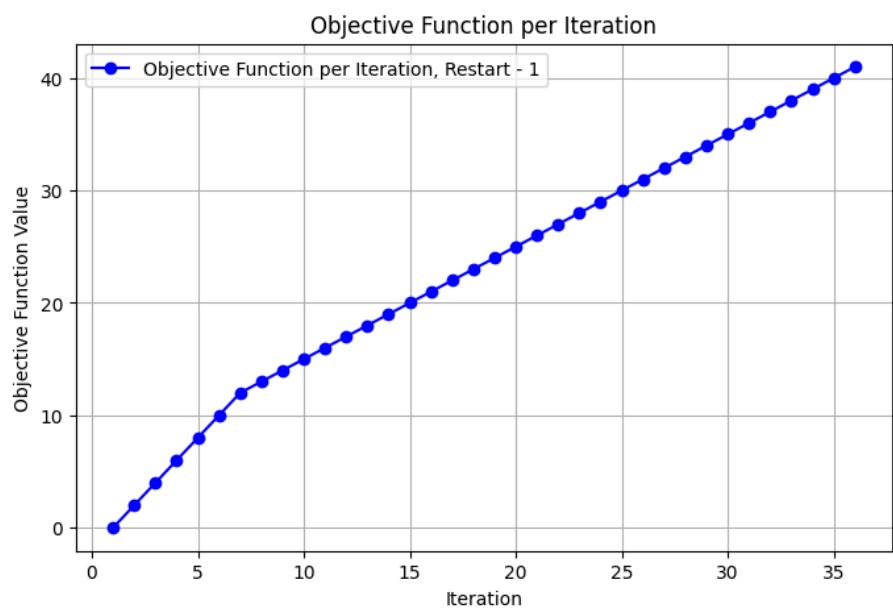
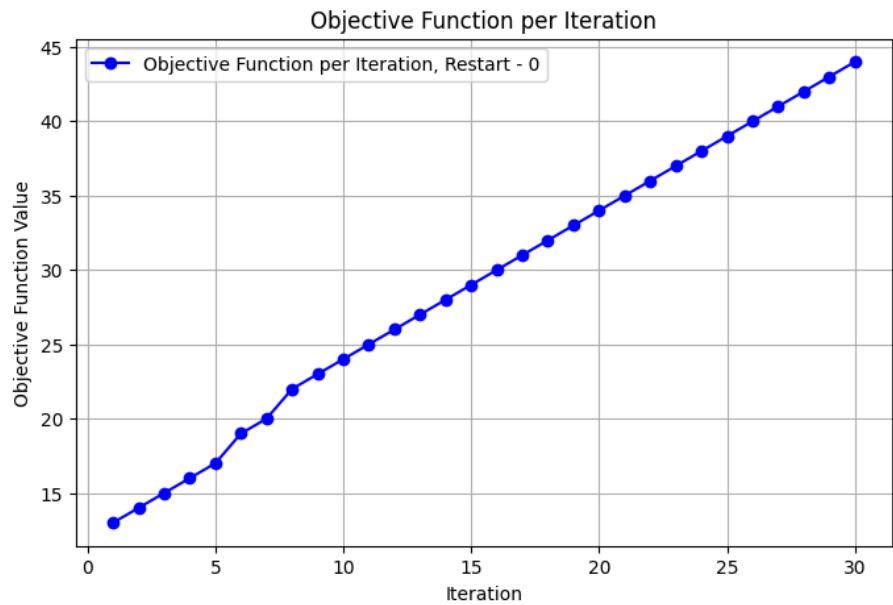
Final state restart - 3

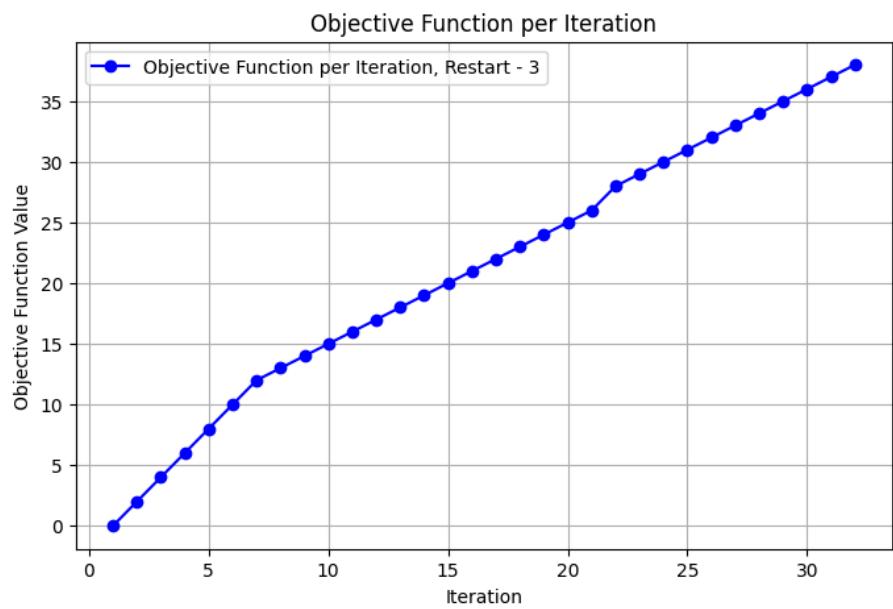
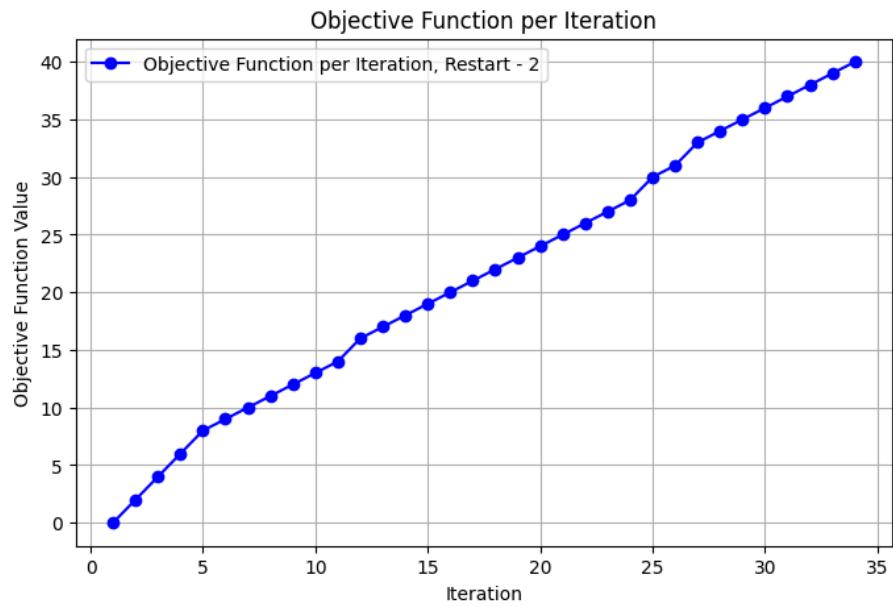
Fitness Value: 38



Nilai Objective Function Terbesar: 44

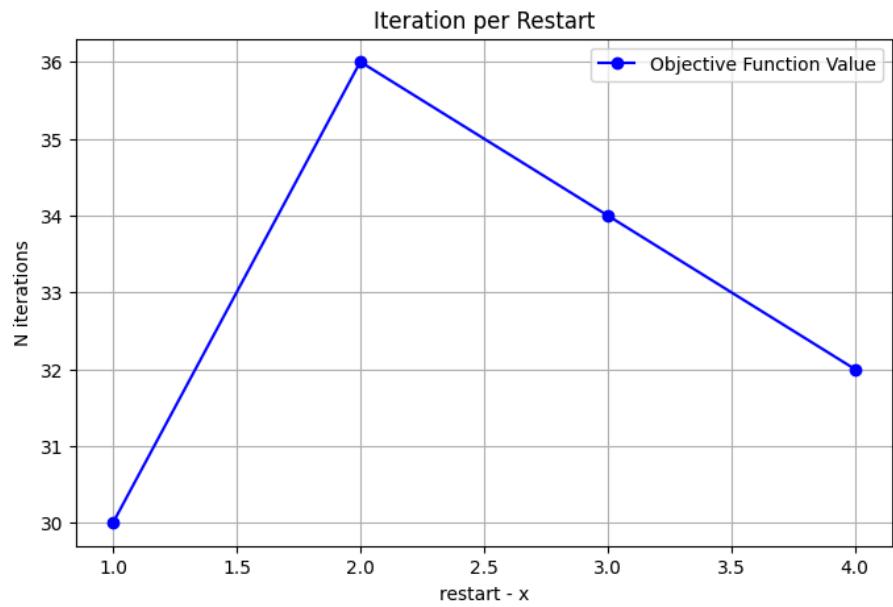
Plot Nilai Objective Function:





Banyak Restart: 3

Banyak Iterasi per Restart:



2.2.3.2. Percobaan 2

State Awal

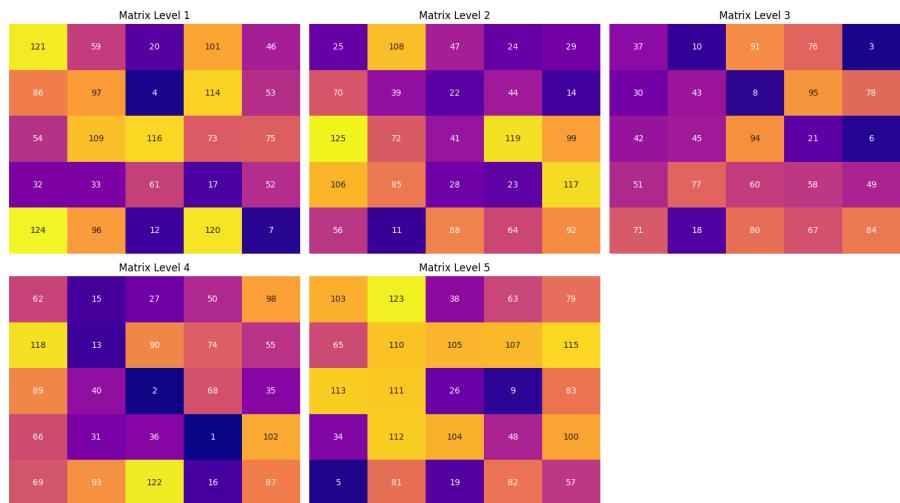


Initial state restart - 1

Fitness Value: 0

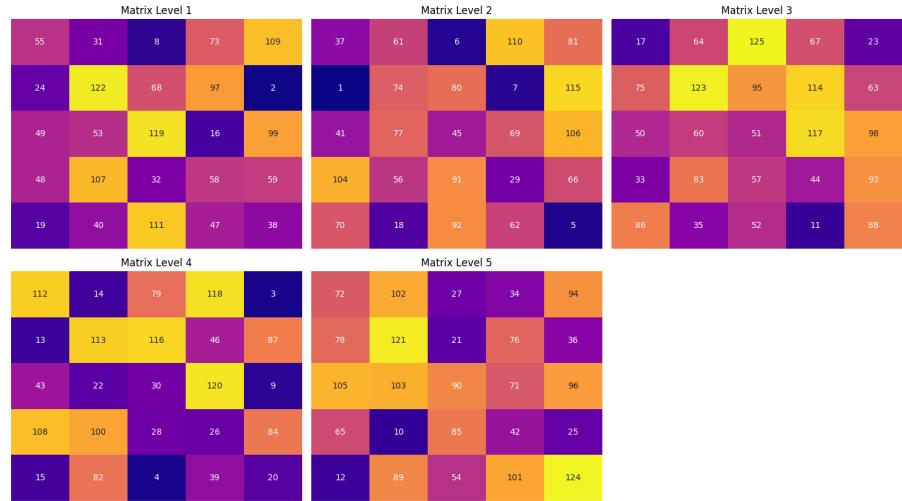
**Initial state restart - 2**

Fitness Value: 3



Initial state restart - 3

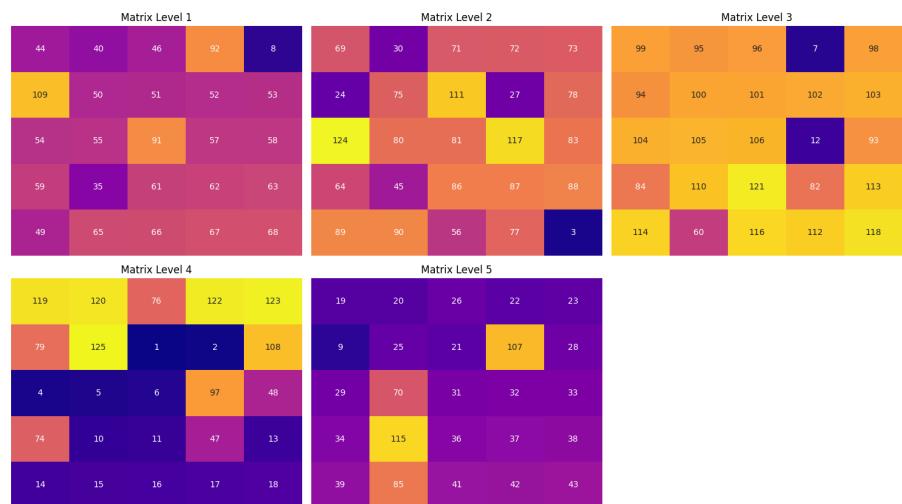
Fitness Value: 1



State Akhir

Final state restart - 0

Fitness Value: 43

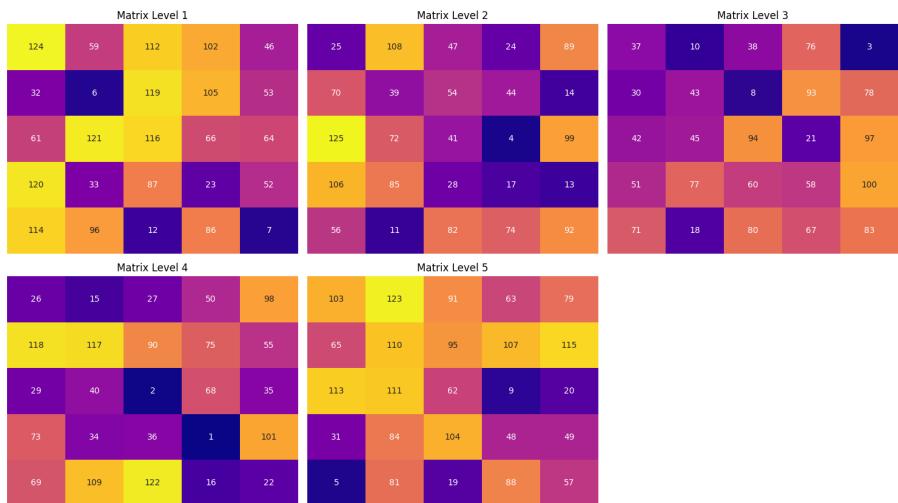


Final state restart - 1

Fitness Value: 39

**Final state restart - 2**

Fitness Value: 35



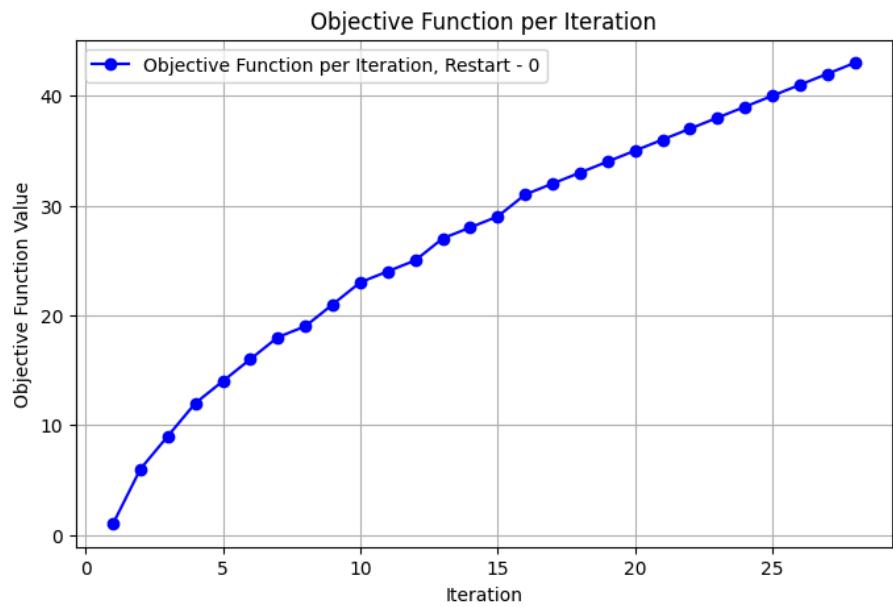
Final state restart - 3

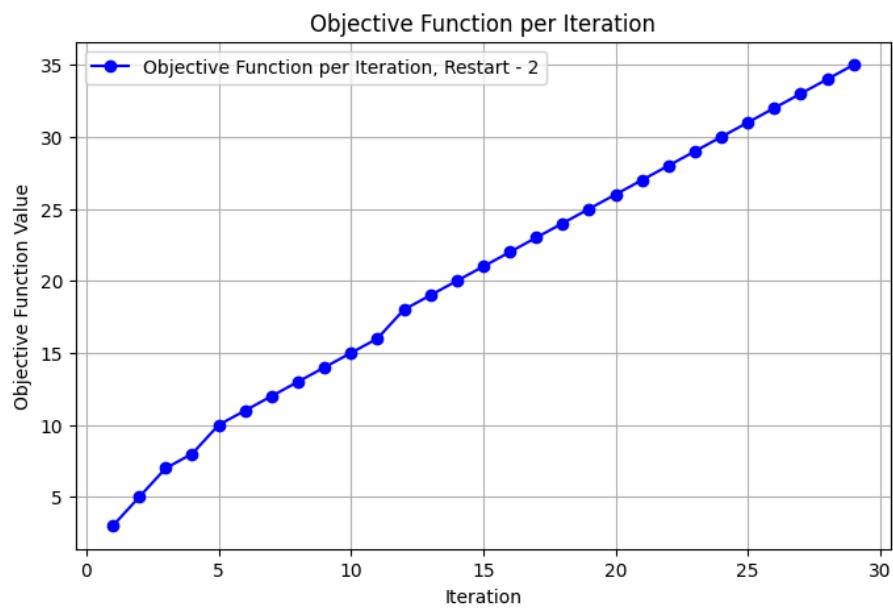
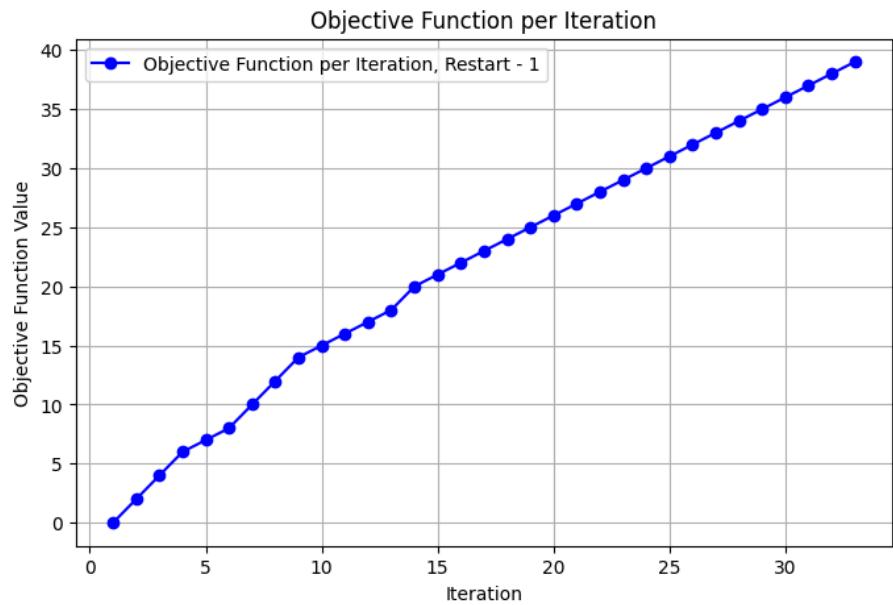
Fitness Value: 38

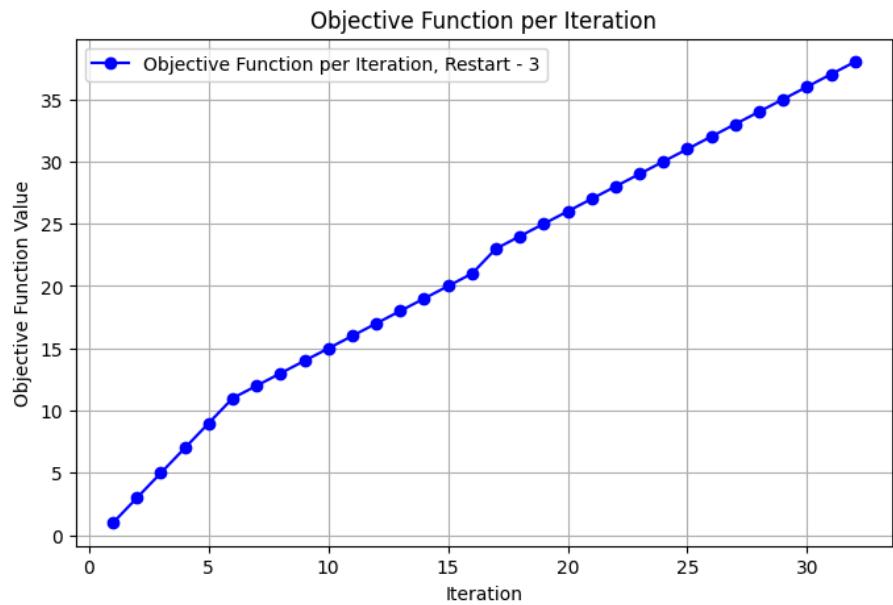
Matrix Level 1					Matrix Level 2					Matrix Level 3				
94	31	8	73	109	39	50	6	110	112	17	15	125	95	16
105	122	68	18	2	78	35	80	7	115	75	123	30	24	63
49	53	96	119	99	67	77	37	107	106	104	60	51	117	98
48	69	54	58	59	61	56	91	29	66	33	83	57	1	93
19	40	111	47	38	70	97	101	62	88	86	121	52	11	45
Matrix Level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

Nilai Objective Function Terbesar: 43

Plot Nilai Objective Function:

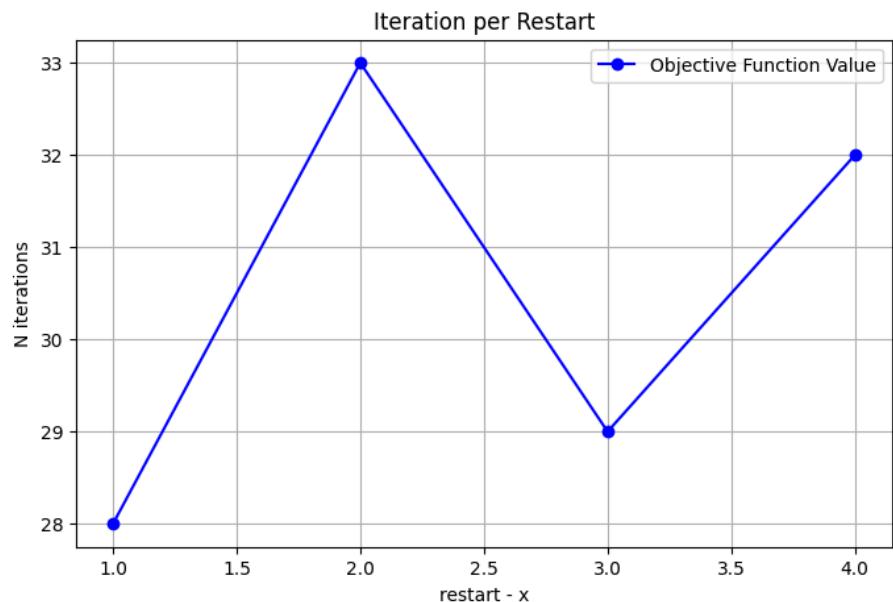






Banyak Restart: 3

Banyak Iterasi per Restart:

