LAPORAN TUGAS BESAR 1

IF3070 Dasar Inteligensi Artifisial

Pencarian Solusi Diagonal Magic Cube dengan Local Search

Disusun oleh:

Kelompok 39

Habib Akhmad Al Farisi / 18222029
Alfandito Rais Akbar / 18222037
Winata Tristan / 18222061
Muhammad Rafi D. / 18222069

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung
Jl. Ganesha 10, Bandung 40132

	Program Studi Sistem dan Teknologi Informasi STEI – ITB	Kelompok	Jumlah Halaman
		K39	**86**

Daftar Isi

Bab I
Deskripsi Persoalan

Diagonal Magic Cube merupakan permasalahan penyusunan angka dalam bentuk kubus dengan properti khusus. Pada tugas ini, digunakan kubus berukuran 5x5x5 yang harus diisi dengan angka 1 hingga 125 (5^3) tanpa pengulangan. Penyusunan angka-angka tersebut harus memenuhi beberapa properti berikut:

A. **Magic Number**:
    a. Terdapat satu angka yang merupakan magic number (315)
    b. Magic number tidak harus dalam rentang 1-125
    c. Magic number bukan merupakan angka yang dimasukkan dalam kubus
B. **Properti Jumlah** Setiap properti berikut harus memiliki jumlah yang sama dengan magic number (315):
    a. Setiap baris dalam kubus
    b. Setiap kolom dalam kubus
    c. Setiap tiang dalam kubus
    d. Seluruh diagonal ruang pada kubus
    e. Seluruh diagonal pada potongan bidang kubus
C. State dan Transisi
    a. Initial state: Susunan acak angka 1-125 dalam kubus 5x5x5
    b. Transisi: Menukar posisi dua angka dalam kubus (tidak harus bersebelahan)
    c. Goal state: Kubus yang memenuhi semua properti magic number
D. **Objective Function** Fungsi objektif dihitung berdasarkan jumlah garis (baris, kolom, tiang, diagonal) yang memiliki jumlah sama dengan magic number (315). Semakin banyak garis yang memenuhi syarat, semakin tinggi nilai objective function.
E. Batasan:
    a. Angka yang digunakan: 1 hingga 125
    b. Setiap angka hanya boleh muncul sekali
    c. Perpindahan hanya boleh dilakukan dengan menukar posisi dua angka
    d. Magic number tetap (315)
F. Tantangan:
    a. Kompleksitas ruang pencarian yang sangat besar
    b. Banyaknya constraint yang harus dipenuhi
    c. Optimasi untuk mencapai goal state atau mendekati goal state
    d. Balance antara eksplorasi dan eksploitasi dalam pencarian solusi

Permasalahan ini akan diselesaikan menggunakan tiga algoritma local search, yaitu: Hill Climbing (salah satu varian), simulated annealing, genetic algorithm.

Masing-masing algoritma akan dijalankan dengan parameter yang berbeda untuk menganalisis efektivitas dan efisiensinya dalam mencari solusi optimal atau mendekati optimal untuk permasalahan Diagonal Magic Cube ini.

Bab 2
Pembahasan

2.1. Objective Function

Objective function menghitung kualitas dari kandidat solusi dengan memperhitungkan seberapa dekat untuk menjadi *magic cube* yang sempurna. *Magic cube* mengharuskan untuk jumlah dari setiap baris, kolom, tiang, diagonal ruang, serta diagonal pada setiap potongan bidangnya untuk sama dengan nilai dari konstanta *magic*.

Berdasarkan hal ini, *objective function* dapat didefinisikan sebagai jumlah kesalahan atau selisih antara jumlah setiap garis (baris, kolom, tiang, diagonal ruang, dan diagonal pada setiap potongan bidang) dengan konstanta *magic*.

2.1.1. Magic Number

Magic number disimbolkan dengan variabel C. Pada *magic cube* $n \times n \times n$, formula yang digunakan untuk mencari *magic number* adalah sebagai berikut:

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

Hal ini dibuktikan dengan:

1. Menghitung total penjumlahan pada setiap angka yang terdapat pada *magic cube*. Angka yang terdapat pada *magic cube* dengan sisi n akan memiliki nilai yang berada pada *range* 1 hingga n^3, sehingga untuk menemukan total jumlah untuk setiap angka dapat menggunakan aritmatika dengan formula:

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

2. Syarat pada *magic cube* yang mana bahwa setiap baris, kolom, tiang harus memiliki total jumlah yang sama dengan *magic number*, sehingga untuk setiap masing-masing dari baris, kolom, dan tiang terdapat sebanyak n^2. Oleh karena itu, nilai dari Sn dibagi dengan banyaknya baris/kolom/tiang n^2 untuk mendapatkan distribusi yang merata untuk setiap garisnya. Maka didapat formula akhir *magic number*:

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

3. Dengan formula yang sudah ada, kita bisa mendapatkan nilai dari *magic number* dengan nilai 315, yang didapat dari $n = 5$:

$$C = \frac{5(1+5^3)}{2} = 315$$

2.2.2. Objective Function

Objective function yang digunakan untuk mencari solusi dari *magic cube* adalah dengan menghitung jumlah *line* pada *magic cube* yang memiliki total penjumlahan yang sama dengan konstanta *magic* yang sudah didapat pada bagian 1.1.

Jumlah *line* dihitung dari total syarat-syarat yang harus dimiliki dari *magic cube*. Berikut adalah perhitungan banyaknya *line* pada setiap *magic cube*:

1. Setiap baris *magic cube* memiliki jumlah sebanyak $n \times n$, dengan n sebagai sisi dari *magic cube.* Dengan demikian, jumlah dari baris pada *magic cube* $5 \times 5 \times 5$ adalah 25.
2. Setiap kolom *magic cube* memiliki jumlah sebanyak $n \times n$, dengan n sebagai sisi dari *magic cube.* Dengan demikian, jumlah dari kolom pada *magic cube* $5 \times 5 \times 5$ adalah 25.
3. Setiap tiang *magic cube* memiliki jumlah sebanyak $n \times n$, dengan n sebagai sisi dari *magic cube.* Dengan demikian, jumlah dari tiang pada *magic cube* $5 \times 5 \times 5$ adalah 25.
4. Setiap *magic cube* mempunyai diagonal ruang sebanyak 4.
5. Setiap *magic cube* mempunyai diagonal bidang, dimana terdapat 15 bidang dari *magic cube* $5 \times 5 \times 5$. Dimana pada setiap bidangnya masing-masing memiliki 2 diagonal. Dengan demikian, jumlah dari diagonal bidang yang dimiliki *magic cube* $5 \times 5 \times 5$ sebanyak 30.

Berdasarkan perhitungan diatas, maka total dari *line* yang terdapat pada *magic cube* $5 \times 5 \times 5$ adalah sebanyak $25 + 25 + 25 + 4 + 30 = 109$. Dengan demikian, untuk mencapai *magic cube* $5 \times 5 \times 5$ yang lebih baik, diperlukan *value successor* yang mendekati atau sama dengan 109.

2.2. Implementasi Magic Cube

2.2.1. MagicCube Class

Class MagicCube digunakan untuk merepresentasikan *magic cube* $5 \times 5 \times 5$ dengan tujuan untuk mendapatkan *value* terbaik terhadap *magic number* yang berarti *magic cube* yang lebih baik. *Magic number* adalah nilai yang seharusnya menjadi total jumlah untuk setiap elemen pada setiap baris, kolom, diagonal ruang, dan diagonal pada potongan bidang di dalam kubus. *Class* ini juga menyediakan berbagai fungsi untuk menginisialisasi *magic cube* dengan *random*, mencari *value* dari *magic cube*, serta menghasilkan tetangga (*successor*) dari konfigurasi saat ini.

2.2.2.1. MagicCube

Konstruktor *MagicCube* digunakan untuk menginisialisasi sebuah *magic cube* baru dengan ukuran $5 \times 5 \times 5$. Di dalam konstruktor, *magic cube* diinisialisasi dengan *magic cube random* dengan menggunakan fungsi *create_random_cube*. Kemudian, *magic number* yang memiliki nilai 315 dan *value* yang dihitung dengan menggunakan fungsi *calculate_value.*

```
class MagicCube(object):
    def __init__(self, cube=None):
        if cube is None:
            self.size = 5
            self.cube = self.create_random_cube()
        else:
            self.size = 5
            self.cube = cube
        self.magic_number = 315
        self.value = self.calculate_value()
```

2.2.2.2. create_random_cube

Fungsi *create_random_cube* digunakan untuk menghasilkan *magic cube* baru dengan elemen-elemen yang diisi secara *random*, memastikan bahwa setiap elemen adalah unik. Fungsi ini akan mengembalikan *random magic cube* berukuran $5 \times 5 \times 5$.

```
def create_random_cube(self):
        numbers = list(range(1, 126))  # 1 to 125 for 5x5x5 cube
        random.shuffle(numbers)

        cube = []
        index = 0
        for i in range(self.size):
            layer = []
            for j in range(self.size):
                row = []
                for k in range(self.size):
                    row.append(numbers[index])
                    index += 1
                layer.append(row)
            cube.append(layer)
        return cube
```

2.2.2.3. calculate_value

Fungsi *calculate_value* digunakan untuk menghitung nilai *value* dari *magic cube*. Fungsi ini akan membandingkan total penjumlahan pada setiap *line* nya dan membandingkannya dengan *magic number* yang sudah ada. Apabila sama, maka nilai *value* dari *magic cube* akan bertambah satu. Nilai *value* memiliki nilai maksimal yaitu 109 yang didapat dari *objective function* yang sudah didefinisikan sebelumnya.

```
def calculate_value(self, cube=None):
        if cube is None:
            cube = self.cube
        value = 0

        # Check rows (75 lines)
        for i in range(self.size):
            for j in range(self.size):
                # rows in xy-plane
                if sum(cube[i][j]) == self.magic_number:
                    value += 1
                # rows in xz-plane
                if sum(cube[i][k][j] for k in range(self.size)) == self.magic_number:
                    value += 1
                # rows in yz-plane
                if sum(cube[k][i][j] for k in range(self.size)) == self.magic_number:
                    value += 1

        # Check diagonals in each 2D plane (15 planes * 2 diagonals = 30 lines)
        for i in range(self.size):
            # diagonals in xy-plane
            if sum(cube[i][j][j] for j in range(self.size)) == self.magic_number:
                value += 1
            if sum(cube[i][j][self.size-1-j] for j in range(self.size)) == self.magic_number:
                value += 1

            # diagonals in xz-plane
            if sum(cube[i][j][j] for j in range(self.size)) == self.magic_number:
                value += 1
            if sum(cube[i][self.size-1-j][j] for j in range(self.size)) == self.magic_number:
                value += 1

            # diagonals in yz-plane
            if sum(cube[j][i][j] for j in range(self.size)) == self.magic_number:
                value += 1
            if sum(cube[j][i][self.size-1-j] for j in range(self.size)) == self.magic_number:
```

```
            value += 1

        # Check space diagonals (4 lines)
        # Main space diagonal
        if sum(cube[i][i][i] for i in range(self.size)) == self.magic_number:
            value += 1
        # Other space diagonals
        if sum(cube[i][i][self.size-1-i] for i in range(self.size)) == self.magic_number:
            value += 1
        if sum(cube[i][self.size-1-i][i] for i in range(self.size)) == self.magic_number:
            value += 1
                    if sum(cube[i][self.size-1-i][self.size-1-i] for i in range(self.size)) ==
self.magic_number:
            value += 1

        return value
```

2.2.2.4. copy_cube

Fungsi *copy_cube* digunakan untuk menduplikasi *magic cube* ke dalam *magic cube* lain. Fungsi ini akan mengembalikan *magic cube* baru yang dapat membantu dalam pencarian *successor* pada fungsi *get_successor*.

```
def copy_cube(self, cube):
        return [[[cube[i][j][k] for k in range(self.size)]
                for j in range(self.size)]
                for i in range(self.size)]
```

2.2.2.5. swap_positions

Fungsi *swap_positions* digunakan untuk melakukan *swap* sebanyak 2 angka secara acak pada *magic cube*. Fungsi ini akan mengembalikan *magic cube* baru dari *magic cube* yang sudah diacak sebelumnya.

```
def swap_positions(self, cube, pos1, pos2):
        new_cube = self.copy_cube(cube)
        i1, j1, k1 = pos1
        i2, j2, k2 = pos2
                        new_cube[i1][j1][k1], new_cube[i2][j2][k2] = new_cube[i2][j2][k2],
new_cube[i1][j1][k1]
        return new_cube
```

2.2.2.6. get_successor

Fungsi *get_successor* digunakan untuk mendapatkan *random* dan *best successor* dari *current state magic cube*. Fungsi ini menghasilkan *successor* dengan memenggil *swap_positions* pada *current state magic cube*. Fungsi ini akan mengembalikan objek *successor magic cube* baru untuk menjadi *neighbor* pada algoritma *local search*.

```
def get_successor(self, mode="random"):
        if mode == "random":
            # Get two random positions
            pos1 = (random.randint(0, 4), random.randint(0, 4), random.randint(0, 4))
            pos2 = (random.randint(0, 4), random.randint(0, 4), random.randint(0, 4))
            while pos1 == pos2:
                pos2 = (random.randint(0, 4), random.randint(0, 4), random.randint(0, 4))

            new_cube = self.swap_positions(self.cube, pos1, pos2)
            return MagicCube(new_cube)

        elif mode == "best":
            current_value = self.value  # Use the stored value
            best_value = current_value
            best_cube = self.copy_cube(self.cube)

            for i1 in range(self.size):
                for j1 in range(self.size):
                    for k1 in range(self.size):
                        for i2 in range(self.size):
                            for j2 in range(self.size):
                                for k2 in range(self.size):
                                    # Skip if same position
                                    if (i1, j1, k1) >= (i2, j2, k2):
                                        continue

                                    # swap position
                                    new_cube = self.swap_positions(self.cube, (i1, j1, k1), (i2, j2, k2))
                                    value = self.calculate_value(new_cube)

                                    if value > best_value:
                                        best_value = value
                                        best_cube = new_cube

            return MagicCube(best_cube)
```

2.2.2.7. print_cube

Fungsi *print_cube* digunakan untuk menampilkan *state* dari *magic cube*. Fungsi ini akan menampilkan isi dari *magic cube* dan *value* dari *magic cube*.

```
def print_cube(self):
        print("\nCurrent Cube State:")
        for i in range(self.size):
            print(f"\nLayer {i+1}:")
            for row in self.cube[i]:
                print([f"{x:3d}" for x in row])
        print(f"\nCurrent Value: {self.value}")
        if self.value == 109:
            print("Congratulations! Magic cube solved!")
        else:
            print("Magic cube not yet solved.")
```

2.3. Implementasi Algoritma Local Search

Berikut merupakan implementasi dari algoritma *local search* yang ada.

2.3.1. Algoritma Hill-Climbing

Berikut merupakan implementasi dari algoritma *local search* Hill-Climbing yang ada.

2.3.1.1. Steepest-Ascent Hill-Climbing

Steepest-Ascent Hill-Climbing adalah salah satu varian dari algoritma *local search* yang mencari solusi terbaik secara bertahap dengan membandingkan semua tetangga dari keadaan saat ini dan memilih tetangga yang memiliki nilai evaluasi tertinggi (yang berarti memperbaiki solusi). Dalam hal ini, algoritma akan mengidentifikasi pergerakan yang memiliki keuntungan paling besar terhadap fungsi tujuan, atau dengan kata lain, memilih "tanjakan" yang paling curam untuk mendaki ke keadaan yang lebih baik. Metode ini menghindari pergerakan acak karena hanya mempertimbangkan langkah-langkah yang memperbaiki nilai secara optimal dari keadaan saat ini.

Keunggulan dari Steepest-Ascent Hill-Climbing adalah dapat dengan cepat mencapai solusi yang lebih baik, tetapi algoritma ini juga berisiko terjebak dalam *local optimum, plateau, atau ridges*, sehingga sulit mencapai *global optimum* tanpa adanya mekanisme tambahan untuk menghindari jebakan tersebut.

a. Konstruktor

```
class steepest_ascent(object):
    def __init__(self):
        self.list_of_value = []
```

```
        self.iteration = 0
        self.duration = 0
```

b. Algoritma Utama

```
def run(self):
        start_time = time.time()

        current = MagicCube()
        self.list_of_value.append(current.value)
        current.print_cube()
        i = 0

        while True:
            successor = current.get_successor("best")
            if successor.value <= current.value:
                break
            else:
                current = successor
            self.list_of_value.append(current.value)
            i += 1

        current.print_cube()
        self.duration = time.time() - start_time
        self.iteration = i
        print(self.duration)
        print(self.iteration)
        self.makePlot()
```

c. Plotting

```
def makePlot(self):
        plt.figure(figsize=(12, 8))

        plt.subplot(2, 1, 1)
        plt.plot(list(range(len(self.list_of_value))), self.list_of_value)
        plt.title("Magic Cube Value over Iterations")
        plt.xlabel("Iteration")
        plt.ylabel("Value")
        plt.grid(True)

        plt.subplot(2, 1, 2)
        plt.axis('off')
```

```
        info_text = (
            f"Initial Value: {self.list_of_value[0]}\n"
            f"Final Value: {self.list_of_value[-1]}\n"
            f"Total Iterations: {self.iteration}\n"
            f"Duration: {self.duration:.2f} seconds"
        )
        plt.text(0.1, 0.5, info_text, fontsize=12, verticalalignment='center')

        plt.tight_layout()
        plt.show()
```

2.3.1.2. Sideways-Move Hill-Climbing

Sideways-Move Hill-Climbing adalah variasi dari Hill-Climbing yang memungkinkan algoritma melakukan pergerakan ke tetangga yang memiliki nilai evaluasi yang sama dengan keadaan saat ini (disebut *sideways move*). Fitur ini dirancang untuk mengatasi masalah plateau atau area yang datar di ruang pencarian, di mana semua tetangga memiliki nilai evaluasi yang sama dan algoritma tidak dapat menemukan arah perbaikan.

Algoritma ini diatur dengan batas tertentu pada jumlah *sideways moves* yang dapat dilakukan secara berurutan untuk mencegah algoritma terus bergerak di sekitar area yang datar tanpa ada kemajuan. Jika batas ini tercapai tanpa adanya peningkatan, algoritma akan berhenti atau melakukan reset.

a. Konstruktor

```
class sideways_move(object):
    def __init__(self, max_sideways_moves = 100):
        self.list_of_value = []
        self.max_sideways_moves = max_sideways_moves
        self.iteration = 0
        self.duration = 0
        self.total_sideways = 0
```

b. Algoritma Utama

```
def run(self):
        start_time = time.time()

        current = MagicCube()
        self.list_of_value.append(current.value)
        current.print_cube()
        i = 0
        sideways_moves = 0
```

```
        while True and sideways_moves < self.max_sideways_moves:
            successor = current.get_successor("best")

            if successor.value == current.value:
                sideways_moves += 1
            elif successor.value < current.value:
                break
            else:
                sideways_moves = 0

            self.list_of_value.append(successor.value)
            current = successor
            i += 1

        current.print_cube()
        self.duration = time.time() - start_time
        self.iteration = i
        self.total_sideways = sideways_moves
        print(self.duration)
        print(self.iteration)
        print(f"Total sideways moves: {sideways_moves}")
        self.makePlot()
```

c. Plotting

```
def makePlot(self):
        plt.figure(figsize=(12, 8))

        plt.subplot(2, 1, 1)
        plt.plot(list(range(len(self.list_of_value))), self.list_of_value)
        plt.title("Magic Cube Value over Iterations")
        plt.xlabel("Iteration")
        plt.ylabel("Value")
        plt.grid(True)

        plt.subplot(2, 1, 2)
        plt.axis('off')
        info_text = (
            f"Initial Value: {self.list_of_value[0]}\n"
            f"Final Value: {self.list_of_value[-1]}\n"
            f"Total Iterations: {self.iteration}\n"
            f"Total sideways: {self.total_sideways}\n"
            f"Duration: {self.duration:.2f} seconds"
```

```
    )
    plt.text(0.1, 0.5, info_text, fontsize=12, verticalalignment='center')

    plt.tight_layout()
    plt.show()
```

2.3.1.3. Stochastic Hill-Climbing

Stochastic Hill-Climbing adalah variasi yang memilih tetangga secara acak dan tidak selalu memilih tetangga dengan nilai evaluasi tertinggi. Algoritma ini memilih pergerakan berdasarkan probabilitas yang ditentukan, di mana tetangga dengan evaluasi lebih baik memiliki peluang lebih tinggi untuk dipilih, tetapi tetap ada kemungkinan tetangga yang kurang optimal juga dipilih.

