

Tugas Besar 1 :

Pencarian Solusi Diagonal Magic Cube

dengan Local Search

IF3070
Dasar Inteligensi Artifisial



Disusun Oleh :
Kelompok 51

Moh Afnan Fawaz	18222111
Aqila Ataa	18222120
Gymnastiar Anwar	18222121
Fadian Alif Mahardika	18222124

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

BAB I

DESKRIPSI PERSOALAN

25	16	80	104	90
115	98	4	1	97
42	111	85	2	75
66	72	27	102	48
67	18	119	106	5
67	18	119	106	5
116	17	14	73	95
40	50	81	65	79
56	120	55	49	35
36	110	46	22	101

Gambar 1.1 *Diagonal Magic Cube* 5x5x5

Diagonal Magic Cube adalah sebuah kubus yang tersusun dari angka 1 hingga n^3 tanpa pengulangan dengan n adalah panjang sisi pada kubus tersebut. Angka angka yang terdapat pada kubus tersebut tersusun sedemikian rupa sehingga memenuhi beberapa properti, yaitu :

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

Pada permasalahan ini, *Diagonal Magic Cube* yang akan kita selesaikan berukuran 5x5x5. Sehingga, nilai n nya merupakan 5 dan initial state dari *Diagonal Magic Cube* ini merupakan susunan angka dari 1 hingga 5^3 secara acak. Perlu

diterapkan algoritma *local search* untuk menyelesaikan permasalahan ini. Tiap iterasi pada algoritma *local search*, langkah yang boleh dilakukan adalah menukar posisi dari 2 angka pada kubus tersebut dan 2 angka yang ditukar tidak harus bersebelahan.

BAB II

PEMBAHASAN

2.1 Pemilihan Objective Function

Objective function adalah fungsi yang memetakan suatu solusi x ke dalam nilai numerik yang menggambarkan kualitas solusi tersebut atau secara singkat *objective function* adalah fungsi untuk menentukan nilai suatu *state* pada suatu permasalahan untuk menentukan kualitas suatu solusi. *Objective function* sering digunakan untuk mengukur kualitas solusi dan membantu dalam pencarian ruang solusi yang sangat besar dengan cara meminimalkan penyimpangan dari target yang diinginkan.

Pada konteks *diagonal magic cube*, kita meminimalkan kesalahan pada susunan angka di baris, kolom, tiang, diagonal, dan diagonal ruang yang seharusnya menghasilkan *magic number*. *Magic number* itu sendiri memiliki rumus sebagai berikut:

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

dengan n adalah 5 (*diagonal magic cube* 5x5x5) dan M adalah *magic number*. Dari rumus itu didapatkan bahwa *magic number* *diagonal magic cube* 5x5x5 adalah 315. Dari sini dapat didapatkan *objective function* untuk *diagonal magic cube* 5x5x5.

$$f(x) = (315 \times 109) - (S_{\text{baris}}(x) + S_{\text{kolom}}(x) + S_{\text{tiang}}(x) + S_{\text{diagonal bidang}}(x) + S_{\text{diagonal ruang}}(x))$$

Dimana pada state x , $f(x)$ adalah *objective function*, $S_{\text{baris}}(x)$ adalah total penjumlahan baris dari semua sisi pada *diagonal magic cube*, $S_{\text{kolom}}(x)$ adalah total penjumlahan kolom dari semua sisi pada *diagonal magic cube*, $S_{\text{tiang}}(x)$ adalah total penjumlahan semua tiang pada *diagonal magic cube*, $S_{\text{diagonal bidang}}(x)$ adalah total penjumlahan diagonal bidang dari semua sisi pada *diagonal magic cube*, dan $S_{\text{diagonal ruang}}(x)$ adalah total penjumlahan semua diagonal ruang pada *diagonal magic cube*. *Objective function* ini dipilih karena tujuan dari *diagonal magic cube* ini sendiri adalah untuk memiliki kesalahan sesedikit mungkin, selain itu 315 pada *objective*

function merupakan *magic number* dan 109 merupakan total dari seluruh baris (25), kolom (25), tiang (25), diagonal bidang (30), dan diagonal ruang (4) pada *diagonal magic cube* 5x5x5. Maka dari itu, pada algoritma *local search* iterasi akan melakukan peminimalan kesalahan hingga mencapai 0, dimana artinya tidak ada kesalahan terjadi dan *diagonal magic cube* ini valid.

2.2 Penjelasan Implementasi Algoritma Local Search (Berisi Deskripsi Fungsi/Kelas Beserta Source Codenya)

2.2.1 Magic Cube

Magic Cube merupakan kelas yang berisi fungsi-fungsi untuk digunakan saat mencari solusi menggunakan algoritma *local search*. Kelas ini dibuat untuk memudahkan dalam memanggil fungsi-fungsi yang akan digunakan berkali-kali di fungsi-fungsi algoritma *local search*. Berikut merupakan fungsi-fungsi yang ada pada kelas *Magic Cube*.

a. Fungsi `__init__`

Deskripsi	Fungsi untuk inisialisasi <i>Magic Cube</i>
Proses	Menginisialisasi <i>n</i> , <i>magic number</i> , <i>total elements</i> , <i>target value</i> , dan <i>cube</i>
<pre>def __init__(self, n): self.n = n self.magic_number = 315 self.total_elements = 109 self.target_value = self.magic_number * self.total_elements self.cube = self.generateRandomStates()</pre>	

b. Fungsi `generateRandomStates`

Deskripsi	Fungsi untuk melakukan <i>generate initial state</i> <i>Magic Cube</i> 5x5x5
-----------	--

Proses	Mengembalikan <i>random magic cube</i> 5x5x5
<pre>def generateRandomStates(self): numbers = np.arange(1, self.n ** 3 + 1) np.random.shuffle(numbers) random_cube = numbers.reshape((self.n, self.n, self.n)) return random_cube</pre>	

c. Fungsi calculateElements

Deskripsi	Fungsi untuk menghitung total angka dari tiap elemen elemen <i>magic cube</i> (baris, kolom, tiang, diagonal bidang, dan diagonal ruang)
Proses	Mengembalikan hasil total angka dari tiap elemen pada <i>magic cube</i> sebagai list
<pre>def calculateElements(self): Sbarisx = [] Skolomx = [] Stiangx = [] Sdiagonalbidangx = [] Sdiagonalruangx = [] for layer in range(self.n): for row in range(self.n): Sbarisx.append(np.sum(self(cube[layer, row, :]))) for layer in range(self.n): for col in range(self.n): Skolomx.append(np.sum(self(cube[layer, :, col]))) for row in range(self.n): for col in range(self.n): Stiangx.append(np.sum(self(cube[:, row, col]))) for layer in range(self.n):</pre>	

```

        diag1 = np.sum([self.cube[layer, i, i] for i in
range(self.n)])
        diag2 = np.sum([self.cube[layer, i, self.n - i - 1] for i
in range(self.n)])
        Sdiagonalbidangx.extend([diag1, diag2])

Sdiagonalruangx.extend([
    np.sum([self.cube[i, i, i] for i in range(self.n)]),
    np.sum([self.cube[i, i, self.n - i - 1] for i in
range(self.n)]),
    np.sum([self.cube[i, self.n - i - 1, i] for i in
range(self.n)]),
    np.sum([self.cube[i, self.n - i - 1, self.n - i - 1] for i in
range(self.n)])
])
return Sbarisx + Skolomx + Stiangx + Sdiagonalbidangx +
Sdiagonalruangx

```

d. Fungsi objectiveFunction

Deskripsi	Fungsi ini berisi rumus <i>objective function</i> yang digunakan oleh kelompok kami pada kasus ini
Proses	Mengembalikan hasil perhitungan <i>objective function</i>
<pre> def objectiveFunction(self): totalSums = self.calculateElements() fx = sum(abs(sums - self.magicnumber) for sums in totalSums) print("Objective function:", fx) return fx </pre>	

e. Fungsi swapElements

Deskripsi	Fungsi untuk melakukan <i>swap</i> angka dan memastikan bahwa angka yang di <i>swap</i> hanya dua angka
Proses	Mengembalikan <i>cube</i> yang telah di <i>swap</i> angka

```

def swapElements(self, pos1, pos2):
    x1, y1, z1 = pos1
    x2, y2, z2 = pos2
    self.cube[x1, y1, z1], self.cube[x2, y2, z2] = self.cube[x2,
y2, z2], self.cube[x1, y1, z1]

```

f. Fungsi isValid

Deskripsi	Fungsi untuk memastikan bahwa <i>magic cube</i> merupakan <i>magic cube</i> yang valid
Proses	Mengembalikan keterangan jika <i>objective function</i> = 0 maka <i>magic cube</i> valid
<pre> def isValid(self): return self.objectiveFunction() == 0 </pre>	

g. Fungsi nextStates

Deskripsi	Fungsi untuk menentukan <i>successor</i> atau <i>neighbor states</i> yang ada
Proses	Mengembalikan nilai <i>next states</i>
<pre> def nextStates(self): nextstates = [] for x1 in range(self.n): for y1 in range(self.n): for z1 in range(self.n): for x2 in range(self.n): for y2 in range(self.n): for z2 in range(self.n): if (x1, y1, z1) != (x2, y2, z2): newCube = MagicCube(self.n) newCube.cube = np.copy(self.cube) newCube.swapElements((x1, y1, z1), (x2, y2, z2)) nextstates.append(new_cube) </pre>	

```
    return nextstates
```

h. Fungsi visualize

Deskripsi	Fungsi untuk memvisualisasikan <i>magic cube</i>
Proses	Mengembalikan visualisasi <i>magic cube</i> berupa 5 <i>layer</i> dengan berisi angka angka berbeda

```
def visualize(self):  
    fig, axes = plt.subplots(1, self.n, figsize=(3 * self.n, 3))  
    fig.suptitle('Visualisasi Diagonal Magic Cube', fontsize=16)  
  
    for layer in range(self.n):  
        ax = axes[layer]  
        ax.imshow(self.cube[layer], cmap='Pastell',  
interpolation='nearest')  
        ax.set_title(f'Layer {layer + 1}')  
        ax.axis('off')  
  
        for i in range(self.n):  
            for j in range(self.n):  
                ax.text(j, i, str(self.cube[layer][i, j]),  
ha='center', va='center', color='black')  
  
    plt.tight_layout()  
    plt.show()
```

2.2.2 Steepest Ascent Hill-climbing

Steepest Ascent Hill-climbing adalah suatu algoritma *local search* yang dimana algoritma ini akan terus mencari *local maximum* tertinggi dari *initial state*. Pada kasus *diagonal magic cube*, algoritma akan dimulai dengan *initial state* yang *random* dan terus melakukan iterasi untuk meminimalisir kesalahan di *magic number* pada *diagonal magic cube* tersebut hingga mencapai 0 kesalahan, namun setelah melakukan iterasi dan apabila *diagonal magic cube*

masih memiliki kesalahan tetapi sudah tidak bisa di minimalisir (sudah mencapai “*peak*” atau terjebak dalam “*flat local maximum*”), algoritma akan berhenti mencari meskipun solusi yang didapatkan belum optimal. Algoritma ini tergolong cepat dan tidak membutuhkan banyak sumber daya komputasi karena tidak melakukan eksplorasi mendalam, namun risiko berhenti di *local maximum* membuatnya kurang ideal dalam menemukan solusi optimal secara konsisten. Berikut merupakan fungsi fungsi yang digunakan.

a. Fungsi hillClimbingSA (cube, maxIter)

Deskripsi	Fungsi untuk menjalankan <i>local search hill-climb steepest ascent</i> dengan menerima 2 parameter, cube untuk kubus dan maxIter untuk total iterasi
Proses	<i>Local search</i> ini akan mencari state terbaik dengan memilih value yang lebih besar daripada <i>state</i> sekarang dan akan mengeluarkan hasilnya bila tidak ada lagi yang lebih besar atau iterasinya yang ditentukan sudah habis

```

def steepest_ascent_hill_climbing(cube, maxIter=50):
    iteration = 0
    history = [] # Setiap ditemukan objective function baru yang lebih baik, maka akan dievaluasi
    timeStart = time.time()

    # Menyimpan nilai objective function awal
    current_objfunc = cube.objectiveFunction()
    history.append(current_objfunc)

    # SteepestAscent HillClimb, terdapat 3 kondisi
    # 1. Jika iterasi sudah mencapai maksimal, mengeluarkan kondisi current state
    while iteration < maxIter:
        nextStates = cube.nextstates()

```

```

bestStates = None
best_objfunc = current_objfunc

# Mencari state terbaik dari neighbor
for nextstate in nextStates:
    neighbor_objfunc = nextstate.objectiveFunction()
    print(f"Iterasi {iteration + 1}: Current: {current_objfunc}, Neighbor: {neighbor_objfunc}")

    if neighbor_objfunc < best_objfunc:
        bestStates = nextstate
        best_objfunc = neighbor_objfunc

# Jika menemukan state yang lebih baik, perbarui state
if bestStates:
    cube = bestStates
    current_objfunc = best_objfunc
    history.append(current_objfunc)

# 2. Jika tidak menemukan state yang lebih baik,
# keluar dari loop search dan keluarkan hasil current
state
else:
    print("Tidak ada solusi yang lebih baik, berhenti pada local maximum")
    break

iteration += 1

# 3. Jika solusi valid ditemukan, berhenti
if cube.is_valid_magic_cube():
    print(f"Solusi ditemukan setelah {iteration} iterasi.")
    break

duration = time.time() - timeStart
print(f"Final State: {current_objfunc}")
return cube, history, iteration, duration

```

b. Fungsi `visualize_experiment(initial_cube, final_cube, history)`

Deskripsi	Fungsi untuk memberikan gambaran secara visual untuk initial state dan final state dari hasil eksperimen penggunaan <i>local search steepest ascent hill climb</i>
Proses	Mengembalikan visualisasi <i>magic cube</i> berupa 5 <i>layer</i> dengan berisi angka-angka berbeda
<pre># Visualisasi dari kode def visualize_experiment(initial_cube, final_cube, history): print("Initial State:") initial_cube.visualize() print("Final State:") final_cube.visualize() plt.plot(range(len(history)), history, color='blue') plt.title('Objective Function terhadap Banyak Iterasi') plt.xlabel('Iterasi') plt.ylabel('Objective Function (f(x))') plt.grid(True) plt.show()</pre>	

2.2.3 Hill-climbing with Sideways Move

Hill-climbing with Sideways Move pada kasus *diagonal magic cube* algoritma ini akan dimulai dengan *initial state* yang *random* dan terus melakukan iterasi untuk meminimalisir kesalahan di *magic number* pada *diagonal magic cube* hingga mencapai 0 kesalahan. Algoritma *Hill-climbing with Sideways Move* memiliki tingkat kesuksesan dalam memecahkan masalah yang lebih tinggi dari algoritma *Steepest Ascent Hill-climbing*, karena algoritma ini hanya akan berhenti apabila sudah mencapai “peak” atau kesalahan pada *magic number* sama dengan 0. Meskipun lebih efektif dalam menangani “flat”

local maximum, algoritma ini hanya membutuhkan sedikit tambahan komputasi dibandingkan *Steepest Ascent Hill-climbing* dan tetap efisien. Berikut merupakan fungsi-fungsi yang digunakan.

a. Fungsi `hillClimbingWSM(cube, maxIter, maxSideways)`

Deskripsi	Fungsi untuk menjalankan local search hill-climb sideways move dengan menerima 3 parameter, cube untuk kubus, maxIter untuk total iterasi, maxSideways untuk total iterasi melakukan gerakan <i>sideways</i>
Proses	<i>Local search</i> ini akan mencari state terbaik dari <i>neighbor</i> -nya dengan memilih <i>value</i> yang lebih besar daripada <i>state</i> sekarang. Namun apabila tidak ada yang lebih besar, akan memilih <i>value</i> yang sama dengan <i>state</i> sekarang lalu mengambil <i>state</i> tersebut untuk mencoba untuk mencari <i>value</i> yang lebih bagus

```

def hillClimbingWSM(cube, maxIter, maxSideways):
    iteration = 0
    sidewaysCount = 0
    history = []
    timeStart = time.time()

    # Menyimpan nilai objective function awal
    current_objfunc = cube.objectiveFunction()
    history.append(current_objfunc)

    # Terdapat 3 kondisi untuk hasil dari SidewaysMove
    # 1. Jika iterasi sudah mencapai maksimal, mengeluarkan kondisi
    current_state
    while iteration < maxIter:
        nextStates = cube.nextStates()
        bestStates = None
        bestNeighbor_objfunc = current_objfunc

```

```

# Mencari state terbaik di neighbor
for next_state in nextStates:
    neighbor_objfunc = next_state.objectiveFunction()
    print(f"Iterasi {iteration}: Current: {current_objfunc},
Neighbor: {neighbor_objfunc}")

    if neighbor_objfunc < bestNeighbor_objfunc:
        bestStates = next_state
        bestNeighbor_objfunc = neighbor_objfunc

# Jika menemukan neighbor yang lebih baik, perbarui state
if bestStates:
    cube = bestStates
    current_objfunc = bestNeighbor_objfunc
    history.append(current_objfunc)
    sidewaysCount = 0 # Reset sideways move counter
    iteration += 1

# Jika tidak ada neighbor yang lebih baik tetapi ada yang
nilainya sama,
# mengambil salah satu neighbor dengan nilai yang sama lalu
menghitung sideways
elif current_objfunc == bestNeighbor_objfunc and
sidewaysCount < maxSideways:
    # Lakukan *sideways move*
    for next_state in nextStates:
        neighbor_objfunc = next_state.objectiveFunction()
        if neighbor_objfunc == current_objfunc:
            cube = next_state
            sidewaysCount += 1
            print(f"Sideways move #{sidewaysCount} at Iterasi
{iteration}")
            break # Pindah ke state yang sama, lalu
lanjutkan
    iteration += 1 # Tambahkan iterasi setelah sideways move

else:

```

```

        # 2. Jika tidak ada neighbor yang lebih baik atau sama
        # dan sideways limit tercapai, berhenti
        print("Tidak ada solusi yang lebih baik atau sideways
        limit tercapai, berhenti pada local maximum atau plateau")
        break

        # 3. Jika solusi valid ditemukan, berhenti
        if cube.isValid():
            print(f"Solusi ditemukan setelah {iteration} iterasi.")
            break

duration = time.time() - timeStart
print(f"Final State: {current_objfunc}")
return cube, history, iteration, duration

```

b. Fungsi `visualize_experiment(initial_cube, final_cube, history)`

Deskripsi	Fungsi untuk memberikan gambaran secara visual untuk initial state dan final state dari hasil eksperimen penggunaan <i>local search sideways move hill climb</i>
Proses	Mengembalikan visualisasi <i>magic cube</i> berupa 5 <i>layer</i> dengan berisi angka angka berbeda

```

# Visualisasi dari kode
def visualize_experiment(initial_cube, final_cube, history):
    print("Initial State:")
    initial_cube.visualize()

    print("Final State:")
    final_cube.visualize()

    plt.plot(range(len(history)), history, color='blue')
    plt.title('Objective Function terhadap Banyak Iterasi')
    plt.xlabel('Iterasi')
    plt.ylabel('Objective Function (f(x))')
    plt.grid(True)

```

```
plt.show()
```

2.2.4 Random Restart Hill-climbing

Random Restart Hill-climbing adalah salah satu varian dari algoritma *local search hill-climbing*. *Random Restart Hill-climbing* mengadopsi pepatah “If at first you don’t succeed, try, try again.” *Random Restart Hill-climbing* akan mulai dengan initial state yang dibuat secara acak hingga menemukan solusi atau *goal* dan mencapai solusi terbaik, pada konteks *diagonal magic cube* 5x5x5, yaitu mencapai atau mendekati 0 kesalahan. Untuk menemukan solusi pada *diagonal magic cube*, algoritma ini seperti yang sudah disebutkan di awal akan mulai bekerja dengan *initial state* yang random, lalu jika hasil belum maksimal, algoritma ini akan mengulang dari awal dengan *initial state* berbeda namun masih dibuat secara random hingga mencapai ataupun mendekati 0 kesalahan. *Restart* akan dilakukan sebanyak yang kita inginkan namun adapun rumus yang dapat kita gunakan untuk menentukan berapa banyak *restart* harus dilakukan, yakni $1/p$ dimana p adalah peluang didapatkannya solusi terbaik, pada percobaan ini, kita menggunakan $p=0.1$ karena magic cube memiliki ruang eksplorasi yang luas sehingga akan dilakukan pengulangan sebanyak 10 kali. Algoritma ini membutuhkan komputasi lebih tinggi dibandingkan dua algoritma sebelumnya karena memerlukan banyak pengulangan *restart*, namun peluangnya lebih besar untuk menemukan solusi optimal jika diberikan waktu komputasi yang cukup.