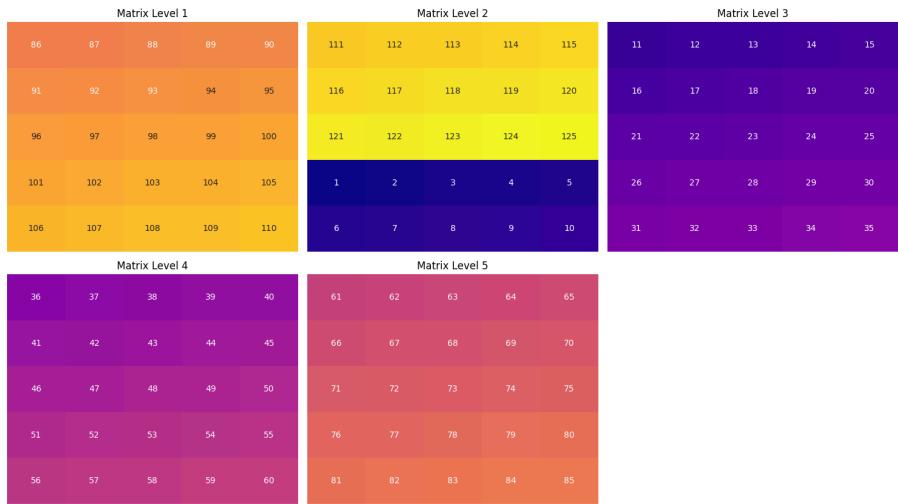


2.2.3.3. Percobaan 3

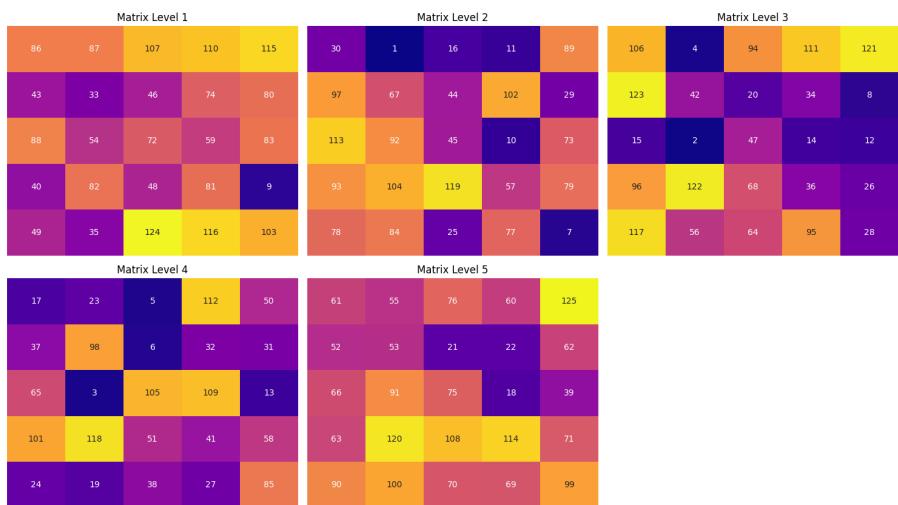
State Awal

Initial state restart - 0

Fitness Value: 4

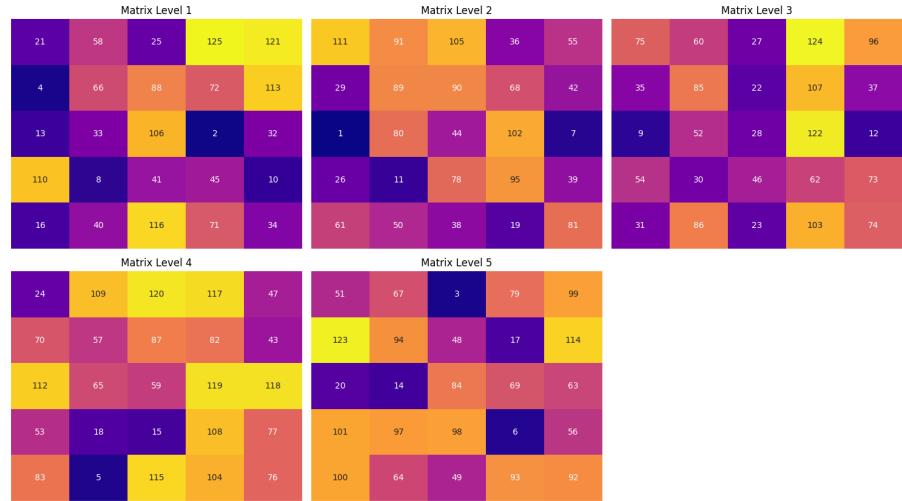
**Initial state restart - 1**

Fitness Value: 1

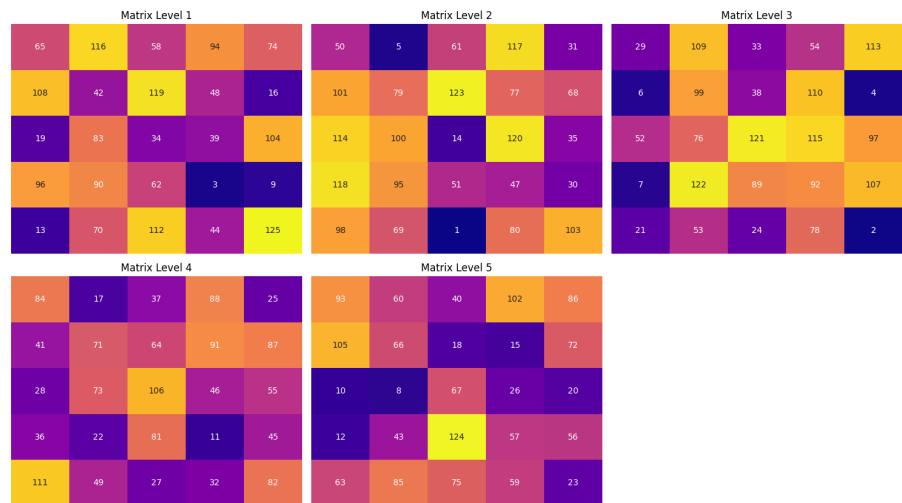


Initial state restart - 2

Fitness Value: 1

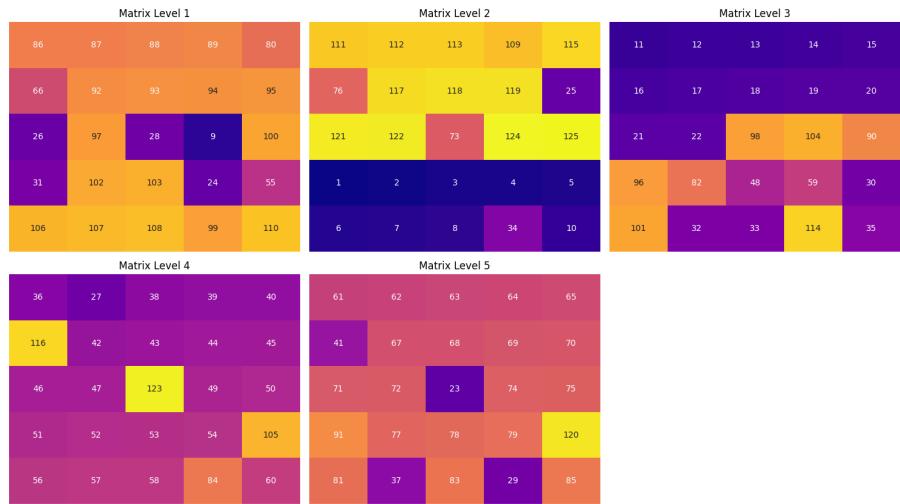
**Initial state restart - 3**

Fitness Value: 2

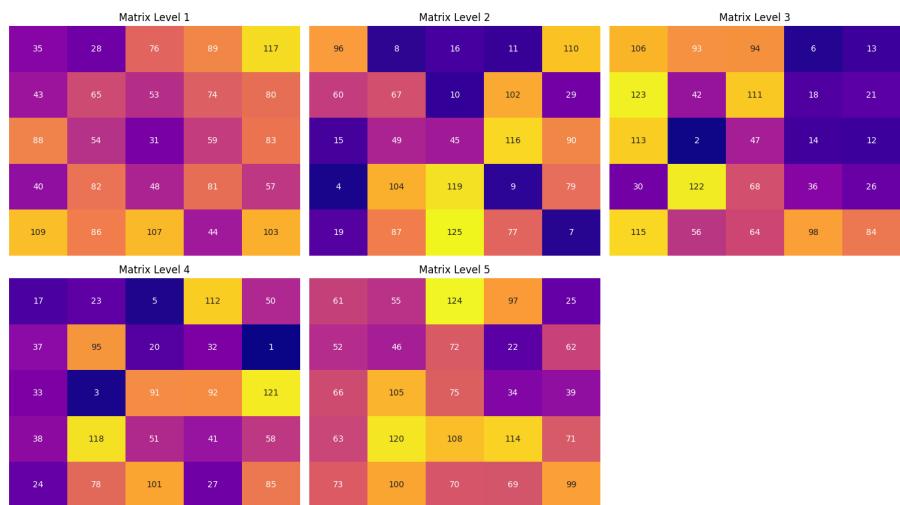
**State Akhir**

Initial state restart - 0

Fitness Value: 43

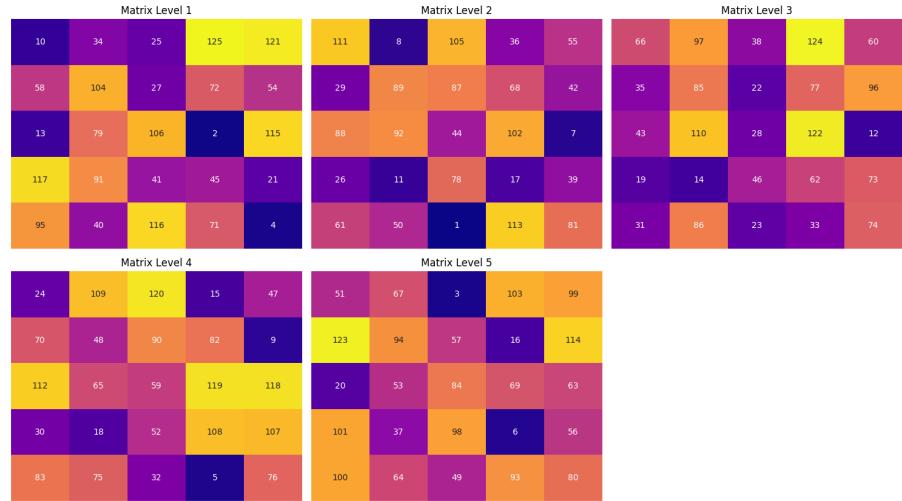
**Initial state restart - 1**

Fitness Value: 39

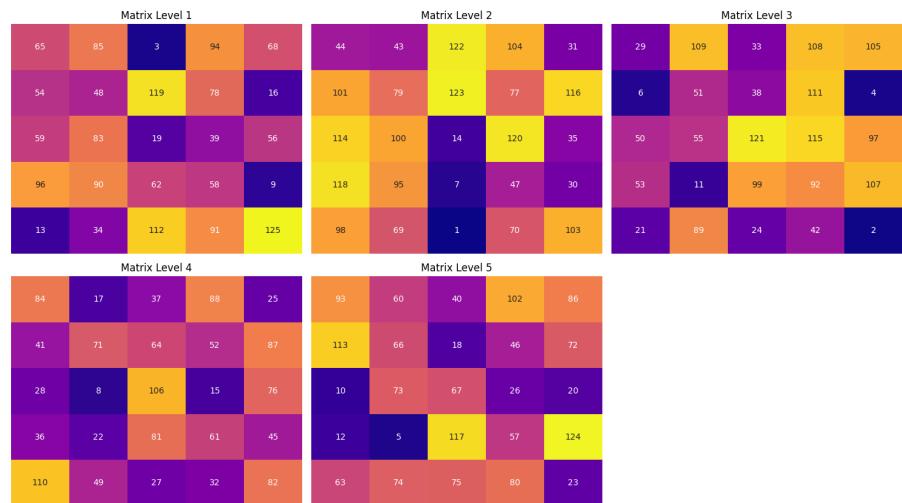


Initial state restart - 2

Fitness Value: 37

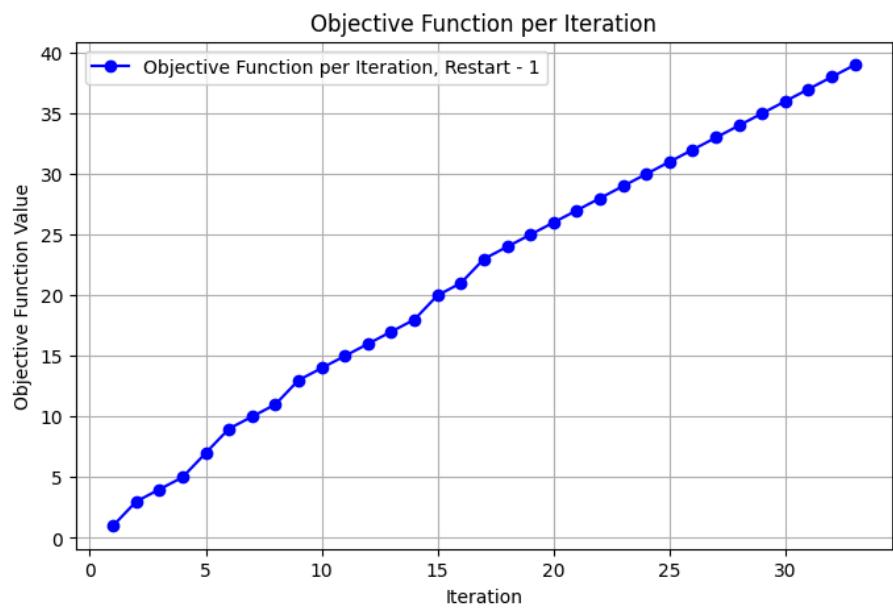
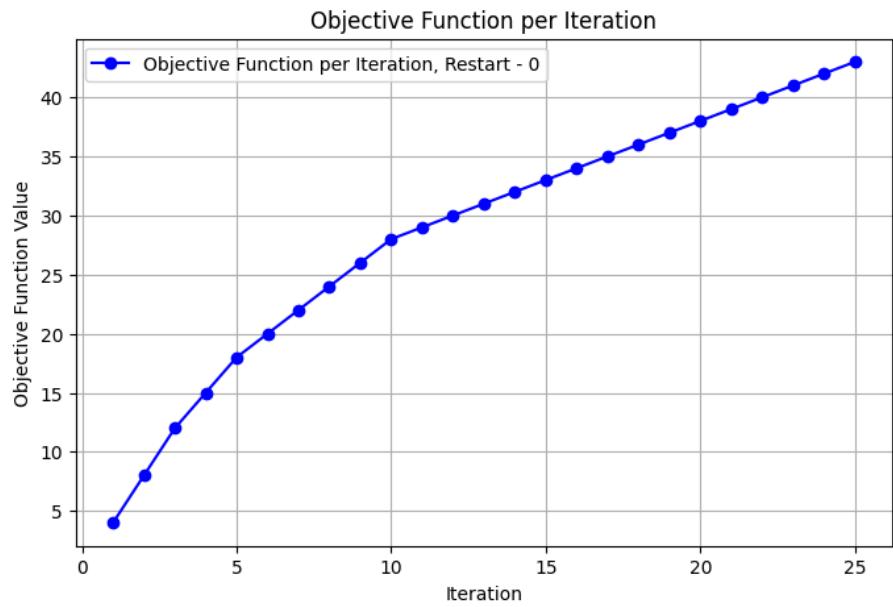
**Initial state restart - 3**

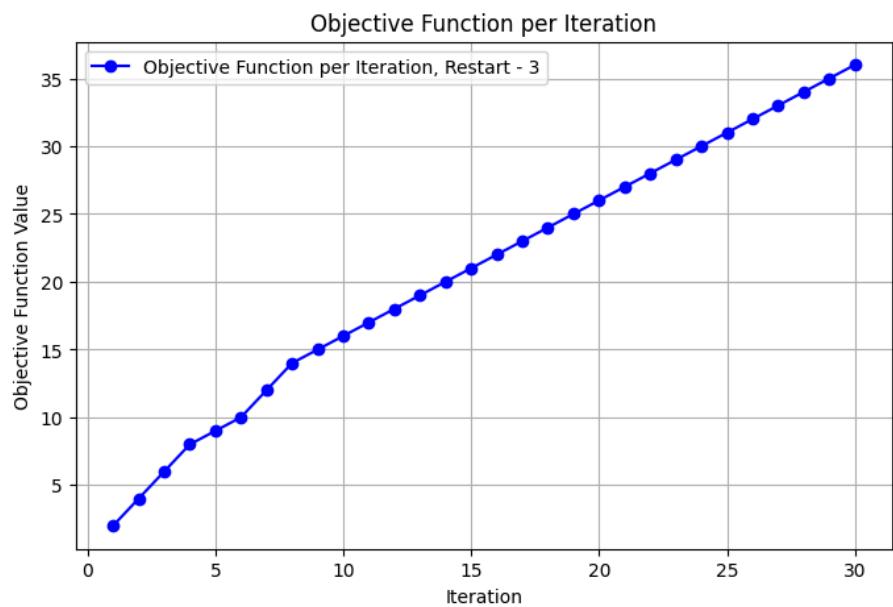
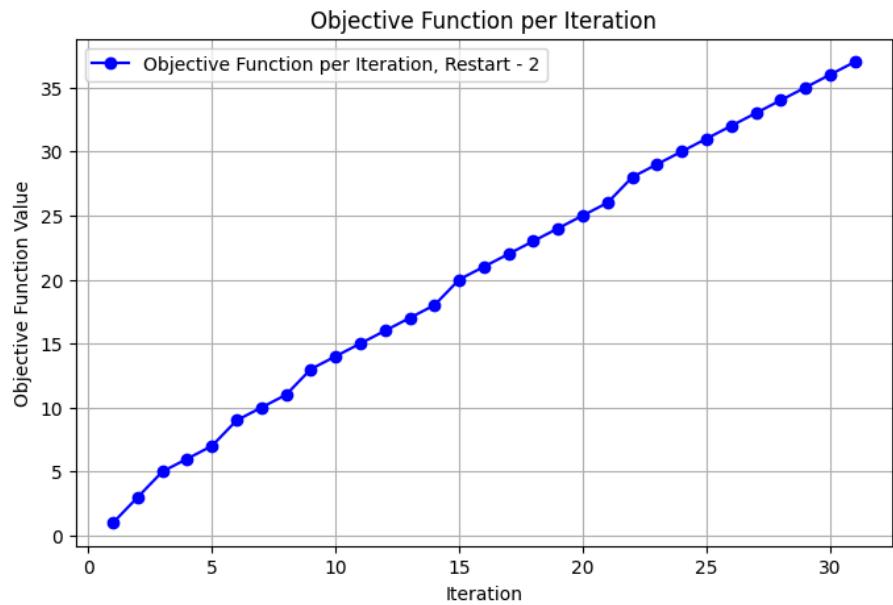
Fitness Value: 36



Nilai Objective Function: 43

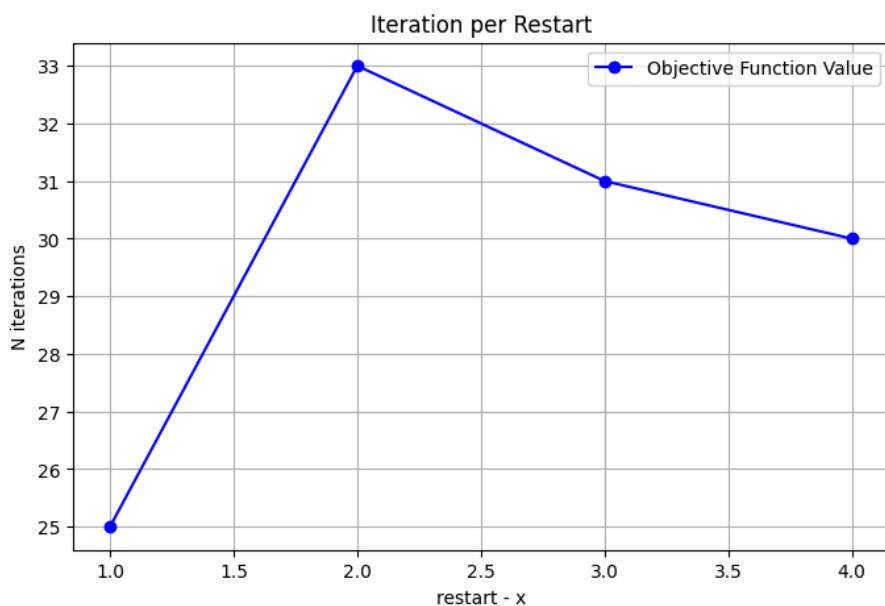
Plot Nilai Objective Function:





Banyak Restart: 3

Banyak Iterasi per Restart:



2.2.4. Stochastic Hill-climbing

Langkah-langkah pada algoritma ini:

1. Membangkitkan cube random untuk dilakukan local search. Diinisiasi juga iterasi untuk menghitung banyaknya iterasi.
2. Mulai loop, bangkitkan cube baru yang merupakan successor random dari state sekarang
3. Jika nilai state dari neighbour tersebut lebih bagus dari state sekarang maka pindah ke state tersebut, jika tidak maka diam di state sekarang
4. Lakukan loop selama iterasi belum melebihi max iterasi
5. Setelah loop selesai, maka mengembalikan state akhir, value dari state akhir, dan banyak iterasi yang dilakukan

Source Code

```
def StochasticHillClimbingCube(init_cube, max_iter):
    current_cube = Cube(5, 5, 5, False, init_cube)
    current_value = current_cube.state_value
    iter = 0

    values = [current_value]
    cubes = [current_cube]

    while iter <= max_iter:
        iter += 1
        random_neighbor = current_cube.copy()
```

```
    z1, x1, y1 = random.randint(0, 4), random.randint(0, 4),
random.randint(0, 4)
    z2, x2, y2 = random.randint(0, 4), random.randint(0, 4),
random.randint(0, 4)
    while (z1, x1, y1) == (z2, x2, y2):
        z2, x2, y2 = random.randint(0, 4), random.randint(0, 4),
random.randint(0, 4)

        temp = random_neighbor.get_value(z1, x1, y1)
        random_neighbor.insert_value(z1, x1, y1,
random_neighbor.get_value(z2, x2, y2))
        random_neighbor.insert_value(z2, x2, y2, temp)
        random_neighbor.calculate_state_value()

        random_value = random_neighbor.state_value

        if random_value > current_value:
            current_cube = random_neighbor
            current_value = random_value
        print("Val:", current_value)

        values.append(current_value)
        cubes.append(current_cube)

    return cubes, values, iter
```

2.2.4.1. Percobaan 1

State Awal

3D Cube Matrix Layers

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

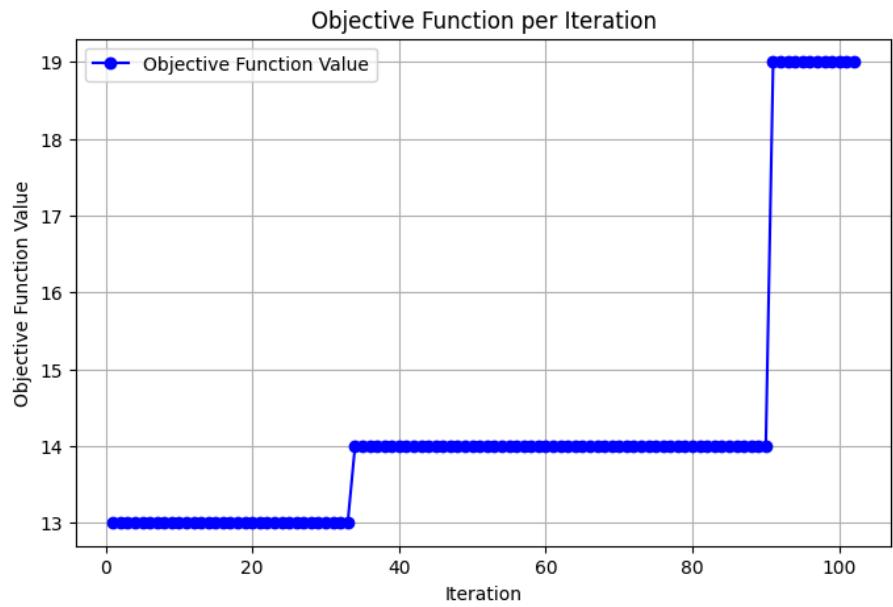
State Akhir

3D Cube Matrix Layers

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

Nilai Objective Function: 19

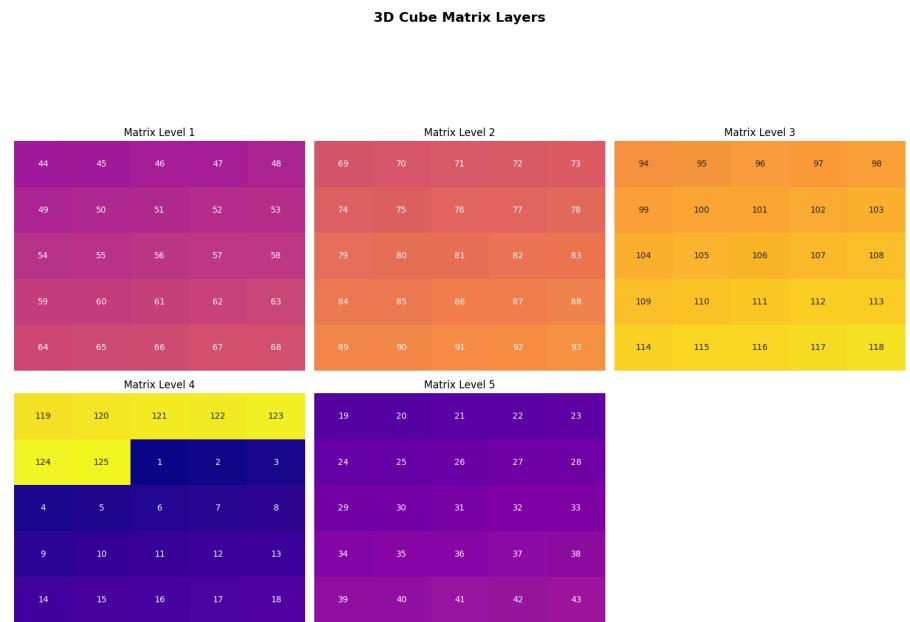
Plot Nilai Objective Function:



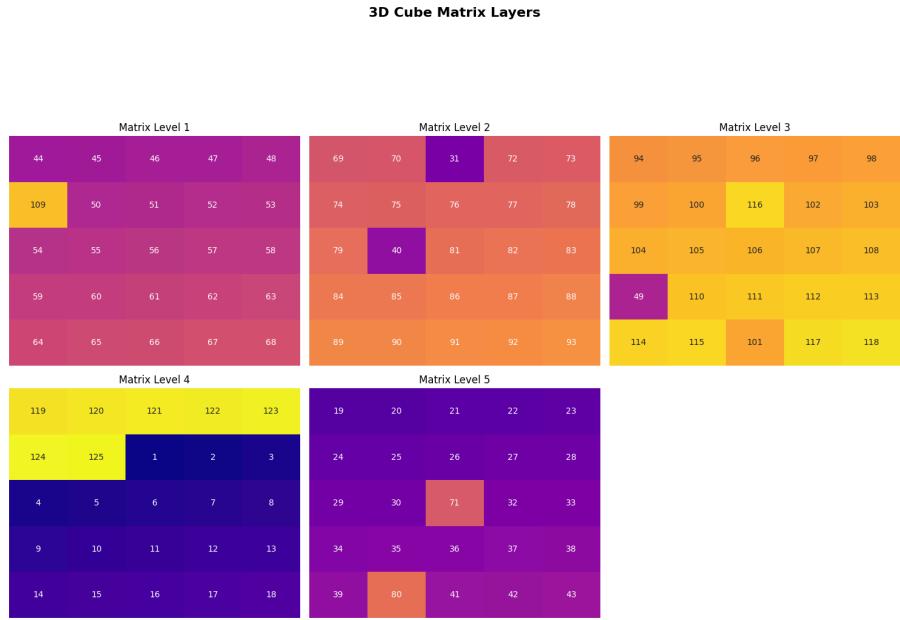
Banyak Iterasi: 101

2.2.4.2. Percobaan 2

State Awal

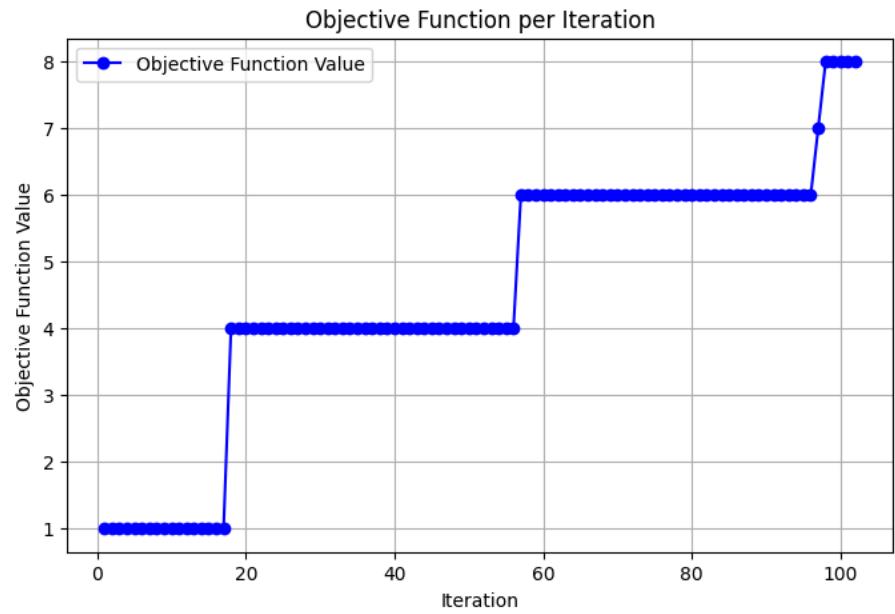


State Akhir



Nilai Objective Function: 8

Plot Nilai Objective Function:

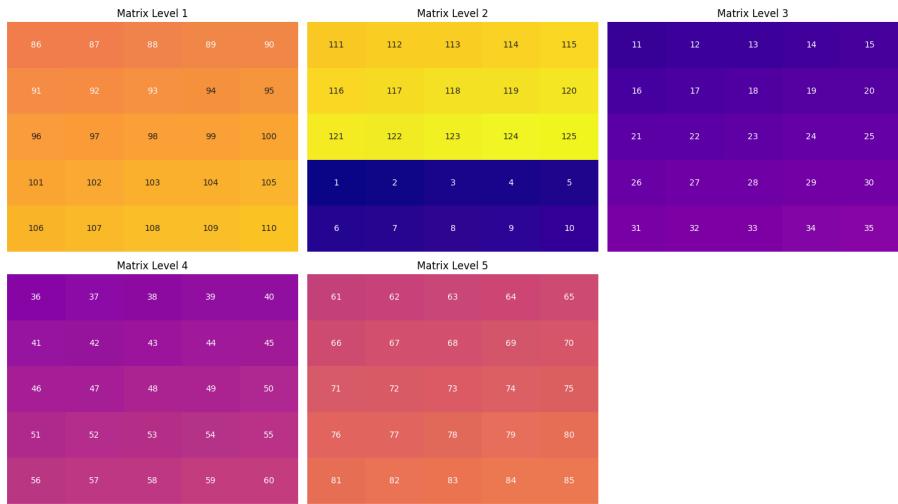


Banyak Iterasi: 101

2.2.4.3. Percobaan 3

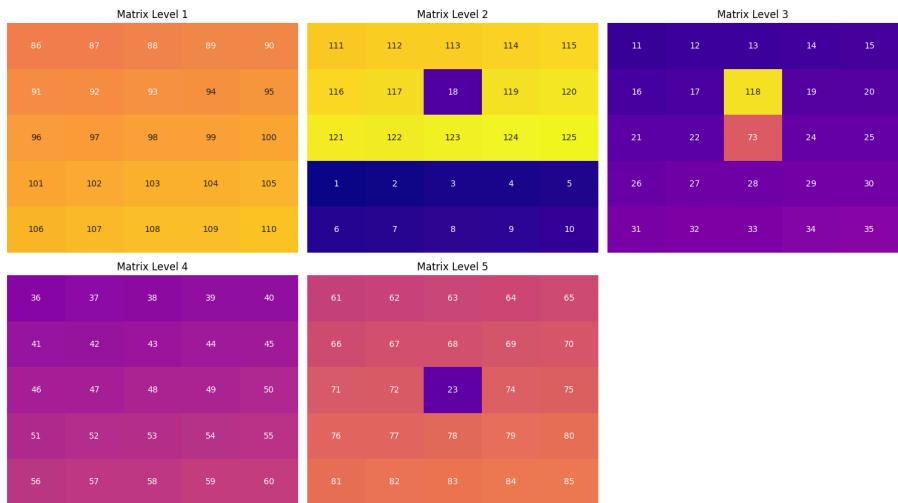
State Awal

3D Cube Matrix Layers



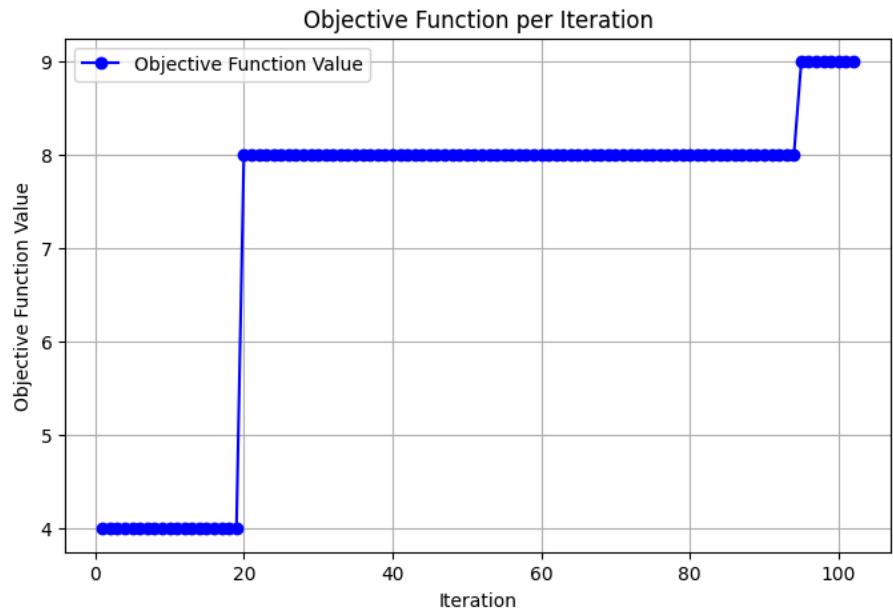
State Akhir

3D Cube Matrix Layers



Nilai Objective Function: 4

Plot Nilai Objective Function:



Banyak Iterasi: 101

2.2.5. Simulated Annealing

Langkah-langkah pada algoritma ini :

1. Membangkitkan cube random untuk dilakukan local search. Diinisiasi juga nilai iterasi, count stuck
2. Lakukan loop dimana pertama dihitung dulu nilai T berdasarkan fungsi schedule
3. Jika T sama dengan 0 maka return
4. Increment jumlah iterasi dan bangkitkan random successor dari current state
5. Hitung nilai deltaE yang merupakan selisih dari neighbour state value dan current state value
6. Jika deltaE positif maka langsung pindah ke state tersebut, jika negatif maka hitung probabilitas menggunakan rumus $e^{\frac{\Delta E}{T}}$, kemudian menggunakan probabilitas tersebut tentukan apakah berpindah atau tidak
7. Fungsi mengembalikan semua cubes yang ditelusuri, semua value yang ditemukan, banyaknya iterasi, semua probabilitas e yang dihitung, dan banyaknya stuck

Source Code

```
def schedule(t, initial_temperature, cooling_rate):  
    return max(initial_temperature * math.pow(cooling_rate, t), 0)
```

```

def SimulatedAnnealingCube(init_cube, initial_temperature=1000,
cooling_rate=0.9, schedule=schedule):
    current_cube = Cube(5, 5, 5, False, init_cube)
    current_value = current_cube.state_value
    t = 1

    cubes = [current_cube]
    values = [current_value]
    count_iter = 0
    e_probs = {}
    count_stuck = 0

    while True:
        T = schedule(t, initial_temperature, cooling_rate)
        if T == 0:
            break
        count_iter += 1

        random_neighbor = current_cube.copy()

        a, b, c = random.randint(0, 4), random.randint(0, 4),
random.randint(0, 4)
        i, j, k = random.randint(0, 4), random.randint(0, 4),
random.randint(0, 4)
        while (a, b, c) == (i, j, k):
            i, j, k = random.randint(0, 4), random.randint(0, 4),
random.randint(0, 4)

        temp = random_neighbor.get_value(a, b, c)
        random_neighbor.insert_value(a, b, c,
random_neighbor.get_value(i, j, k))
        random_neighbor.insert_value(i, j, k, temp)
        random_neighbor.calculate_state_value()

        random_value = random_neighbor.state_value
        deltaE = random_value - current_value

        if deltaE > 0:
            current_cube = random_neighbor
            current_value = random_value
        else:
            if deltaE < 0:
                e_probs[count_iter] = math.exp(deltaE / T)
                if random.random() < math.exp(deltaE / T):

```

```

    count_stuck += 1
    current_cube = random_neighbor
    current_value = random_value

    cubes.append(current_cube)
    values.append(current_value)
    t += 1

return cubes, values, count_iter, e_probs, count_stuck

```

2.2.5.1. Percobaan 1

State Awal

3D Cube Matrix Layers

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

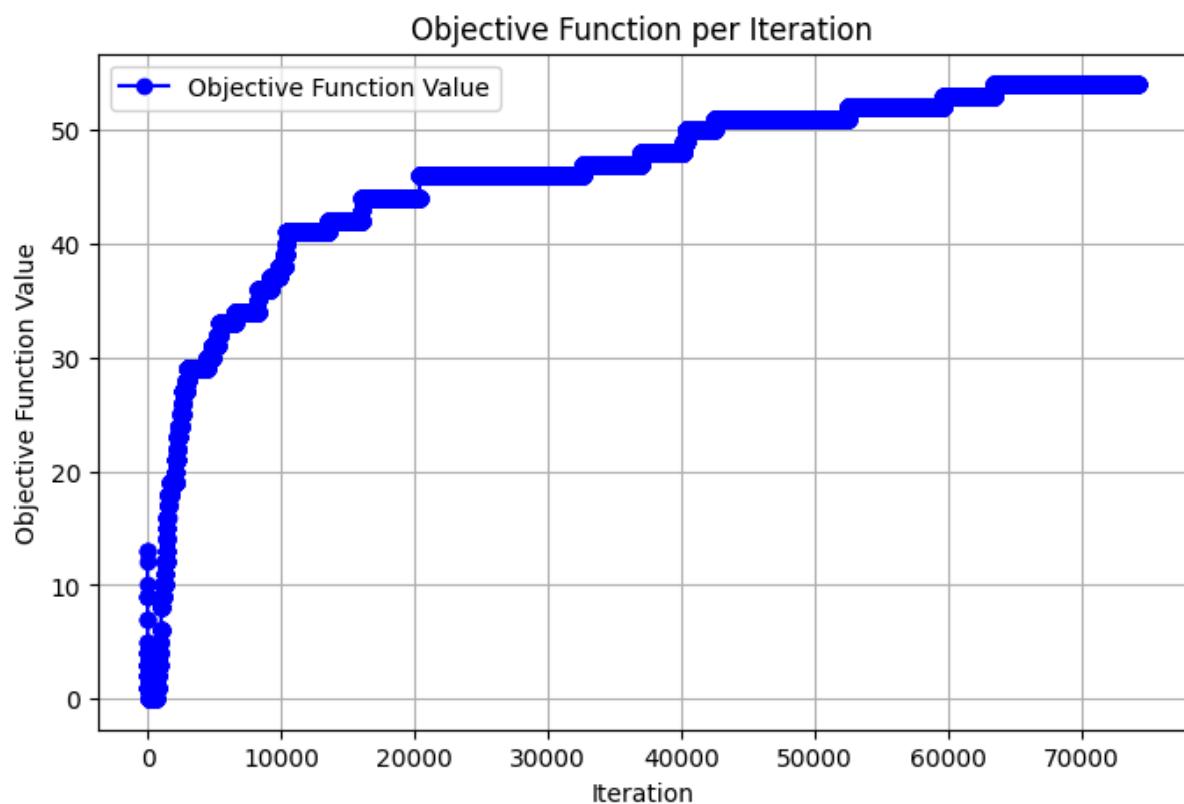
State Akhir

3D Cube Matrix Layers

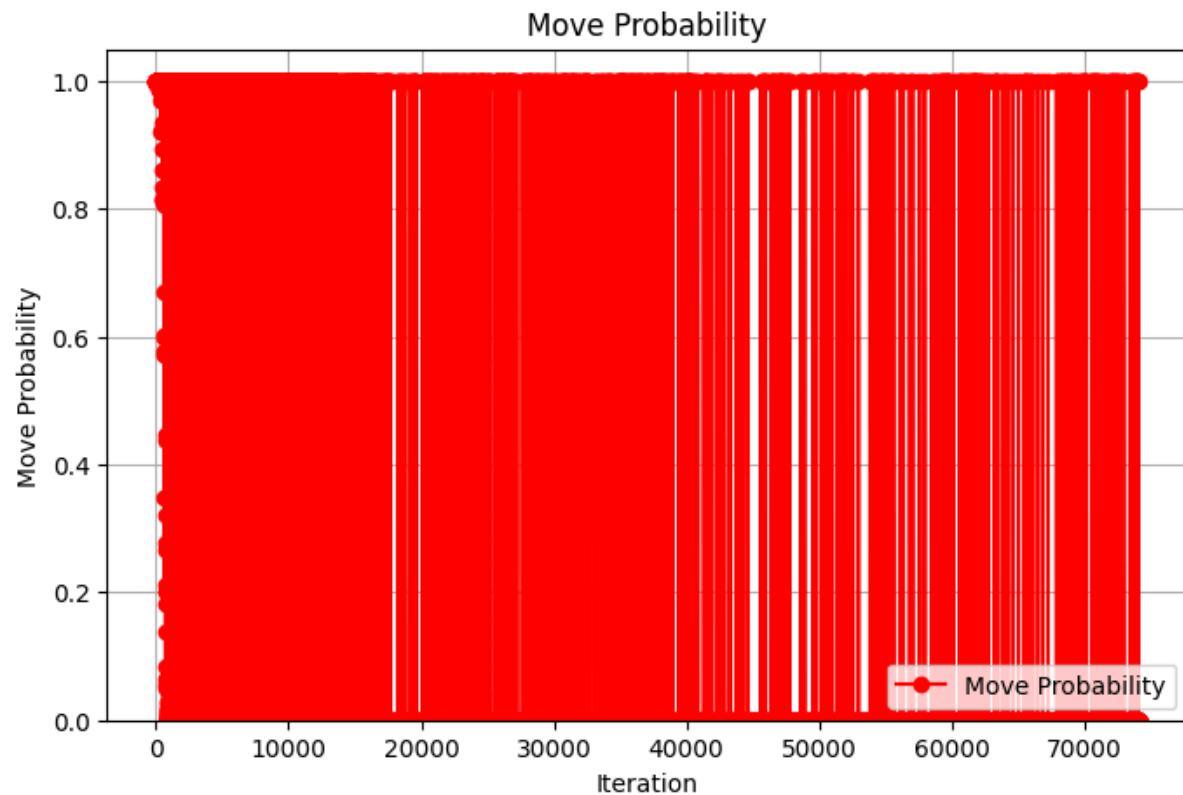
Matrix Level 1					Matrix Level 2					Matrix Level 3				
81	28	73	107	26	120	10	95	76	14	20	119	24	63	89
122	67	6	112	8	105	5	53	58	94	116	85	59	29	13
57	16	51	99	92	70	31	15	74	125	34	65	106	108	2
12	83	72	55	93	109	39	104	9	54	1	4	86	18	68
43	121	46	44	61	88	47	48	71	118	87	42	40	97	49
Matrix Level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

Nilai Objective Function: 54

Plot Nilai Objective Function:



Plot $e^{\frac{\Delta E}{T}}$:



Frekuensi Stuck: 2460

2.2.5.2. Percobaan 2

State Awal

3D Cube Matrix Layers

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

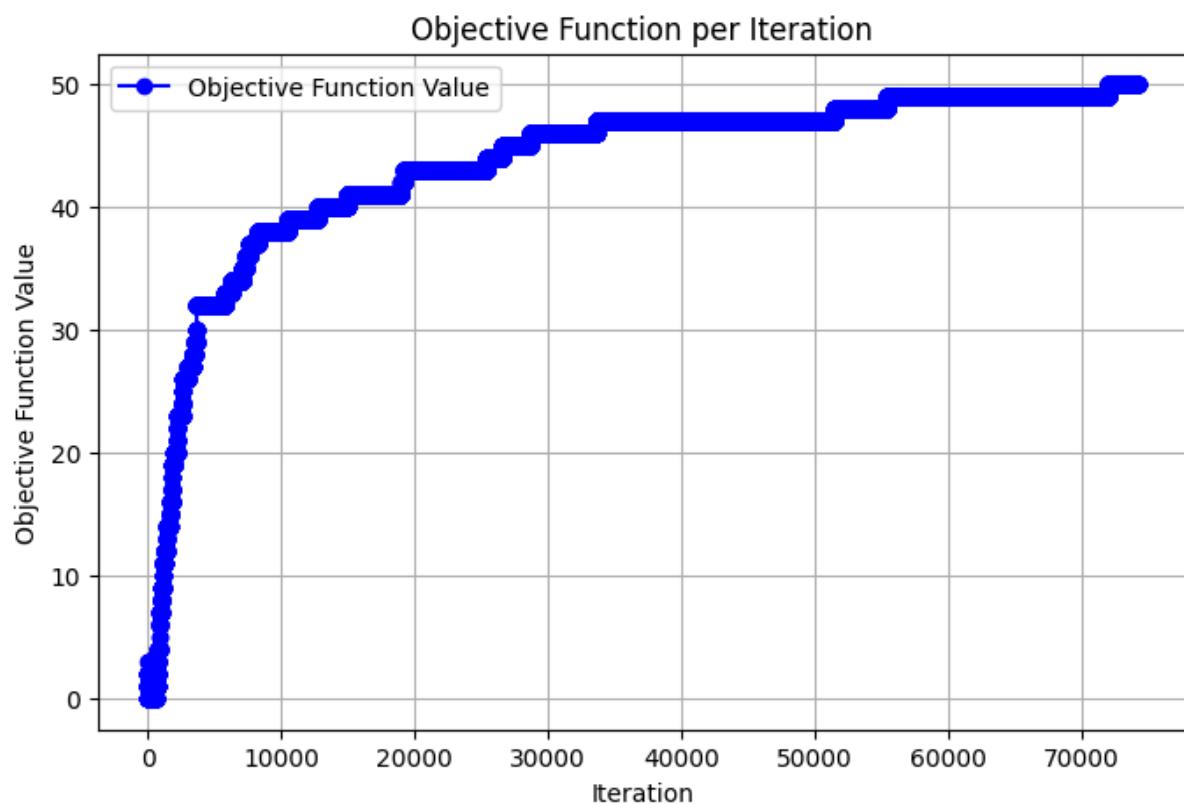
State Akhir

3D Cube Matrix Layers

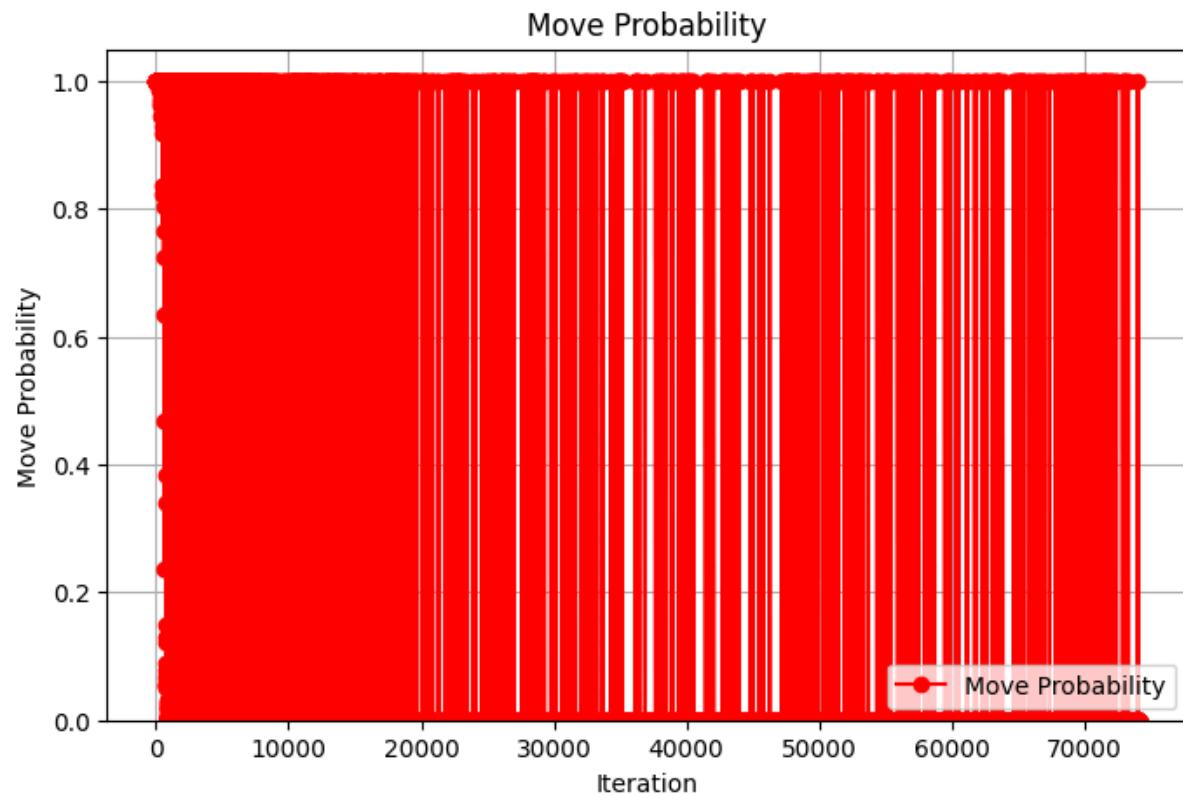
Matrix Level 1					Matrix Level 2					Matrix Level 3				
47	90	112	42	24	61	120	35	60	39	89	51	97	29	49
81	58	115	62	122	6	105	1	50	34	59	107	43	101	5
79	111	83	20	22	30	15	13	102	103	92	8	68	38	109
87	125	110	19	28	104	53	10	75	73	4	82	48	116	65
21	37	86	52	119	114	17	26	27	66	71	67	123	31	23
Matrix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

Nilai Objective Function: 50

Plot Nilai Objective Function:



Plot $e^{\frac{\Delta E}{T}}$:



Frekuensi Stuck: 2012

2.2.5.3. Percobaan 3

State Awal

3D Cube Matrix Layers

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

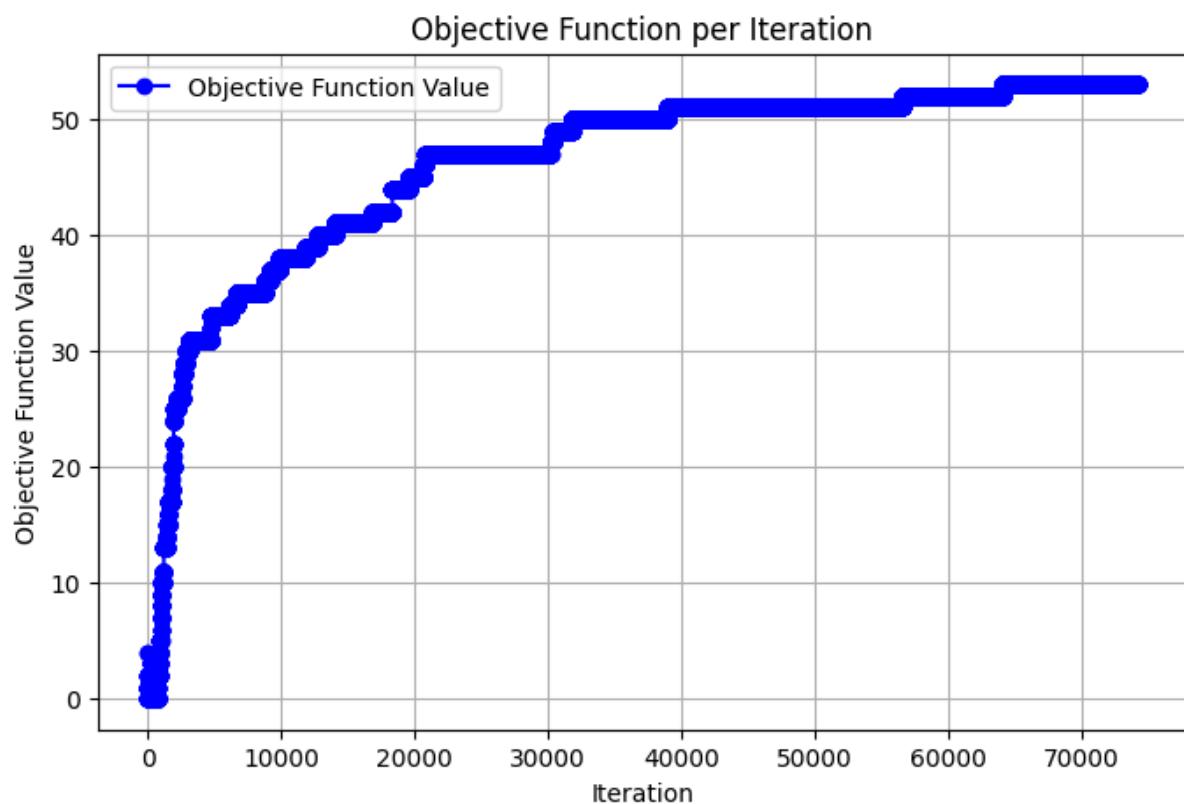
State Akhir

3D Cube Matrix Layers

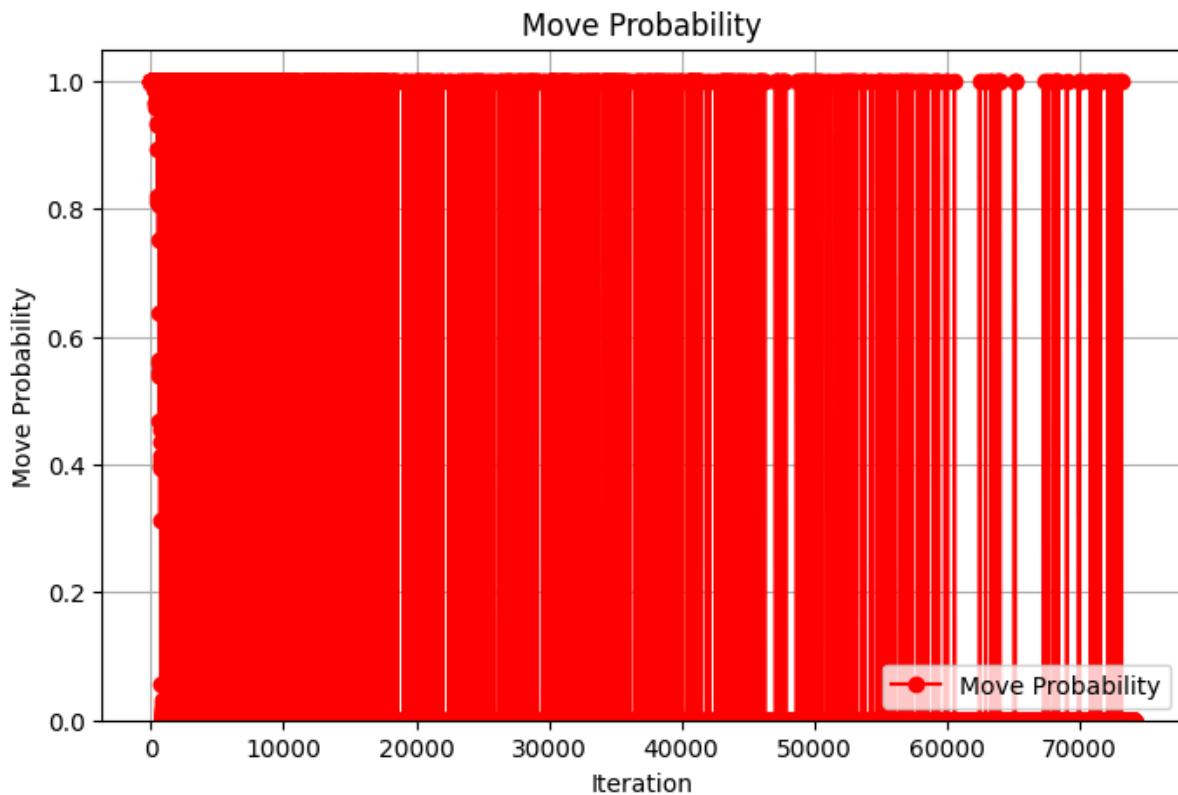
Matrix Level 1					Matrix Level 2					Matrix Level 3				
118	89	59	65	5	9	15	124	38	97	19	52	3	78	37
83	46	75	47	64	1	41	62	103	108	71	53	81	119	94
31	44	84	114	42	26	123	91	8	122	82	36	101	68	28
43	102	56	54	60	116	70	17	57	96	88	67	112	20	73
40	34	74	35	13	115	66	21	109	4	55	107	18	30	105
Matrix Level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

Nilai Objective Function: 53

Plot Nilai Objective Function:



Plot $e^{\frac{\Delta E}{T}}$:



Frekuensi Stuck: 1134

2.2.6. Genetic Algorithm

Langkah-langkah pada algoritma ini :

1. Menghasilkan populasi awal sebanyak N individu (*cube*).
2. Dari antara individu dalam populasi tersebut, dicari *current_max_fit* yakni individu (dalam contoh ini *cube*) yang memiliki nilai fitness terbesar
3. Setelah itu, dimulainya proses pembentukan populasi baru.
4. Pembentukan populasi baru dimulai dengan mengisi sebanyak *elite_size* individu yang memiliki fitness value terbesar.
5. Setelah itu dimulainya proses *roulette wheel selection* dari antara populasi awal untuk memilih *parents* yang akan digunakan pada proses *crossover*.
6. Setelah terpilih dua individu sebagai *parents*, dilakukan proses *crossover* (terpilih parent1 dan parent2).
7. Proses *crossover* dilakukan dengan metode *Ordered Crossover*, metode ini dimulai dengan memilih dua index yang akan menjadi *start* dan *end* area terpotong antara parent1 dan parent2 yang kemudian kedua area tersebut akan ditukar satu sama lain menghasilkan child1 dan child2.
8. Area kosong pada child1 dan child2 kemudian diisi dengan nilai - nilai pada salah satu parent (parent yang tidak memberikan potongan areanya).

9. child1 dan child2 kemudian melalui proses *mutation*, proses ini terjadi berdasarkan probabilitas tertentu.
10. Proses *mutation* dilakukan dengan melakukan *swap* antara dua element dalam *magic cube*.
11. Setelah melakukan proses *mutation*, dipilihnya antara child1 dan child2 sebagai hasil keturunan, child yang dipilih berdasarkan nilai fitness terbesar.
12. Langkah 5 sampai 11 dilakukan ulang sampai populasi baru (*new_population*) memiliki ukuran sebesar N.
13. Setelah mendapatkan *new_population* maka dipilihnya *current_max_fit* yakni individu (*cube*) yang memiliki *fitness value* terbesar dalam *new_population*.
14. *Population* kemudian diganti dengan *new_population*.
15. Langkah 3 - 14 dilakukan ulang sebanyak *max_genexrations*.