Pendekatan ini dapat membantu algoritma menjelajahi ruang pencarian yang lebih luas dan menghindari *local optimum*, karena tidak selalu memilih langkah terbaik yang tersedia. Metode ini cocok untuk masalah di mana ruang pencarian sangat besar dan keberagaman solusi penting untuk menghindari terjebak pada solusi lokal.

a. Konstruktor

```
class stochastic(object):
    def __init__(self, max_iterations=100000):
        self.max_iterations = max_iterations
        self.list_of_value = []
        self.iteration = 0
        self.duration = 0
```

b. Algoritma Utama

```
def run(self):
        start_time = time.time()

        current = MagicCube()
        self.list_of_value.append(current.value)
        current.print_cube()
        it = 0

        for i in range(self.max_iterations):
            successor = current.get_successor("random")
            if successor.value > current.value:
                current = successor

            self.list_of_value.append(current.value)
            it += 1
```

```
        current.print_cube()
        self.duration = time.time() - start_time
        self.iteration = i + 1
        print(self.duration)
        print(self.iteration)
        self.makePlot()
```

c. Plotting

```
def makePlot(self):
        plt.figure(figsize=(12, 8))

        plt.subplot(2, 1, 1)
        plt.plot(list(range(len(self.list_of_value))), self.list_of_value)
        plt.title("Magic Cube Value over Iterations")
        plt.xlabel("Iteration")
        plt.ylabel("Value")
        plt.grid(True)

        plt.subplot(2, 1, 2)
        plt.axis('off')
        info_text = (
            f"Initial Value: {self.list_of_value[0]}\n"
            f"Final Value: {self.list_of_value[-1]}\n"
            f"Total Iterations: {self.iteration}\n"
            f"Duration: {self.duration:.2f} seconds"
        )
        plt.text(0.1, 0.5, info_text, fontsize=12, verticalalignment='center')

        plt.tight_layout()
        plt.show()
```

2.3.1.4. Random Restart Hill-Climbing

Random-Restart Hill-Climbing adalah variasi dari algoritma hill-climbing yang berupaya mengatasi keterjebakan pada local optimum dengan melakukan beberapa kali restart dari titik awal yang berbeda. Dalam metode ini, ketika algoritma menemui solusi lokal yang tidak optimal, proses pencarian akan dihentikan, dan pencarian baru akan dimulai dari titik acak yang berbeda di ruang pencarian.

Pendekatan ini meningkatkan kemungkinan untuk menemukan solusi global optimal dengan mengulangi proses pencarian dari posisi yang berbeda, sehingga lebih banyak area dari ruang pencarian yang dapat dijelajahi. Metode ini sangat cocok untuk masalah dengan

ruang pencarian yang luas dan kompleks, di mana terdapat banyak local optimum yang dapat memerangkap algoritma dalam solusi suboptimal.

a. Konstruktor

```
def __init__(self, max_restarts: int = 10):
    self.list_of_value: List[int] = []
    self.max_restarts = max_restarts
    self.num_restarts = 0
    self.total_iterations = 0
    self.start_time = 0
    self.end_time = 0
```

b. Algoritma Utama

```
def run(self) -> None:
    self.start_time = time.time()

    best_cube = MagicCube()
    best_value = best_cube.value

    print("Initial state:")
    best_cube.print_cube()
    print(f"Initial value: {best_value}\n")

    while self.num_restarts < self.max_restarts:
        current = MagicCube()  # Start from a new random state
        current, iterations = self.hill_climbing(current)
        self.total_iterations += iterations

        if current.value > best_value:
            best_cube = current
            best_value = current.value
        self.num_restarts += 1
        print(f"Restart {self.num_restarts} - Current best value: {best_value}")

        if best_value == 109:  # Found optimal solution
            break

    self.end_time = time.time()
```

```
        print("\nFinal state:")
        best_cube.print_cube()
        print("\nResults:")
        print(f"Number of restarts: {self.num_restarts}")
        print(f"Total iterations: {self.total_iterations}")
        print(f"Best value found: {best_value}")
        print(f"Time taken: {self.end_time - self.start_time:.2f} seconds")

        self.make_plot()
```

c. Plotting

```
def make_plot(self) -> None:
        plt.figure(figsize=(12, 6))
        plt.plot(list(range(len(self.list_of_value))), self.list_of_value)
        plt.title("Magic Cube Value over Iterations (Random Restart Hill Climbing)")
        plt.xlabel("Iteration")
        plt.ylabel("Value")
        plt.grid()
        plt.show()
```

2.3.2. Simulated Annealing

Simulated Annealing adalah algoritma local search yang terinspirasi dari proses annealing dalam metalurgi, di mana logam dipanaskan dan didinginkan secara perlahan untuk mencapai struktur kristal yang optimal.

a. Parameter Utama

```
def __init__(self, initial_temp=1000000.0, cooling_rate=0.99995, min_temp=0.0001,
max_iterations=500000):
    self.initial_temp = initial_temp
    self.cooling_rate = cooling_rate
    self.min_temp = min_temp
    self.max_iterations = max_iterations
    self.stuck_threshold = 175000
```

b. Probabilitas Penerimaan

```
def accept_probability(self, current_value, neighbor_value, temperature):
```

```
    if neighbor_value >= current_value:
        return 1.0

    delta_E = neighbor_value - current_value
    if temperature > self.initial_temp * 0.8:
        prob = math.exp(delta_E / (temperature * 0.01))
    elif temperature > self.initial_temp * 0.4:
        prob = math.exp(delta_E / (temperature * 0.05))
    else:
        prob = math.exp(delta_E / temperature)
```

c. Algoritma Utama

```
def run(self, magic_cube):
    start_time = time.time()
    current = MagicCube(magic_cube.cube)
    best = MagicCube(current.cube)
    temperature = self.initial_temp

    while temperature > self.min_temp and total_iterations < self.max_iterations:
        for _ in range(300):
            best_neighbor = get_best_neighbor(current, 10)
            if accept_probability(current.value, best_neighbor.value, temperature):
                current = best_neighbor
                if current.value > best.value:
                    best = MagicCube(current.cube)

        if stuck:
            if best.value < 40:
                temperature = self.initial_temp * 0.95
            elif best.value < 60:
                temperature = self.initial_temp * 0.8
            elif best.value < 80:
                temperature = self.initial_temp * 0.6

        if in_plateau:
            temperature *= (cooling_rate ** 2)
        elif best.value > 80:
            temperature *= (cooling_rate ** 0.1)
        else:
            temperature *= cooling_rate
```

d. Strategi Optimasi

- Eksplorasi vs Eksploitasi:
    1) Eksplorasi tinggi di temperature tinggi
    2) Eksploitasi meningkat seiring penurunan temperature
    3) Penerimaan solusi buruk menurun secara gradual
- Adaptive Cooling

```
if best.value > 80:
    temperature *= (cooling_rate ** 0.1)
elif best.value > 60:
    temperature *= (cooling_rate ** 0.25)
elif best.value > 40:
    temperature *= (cooling_rate ** 0.5)
```

- Reheat Strategy

```
if best.value < 40:
    temperature = initial_temp * 0.95
elif best.value < 60:
    temperature = initial_temp * 0.8
elif best.value < 80:
    temperature = initial_temp * 0.6
```

e. Tracking dan Visualisasi

```
self.objective_values = []
self.temperatures = []
self.exp_deltaE_T = []
self.stuck_count = 0
```

- Visualisasi:
    1) Plot objective function vs iterations
    2) Plot temperature vs iterations
    3) Plot e^(ΔE/T) vs iterations
    4) Summary statistics dan informasi eksperimen
- Parameter Optimal yang Ditemukan:
    1) Initial Temperature: 1,000,000.0
    2) Cooling Rate: 0.99995
    3) Minimum Temperature: 0.0001
    4) Maximum Iterations: 500,000
    5) Stuck Threshold: 175,000

Dengan implementasi ini, algoritma berhasil mencapai nilai objektif antara 30-35 dengan konsistensi yang baik dan waktu komputasi yang reasonable (sekitar 5-7 menit per eksperimen).

2.3.3. Genetic Algorithm

Genetic Algorithm adalah algoritma yang melakukan pemilihan beberapa individu sehingga tercipta populasi yang kemudian tiap populasi akan dilakukan proses penentuan nilai *fitness* tiap individu di populasi, dilanjutkan dengan memperlakukan seleksi untuk memilih induk yang digunakan untuk dilakukan *crossover*, yang kemudian dilanjutkan dengan dilakukannya mutasi pada beberapa individu yang terpilih. Berikut merupakan penjelasan algoritma *genetic algorithm* yang telah berhasil dibuat menggunakan bahasa pemrograman python.

a. Konstruktor

```
def __init__(self, population_size=100, iterations=100):
    self.population_size = population_size
    self.mutation_rate = 0.1
    self.iterations = iterations
    self.avg_fitness_history = []
    self.best_fitness_history = []
    self.execution_time = None
    self.initial_fitness = None
    self.final_fitness = None
```

b. Algoritma Utama

```
def run(self, init_state: MagicCube):
    start_time = time.time()
    self.initial_fitness = init_state.value

    population = [init_state]
    for _ in range(self.population_size - 1):
        population.append(MagicCube())

    best_fitness = init_state.value
    best_cube = init_state

    print(f"\nInitial cube (fitness: {best_fitness}/109):")
    init_state.print_cube()
    print("\nStarting optimization...\n")

    for generation in range(self.iterations):
        fitness = self.calculate_fitness(population)
        current_best = max(fitness)
        avg_fitness = sum(fitness) / len(fitness)
```

```
        self.best_fitness_history.append(current_best)
        self.avg_fitness_history.append(avg_fitness)

        if current_best > best_fitness:
            best_fitness = current_best
            best_cube = population[fitness.index(current_best)]

        if best_fitness == 109:
            print(f"\nSolution found at generation {generation + 1}")
            break

        population = self.selection(population, fitness)
        population = self.crossover(population)
        population = self.mutation(population)

    self.execution_time = time.time() - start_time
    self.final_fitness = best_fitness

    print(f"\nExecution time: {self.execution_time:.2f} seconds")
    print(f"Final best fitness: {best_fitness}/109")
    print("\nFinal best cube structure:")
    best_cube.print_cube()

    self.plot_progress()

    return best_cube, best_fitness
```

c. Fungsi *Fitness*

```
def calculate_fitness(self, population: List[MagicCube]) -> List[float]:
    return [cube.value for cube in population]
```

Penerapan *fitness* pada algoritma ini adalah menggunakan value masing-masing individu pada tiap populasi yang kemudian hasilnya akan digunakan pada fungsi seleksi.

d. Fungsi Seleksi

```
def selection(self, population: List[MagicCube], fitness: List[float]) -> List[MagicCube]:
    population_with_fitness = list(zip(fitness, population))
    population_with_fitness.sort(key=lambda x: x[0], reverse=True)
    sorted_population = [cube for _, cube in population_with_fitness]
    elite_size = max(2, self.population_size // 4)
    elite = sorted_population[:elite_size]
    new_population = []
    new_population.extend(elite)
```

```
    while len(new_population) < self.population_size:
        chosen = np.random.choice(elite)
        new_population.append(MagicCube(chosen.copy_cube(chosen.cube)))

    return new_population
```

Fungsi seleksi ini melakukan seleksi dengan cara sebagai berikut.

1. Mengurutkan populasi berdasarkan nilai *fitness*.
2. Memilih 25% individu terbaik, dengan seminimal-minimalnya dua individu. Individu terbaik ini sudah pasti akan masuk ke tahap *crossover*.
3. Sisa populasi dilengkapi dengan salinan yang secara acak dipilih dari individu terbaik yang telah dipilih pada poin 2.

e. Fungsi *Crossover*

```
def crossover(self, population: List[MagicCube]) -> List[MagicCube]:
    children = []
    elite_size = max(2, self.population_size // 10)
    children.extend(population[:elite_size])

    while len(children) < self.population_size:
        parent1, parent2 = np.random.choice(population[:elite_size], 2, replace=False)
        child_cube = parent1.copy_cube(parent1.cube)
        layers = np.random.choice(5, np.random.randint(1, 4), replace=False)

        for layer in layers:
            child_cube[layer] = parent2.copy_cube(parent2.cube)[layer]
        children.append(MagicCube(child_cube))

    return children
```

Fungsi *crossover* ini melakukan *crossover* dengan cara sebagai berikut.

1. Dilakukan penentuan individu elite dengan total maksimal 10% dari total populasi atau minimal dua individu
2. Pengisian sisa yaitu menerapkan pemilihan dua *parent* secara acak dari individu elit pada poin 1.
3. Salah satu *parent* menjadi basisnya yang kemudian dipilih 1 hingga 3 layer dari 5 layer yang akan diganti menggunakan layer yang sama namun dari *parent* lainnya.
4. Proses berulang hingga tercapainya jumlah populasi.

f. Fungsi mutasi

```
def mutation(self, population: List[MagicCube]) -> List[MagicCube]:
    mutated = []
    for cube in population:
        if np.random.random() < self.mutation_rate:
            mutated_cube = cube

            for _ in range(np.random.randint(1, 4)):
                pos1 = (np.random.randint(0, 5), np.random.randint(0, 5), np.random.randint(0, 5))
                pos2 = (np.random.randint(0, 5), np.random.randint(0, 5), np.random.randint(0, 5))

                while pos1 == pos2:
                    pos2 = (np.random.randint(0, 5), np.random.randint(0, 5), np.random.randint(0, 5))
                new_cube = mutated_cube.swap_positions(mutated_cube.cube, pos1, pos2)
                mutated_cube = MagicCube(new_cube)

            mutated.append(mutated_cube)
        else:
            mutated.append(cube)
    return mutated
```

Seperti umumnya mutasi, tiap individu pada populasi memiliki peluang untuk terjadi mutasi. Dalam hal ini, semisal suatu individu terpilih, maka individu tersebut akan mengalami 1 sampai 3 kali pertukaran posisi secara acak. Berikut penjelasan pertukaran posisi secara acak yang terjadi.

1. Dipilih dua posisi berbeda secara acak pada individu yang terpilih untuk mengalami mutasi
2. Dua posisi yang berbeda pada individu akan terjadi pertukaran nilai.
3. Individu yang tidak terpilih untuk mutasi akan tetap dipertahankan.

g. Fungsi *Plotting*

```
def plot_progress(self):
    fig = plt.figure(figsize=(15, 10))

    ax1 = plt.subplot(2, 1, 1)
    ax1.plot(self.best_fitness_history, label='Best Value', color='blue')
    ax1.plot(self.avg_fitness_history, label='Population Average', color='red')
    ax1.set_xlabel('Generation')
    ax1.set_ylabel('Fitness')
    ax1.set_title('Magic Cube Optimization Progress')
    ax1.legend()
    ax1.grid(True)

    info_text = (
        f'Optimization Info:\n'
```

```
        f'Population Size: {self.population_size}\n'
        f'Total Generations: {len(self.best_fitness_history)}\n'
        f'Duration: {self.execution_time:.2f} seconds\n'
        f'Initial Value: {self.initial_fitness}/109\n'
        f'Final Value: {self.final_fitness}/109'
    )

        plt.figtext(0.15, 0.15, info_text, bbox=dict(facecolor='white', alpha=0.8, edgecolor='gray'),
fontsize=10, family='monospace')

    ax2 = plt.subplot(2, 1, 2)
    ax2.axis('off')

    plt.tight_layout()
    plt.show()
```

2.4. Implementasi Algoritma Visualizer

Berikut merupakan implementasi dari algoritma *visualizer* untuk menayangkan *replay* dari proses pencarian secara detail per tahap/iterasinya.

2.4.1. Fungsi make_file

Fungsi ini digunakan untuk membuat *save file* dengan filename yang sudah ditentukan untuk setiap algoritma *local search* yang ada. Fungsi ini terdapat pada setiap algoritma *local search*. Ketika dipanggil, program akan mengembalikan nama *path save file* untuk algoritma *local search* yang sedang dijalankan.

```
def make_file(self, name):
        directory = ".\\save_file"
        os.makedirs(directory, exist_ok=True)

        counter = 1
        while True:
            filename = f"{name}{counter}.txt"
            filepath = os.path.join(directory, filename)

            if not os.path.exists(filepath):
                break
            counter += 1

        return filepath
```

2.4.2. Fungsi make_file

Fungsi ini digunakan untuk melakukan *save file* suatu *state* dari *magic cube*. Fungsi ini memiliki parameter *file path* yang merupakan *path* dari *save file* yang sudah dibuat sebelumnya. Fungsi ini berada pada *class MagicCube* untuk mengambil isi dari *magic cube* yang ada.

```
def save_state(self, filepath):
        with open(filepath, "a") as file:
            for row in range(self.size):
                for col in range(self.size):
                    for depth in range(self.size):
                        file.write(str(self.cube[row][col][depth])
+ " ")
                file.write("\n")
            file.write(f";\n")
```

2.4.3. Class visualizer

Class ini digunakan untuk melakukan visualisasi *replay* dari proses pencarian per iterasinya.

a. Konstruktor

Fungsi ini digunakan untuk menginisialisasi objek visualizer.

```
class Visualizer:
    def __init__(self):
        self.list_of_magiccube = []
        self.current_index = 0
        self.is_playing = False
        self.is_reverse = False
        self.speed = 1.0
        self.page = None
```

b. Load_file

Fungsi ini digunakan untuk melakukan *load save file* yang sudah disimpan sebelumnya. Isis dari *load file* akan dimasukkan kedalam list yang berisi *magic cube* dengan iterasi yang berurutan.

```
def load_file(self, filename):
        try:
            directory = ".\\save_file"
            filepath = os.path.join(directory, filename)
```

```
            with open(filepath, 'r') as file:
                content = file.read()
                states = content.strip().split(';')

                self.list_of_magiccube = []

                for state in states:
                    if state.strip():
                        cube = MagicCube()
                        rows = state.strip().split('\n')

                        for i, row in enumerate(rows):
                            numbers = [int(num) for num in
row.strip().split()]

                            for j in range(cube.size):
                                for k in range(cube.size):
                                    cube.cube[i][j][k] = numbers[j
* cube.size + k]


self.list_of_magiccube.append(MagicCube(cube.cube))

                self.current_index = 0
                self.is_playing = False
                self.is_reverse = False

                if self.page:

self.update_visualization(self.list_of_magiccube[0])
                    self.file_path_text.value = f"Loaded:
{filename}"
                    self.file_path_text.update()

                    self.progress_slider.max =
len(self.list_of_magiccube) - 1
                    self.progress_slider.value = 0
                    self.progress_slider.disabled = False
                    self.progress_slider.update()

                return True
```

```
        except FileNotFoundError:
            if self.page:
                self.show_error_dialog(f"File {filename} not
found")
            return False
        except Exception as e:
            if self.page:
                self.show_error_dialog(f"Error occurred:
{str(e)}")
            return False
```

c. Show_error_dialog

Fungsi ini digunakan untuk menampilkan *error* apabila terjadi

```
def show_error_dialog(self, message):
        def close_dialog(e):
            dialog.open = False
            self.page.update()

        dialog = ft.AlertDialog(
            modal=True,
            title=ft.Text("Error"),
            content=ft.Text(message),
            actions=[
                ft.TextButton("OK", on_click=close_dialog),
            ],
        )

        self.page.dialog = dialog
        dialog.open = True
        self.page.update()
```

d. Pick_files_result

Fungsi ini digunakan untuk meng-*update visualiser* ketika *load file* berhasil dibaca dan dijalankan.

```
def pick_files_result(self, e: ft.FilePickerResultEvent):
        if e.files:
            file_path = e.files[0].path
            filename = os.path.basename(file_path)
```

```
            success = self.load_file(filename)

            if success:
                self.play_button.disabled = False
                self.playback_button.disabled = False
                self.reset_button.disabled = False
                self.prev_button.disabled = self.current_index <=
0
                self.next_button.disabled = self.current_index >=
len(self.list_of_magiccube) - 1
                self.progress_slider.disabled = False

                self.play_button.update()
                self.playback_button.update()
                self.reset_button.update()
                self.prev_button.update()
                self.next_button.update()
                self.progress_slider.update()
```

e. Progress_changed

Fungsi ini digunakan untuk mengatur *slider* yang berfungsi sebaga progress bar.

```
def progress_changed(self, e):
        self.current_index = int(e.control.value)

self.update_visualization(self.list_of_magiccube[self.current_inde
x])
```