- a. randomRestartHC(cubeOrdo, maxRestart, maxIter)

Deskripsi	Fungsi untuk menjalankan local search hill-climb random restart dengan menerima 3 parameter, cubeOrdo untuk memasukkan ordo dari kubus (ini hanya akan menggunakan angka 5 karena kita
-----------	--

	<p>melakukan eksperimen untuk cube 5x5x5), maxRestart untuk jumlah maksimum melakukan restart, dan maxIter untuk jumlah maksimum melakukan iterasi per restartnya</p>
Proses	<p><i>Local search</i> ini melakukan pencarian state yang mirip seperti <i>Steepest Ascent Hill Climb</i>, namun perbedaannya adalah ketika sudah mencapai lokal maksimum, restart akan dilakukan dengan mencoba initial state yang baru dengan harapan mendapatkan lokal maksimum yang lebih mendekati global maksimum</p>
	<pre> def randomRestartHC(cubeOrdo, maxRestart, maxIter): best_cubeFunction = float('inf') best_initialState = None # Menyimpan initial cube terbaik best_finalState = None # Menyimpan final cube terbaik totalIter = 0 history = [] timeStart = time.time() for restart in range(maxRestart): print(f"Restart: {restart + 1}/{maxRestart}") # Membuat cube baru dengan kondisi acak di setiap restart newCube = MagicCube(cubeOrdo) # Membuat cube acak baru current_objfunc = newCube.objectiveFunction() history.append(current_objfunc) initialCube = newCube # Menyimpan cube yang digunakan pada restart ini iteration = 0 while iteration < maxIter: nextStates = newCube.nextStates() </pre>

```

        bestState = None # Untuk menyimpan state terbaik di
setiap iterasi
        bestNeighbor_objfunc = current_objfunc

        # Mencari state terbaik di antara semua next states
        for next_state in nextStates:
            neighbor_objfunc = next_state.objectiveFunction()
            print(f"Iterasi {iteration + 1}: Current:
{current_objfunc}, Neighbor: {neighbor_objfunc}")
            if neighbor_objfunc < bestNeighbor_objfunc:
                bestState = next_state
                bestNeighbor_objfunc = neighbor_objfunc

        if bestState:
            newCube = bestState
            current_objfunc = bestNeighbor_objfunc
            history.append(current_objfunc)

            # Jika tidak menemukan state yang lebih baik, berhenti
pada Local maximum
        else:
            print("Tidak ada solusi yang lebih baik, berhenti
pada local maximum")
            break

        iteration += 1

        # Jika solusi valid ditemukan, berhenti
        if newCube.isValid():
            print(f"Solusi ditemukan setelah {iteration} iterasi
pada restart {restart + 1}")
            best_cubeFunction = current_objfunc
            bestInitialState = initialCube # Simpan initial
cube terbaik
            best_finalState = newCube # Menyimpan final cube
terbaik
            totalIter += iteration
            duration = time.time() - timeStart

```

```

        print(f"Final State: {best_cubeFunction}")
        return best_initialState, best_finalState, history,
totalIter, duration

    # Jika current_objfunc Lebih baik dari best_cubeFunction,
    perbarui cube terbaik
    if current_objfunc < best_cubeFunction:
        best_cubeFunction = current_objfunc
        best_initialState = initialCube # Simpan initial cube
terbaik
        best_finalState = newCube # Simpan final cube terbaik
        totalIter += iteration

duration = time.time() - timeStart
print(f"Final State after Random Restart: {best_cubeFunction}")
return best_initialState, best_finalState, history, totalIter,
duration

```

b. Fungsi `visualize_experiment(initial_cube, final_cube, history)`

Deskripsi	Fungsi untuk memberikan gambaran secara visual untuk <i>initial state</i> dan <i>final state</i> dari hasil eksperimen penggunaan <i>local search random restart hill climb</i>
Proses	Mengembalikan visualisasi <i>magic cube</i> berupa 5 <i>layer</i> dengan berisi angka angka berbeda

```

# Visualisasi dari kode

def visualize_experiment(initial_cube, final_cube, history):
    print("Initial State:")
    initial_cube.visualize()

    print("Final State:")
    final_cube.visualize()

    plt.plot(range(len(history)), history, color='blue')
    plt.title('Objective Function terhadap Banyak Iterasi')

```

```

plt.xlabel('Iterasi')
plt.ylabel('Objective Function (f(x))')
plt.grid(True)
plt.show()

```

2.2.5 Stochastic Hill-climbing

Stochastic Hill-climbing dalam konteks masalah *diagonal magic cube* 5x5x5, kita harus menentukan iterasi maksimum yang akan dilakukan yaitu sebanyak 50, lalu algoritma akan melakukan *looping* dalam mencari hasil solusi terbaik yaitu 0 kesalahan atau ketika tidak ada solusi yang lebih baik, namun dengan tahap penukaran angka yang *random* hingga iterasi memenuhi iterasi maksimum atau ketika tidak ada solusi yang lebih baik. Algoritma ini sedikit lebih berat dibandingkan *hill-climbing* lainnya karena sifat pemilihan random, namun tetap efisien dalam ruang solusi besar dan membutuhkan iterasi lebih tinggi untuk hasil optimal. Berikut merupakan fungsi fungsi yang kami gunakan.

- Fungsi stochasticHC(cube, maxIteration=50)

Deskripsi	Fungsi ini berisi dan akan menjalankan algoritma dari <i>stochastic hill-climbing</i>
Proses	Menerima <i>problem (initial state)</i> dan mengembalikan <i>state yang local maximum</i> , keterangan durasi, <i>cube</i> terbaru, jumlah iterasi, dan histori

```

def stochasticHC(cube, maxIteration=50):
    iteration = 0
    history = []
    startTime = time.time()

    currentFx = cube.objectiveFunction()
    history.append(currentFx)

```

```

while iteration < maxIteration:
    betterMoves = []
    nextstates = cube.nextstates()
    for nextstate in nextstates:
        neighborFx = nextstate.objectiveFunction()

        if neighborFx < currentFx:
            betterMoves.append(nextstate)

    if betterMoves:
        chosenState = random.choice(betterMoves)
        cube = chosenState
        currentFx = cube.objectiveFunction()
        history.append(currentFx)
    else:
        print("Tidak ada solusi yang lebih baik, berhenti pada local maximum")
        break

    iteration += 1

    if cube.isValid():
        print(f"Solusi ditemukan setelah {iteration} iterasi.")
        break

duration = time.time() - startTime
print(f"Final State: {currentFx}")
return cube, history, iteration, duration

```

- b. Fungsi `visualize_experiment(initialCube, finalCube, history)`

Deskripsi	Fungsi ini berfungsi untuk menunjukkan visualisasi dari hasil <i>stochastic hill-climbing</i>
Proses	Menerima <code>initial_cube</code> , <code>final_cube</code> , <code>history</code> dan mengembalikan tampilan <i>5 layer cube</i> pada posisi awal dan posisi akhir juga menampilkan grafik nilai

objective function terhadap banyak iterasi yang telah dilewati

```
def visualize_experiment(initialCube, finalCube, history):
    print("Initial State:")
    initialCube.visualize()

    print("Final State:")
    finalCube.visualize()

    plt.plot(range(len(history)), history, color='blue')
    plt.title('Objective Function terhadap Banyak Iterasi')
    plt.xlabel('Iterasi')
    plt.ylabel('Objective Function (f(x))')
    plt.grid(True)
    plt.show()
```

2.2.6 Simulated Annealing

Simulated Annealing adalah metode *local search* yang lebih berkembang dibandingkan *hill-climbing*. *Simulated Annealing* mengombinasikan *hill-climbing (stochastic hill-climbing)* dengan *random walk (complete search)*. Dalam metalurgi, *annealing* adalah proses yang digunakan untuk melunakkan atau mengeraskan logam dan kaca dengan memanaskannya pada suhu tinggi dan kemudian mendinginkannya secara bertahap, sehingga memungkinkan material mencapai keadaan kristal berenergi rendah. *Simulated Annealing* mengizinkan kita untuk melakukan “*bad*” *moves* atau melakukan pemunduran/penurunan pada nilai hasil dengan cara berpindah kepada *neighbor* yang buruk. *Simulated Annealing* juga memiliki T yaitu suhu yang akan menurun seiring algoritma berjalan, juga memiliki ΔE yaitu *neighbor.value - current.value*, selain itu ada juga probabilitas pada *simulated annealing* dengan rumus $e^{\Delta E/T}$. Ada beberapa aturan pada *simulated annealing* yaitu jika $\Delta E > 0$ (*neighbor* lebih baik) maka algoritma akan berpindah ke *neighbor* sedangkan jika $\Delta E < 0$ akan dua pilihan,

yaitu jika T masih tinggi maka probabilitas akan tinggi juga sehingga algoritma akan tetap berpindah ke *neighbor* walaupun nilainya lebih buruk dari *current value*, sebaliknya, jika T rendah maka algoritma tidak akan berpindah ke *neighbor*. Pada *simulated annealing*, T atau suhu akan kita pilih sendiri titik awalnya dan seberapa cepat turunnya suhu kemudian program akan memperbarui suhu dengan fungsi *schedule* yang sudah ditentukan dari awal, begitu juga dengan batas probabilitas, kita bisa memilih ingin langsung memberikan batas yang statik atau yang acak.

Pada konteks *diagonal magic cube 5x5x5, initial state* akan berupa *random*. Ditentukan juga suhu yang akan digunakan (misalnya $T=100$) dengan pengurangan *geometric cooling* ($T = \alpha \times T$) karena *geometric cooling* menjaga keseimbangan dalam eksplorasi secara bertahap. Setelah itu, kami memberikan batasan probabilitas sebesar 0.5 agar menjaga keseimbangan dalam eksplorasi. algoritma akan melakukan *looping* dalam mencari hasil solusi terbaik yaitu 0 kesalahan hingga $T=0$ atau sudah memenuhi jumlah iterasi, namun dengan tahap penukaran angka yang *random* layaknya *stochastic hill-climbing* akan tetapi pada algoritma ini akan ada “*bad*” moves yang terjadi. Meskipun membutuhkan lebih banyak komputasi karena eksplorasi luas dan pengaturan suhu, *simulated annealing* lebih bagus dalam menghindari jebakan *local maximum*, terutama pada masalah yang kompleks seperti *diagonal magic cube*.

a. Fungsi SimulatedAnnealing(mc, initialTemp, coolingRate, maxIterations)

Deskripsi	Fungsi ini berisi algoritma dari <i>Simulated Annealing</i> dengan menerima parameter magic cube, <i>initial temperature</i> , <i>cooling rate</i> , dan iterasi maksimal
Proses	Local Search ini dimulai dengan melakukan iterasi dan menukar secara acak. Suhunya akan turun seiring iterasi yang memungkinkan untuk menerima solusi yang lebih buruk untuk menghindari lokal optimum.

Fungsi ini akan mengembalikan *bestCube*,
bestObjective, *objectives*, *temperatures*, *deltaEvalues*,
duration

```
# Fungsi Simulated Annealing
def simulatedAnnealing(mc, initialTemp, coolingRate,
maxIterations):

    # Menghasilkan kubus acak sebagai state awal
    currentCube = mc.generateRandomStates()
    mc.cube = currentCube
    currentObjective = mc.objectiveFunction()
    bestCube = currentCube.copy()
    bestObjective = currentObjective

    # List untuk melacak nilai selama iterasi
    temperatures = []
    objectives = []
    deltaEvalues = []

    startTime = time.time()

    for iteration in range(maxIterations):

        # Menghitung suhu untuk iterasi saat ini
        temperature = initialTemp * (coolingRate ** iteration)
        if temperature <= 0:
            break

        # Memilih dua posisi acak dan nge swap menggunakan metode
        swapElements()
        pos1 = (np.random.randint(0, mc.n), np.random.randint(0,
mc.n), np.random.randint(0, mc.n))
        pos2 = (np.random.randint(0, mc.n), np.random.randint(0,
mc.n), np.random.randint(0, mc.n))
        mc.swapElements(pos1, pos2)
```

```

# Evaluasi kubus baru
newObjective = mc.objectiveFunction()
deltaE = newObjective - currentObjective
deltaEvalues.append(np.exp(-deltaE / temperature) if
temperature > 0 else 0)

# Kondisi penerimaan solusi
if deltaE < 0 or np.random.rand() < np.exp(-deltaE /
temperature):
    currentCube = mc.cube.copy()
    currentObjective = newObjective

    # Update kubus terbaik jika ditemukan solusi lebih baik
    if currentObjective < bestObjective:
        bestCube = currentCube.copy()
        bestObjective = currentObjective
    else:

        # Jika solusi baru tidak diterima, balikkan swap untuk
        kembali ke state sebelumnya
        mc.swapElements(pos1, pos2)

    # Menyimpan nilai untuk visualisasi
    temperatures.append(temperature)
    objectives.append(currentObjective)

    # Cek jika sudah magic cube (objective function = 0),
    keluar dari loop
    if mc.isValid():
        print("Magic Cube ditemukan dengan nilai objective
function = 0!")
        break

endTime = time.time()
duration = endTime - startTime

return bestCube, bestObjective, objectives, temperatures,
deltaEvalues, duration

```

b. Fungsi runExperiment()

Deskripsi	Fungsi ini akan menjalankan eksperimen algoritma <i>Simulated Annealing</i> sebanyak 3 kali untuk mencari solusi terbaik pada masalah magic cube
Proses	Akan dilakukan inisialisasi kubus <i>random</i> dan menampilkan visualisasi dari susunan kubus yang acak sebagai state awal. Lalu akan dijalankan algoritma <i>Simulated Annealing</i> . Setelah algoritma selesai, state akhir akan ditampilkan. Hasil akhir yang ditampilkan yaitu: objektif yang dicapai, durasi, plot nilai objektif terhadap iterasi, plot $e^{\frac{\Delta E}{T}}$ terhadap iterasi, dan frekuensi terjebak di local optima

```
# Menjalankan eksperimen 3 kali
def run():
    for run in range(3):
        print(f"\n===== Run {run + 1} =====")

        # Inisialisasi Magic Cube baru untuk setiap run
        mc = MagicCube(n)

        # Menampilkan state awal dari kubus
        print("State Awal Kubus:")
        mc.visualize()

        # Run algoritma Simulated Annealing
        bestCube, bestObjective, objectives, temperatures,
        deltaEvalues, duration = simulatedAnnealing(
            mc, initialTemp, coolingRate, maxIterations
        )
```

```

# Menampilkan state akhir dari kubus
print("State Akhir Kubus:")
mc.cube = bestCube
mc.visualize()

# Menampilkan nilai objective function akhir yang dicapai
print("Nilai Objective Function Akhir:", bestObjective)

# Plot nilai objective function terhadap iterasi
plt.plot(objectives)
plt.title(f"Objective Function terhadap Iterasi (Run {run + 1})")
plt.xlabel("Iterasi")
plt.ylabel("Objective Function")
plt.show()

# Menampilkan durasi proses pencarian
print("Durasi Proses Pencarian:", duration, "detik")

# Plot ( $e^{(\delta E / T)}$ ) terhadap iterasi
plt.plot(deltaValues)
plt.title(f"Plot ( $e^{(\delta E / T)}$ ) terhadap Iterasi (Run {run + 1})")
plt.xlabel("Iterasi")
plt.ylabel("( $e^{(\delta E / T)}$ )")
plt.show()

# Menghitung dan menampilkan frekuensi stuck di local optima
stuckCount = sum(1 for i in range(1, len(objectives)) if objectives[i] == objectives[i - 1])
print("Frekuensi Stuck di Local Optima:", stuckCount)

```

2.2.7 Genetic Algorithm

Genetic Algorithm adalah metode *local search* yang meniru proses evolusi alam, bekerja dengan populasi solusi yang berkembang melalui seleksi, *crossover*, dan mutasi untuk menemukan solusi terbaik. Dalam konteks masalah *diagonal magic cube 5x5x5*, setiap solusi direpresentasikan sebagai kromosom yang berisi elemen-elemen diagonal. Algoritma ini dimulai dengan populasi *random*, kemudian populasi dengan *fitness* terbaik (kesalahan paling sedikit dalam mencapai *magic number*) dipilih untuk bereproduksi. *Crossover* menyilangkan dua solusi untuk membentuk solusi baru, sementara mutasi menjaga keberagaman dengan mengubah elemen secara acak. Dengan terus memperbaiki populasi, *Genetic Algorithm* dapat meminimalisir terjadinya *local maximum* dan lebih efektif menemukan *global maximum* untuk meminimalisir kesalahan dalam *diagonal magic cube*.

- a. Fungsi GENETICALGORITHM(*initial_cube*, *population_size*,
max_generations, *tournament_size*)

Deskripsi	Fungsi ini berisi algoritma dari <i>Genetic Algorithm</i> dengan menerima parameter <i>initial cube</i> , jumlah populasi, banyaknya iterasi/generasi, dan ukuran tournament (untuk melakukan selection)
Proses	<i>Local Search</i> ini dimulai dengan membentuk populasi acak dan melakukan seleksi, <i>crossover</i> , serta mutasi untuk menghasilkan solusi baru. Setiap generasi, elitisme diterapkan untuk mempertahankan solusi terbaik. Proses ini diulang hingga ditemukan solusi optimal atau mencapai jumlah generasi maksimum. Fungsi ini mengembalikan solusi terbaik, riwayat fitness terbaik dan rata-rata, jumlah generasi, serta waktu eksekusi

```

def GENETICALGORITHM(initial_cube, population_size,
max_generations, tournament_size=20):
    def initialize_population(size):
        return [MagicCube(5) for _ in range(size)]

    def fitnessFunction(cube): # Fitness Function
        return 1 / (1 + cube.objectiveFunction())

    def tournament_selection(population, size): # Selection
        menggunakan tournament
        tournament = random.sample(population, size)
        return max(tournament, key=fitnessFunction)

    def crossover(parent1, parent2): # Crossover
        menggunakan metode uniform crossover
        n = parent1.cube.shape[0]
        child = MagicCube(n)

        mask = np.random.rand(*parent1.cube.shape) < 0.5
        child.cube[mask] = parent1.cube[mask]
        child.cube[~mask] = parent2.cube[~mask]

    return child

    def repair(child): # Untuk mengurus value duplikat dan
        value yang hilang
        unique_numbers = np.unique(child.cube)
        all_numbers = set(range(1, child.n**3 + 1))
        duplicates = list(set(unique_numbers[unique_numbers
        != 0]) - all_numbers)
        missing_numbers = list(all_numbers -
        set(unique_numbers))

        for i in range(child.n):
            for j in range(child.n):

```

```

        for k in range(child.n):
            if
                list(child.cube.flatten()).count(child.cube[i, j, k]) > 1 and
                missing_numbers:
                    child.cube[i, j, k] =
                    missing_numbers.pop(0)

    def mutate(cube): #
        num_mutations = max(1, cube.n // 5)
        for _ in range(num_mutations):
            pos1 = (random.randint(0, cube.n - 1),
            random.randint(0, cube.n - 1), random.randint(0, cube.n - 1))
            pos2 = (random.randint(0, cube.n - 1),
            random.randint(0, cube.n - 1), random.randint(0, cube.n - 1))
            cube.swapElements(pos1, pos2)
            repair(cube)

    population = initialize_population(population_size) #
Inisialisasi populasi
    best_fitness_history = [] # Untuk menyimpan history best
fitness
    avg_fitness_history = [] # Untuk menyimpan history
average fitness
    start_time = time.time() # Untuk menghitung waktu
eksekusi

    for generation in range(max_generations): # Looping
untuk setiap generasi
        new_population = []

        num_elites = 2
        elites = sorted(population, key=fitnessFunction,
reverse=True)[:num_elites] # Mengambil 2 individu terbaik
        new_population.extend(elites) # Menambahkan 2
individu terbaik ke new_population

```

```

        while len(new_population) < population_size: #
Looping untuk mengisi new_population
        parent1 = tournament_selection(population,
tournament_size)
        parent2 = tournament_selection(population,
tournament_size)
        child = crossover(parent1, parent2)
        repair(child)
        mutate(child)
        new_population.append(child)

population = new_population # Update populasi

fitnesses = [fitnessFunction(cube) for cube in
population] # Menghitung fitness untuk setiap individu
best_fitness = max(fitnesses) # Mengambil fitness
terbaik
avg_fitness = sum(fitnesses) / len(fitnesses) # Menghitung rata-rata fitness

best_obj_value = elites[0].objectiveFunction() #
Mengambil nilai fungsi objektif terbaik
avg_obj_value = sum(cube.objectiveFunction() for cube
in population) / len(population) # Menghitung rata-rata
nilai fungsi objektif

best_fitness_history.append(best_obj_value)
avg_fitness_history.append(avg_obj_value)

print(f"Generation {generation}: Best f(x) =
{best_obj_value}, Avg f(x) = {avg_obj_value}")

if best_obj_value == 0: # Jika mencapai objective
function, maka berhenti

```

```

        break

duration = time.time() - start_time
best_solution = max(population, key=fitnessFunction) #

Mengambil individu terbaik

return best_solution, best_fitness_history,
avg_fitness_history, generation + 1, duration #
Mengembalikan individu terbaik, history best fitness,
history average fitness, jumlah generasi, dan waktu
eksekusi

```

- b. Fungsi visualize_experiment(initial_cube, final_cube, best_history, avg_history)

Deskripsi	Fungsi ini berfungsi untuk menunjukkan visualisasi dari hasil <i>genetic algorithm</i>
Proses	Menerima initial_cube, final_cube, best_history, avg_history dan mengembalikan tampilan 5 <i>layer cube</i> pada posisi awal dan posisi akhir juga menampilkan grafik nilai <i>objective function</i> terbaik dan rata-rata <i>objective function</i> terhadap banyak iterasi/generasi yang telah dilewati

2.3 Hasil Eksperimen dan Analisis (Disertai dengan Visualisasi dari Program yang Telah Dibuat)

Setelah melakukan eksperimen penerapan algoritma *local search* dengan program yang telah kami buat pada pencarian solusi *diagonal magic cube* kami mendapatkan hasil hasil dari masing-masing algoritma yang akan kami jabarkan pada poin-poin berikut.