Source code

```

import random
import heapq
import multiprocessing as mp
import Cube as c

def mutate(child, mutation_rate=0.01):
    size = len(child.array)
    if random.random() < mutation_rate:
        i, j = random.sample(range(size), 2)
        child.array[i], child.array[j] = child.array[j],
        child.array[i]
    child.calculate_state_value()
    return child

def crossover(parent1, parent2):
    size = len(parent1.array)
    child1_array = [None] * size
    child2_array = [None] * size
    start, end = sorted(random.sample(range(size), 2))

    child1_array[start : end+1] = parent1.array[start : end+1]
    child2_array[start : end+1] = parent2.array[start : end+1]

    used1 = set(child1_array[start : end+1])
    used2 = set(child2_array[start : end+1])

    # Create remaining values lists

```

```

remaining1 = [x for x in parent2.array if x not in used1]
remaining2 = [x for x in parent1.array if x not in used2]

idx1 = idx2 = 0
for i in range(size):
    if i < start or i > end:
        child1_array[i] = remaining1[idx1]
        child2_array[i] = remaining2[idx2]
        idx1 += 1
        idx2 += 1

child1 = c.Cube(parent1.x_size, parent1.y_size, parent1.z_size,
array=child1_array)
child2 = c.Cube(parent2.x_size, parent2.y_size, parent2.z_size,
array=child2_array)

return child1, child2

def roulette_wheel_selection(population):
    total_fitness = sum(ind.state_value for ind in population)
    pick = random.uniform(0, total_fitness)
    current = 0
    for individual in population:
        current += individual.state_value
        if current > pick:
            return individual

def create_child_task(args):
    parent1, parent2, mutation_rate = args
    child1, child2 = crossover(parent1, parent2)
    child1 = mutate(child1, mutation_rate)
    child2 = mutate(child2, mutation_rate)
    return child1 if child1.state_value > child2.state_value else
child2

def genetic(N=100, max_generations=1000, initial_mutation_rate=0.05,
stagnation_limit=100):
    population = [c.Cube(5, 5, 5, True) for _ in range(N)]
    current_max_fit = max(population, key=lambda x: x.state_value)
    values = [current_max_fit.state_value]
    no_improvement_count = 0
    mutation_rate = initial_mutation_rate
    elite_size = int(0.05 * N)
    cubes = []

```

```

        with mp.Pool(mp.cpu_count()) as pool:
            for generation in range(max_generations):
                new_population = heapq.nlargest(elite_size, population,
key=lambda x: x.state_value)

                    mutation_rate = min(0.3, mutation_rate * 1.2) if
no_improvement_count > stagnation_limit / 2 else
initial_mutation_rate

                tasks = [
                    roulette_wheel_selection(population),
roulette_wheel_selection(population, mutation_rate)
                    for _ in range((N - elite_size) // 2)
                ]

                children = pool.map(create_child_task, tasks)
new_population.extend(children)

                population = heapq.nlargest(N, new_population, key=lambda
x: x.state_value)
                best_child = population[0]

                if best_child.state_value > current_max_fit.state_value:
                    current_max_fit = best_child
                    no_improvement_count = 0
                else:
                    no_improvement_count += 1

                if(generation%100 == 0):
                    print(f"Generation {generation}: Current max fitness
= {current_max_fit.state_value}")

                if no_improvement_count >= stagnation_limit:
                    print("STAGNATION ALERT !!!")
                    random_individuals = [c.Cube(5, 5, 5, True) for _ in
range(int(0.5 * N))]
                    population = population[:int(0.5 * N)] +
random_individuals
                    no_improvement_count = 0

                values.append(current_max_fit.state_value)
                cubes.append(current_max_fit)

```

```

        print("Final solution:")
        current_max_fit.print_cube()

    return current_max_fit, values, cubes, generation + 1

```

2.2.6.1. Percobaan 1

State Awal

3D Cube Matrix Layers

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

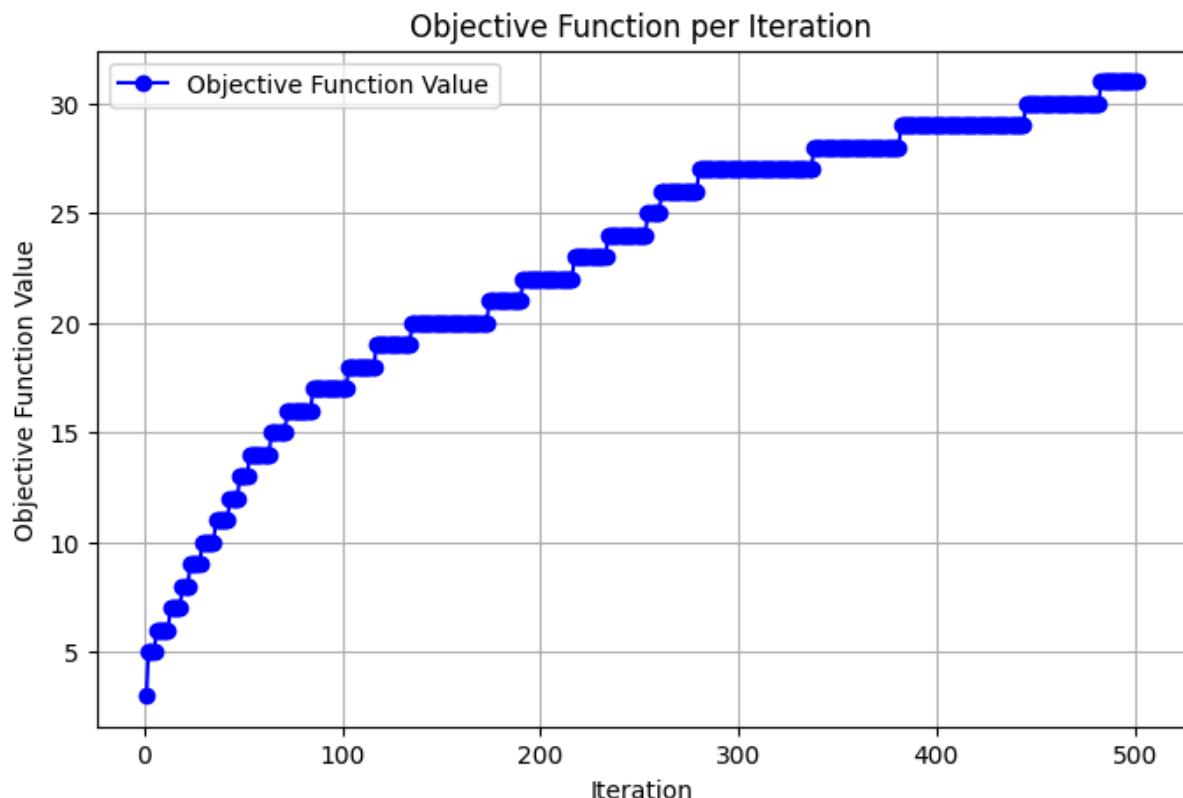
State Akhir

3D Cube Matrix Layers

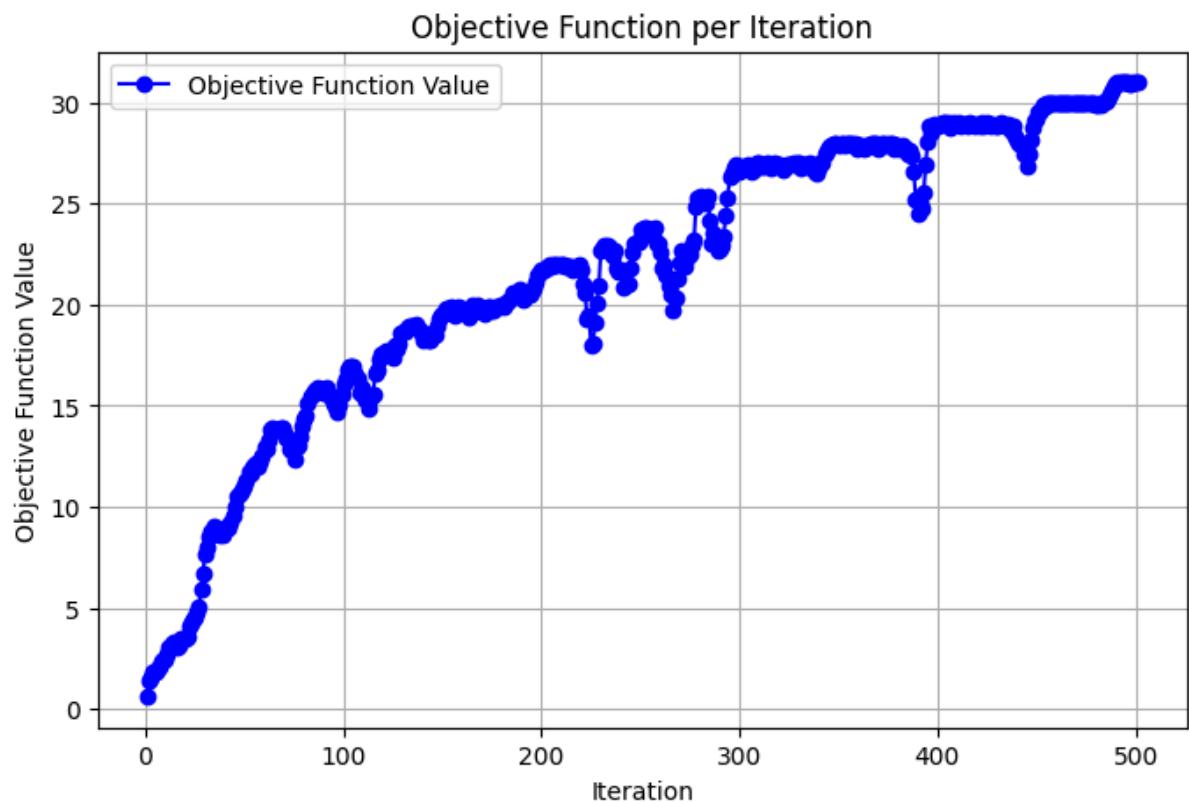
Matrix Level 1					Matrix Level 2					Matrix Level 3				
41	8	1	33	106	58	87	113	52	105	91	99	31	53	122
12	70	49	10	80	7	96	6	83	20	109	13	78	40	76
66	73	117	22	59	112	24	82	114	72	30	43	28	124	115
62	55	67	84	47	102	71	75	111	118	97	36	15	121	17
27	74	81	119	3	98	37	39	93	48	89	21	32	65	108
Matrix Level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

Nilai Objective Function: 31

Plot Nilai Objective Function Maximum:



Plot Nilai Objective Function Rata-Rata:



Jumlah Populasi: 200

Banyak Iterasi: 500

Durasi Proses Pencarian: 12 seconds

2.2.6.2. Percobaan 2

State Awal

3D Cube Matrix Layers

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

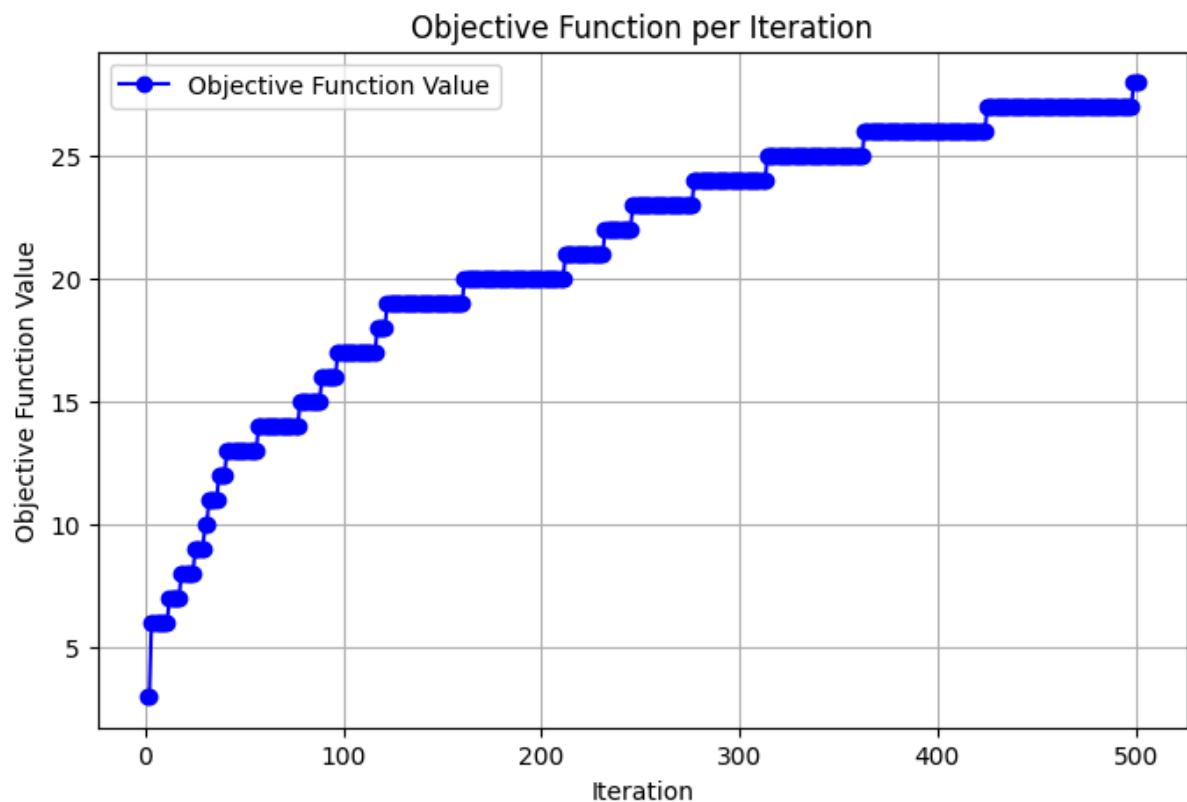
State Akhir

3D Cube Matrix Layers

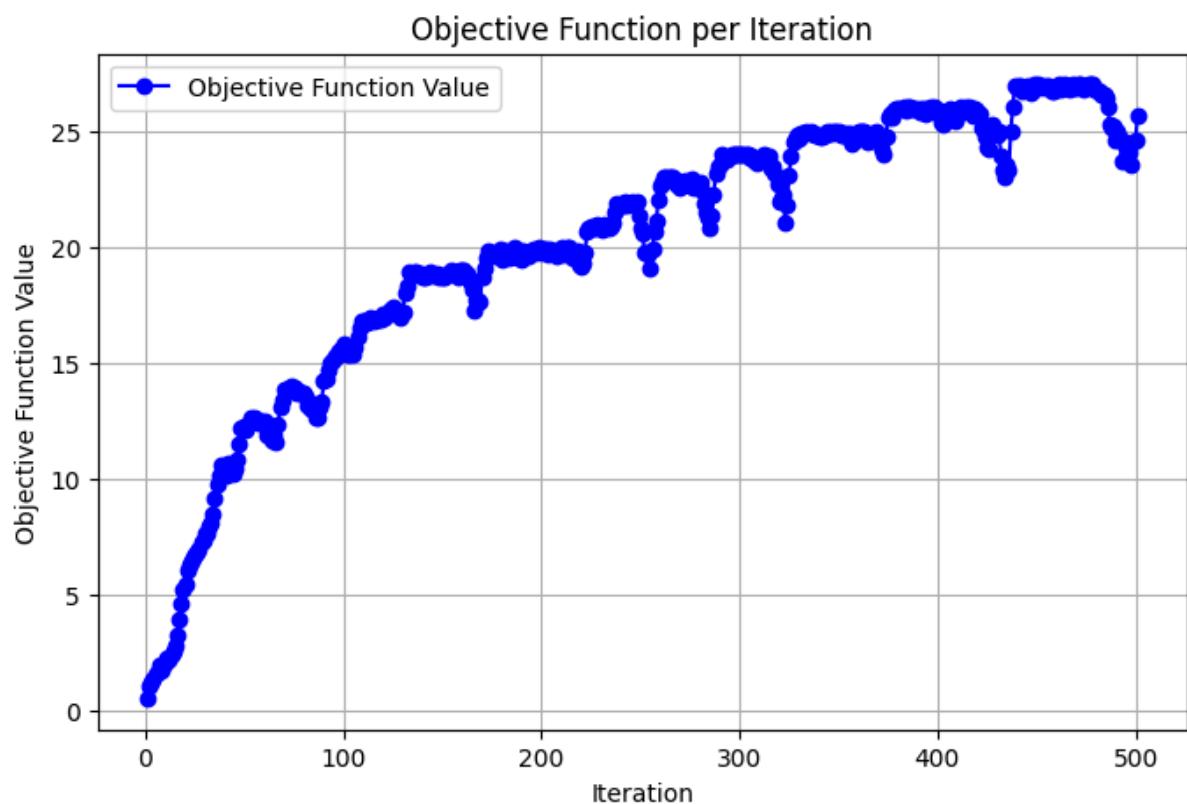
Matrix Level 1					Matrix Level 2					Matrix Level 3				
53	72	78	111	118	65	113	81	28	64	36	42	19	39	61
80	6	1	123	15	30	97	29	2	115	92	11	40	121	51
86	90	62	44	33	3	70	41	107	63	96	84	74	120	5
99	93	114	46	66	26	17	34	88	25	49	91	103	117	10
69	52	60	4	110	45	18	106	98	48	108	125	79	85	67
Matrix Level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

Nilai Objective Function: 31

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Average:



Jumlah Populasi: 200

Banyak Iterasi: 500

Durasi Proses Pencarian: 12 Seconds

2.2.6.3. Percobaan 3

State Awal

3D Cube Matrix Layers

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

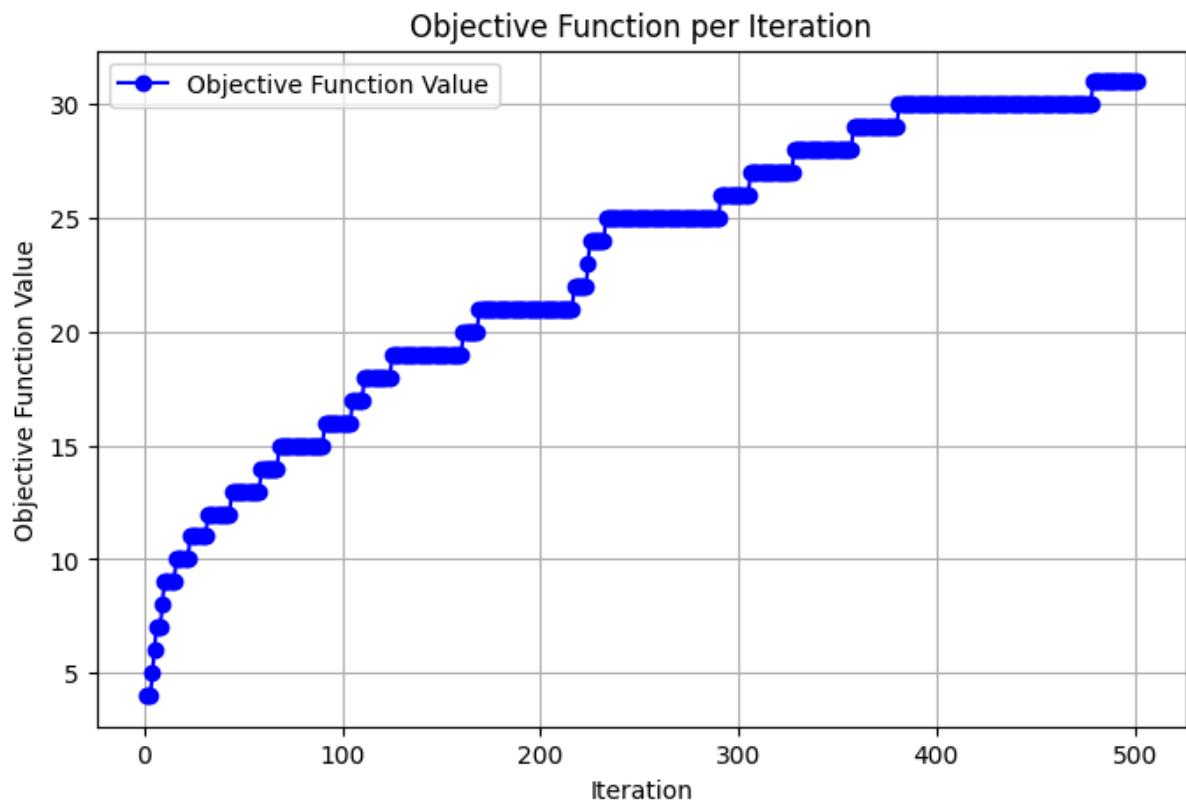
State Akhir

3D Cube Matrix Layers

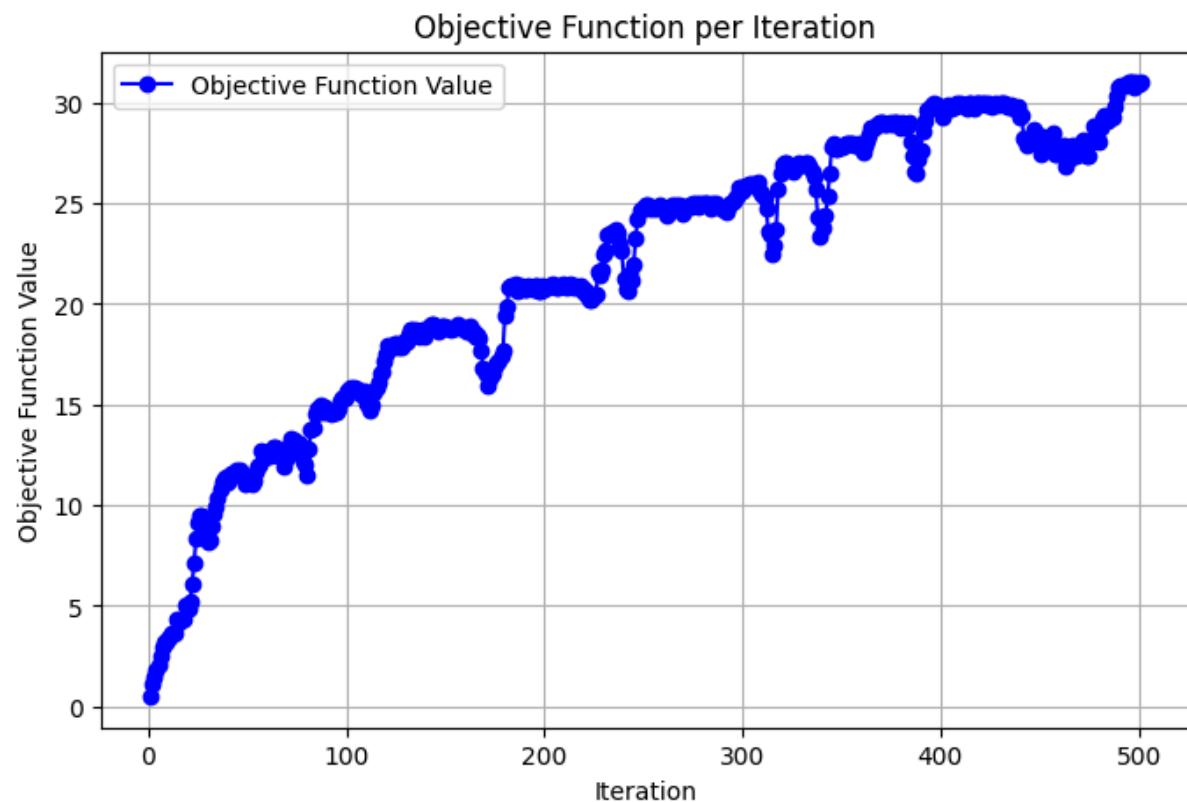
Matrix Level 1					Matrix Level 2					Matrix Level 3				
103	97	72	115	100	17	90	4	7	6	26	69	13	107	67
39	91	44	18	77	75	15	58	80	87	114	40	12	56	93
2	42	20	63	70	88	31	29	74	57	24	83	117	104	60
9	82	98	23	106	108	59	122	8	73	34	64	52	116	49
55	5	81	96	78	125	120	102	86	84	11	92	121	37	54
Matrix Level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

Nilai Objective Function: 31

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 200

Banyak Iterasi: 500

Durasi Proses Pencarian: 1.5 Second

2.2.6.4. Percobaan 4

State Awal

3D Cube Matrix Layers

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

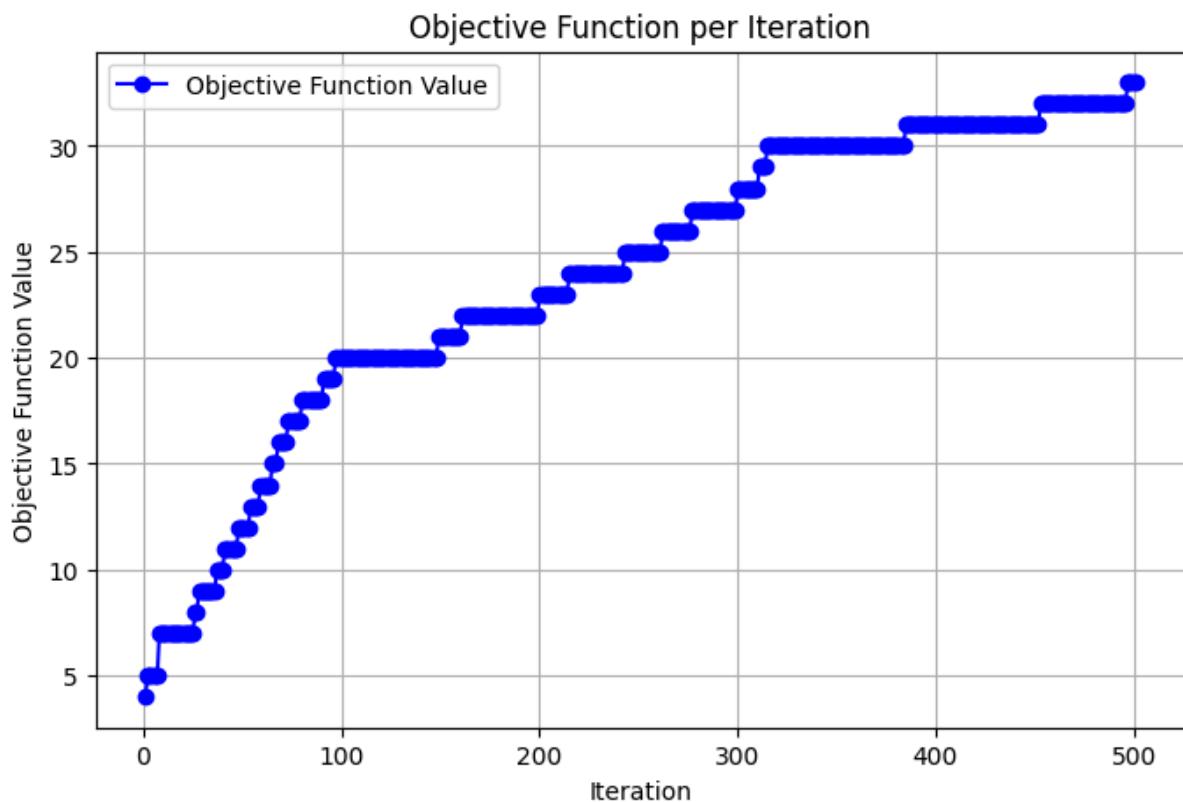
State Akhir

3D Cube Matrix Layers

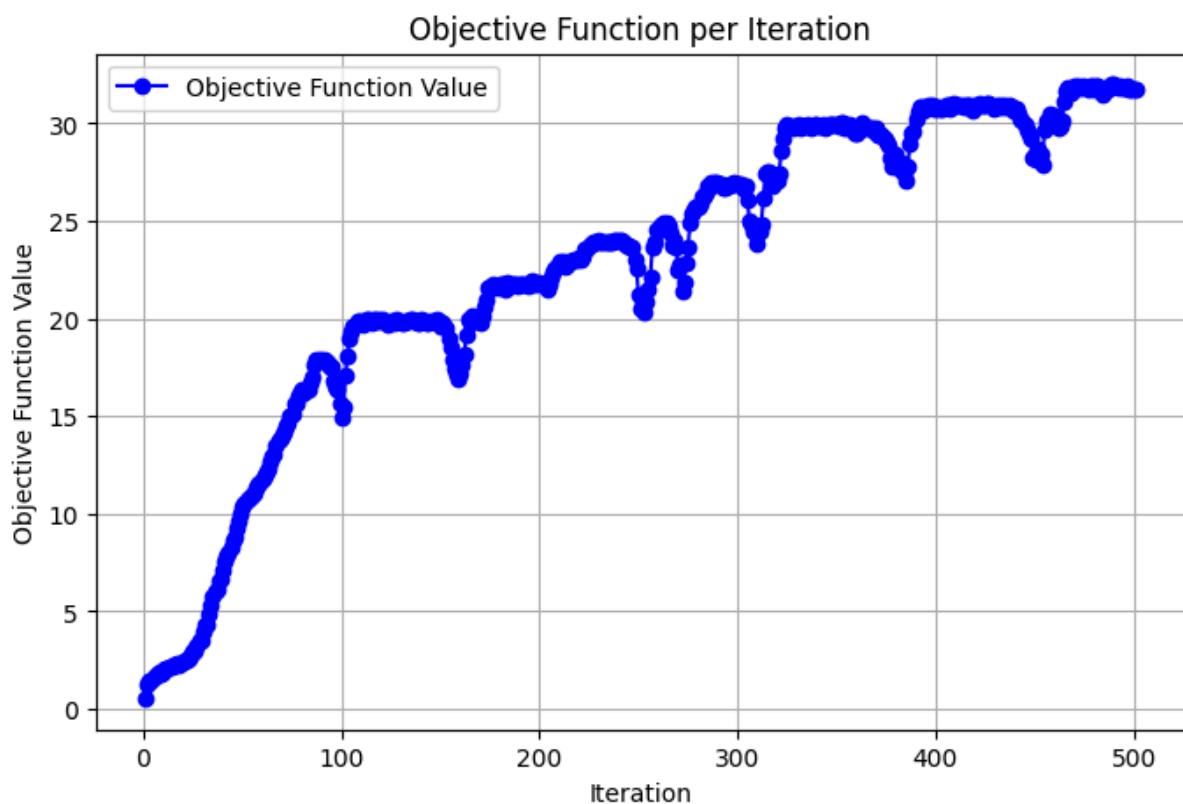
Matrix Level 1					Matrix Level 2					Matrix Level 3				
50	103	94	122	43	83	3	66	117	46	64	100	9	81	61
30	2	10	18	69	63	92	42	16	25	39	109	41	21	105
29	27	51	54	36	1	17	6	47	32	71	70	125	12	37
87	111	67	7	58	88	124	13	85	91	73	31	65	113	108
119	72	93	49	77	123	68	104	110	121	120	40	75	20	60
Matrix Level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