f. Create_face

Fungsi ini digunakan untuk membuat visualisasi layer/*face* dari *magic cube* yang sebelumnya disimpan.

```
def create_face(self, numbers, offset_x, offset_y, color,
cube_size=200):
        cell_size = cube_size / 5
        spacing = 2

        face = ft.Container(
            width=cube_size,
            height=cube_size,
            left=offset_x,
```

```
            top=offset_y,
            border=ft.border.all(2, ft.colors.BLACK),
            bgcolor=color,
            content=ft.Column(
                spacing=spacing,
                controls=[
                    ft.Row(
                        spacing=spacing,
                        controls=[
                            ft.Container(
                                width=cell_size - spacing,
                                height=cell_size - spacing,
                                border=ft.border.all(1,
ft.colors.BLACK),
                                content=ft.Text(
                                    str(numbers[i][j]),
                                    size=16,
                                    weight=ft.FontWeight.BOLD,
                                    text_align=ft.TextAlign.CENTER
                                ),
                                alignment=ft.alignment.center,
                                bgcolor=ft.colors.WHITE10,
                            ) for j in range(5)
                        ],
                    ) for i in range(5)
                ]
            ),
        )
        return face
```

g. Update_visualization

Fungsi ini digunakan untuk meng-*update* visualisasi setiap iterasi berganti atau terdapat masukan dari user.

```
def update_visualization(self, magic_cube):
        if not self.page:
            return

        # Layer state
        layers = [
            [[magic_cube.cube[k][i][j] for j in range(5)] for i in
```

```
range(5)]
            for k in range(5)
        ]

        # position
        cube_size = 200
        mid_x = cube_size

        # Create faces with different colors
        colors = [ft.colors.GREEN_50, ft.colors.ORANGE_50,
ft.colors.RED_50,
                  ft.colors.PURPLE_50, ft.colors.YELLOW_50]

        faces = [
            self.create_face(layers[i], mid_x * i, 0, colors[i])
            for i in range(5)
        ]

        # Update layer
        self.layer.controls = faces

        # Update iteration information
        self.iteration_information.value = f"Iteration:
{self.current_index + 1}/{len(self.list_of_magiccube)}"
        self.iteration_information.update()

        # Update value information
        self.value_information.value = f"Value:
{magic_cube.value}"
        self.value_information.update()

        # Update nav button
        self.prev_button.disabled = self.current_index <= 0
        self.next_button.disabled = self.current_index >=
len(self.list_of_magiccube) - 1
        self.prev_button.update()
        self.next_button.update()

        self.layer.update()
```

h. Play_sequence

Fungsi ini digunakan untuk menjalankan visualisasi atau *video player* sesuai dengan keinginan user dan dengan bantuan fungsi sebelumnya.

```
def play_sequence(self):
        while self.is_playing:
            time.sleep(self.speed)
            if self.is_reverse:
                if self.current_index > 0:
                    self.current_index -= 1
                    self.progress_slider.value =
self.current_index
                    self.progress_slider.update()

self.update_visualization(self.list_of_magiccube[self.current_inde
x])
                else:
                    self.is_playing = False
                    self.play_button.text = "Play Forward"
                    self.play_button.update()
            else:
                if self.current_index <
len(self.list_of_magiccube) - 1:
                    self.current_index += 1
                    self.progress_slider.value =
self.current_index
                    self.progress_slider.update()

self.update_visualization(self.list_of_magiccube[self.current_inde
x])
                else:
                    self.is_playing = False
                    self.play_button.text = "Play Forward"
                    self.play_button.update()
```

i. Play_button_clicked

Fungsi ini digunakan untuk mengatur navigasi *play* (animasi maju).

```
def play_button_clicked(self, e):
        self.is_playing = not self.is_playing
        self.is_reverse = False
        e.control.text = "Pause" if self.is_playing else "Play
```

```
Forward"
        e.control.update()

        if self.is_playing:
            threading.Thread(target=self.play_sequence,
daemon=True).start()
```

j. Playback_button_clicked

Fungsi ini digunakan untuk mengatur navigasi *playback* (animasi mundur).

```
def playback_button_clicked(self, e):
        self.is_playing = not self.is_playing
        self.is_reverse = True
        e.control.text = "Pause" if self.is_playing else "Play
Backward"
        e.control.update()

        if self.is_playing:
            threading.Thread(target=self.play_sequence,
daemon=True).start()
```

k. Next_button_clicked

Fungsi ini digunakan untuk mengatur navigasi *next* (iterasi maju sebanyak 1 kali).

```
def next_button_clicked(self, e):
        if self.current_index < len(self.list_of_magiccube) - 1:
            self.current_index += 1

self.update_visualization(self.list_of_magiccube[self.current_inde
x])
```

l. Prev_button_clicked

Fungsi ini digunakan untuk mengatur navigasi *previous* (iterasi mundur sebanyak 1 kali).

```
def prev_button_clicked(self, e):
        if self.current_index > 0:
            self.current_index -= 1

self.update_visualization(self.list_of_magiccube[self.current_inde
x])
```

m. Reset_button_clicked

Fungsi ini digunakan untuk mengatur navigasi *reset* (iterasi kembali ke 0).

```
def reset_button_clicked(self, e):
        self.is_playing = False
        self.is_reverse = False
        self.current_index = 0
        self.play_button.text = "Play Forward"
        self.playback_button.text = "Play Backward"
        self.play_button.update()
        self.playback_button.update()
        self.progress_slider.value = 0
        self.progress_slider.update()

self.update_visualization(self.list_of_magiccube[self.current_inde
x])
```

n. Speed_change

Fungsi ini digunakan untuk mengatur navigasi *playing speed*.

```
def speed_changed(self, e):
        self.speed = float(e.control.value)
        self.speed_information.value = f"Speed: {self.speed:.1f}s"
        self.speed_information.update()
```

o. Main

Fungsi ini digunakan sebagai fungsi utama untuk menghubungkan visualisasi-visualisasi yang ada.

```
def main(self, page: ft.Page):
        self.page = page
        page.title = "Magic Cube Sequence Visualization"
        page.padding = 50
        page.theme_mode = "light"

        self.pick_files_dialog = ft.FilePicker(
            on_result=self.pick_files_result
        )

        self.file_path_text = ft.Text(
```

```
            "No file loaded",
            size=14,
            color=ft.colors.GREY_700
        )

        load_file_button = ft.ElevatedButton(
            "Load File",
            icon=ft.icons.UPLOAD_FILE,
            on_click=lambda _: self.pick_files_dialog.pick_files(
                allow_multiple=False,
                allowed_extensions=["txt"]
            )
        )

        file_section = ft.Row(
            controls=[
                load_file_button,
                self.file_path_text
            ],
            alignment=ft.MainAxisAlignment.CENTER
        )

        page.overlay.append(self.pick_files_dialog)

        title = ft.Text(
            "Magic Cube Sequence Visualization",
            size=24,
            weight=ft.FontWeight.BOLD,
            text_align=ft.TextAlign.CENTER
        )

        # Info
        self.iteration_information = ft.Text(
            f"Iteration: 0/0",
            size=16,
            weight=ft.FontWeight.BOLD,
            text_align=ft.TextAlign.CENTER,
        )

        self.value_information = ft.Text(
            f"Value: 0",
            size=16,
```

```
            weight=ft.FontWeight.BOLD,
            text_align=ft.TextAlign.CENTER,
        )

        # Layer
        self.layer = ft.Stack(
            width=200 * 5,
            height=200,
        )

        # Progress slider
        self.progress_slider = ft.Slider(
            min=0,
            max=0,
            value=0,
            divisions=None,
            label="{value}",
            on_change=self.progress_changed,
            disabled=True
        )

        progress_container = ft.Container(
            content=self.progress_slider,
            padding=ft.padding.symmetric(horizontal=20),
            width=600
        )

        # Nav button
        self.prev_button = ft.IconButton(
            icon=ft.icons.ARROW_BACK,
            on_click=self.prev_button_clicked,
            disabled=True
        )

        self.next_button = ft.IconButton(
            icon=ft.icons.ARROW_FORWARD,
            on_click=self.next_button_clicked,
            disabled=True
        )

        self.play_button = ft.ElevatedButton(
            text="Play Forward",
```

```
            on_click=self.play_button_clicked,
            disabled=True
        )

        self.playback_button = ft.ElevatedButton(
            text="Play Backward",
            on_click=self.playback_button_clicked,
            disabled=True
        )

        self.reset_button = ft.ElevatedButton(
            text="Reset",
            on_click=self.reset_button_clicked,
            disabled=True
        )

        self.speed_information = ft.Text(f" Playing Speed:
{self.speed:.1f}s")
        speed_slider = ft.Slider(
            min=0.1,
            max=2.0,
            value=self.speed,
            divisions=19,
            label="{value}s",
            on_change=self.speed_changed
        )

        navigation = ft.Row(
            controls=[
                self.prev_button,
                self.next_button,
            ],
            alignment=ft.MainAxisAlignment.CENTER
        )

        controls = ft.Row(
            controls=[
                self.playback_button,
                self.play_button,
                self.reset_button,
            ],
            alignment=ft.MainAxisAlignment.CENTER
```

```
        )

        anim_speed = ft.Row(
            controls=[
                speed_slider,
                self.speed_information
            ],
            alignment=ft.MainAxisAlignment.CENTER
        )

        layer_information = ft.Row(
            controls=[
                ft.Text("Layer 1", size=16,
weight=ft.FontWeight.BOLD, color=ft.colors.GREEN),
                ft.Text("Layer 2", size=16,
weight=ft.FontWeight.BOLD, color=ft.colors.ORANGE),
                ft.Text("Layer 3", size=16,
weight=ft.FontWeight.BOLD, color=ft.colors.RED),
                ft.Text("Layer 4", size=16,
weight=ft.FontWeight.BOLD, color=ft.colors.PURPLE),
                ft.Text("Layer 5", size=16,
weight=ft.FontWeight.BOLD, color=ft.colors.YELLOW_600),
            ],
            alignment=ft.MainAxisAlignment.CENTER,
            spacing=20
        )

        content = ft.Column(
            controls=[
                title,
                file_section,
                self.iteration_information,
                self.value_information,
                self.layer,
                progress_container,
                navigation,
                controls,
                anim_speed,
                layer_information,
            ],
            horizontal_alignment=ft.CrossAxisAlignment.CENTER,
            spacing=20
```

```
        )

        page.add(content)
```

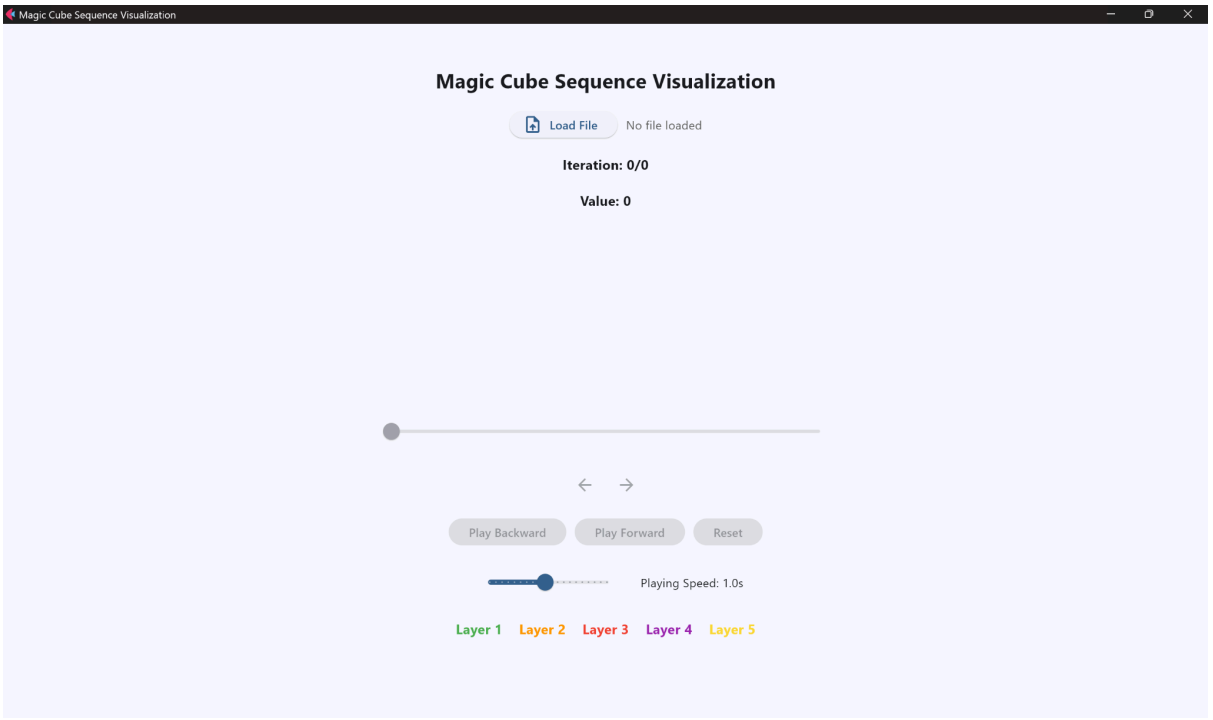
p. Visualize

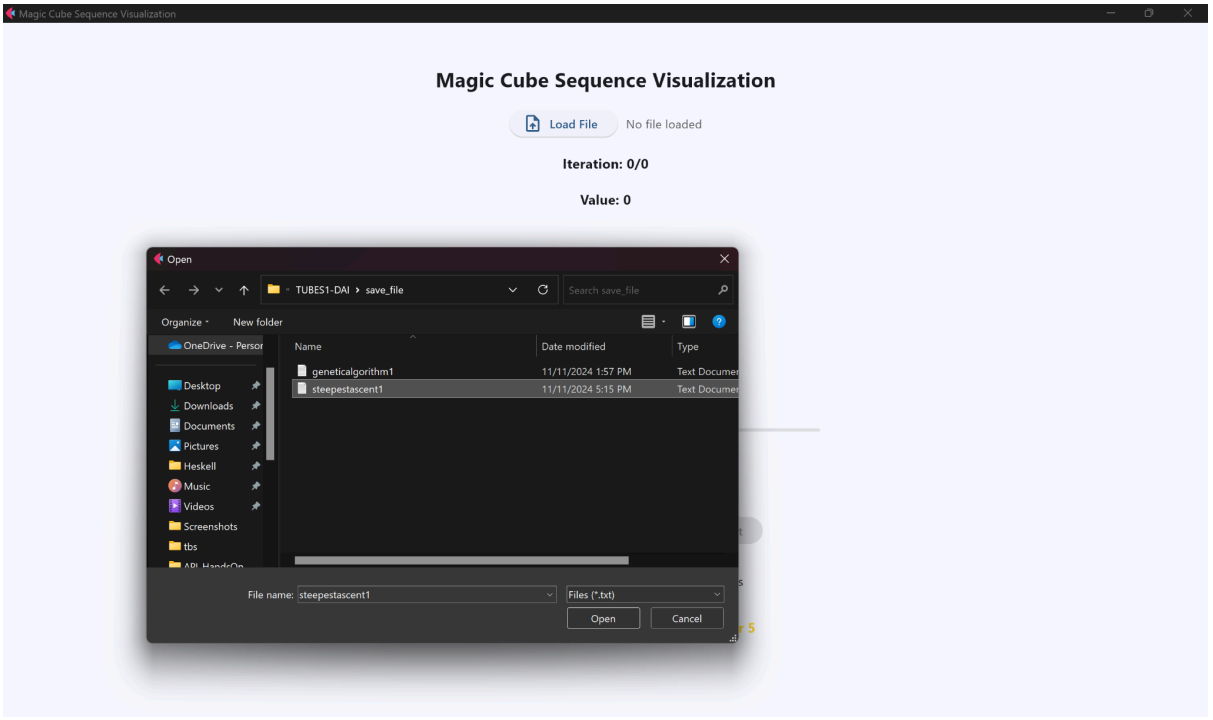
Fungsi ini digunakan untuk menjalankan fungsi main.

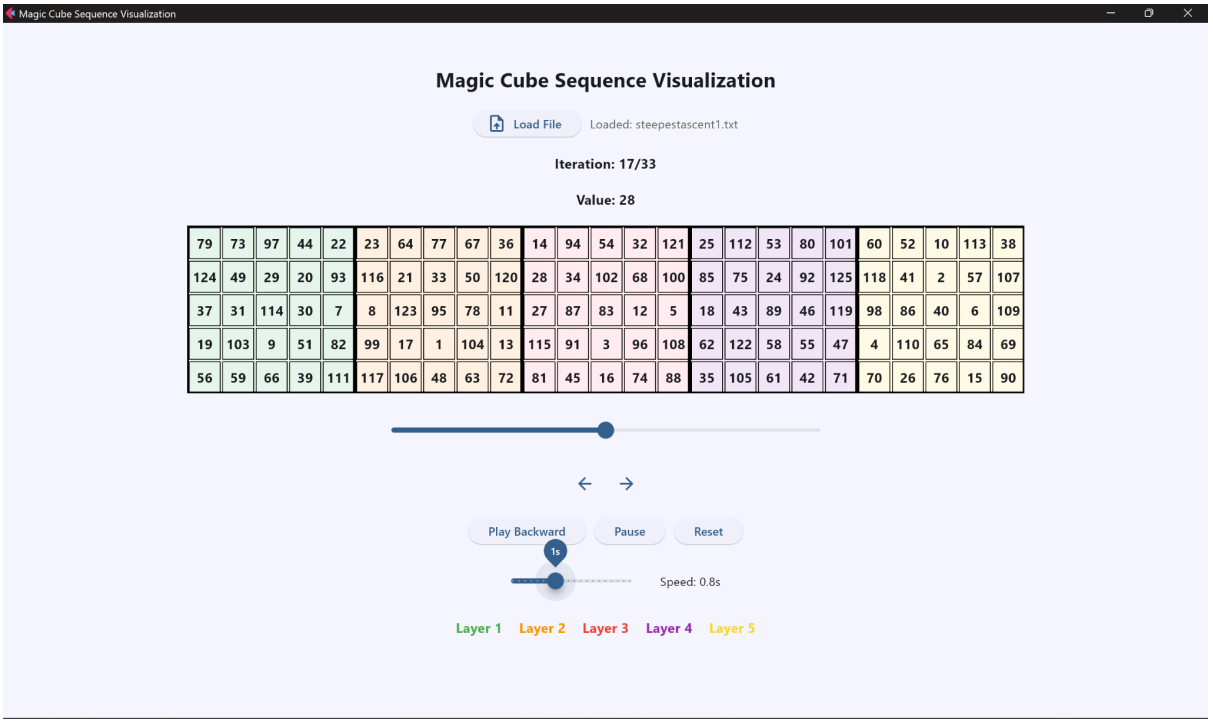
```
def visualize(self):
        ft.app(target=self.main)
```

2.4.3. Hasil output

Berikut merupakan hasil menayangkan *replay* dari proses pencarian secara detail per tahap/iterasinya.