2.3.1 Steepest Ascent Hill-climbing

Kami melakukan percobaan dengan cara menjalankan algoritma sebanyak tiga kali. Dari tiga kali percobaan itu, kami mendapatkan hasil yang berbeda-beda dari *Steepest Ascent Hill-climbing*. Berikut merupakan hasil yang kami dapatkan.

A. Percobaan Pertama

Hasil *objective function* yang didapatkan sebesar **577** dengan durasi sebesar **376.95 detik** dan melakukan iterasi sebanyak 50 kali (berhenti karena iterasi telah terpenuhi). Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

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

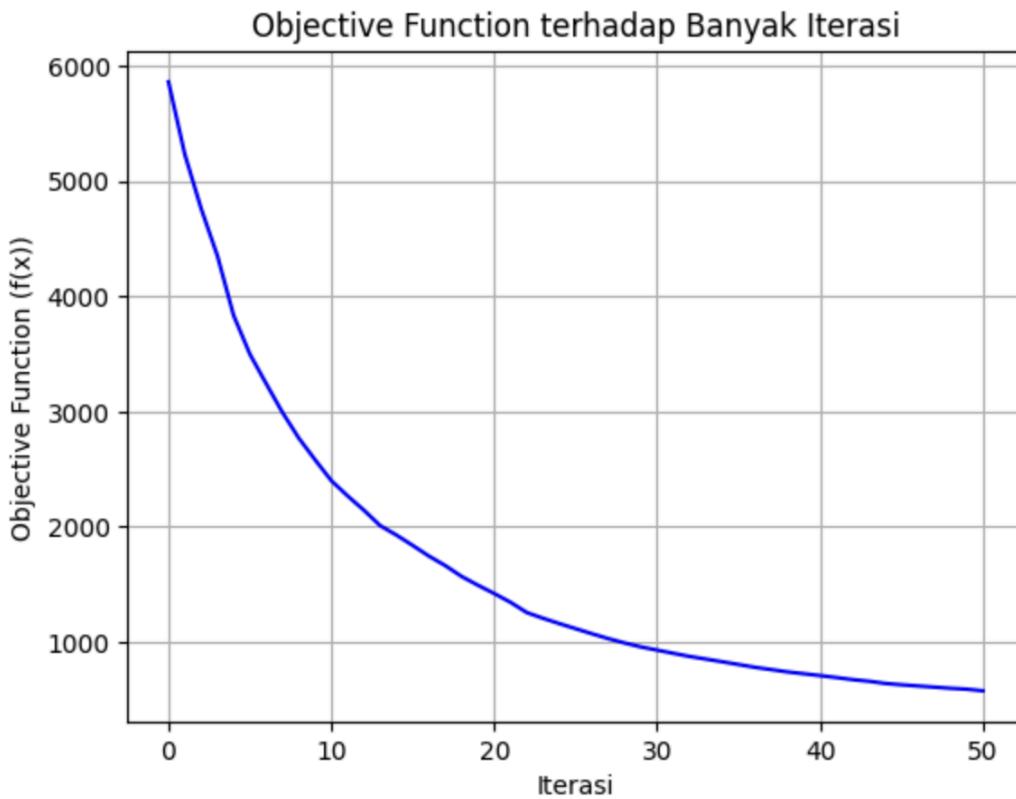
Gambar 2.4 Visualisasi *Diagonal Magic Cube* Awal 1

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan algoritma *steepest ascent hill-climbing*.

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

Gambar 2.5 Visualisasi *Diagonal Magic Cube* Akhir 1

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.6 Plot *Objective Function* terhadap Banyak Iterasi 1

B. Percobaan Kedua

Hasil *objective function* yang didapatkan sebesar **433** dengan durasi sebesar **382.31 detik** dan melakukan iterasi sebanyak 50 kali (berhenti karena iterasi telah terpenuhi). Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

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

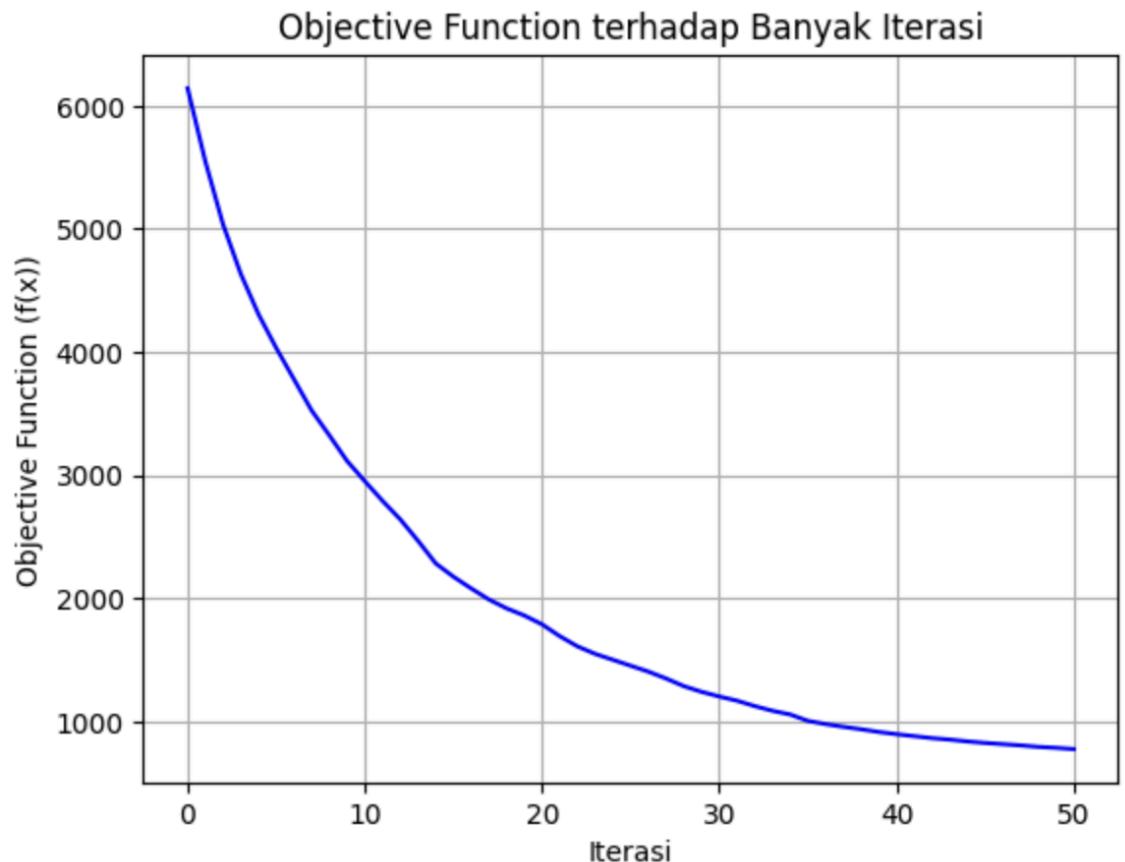
Gambar 2.1 Visualisasi *Diagonal Magic Cube* Awal 2

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan algoritma *steepest ascent hill-climbing*.

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

Gambar 2.2 Visualisasi *Diagonal Magic Cube* Akhir 2

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.3 Plot *Objective Function* terhadap Banyak Iterasi 2

C. Percobaan Ketiga

Hasil *objective function* yang didapatkan sebesar **775** dengan durasi sebesar **372.83 detik** dan melakukan iterasi sebanyak 50 kali (berhenti karena iterasi telah terpenuhi). Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

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

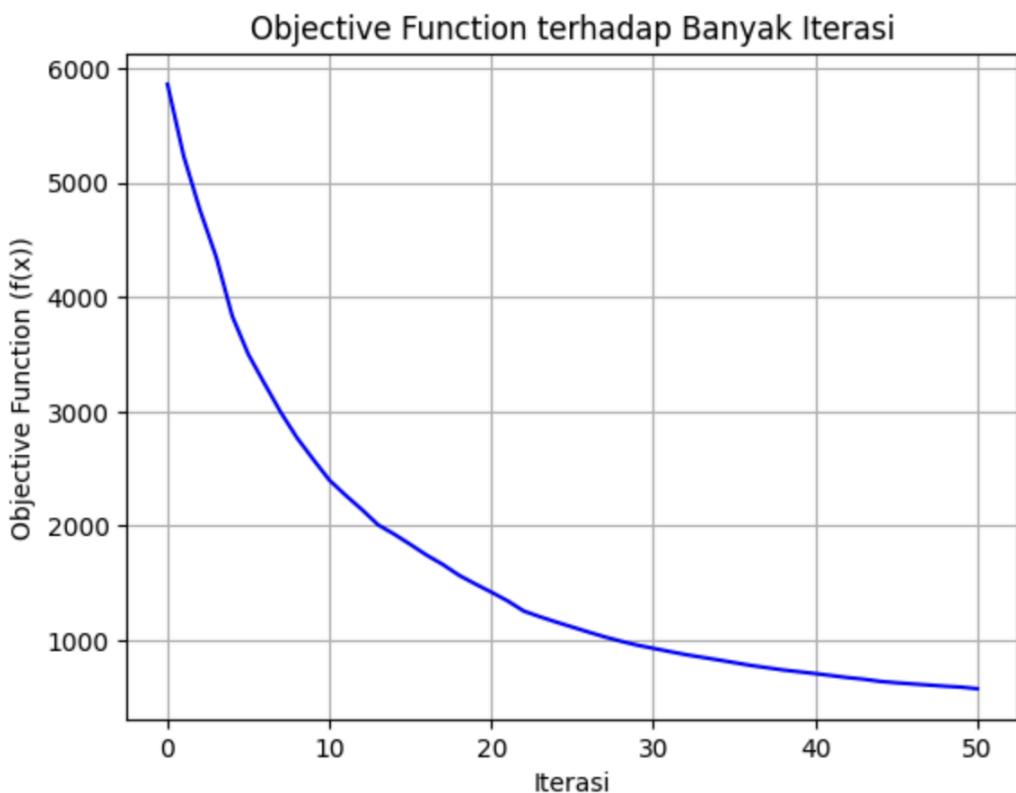
Gambar 2.7 Visualisasi *Diagonal Magic Cube* Awal 3

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan algoritma *steepest ascent hill-climbing*.

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

Gambar 2.8 Visualisasi *Diagonal Magic Cube* Akhir 3

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.9 Plot *Objective Function* terhadap Banyak Iterasi 3

Dari hasil hasil tersebut dapat dilakukan analisis tentang hasil eksperimen tersebut yang kami jabarkan melalui tabel berikut.

Tabel 2.1 Analisis Hasil Eksperimen *Steepest Ascent*

Analisis	Hasil (Average)
Hasil Algoritma (Objective Function)	595
Waktu yang diperlukan	377,4s
Kedekatan dengan <i>Global Optima</i> (<i>Proximity</i>)	98,27%
Konsistensi Hasil	28.86%

Dapat dilihat bahwa *proximity* atau kedekatan hasil ke *global optimia* sangatlah tinggi, hal ini dapat diakibatkan karena *steepest ascent hill-climbing*

langsung mencari nilai *neighbour* yang memiliki *objective function* terbaik. Algoritma pencarian *successor / neighbor* kami tidak memiliki batasan dan mencari *neighbor* sebanyak mungkin, sehingga wajar saja jika *steepest ascent hill-climbing* memiliki nilai *proximity* yang tinggi karena jumlah *neighbor* meningkatkan efektifitas *steepest ascent hill-climbing* walaupun mengorbankan keefisienan. Namun, konsistensi hasil masih bervariasi (semakin mendekati 0% semakin konsisten), hal ini disebabkan oleh penentuan hasil dari algoritma ini tergantung kepada *initial state diagonal magic cube*.

2.3.2 Hill-climbing with Sideways Move

Kami melakukan percobaan dengan cara menjalankan algoritma sebanyak tiga kali, dengan batasan iterasi 50 dan batasan *sideways move* 10. Dari tiga kali percobaan itu, kami mendapatkan hasil yang berbeda-beda dari *Hill-climbing with Sideways Move*. Berikut merupakan hasil yang kami dapatkan.

A. Percobaan Pertama

Hasil *objective function* yang didapatkan sebesar **785** dengan durasi sebesar **386.32 detik** dan melakukan iterasi sebanyak 50 kali (berhenti karena iterasi telah terpenuhi). Berikut merupakan visualisasi *state awal diagonal magic cube*.

Visualisasi Diagonal Magic Cube																			
Layer 1					Layer 2					Layer 3					Layer 4				
35	107	10	9	85	79	104	44	4	96	106	41	113	100	93	68	55	2	101	112
63	54	99	114	23	31	60	61	82	48	125	12	26	36	90	6	98	84	1	57
73	42	45	32	37	59	75	39	52	92	117	74	7	115	123	17	72	27	122	102
15	28	80	34	5	71	18	67	95	16	46	43	105	109	3	58	83	89	91	76
116	119	81	25	29	69	11	30	87	103	8	21	118	20	24	19	97	121	77	70

Gambar 2.10 Visualisasi *Diagonal Magic Cube* Awal 1

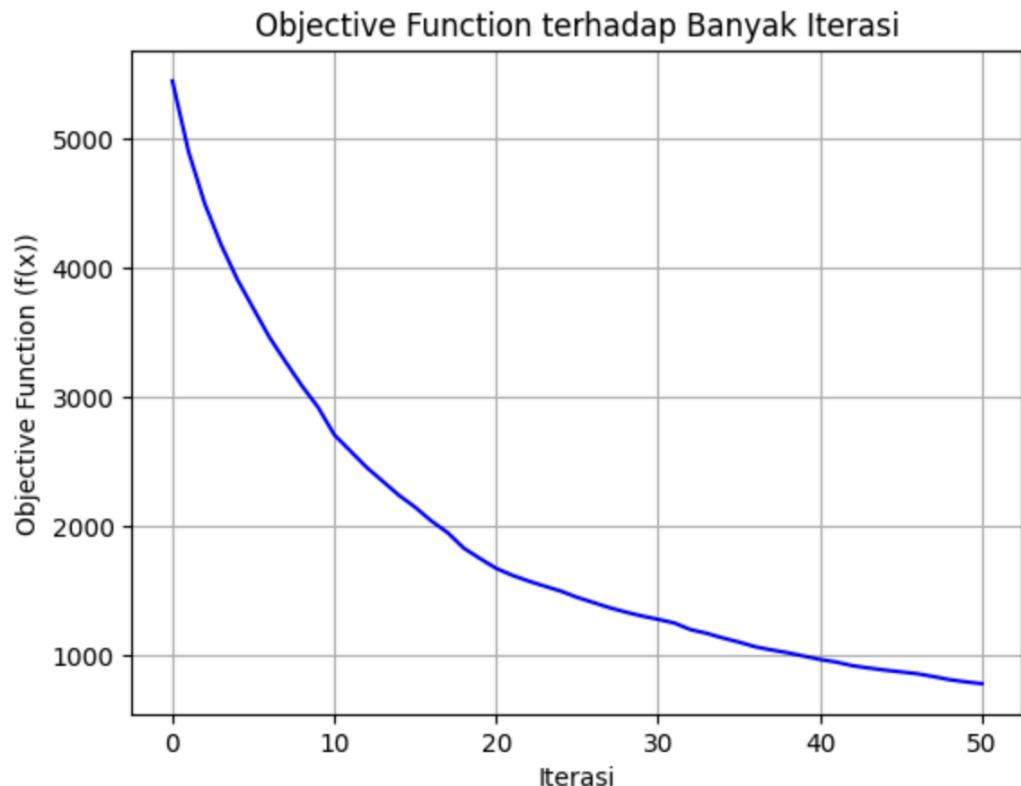
Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah

diterapkan algoritma *hill-climbing with sideways move*.

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

Gambar 2.11 Visualisasi *Diagonal Magic Cube* Akhir 1

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.12 Plot *Objective Function* terhadap Banyak Iterasi 1

B. Percobaan Kedua

Hasil *objective function* yang didapatkan sebesar **565** dengan durasi sebesar **388.06 detik** dan melakukan iterasi sebanyak 50 kali (berhenti karena iterasi telah terpenuhi). Berikut merupakan visualisasi *state* awal *diagonal magic*

cube.

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

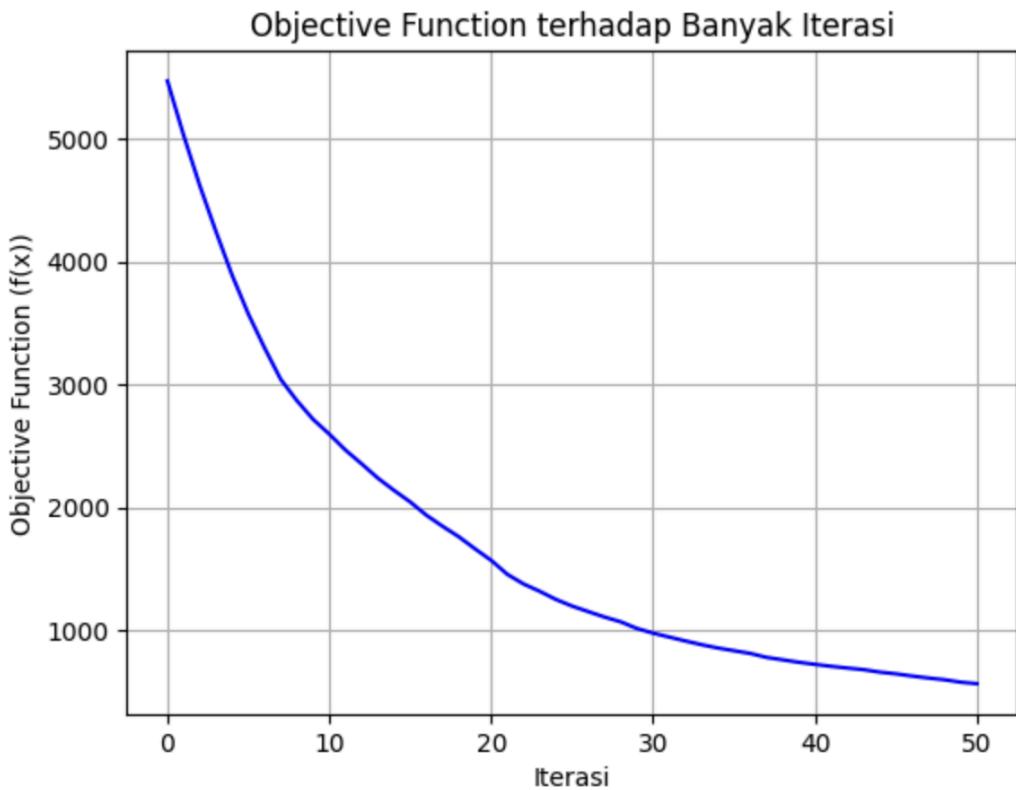
Gambar 2.13 Visualisasi *Diagonal Magic Cube* Awal 2

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan algoritma *hill-climbing with sideways move*.