Nilai Objective Function: 33

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 400

Banyak Iterasi: 500

Durasi Proses Pencarian: 32 Second

2.2.6.5. Percobaan 5

State Awal

3D Cube Matrix Layers

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

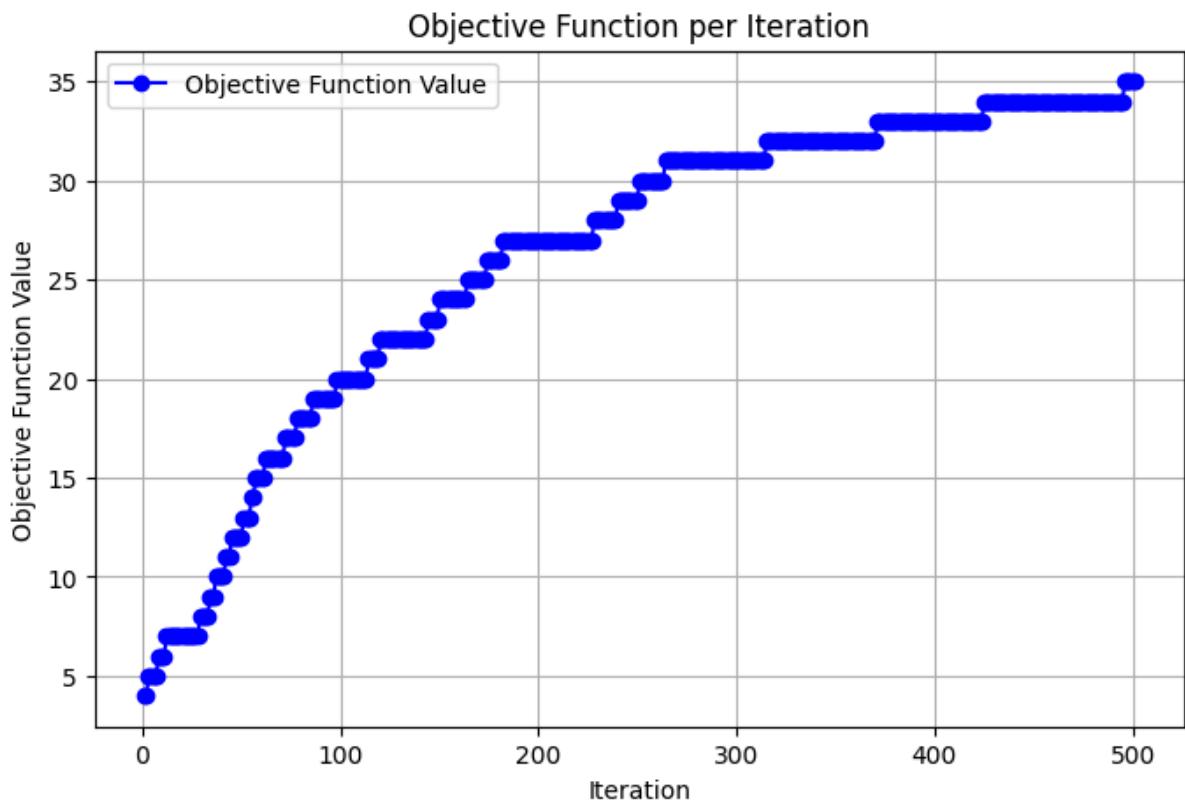
State Akhir

3D Cube Matrix Layers

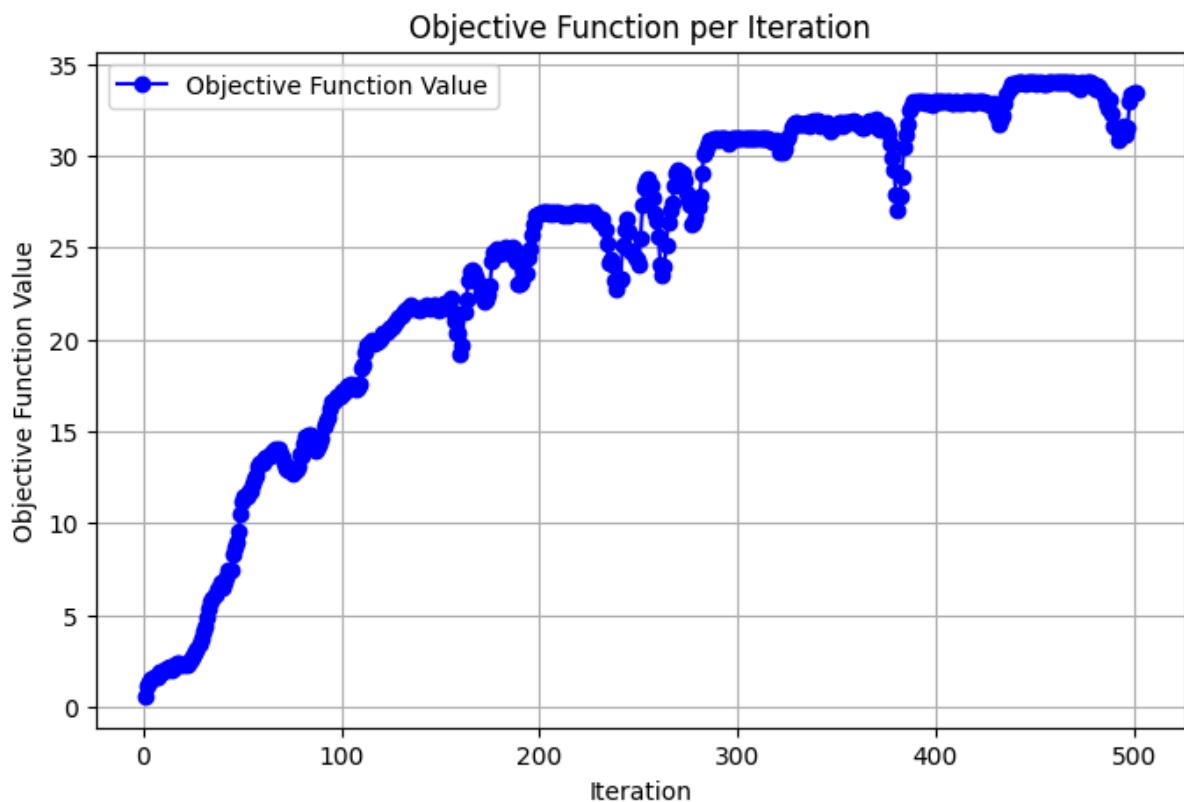
Matrix Level 1					Matrix Level 2					Matrix Level 3				
43	57	37	12	27	112	99	94	95	8	35	98	62	23	123
53	29	78	66	124	24	84	118	25	64	119	109	55	63	5
81	58	70	100	71	111	38	11	65	45	52	41	22	88	79
34	80	68	87	7	67	47	18	3	85	107	105	122	110	69
104	92	91	50	86	1	120	49	32	113	2	56	9	77	39
Matrix Level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

Nilai Objective Function: 35

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 400

Banyak Iterasi: 500

Durasi Proses Pencarian: 37 Second

2.2.6.6. Percobaan 6

State Awal

3D Cube Matrix Layers

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

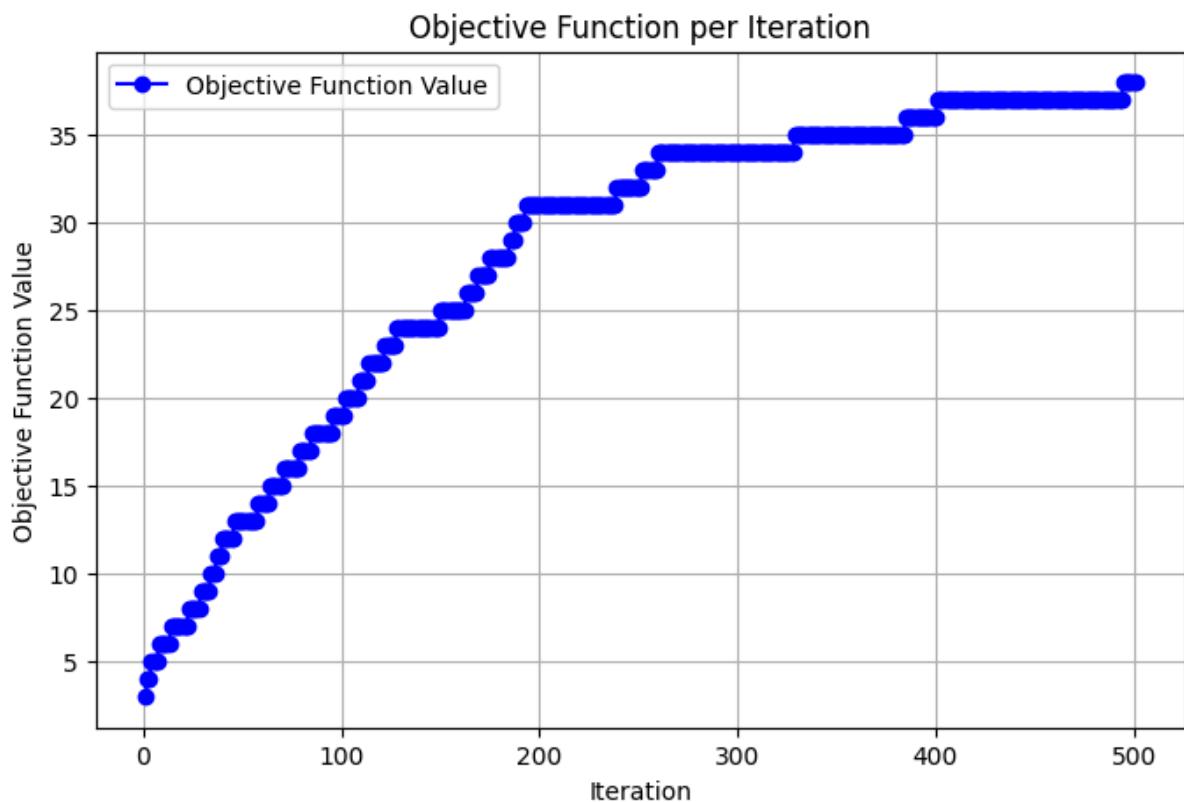
State Akhir

3D Cube Matrix Layers

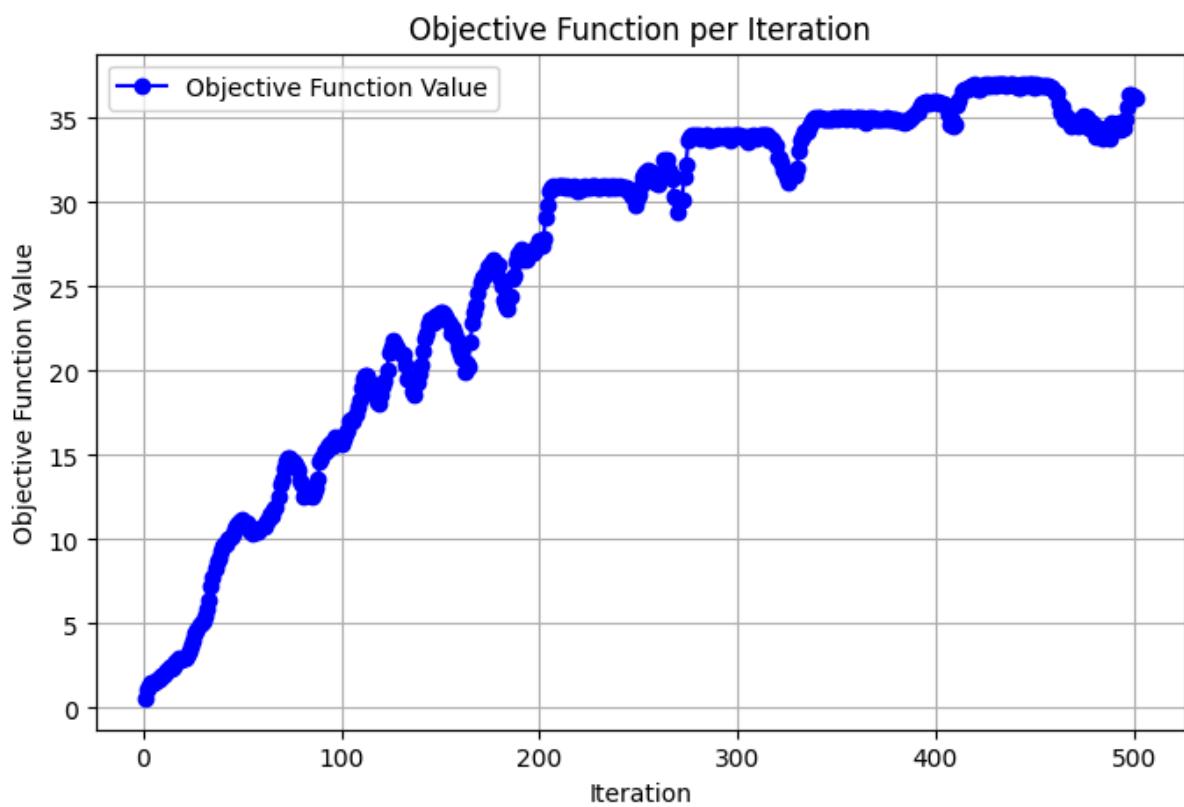
Matrix Level 1					Matrix Level 2					Matrix Level 3				
49	113	112	89	80	77	122	47	100	66	38	107	58	16	96
59	87	50	17	102	20	79	121	68	27	97	106	5	31	76
61	25	33	83	1	11	78	116	69	41	125	99	98	105	30
120	18	108	15	118	51	91	56	3	114	34	39	53	6	60
35	45	12	111	14	103	104	92	75	40	21	84	101	42	67
Matrix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

Nilai Objective Function: 38

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 400

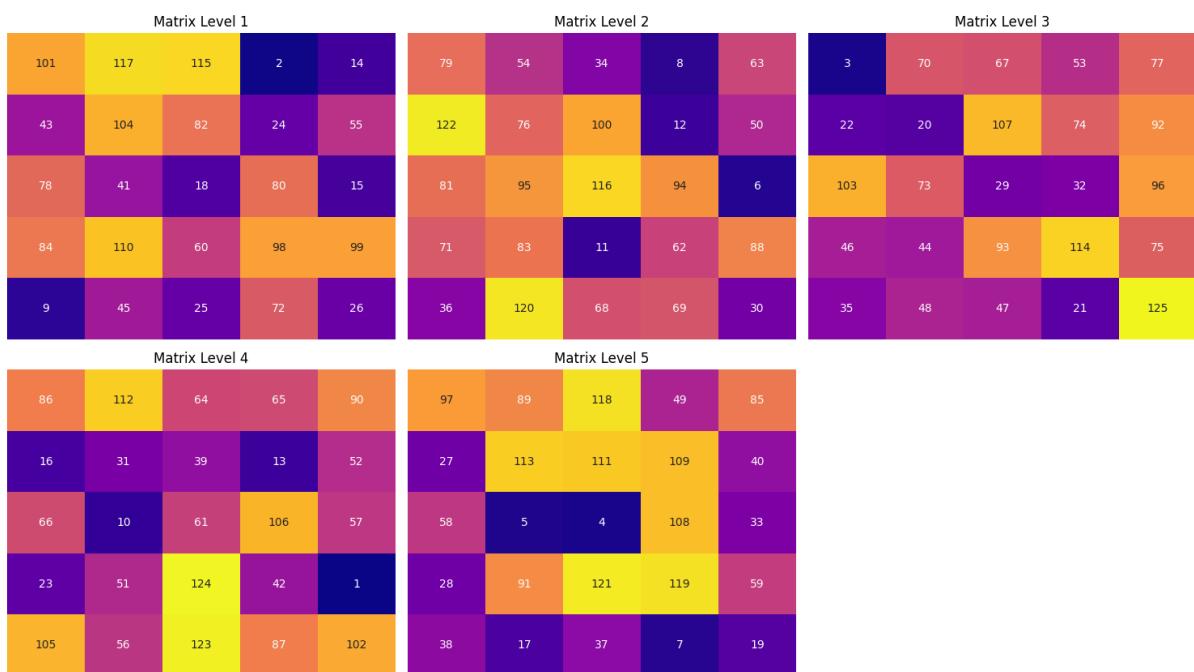
Banyak Iterasi: 500

Durasi Proses Pencarian: 41 Second

2.2.6.7. Percobaan 7

State Awal

3D Cube Matrix Layers



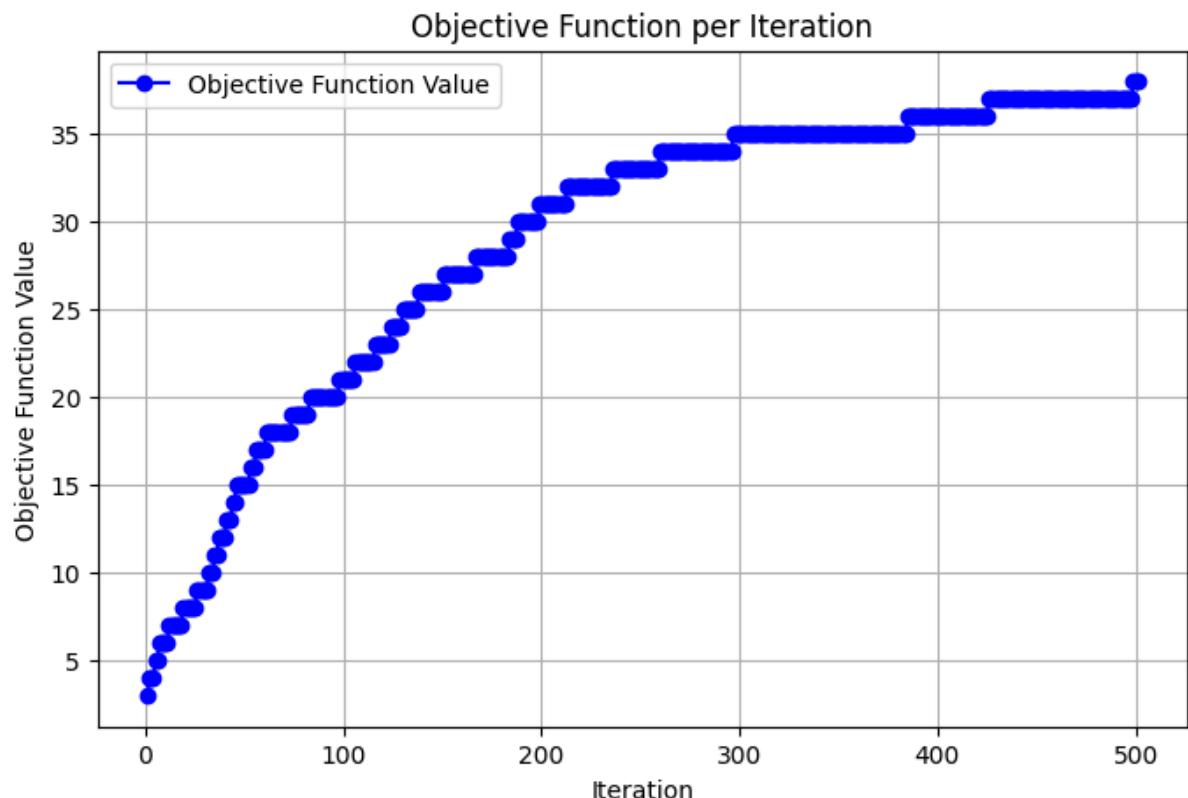
State Akhir

3D Cube Matrix Layers

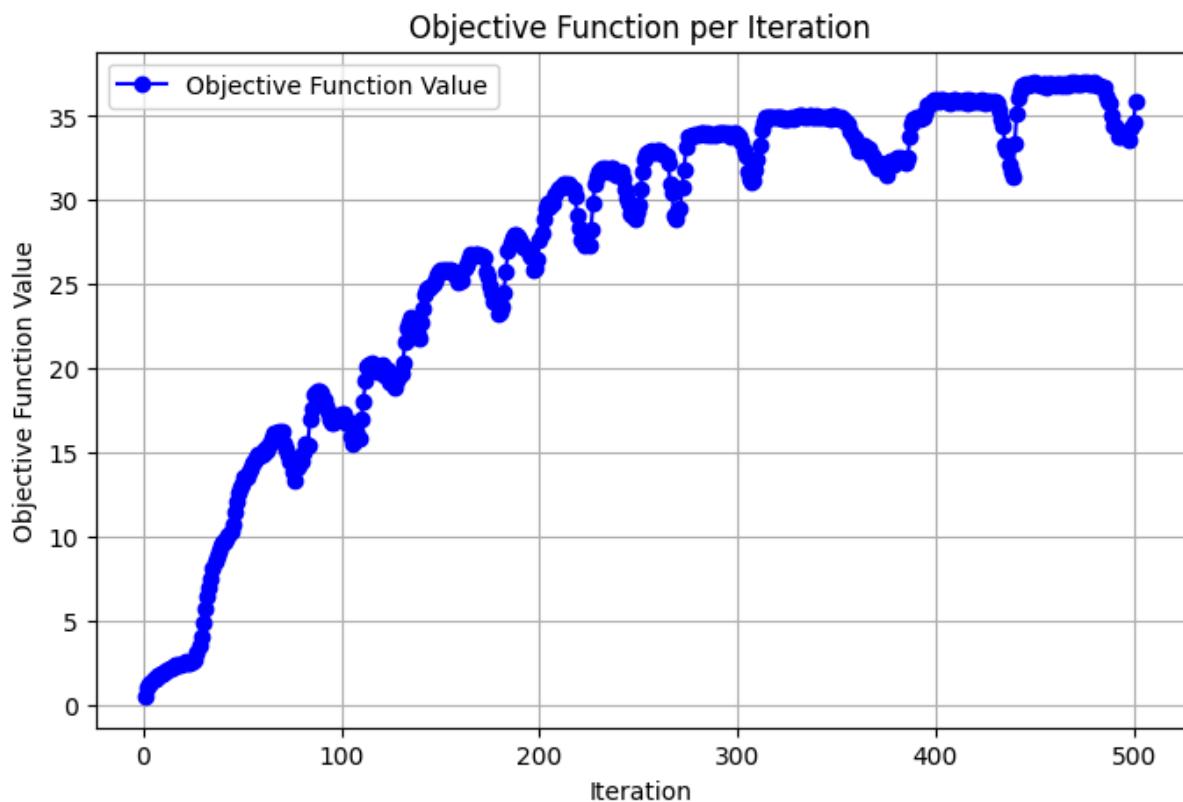
Matrix Level 1					Matrix Level 2					Matrix Level 3				
121	104	32	20	38	75	43	30	60	96	46	67	86	17	99
41	94	105	13	62	35	73	45	110	98	80	111	50	61	115
23	77	1	66	89	112	36	97	15	55	26	37	119	91	42
125	16	54	95	25	44	10	27	22	117	100	82	71	113	31
5	24	123	49	4	21	8	116	78	48	108	63	83	33	28
Matrix Level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

Nilai Objective Function: 38

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 600

Banyak Iterasi: 500

Durasi Proses Pencarian: 69 Second

2.2.6.8. Percobaan 8

State Awal

3D Cube Matrix Layers

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

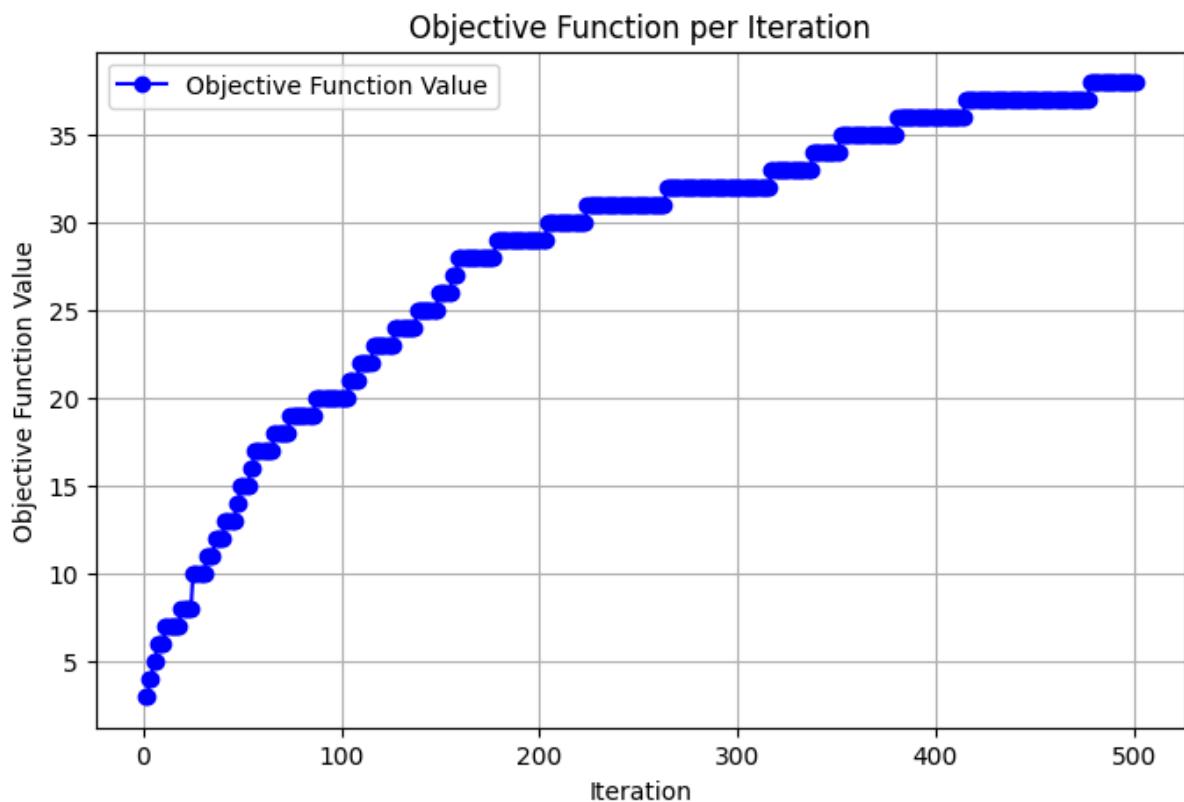
State Akhir

3D Cube Matrix Layers

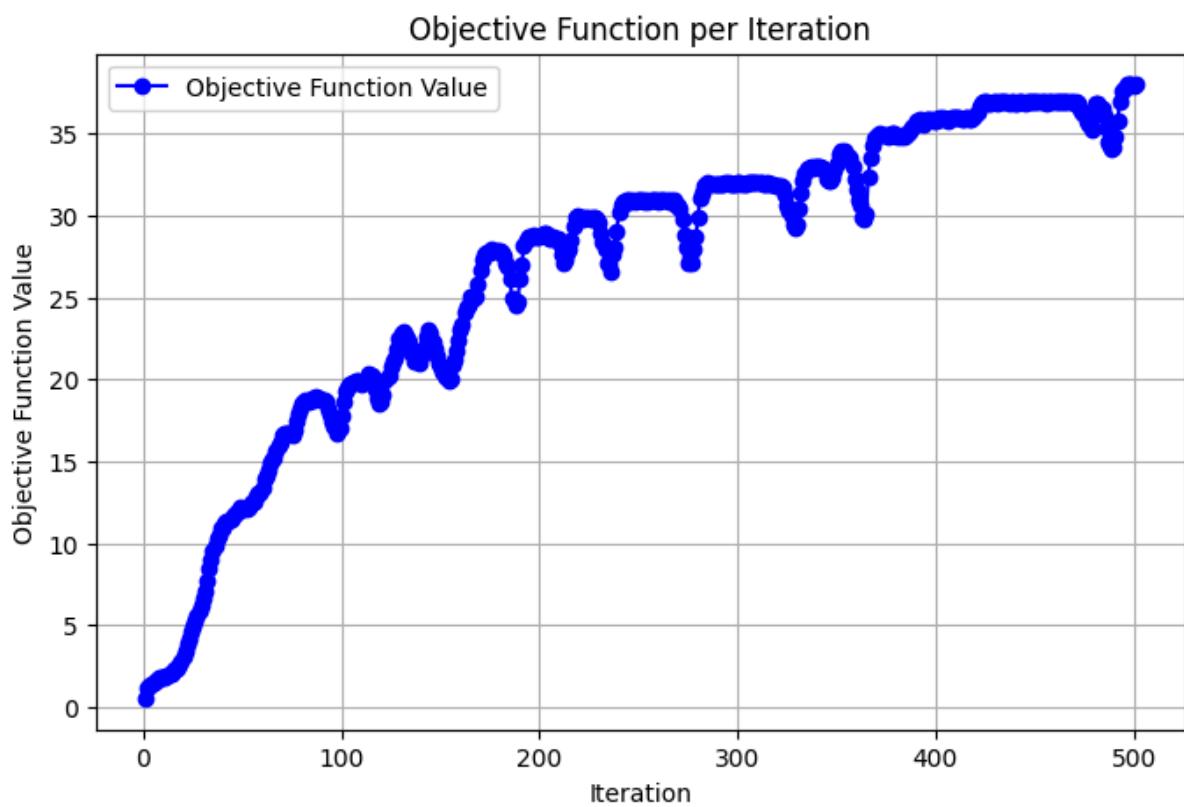
Matrix Level 1					Matrix Level 2					Matrix Level 3				
64	79	49	70	40	13	83	102	57	60	117	32	88	48	30
86	22	38	55	52	35	124	101	28	109	125	39	31	66	46
92	44	105	24	50	100	43	73	116	29	3	114	27	68	56
8	95	2	21	93	106	77	36	42	54	12	119	67	18	99
20	118	121	25	80	61	47	87	72	63	58	11	81	94	84
Matrix Level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