Magic Cube Sequence Visualization
Magic Cube Sequence Visualization
Load File
No file loaded
Iteration: 0/0
Value: 0
Open
TUBES1-DAI › save_file
Search save_file
Organize
New folder
OneDrive - Perso
Name
Date modified
Type
geneticalgorithm1
11/11/2024 1:57 PM
steepestascent1
11/11/2024 5:15 PM
Desktop
Downloads
Documents
Pictures
Heskell
Music
Videos
Screenshots
tbs
File name: steepestascent1
Files (*.txt)
Open
Cancel


Magic Cube Sequence Visualization
Magic Cube Sequence Visualization
Load File
Loaded: steepestascent1.txt
Iteration: 17/33
Value: 28
79 73 97 44 22 23 64 77 67 36 14 94 54 32 121 25 112 53 80 101 60 52 10 113 38
124 49 29 20 93 116 21 33 50 120 28 34 102 68 100 85 75 24 92 125 118 41 2 57 107
37 31 114 30 7 8 123 95 78 11 27 87 83 12 5 18 43 89 46 119 98 86 40 6 109
19 103 9 51 82 99 17 1 104 13 115 91 3 96 108 62 122 58 55 47 4 110 65 84 69
56 59 66 39 111 117 106 48 63 72 81 45 16 74 88 35 105 61 42 71 70 26 76 15 90
Play Backward
Pause
Reset
1s
Speed: 0.8s
Layer 1
Layer 2
Layer 3
Layer 4
Layer 5

Bab 3
Hasil Eksperimen dan Analisis

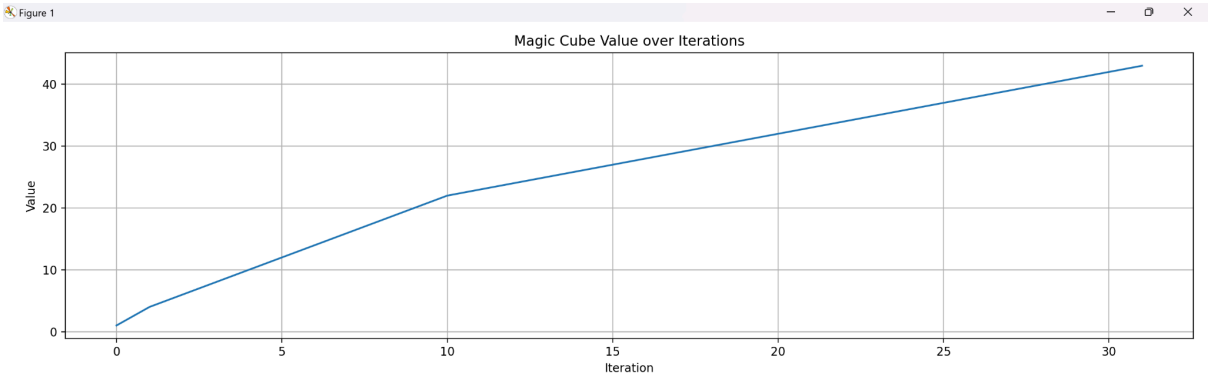
3.1. Algoritma Hill-Climbing

3.1.1. Steepest-Ascent Hill-Climbing

3.1.1.1. Eksperimen 1

```
Current Cube State:

Layer 1:
['102', ' 32', '115', '  6', '  2']
[' 13', ' 75', ' 69', '120', ' 30']
[' 25', ' 17', '103', ' 12', ' 55']
[' 26', ' 65', '  7', '  9', ' 93']
[' 22', ' 28', '100', ' 92', ' 40']

Layer 2:
[' 42', ' 24', '117', ' 82', ' 10']
[' 62', '107', '116', '101', '104']
['123', ' 63', ' 72', ' 15', ' 39']
[' 61', ' 21', '  8', ' 79', ' 76']
[' 31', ' 73', ' 95', ' 88', ' 52']

Layer 3:
[' 74', ' 86', ' 47', '121', ' 41']
['111', ' 11', ' 80', '110', ' 98']
[' 91', ' 14', ' 29', ' 53', '  3']
[' 43', ' 59', ' 94', '108', ' 64']
[' 68', '  1', ' 50', '  4', ' 20']

Layer 4:
[' 60', ' 18', ' 51', ' 33', '125']
['114', ' 45', ' 58', ' 27', ' 81']
[' 97', ' 99', '122', ' 71', ' 34']
[' 54', ' 84', ' 96', ' 70', ' 87']
[' 83', '118', ' 16', '119', '124']

Layer 5:
[' 36', ' 85', '113', ' 89', '109']
[' 49', ' 90', '112', ' 67', ' 23']
[' 38', ' 37', '105', ' 66', ' 48']
[' 57', ' 78', ' 56', ' 44', ' 35']
['  5', ' 77', ' 19', ' 46', '106']

Current Value: 1
Magic cube not yet solved.
```

```
Current Cube State:

Layer 1:
[' 88', ' 32', '115', '  6', ' 74']
[' 13', ' 75', ' 69', ' 51', '107']
['104', ' 17', '103', ' 12', ' 79']
['100', ' 65', '125', '  9', ' 93']
[' 22', ' 46', ' 59', ' 58', ' 40']

Layer 2:
[' 42', ' 64', '117', ' 82', ' 10']
[' 62', ' 94', ' 33', '101', ' 25']
['119', ' 63', ' 72', '113', '105']
[' 61', ' 21', '  8', ' 55', ' 76']
[' 31', ' 73', ' 95', '102', ' 52']

Layer 3:
['  2', ' 86', ' 47', '121', ' 41']
['111', ' 11', ' 85', '110', ' 98']
[' 91', ' 14', ' 29', ' 53', '  3']
[' 43', ' 67', ' 96', '108', ' 24']
[' 68', '  1', ' 50', '123', ' 99']

Layer 4:
[' 70', ' 37', '120', '  7', ' 81']
[' 80', ' 45', ' 16', ' 27', '116']
[' 20', ' 97', '122', ' 71', ' 34']
[' 54', ' 84', ' 30', ' 60', ' 87']
[' 83', '118', ' 92', '  4', ' 18']

Layer 5:
[' 36', '114', ' 89', ' 15', '109']
[' 49', ' 90', '112', ' 26', ' 23']
[' 38', '124', ' 39', ' 66', ' 48']
[' 57', ' 78', ' 56', ' 44', ' 35']
['  5', ' 77', ' 19', ' 28', '106']

Current Value: 43
Magic cube not yet solved.
14.311530113220215
31
```



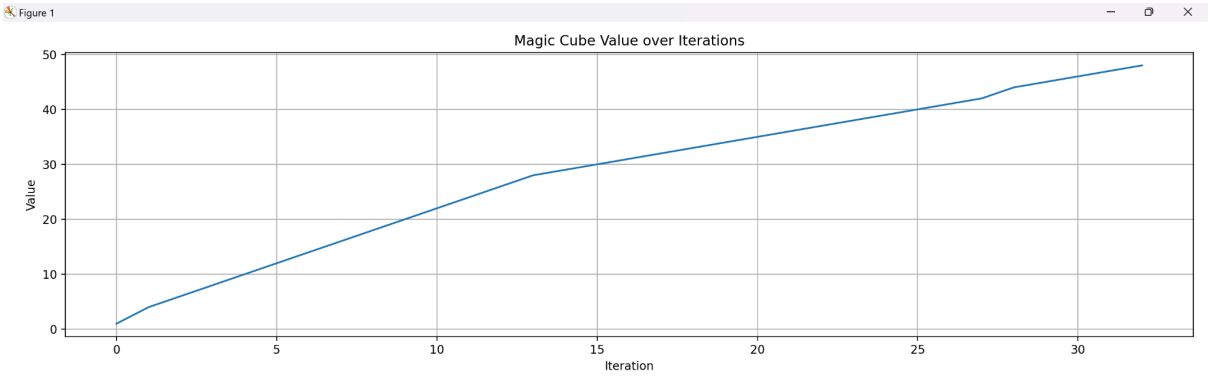
Pada eksperimen pertama, hasil yang didapatkan adalah sebagai berikut:

- **Initial state value:** 1
- **Final state value:** 43
- **Total iterations:** 31
- **Time duration:** 14.31 detik

3.1.1.2. Eksperimen 2

```
Current Cube State:

Layer 1:
[' 90', ' 85', ' 65', '107', '124']
['102', '104', ' 38', ' 68', ' 61']
['123', ' 55', ' 60', ' 12', ' 99']
[' 69', ' 45', ' 88', '  9', ' 28']
[' 49', ' 66', ' 41', ' 75', '117']

Layer 2:
['  4', '  7', ' 57', ' 83', ' 37']
[' 58', '116', ' 67', ' 78', ' 35']
[' 56', '106', ' 30', '114', ' 42']
[' 73', '  3', ' 87', ' 32', ' 13']
['103', ' 84', ' 62', ' 15', '122']

Layer 3:
[' 11', ' 54', ' 77', ' 20', ' 40']
[' 43', '105', '  2', '108', ' 98']
[' 48', '  8', '115', ' 27', ' 59']
[' 53', ' 89', '120', ' 82', ' 51']
[' 97', ' 34', ' 25', ' 10', ' 96']

Layer 4:
[' 29', '119', '  6', ' 94', ' 91']
[' 80', '125', ' 16', ' 23', ' 19']
[' 24', ' 71', ' 70', ' 64', ' 86']
[' 72', ' 47', ' 46', ' 63', '118']
[' 36', '112', '101', ' 18', ' 22']

Layer 5:
[' 81', '111', '  1', ' 95', '110']
['113', ' 39', ' 44', ' 79', ' 76']
[' 31', ' 33', ' 50', ' 14', ' 93']
['100', ' 17', '121', ' 92', ' 26']
[' 52', ' 21', '109', ' 74', '  5']

Current Value: 1
Magic cube not yet solved.
```

```
Current Cube State:

Layer 1:
[' 25', ' 85', ' 99', ' 13', ' 93']
[' 75', '104', '  7', ' 68', ' 61']
[' 12', ' 55', ' 60', '123', ' 65']
[' 63', ' 45', '119', '  9', '100']
[' 49', ' 26', ' 30', '102', '117']

Layer 2:
['  4', ' 54', ' 97', ' 83', ' 90']
[' 58', '116', ' 28', ' 78', ' 35']
[' 77', '106', ' 41', '107', ' 42']
[' 73', '  3', ' 87', ' 32', '120']
['103', ' 36', ' 62', ' 15', '122']

Layer 3:
[' 11', ' 79', '112', ' 88', ' 40']
[' 43', '105', ' 31', '108', '124']
[' 48', '  8', ' 21', ' 27', ' 72']
[' 53', ' 89', '114', ' 82', '109']
[' 57', ' 34', ' 37', ' 10', ' 96']

Layer 4:
[' 29', ' 16', '  6', ' 14', ' 91']
[' 80', '125', ' 20', ' 23', ' 19']
[' 24', ' 71', ' 70', ' 64', ' 86']
[' 59', ' 47', ' 46', ' 69', ' 94']
[' 84', ' 56', '101', ' 18', ' 22']

Layer 5:
[' 81', '111', '  1', ' 95', '110']
['113', ' 39', ' 44', ' 38', ' 76']
[' 2', ' 33', ' 98', '118', ' 50']
[' 67', ' 17', '121', ' 92', ' 66']
[' 52', '115', ' 51', ' 74', '  5']

Current Value: 48
Magic cube not yet solved.
17.065733432769775
32
```



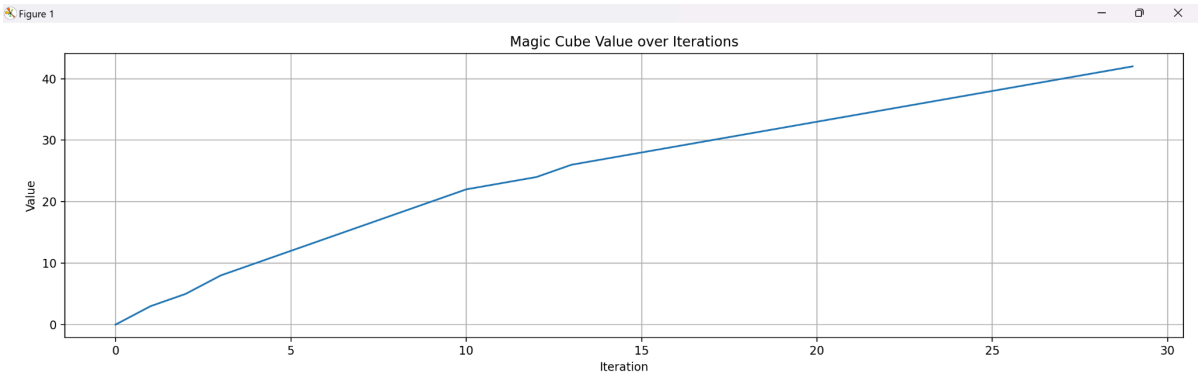
Pada eksperimen kedua, hasil yang didapatkan adalah sebagai berikut:

- **Initial state value:** 1
- **Final state value:** 48
- **Total iterations:** 32
- **Time duration:** 17.07 detik

3.1.1.3. Eksperimen 3

```
Current Cube State:

Layer 1:
['104', ' 86', ' 17', ' 32', '120']
[' 52', ' 65', ' 67', ' 15', '117']
['101', ' 90', ' 26', '118', ' 20']
[' 58', ' 11', ' 40', ' 54', ' 76']
['108', ' 93', ' 16', ' 31', ' 69']

Layer 2:
[' 64', ' 96', ' 47', ' 94', ' 38']
['109', ' 41', ' 55', ' 68', ' 23']
['  2', ' 30', ' 25', ' 22', ' 91']
['116', '119', ' 35', ' 29', ' 50']
[' 57', '106', '  4', ' 89', ' 19']

Layer 3:
[' 72', '  3', ' 99', ' 60', ' 13']
[' 75', ' 74', ' 95', '115', '  1']
[' 45', ' 37', '105', '102', ' 92']
[' 62', ' 39', ' 14', ' 10', '  5']
[' 46', ' 85', ' 42', '111', ' 84']

Layer 4:
[' 87', ' 56', ' 49', ' 28', '110']
[' 73', ' 24', ' 59', '107', ' 12']
[' 82', '  6', '  9', ' 34', ' 78']
[' 18', '114', '  7', '  8', '100']
[' 66', '112', '122', ' 27', ' 98']

Layer 5:
[' 21', ' 88', ' 70', ' 97', ' 51']
['124', ' 81', ' 79', '123', ' 83']
[' 43', ' 61', ' 36', ' 80', '121']
[' 63', ' 53', ' 71', ' 44', '103']
['113', ' 33', ' 77', ' 48', '125']

Current Value: 0
Magic cube not yet solved.
```

```
Current Cube State:

Layer 1:
['101', ' 45', ' 17', ' 32', '120']
[' 80', ' 65', ' 67', ' 50', ' 63']
[' 61', ' 90', ' 26', '118', ' 20']
[' 75', ' 11', '107', ' 54', ' 68']
['108', '104', ' 40', ' 35', ' 69']

Layer 2:
[' 64', '123', ' 47', ' 94', ' 38']
['  8', ' 41', ' 55', ' 76', ' 23']
[' 70', ' 30', ' 25', ' 22', ' 91']
['116', '119', ' 31', ' 34', ' 15']
[' 57', '106', '  4', ' 89', ' 59']

Layer 3:
[' 42', '  3', '117', ' 60', ' 13']
[' 33', ' 74', ' 95', '112', '  1']
[' 86', ' 96', '105', '102', '  5']
[' 92', ' 39', ' 99', ' 10', ' 37']
[' 46', '  2', ' 72', '111', ' 84']

Layer 4:
[' 87', ' 56', ' 49', ' 28', '110']
[' 73', ' 24', ' 19', ' 16', ' 12']
[' 82', '  6', '  9', ' 29', ' 78']
[' 18', '114', '  7', ' 97', '100']
[' 66', '115', '122', ' 27', ' 98']

Layer 5:
[' 21', ' 88', ' 85', '109', ' 51']
['124', ' 81', ' 79', ' 62', ' 83']
[' 43', ' 93', ' 36', ' 44', '121']
[' 14', ' 53', ' 71', ' 52', '103']
['113', ' 58', ' 77', ' 48', '125']

Current Value: 42
Magic cube not yet solved.
18.27035140991211
29
```



Pada eksperimen ketiga, hasil yang didapatkan adalah sebagai berikut:

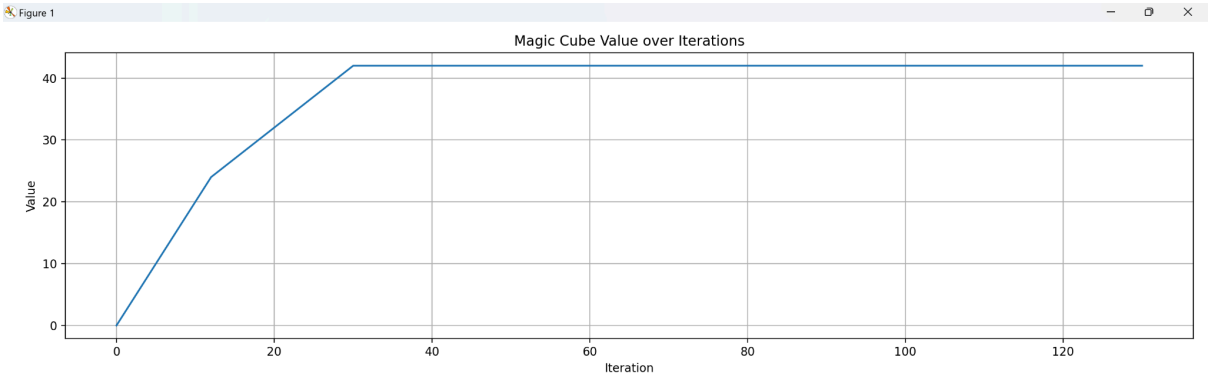
- **Initial state value:** 0
- **Final state value:** 42
- **Total iterations:** 29
- **Time duration:** 18.27 detik

3.1.2. Sideways-Moves Hill-Climbing

3.1.2.1. Eksperimen 1

```
Current Cube State:

Layer 1:
['101', ' 69', ' 67', ' 82', ' 51']
[' 13', ' 38', ' 77', ' 32', ' 71']
[' 59', '105', '  5', ' 79', '104']
[' 90', ' 93', ' 75', ' 30', '117']
[' 84', ' 86', ' 29', ' 63', ' 49']

Layer 2:
[' 25', ' 88', ' 66', '115', '  7']
['125', '100', ' 53', ' 14', '102']
[' 26', ' 42', '109', ' 11', ' 96']
[' 46', ' 73', ' 70', ' 12', '112']
[' 18', ' 48', ' 87', ' 28', ' 36']

Layer 3:
[' 56', '  1', '  8', ' 24', '120']
[' 78', ' 34', ' 10', ' 39', ' 81']
[' 35', ' 76', ' 19', '111', ' 85']
[' 64', '124', ' 62', '  4', ' 92']
['  9', ' 91', ' 54', '103', ' 31']

Layer 4:
['107', '108', '  2', ' 99', ' 72']
[' 61', '106', ' 16', ' 58', ' 45']
['113', '121', '116', ' 22', ' 43']
[' 23', '  6', ' 27', ' 68', ' 55']
['123', '122', ' 57', '118', '110']

Layer 5:
[' 41', ' 44', ' 50', ' 21', ' 17']
[' 37', ' 20', ' 97', ' 94', ' 60']
[' 95', ' 52', ' 74', ' 33', ' 83']
[' 98', '119', '  3', ' 65', ' 47']
['114', ' 15', ' 80', ' 89', ' 40']

Current Value: 0
Magic cube not yet solved.
```

```
Current Cube State:

Layer 1:
[' 51', ' 30', ' 15', '118', '101']
[' 31', ' 38', ' 77', ' 32', ' 71']
[' 59', ' 68', '  5', ' 79', '104']
[' 90', ' 93', ' 42', ' 36', '117']
[' 84', ' 86', ' 29', ' 50', ' 92']

Layer 2:
[' 25', ' 88', ' 66', ' 53', '  7']
['125', '100', '115', '108', '110']
[' 26', '  9', '109', '114', ' 96']
[' 48', ' 73', ' 70', ' 12', '112']
[' 18', ' 75', ' 87', ' 28', ' 69']

Layer 3:
[' 64', '  1', ' 60', ' 24', '120']
[' 78', ' 34', ' 10', ' 39', ' 81']
[' 35', ' 65', ' 19', '111', ' 85']
[' 56', '124', ' 82', '  4', ' 49']
[' 13', ' 91', ' 62', '103', ' 46']

Layer 4:
[' 55', ' 14', '107', ' 99', ' 72']
[' 61', '106', ' 16', ' 58', ' 45']
['113', '121', '116', ' 22', ' 43']
[' 23', '  6', ' 27', '105', '  2']
[' 63', '122', ' 57', ' 54', '102']

Layer 5:
[' 41', ' 89', ' 67', ' 21', ' 17']
[' 20', ' 37', ' 97', ' 94', '  8']
[' 95', ' 52', ' 74', ' 33', ' 83']
[' 98', '119', '  3', '123', ' 47']
[' 11', ' 76', ' 80', ' 44', ' 40']

Current Value: 42
Magic cube not yet solved.
71.12922477722168
130
Total sideways moves: 100
```