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

Gambar 2.14 Visualisasi *Diagonal Magic Cube* Akhir 2

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.15 Plot *Objective Function* terhadap Banyak Iterasi 2

C. Percobaan Ketiga

Hasil *objective function* yang didapatkan sebesar **497** dengan durasi sebesar **404.27 detik** dan melakukan iterasi sebanyak 50 kali (berhenti karena iterasi telah terpenuhi). Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

Visualisasi Diagonal Magic Cube																			
Layer 1					Layer 2					Layer 3					Layer 4				
36	115	8	9	93	108	102	16	109	118	30	18	80	112	51	71	48	54	17	84
12	49	68	65	103	45	114	100	25	67	19	64	37	96	59	122	33	31	86	57
5	72	22	82	116	40	76	69	88	41	38	63	95	46	107	13	26	56	113	52
81	70	53	106	83	120	10	29	78	55	15	101	3	2	74	6	124	97	99	24
105	11	1	28	125	58	87	98	43	23	89	39	110	92	111	4	60	79	123	73

Gambar 2.16 Visualisasi *Diagonal Magic Cube* Awal 3

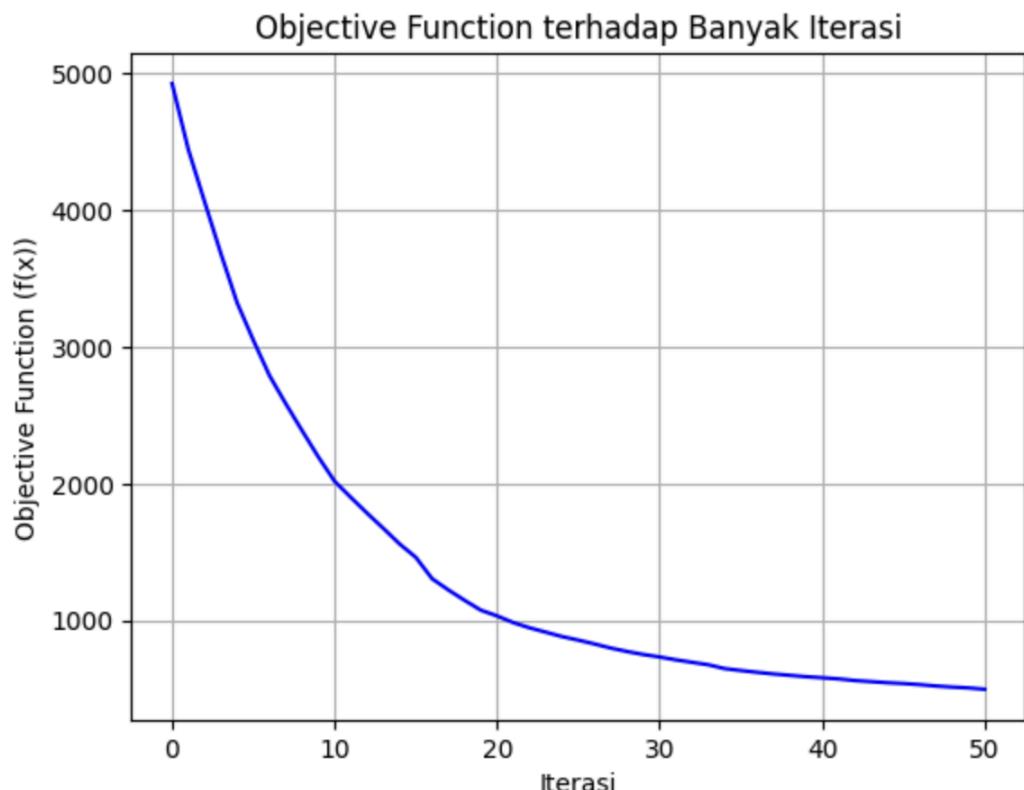
Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah

diterapkan algoritma *hill-climbing with sideways move*.

Visualisasi Diagonal Magic Cube																			
Layer 1				Layer 2				Layer 3		Layer 4			Layer 5						
50	117	95	35	15	61	16	7	109	124	88	18	80	112	17	71	60	54	51	84
8	26	115	65	103	45	114	110	25	23	19	108	37	83	67	125	33	31	52	59
116	92	22	87	1	40	76	69	91	41	5	63	93	46	107	111	49	56	12	86
30	70	39	100	74	120	10	53	43	96	113	97	3	2	99	6	118	101	78	13
105	11	38	28	122	58	98	82	48	27	89	29	102	72	24	4	55	79	123	73

Gambar 2.17 Visualisasi *Diagonal Magic Cube* Akhir 3

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.18 Plot *Objective Function* terhadap Banyak Iterasi 3

Dari hasil hasil tersebut dapat dilakukan analisis tentang hasil eksperimen tersebut yang kami jabarkan melalui tabel berikut.

Tabel 2.2 Analisis Hasil Eksperimen *Hill-climbing with Sideways Move*

Analisis	Hasil (Average)
Hasil Algoritma	615,67
Waktu yang diperlukan	392,8s
Kedekatan dengan <i>Global Optima (Proximity)</i>	98,2%
Konsistensi Hasil	24,45%

Dapat dilihat sama seperti *steepest ascent hill-climbing* bahwa *proximity* atau kedekatan hasil ke *global* optima sangatlah tinggi. Algoritma pencarian *successor / neighbor* kami tidak memiliki batasan dan mencari *neighbor* sebanyak mungkin, sehingga wajar saja jika *hill-climbing with sideways move* memiliki nilai *proximity* yang tinggi karena jumlah *neighbor* meningkatkan keefektifitasan *hill-climbing with sideways move* walaupun mengorbankan keefisienan. Walaupun memiliki hasil yang mirip dengan *steepest ascent hill-climbing*, hasil, waktu, dan *proximity* sedikit lebih buruk dari *steepest ascent hill-climbing*. Hasil bisa diakibatkan karena *hill-climbing with sideways move* akan berpindah ke *neighbor* yang setara dengan *current value*, memungkinkan mengabaikan nilai *neighbor* yang lebih baik, selain itu cara itulah yang membuat algoritma berjalan sedikit lebih lama. Waktu yang lebih lama juga diakibatkan oleh *sideways moves* yang diambil oleh *hill-climbing with sideways move*. Namun, konsistensi hasil dari algoritma ini lebih konsisten dibandingkan *steepest ascent hill-climbing*.

2.3.3 Random Restart Hill-climbing

Kami melakukan percobaan dengan cara menjalankan algoritma sebanyak tiga kali, dengan batasan iterasi 50 dan batasan *restart* 5. Dari tiga kali percobaan itu, kami mendapatkan hasil yang berbeda-beda dari *Random Restart Hill-climbing*. Berikut merupakan hasil yang kami dapatkan.

A. Percobaan Pertama

Hasil *objective function* yang didapatkan sebesar **333** dengan durasi sebesar **1970.76 detik** dan melakukan iterasi sebanyak 200 kali dan *restart* sebanyak 5 kali(berhenti karena *restart* telah terpenuhi). Berikut merupakan visualisasi *state* awal *diagonal magic cube*.



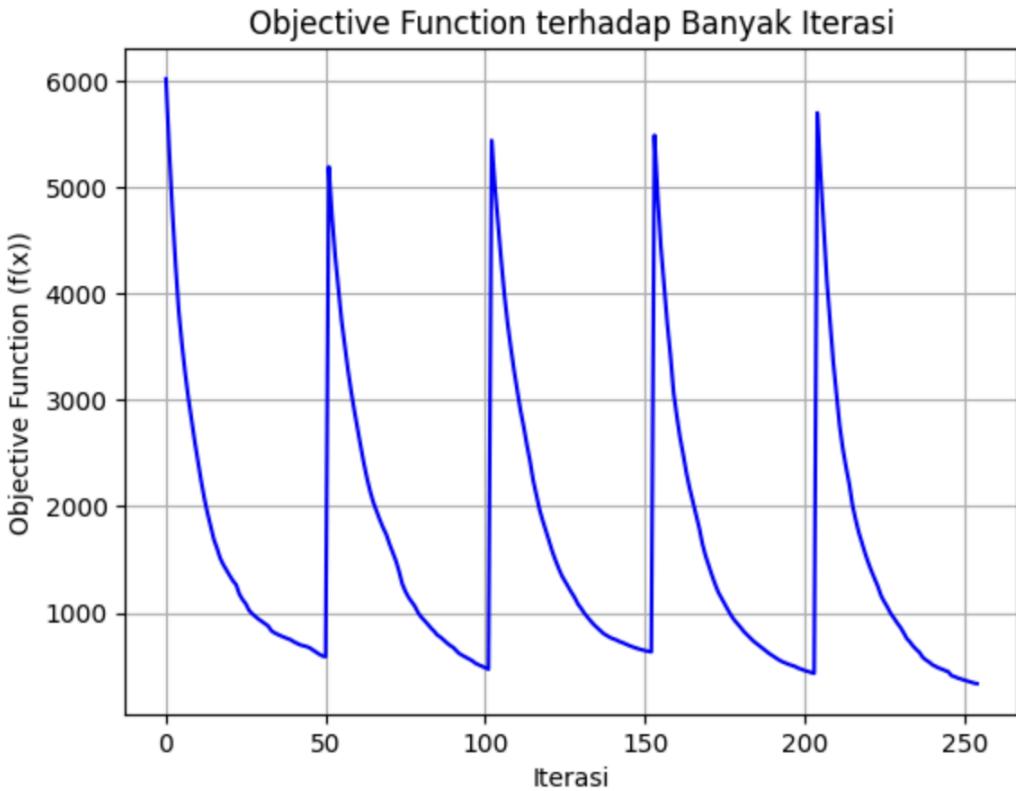
Gambar 2.19 Visualisasi *Diagonal Magic Cube* Awal 1

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan algoritma *random restart hill-climbing*.



Gambar 2.20 Visualisasi *Diagonal Magic Cube* Akhir 1

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.21 Plot *Objective Function* terhadap Banyak Iterasi 1

B. Percobaan Kedua

Hasil *objective function* yang didapatkan sebesar **394** dengan durasi sebesar **1898.98 detik** dan melakukan iterasi sebanyak 200 kali dan *restart* sebanyak 5 kali(berhenti karena *restart* telah terpenuhi). Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
14	103	27	92	111		119	68	34	82	120		47	18	91	116	86
84	43	104	77	80		102	29	60	105	121		70	112	48	42	57
6	30	32	58	51		28	41	31	63	22		94	9	40	17	122
55	75	37	66	114		69	125	39	62	67		56	95	38	72	109
52	83	118	10	85		113	3	1	100	23		25	59	46	36	45

Gambar 2.22 Visualisasi *Diagonal Magic Cube* Awal 2

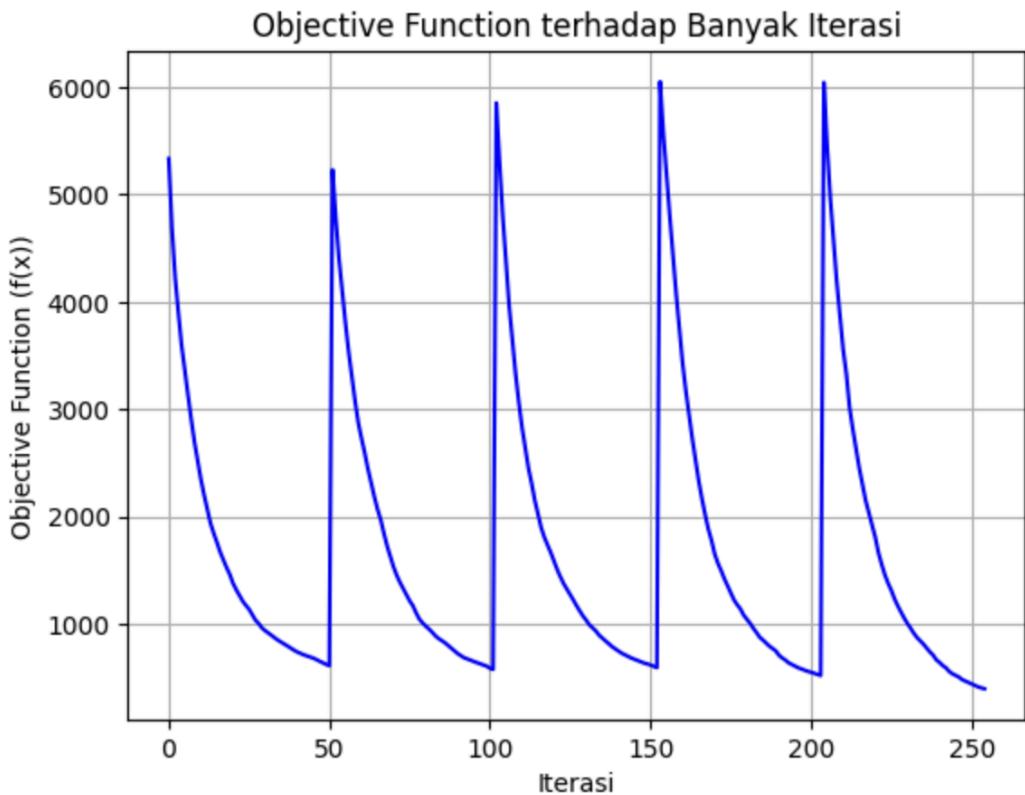
Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah

diterapkan algoritma *random restart hill-climbing*.

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

Gambar 2.23 Visualisasi *Diagonal Magic Cube* Akhir 2

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.24 Plot *Objective Function* terhadap Banyak Iterasi 2

C. Percobaan Ketiga

Hasil *objective function* yang didapatkan sebesar **351** dengan durasi sebesar **1928.91 detik** dan melakukan iterasi sebanyak 200 kali dan *restart* sebanyak 5 kali(berhenti karena *restart* telah terpenuhi). Berikut merupakan

visualisasi *state* awal *diagonal magic cube*.

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

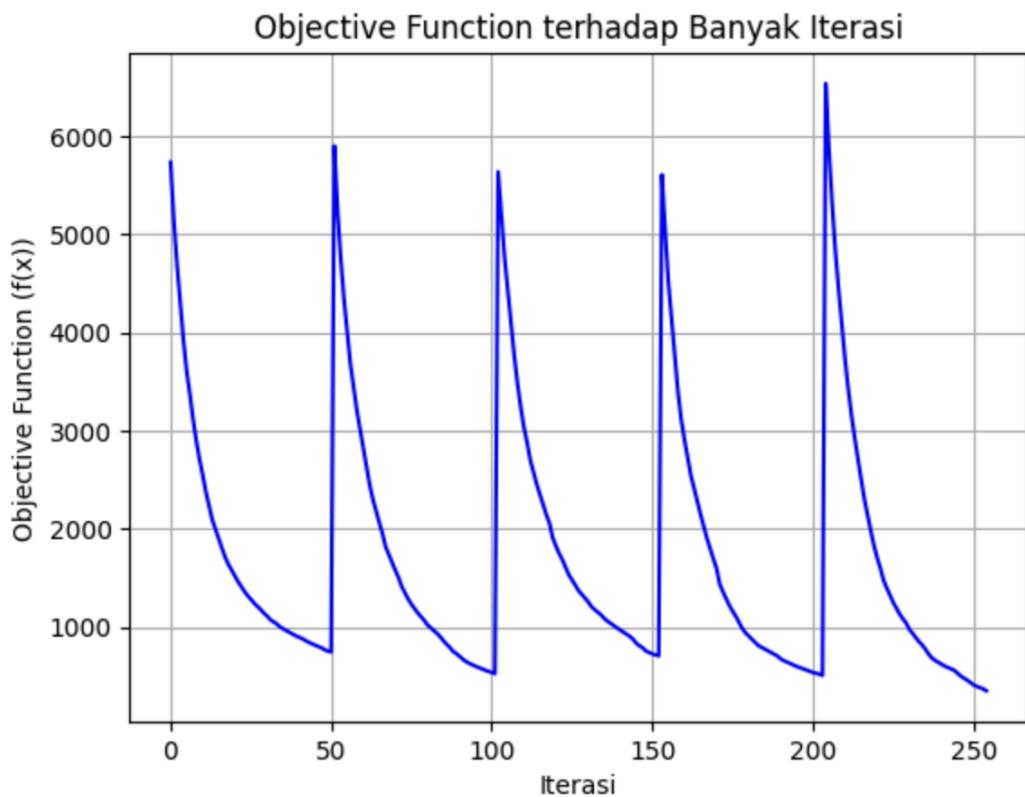
Gambar 2.25 Visualisasi *Diagonal Magic Cube* Awal 3

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan algoritma *random restart hill-climbing*.

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

Gambar 2.26 Visualisasi *Diagonal Magic Cube* Akhir 3

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.27 Plot *Objective Function* terhadap Banyak Iterasi 3

Dari hasil hasil tersebut dapat dilakukan analisis tentang hasil eksperimen tersebut yang kami jabarkan melalui tabel berikut.

Tabel 2.3 Analisis Hasil Eksperimen *Random Restart hill-climbing*

Analisis	Hasil (Average)
Hasil Algoritma	359,3
Waktu yang diperlukan	1932,88 s
Kedekatan dengan <i>Global Optima (Proximity)</i>	98,95%
Konsistensi Hasil	8,72%

Dibandingkan semua algoritma *hill-climbing*, *random restart hill-climbing* memiliki hasil yang paling baik dengan *proximity* yang sangat

tinggi dan hasil yang paling konsisten. Hal ini karena *random restart* akan melakukan *restart* setiap algoritma menemui *local optima*, sehingga memiliki eksplorasi yang tinggi namun hasil yang baik dan konsisten. Namun, terdapat kekurangan pada algoritma ini yaitu waktunya yang sangat lama dikarenakan *restart* yang diulang berkali kali dimana ini memerlukan pengulangan dalam menjalankan algoritma. Sehingga algoritma ini efektif namun tidak efisien.

2.3.4 Stochastic Hill-climbing

Kami melakukan percobaan dengan cara menjalankan algoritma sebanyak tiga kali, dengan batasan iterasi 50. Dari tiga kali percobaan itu, kami mendapatkan hasil yang berbeda-beda dari *Stochastic Hill-climbing*. Berikut merupakan hasil yang kami dapatkan.

A. Percobaan Pertama

Hasil *objective function* yang didapatkan sebesar **3326** dengan durasi sebesar **383.01 detik** dan melakukan iterasi sebanyak 50 kali (berhenti karena iterasi telah terpenuhi). Berikut merupakan visualisasi *state awal diagonal magic cube*.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
71	28	31	32	85		87	23	88	38	49		52	122	93	117	86
110	50	2	89	104		103	10	56	45	6		83	96	34	1	41
60	21	98	62	111		54	59	36	65	84		79	102	116	112	124
67	120	125	109	33		105	119	95	44	100		66	39	68	35	8
108	51	26	18	92		115	97	113	76	16		57	24	99	30	101
						123	4	53	63	73		37	25	40	72	82
												20	9	75	81	43

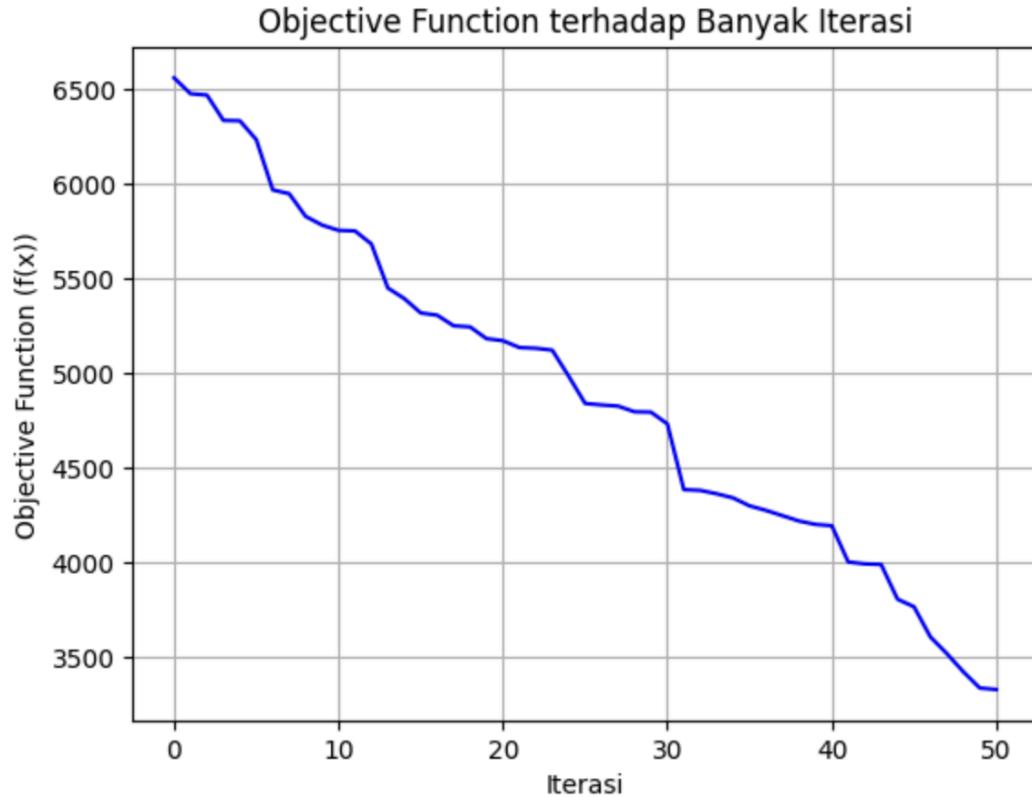
Gambar 2.28 Visualisasi *Diagonal Magic Cube* Awal 1

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan algoritma *stochastic hill-climbing*.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
106	28	31	102	13		82	23	7	91	111		15	64	83	99	90
88	53	20	89	104		121	108	63	114	6		3	39	118	80	70
47	21	98	125	49		54	50	36	59	119		105	94	40	30	43
58	120	62	5	33		16	87	51	48	100		107	14	68	116	8
10	84	26	18	92		37	112	113	76	9		123	4	65	86	73
												115	97	17	72	44
												2	69	75	81	45

Gambar 2.29 Visualisasi *Diagonal Magic Cube* Akhir 1

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.30 Plot *Objective Function* terhadap Banyak Iterasi 1

B. Percobaan Kedua

Hasil *objective function* yang didapatkan sebesar **2711** dengan durasi sebesar **379.62 detik** dan melakukan iterasi sebanyak 50 kali (berhenti karena iterasi telah terpenuhi). Berikut merupakan visualisasi *state awal diagonal magic cube*.