Nilai Objective Function: 38

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 600

Banyak Iterasi: 500

Durasi Proses Pencarian: 72 Second

2.2.6.9. Percobaan 9

State Awal

3D Cube Matrix Layers

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

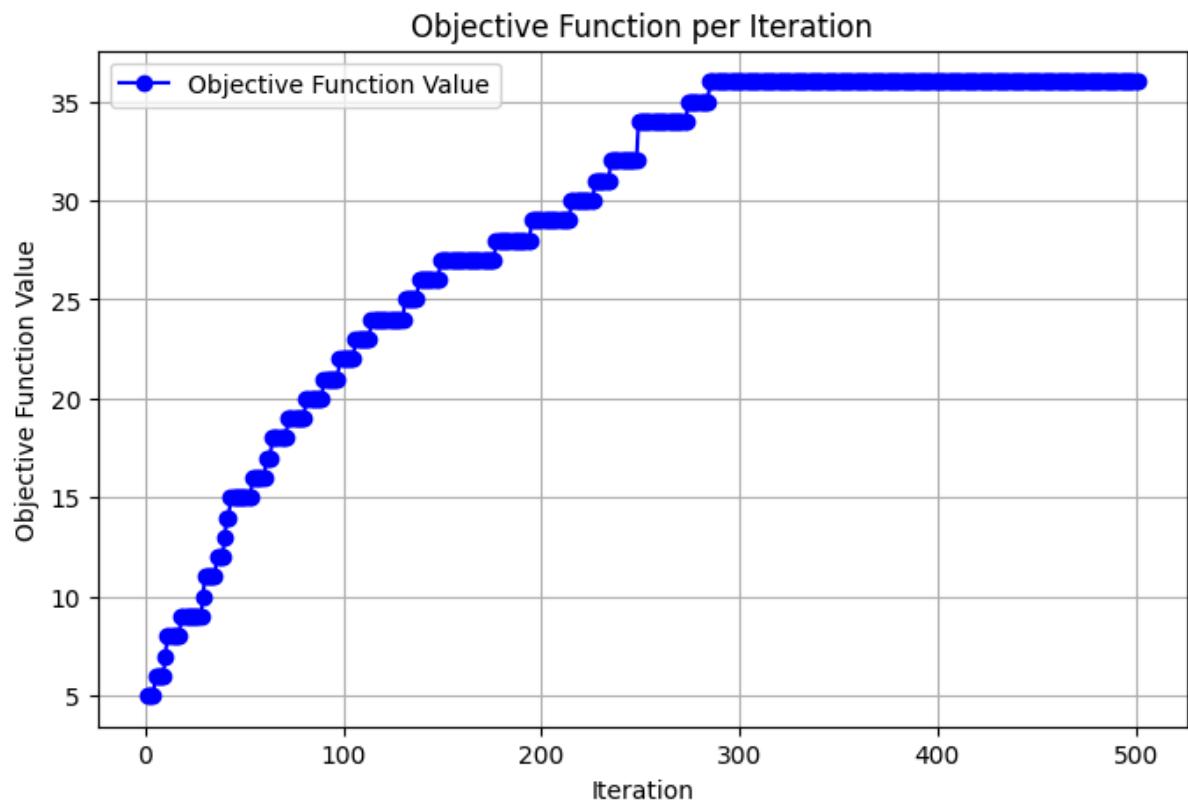
State Akhir

3D Cube Matrix Layers

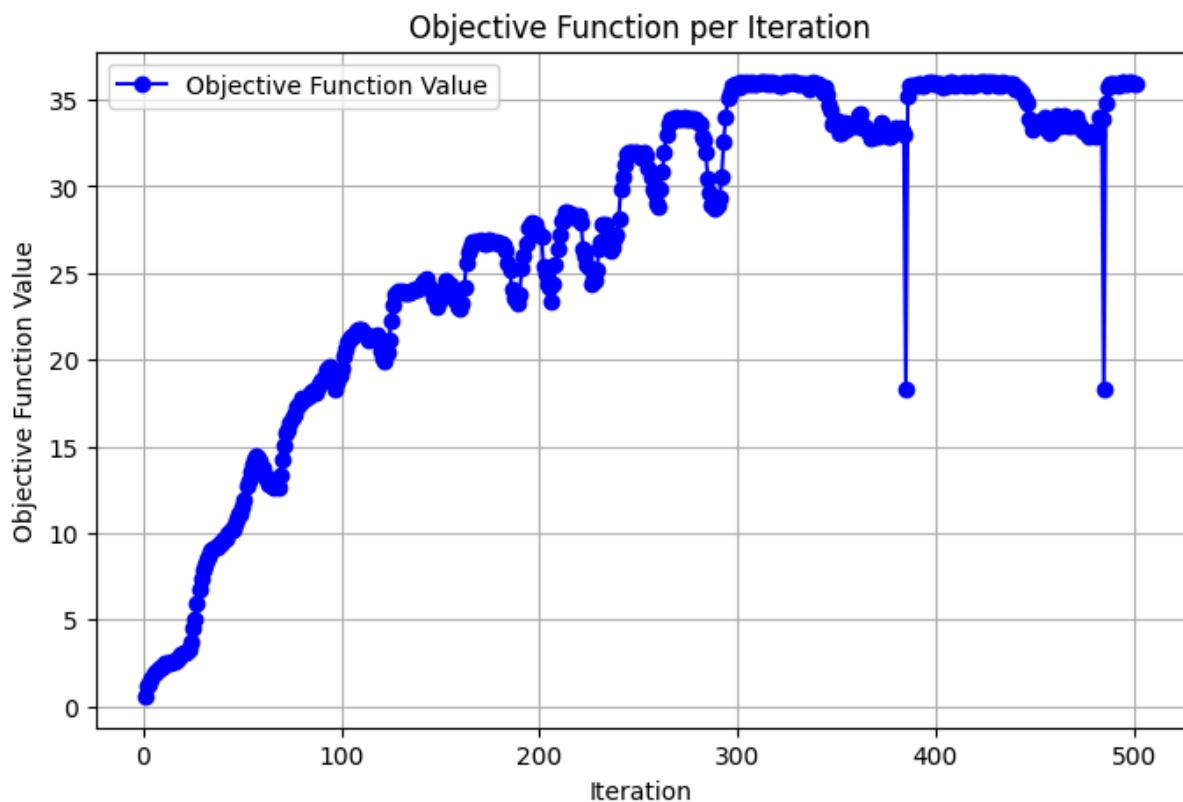
Matrix Level 1					Matrix Level 2					Matrix Level 3				
45	72	46	98	54	115	36	12	68	34	64	56	101	14	63
38	4	39	123	111	5	89	76	32	92	107	97	1	88	22
66	102	106	23	18	117	124	19	3	52	82	55	81	42	17
108	27	26	2	77	47	91	62	78	37	113	13	122	16	118
120	110	86	69	15	31	51	61	99	100	73	94	90	80	57
Matrix Level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

Nilai Objective Function: 36

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 600

Banyak Iterasi: 500

Durasi Proses Pencarian: 73 Second

2.2.6.10. Percobaan 10

State Awal

3D Cube Matrix Layers

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

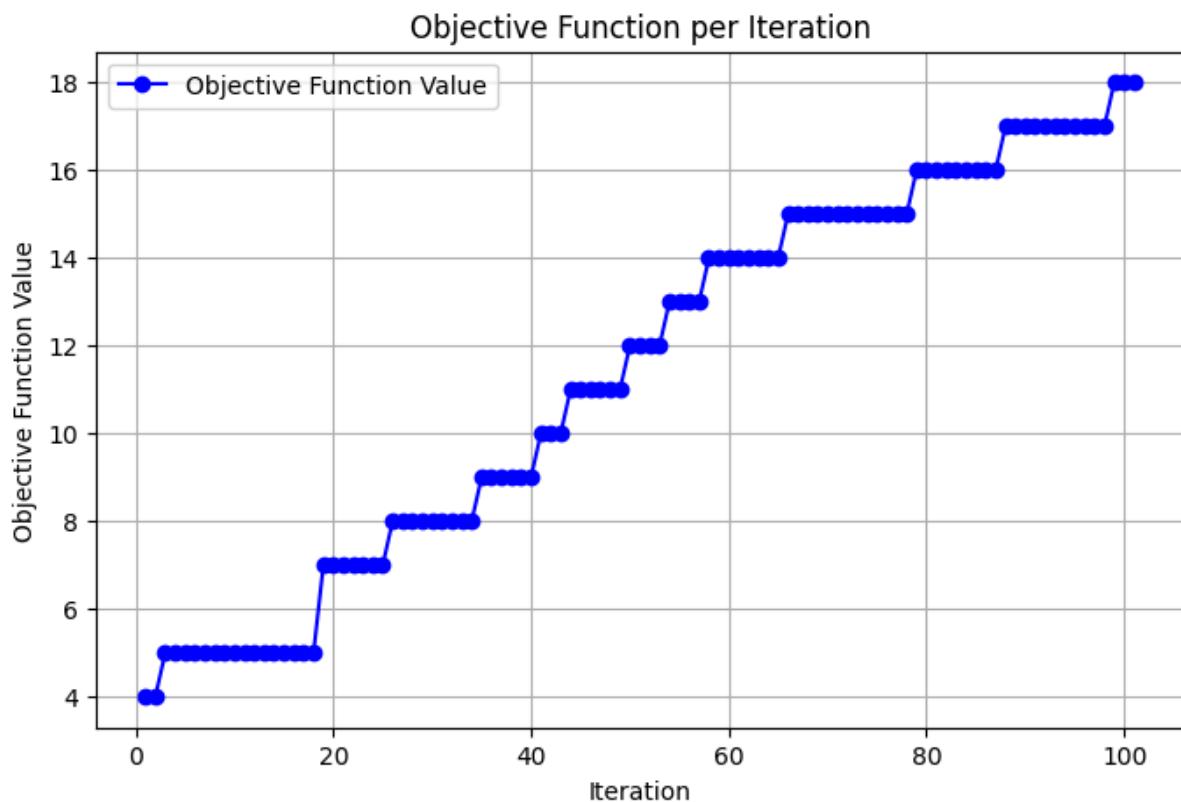
State Akhir

3D Cube Matrix Layers

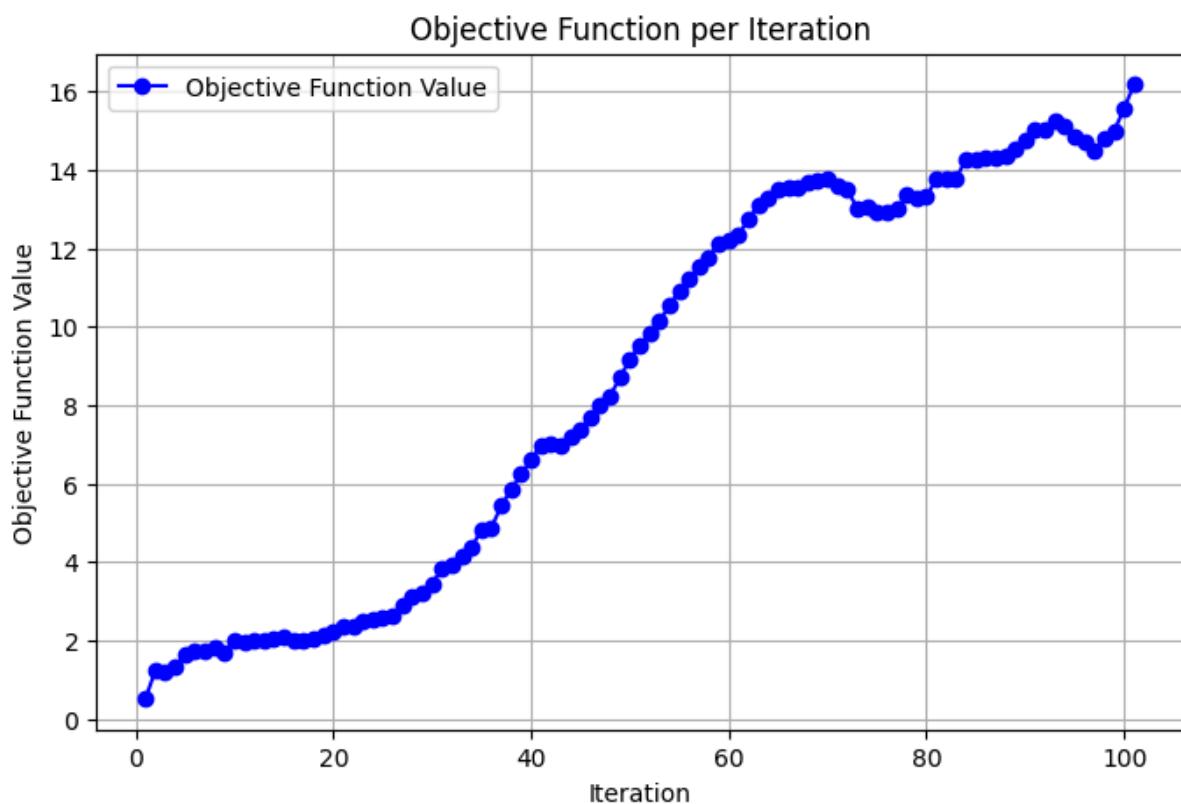
Matrix Level 1					Matrix Level 2					Matrix Level 3				
110	23	18	24	52	77	84	21	20	113	55	108	8	61	74
19	31	62	72	41	87	11	28	80	109	6	44	111	120	34
58	67	98	33	59	22	116	54	82	122	27	90	89	70	39
50	13	101	65	107	104	112	95	100	119	92	25	3	47	16
78	125	37	115	51	29	97	121	15	53	7	88	64	26	106
Matrix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

Nilai Objective Function: 18

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 300

Banyak Iterasi: 100

Durasi Proses Pencarian: 6 Second

2.2.6.11. Percobaan 11

State Awal

3D Cube Matrix Layers

Matrix Level 1					Matrix Level 2					Matrix Level 3				
87	29	109	16	71	14	65	62	38	7	47	43	59	102	36
18	17	103	123	125	33	111	94	92	51	115	116	42	96	79
10	78	26	52	49	80	108	46	56	118	105	4	13	106	83
107	31	91	53	74	85	86	98	67	12	6	69	100	81	113
37	22	76	28	30	114	58	112	124	104	75	25	119	32	61

Matrix Level 4					Matrix Level 5				
110	48	50	77	23	15	122	20	54	41
88	68	35	99	21	19	66	60	8	5
84	97	93	121	2	1	90	57	72	63
9	73	64	89	44	40	39	70	24	95
27	34	82	117	55	45	3	120	11	101

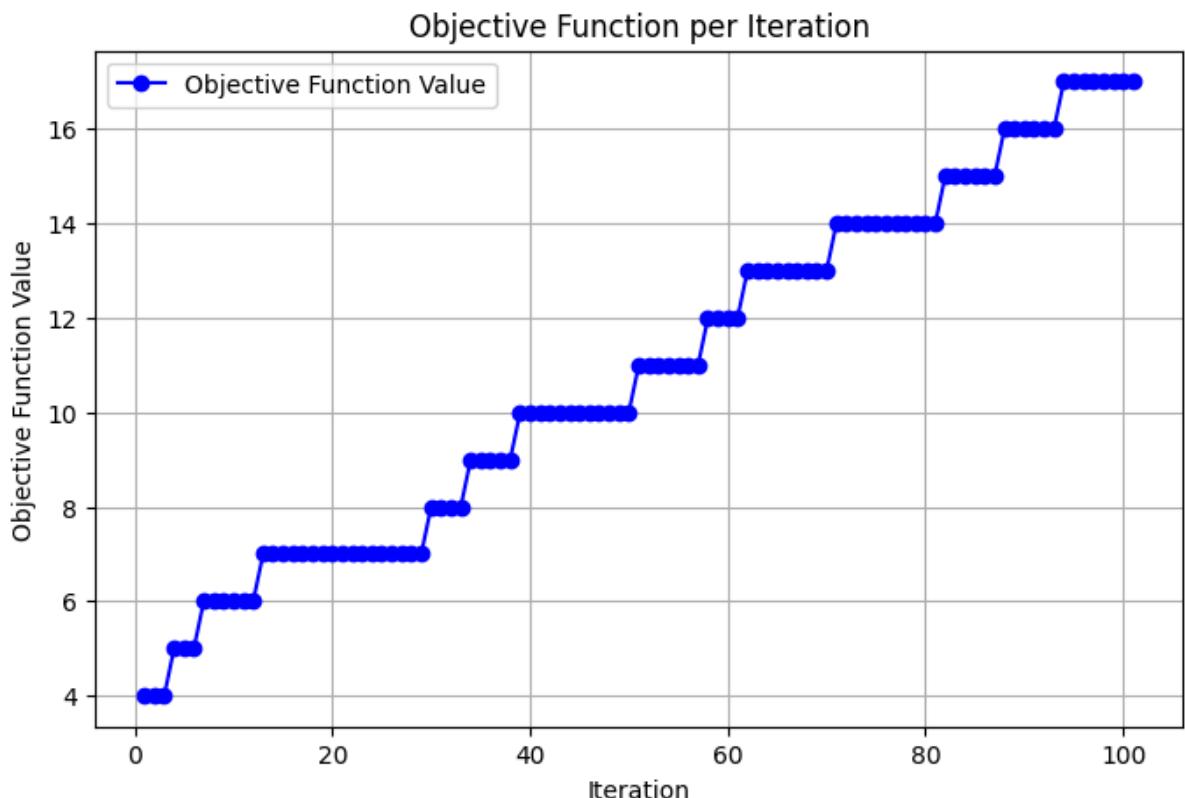
State Akhir

3D Cube Matrix Layers

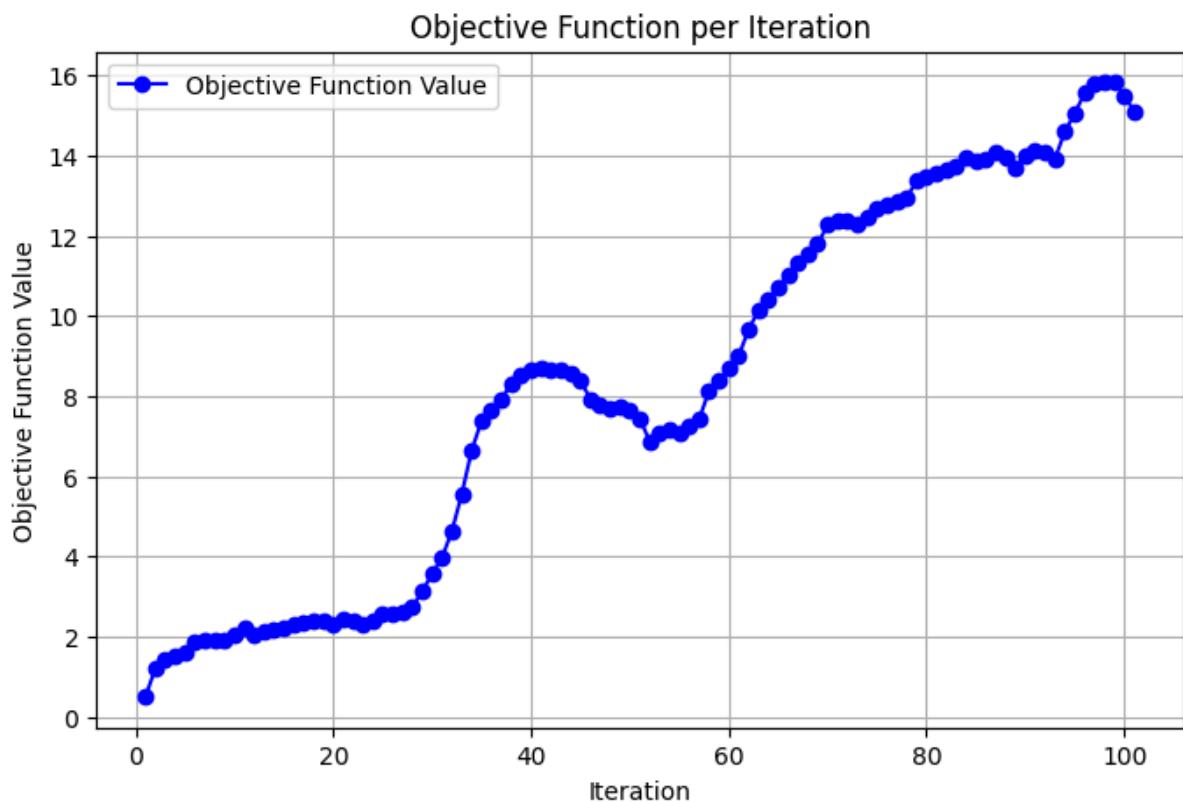
Matrix Level 1					Matrix Level 2					Matrix Level 3				
76	12	53	66	98	119	71	9	124	61	94	106	113	3	109
78	101	51	10	40	30	123	74	55	33	108	82	48	29	75
8	52	58	116	45	15	32	65	72	20	86	17	11	114	99
110	46	41	39	70	107	69	56	36	83	59	5	35	16	25
43	68	112	84	89	19	23	96	28	118	4	102	64	115	7
Matrix Level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

Nilai Objective Function: 17

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 300

Banyak Iterasi: 100

Durasi Proses Pencarian: 6 Second

2.2.6.12. Percobaan 12

State Awal

3D Cube Matrix Layers

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

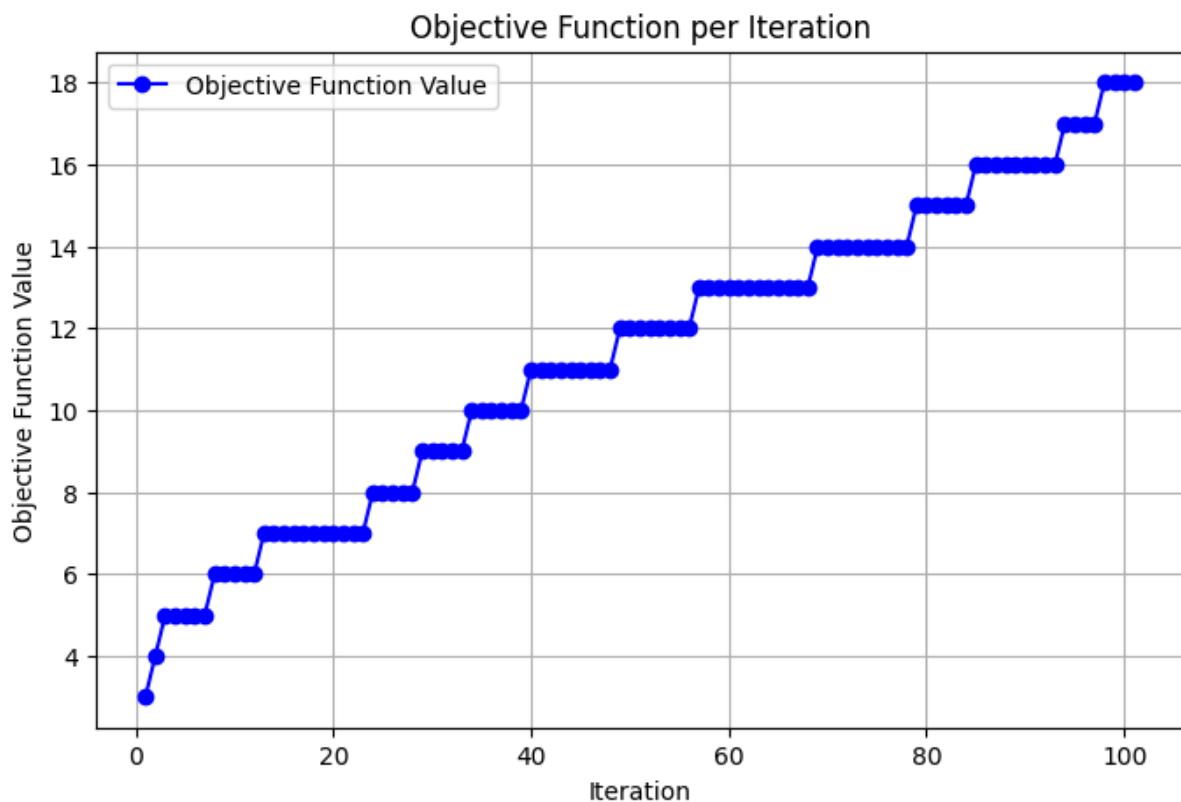
State Akhir

3D Cube Matrix Layers

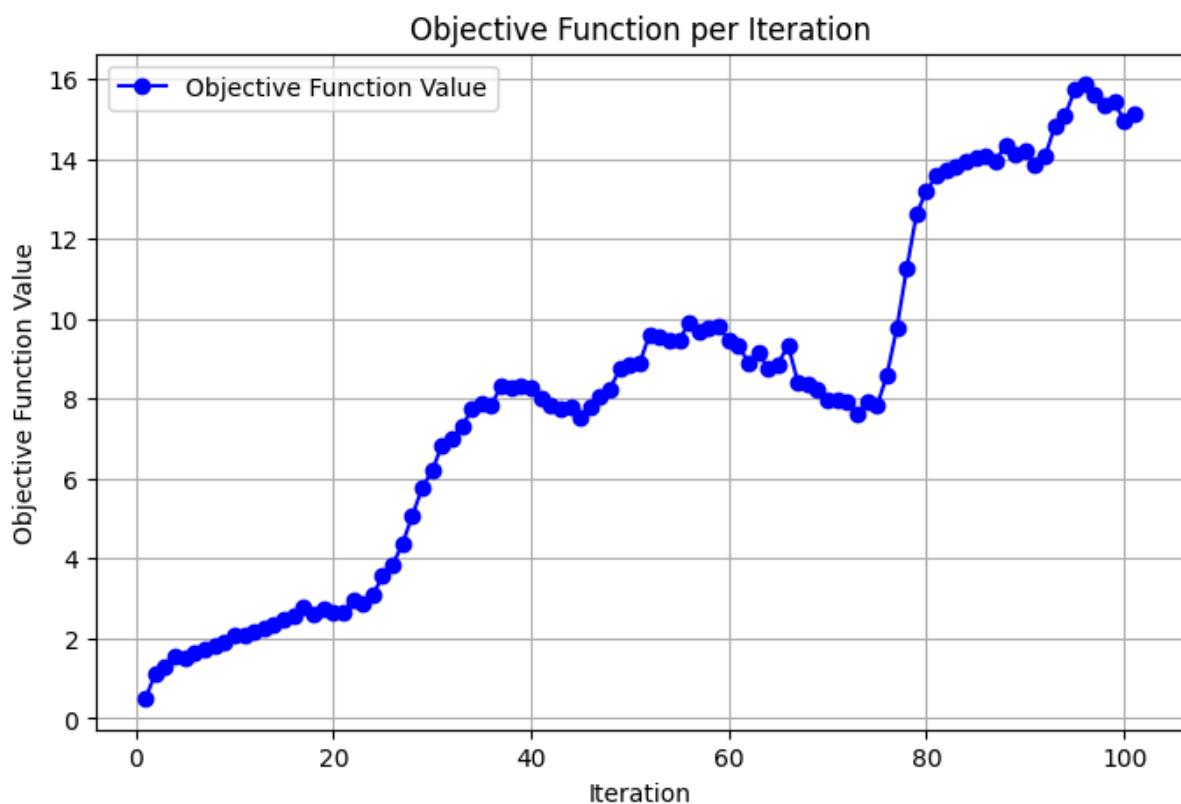
Matrix Level 1					Matrix Level 2					Matrix Level 3				
107	77	52	112	113	110	60	5	24	116	120	58	9	115	13
33	64	117	8	31	100	7	26	109	73	28	74	1	104	89
79	108	86	76	78	29	102	65	91	50	75	68	55	101	106
82	27	94	16	61	114	105	34	45	17	6	95	4	49	87
47	39	98	38	118	3	2	54	23	88	48	59	70	46	44
Matrix Level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