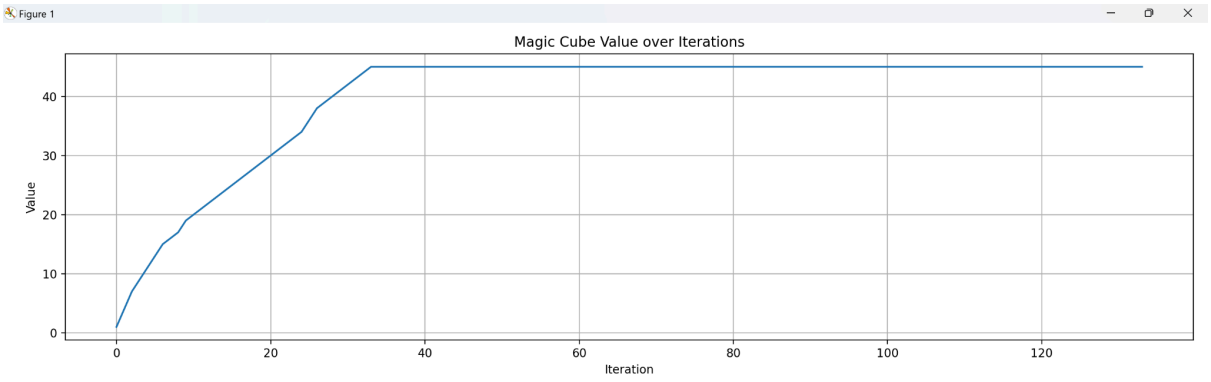
Pada eksperimen pertama, hasil yang didapatkan adalah sebagai berikut:

- **Initial state value:** 0
- **Final state value:** 42
- **Total iterations:** 130
- **Max sideways:** 100
- **Total sideways:** 100
- **Time duration:** 71.13 detik

3.1.2.2. Eksperimen 2

```
Current Cube State:

Layer 1:
[' 42', ' 65', '118', '  9', ' 12']
[' 87', ' 59', ' 22', ' 25', ' 94']
[' 74', ' 33', ' 16', ' 39', ' 95']
['106', ' 98', '123', ' 90', ' 57']
[' 28', ' 71', ' 50', '109', ' 32']

Layer 2:
[' 17', ' 97', ' 93', ' 44', ' 79']
[' 64', ' 19', ' 37', '124', '112']
[' 40', ' 76', ' 81', ' 36', ' 96']
[' 63', '  1', ' 14', ' 29', '120']
['108', '125', ' 83', ' 80', '117']

Layer 3:
[' 88', ' 43', ' 46', ' 70', ' 56']
[' 55', ' 85', ' 47', '114', ' 23']
[' 75', '  8', ' 67', ' 21', ' 20']
['116', '  6', '  7', ' 68', ' 31']
[' 11', '102', '107', ' 58', '  4']

Layer 4:
['101', '104', ' 35', '  3', '103']
[' 89', ' 38', ' 18', ' 15', ' 24']
[' 13', ' 41', ' 34', '122', '111']
[' 78', ' 72', ' 45', ' 48', ' 86']
[' 82', ' 61', ' 54', '110', '  2']

Layer 5:
['119', '115', ' 69', ' 26', ' 10']
['  5', ' 92', ' 91', ' 53', '100']
[' 60', ' 62', '113', ' 66', ' 27']
[' 30', ' 73', ' 52', ' 49', ' 84']
[' 77', ' 99', '105', '121', ' 51']

Current Value: 1
Magic cube not yet solved.
```

```
Current Cube State:

Layer 1:
['118', '106', ' 70', '  9', '105']
['124', ' 59', ' 56', ' 68', '  8']
[' 23', ' 33', ' 16', ' 39', '125']
[' 48', ' 98', '123', ' 90', ' 45']
[' 28', ' 19', ' 50', '109', ' 32']

Layer 2:
[' 17', ' 97', ' 93', ' 80', ' 79']
[' 42', ' 71', ' 44', ' 46', '112']
[' 40', ' 76', ' 81', ' 22', ' 96']
[' 43', '  1', ' 14', ' 29', ' 69']
['108', ' 95', ' 83', ' 38', '117']

Layer 3:
[' 58', ' 99', ' 87', ' 64', '  7']
[' 55', ' 52', ' 20', '114', ' 74']
[' 75', '102', ' 67', ' 24', ' 47']
['116', '  6', ' 36', ' 25', ' 31']
[' 11', ' 82', '107', ' 88', '113']

Layer 4:
[' 18', '104', ' 35', '  3', '103']
[' 89', ' 37', '101', ' 15', ' 21']
[' 13', ' 41', ' 34', '122', '111']
[' 78', ' 72', ' 57', ' 65', ' 86']
[' 91', ' 61', ' 63', '110', '  2']

Layer 5:
['119', '115', '120', ' 26', ' 10']
['  5', ' 92', ' 94', ' 53', '100']
[' 60', ' 62', '  4', ' 66', ' 27']
[' 30', ' 73', ' 85', ' 49', ' 84']
[' 77', ' 54', ' 12', '121', ' 51']

Current Value: 45
Magic cube not yet solved.
75.31842088699341
133
Total sideways moves: 100
```



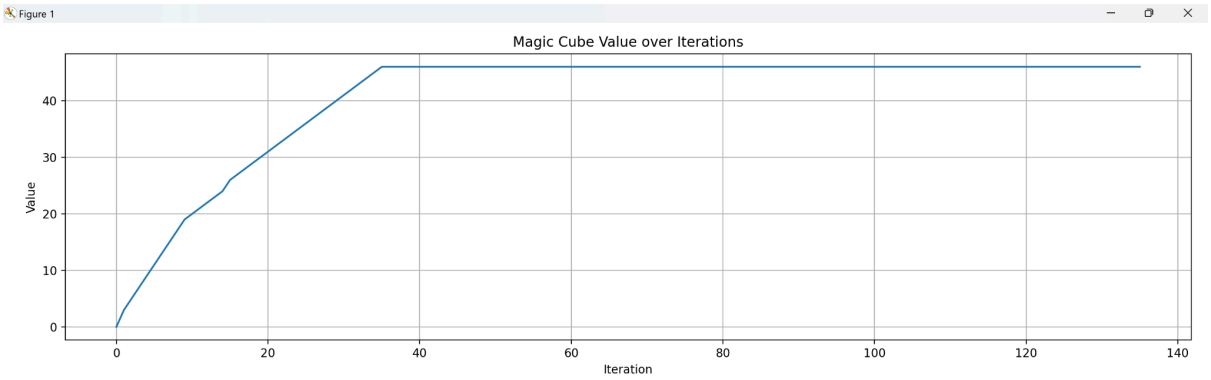
Pada eksperimen kedua, hasil yang didapatkan adalah sebagai berikut:

- **Initial state value:** 1
- **Final state value:** 45
- **Total iterations:** 133
- **Max sideways:** 100
- **Total sideways:** 100
- **Time duration:** 75.32 detik

3.1.2.3. Eksperimen 3

```
Current Cube State:

Layer 1:
[' 53', ' 29', '112', '  8', '  1']
['  9', ' 63', ' 24', '105', ' 14']
[' 60', ' 11', ' 72', '100', '  5']
['122', ' 71', ' 90', ' 52', ' 33']
[' 83', ' 32', ' 19', '120', ' 88']

Layer 2:
['102', ' 91', ' 82', ' 96', ' 39']
['104', '118', ' 20', ' 66', ' 64']
['123', ' 36', ' 89', ' 55', ' 98']
['114', ' 40', ' 38', '115', ' 65']
['116', ' 70', '110', ' 57', ' 41']

Layer 3:
[' 12', ' 69', ' 15', '113', '109']
['107', ' 76', ' 25', ' 61', ' 51']
[' 94', ' 99', ' 23', ' 58', '108']
['  7', ' 50', ' 54', ' 22', ' 49']
[' 93', ' 43', ' 59', '119', ' 44']

Layer 4:
[' 26', ' 42', ' 35', '103', ' 30']
[' 34', ' 67', ' 80', '101', ' 95']
[' 81', ' 46', '111', ' 68', ' 28']
[' 85', ' 45', '  4', ' 84', ' 56']
[' 18', ' 31', ' 78', ' 73', ' 37']

Layer 5:
['106', ' 47', '124', ' 87', ' 77']
[' 10', ' 79', ' 16', ' 48', '117']
[' 74', ' 27', ' 13', ' 17', ' 75']
[' 97', '  3', '  2', '125', ' 62']
['  6', ' 92', ' 21', ' 86', '121']

Current Value: 0
Magic cube not yet solved.
```

```
Current Cube State:

Layer 1:
[' 57', '120', ' 59', '  8', ' 71']
[' 60', ' 63', ' 73', '105', ' 14']
['  9', ' 33', ' 55', '100', '118']
['110', '  1', ' 36', ' 52', ' 24']
[' 83', ' 31', ' 22', ' 50', ' 88']

Layer 2:
['102', ' 98', ' 35', ' 96', ' 39']
['104', ' 30', ' 51', ' 66', ' 64']
[' 97', ' 43', ' 89', ' 72', '106']
['114', '  5', ' 78', ' 53', ' 65']
['116', ' 70', '122', ' 77', ' 41']

Layer 3:
[' 81', ' 69', ' 15', '113', '109']
['107', ' 76', ' 85', ' 61', ' 25']
[' 27', ' 99', ' 23', ' 58', '108']
['  7', ' 29', ' 54', ' 91', ' 49']
[' 93', ' 90', '112', '119', ' 44']

Layer 4:
[' 16', ' 42', ' 82', ' 11', ' 40']
[' 34', ' 67', ' 80', '101', ' 95']
[' 12', ' 46', '111', ' 68', ' 28']
[' 20', ' 45', '  4', ' 84', '115']
[' 18', ' 32', ' 38', '103', ' 37']

Layer 5:
[' 19', ' 47', '124', ' 87', ' 56']
[' 10', ' 79', ' 26', ' 48', '117']
[' 74', ' 94', ' 13', ' 17', ' 86']
['123', '  3', '  2', '125', ' 62']
['  6', ' 92', ' 21', ' 75', '121']

Current Value: 46
Magic cube not yet solved.
72.44485402107239
135
Total sideways moves: 100
```



Pada eksperimen ketiga, hasil yang didapatkan adalah sebagai berikut:

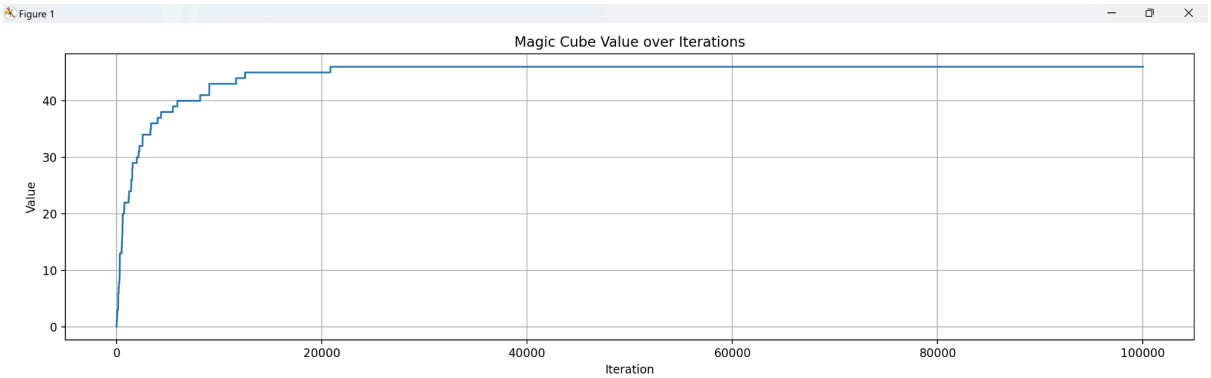
- **Initial state value:** 0
- **Final state value:** 46
- **Total iterations:** 135
- **Max sideways:** 100
- **Total sideways:** 100
- **Time duration:** 72.44 detik

3.1.3. Stochastic Hill-Climbing

3.1.3.1. Eksperimen 1

```
Current Cube State:

Layer 1:
['  7', '115', ' 92', '108', ' 23']
['123', ' 91', '106', ' 24', ' 38']
[' 32', ' 52', '102', ' 44', ' 61']
[' 39', ' 41', ' 70', ' 67', ' 31']
[' 82', ' 57', ' 59', ' 29', ' 49']

Layer 2:
[' 94', ' 65', '121', ' 36', ' 87']
['111', ' 22', '  9', '113', ' 30']
['114', '103', ' 89', ' 60', ' 88']
['125', '  6', '101', ' 28', '119']
[' 76', ' 77', '  8', ' 19', '  1']

Layer 3:
[' 71', '  2', ' 68', ' 81', ' 27']
[' 46', '110', ' 53', ' 42', '104']
['116', ' 84', '112', ' 16', ' 17']
[' 80', ' 95', '120', ' 13', ' 75']
[' 35', ' 62', '117', '122', ' 93']

Layer 4:
[' 90', ' 64', ' 48', ' 74', '  5']
[' 98', ' 69', ' 79', ' 45', ' 21']
[' 55', ' 86', ' 66', ' 96', ' 37']
[' 78', ' 85', '105', ' 33', ' 25']
[' 40', ' 63', '109', ' 26', ' 73']

Layer 5:
[' 50', '124', ' 51', ' 58', ' 99']
[' 15', '118', ' 97', '107', ' 43']
[' 47', ' 11', '  3', '  4', ' 83']
[' 14', ' 34', ' 56', '100', ' 10']
[' 12', ' 54', ' 72', ' 20', ' 18']

Current Value: 0
Magic cube not yet solved.
```

```
Current Cube State:

Layer 1:
[' 80', '115', ' 52', '108', ' 75']
['123', ' 91', '106', '124', ' 38']
[' 32', ' 31', ' 28', '116', ' 61']
[' 39', ' 47', ' 70', ' 67', ' 92']
[' 41', '  8', ' 59', '117', ' 49']

Layer 2:
['105', ' 22', ' 23', ' 78', ' 87']
[' 10', ' 18', '  9', ' 57', '119']
['  4', '103', ' 89', ' 30', ' 88']
['125', '  6', ' 62', '102', ' 20']
[' 76', ' 77', '113', ' 48', '  1']

Layer 3:
[' 96', ' 42', '121', ' 81', ' 27']
['  2', '110', ' 53', ' 46', '104']
[' 44', ' 84', '112', ' 58', ' 17']
[' 94', ' 95', '120', ' 13', ' 19']
[' 35', ' 15', ' 50', '122', ' 93']

Layer 4:
[' 74', ' 12', ' 68', ' 90', '  5']
['101', ' 69', ' 79', ' 45', ' 21']
[' 55', ' 86', ' 66', ' 71', ' 37']
[' 43', ' 85', '  7', ' 33', ' 25']
['114', ' 63', '109', ' 82', ' 73']

Layer 5:
[' 29', ' 24', ' 51', ' 16', '107']
[' 98', '118', ' 97', ' 99', ' 36']
[' 26', ' 11', '  3', ' 40', ' 83']
[' 14', ' 34', ' 56', '100', '111']
[' 72', ' 54', ' 64', ' 60', ' 65']

Current Value: 46
Magic cube not yet solved.
7.248444318771362
100000
```



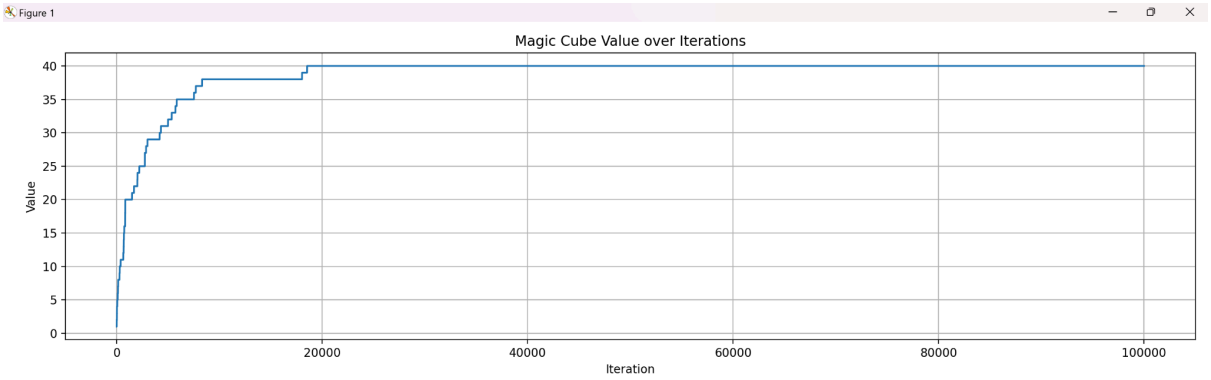
Pada eksperimen pertama, hasil yang didapatkan adalah sebagai berikut:

- **Initial state value:** 0
- **Final state value:** 46
- **Total iterations:** 100000
- **Time duration:** 7.25 detik

3.1.3.2. Eksperimen 2

```
Current Cube State:

Layer 1:
[' 95', '124', ' 46', '114', ' 37']
[' 80', ' 86', ' 50', '107', ' 24']
['105', ' 63', '115', ' 25', ' 45']
[' 43', '  1', '117', '101', ' 15']
['  3', ' 66', '108', '104', ' 20']

Layer 2:
[' 96', '113', '106', ' 64', '122']
[' 17', ' 71', ' 79', ' 85', ' 75']
['111', ' 68', ' 89', ' 52', ' 54']
[' 82', '102', ' 42', '  7', ' 88']
['109', ' 51', '118', '  9', ' 29']

Layer 3:
[' 33', ' 92', ' 40', ' 99', ' 77']
[' 81', '  8', ' 47', '  4', ' 39']
[' 55', ' 22', '  6', ' 30', ' 12']
[' 78', ' 98', ' 32', ' 38', '  2']
['123', ' 36', ' 90', '125', ' 60']

Layer 4:
[' 26', ' 83', ' 70', ' 53', ' 21']
['103', ' 49', '116', ' 74', ' 44']
[' 11', ' 18', ' 65', ' 57', ' 34']
[' 28', '112', ' 87', ' 91', ' 56']
[' 76', ' 67', ' 10', '110', ' 16']

Layer 5:
['120', ' 84', ' 31', ' 73', ' 41']
[' 62', '121', ' 59', ' 97', ' 27']
['100', ' 69', ' 23', ' 14', ' 72']
[' 19', ' 13', ' 93', '  5', ' 94']
['119', ' 48', ' 58', ' 35', ' 61']

Current Value: 1
Magic cube not yet solved.
```

```
Current Cube State:

Layer 1:
[' 95', '124', ' 46', ' 13', ' 37']
[' 80', '  8', ' 79', ' 68', '  2']
['105', ' 63', ' 77', ' 25', ' 45']
[' 43', '  1', '117', '101', ' 53']
['  3', ' 66', '108', '104', ' 34']

Layer 2:
[' 96', '114', '106', ' 64', '122']
[' 17', ' 71', ' 19', ' 85', ' 22']
['111', ' 12', '112', ' 52', ' 54']
['121', ' 16', ' 27', '  7', ' 88']
['109', ' 51', ' 49', '107', ' 29']

Layer 3:
[' 33', ' 92', ' 40', ' 35', '115']
[' 81', ' 28', ' 42', '125', ' 39']
[' 55', '103', '118', ' 30', '  9']
[' 23', ' 78', ' 32', ' 38', ' 24']
['123', ' 36', ' 90', ' 87', ' 98']

Layer 4:
[' 31', ' 83', '  5', ' 15', ' 11']
[' 75', '  6', '116', ' 74', ' 44']
[' 21', ' 70', ' 65', ' 57', '102']
[' 86', ' 89', '119', '120', ' 56']
[' 76', ' 67', ' 10', '110', ' 93']

Layer 5:
[' 91', ' 84', ' 26', ' 73', ' 41']
[' 62', ' 50', ' 59', ' 97', ' 47']
['100', ' 69', ' 60', ' 14', ' 72']
[' 82', '113', ' 20', ' 18', ' 94']
['  4', ' 48', ' 58', ' 99', ' 61']

Current Value: 40
Magic cube not yet solved.
7.693896293640137
100000
```