Visualisasi Diagonal Magic Cube

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

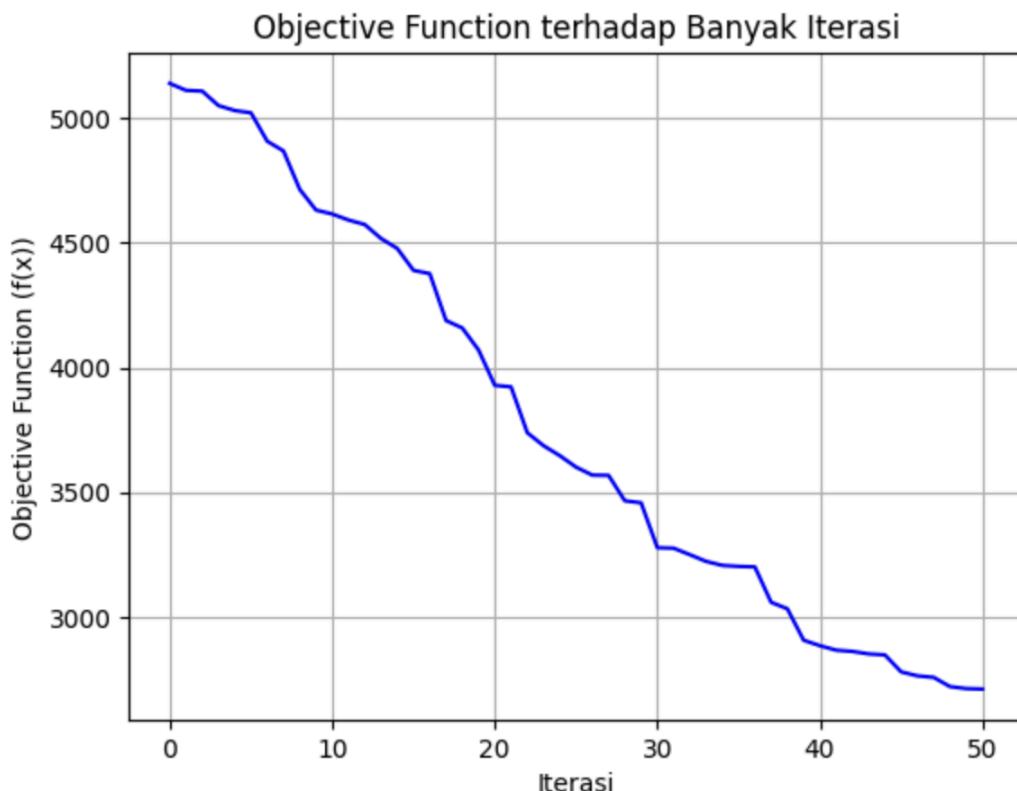
Gambar 2.31 Visualisasi *Diagonal Magic Cube* Awal 2

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan algoritma *stochastic hill-climbing*.

Visualisasi Diagonal Magic Cube														
Layer 1					Layer 2			Layer 3		Layer 4		Layer 5		
120	8	39	15	76	88	42	121	59	28	49	29	86	72	58
82	18	118	109	10	11	106	43	83	104	74	60	23	94	81
34	95	17	45	114	110	14	102	50	41	117	31	57	93	25
66	79	116	51	107	20	98	44	30	96	4	75	84	65	33
37	92	19	108	54	85	64	6	97	38	80	119	73	3	53

Gambar 2.32 Visualisasi *Diagonal Magic Cube* Akhir 2

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.33 Plot *Objective Function* terhadap Banyak Iterasi 2

C. Percobaan Ketiga

Hasil *objective function* yang didapatkan sebesar **2942** dengan durasi sebesar **384.22 detik** dan melakukan iterasi sebanyak 50 kali (berhenti karena

iterasi telah terpenuhi). Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

Visualisasi Diagonal Magic Cube														
Layer 1					Layer 2			Layer 3		Layer 4		Layer 5		
42	8	31	49	76	80	23	93	36	71	67	11	90	51	53
83	115	79	85	94	60	78	113	46	64	63	68	6	61	65
91	14	74	125	13	5	37	84	101	16	103	77	98	72	2
48	59	41	118	55	24	121	105	47	32	114	57	19	73	87
35	40	107	29	21	20	17	117	122	75	111	82	108	10	45

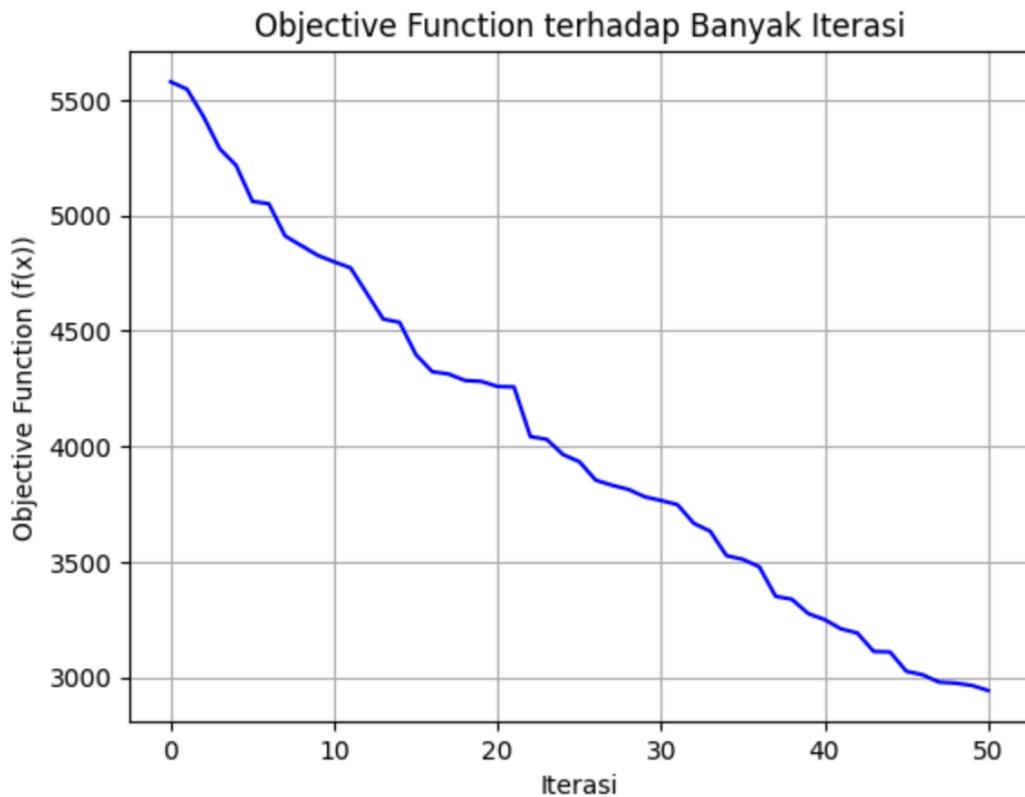
Gambar 2.34 Visualisasi *Diagonal Magic Cube* Awal 3

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan algoritma *stochastic hill-climbing*.

Visualisasi Diagonal Magic Cube														
Layer 1					Layer 2			Layer 3		Layer 4		Layer 5		
42	3	63	86	105	93	111	43	36	55	119	28	34	92	33
99	110	70	8	108	90	78	45	46	64	80	16	102	100	38
66	19	74	107	13	37	5	84	101	77	4	89	109	26	69
85	59	41	75	71	25	121	76	24	47	122	94	14	73	97
35	95	44	29	21	20	17	82	114	81	23	117	30	10	88

Gambar 2.35 Visualisasi *Diagonal Magic Cube* Akhir 3

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2.36 Plot *Objective Function* terhadap Banyak Iterasi 3

Dari hasil hasil tersebut dapat dilakukan analisis tentang hasil eksperimen tersebut yang kami jabarkan melalui tabel berikut.

Tabel 2.4 Analisis Hasil Eksperimen *Stochastic Hill-climbing*

Analisis	Hasil (Average)
Hasil Algoritma	2993
Waktu yang diperlukan	382,3 s
Kedekatan dengan <i>Global Optima (Proximity)</i>	91,3%
Konsistensi Hasil	10,38%

Dapat dilihat bahwa *proximity* atau kedekatan hasil ke *global optima* tinggi walaupun tidak setinggi algoritma yang lain. Hasil *stochastic hill-climbing*

tidak mencapai ratusan dapat disebabkan oleh pemilihan *neighbor* yang *random* walaupun dia memilih yang lebih baik namun yang dipilih bukanlah yang terbaik, seperti pada grafik diatas, hanya grafik *stochastic hill-climbing* yang tidak secara lancar menurun, walaupun hal ini dapat membuka jalan ke hasil yang lebih baik, namun kemungkinannya untuk mendapatkan hasil yang lebih buruk lebih tinggi. Walaupun begitu, ternyata konsistensi hasil pada algoritma ini lumayan baik dan konsisten, hal ini bisa disebabkan oleh tingkat eksplorasi yang tinggi dan tujuannya dalam menghindari *local optima*.

2.3.5 Simulated Annealing

Kami melakukan percobaan dengan cara menjalankan algoritma sebanyak tiga kali. Dari tiga kali percobaan itu, kami mendapatkan hasil yang berbeda-beda dari *Simulated Annealing*. Berikut merupakan hasil yang kami dapatkan.

A. Percobaan Pertama

Hasil *objective function* yang didapatkan sebesar **193** dengan durasi sebesar **25.63 detik**, frekuensi terjebak di *local optima* sebesar **69395**, dan iterasi sebanyak 70000 kali. Berikut merupakan visualisasi *state awal diagonal magic cube*.

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

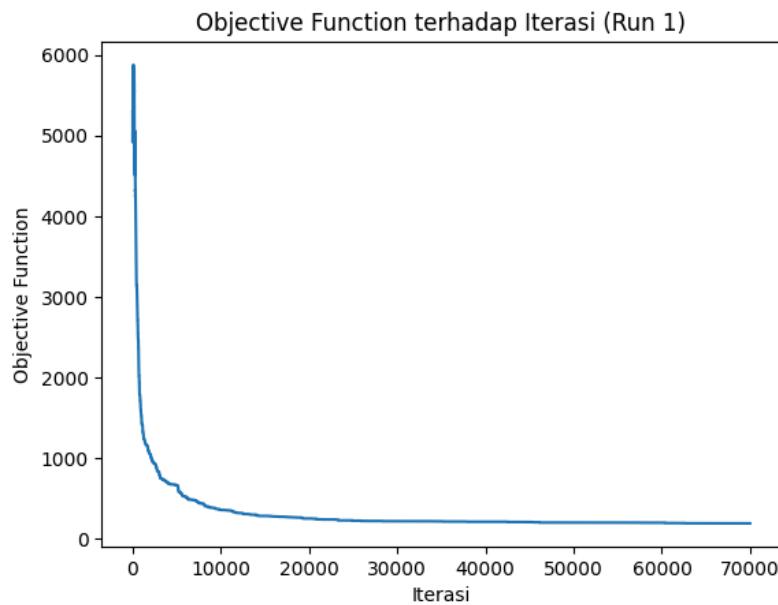
Gambar 2.37 Visualisasi *Diagonal Magic Cube* Awal 1

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan algoritma *Simulated Annealing*.

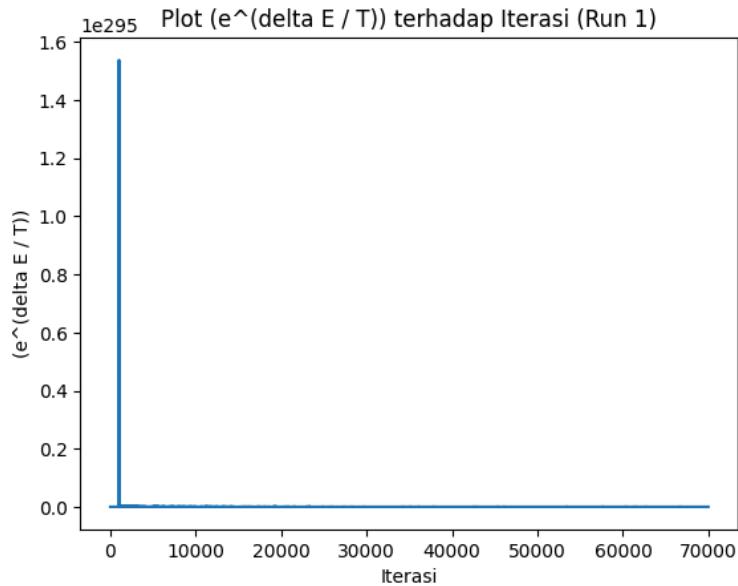
Visualisasi Diagonal Magic Cube														
Layer 1					Layer 2			Layer 3		Layer 4		Layer 5		
22	113	121	5	56	82	48	31	74	73	63	3	119	93	41
16	85	97	109	8	108	42	25	64	76	79	96	4	106	30
118	6	52	102	39	86	20	67	111	32	55	51	54	40	115
99	12	17	65	122	38	112	124	18	24	62	87	50	43	72
60	101	28	34	92	2	94	69	47	104	57	77	88	33	59

Gambar 2.38 Visualisasi *Diagonal Magic Cube* Akhir 1

Selain itu, untuk meningkatkan pemahaman atas plot nilai *objective function* terhadap banyak iterasi dan Plot $e^{\frac{\Delta E}{T}}$ terhadap banyak iterasi, akan ditunjukkan melalui grafik plot dibawah ini.



Gambar 2.39 Plot *Objective Function* terhadap Banyak Iterasi 1



Gambar 2.40 Plot *Objective Function* terhadap Plot $e^{\frac{\Delta E}{T}}$ terhadap Banyak Iterasi
1

B. Percobaan Kedua

Hasil *objective function* yang didapatkan sebesar **75** dengan durasi sebesar **24.42 detik**, frekuensi terjebak di lokal optimum sebesar **69402**, dan iterasi sebanyak 70000 kali. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

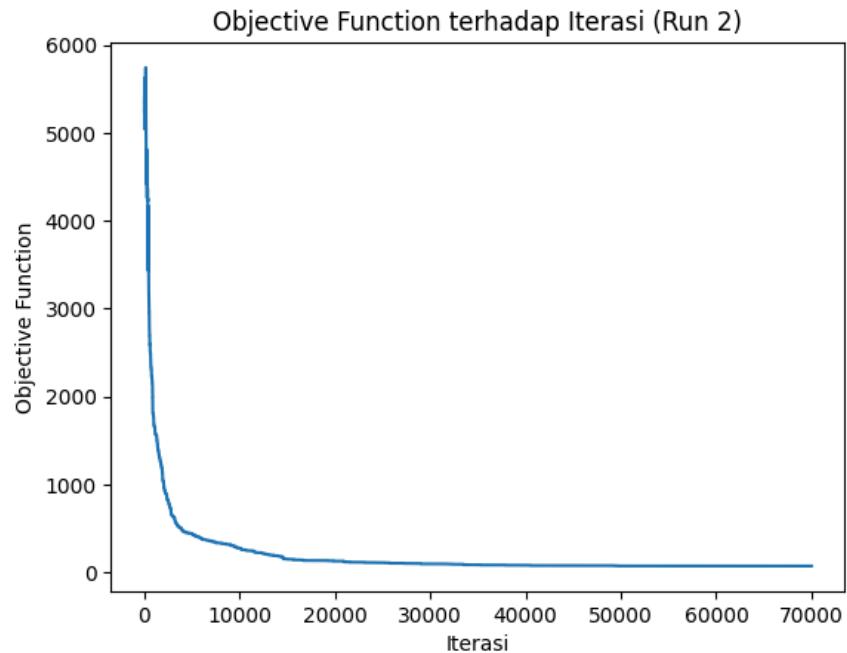
Visualisasi Diagonal Magic Cube																								
Layer 1					Layer 2					Layer 3					Layer 4					Layer 5				
61	66	59	98	17	13	68	22	81	112	40	10	104	120	56	80	16	106	15	12	99	100	94	49	102
32	2	82	116	63	69	64	87	73	123	14	6	55	53	11	52	71	65	117	83	108	30	47	103	115
37	24	38	95	50	96	74	51	114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

Gambar 2.41 Visualisasi *Diagonal Magic Cube* Awal 2

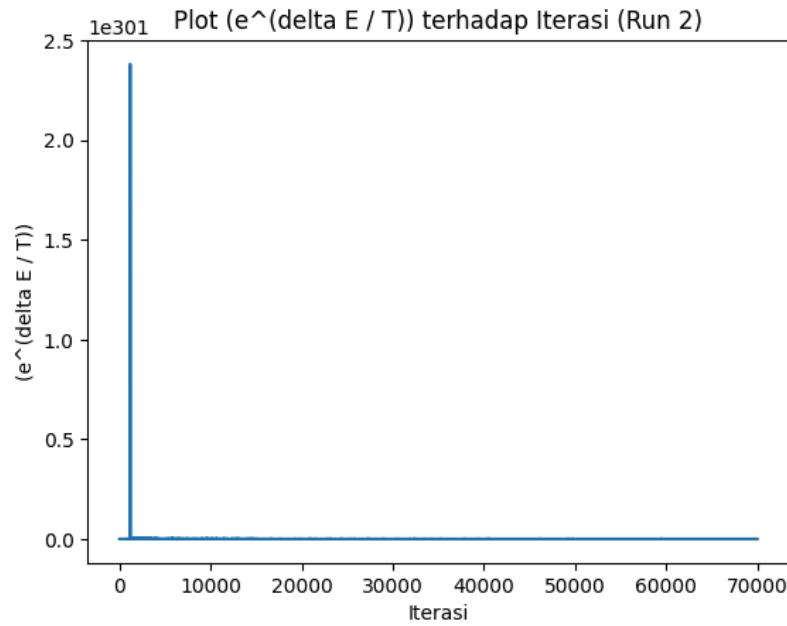
Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan algoritma *Simulated Annealing*.

Visualisasi Diagonal Magic Cube								
Layer 1			Layer 2			Layer 3		
116	10	51	64	75		54	81	91
18	71	58	76	92		73	84	68
70	78	48	36	83		27	88	21
104	109	38	39	25		69	23	123
7	47	120	100	40		74	125	20
						4	101	50
						107	112	110
						105	60	77
						109	114	93
						53	119	22
						121	32	59
						95	26	11
						17	56	122
						53	119	30

Gambar 2.42 Visualisasi *Diagonal Magic Cube* Akhir 2



Gambar 2.43 Plot *Objective Function* terhadap Banyak Iterasi 2



Gambar 2.44 Plot *Objective Function* terhadap Plot $e^{\frac{\Delta E}{T}}$ terhadap Banyak Iterasi 2

C. Percobaan Ketiga

Hasil *objective function* yang didapatkan sebesar **184** dengan durasi sebesar **26.03 detik**, frekuensi terjebak di lokal optimum sebesar **69393**, dan iterasi sebanyak 70000 kali. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

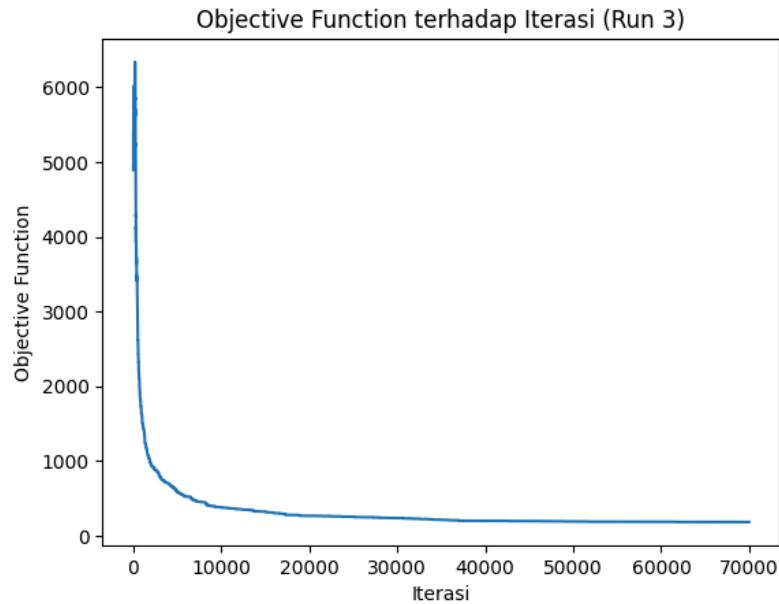


Gambar 2.45 Visualisasi *Diagonal Magic Cube* Awal 3

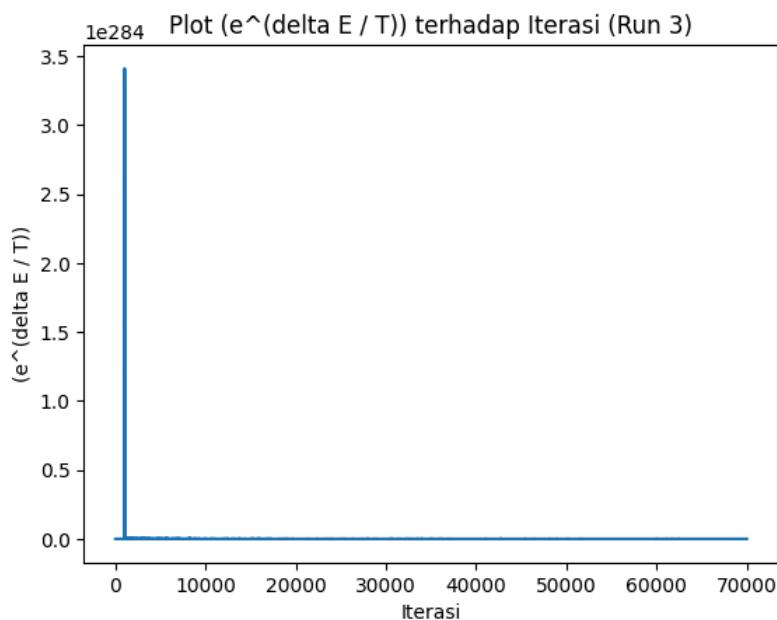
Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan algoritma *Simulated Annealing*.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
91	106	62	20	35		57	32	50	99	77		100	39	31	81	64
26	54	120	110	6		29	74	11	86	115		124	27	45	66	47
93	1	17	114	90		80	52	79	75	30		4	71	72	67	102
22	70	111	42	76		125	53	56	46	34		89	18	96	109	3
83	92	5	28	108		24	104	119	9	59		117	10	78	73	37

Gambar 2.46 Visualisasi *Diagonal Magic Cube* Akhir 3



Gambar 2.47 Plot *Objective Function* terhadap Banyak Iterasi 3



Gambar 2.48 Plot *Objective Function* terhadap Plot $e^{\frac{\Delta E}{T}}$ terhadap Banyak Iterasi 3

Dari hasil hasil tersebut dapat dilakukan analisis tentang hasil eksperimen tersebut yang kami jabarkan melalui tabel berikut.