Nilai Objective Function: 18

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 300

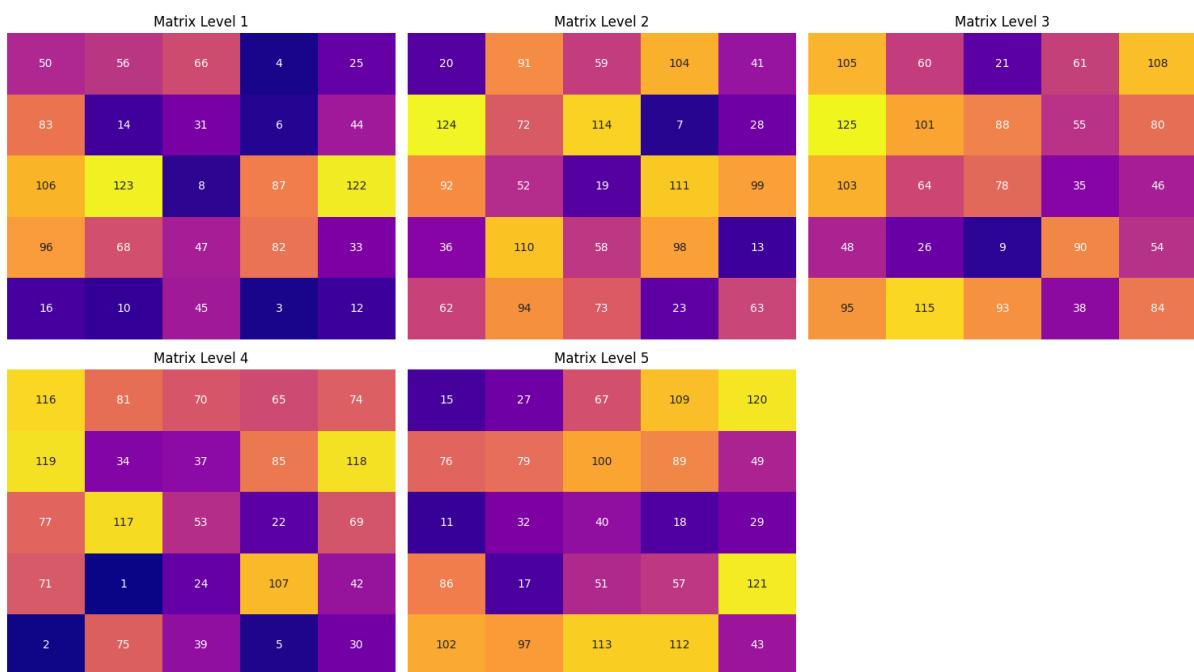
Banyak Iterasi: 100

Durasi Proses Pencarian: 6 Second

2.2.6.13. Percobaan 13

State Awal

3D Cube Matrix Layers



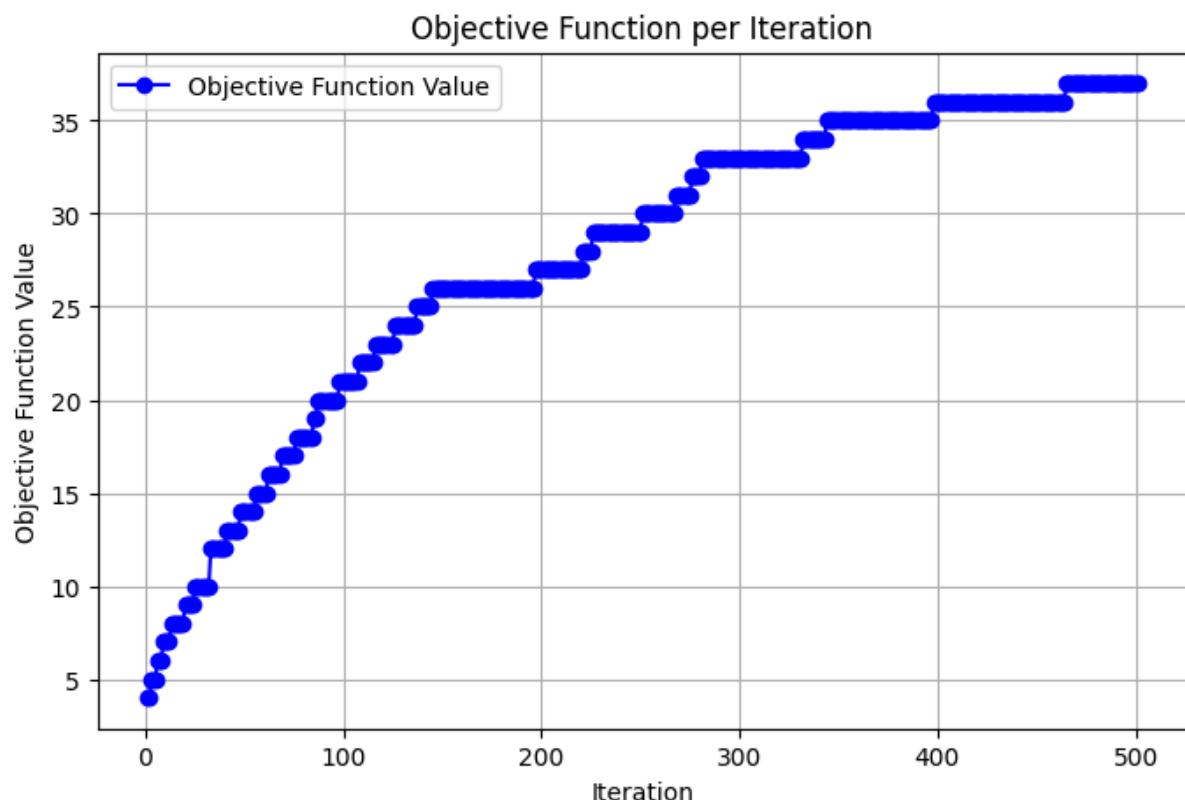
State Akhir

3D Cube Matrix Layers

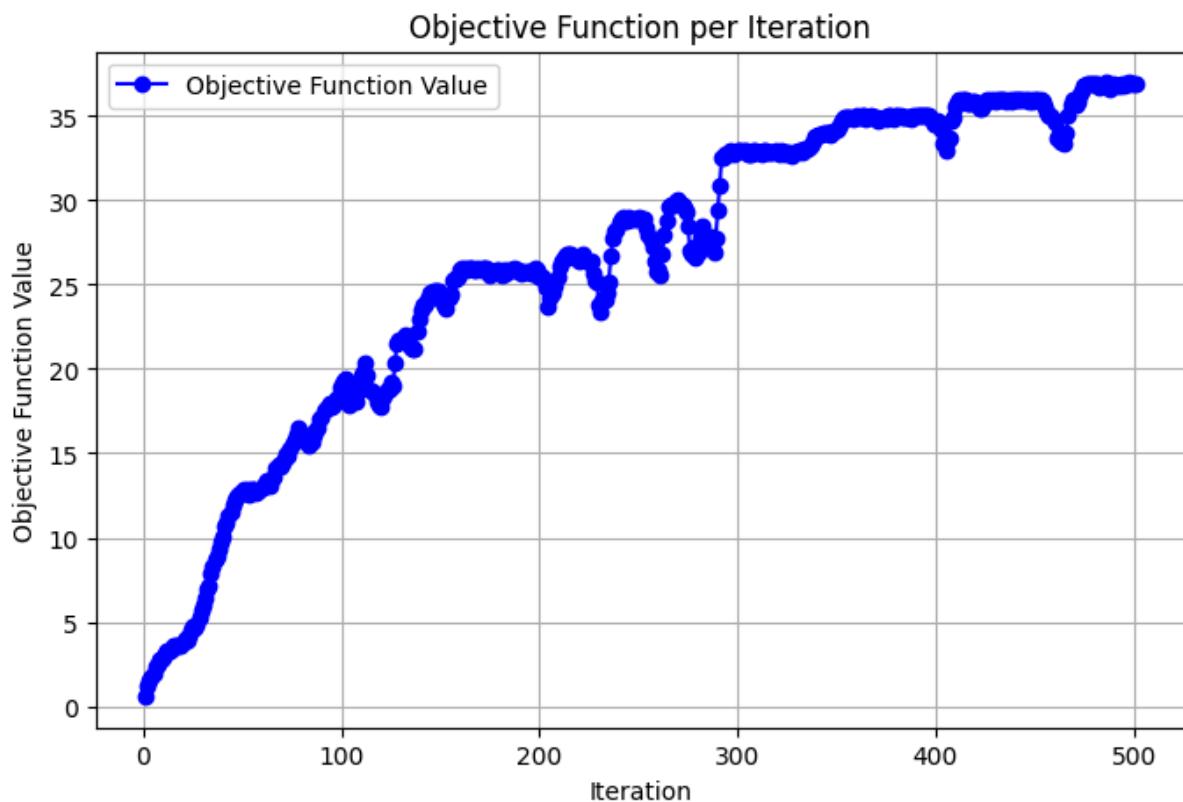
Matrix Level 1					Matrix Level 2					Matrix Level 3				
56	10	3	124	91	71	18	112	76	75	107	60	21	78	49
59	104	114	33	5	45	41	20	7	65	9	117	93	90	43
86	24	72	27	39	92	52	19	111	99	121	14	83	46	51
82	87	11	67	95	36	110	58	98	13	47	64	2	57	88
32	106	40	125	12	62	94	73	23	63	31	74	116	44	84
Matrix Level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

Nilai Objective Function: 37

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 300

Banyak Iterasi: 500

Durasi Proses Pencarian: 1.6 Second

2.2.6.14. Percobaan 14

State Awal

3D Cube Matrix Layers

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

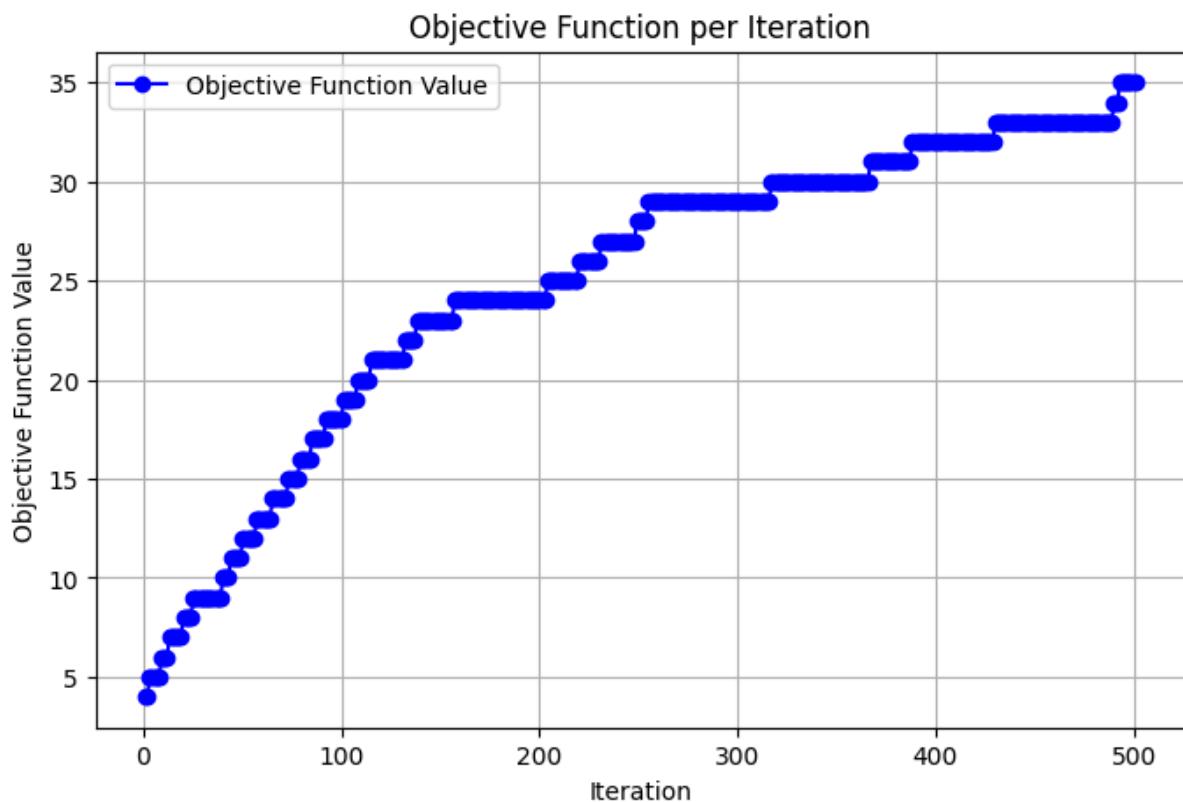
State Akhir

3D Cube Matrix Layers

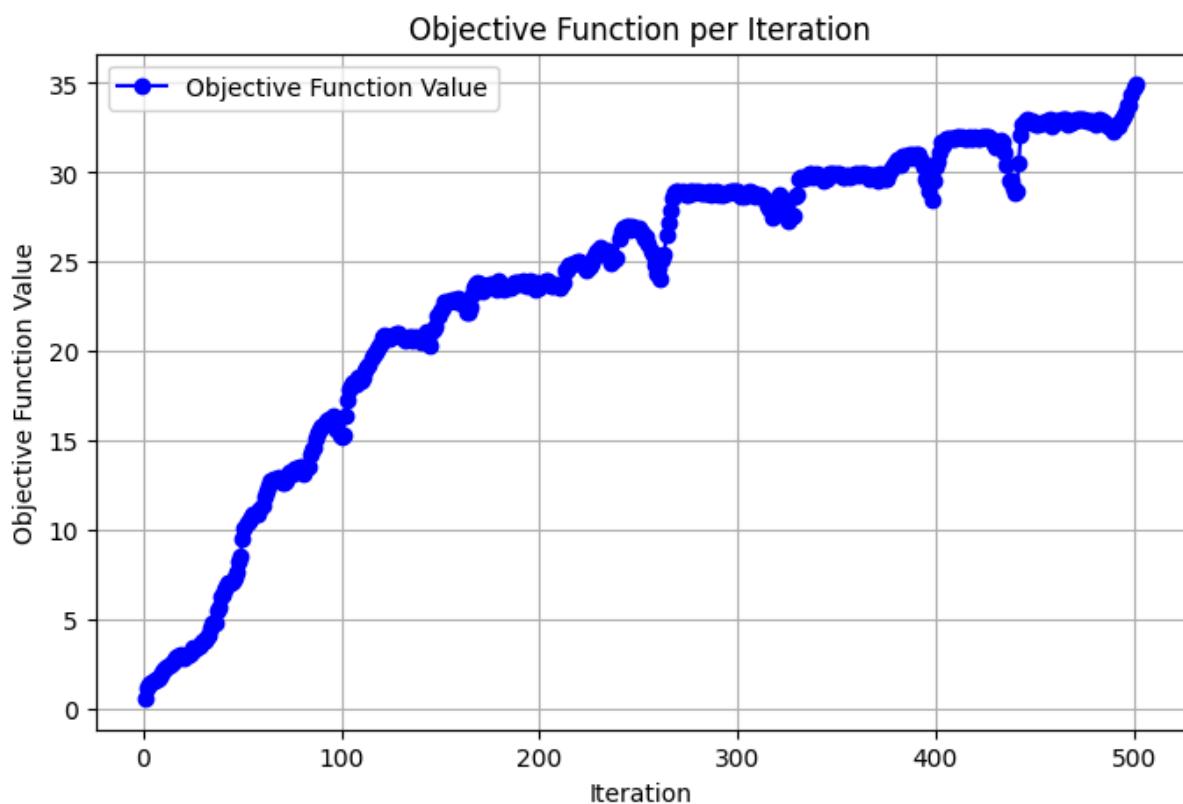
Matrix Level 1					Matrix Level 2					Matrix Level 3				
5	95	84	31	100	32	47	78	16	1	15	36	92	121	51
49	101	63	4	98	44	103	17	72	79	104	14	46	94	50
38	70	45	48	76	107	30	93	83	118	13	91	27	97	28
112	110	61	24	8	66	7	82	85	75	9	88	64	90	117
68	43	113	58	33	124	89	11	19	42	67	86	25	54	69
Matrix Level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

Nilai Objective Function: 35

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 300

Banyak Iterasi: 500

Durasi Proses Pencarian: 30 Second

2.2.6.15. Percobaan 15

State Awal

3D Cube Matrix Layers

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

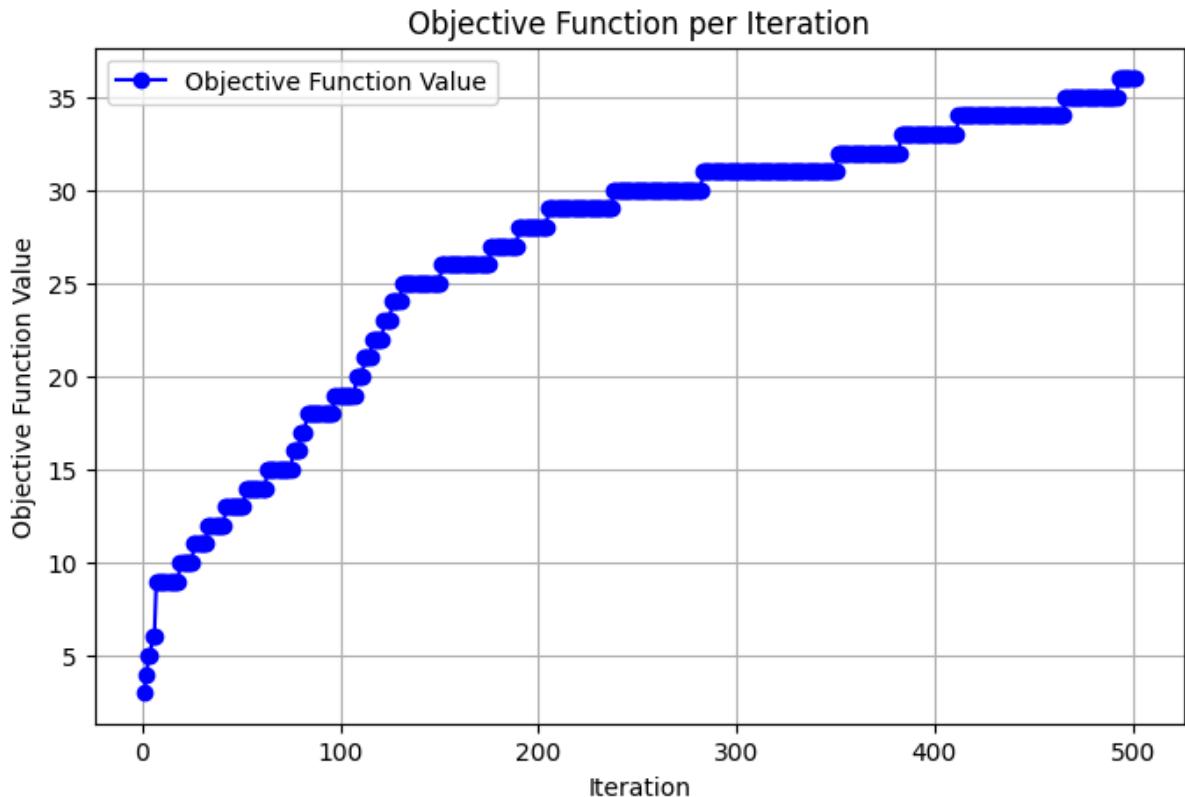
State Akhir

3D Cube Matrix Layers

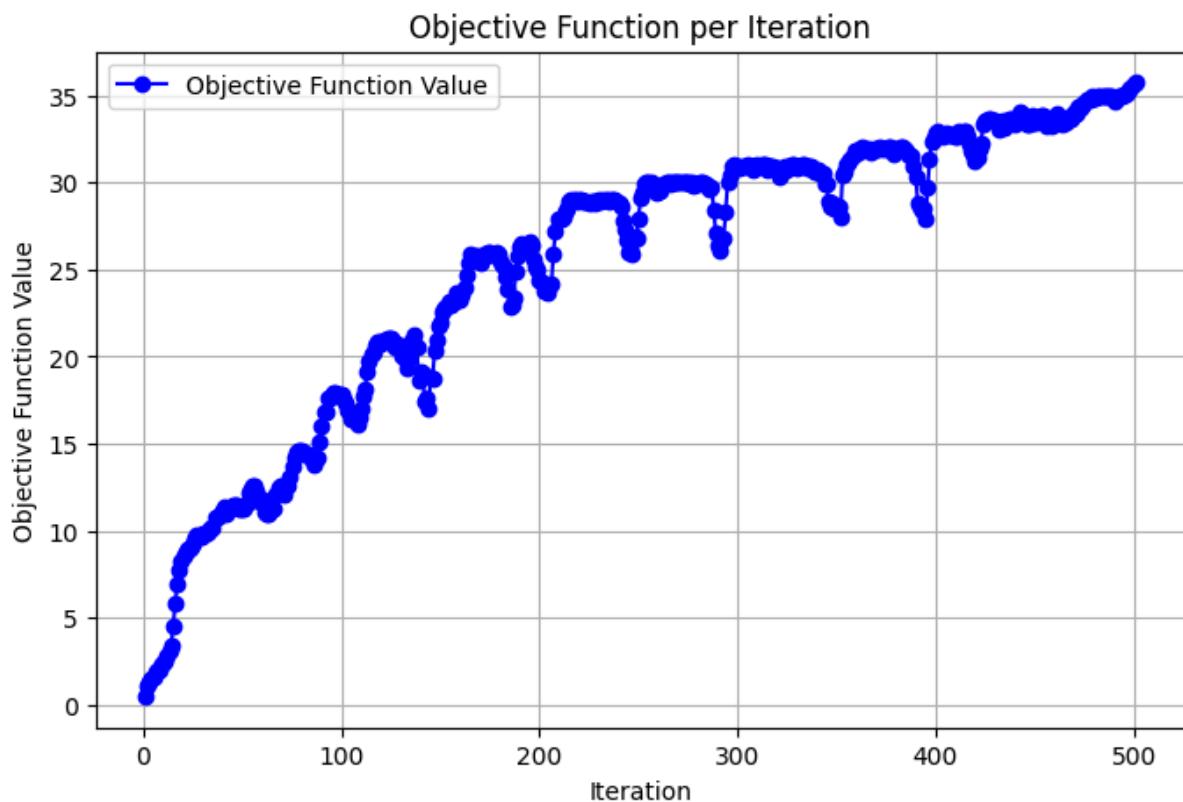
Matrix Level 1					Matrix Level 2					Matrix Level 3				
3	113	47	87	1	58	112	102	27	43	75	93	33	52	20
118	37	16	117	100	88	51	119	69	110	92	22	65	41	95
44	85	7	90	89	78	81	121	26	120	83	105	84	86	19
99	82	49	13	24	18	9	79	17	64	114	31	94	28	48
108	11	103	8	101	73	62	5	107	68	109	40	39	21	106
Matrix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

Nilai Objective Function: 36

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 300

Banyak Iterasi: 500

Durasi Proses Pencarian: 30 Second

2.2.6.16. Percobaan 16

State Awal

3D Cube Matrix Layers

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

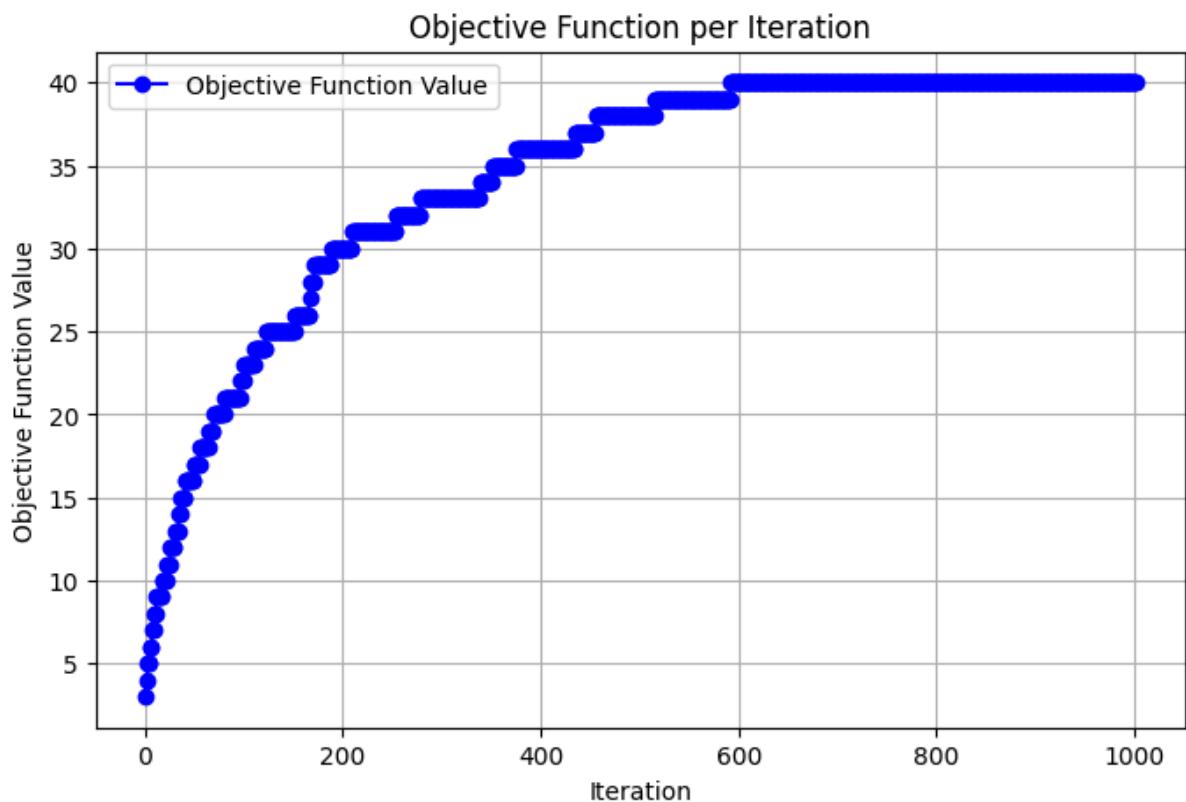
State Akhir

3D Cube Matrix Layers

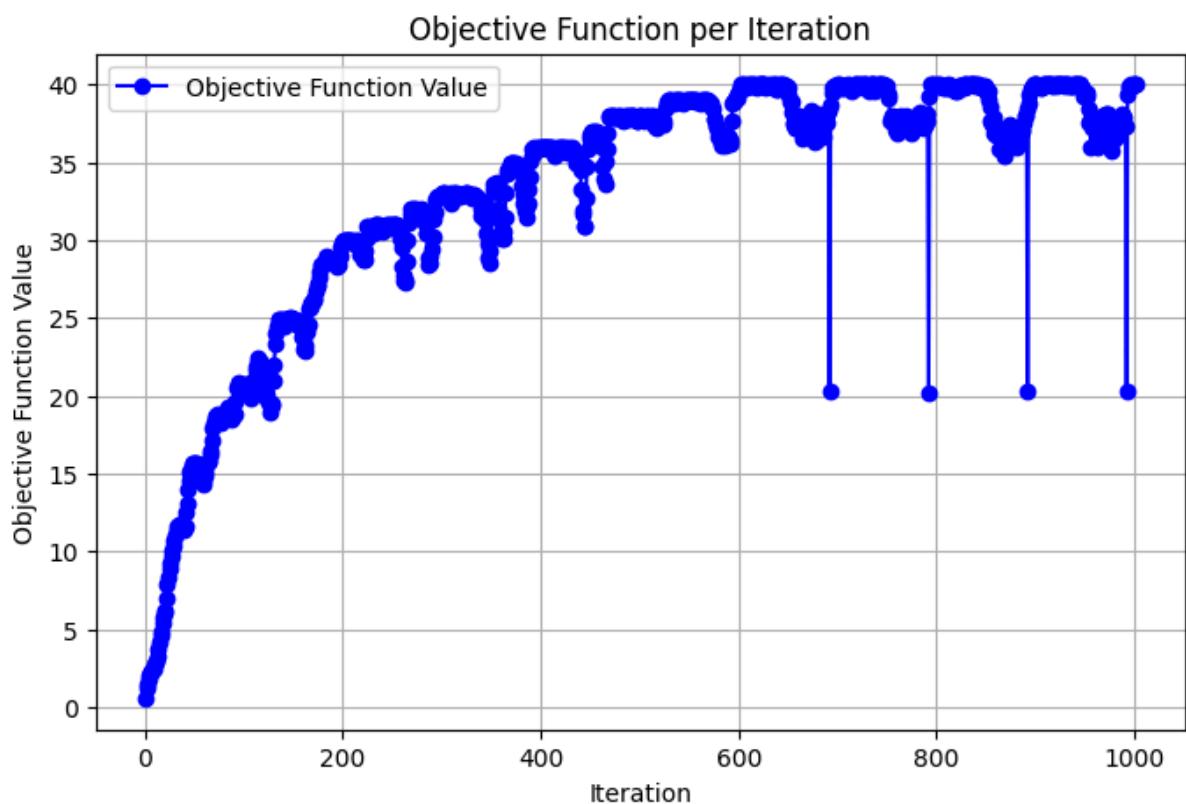
Matrix Level 1					Matrix Level 2					Matrix Level 3				
71	40	5	120	99	73	25	30	26	36	119	50	95	108	124
29	122	86	62	16	113	112	94	64	70	63	32	98	8	114
23	54	57	79	102	81	41	37	82	74	4	109	75	7	45
123	43	96	52	1	80	17	88	110	12	14	66	38	24	67
91	56	69	2	97	28	46	115	104	22	42	116	20	72	65
Matrix Level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