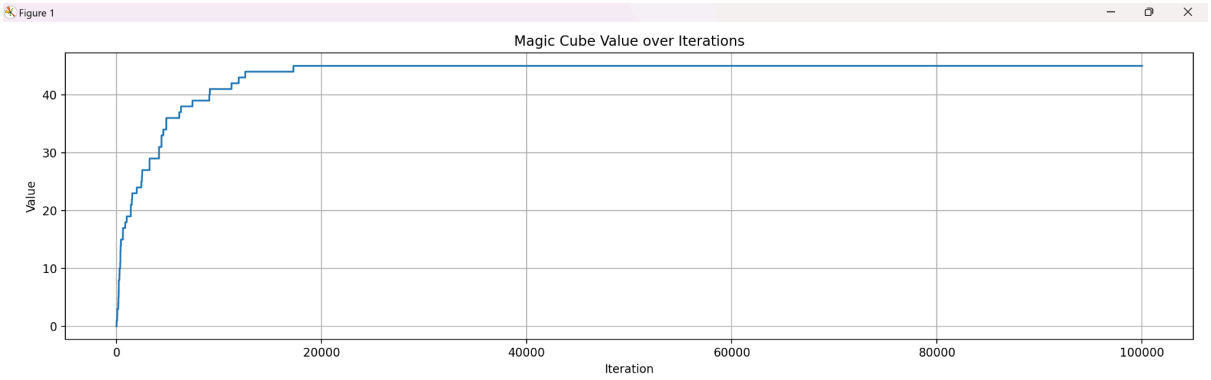
Pada eksperimen kedua, hasil yang didapatkan adalah sebagai berikut:

- **Initial state value:** 1
- **Final state value:** 40
- **Total iterations:** 100000
- **Time duration:** 7.69 detik

3.1.3.3. Eksperimen 3

```
Current Cube State:

Layer 1:
[' 74', '117', '  5', '  2', ' 22']
['107', ' 91', '  9', '121', ' 75']
[' 45', ' 90', ' 52', ' 43', ' 93']
[' 26', ' 67', ' 21', ' 17', ' 99']
[' 97', '  6', '  4', ' 30', '114']

Layer 2:
[' 95', '100', ' 69', ' 29', ' 65']
[' 58', ' 46', ' 28', ' 81', '101']
['105', ' 14', ' 15', ' 96', ' 24']
[' 60', ' 64', ' 55', ' 73', ' 35']
[' 80', ' 34', ' 94', ' 68', ' 82']

Layer 3:
['123', '102', '125', ' 63', ' 39']
['122', ' 62', '108', ' 47', ' 77']
[' 89', ' 98', '113', ' 88', ' 70']
[' 59', ' 19', ' 33', ' 27', ' 72']
[' 23', '118', ' 31', '106', ' 40']

Layer 4:
[' 25', ' 53', ' 20', ' 18', '120']
[' 83', ' 92', '  8', ' 61', '124']
['111', '  1', ' 49', ' 37', ' 36']
[' 12', '  3', '103', ' 84', ' 79']
[' 54', '119', ' 50', ' 41', ' 85']

Layer 5:
[' 51', ' 86', ' 11', ' 87', ' 32']
[' 7', ' 76', ' 78', ' 57', ' 44']
[' 42', '116', '104', '115', ' 66']
[' 10', ' 38', ' 71', ' 16', '109']
[' 56', ' 48', '112', '110', ' 13']

Current Value: 0
Magic cube not yet solved.
```

```
Current Cube State:

Layer 1:
[' 25', '120', ' 59', '  2', ' 57']
['107', '101', ' 66', ' 42', ' 65']
[' 45', ' 21', ' 52', '104', ' 93']
[' 41', ' 67', ' 20', '111', ' 76']
[' 97', '  6', '118', ' 83', ' 32']

Layer 2:
[' 95', '100', ' 69', ' 29', ' 75']
[' 58', ' 46', ' 31', ' 81', ' 99']
[' 36', ' 14', ' 15', ' 60', ' 24']
[' 96', ' 64', ' 55', ' 77', ' 35']
[' 80', ' 91', ' 12', ' 68', ' 82']

Layer 3:
['123', ' 78', '125', ' 79', ' 44']
['  1', ' 62', '108', ' 71', ' 73']
[' 89', ' 98', ' 63', ' 88', ' 70']
[' 74', '109', ' 33', ' 27', ' 72']
[' 28', '  4', ' 23', '106', ' 40']

Layer 4:
['  5', ' 53', '105', ' 18', '117']
[' 30', ' 92', '  8', ' 61', '124']
['122', ' 17', ' 49', ' 37', ' 90']
[' 94', ' 34', '103', ' 84', '113']
[' 54', '119', ' 50', '115', ' 85']

Layer 5:
[' 51', '110', ' 11', ' 87', ' 22']
['  7', '  3', '102', '114', ' 39']
['121', '116', ' 43', ' 26', '  9']
[' 10', ' 38', ' 47', ' 16', ' 19']
[' 56', ' 48', '112', ' 86', ' 13']

Current Value: 45
Magic cube not yet solved.
7.1169114112854
100000
```



Pada eksperimen ketiga, hasil yang didapatkan adalah sebagai berikut:

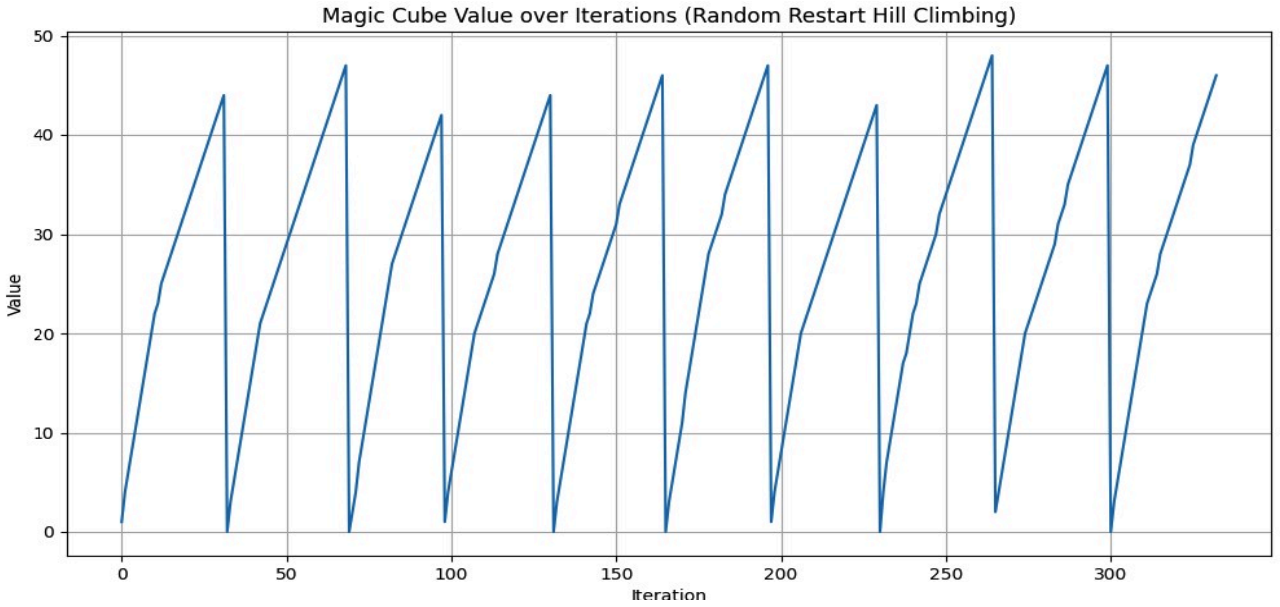
- **Initial state value:** 0
- **Final state value:** 45
- **Total iterations:** 100000
- **Time duration:** 7.12 detik

3.1.4. Random Restart Hill-Climbing

3.1.4.1. Eksperimen 1

```
Trial 1:
Initial state:

Current Cube State:

Layer 1:
['  8', '110', '  7', ' 47', ' 53']
[' 91', ' 95', ' 97', ' 48', ' 31']
[' 59', ' 36', ' 88', ' 89', ' 84']
[' 39', '123', ' 98', ' 15', ' 68']
['  9', ' 62', '108', ' 60', ' 81']

Layer 2:
[' 20', '109', ' 33', '113', '  5']
[' 28', ' 94', '102', ' 46', ' 23']
[' 40', ' 80', ' 76', ' 65', ' 12']
[' 32', ' 82', '124', '120', '  4']
[' 14', '117', ' 92', ' 13', '119']

Layer 3:
[' 37', ' 75', '112', ' 83', ' 10']
[' 66', ' 93', ' 56', ' 73', ' 30']
[' 42', ' 67', ' 35', ' 85', '115']
['118', '107', ' 72', '114', '101']
['  6', ' 77', ' 11', '100', ' 54']

Layer 4:
[' 26', '  3', '106', ' 34', '122']
['125', '  2', '121', ' 79', ' 25']
[' 74', ' 86', ' 55', ' 61', ' 24']
[' 51', ' 21', ' 57', ' 22', ' 43']
['111', ' 58', ' 70', ' 96', ' 50']

Layer 5:
[' 64', ' 44', ' 49', ' 38', ' 27']
[' 71', ' 16', ' 19', ' 18', '  1']
[' 87', ' 17', ' 41', ' 90', ' 69']
[' 52', '103', '116', '105', ' 45']
[' 78', ' 63', ' 99', '104', ' 29']

Current Value: 1
Magic cube not yet solved.
Initial value: 1

Restart 1 - Current best value: 51
Restart 2 - Current best value: 51
Restart 3 - Current best value: 51
Restart 4 - Current best value: 51
Restart 5 - Current best value: 51
Restart 6 - Current best value: 51
Restart 7 - Current best value: 51
Restart 8 - Current best value: 51
Restart 9 - Current best value: 51
Restart 10 - Current best value: 51
```

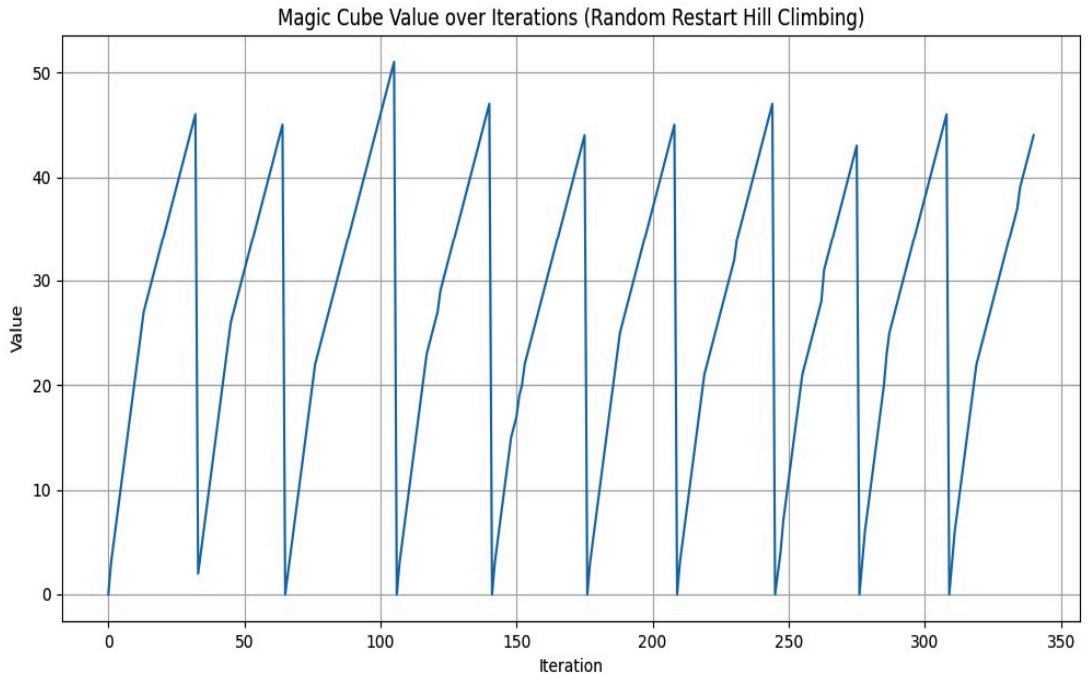
```
Final state:

Current Cube State:

Layer 1:
['108', ' 85', '113', ' 59', ' 68']
['110', '  1', ' 92', ' 87', ' 25']
[' 86', ' 78', ' 66', ' 30', ' 55']
[' 73', ' 79', ' 21', ' 24', '118']
[' 15', ' 72', ' 23', ' 89', '116']

Layer 2:
[' 51', ' 44', '105', '120', '  9']
[' 38', '  7', ' 56', ' 27', ' 91']
[' 22', '115', '109', ' 52', ' 17']
[' 49', ' 67', ' 43', ' 42', '114']
['103', ' 62', '  2', ' 74', '106']

Layer 3:
[' 47', ' 71', ' 48', ' 16', '  6']
[' 14', ' 33', ' 45', '117', '102']
[' 10', ' 53', ' 90', ' 61', '101']
[' 19', ' 97', ' 83', ' 20', '111']
[' 5', ' 75', ' 46', ' 64', '125']

Layer 4:
[' 57', ' 81', ' 37', '  8', ' 58']
[' 31', ' 98', ' 70', ' 84', ' 32']
['104', ' 26', ' 18', ' 13', ' 82']
['124', ' 95', ' 69', ' 88', ' 63']
[' 60', ' 77', '121', '  3', ' 54']

Layer 5:
[' 94', ' 34', ' 12', '112', '107']
['122', ' 76', ' 40', ' 36', ' 65']
[' 93', ' 80', ' 41', ' 39', ' 11']
[' 50', ' 96', ' 99', '100', '119']
[' 35', ' 29', '123', ' 28', '  4']

Current Value: 51
Magic cube not yet solved.

Results:
Number of restarts: 10
Total iterations: 327
Best value found: 51
Time taken: 280.06 seconds
```



3.1.4.2. Eksperimen 2

```
Trial 2:
Initial state:

Current Cube State:

Layer 1:
[' 50', ' 80', ' 87', ' 93', ' 90']
[' 44', ' 15', '  3', '121', ' 33']
['104', ' 62', ' 94', ' 35', ' 10']
[' 63', '108', '105', '120', ' 84']
[' 61', '123', ' 78', ' 73', ' 24']

Layer 2:
[' 41', '122', '111', ' 58', ' 92']
[' 18', ' 85', ' 45', '119', '109']
[' 23', '100', ' 66', '101', ' 72']
[' 40', ' 39', '  6', ' 79', ' 81']
[' 25', '115', ' 38', '110', ' 68']

Layer 3:
['124', '  7', ' 77', ' 26', ' 28']
[' 27', ' 43', ' 83', ' 71', ' 49']
[' 21', ' 75', ' 11', ' 14', ' 22']
['118', ' 37', ' 57', ' 82', '114']
[' 13', ' 55', ' 60', ' 42', ' 91']

Layer 4:
['  8', ' 99', '113', '107', ' 46']
['  4', '  9', ' 76', ' 16', ' 96']
[' 31', ' 51', ' 54', ' 67', ' 89']
[' 29', ' 88', '102', ' 64', ' 70']
[' 36', ' 59', ' 32', ' 30', ' 53']

Layer 5:
['103', ' 97', ' 56', '116', ' 65']
[' 98', '112', ' 12', ' 74', '106']
[' 48', ' 95', ' 34', '  2', ' 86']
['  5', ' 69', ' 47', ' 20', ' 17']
[' 52', '  1', ' 19', '125', '117']

Current Value: 0
Magic cube not yet solved.
Initial value: 0

Restart 1 - Current best value: 46
Restart 2 - Current best value: 46
Restart 3 - Current best value: 51
Restart 4 - Current best value: 51
Restart 5 - Current best value: 51
Restart 6 - Current best value: 51
Restart 7 - Current best value: 51
Restart 8 - Current best value: 51
Restart 9 - Current best value: 51
Restart 10 - Current best value: 51
```

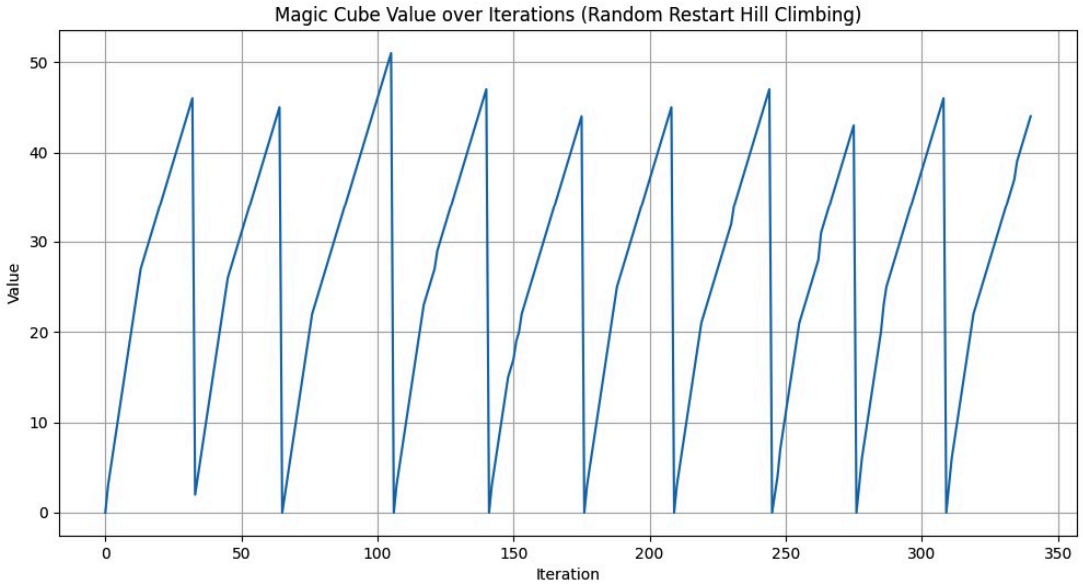
```
Final state:

Current Cube State:

Layer 1:
[' 86', '111', '  2', '121', ' 56']
['125', ' 25', ' 91', ' 67', '  7']
[' 97', ' 18', ' 42', ' 45', '103']
[' 43', '117', ' 74', ' 52', ' 39']
[' 95', ' 44', '106', ' 30', '110']

Layer 2:
[' 78', '108', ' 72', ' 17', ' 40']
[' 94', ' 90', ' 29', '105', ' 19']
[' 77', ' 80', ' 68', ' 62', ' 28']
[' 46', ' 82', ' 61', ' 57', ' 69']
[' 20', ' 76', ' 85', '112', ' 22']

Layer 3:
[' 11', ' 63', ' 36', ' 48', '119']
[' 23', ' 92', '  8', ' 83', '109']
[' 14', ' 81', ' 21', ' 89', '123']
[' 37', ' 66', '120', ' 38', ' 84']
[' 26', ' 13', ' 15', ' 64', ' 65']

Layer 4:
[' 99', '  9', '102', ' 58', ' 47']
[' 24', ' 73', ' 16', ' 87', '115']
['100', ' 12', '107', '113', '  1']
[' 88', ' 70', ' 50', '  3', '104']
['  4', '114', ' 93', ' 54', ' 33']

Layer 5:
[' 41', '118', ' 32', ' 71', ' 53']
[' 49', ' 35', '116', ' 79', '  5']
[' 27', '124', ' 98', '  6', ' 60']
['101', ' 51', ' 10', ' 31', '122']
[' 34', ' 96', ' 59', ' 55', ' 75']

Current Value: 51
Magic cube not yet solved.

Results:
Number of restarts: 10
Total iterations: 331
Best value found: 51
Time taken: 282.82 seconds
```



3.1.4.3. Eksperimen 3

```
Trial 3:
Initial state:

Current Cube State:

Layer 1:
['106', ' 24', ' 54', ' 27', ' 97']
[' 81', ' 53', '124', '118', ' 16']
[' 22', ' 12', '108', ' 85', ' 48']
['  9', ' 34', ' 19', '121', ' 68']
['  5', '  3', ' 99', ' 46', '101']

Layer 2:
[' 96', '  4', ' 49', '113', '123']
['120', ' 17', ' 30', '  6', ' 47']
[' 57', ' 60', '103', ' 65', '104']
['122', '105', ' 61', ' 92', ' 51']
[' 39', ' 10', ' 20', '102', ' 37']

Layer 3:
[' 73', ' 58', ' 98', ' 77', ' 86']
[' 26', ' 55', ' 95', ' 36', ' 70']
[' 21', ' 88', '115', '116', ' 83']
[' 74', ' 11', ' 80', ' 15', ' 35']
[' 42', ' 82', ' 31', ' 33', '  7']

Layer 4:
[' 67', ' 13', '111', ' 14', ' 75']
['107', ' 63', ' 66', ' 84', ' 89']
[' 72', ' 56', '117', ' 94', ' 38']
[' 62', '  1', '125', ' 29', '114']
[' 50', ' 23', '119', ' 52', ' 59']

Layer 5:
[' 43', ' 93', ' 32', ' 90', '109']
[' 71', '  8', ' 25', ' 69', ' 87']
[' 18', '  2', ' 78', ' 79', ' 91']
[' 28', '112', '100', ' 45', ' 76']
[' 44', ' 40', '110', ' 64', ' 41']

Current Value: 0
Magic cube not yet solved.
Initial value: 0

Restart 1 - Current best value: 46
Restart 2 - Current best value: 46
Restart 3 - Current best value: 49
Restart 4 - Current best value: 49
Restart 5 - Current best value: 49
Restart 6 - Current best value: 50
Restart 7 - Current best value: 50
Restart 8 - Current best value: 50
Restart 9 - Current best value: 51
Restart 10 - Current best value: 51
```

```
Final state:

Current Cube State:

Layer 1:
['101', ' 80', ' 87', '  9', ' 41']
[' 96', ' 22', ' 46', '120', ' 31']
[' 70', '  4', '111', ' 73', ' 51']
[' 16', ' 25', '  8', ' 47', ' 23']
[' 18', ' 58', ' 63', ' 66', ' 34']

Layer 2:
[' 54', ' 85', '106', ' 30', ' 40']
['124', ' 12', '122', ' 50', '  7']
[' 89', ' 88', '116', ' 72', ' 64']
[' 29', ' 35', ' 84', ' 55', '112']
[' 74', ' 69', ' 36', '108', ' 78']

Layer 3:
['103', ' 27', ' 17', '117', ' 98']
['  2', ' 28', '123', ' 77', ' 97']
[' 91', ' 53', ' 52', ' 19', '100']
['113', ' 82', '105', ' 94', ' 15']
['  6', '125', ' 56', ' 90', ' 38']

Layer 4:
[' 59', '118', ' 81', '102', '110']
[' 92', ' 49', ' 48', ' 83', ' 43']
[' 20', ' 95', ' 32', ' 37', ' 14']
[' 65', ' 11', ' 61', ' 60', '  5']
[' 79', ' 42', ' 93', '  3', '115']

Layer 5:
['109', ' 44', ' 24', '119', ' 10']
[' 62', ' 71', ' 39', ' 68', '  1']
[' 45', ' 75', ' 26', '114', ' 86']
['121', '104', ' 57', ' 76', ' 99']
['107', ' 21', ' 67', ' 13', ' 33']

Current Value: 51
Magic cube not yet solved.

Results:
Number of restarts: 10
Total iterations: 340
Best value found: 51
Time taken: 291.38 seconds
```



Eksperimen dilakukan sebanyak tiga kali dengan parameter yang sama (10 kali restart per percobaan), menghasilkan data spesifik pada setiap percobaan. Berikut ini adalah rangkuman hasil berdasarkan data pada setiap percobaan.