Tabel 2.1 Analisis Hasil Eksperimen *Simulated Annealing*

Analisis	Hasil (Average)
Hasil Algoritma (<i>Objective Function</i>)	150, 667
Waktu yang diperlukan	25,36 s
Kedekatan dengan <i>Global Optima</i> (<i>Proximity</i>)	99, 56%
Konsistensi Hasil	43, 6%

Dari hasil eksperimen, didapatkan *proximity* sebesar 99, 56% terhadap global optimum, yang menunjukkan bahwa algoritma ini mampu mencari solusi yang sangat mendekati solusi optimal. Dari grafik yang dihasilkan, pada awal iterasi nilai *objective function* menurun dengan cepat. Namun seiring

berjalananya waktu, algoritma akan mendekati lokal optimal dan kesulitan dalam menemukan *neighbor* yang lebih baik.

Frekuensi *stuck* yang di dapat terbilang tinggi, ini menandakan tantangan untuk keluar dari terbilang kompleks. Berdasarkan grafik eksponensial menandakan perubahan energi terhadap suhu menunjukkan nilai yang sangat besar di awal iterasi. Hal ini sesuai dengan karakteristik *simulated annealing* di mana probabilitas penerimaan solusi lebih buruk menurun dengan pendinginan.

Konsistensi hasil sebesar 43,6% menunjukkan bahwa algoritma tidak selalu menghasilkan solusi yang serupa di setiap percobaan. Hal ini dipengaruhi oleh sifat stokastik yang menyebabkan hasil yang bervariasi di setiap percobaan, serta pengaruh parameter seperti suhu dan cooling rate yang tidak selalu stabil.

Hasil akhir dari dari percobaan menunjukkan bahwa algoritma ini bergantung pada kondisi awal. Dengan parameter seperti suhu awal yang tinggi dan *cooling rate* yang stabil, algoritma mampu mengeksplorasi solusi secara luas sebelum akhirnya mengeksplorasi area sekitar solusi yang sudah ditemukan.

2.3.6 *Genetic Algorithm*

Kami melakukan percobaan dengan cara menjalankan algoritma sebanyak tiga kali dengan parameter jumlah populasi sebagai kontrol dan tiga kali dengan parameter banyaknya iterasi atau generasi sebagai kontrol. Untuk kontrol dengan parameter jumlah populasi yang kami tentukan adalah **100** dan variasi banyaknya iterasi atau generasi yang telah kami tentukan adalah **100**, **300**, dan **500**. Lalu untuk kontrol dengan banyaknya iterasi atau generasi yang telah kami tentukan adalah **200** dan variasi jumlah populasi yang telah kami tentukan adalah **50**, **100**, dan **200**. Setiap konfigurasi parameter masing-masing dijalankan tiga kali dan dari semua percobaan yang telah dilakukan, kami mendapatkan hasil yang berbeda-beda dari *Genetic Algorithm*. Berikut merupakan hasil yang kami dapatkan.

A. Jumlah Populasi : 100

Pada percobaan ini, jumlah populasi menjadi kontrol sehingga eksperimen akan dilakukan dengan jumlah populasi yang tetap dan generasi yang berubah-ubah.

1. Percobaan Pertama (Generasi 100)

- Hasil *objective function* yang didapat sebesar 931 dengan durasi sebesar **116.74 detik** dan melakukan iterasi (generasi) berjumlah 100 generasi. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

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

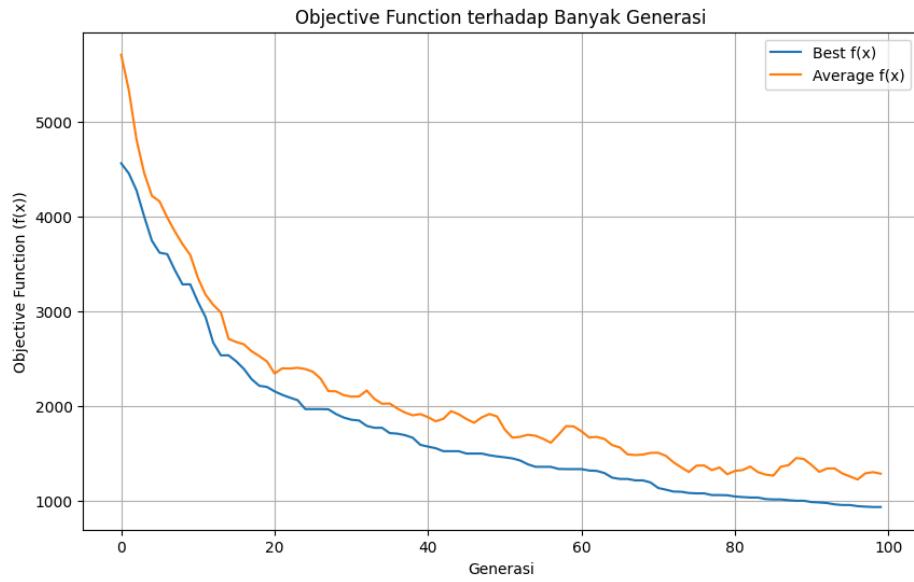
Gambar 2. Visualisasi *Cube* Awal 1 Generasi 100

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan *genetic algorithm* dengan generasi berjumlah 100.

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

Gambar 2. Visualisasi *Cube* Akhir 1 Generasi 100

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 1

- b. Hasil *objective function* yang didapat sebesar 927 dengan durasi sebesar **117.70 detik** dan melakukan iterasi (generasi) berjumlah 100 generasi. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

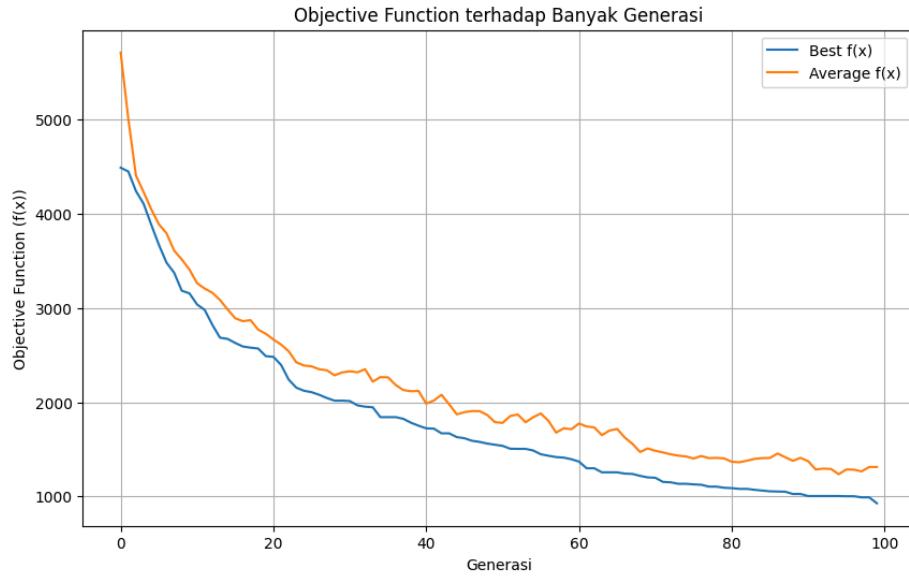


Gambar 2. Visualisasi *Cube* Awal 2 Generasi 100
Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan *genetic algorithm* dengan generasi berjumlah 100.



Gambar 2. Visualisasi *Cube* Awal 2 Generasi 100

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 2

- c. Hasil *objective function* yang didapat sebesar 793 dengan durasi sebesar **118.05 detik** dan melakukan iterasi (generasi) berjumlah 100 generasi. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

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

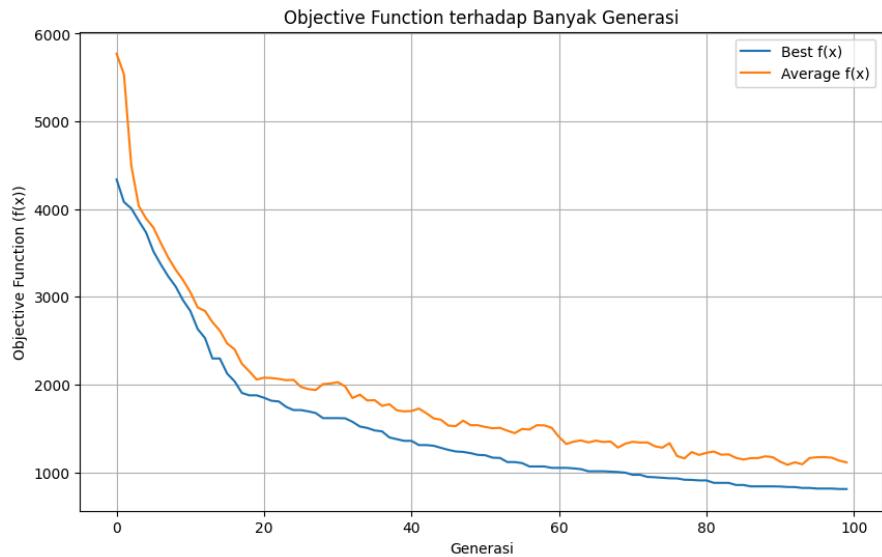
Gambar 2. Visualisasi *Cube* Awal 3 Generasi 100

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan *genetic algorithm* dengan generasi berjumlah 100.

Visualisasi Diagonal Magic Cube														
Layer 1					Layer 2			Layer 3		Layer 4		Layer 5		
49	98	57	10	115	28	101	85	46	63	48	77	11	108	75
104	21	72	42	73	67	102	123	17	2	65	82	80	14	76
105	110	26	41	27	12	23	94	124	8	47	36	92	15	120
13	86	68	109	37	114	30	9	32	125	87	66	43	100	38
45	1	96	122	51	103	59	7	34	112	71	56	91	97	6

Gambar 2. Visualisasi *Cube* Awal 3 Generasi 100

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi (Iterasi) 3

2. Percobaan Kedua (Generasi 300)

- a. Hasil *objective function* yang didapat sebesar 313 dengan durasi sebesar **353.13 detik** dan melakukan iterasi (generasi) berjumlah 300 generasi. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
44	105	12	110	71		31	98	10	104	36		52	47	82	46	117
125	80	103	51	101		68	81	84	123	38		2	113	26	74	56
37	91	75	66	77		96	108	118	106	39		8	1	97	90	13
72	14	5	100	21		111	7	50	85	95		79	20	41	59	11
83	16	76	53	48		87	115	30	24	70		33	17	45	94	25

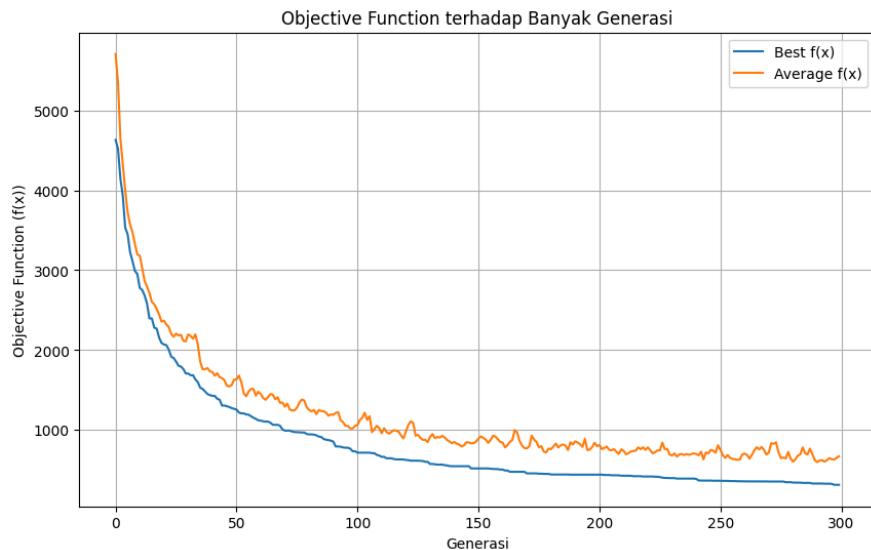
Gambar 2. Visualisasi *Cube* Awal 1 Generasi 300

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan *genetic algorithm* dengan generasi berjumlah 300.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
58	88	97	17	54		112	50	56	29	67		96	82	3	105	28
61	44	23	108	77		25	81	122	5	76		20	21	98	114	62
32	43	85	102	52		63	87	30	89	47		66	69	86	7	91
104	10	70	64	68		12	94	14	83	117		120	95	39	19	42
59	123	40	24	65		118	4	93	103	8		9	48	90	75	92

Gambar 2. Visualisasi *Cube* Akhir 1 Generasi 300

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi (Iterasi) 1

b. Hasil *objective function* yang didapat sebesar 367 dengan durasi sebesar **351.21 detik** dan melakukan iterasi (generasi) berjumlah 300 generasi. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

Visualisasi Diagonal Magic Cube																			
Layer 1				Layer 2				Layer 3				Layer 4				Layer 5			
114	28	85	50	5	78	73	111	116	69	86	43	115	15	110	21	76	88	62	107
77	17	57	34	23	103	6	12	120	87	102	94	112	52	67	47	121	105	113	90
32	68	35	7	65	44	16	18	20	98	124	39	99	63	48	49	75	46	24	37
72	14	3	10	53	22	19	106	60	27	123	29	58	11	42	8	54	36	84	82
79	108	61	109	97	41	119	96	40	13	55	81	104	93	2	51	38	117	92	30

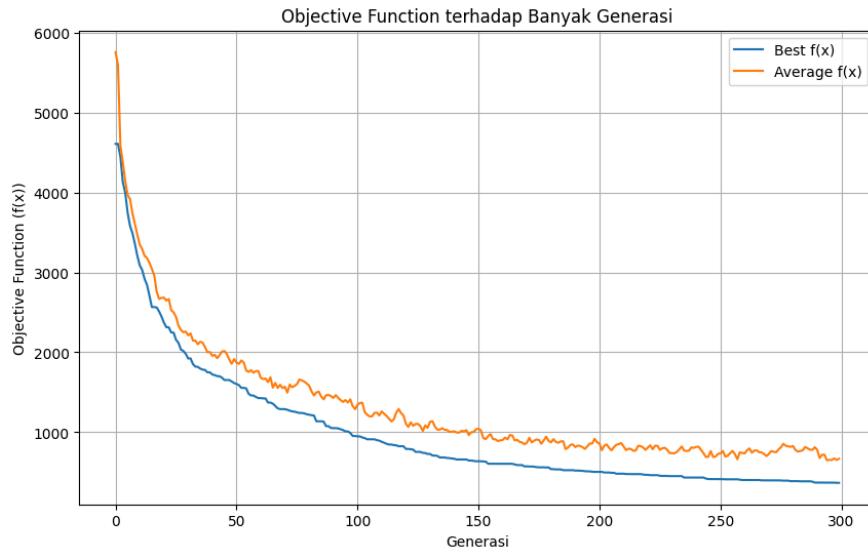
Gambar 2. Visualisasi *Cube* Awal 2 Generasi 300

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan *genetic algorithm* dengan generasi berjumlah 300.

Visualisasi Diagonal Magic Cube																			
Layer 1				Layer 2				Layer 3				Layer 4				Layer 5			
50	90	5	88	82	15	8	98	112	80	74	71	48	40	75	107	47	70	22	76
108	97	109	2	7	68	72	32	27	114	29	30	122	91	43	93	44	35	106	31
21	10	96	101	84	121	116	14	42	25	56	81	57	6	119	77	26	39	117	52
62	58	4	64	124	1	83	49	113	69	100	37	67	92	19	28	85	118	24	60
78	63	103	61	11	110	34	123	18	33	55	95	23	87	59	9	111	46	45	102

Gambar 2. Visualisasi *Cube* Awal 2 Generasi 300

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 2

- c. Hasil *objective function* yang didapat sebesar 365 dengan durasi sebesar **352.45 detik** dan melakukan iterasi (generasi) berjumlah 300 generasi. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.



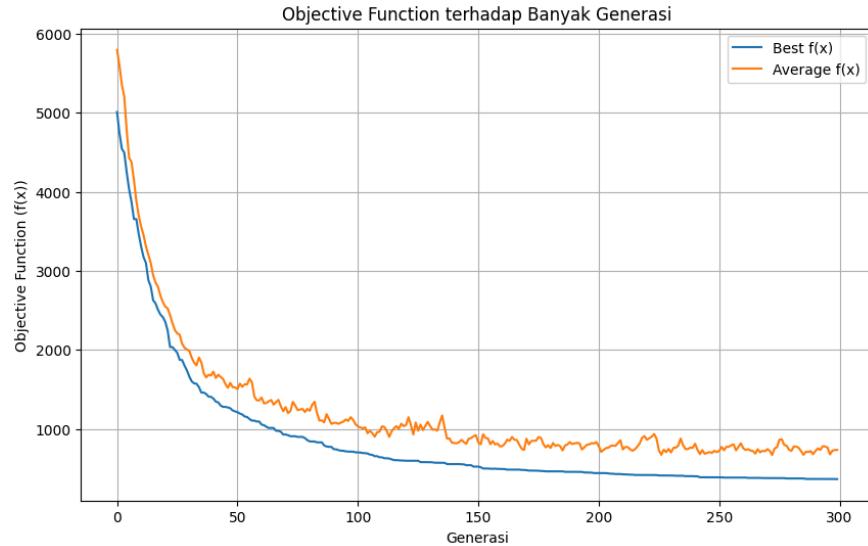
Gambar 2. Visualisasi *Cube* Awal 3 Generasi 300

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan *genetic algorithm* dengan generasi berjumlah 300.