Nilai Objective Function: 40

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 300

Banyak Iterasi: 1000

Durasi Proses Pencarian: 63 Second

2.2.6.17. Percobaan 17

State Awal

3D Cube Matrix Layers

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

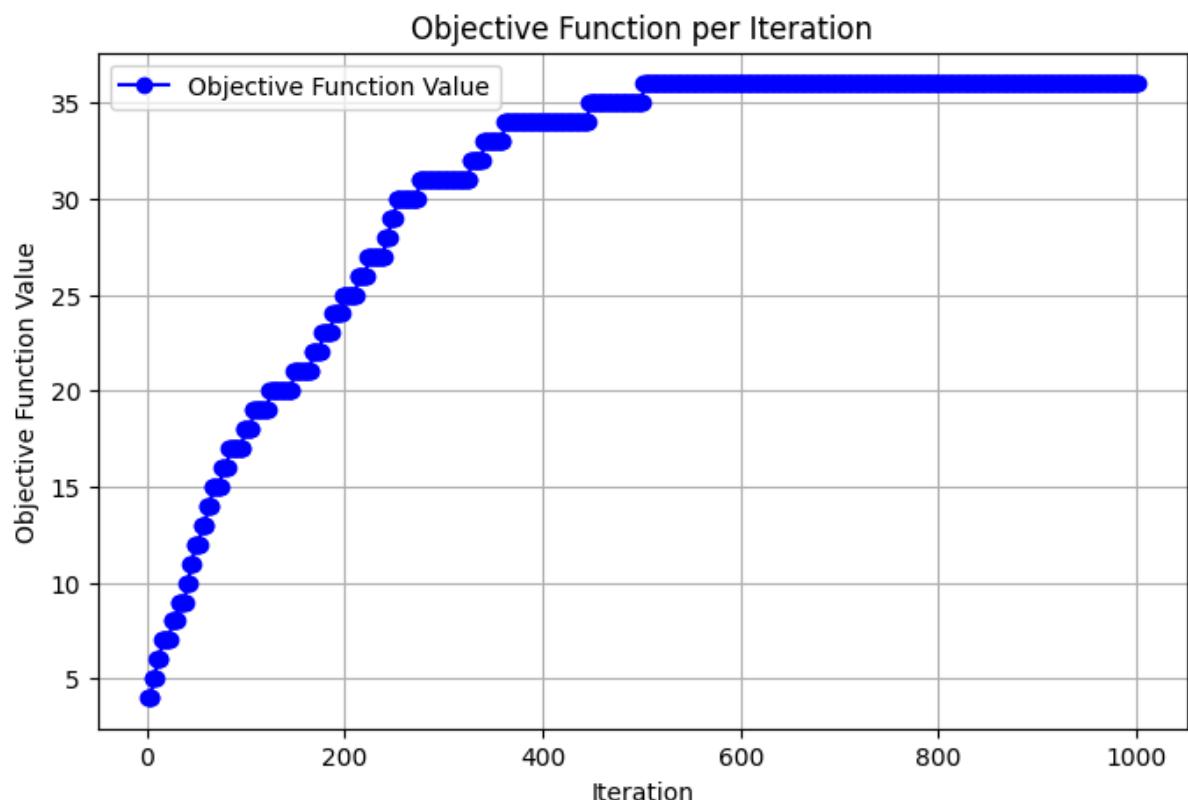
State Akhir

3D Cube Matrix Layers

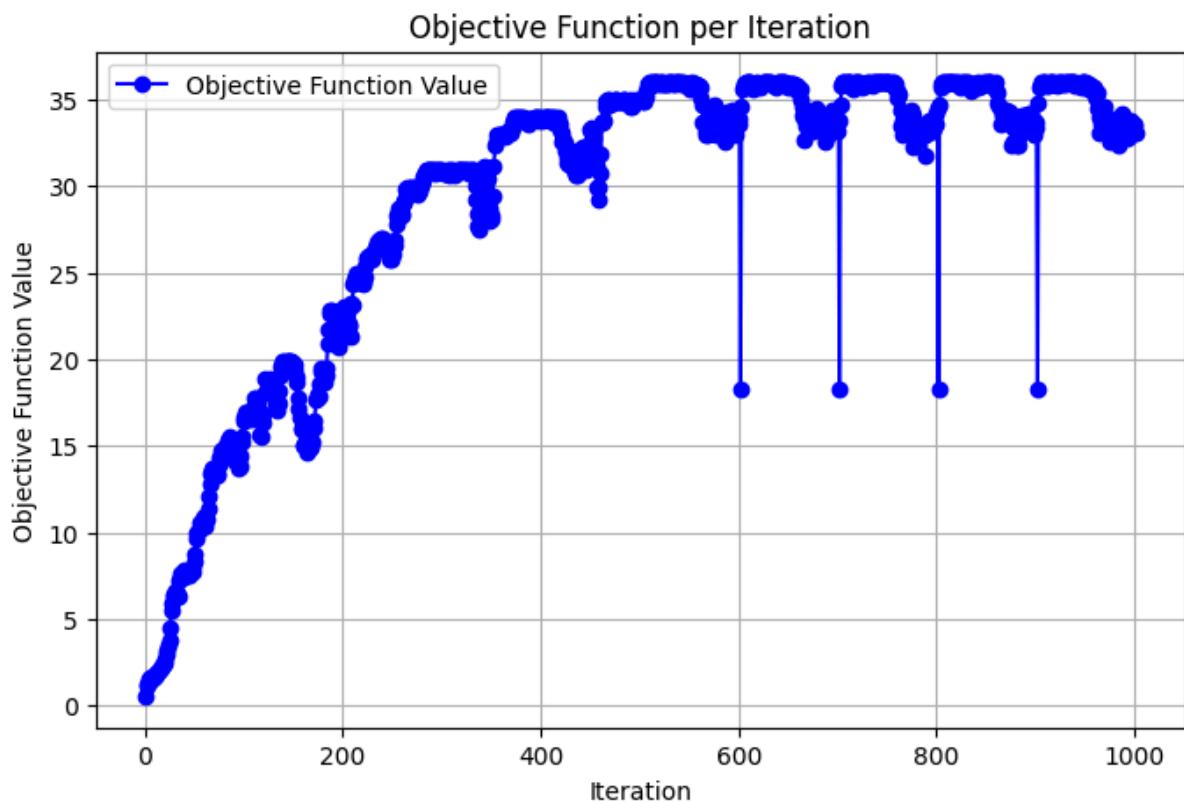
Matrix Level 1					Matrix Level 2					Matrix Level 3				
16	102	12	64	2	68	78	58	88	23	5	13	55	24	123
33	91	60	117	14	95	86	81	28	57	25	43	3	21	31
79	96	72	122	100	105	1	38	52	119	56	40	89	51	84
87	11	80	45	92	70	44	109	77	73	8	37	20	103	39
99	15	26	22	107	10	106	29	98	94	30	18	76	116	75
Matrix Level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

Nilai Objective Function: 36

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 300

Banyak Iterasi: 1000

Durasi Proses Pencarian: 63 Second

2.2.6.18. Percobaan 18

State Awal

3D Cube Matrix Layers

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

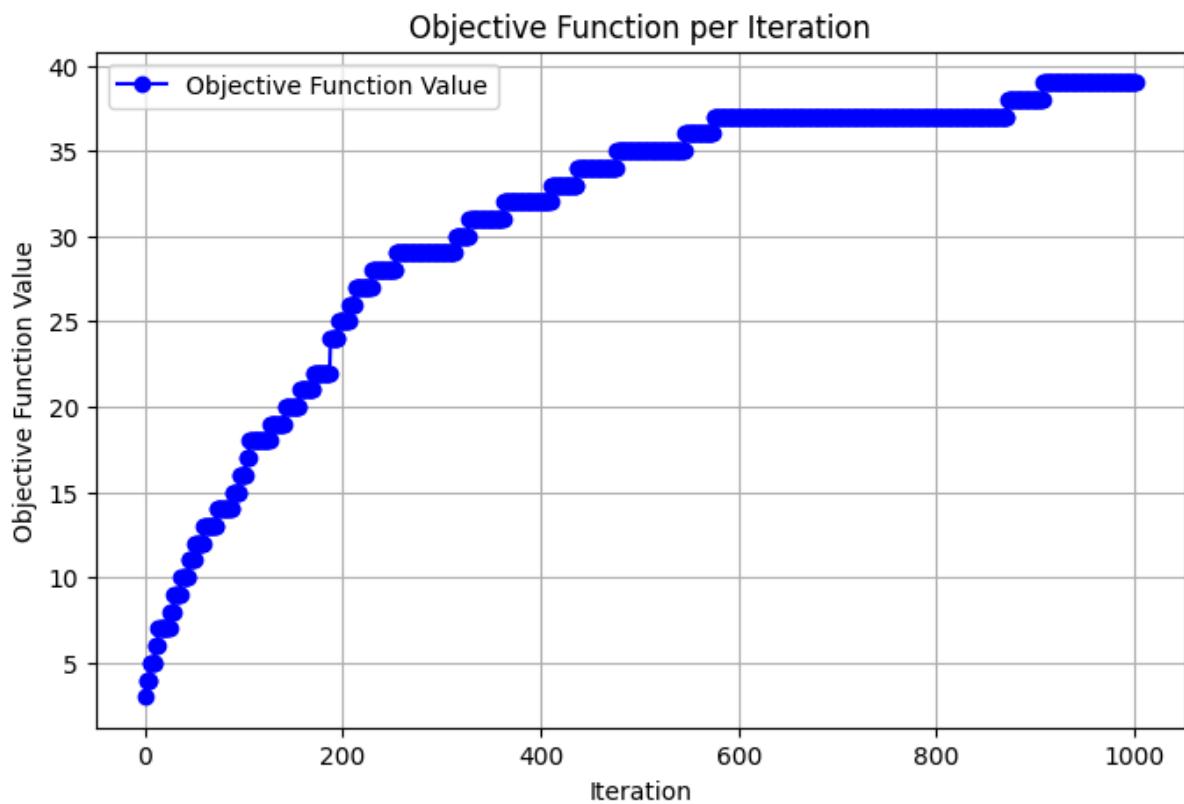
State Akhir

3D Cube Matrix Layers

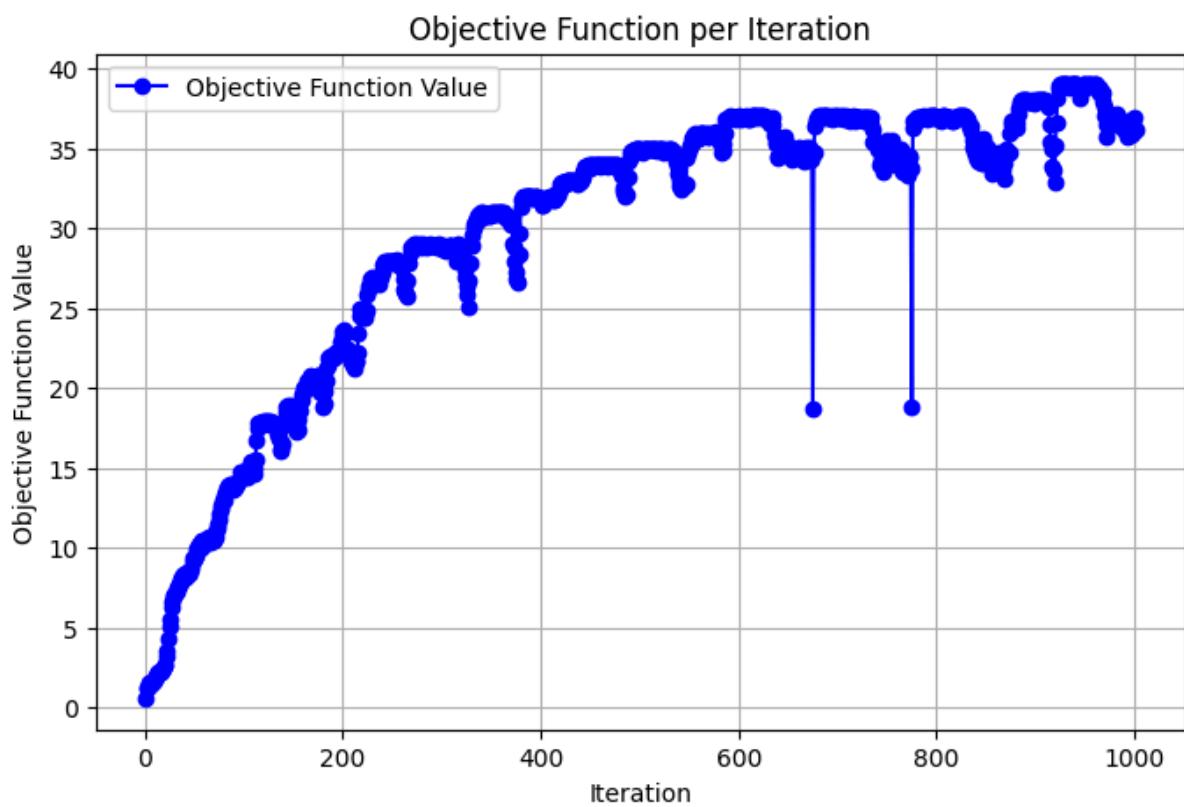
Matrix Level 1					Matrix Level 2					Matrix Level 3				
19	98	48	1	94	8	65	125	81	36	45	22	15	50	13
74	69	64	96	12	71	105	79	83	115	5	2	56	41	9
100	11	70	52	82	28	40	39	124	84	24	86	23	109	62
18	32	34	111	120	75	31	44	59	106	119	116	43	57	121
104	42	99	37	33	123	27	78	73	14	122	89	93	58	110
Matrix Level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

Nilai Objective Function: 39

Plot Nilai Objective Function Max:



Plot Nilai Objective Function Avg:



Jumlah Populasi: 300

Banyak Iterasi: 1000

Durasi Proses Pencarian: 62 Second

BAB 3

Analisis

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

Masing - masing algoritma memiliki kemampuan mendekati global optima yang berbeda - beda. Perlu diketahui bahwa global optima memiliki nilai *objective function/fitness value* yakni 109. Berikut merupakan ringkasan hasil yang didapatkan :

1. *Genetic Algorithm* : berdasarkan hasil analisis penulis, algoritma ini cenderung memiliki nilai *fitness value* yang kecil dibandingkan dengan algoritma lainnya, tetapi algoritma ini mampu menghindari solusi lokal dengan baik dengan menggunakan metode *crossover* dan *mutation*. Alasan algoritma ini memiliki *fitness value* yang sedikit jauh dari global optima adalah maksimum iterasinya. Algoritma ini memiliki parameter maksimum iterasi yang menyebabkan tidak dicapainya global optima walaupun mampu menghindari solusi lokal. Alasan lain yang menghambat algoritma ini adalah kebutuhan komputasionalnya yang berat dikarenakan perlunya membuat populasi baru untuk setiap iterasi.
2. *Simulated Annealing* : Dari antara algoritma lain yang penulis analisis, algoritma ini merupakan algoritma yang paling mungkin mendekati global optima. Algoritma *simulated annealing* memiliki kemampuan untuk menghindari solusi lokal karena algoritma ini memperbolehkan pemilihan *state* yang lebih buruk berdasarkan *temperature* dan *schedule*. Algoritma ini juga tidak berat secara komputasional sehingga dapat mencari solusi dengan cepat dan cukup dekat dengan global optima.
3. *Steepest Ascent Hill-climbing* : Berdasarkan analisis penulis, algoritma ini menghasilkan solusi yang cukup dekat dengan global optima dengan *fitness value* yang besar, hanya saja algoritma ini cenderung terjebak dalam solusi lokal dibandingkan dengan algoritma lainnya.
4. *Hill-climbing with sideways move* : Berdasarkan analisis penulis, algoritma ini menghasilkan solusi yang cukup dekat dengan global optima dibandingkan dengan *Steepest Ascent Hill-climbing* karena algoritma ini menghindari solusi lokal dengan memperbolehkan pemilihan suksesor dengan nilai yang sama dengan *current cube*. Hal ini memberikan algoritma sedikit lebih fleksibilitas dalam memilih suksesor dibandingkan dengan *Steepest Ascent Hill-climbing*.

5. *Random Restart Hill-climbing* : Berdasarkan analisis penulis, algoritma ini menghasilkan solusi yang cukup dekat dengan global optima dengan *fitness value* yang cukup besar serta algoritma ini dapat menghindari solusi lokal dengan cara mengulangi algoritma *hill climbing* dengan *initial state* yang berbeda, hanya saja algoritma ini tidak sebaik *Simulated Annealing* dalam proses menghindari solusi lokal tersebut.
6. *Stochastic Hill-climbing* : Berdasarkan analisis penulis, algoritma ini menghasilkan solusi yang cukup **jauh** dengan global optima dengan *fitness value* yang cukup rendah. Hal ini disebabkan cara algoritma memilih *cube* berikutnya sebagai suksesor yakni secara *random*.

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

Jika kita bandingkan algoritma pencarian local search yang kita pakai, dapat disimpulkan beberapa hal

1. Steepest Ascent Hill-climbing : Menghasilkan solusi dengan *fitness value* yang relatif lebih besar dibandingkan *Stochastic Hill-climbing* dan *genetic algorithm*.
2. Sideways Move Hill Climbing : Menghasilkan solusi dengan *fitness value* yang relatif lebih besar dibandingkan *Stochastic Hill-climbing*, *genetic algorithm*, *Steepest Ascent Hill-climbing*, dan *Random Restart Hill-climbing*.
3. Random Restart Hill Climbing : Menghasilkan solusi dengan *fitness value* yang relatif lebih besar dibandingkan *Stochastic Hill-climbing*, *genetic algorithm*, dan *Steepest Ascent Hill-climbing*.
4. Stochastic Hill Climbing : Menghasilkan solusi dengan nilai *fitness value* yang paling rendah dibandingkan dengan algoritma yang lain.
5. Simulated Annealing : Menghasilkan solusi dengan nilai *fitness value* yang paling besar dibandingkan dengan algoritma yang lain.
6. Genetic Algorithm : Menghasilkan solusi dengan *fitness value* yang relatif lebih besar dibandingkan *Stochastic Hill-climbing*.

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

1. Steepest Ascent Hill Climbing : Algoritma ini memiliki durasi yang lumayan cepat dibandingkan yang lain yaitu dari hasil rata-rata memiliki durasi 30 detik
2. Sideways Move Hill Climbing : Algoritma ini memiliki durasi yang lumayan cepat dibandingkan yang lain yaitu dari hasil rata-rata memiliki durasi 30 detik
3. Random Restart Hill Climbing : Algoritma ini memiliki durasi yang lumayan lama dibandingkan yang lain yaitu dari hasil rata-rata memiliki durasi 110 detik.

4. Stochastic Hill Climbing : Algoritma ini memiliki durasi paling cepat dengan hasil dari eksperimen mendapatkan durasi rata-rata 0.3 detik.
5. Simulated Annealing : Algoritma ini memiliki durasi yang lumayan cepat dibandingkan yang lain yaitu dari hasil rata-rata memiliki durasi 16 detik.
6. Genetic Algorithm : Algoritma ini memiliki durasi yang cukup bervariatif tergantung parameter yang dimasukkan, mulai dari 1 detik hingga 60 detik.

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

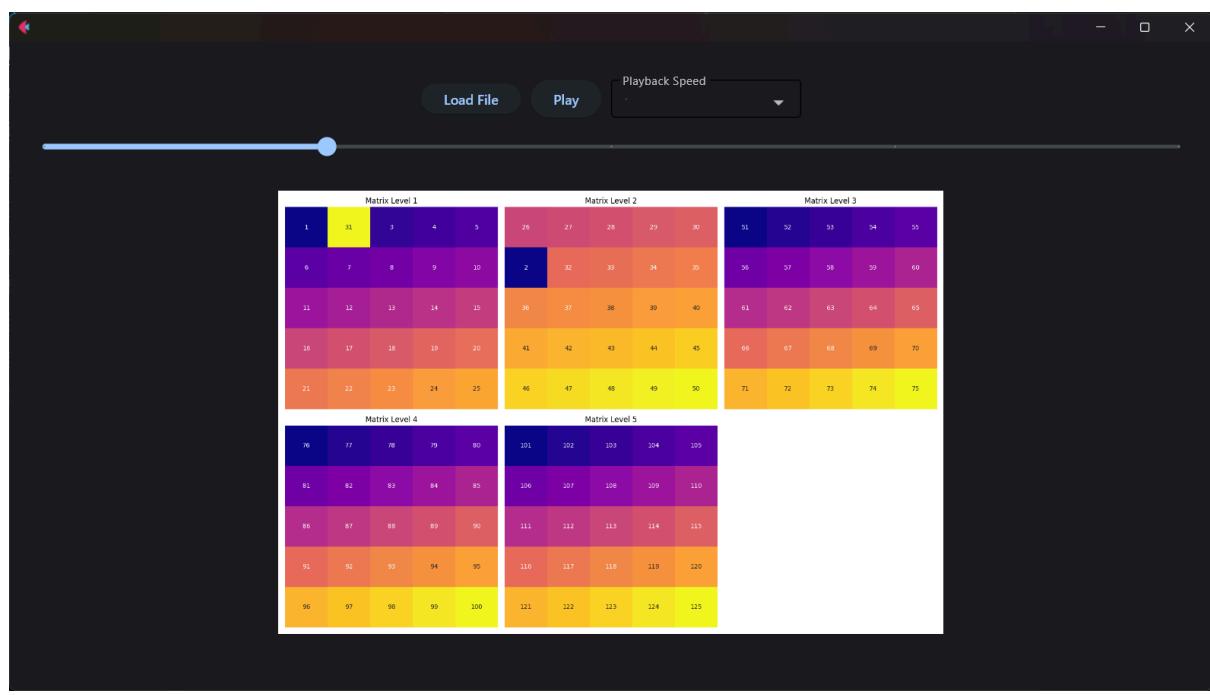
1. Steepest Ascent Hill Climbing : Berdasarkan eksperimen algoritma ini menghasilkan hasil yang cukup konsisten dengan state akhir yang memiliki value 40-44.
2. Sideways Move Hill Climbing : Berdasarkan eksperimen algoritma ini menghasilkan hasil yang cukup konsisten dengan state akhir yang memiliki value 41-45.
3. Random Restart Hill Climbing : Berdasarkan eksperimen algoritma ini menghasilkan hasil yang sedikit tidak konsisten dengan state akhir yang memiliki value 35-44.
4. Stochastic Hill Climbing : Berdasarkan eksperimen algoritma ini menghasilkan hasil yang tidak konsisten dengan state akhir yang memiliki value 8-19.
5. Simulated Annealing : Berdasarkan eksperimen algoritma ini menghasilkan hasil yang cukup konsisten dengan state akhir yang memiliki value 50-53.
6. Genetic Algorithm : Berdasarkan eksperimen algoritma ini menghasilkan hasil yang cukup konsisten dengan state akhir yang memiliki value tergantung parameter namun dari hasil eksperimen cukup konsisten dalam parameter yang sama.

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

Berdasarkan hasil percobaan, didapatkan bahwa semakin banyak jumlah iterasi maka nilai *objective function* yang dihasilkan juga semakin besar. Sedangkan untuk jumlah populasi yang digunakan tidak begitu berpengaruh terlalu besar, tetapi tetap terdapat korelasi antara nilai *objective function* dengan jumlah populasi.

BAB 4

GUI



Fitur:

- Load File: membaca file json berisi state hasil eksperimen.
- Play and Pause: memulai dan memberhentikan progress secara otomatis
- Drag and Drop: menampilkan visualisasi state yang dituju dengan meng-*drag and drop* progress bar
- Playback speed: dapat mengubah speed dengan pilihan 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, dan 128.0 dengan pendefinisian speed sebagai 1/speed terpilih.

BAB 5

Kesimpulan

Berdasarkan percobaan dari kode yang telah dibuat, dapat disimpulkan bahwa pencarian dengan algoritma *local search* yang bervariasi mulai dari *Hill Climbing* hingga *Genetic Algorithm* memiliki kelebihan dan kekurangannya masing-masing. Pemilihan algoritma akan mempengaruhi durasi yang didapatkan dan juga state value yang dihasilkan. Selain itu juga parameter yang digunakan untuk setiap algoritma juga berpengaruh terhadap nilai *objective function* yang dihasilkan. Berdasarkan eksperimen yang kita lakukan juga, dapat disimpulkan juga bahwa algoritma *local search* dengan hasil terbaik dan waktu yang tidak terlalu lama adalah ***Simulated Annealing***.

LAMPIRAN

Repository: https://github.com/Zechtro/Tubes1_AI

Pembagian Tugas

NIM	Responsibilities
13522061	Steepest Ascent HC, HC with Sideways Move, Random Restart HC, GUI, Report
13522075	Steepest Ascent HC, HC with Sideways Move, Random Restart HC, Stochastic HC, Simulated Annealing, Genetic Algorithm, Report
13522083	Steepest Ascent HC, HC with Sideways Move, Random Restart HC, GUI (bonus), Report
13522103	Visualisation, Genetic Algorithm, Report
13521154	Simulated Annealing, Stochastic HC, Report

DAFTAR PUSTAKA

Institut Teknologi Bandung. (n.d.). LMS Indonesia.

<https://edunex.itb.ac.id/courses/44315/preview/137704>

Russell, Stuart J., and Peter Norvig. 2021. *Artificial Intelligence: A Modern Approach - Global Edition*. 4th ed. Harlow, United Kingdom: Pearson.

“Features of the Magic Cube.” n.d. Magisch Vierkant.

<https://www.magischvierkant.com/three-dimensional-eng/magic-features/>.

“Perfect Magic Cubes.” n.d.

<https://www.trump.de/magic-squares/magic-cubes/cubes-1.html>.

Wikipedia contributors. 2024. “Magic Cube.” Wikipedia. May 27, 2024.

https://en.wikipedia.org/wiki/Magic_cube.