Eksperimen 1

- **Nilai Awal:** 2
- **Nilai Akhir Terbaik:** 48
- **Jumlah Iterasi:** 323
- **Jumlah Restart:** 10
- **Durasi Eksekusi:** 279.73 detik

Pada percobaan pertama, algoritma mencapai nilai tertinggi **48** setelah 10 kali restart, dengan jumlah iterasi total sebanyak **323**. Durasi eksekusi mencapai **279.73 detik**, menunjukkan waktu komputasi yang relatif singkat dibandingkan percobaan lainnya. Nilai ini cukup stabil, meskipun tidak mencapai nilai optimal.

Eksperimen 2

- **Nilai Awal:** 0
- **Nilai Akhir Terbaik:** 50
- **Jumlah Iterasi:** 329
- **Jumlah Restart:** 10
- **Durasi Eksekusi:** 285.12 detik

Pada percobaan kedua, algoritma berhasil mencapai nilai tertinggi **50** dengan jumlah iterasi sedikit lebih tinggi, yakni **329**. Durasi eksekusi **285.12 detik**, menunjukkan peningkatan kecil dari percobaan pertama. Hal ini menunjukkan bahwa, meskipun membutuhkan iterasi tambahan, algoritma masih dapat menjaga stabilitas waktu pemrosesan dan mencapai nilai yang lebih baik.

Eksperimen 3

- **Nilai Awal:** 0
- **Nilai Akhir Terbaik:** 51
- **Jumlah Iterasi:** 340
- **Jumlah Restart:** 10
- **Durasi Eksekusi:** 291.38 detik

Percobaan ketiga menunjukkan hasil terbaik dengan nilai akhir **51** setelah **340 iterasi**. Ini adalah nilai tertinggi di antara ketiga percobaan, meskipun membutuhkan durasi eksekusi yang lebih lama, yaitu **291.38 detik**. Meskipun ada kenaikan durasi, hasil akhir yang

lebih tinggi menunjukkan bahwa algoritma mampu mencapai nilai terbaik dengan trade-off durasi eksekusi.

3.1.4.4 Rata-Rata Keseluruhan

- Nilai Akhir Rata-rata: 49.67
- Durasi Eksekusi Rata-rata: 285.41 detik
- Jumlah Iterasi Rata-rata per Restart: 33.1

Dari ketiga percobaan, rata-rata nilai akhir yang dicapai adalah 49.67, dengan rata-rata durasi eksekusi 285.41 detik. Hal ini menunjukkan bahwa algoritma cukup konsisten dalam mencapai nilai yang tinggi meskipun tidak optimal. Jumlah iterasi rata-rata per restart adalah sekitar 33.1, yang menunjukkan stabilitas dalam proses iterasi pada setiap percobaan.

3.1.4.5 Analisis Hasil Berdasarkan Data

• Stabilitas Nilai Akhir: Nilai tertinggi pada tiap percobaan berada dalam kisaran 48 hingga 51, yang menunjukkan bahwa algoritma memiliki stabilitas dalam pencapaian nilai akhir meskipun terdapat fluktuasi kecil.

• Efisiensi Waktu Eksekusi: Meskipun durasi eksekusi sedikit bervariasi, algoritma tetap berada dalam rentang yang cukup efisien, dengan durasi rata-rata sekitar 285 detik.

• Jumlah Restart yang Konsisten: Algoritma melakukan 10 restart pada setiap percobaan tanpa modifikasi adaptif. Ini menunjukkan konsistensi dalam mekanisme restart yang membantu pencapaian nilai lebih tinggi.

3.1.4.6 Kelebihan Berdasarkan Data

• Konsistensi dalam Durasi dan Nilai Akhir: Dari ketiga percobaan, algoritma menunjukkan konsistensi dalam durasi pemrosesan dan hasil akhir, yang berguna dalam lingkungan di mana waktu eksekusi yang terukur penting.

• Kemampuan Menghindari Local Optima: Meskipun nilai akhir tidak mencapai optimum global (109), algoritma berhasil mencapai nilai lebih tinggi dibandingkan nilai awal dengan setiap restart, menunjukkan efektivitasnya dalam keluar dari *local optima*.

3.1.4.7 Keterbatasan Berdasarkan Data

• Nilai Akhir Masih di Bawah Optimum Global: Hasil terbaik yang diperoleh adalah 51, masih jauh dari nilai optimum 109. Hal ini menunjukkan bahwa algoritma mungkin memerlukan modifikasi tambahan untuk mencapai hasil yang lebih tinggi.

• Biaya Komputasi yang Stabil Tapi Tidak Optimal: Meskipun waktu eksekusi stabil, durasi sekitar 285 detik dapat dianggap tinggi jika diaplikasikan pada permasalahan skala besar.

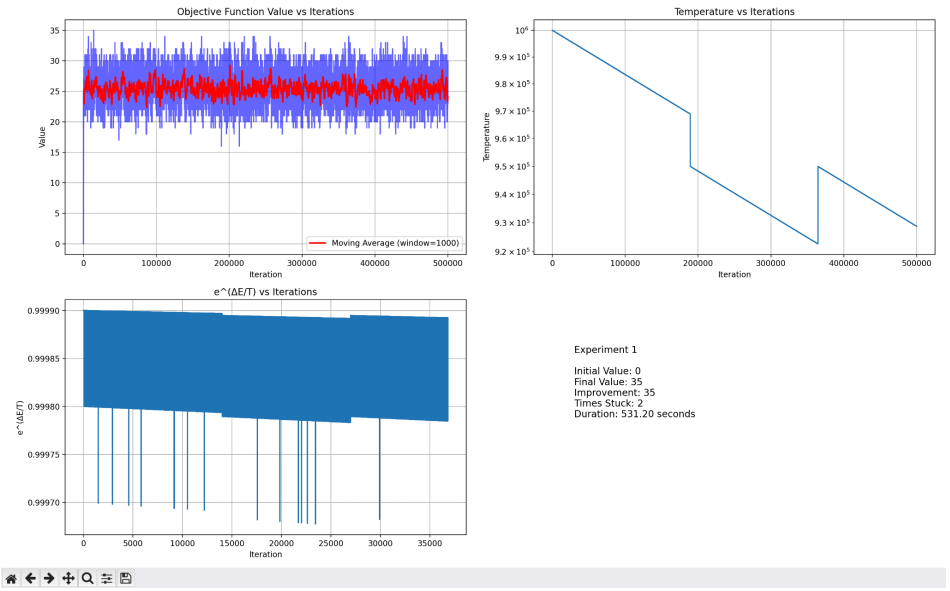
3.1.4.8 Kesimpulan Berdasarkan Data

Berdasarkan data dari ketiga percobaan, algoritma *random restart hill climbing* menunjukkan stabilitas dalam hal durasi dan pencapaian nilai akhir. Dengan rata-rata nilai akhir 49.67, algoritma ini memberikan hasil yang konsisten meskipun masih jauh dari optimum. Metode ini cocok untuk permasalahan yang membutuhkan eksplorasi luas dengan beberapa titik awal acak untuk menghindari jebakan *local optima*, namun perbaikan diperlukan untuk meningkatkan efisiensi dan mencapai hasil yang lebih dekat ke nilai optimal.

3.2. Simulated Annealing

3.2.1 Hasil Eksperimen

3.2.1.1 Eksperimen 1



```
rafidhiyaulhaq@Rafis-MacBook-Air src % python3 simulated_annealing.py

Running experiment 1/3
2024-11-09 13:26:09.152 Python[2178:27066] +[IMKClient subclass]: chose IMKClient_Legacy
2024-11-09 13:26:09.153 Python[2178:27066] +[IMKInputSession subclass]: chose IMKInputSession_Legacy

Initial State:

Current Cube State:

Layer 1:
['105', ' 24', '106', ' 71', ' 34']
['121', ' 97', '119', '116', ' 37']
['107', ' 56', ' 98', '100', ' 78']
[' 46', ' 76', ' 41', ' 61', ' 12']
['  5', ' 15', ' 91', '108', '  2']

Layer 2:
[' 26', ' 31', ' 66', '115', ' 29']
['  3', ' 16', '125', ' 64', ' 18']
[' 13', ' 36', ' 62', ' 17', ' 50']
[' 81', ' 96', ' 28', ' 10', ' 43']
['112', ' 27', ' 87', '113', ' 32']

Layer 3:
[' 51', ' 54', ' 82', ' 60', ' 63']
['109', ' 89', ' 68', ' 22', ' 47']
[' 49', ' 92', '  7', ' 40', '118']
[' 83', ' 69', ' 35', '122', ' 20']
[' 53', ' 94', ' 90', ' 38', '  9']

Layer 4:
[' 42', ' 48', ' 65', ' 70', '  1']
[' 14', '114', '101', ' 86', '123']
[' 99', '  8', ' 21', ' 25', ' 58']
[' 59', ' 73', ' 57', ' 19', ' 44']
[' 79', ' 39', ' 74', '110', '111']
```

```
Layer 5:
[' 52', '102', '104', ' 77', ' 23']
['117', ' 93', '124', ' 33', '  4']
[' 45', ' 30', ' 85', ' 11', '103']
[' 72', ' 95', ' 88', '  6', ' 84']
['120', ' 67', ' 75', ' 55', ' 80']

Current Value: 0
Magic cube not yet solved.

Final State:

Current Cube State:

Layer 1:
[' 84', '103', ' 40', ' 36', '115']
[' 38', ' 55', '125', ' 25', ' 94']
['  4', ' 67', ' 77', ' 46', ' 74']
['124', ' 33', '119', ' 82', ' 15']
[' 65', ' 69', '  6', ' 47', ' 17']

Layer 2:
['107', ' 50', ' 43', ' 10', ' 20']
[' 66', '  1', ' 14', '123', '111']
[' 18', ' 95', ' 75', ' 23', ' 93']
[' 89', ' 70', ' 22', ' 88', ' 83']
[' 27', '112', '104', ' 71', ' 44']

Layer 3:
[' 26', ' 90', ' 31', '101', ' 60']
[' 61', ' 97', '116', '117', ' 54']
[' 99', ' 62', ' 24', '110', '  3']
['108', ' 79', ' 51', ' 72', '102']
[' 35', '  2', ' 56', ' 85', ' 96']

Layer 4:
[' 92', '  8', '  9', ' 32', '  5']
['100', ' 64', ' 19', '121', ' 11']
['113', ' 29', ' 52', ' 42', ' 59']
['  7', '109', '122', ' 86', ' 39']
```

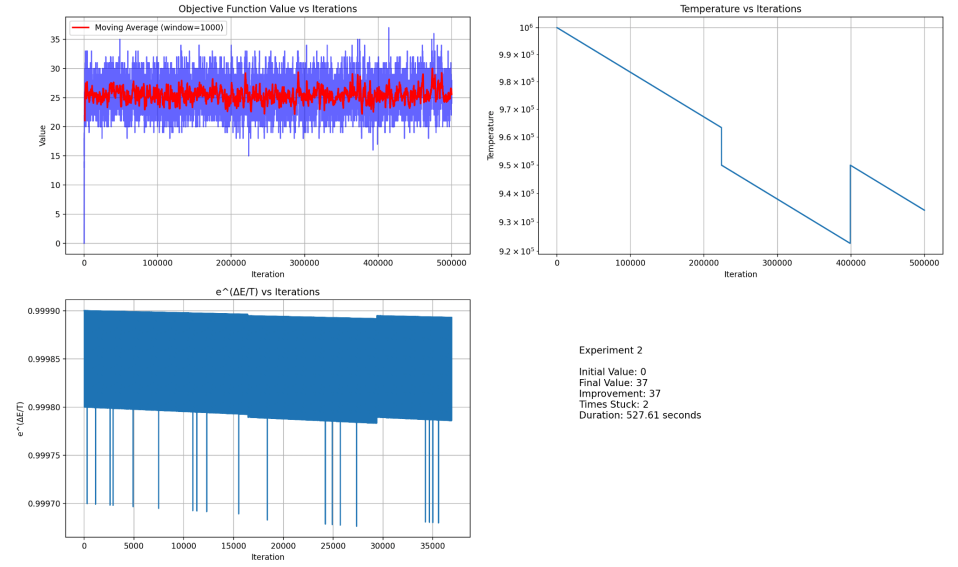
```
Layer 1:
[' 84', '103', ' 40', ' 36', '115']
[' 38', ' 55', '125', ' 25', ' 94']
['  4', ' 67', ' 77', ' 46', ' 74']
['124', ' 33', '119', ' 82', ' 15']
[' 65', ' 69', '  6', ' 47', ' 17']

Layer 2:
['107', ' 50', ' 43', ' 10', ' 20']
[' 66', '  1', ' 14', '123', '111']
[' 18', ' 95', ' 75', ' 23', ' 93']
[' 89', ' 70', ' 22', ' 88', ' 83']
[' 27', '112', '104', ' 71', ' 44']

Layer 3:
[' 26', ' 90', ' 31', '101', ' 60']
[' 61', ' 97', '116', '117', ' 54']
[' 99', ' 62', ' 24', '110', '  3']
['108', ' 79', ' 51', ' 72', '102']
[' 35', '  2', ' 56', ' 85', ' 96']

Layer 4:
[' 92', '  8', '  9', ' 32', '  5']
['100', ' 64', ' 19', '121', ' 11']
['113', ' 29', ' 52', ' 42', ' 59']
[' 7', '109', '122', ' 86', ' 39']
[' 28', '105', ' 16', ' 34', ' 21']

Layer 5:
[' 53', '106', ' 63', ' 73', '118']
[' 91', ' 80', ' 41', ' 13', ' 45']
[' 81', ' 12', ' 87', ' 57', ' 78']
['114', ' 48', ' 68', ' 58', ' 76']
[' 49', '120', ' 98', ' 30', ' 37']

Current Value: 35
Magic cube not yet solved.
```

Hasil eksperimen 1:

- Initial Value: 0
- Final Value: 35

- Improvement: 35
- Times Stuck: 2
- Duration: 531.19 seconds

3.2.1.2 Eksperimen 2




```
Running experiment 2/3

Initial State:

Current Cube State:

Layer 1:
[' 14', '119', ' 39', ' 65', '  8']
['108', ' 87', ' 47', ' 12', ' 97']
[' 26', ' 58', ' 78', '  7', ' 31']
[' 18', ' 17', ' 10', ' 29', ' 79']
['120', ' 85', ' 34', ' 50', ' 35']

Layer 2:
[' 22', ' 23', ' 51', ' 83', '118']
[' 53', ' 99', ' 37', '  4', ' 61']
[' 80', ' 62', ' 73', ' 30', ' 36']
[' 60', '124', ' 94', ' 84', '  3']
[' 28', '  1', '  9', '104', ' 15']

Layer 3:
[' 68', ' 86', ' 88', ' 24', '103']
[' 71', ' 98', ' 57', '  5', ' 59']
[' 49', '123', ' 95', ' 20', ' 75']
['102', '100', ' 40', ' 90', ' 16']
[' 82', ' 70', ' 33', ' 67', ' 69']

Layer 4:
[' 72', '106', ' 25', '116', '107']
[' 38', '110', ' 11', ' 19', ' 54']
[' 66', '114', ' 56', ' 44', ' 74']
[' 81', '121', ' 93', '112', ' 27']
['  6', ' 76', ' 13', ' 45', '122']

Layer 5:
[' 92', ' 48', ' 43', ' 42', ' 89']
['115', '113', ' 55', '101', ' 46']
['109', '111', ' 96', ' 91', ' 63']
[' 32', ' 21', ' 41', ' 52', ' 77']
```

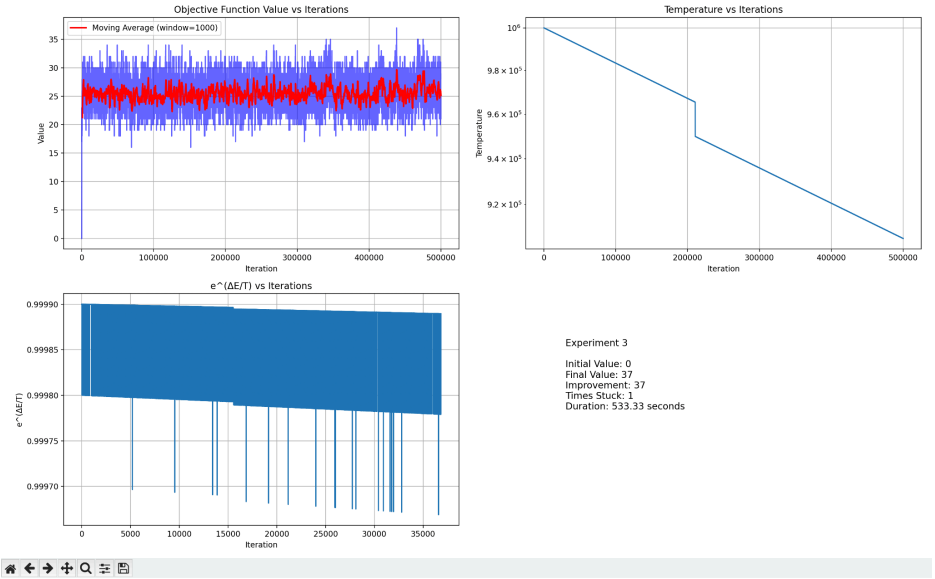
```
Layer 5:
[' 92', ' 48', ' 43', ' 42', ' 89']
['115', '113', ' 55', '101', ' 46']
['109', '111', ' 96', ' 91', ' 63']
[' 32', ' 21', ' 41', ' 52', ' 77']
[' 64', '105', '125', '  2', '117']

Current Value: 0
Magic cube not yet solved.

Final State:

Current Cube State:

Layer 1:
['113', ' 23', '  4', ' 97', ' 78']
[' 44', ' 60', ' 11', ' 46', ' 85']
[' 36', ' 29', '  7', ' 57', ' 37']
[' 89', ' 81', ' 45', '110', ' 72']
['103', '  3', ' 86', '117', ' 25']

Layer 2:
[' 95', '123', ' 20', ' 65', ' 12']
[' 54', ' 53', ' 50', '119', ' 39']
[' 91', ' 90', ' 83', ' 24', ' 52']
[' 27', ' 80', ' 77', ' 69', ' 62']
[' 21', '  9', '  8', ' 49', ' 15']

Layer 3:
[' 74', ' 48', ' 47', ' 19', ' 55']
[' 84', ' 41', ' 16', ' 43', ' 38']
[' 51', ' 70', '108', '109', ' 76']
[' 10', ' 98', ' 58', ' 94', '114']

Layer 4:
[' 68', ' 86', ' 88', ' 24', '103']
[' 71', ' 98', ' 57', '  5', ' 59']
[' 49', '123', ' 95', ' 20', ' 75']
['102', '100', ' 40', ' 90', ' 16']
[' 82', ' 70', ' 33', ' 67', ' 69']
```

```
Layer 2:
[' 95', '123', ' 20', ' 65', ' 12']
[' 54', ' 53', ' 50', '119', ' 39']
[' 91', ' 90', ' 83', ' 24', ' 52']
[' 27', ' 80', ' 77', ' 69', ' 62']
[' 21', '  9', '  8', ' 49', ' 15']

Layer 3:
['  2', ' 42', ' 34', '106', ' 56']
[' 22', '111', ' 68', ' 96', ' 75']
['115', '124', ' 82', ' 13', ' 88']
['101', ' 71', ' 73', '  6', ' 64']
[' 10', ' 98', ' 58', ' 94', '114']

Layer 4:
[' 74', ' 48', ' 47', ' 19', ' 55']
[' 84', ' 41', ' 16', ' 43', ' 38']
[' 51', ' 70', '108', '109', ' 76']
[' 14', ' 17', '  1', ' 26', '104']
[' 92', ' 67', '120', '118', ' 66']

Layer 5:
[' 31', ' 79', '107', ' 93', '  5']
[' 32', ' 63', ' 33', '100', ' 87']
['102', ' 28', ' 35', '112', '105']
[' 99', ' 59', ' 30', '125', ' 18']
['116', '122', ' 40', '121', ' 61']

Current Value: 37
Magic cube not yet solved.
```