Gambar 2. Visualisasi *Cube* Awal 3 Generasi 300

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 3

3. Percobaan Ketiga (Generasi 500)

- a. Hasil *objective function* yang didapat sebesar 269 dengan durasi sebesar **595.31 detik** dan melakukan iterasi (generasi) berjumlah 500 generasi. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

Visualisasi Diagonal Magic Cube														
Layer 1					Layer 2		Layer 3			Layer 4		Layer 5		
19	38	20	106	69	119	5	37	66	118	62	7	93	108	59
121	111	29	81	24	75	47	30	109	9	124	89	94	32	125
123	10	36	100	54	105	96	17	3	101	48	41	99	33	97
8	51	91	49	88	86	77	122	15	84	120	14	76	71	90
16	80	2	57	40	98	61	74	34	44	18	104	28	45	25

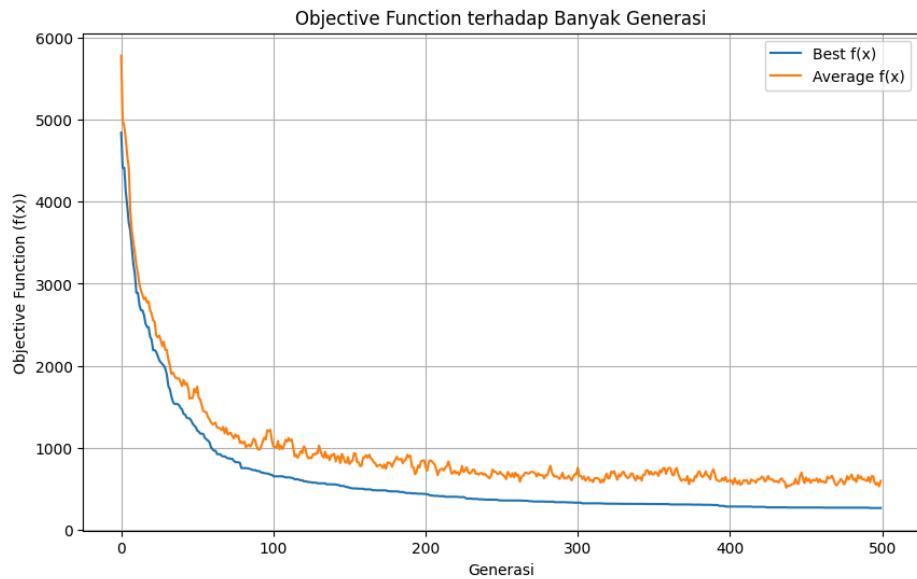
Gambar 2. Visualisasi *Cube* Awal 1 Generasi 500

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan *genetic algorithm* dengan generasi berjumlah 500.

Visualisasi Diagonal Magic Cube																			
Layer 1				Layer 2				Layer 3				Layer 4							
96	71	88	45	20	8	11	64	116	120	58	63	46	30	117	84	125	25	14	53
101	62	37	66	49	78	105	22	56	54	15	82	106	67	47	118	26	31	87	51
13	70	18	103	108	109	83	52	42	29	102	90	65	24	34	74	36	89	98	19
7	104	57	81	60	93	59	100	55	17	73	1	33	97	112	68	86	61	95	5
99	9	115	16	77	27	72	79	43	94	12	23	50	123	110	114	76	3	80	41

Gambar 2. Visualisasi *Cube Akhir 1 Generasi 500*

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi (Iterasi) 1

- b. Hasil *objective function* yang didapat sebesar 397 dengan durasi sebesar **599.15 detik** dan melakukan iterasi (generasi) berjumlah 500 generasi. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

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

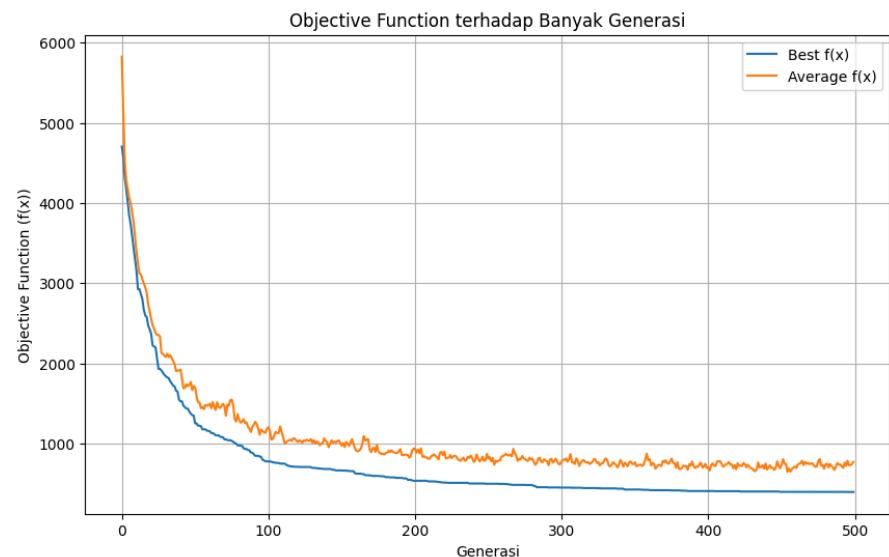
Gambar 2. Visualisasi *Cube* Awal 2 Generasi 500

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan *genetic algorithm* dengan generasi berjumlah 500.

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

Gambar 2. Visualisasi *Cube* Akhir 2 Generasi 500

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi (Iterasi) 2

c. Hasil *objective function* yang didapat sebesar 353 dengan durasi sebesar **606.69 detik** dan melakukan iterasi (generasi) berjumlah 500 generasi. Berikut merupakan visualisasi *state* awal *diagonal magic cube*.

Visualisasi Diagonal Magic Cube																			
Layer 1				Layer 2				Layer 3				Layer 4				Layer 5			
105	50	101	71	6	113	76	66	122	27	41	9	55	64	118	34	82	45	114	61
79	99	98	56	69	25	17	21	54	60	16	1	124	18	14	106	116	102	26	84
52	57	104	80	3	32	75	24	35	13	53	63	49	77	31	88	28	4	85	43
51	73	15	67	12	7	29	115	100	48	110	103	39	22	37	91	81	86	47	108
62	92	87	95	19	5	46	72	36	70	40	107	2	94	97	30	74	117	10	93

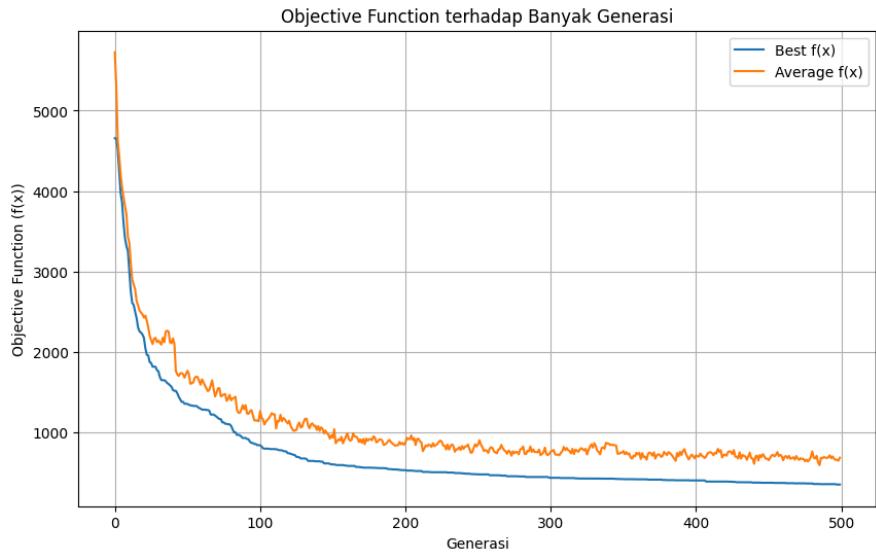
Gambar 2. Visualisasi *Cube* Awal 3 Generasi 500

Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan *genetic algorithm* dengan generasi berjumlah 500.

Visualisasi Diagonal Magic Cube																			
Layer 1				Layer 2				Layer 3				Layer 4				Layer 5			
124	57	55	12	65	83	94	39	4	98	32	119	22	113	29	45	5	97	120	49
28	63	43	71	110	101	19	114	64	17	75	66	109	47	18	8	104	24	87	92
31	14	107	121	42	91	122	38	20	44	10	59	100	58	88	116	33	34	54	80
117	56	41	13	90	1	77	9	123	105	112	50	51	16	85	81	79	96	48	11
15	125	69	99	7	36	3	115	108	53	86	21	30	82	95	70	93	67	2	84

Gambar 2. Visualisasi *Cube* Akhir 3 Generasi 500

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 3

B. Jumlah Generasi : 200

Pada percobaan ini, jumlah generasi menjadi kontrol sehingga eksperimen akan dilakukan dengan jumlah generasi yang tetap dan populasi yang berubah-ubah.

1. Percobaan Pertama (Populasi 50)

- Hasil *objective function* yang didapat sebesar 451 dengan durasi sebesar **116.61 detik** dengan jumlah populasi sebesar 50 dan generasi sebanyak 200.

Visualisasi Diagonal Magic Cube																			
Layer 1					Layer 2					Layer 3					Layer 4				
113	31	122	61	2	89	99	92	115	22	4	26	29	111	52	125	103	47	87	6
108	27	105	21	40	83	43	1	12	106	64	118	70	54	110	109	41	112	98	28
100	13	71	65	50	59	10	14	79	74	16	101	102	60	42	116	17	119	77	75
49	95	35	3	32	76	58	45	123	8	11	9	117	107	48	34	82	124	62	86
20	37	57	96	15	5	46	73	7	38	55	121	24	84	97	33	67	120	53	88

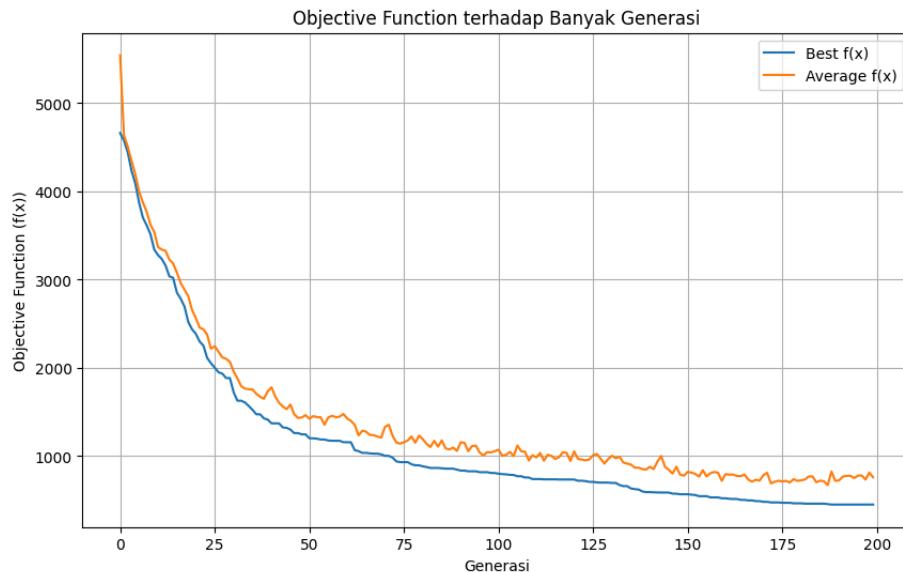
Gambar 2. Visualisasi *Cube* Awal 1 Populasi 50

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan *genetic algorithm* dengan populasi berjumlah 50.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
40	80	2	123	73		114	3	70	5	121		93	59	49	56	50
31	109	125	35	11		118	82	34	68	19		69	44	104	28	74
105	1	26	75	106		12	103	89	95	14		7	63	66	119	60
76	117	52	36	32		62	20	83	33	113		98	120	37	23	47
57	8	110	42	97		10	108	38	116	46		51	29	58	90	86
												84	65	77	6	85
												92	99	45	71	9

Gambar 2. Visualisasi *Cube Akhir 1 Populasi 50*

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 1

- b. Hasil *objective function* yang didapat sebesar 473 dengan durasi sebesar **117.02 detik** dengan jumlah populasi sebesar 50 dan generasi sebanyak 200.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
69	19	87	25	53		80	20	41	57	101		72	18	73	3	99
23	7	98	15	64		116	77	51	8	49		124	102	106	71	33
122	117	84	30	5		123	66	113	4	109		93	16	97	32	112
111	59	120	28	50		92	75	22	27	100		89	36	62	31	104
110	107	95	70	125		81	67	105	45	68		114	91	21	118	35
												108	56	90	119	76
												1	65	79	54	11

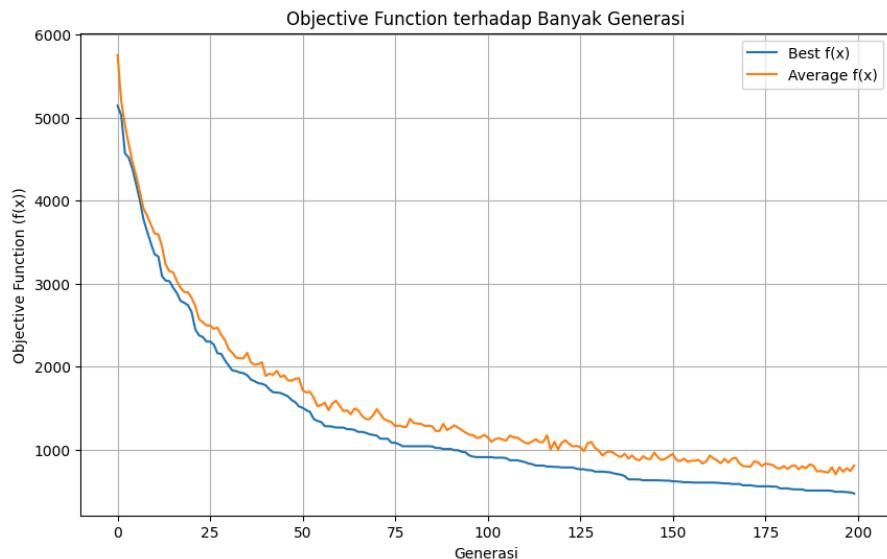
Gambar 2. Visualisasi *Cube* Awal 2 Populasi 50

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan *genetic algorithm* dengan populasi berjumlah 50.

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

Gambar 2. Visualisasi *Cube* Akhir 2 Populasi 50

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 2

- c. Hasil *objective function* yang didapat sebesar 501 dengan durasi sebesar **115.99 detik** dengan jumlah populasi sebesar 50 dan generasi sebanyak 200

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
13	124	98	59	90		27	111	83	21	63		80	73	18	32	101
120	1	105	17	3		122	118	75	8	51		88	70	9	102	37
60	36	65	67	43		22	54	49	108	74		61	6	39	95	11
7	119	87	48	24		40	57	117	34	85		28	92	23	52	103
20	72	50	68	79		5	62	31	96	45		93	109	2	29	115
												97	30	84	78	44
												121	99	113	110	86
												107	76	35	71	66
												82	16	56	69	106
												33	125	25	47	91
												12	53	41	4	42

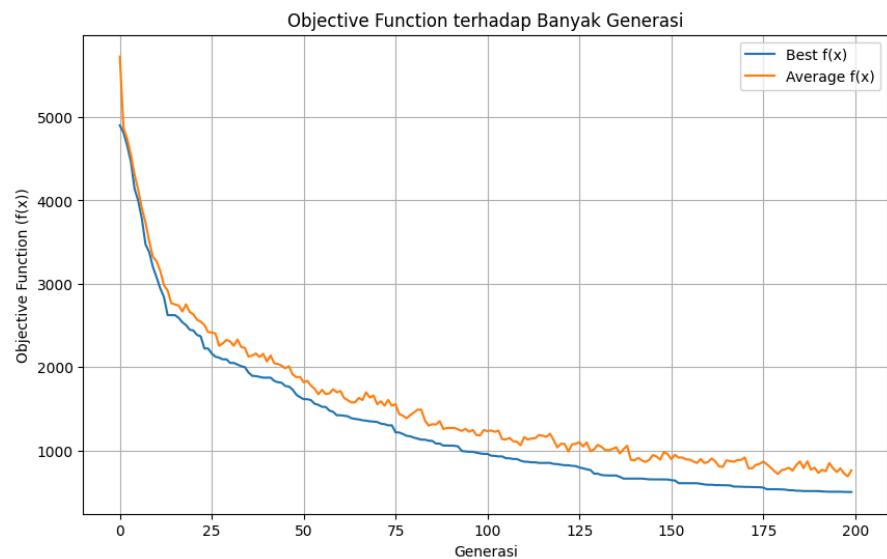
Gambar 2. Visualisasi *Cube* Awal 3 Populasi 50

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan *genetic algorithm* dengan populasi berjumlah 50.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
48	81	5	59	100		64	68	35	53	98		63	1	88	76	84
123	50	92	10	38		14	97	16	86	99		45	85	111	42	31
9	124	57	41	105		113	78	17	101	3		91	19	73	107	23
15	34	108	122	33		29	43	125	47	69		114	103	13	6	83
117	26	54	82	36		93	30	116	27	49		11	106	25	89	90
												67	96	8	70	71
												80	118	109	7	12
												115	2	4	121	72
												37	79	75	21	104
												51	74	20	110	61
												28	55	112	52	65

Gambar 2. Visualisasi *Cube* Akhir 3 Populasi 50

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi (Iterasi) 3

2. Percobaan Kedua (Populasi 100)

- a. Hasil *objective function* yang didapat sebesar 650 dengan durasi sebesar **238.82 detik** dengan jumlah populasi sebesar 100 dan generasi sebanyak 200.

Visualisasi Diagonal Magic Cube																
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5				
45	120	22	41	108		99	50	33	34	64		114	98	40	19	54
32	57	86	62	38		46	94	7	61	71		56	102	13	95	124
74	26	117	37	104		1	47	28	59	65		88	44	58	81	23
93	42	18	20	9		91	107	118	112	121		85	110	11	43	36
96	77	70	55	27		14	100	25	113	119		60	92	66	105	15
												76	82	111	84	2
												89	103	35	48	101

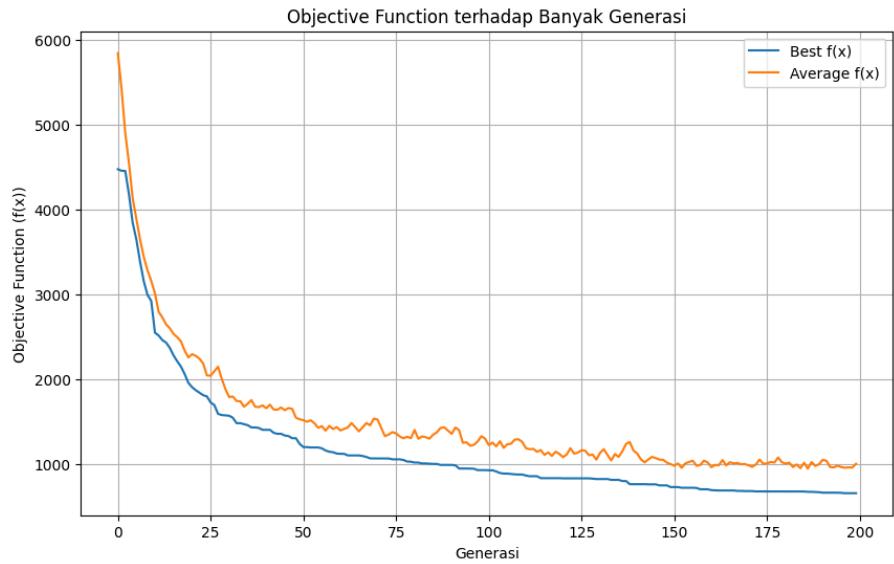
Gambar 2. Visualisasi *Cube* Awal 1 Populasi 100

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan *genetic algorithm* dengan populasi berjumlah 100.

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

Gambar 2. Visualisasi *Cube* Akhir 1 Populasi 100

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 1

b. Hasil *objective function* yang didapat sebesar 526 dengan durasi sebesar **245.28 detik** dengan jumlah populasi sebesar 100 dan generasi sebanyak 200.

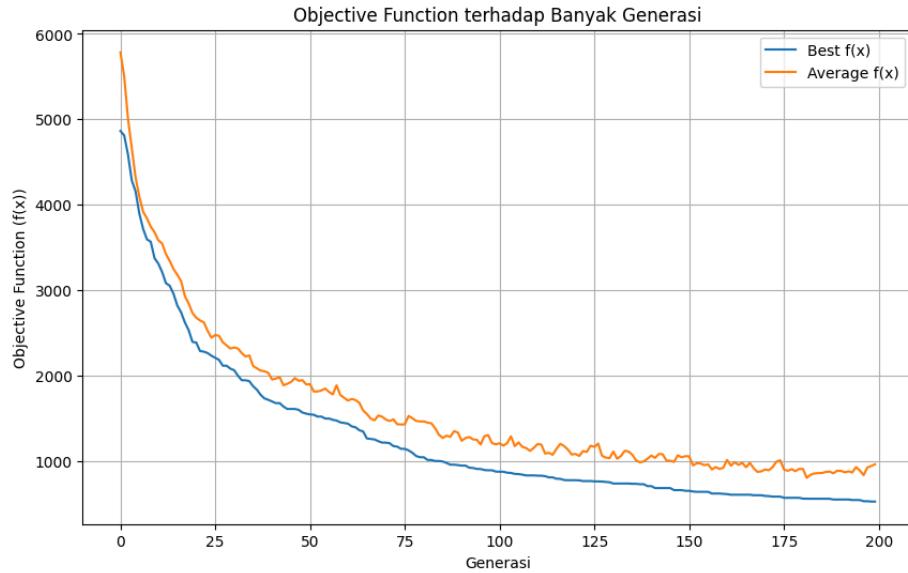


Gambar 2. Visualisasi *Cube* Awal 2 Populasi 100
Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan *genetic algorithm* dengan populasi berjumlah 100.



Gambar 2. Visualisasi *Cube* Akhir 2 Populasi 100

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 2

- c. Hasil *objective function* yang didapat sebesar 527 dengan durasi sebesar **241.97 detik** dengan jumlah populasi sebesar 100 dan generasi sebanyak 200.

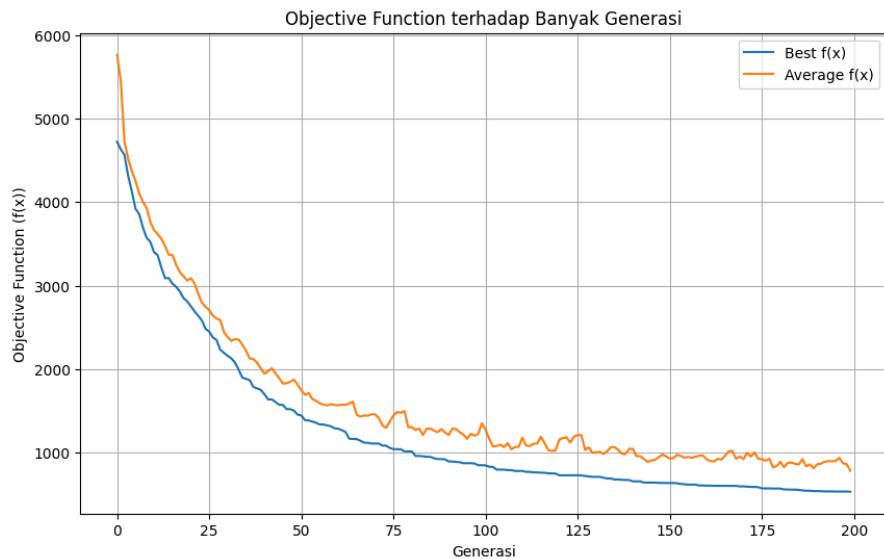
Visualisasi Diagonal Magic Cube														
Layer 1					Layer 2			Layer 3		Layer 4		Layer 5		
69	105	80	114	39	62	28	100	96	115	42	43	120	108	11
35	50	122	36	102	33	34	87	37	9	124	31	113	13	75
64	85	44	70	48	12	59	17	6	1	119	88	112	38	55
111	77	84	125	41	5	21	16	20	10	118	73	63	116	107
117	26	92	68	79	49	29	67	60	95	15	91	3	93	74

Gambar 2. Visualisasi *Cube* Awal 3 Populasi 100
Berikut merupakan visualisasi *state* akhir *diagonal magic cube* setelah diterapkan *genetic algorithm* dengan populasi berjumlah 100.

Visualisasi Diagonal Magic Cube																	
Layer 1					Layer 2					Layer 3					Layer 4		
67	117	33	10	97	31	75	109	57	47	24	68	91	29	89	101	35	14
64	50	27	110	65	108	125	38	30	7	56	82	9	60	121	41	32	119
86	58	43	106	16	42	1	55	123	96	100	95	73	34	8	2	84	59
80	40	104	77	15	36	99	44	12	102	81	23	28	107	78	118	61	113
18	45	114	11	122	98	5	63	85	66	52	48	112	83	21	54	105	6

Gambar 2. Visualisasi *Cube Akhir 3 Populasi 100*

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 3

3. Percobaan Ketiga (Populasi 200)

- a. Hasil *objective function* yang didapat sebesar 517 dengan durasi sebesar **482.62 detik** dengan jumlah populasi sebesar 200 dan generasi sebanyak 200.

Visualisasi Diagonal Magic Cube																		
Layer 1			Layer 2				Layer 3			Layer 4				Layer 5				
105	33	1	118	65			90	48	108	27	47			29	88	43	86	70
77	32	97	39	74			100	89	6	68	52			63	82	21	49	101
3	78	59	122	53			51	4	72	94	107			83	42	124	56	8
91	116	87	10	11			58	109	76	13	62			111	9	25	54	125
37	55	93	18	113			20	79	45	120	50			30	95	98	67	26
														121	22	75	92	17

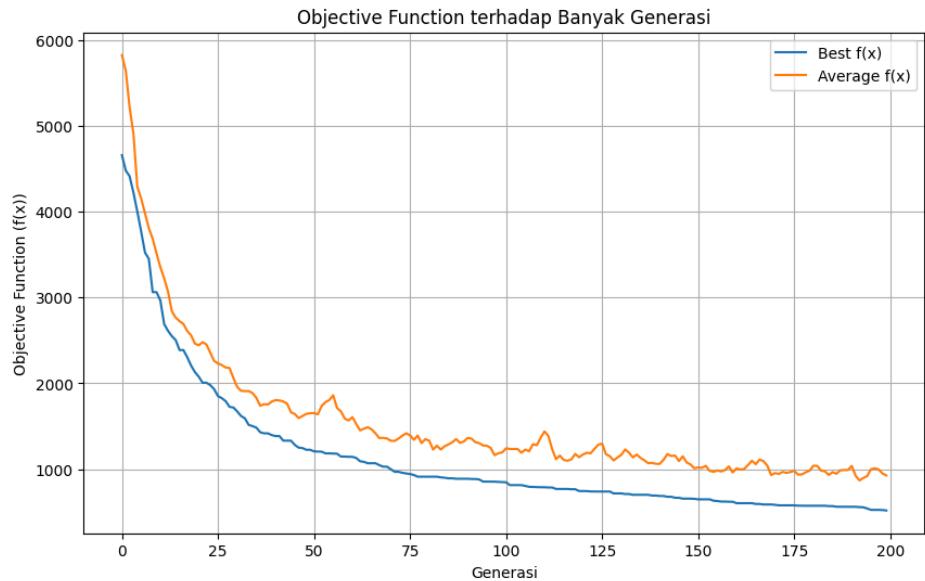
Gambar 2. Visualisasi *Cube Awal* 1 Populasi 200

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan *genetic algorithm* dengan populasi berjumlah 200.

Visualisasi Diagonal Magic Cube																		
Layer 1			Layer 2				Layer 3			Layer 4				Layer 5				
105	33	1	118	65			90	48	108	27	47			29	88	43	86	70
77	32	97	39	74			100	89	6	68	52			63	82	21	49	101
3	78	59	122	53			51	4	72	94	107			83	42	124	56	8
91	116	87	10	11			58	109	76	13	62			111	9	25	54	125
37	55	93	18	113			20	79	45	120	50			30	95	98	67	26
														121	22	75	92	17

Gambar 2. Visualisasi *Cube Akhir* 1 Populasi 200

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi (Iterasi) 1

- b. Hasil *objective function* yang didapat sebesar 553 dengan durasi sebesar **473.20 detik** dengan jumlah populasi sebesar 200 dan generasi sebanyak 200.

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

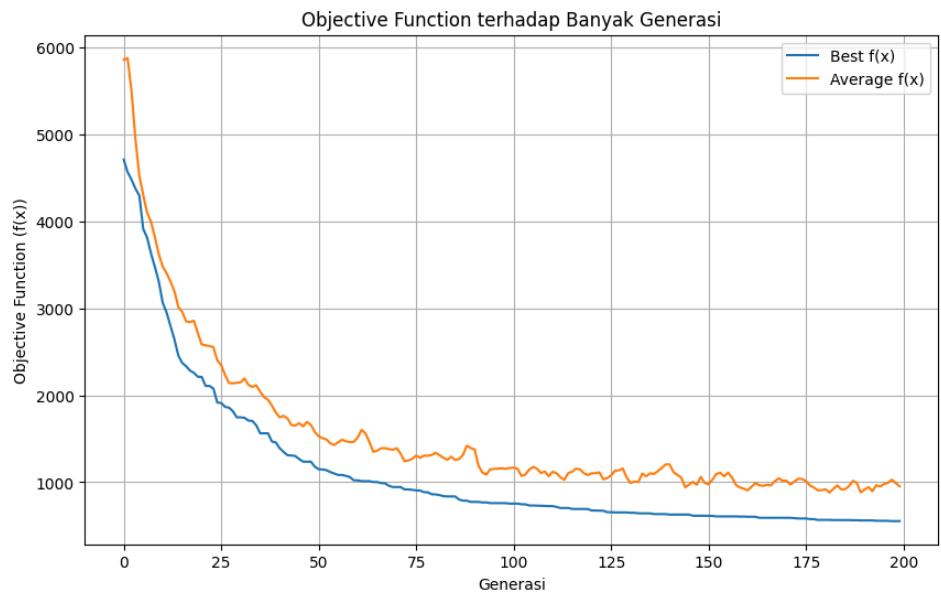
Gambar 2. Visualisasi *Cube* Awal 2 Populasi 200

Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan *genetic algorithm* dengan populasi berjumlah 100.

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

Gambar 2. Visualisasi *Cube* Akhir 2 Populasi 200

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 2

- c. Hasil *objective function* yang didapat sebesar 513 dengan durasi sebesar **476.04 detik** dengan jumlah populasi sebesar 200 dan generasi sebanyak 200.

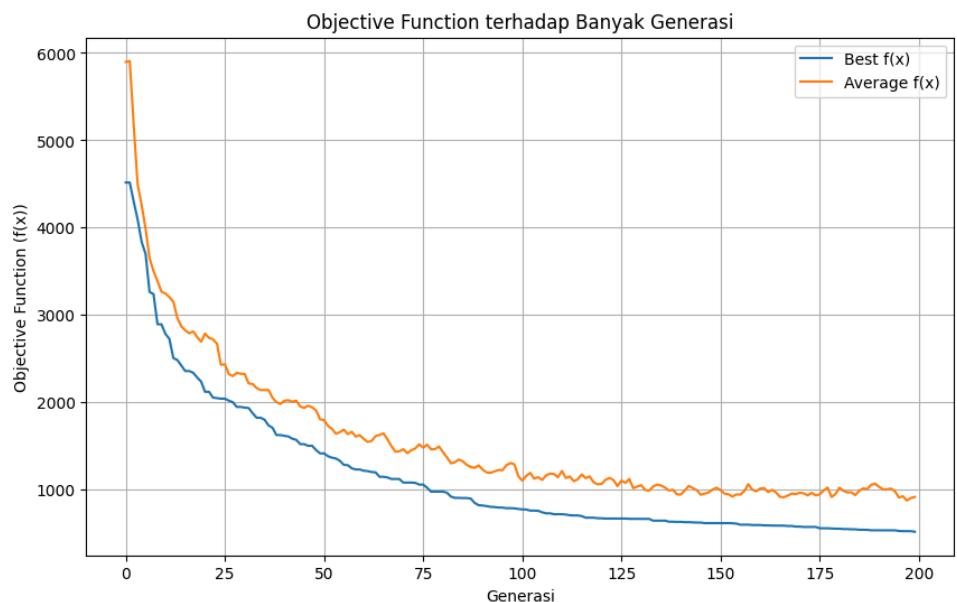


Gambar 2. Visualisasi *Cube* Awal 3 Populasi 200
Berikut merupakan visualisasi *state akhir diagonal magic cube* setelah diterapkan *genetic algorithm* dengan populasi berjumlah 200.

Visualisasi Diagonal Magic Cube																	
Layer 1			Layer 2			Layer 3			Layer 4			Layer 5					
104	116	34	42	19		72	91	24	122	8		38	7	69	110	88	
48	40	98	96	33		45	37	125	53	56		124	100	30	9	47	
81	49	73	5	108		23	101	64	18	109		76	61	63	112	2	
21	71	32	75	120		106	90	89	26	22		114	17	117	50	14	
58	43	77	105	31		85	3	12	113	118		35	86	66	11	115	
												51	44	119	68	29	
												84	123	46	13	27	

Gambar 2. Visualisasi *Cube Akhir 3 Populasi 200*

Selain gambar visual terkait state untuk *Magic Cube*, untuk meningkatkan pemahaman terkait plot nilai *objective function* kepada setiap iterasi, grafik plot *objective function* terhadap banyak iterasi yang telah dilewati akan ditunjukkan.



Gambar 2. Plot *Objective Function* terhadap banyak Generasi
(Iterasi) 3

Dari hasil hasil tersebut dapat dilakukan analisis tentang hasil eksperimen tersebut yang kami jabarkan melalui tabel berikut.

Tabel 2.4 Analisis Hasil Eksperimen *Genetic Algorithm*

Analisis	Hasil (Average)

Hasil Algoritma	523,67
Waktu yang diperlukan	317.67 s
Kedekatan dengan <i>Global Optima (Proximity)</i>	98,47%
Konsistensi Hasil	36,78%

Dari hasil eksperimen, didapatkan *proximity* sebesar 98, 47% terhadap global optimum, yang menunjukkan bahwa algoritma ini mampu mencari solusi yang sangat mendekati solusi optimal. Dari grafik yang dihasilkan, pada awal iterasi nilai *objective function* menurun dengan cepat. Namun seiring berjalannya waktu, algoritma akan mendekati lokal optimal dan kesulitan dalam menemukan *neighbor* yang lebih baik. Hasil akhir dari percobaan menunjukkan bahwa algoritma ini bergantung pada kondisi awal. Algoritma ini juga sangat bergantungan dengan parameter populasi dan iterasinya serta fase-fase pada metode *selection*, *crossover*, dan *mutation*. Disebabkan hasil dari fase-fase yang ada pada *genetic algorithm*, konsistensi hasil yang dimiliki oleh algoritma ini cukup rendah karena hasil dari logika ini dapat bersifat naik turun.

2.3.7 Analisis Keseluruhan

Berdasarkan hasil hasil dari seluruh algoritma *local search* yang telah kami coba, berikut merupakan perbandingan dari hasil hasil tersebut.

Tabel 2.7 Analisis Keseluruhan Algoritma

Analisis	<i>Steepest Ascent</i>	<i>Sideways move</i>	<i>Random Restart</i>	<i>Stochastic</i>	<i>Simulated Annealing</i>	<i>Genetic</i>
Hasil (<i>Objective Function</i>)	595	615,67	359,3	2993	150, 667	523,67

Waktu	377,4s	392,8s	1932,88 s	382,3s	25,36 s	317,67s
<i>Proximity</i>	98,27%	98,2%	98,95%	91,3%	99, 56%	98,47%
Konsistensi	28.86%	24,45%	8,72%	10,38%	43, 6%	36,78%

Berdasarkan hasil, *simulated annealing* memiliki nilai terbaik sebesar 150,667 (*Average*) dan *stochastic* memiliki nilai terburuk sebesar 2993. Hal ini menunjukkan bahwa *simulated annealing* memiliki tingkat efektivitas tertinggi dan *stochastic* terendah. Dilanjut dengan *random restart* dengan nilai 359,3 sebagai terbaik kedua, *genetic algorithm* sebesar 523,67, *steepest ascent* sebesar 595, dan *sideways move* sebesar 615,67.

Berdasarkan waktu, *simulated annealing* menghasilkan durasi pencarian terpendek yaitu sebesar 23,56 detik (*Average*), yang menunjukkan efisiensi tinggi dalam menemukan solusi mendekati optimal. Algoritma lain seperti *Random Restart* memerlukan waktu jauh lebih lama yaitu sebesar 1932,88 detik, ini menunjukkan biaya waktu yang tinggi untuk mencapai hasil yang serupa. Dilanjut dengan *genetic algorithm* sebesar 317,67 detik, *steepest ascent* sebesar 377,4 detik, *stochastic* sebesar 382,3 detik, dan *sideways move* sebesar 392,8 detik.

Berdasarkan konsistensi, *random restart* memiliki konsistensi terbaik sebesar 8,72% (semakin mendekati 0% semakin konsisten) dan *simulated annealing* memiliki konsistensi terburuk yaitu 43,6% dimana menunjukkan hasil yang dihasilkan akan sangat bervariasi. Walaupun begitu, hal ini tidak benar benar berarti apa apa karena walaupun tidak konsisten, hasil dari *simulated annealing* akan selalu lebih baik dari *random restart*.

Dapat dilihat bahwa *simulated annealing* merupakan algoritma terbaik dari sisi efisiensi dan efektivitas. Walaupun *random restart* merupakan algoritma paling konsisten dan memiliki hasil terbaik kedua, tetapi algoritma *random*

restart kurang efisien karena membutuhkan waktu yang sangat lama untuk menemukan solusinya. Di sisi lain, *stochastic* merupakan algoritma terburuk dari hasil yang ada dimana algoritma *stochastic* tidak efisien maupun efektif. *Genetic algorithm* merupakan algoritma terbaik kedua dari sisi efisiensi dan efektivitas, dilanjut dengan *steepest ascent*, *sideways move*, dan *random restart*.

BAB III

KESIMPULAN DAN SARAN

Local Search Algorithm yang digunakan untuk menyelesaikan permasalahan *Diagonal Magic Cube 5x5x5* memberikan beberapa hasil penelitian yang menyatakan bahwa *local search* mungkin mampu untuk menyelesaikan sebuah *Diagonal Magic Cube* apabila memiliki konfigurasi dan jumlah parameter yang sesuai. Beberapa algoritma yang diujikan adalah *Steepest Ascent Hill Climbing*, *Hill Climbing with Sideways Move*, *Random Restart Hill Climbing*, *Stochastic Hill Climbing*, *Simulated Annealing*, dan *Genetic Algorithm*.

Dengan melakukan pengujian berulang dan mengevaluasi hasil dari algoritma, waktu eksekusi, dan konsistensi hasil, hasil eksperimen menunjukkan bahwa *Steepest Ascent Hill Climbing* dan *Hill Climbing with Sideways Move* efektif mendekati solusi optimal namun memiliki risiko terjebak di *local optimum*. *Random Restart Hill Climbing* berhasil mendapatkan solusi lebih baik dengan konsistensi tinggi, tetapi mengonsumsi waktu yang cukup lama. *Simulated Annealing* efektif menghindari jebakan *local optimum* melalui pendekatan stokastik, menghasilkan solusi yang mendekati optimal dengan waktu eksekusi relatif singkat. *Genetic Algorithm* menawarkan keseimbangan antara waktu eksekusi karena memiliki kemampuan untuk mencari kemungkinan secara lebih luas sehingga dapat memberikan hasil yang lebih baik walaupun timbal balik yang diberikan adalah durasi dan *cost* yang cukup besar.

Selain itu, ada beberapa faktor yang memengaruhi hasil dari algoritma-algoritma yang ada, yaitu parameter konfigurasi, jumlah *neighbor*, pengaturan *restart* dan *randomness*, jumlah iterasi, dan mekanisme eksplorasi. Contohnya parameter konfigurasi sangat berpengaruh kepada *genetic algorithm* dan *simulated annealing*. Jumlah *neighbor* memengaruhi hasil ke seluruh algoritma *local search* namun mengorbankan efisiensi algoritma. Pengaturan *restart* dan *randomness* berpengaruh kepada *random restart* dan *stochastic*. Terakhir, mekanisme eksplorasi tentunya sangat berpengaruh kepada hasil, misalnya pada *simulated annealing* yang mengizinkan

eksplorasi lebih jauh kepada nilai nilai yang ada, menjadikan *simulated annealing* dapat menghasilkan nilai yang sangat baik.

Terakhir, untuk mencapai kinerja optimal, ada baiknya untuk melakukan tuning pada parameter-parameter seperti ukuran populasi, jumlah generasi, dan cooling rate, terutama untuk algoritma berbasis parameter seperti *Genetic Algorithm* dan *Simulated Annealing*. Selain itu, dapat menggabungkan kelebihan dari *Genetic Algorithm* dan *Simulated Annealing*, atau algoritma stokastik lainnya, sehingga dapat memperluas eksplorasi solusi sambil tetap menjaga efisiensi waktu.

BAB IV

PEMBAGIAN TUGAS

Berikut merupakan tabel rincian pembagian tugas dalam pengeroaan tugas besar 1 oleh anggota kelompok 51 :

Tabel 4.1 Pembagian Tugas

No	Nama	NIM	Tugas
1	Moh Afnan Fawaz	18222111	Mengerjakan laporan dan membuat <i>algoritma local search Steepest Ascent Hill Climb, Random Restart Hill Climb, dan Sideways Move Hill Climb</i>
2	Aqila Ataa	18222120	Mengerjakan laporan dan membuat <i>algoritma local search Stochastic</i> dan class MagicCube
3	Gymnastiar Anwar	18222121	Mengerjakan laporan dan membuat <i>algoritma local search Simulated Annealing</i>
4	Fadian Alif Mahardika	18222124	Mengerjakan laporan dan membuat <i>algoritma local search Genetic Algorithm</i>

REFERENSI

1. Spesifikasi Tugas Besar 1 IF3070 Dasar Inteligensi Artifisial 2024/2025.
2. Slide Kuliah IF3070 - Dasar Inteligensi Artifisial 2024.
3. Stuart J Russell & Peter Norvig, Artificial Intelligence: A Modern Approach, 4th Edition, Prentice-Hall International, Inc, 2022, Textbook
4. Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. Springer Science & Business Media.
5. Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1974). *A Comparative Study of Methods for Combinatorial Optimization*. SIAM Journal on Computing, 3(4), 268-281.