Hasil eksperimen 2:

- Initial Value: 0
- Final Value: 37
- Improvement: 37
- Times Stuck: 2
- Duration: 527.61 seconds

3.2.1.3 Eksperimen 3



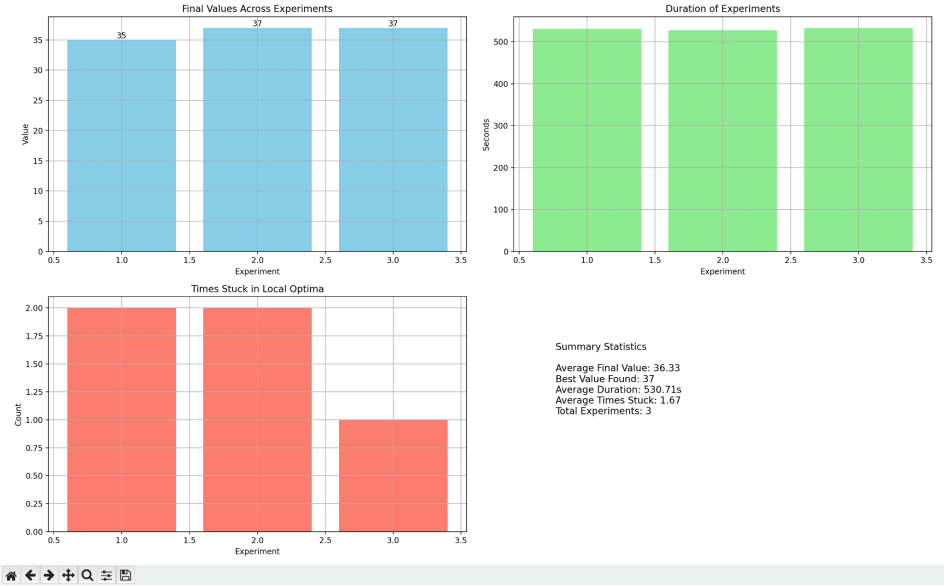
```
Running experiment 3/3

Initial State:

Current Cube State:

Layer 1:
[' 84', ' 40', ' 77', '115', '  8']
['  3', ' 58', ' 49', ' 10', '102']
[' 82', '120', ' 95', ' 99', ' 71']
[' 39', '  4', ' 44', '123', ' 85']
[' 53', '107', ' 65', '101', '118']

Layer 2:
[' 60', '117', '  1', ' 55', ' 67']
[' 72', '113', ' 20', ' 96', ' 32']
['104', ' 61', ' 18', ' 70', '  5']
[' 56', '114', ' 29', ' 16', ' 26']
[' 87', ' 75', '116', ' 22', ' 59']

Layer 3:
[' 69', ' 64', '103', ' 78', ' 37']
[' 76', ' 21', '  9', ' 15', ' 98']
['112', ' 74', ' 81', ' 34', ' 92']
['110', ' 45', '100', ' 80', ' 50']
['125', ' 97', '  7', ' 57', ' 28']

Layer 4:
[' 47', '109', ' 94', ' 11', ' 48']
['121', ' 89', ' 79', ' 41', ' 13']
[' 33', ' 91', ' 43', ' 31', '106']
[' 93', ' 19', ' 42', ' 63', '111']
[' 66', ' 17', ' 73', ' 38', '105']

Layer 5:
[' 36', ' 83', ' 35', ' 25', '124']
[' 52', ' 62', ' 30', '119', ' 54']
['  6', ' 46', ' 68', ' 51', '122']
[' 90', ' 12', ' 88', ' 14', ' 24']
[' 23', '  2', ' 86', '108', ' 27']
```

```
Layer 5:
[' 36', ' 83', ' 35', ' 25', '124']
[' 52', ' 62', ' 30', '119', ' 54']
['  6', ' 46', ' 68', ' 51', '122']
[' 90', ' 12', ' 88', ' 14', ' 24']
[' 23', '  2', ' 86', '108', ' 27']

Current Value: 0
Magic cube not yet solved.

Final State:

Current Cube State:

Layer 1:
[' 79', ' 28', ' 31', '110', ' 81']
['104', ' 38', ' 51', ' 20', ' 74']
[' 99', '118', '  6', ' 46', '115']
[' 54', ' 89', ' 39', ' 95', '116']
['119', ' 25', ' 12', ' 85', ' 97']

Layer 2:
['  8', '103', '125', ' 68', '  5']
[' 49', ' 94', ' 57', ' 88', ' 27']
['113', ' 35', '123', '  2', ' 73']
['  9', ' 32', ' 93', ' 53', '122']
[' 67', '124', ' 10', ' 83', ' 37']

Layer 3:
[' 86', '120', ' 44', ' 18', ' 47']
[' 13', ' 70', '108', ' 90', '  3']
[' 29', ' 52', ' 40', ' 19', '117']
[' 98', ' 78', '112', ' 43', ' 56']
[' 60', '  4', ' 75', '100', ' 76']

Layer 4:
[' 77', '  1', ' 17', ' 48', ' 15']
[' 42', ' 92', '101', ' 87', ' 22']
[' 24', ' 84', ' 66', ' 96', ' 62']
[' 61', ' 36', ' 82', ' 11', ' 72']
['111', '102', ' 16', ' 45', ' 69']

Layer 5:
[' 65', '121', '109', ' 55', ' 41']
['107', ' 21', ' 64', ' 30', ' 63']
[' 59', ' 26', ' 80', ' 23', ' 14']
[' 34', '114', ' 71', ' 58', '106']
[' 50', ' 33', '  7', '105', ' 91']

Current Value: 37
Magic cube not yet solved.
```

Hasil eksperimen 3:

- Initial Value: 0
- Final Value: 37
- Improvement: 37
- Times Stuck: 1
- Duration: 533.33 seconds

3.2.2 Statistik Keseluruhan



Statistik Keseluruhan:

- Average Final Value: 36.33
- Best Value Found: 37
- Average Duration: 530.71 seconds
- Average Times Stuck: 1.67
- Total Experiments: 3

3.2.3 Analisis Performa

3.2.3.1 Konsistensi

- Nilai akhir konsisten di range 35-37
- Durasi konsisten sekitar 530 detik
- Times stuck konsisten rendah (1-2 kali)
- Moving average stabil di sekitar 25

3.2.3.2 Karakteristik Pencarian

A. Temperature Schedule:
   a. Penurunan temperature smooth dari 10^6 ke 9.2×10^5
   b. Reheat terkontrol saat stuck
   c. Cooling rate efektif dalam mengatur eksplorasi vs eksploitasi

B. Objective Function:
    a. Range nilai stabil antara 20-35
    b. Fluktuasi sehat menunjukkan eksplorasi baik
    c. Moving average konsisten menunjukkan stabilitas pencarian

3.2.3.3 Kelebihan

- Konsistensi hasil antar eksperimen
- Times stuck yang rendah
- Eksplorasi yang baik (ditunjukkan oleh fluktuasi nilai)
- Peningkatan nilai yang signifikan dari initial state

3.2.3.4 Keterbatasan

- Durasi eksekusi cukup lama (±9 menit per eksperimen)
- Masih jauh dari nilai optimal (109)
- Membutuhkan resource komputasi yang cukup besar

3.2.3.5 Kesimpulan Eksperimen dengan Simulated Annealing

- Algoritma berhasil menemukan solusi yang cukup baik (35-37) secara konsisten
- Performansi stabil antar eksperimen
- Balance yang baik antara eksplorasi dan eksploitasi
- Reheat mechanism efektif dalam menghindari local optima (times stuck rendah)

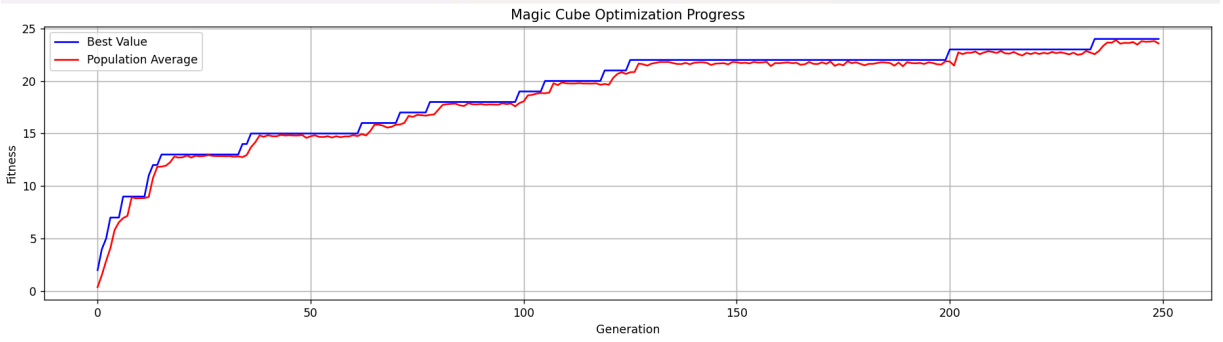
3.3 Genetic Algorithm

3.3.1. 100 Populasi 250 Iterasi

3.3.1.1. Eksperimen 1

```
Layer 1:
[ 51, 112,  11,  44,   5]
[114,  15, 118,  67,  83]
[ 14, 119,  89,  40,  79]
[ 24,  55,  86,  71,  23]
[ 30,  35,  50,   6,  16]

Layer 2:
[ 90,  82,  34,  69,  92]
[ 88, 123,  59, 116,  21]
[ 72,  33,  20,  56,  26]
[ 81, 109,  42,  27, 104]
[ 78,  32,   1,  65,  17]

Layer 3:
[ 12,  74,  57,  46,  52]
[ 66,  84,  53, 102,  63]
[ 73,  95,  41,  58,  60]
[117,  85,  77, 110,  97]
[ 38,  29,  25,  96,  76]

Layer 4:
[ 28,  43,  87,   7,   9]
[ 45,  98,  31,  64, 125]
[107, 103,  36, 111,  61]
[124,  48,  19,  68,  18]
[  4,  49, 120,  10,   2]

Layer 5:
[ 39,  93,  80,  13,  54]
[ 91,  70, 105, 108,  99]
[106, 122,  22, 100,  94]
[ 37,   8,  62, 121,  75]
[ 47,   3, 113, 101, 115]
```

```
Layer 1:
[ 35,  79,  93,  67,  29]
[ 28, 121, 124,  53, 104]
[ 41,  40,  13,  73, 117]
[106, 115,  48, 111,  46]
[105, 104,  80,  51,  89]

Layer 2:
[ 59,  71,  93,  54,  53]
[  2,  39,  70,  34, 100]
[ 55,   3, 106, 121,  30]
[105,  47,  26,  65,  72]
[ 75,   8,  18,  81,  60]

Layer 3:
[ 25,  58,  18,  48,  53]
[ 73, 112,  52,  57,  69]
[ 97, 105,  66,  79,  13]
[  5,  57, 116,   3,  71]
[ 47,  65,  63,  31, 109]

Layer 4:
[ 83,  87,  14,   1,  22]
[ 51, 115,  90,  32,  88]
[ 53,  75, 105,  36,  46]
[ 34,  23,  28, 125,  45]
[ 57,   4,  78, 121,  38]

Layer 5:
[113,  20, 119, 100,  39]
[ 19,  85, 100,   8,  34]
[ 64,   1,  90,  86,  31]
[ 65,  73,  69,  94,  81]
[105,  63, 115,  67,  83]
```

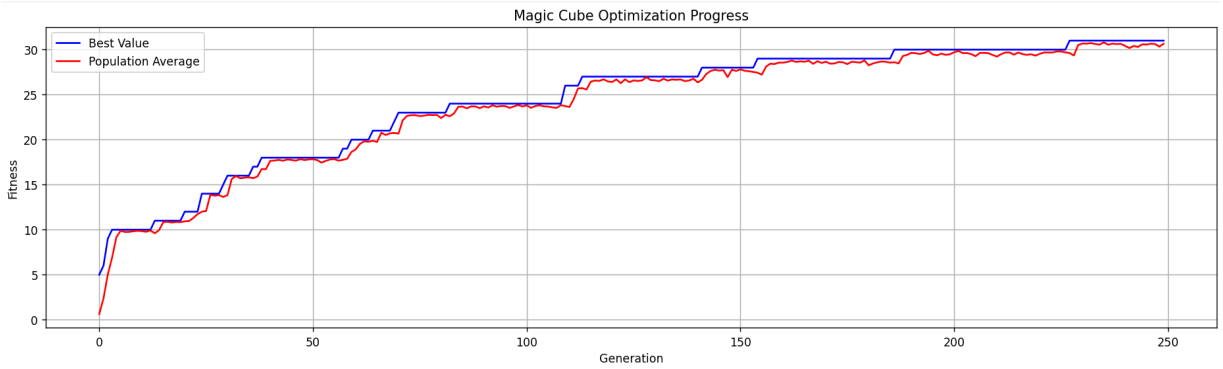


Nilai *objective function* awal: 0
Nilai *objective function* terakhir: 24
Jumlah populasi: 100
Banyak iterasi: 250
Durasi: 19.18 detik

3.3.1.2. Eksperimen 2

```
Layer 1:
[ 55,  42,  84,  21, 112]
[ 90,  16,  18, 109,  40]
[ 11, 107,  53,  46,  20]
[ 78,  39,  54,  25,  61]
[ 95,  37,  62,  81,  59]

Layer 2:
[ 28,  56,  13,  83, 111]
[ 22,   5,   6, 115,  30]
[ 50,  58,  48,  45,  66]
[ 12, 108,  93,  67,  49]
[113,  87,  65, 125, 101]

Layer 3:
[ 68,  76,  75, 124,  60]
[ 29,  77,  32, 116,  86]
[ 52,  99, 114,  51, 104]
[ 26,  89,  23,  96,  27]
[ 92,  57,  97,   1,  24]

Layer 4:
[  8,  88,  33, 123,  63]
[117, 120,  70,  79,  44]
[  3, 103,  10,  36,  64]
[ 41, 102,  71, 121,  82]
[ 80,  35,  38,  15, 106]

Layer 5:
[ 73,  72,  98,  19, 119]
[ 94,   9,  31,  14,   4]
[ 69,  85,  91, 105,   7]
[ 43, 122,  17,   2,  34]
[ 47,  74, 110, 100, 118]
```

```
Layer 1:
[ 51,  35,  37,  77,  75]
[ 14,  61,  34,  21,  90]
[ 87, 102, 118,  83,  84]
[117,  81, 117, 115,  18]
[ 20,  83,  42, 124,  98]

Layer 2:
[ 46,  96,   9,  76,  91]
[ 28, 120,  84,  32,  38]
[ 30,  58,   8,  89,  56]
[ 25,  17, 113,  58, 102]
[ 10,  24,  87,  28,  83]

Layer 3:
[ 89, 114,  36,  96,  85]
[ 67,  33,  63,   6,  22]
[ 46,  29,  38,   4,  45]
[ 62, 120,  28, 105, 113]
[ 66, 124, 102, 124,  50]

Layer 4:
[ 86,  28,  14,  71,  88]
[ 69,  61, 107,  22,  56]
[ 44, 110, 110,  34,  17]
[ 78,  10,   9,  42,  84]
[ 85,  86,  69,  68,  16]

Layer 5:
[ 61,  59,  38, 100,  60]
[ 69,  40,  27,  70, 109]
[ 79,  16,  77, 117, 113]
[ 85,  87, 124,   9, 111]
[ 21,  95,  73,  25, 101]
```

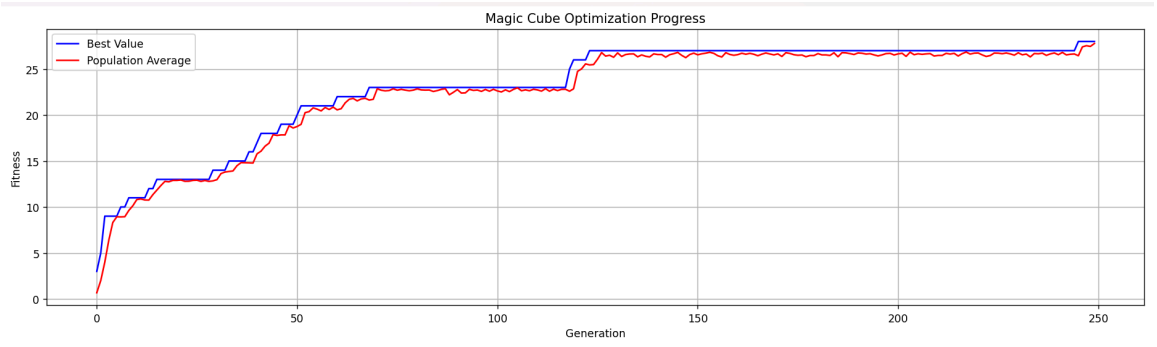


Nilai *objective function* awal: 1
Nilai *objective function* terakhir: 31
Jumlah populasi: 100
Banyak iterasi: 250
Durasi: 20.85 detik

3.3.1.3. Eksperimen 3

```
Layer 1:
[  6,   1, 105,  49, 111]
[ 90,  99,   2,  95, 120]
[ 72,  47,  57,  23,  56]
[110,  93,  37,  53,  84]
[ 20,  74,  50,  32, 103]

Layer 2:
[  7,   5,  86,  80,  30]
[ 68,  67, 106,  10,   8]
[ 17,   3,  75,  88,   9]
[ 78,  40,  43,  46, 101]
[ 59,  52,  27, 121,  26]

Layer 3:
[ 66,  97,  13,  28,  48]
[ 35,  39, 118,  42,  82]
[ 29,  71,  96,  85,  19]
[ 91,  62,  83,  92, 112]
[119,  98,  11, 100, 113]

Layer 4:
[  4,  94,  31, 116,  25]
[ 61,  76,  60, 115,  51]
[ 14,  65,  44,  81, 107]
[ 41, 102,  16, 122,  22]
[124,  63,  54, 123,  79]

Layer 5:
[ 70,  77,  24,  36, 104]
[ 45,  89,  87,  69, 114]
[ 12,  55,  58, 109,  38]
[ 34,  18,  33, 108,  21]
[ 73, 125,  64,  15, 117]
```

```
Layer 1:
[ 63,  23,  27,  89, 113]
[121,  96,  24,   9,  77]
[ 48,  84,  67,  91,  58]
[  2, 107,  90,  71,  49]
[ 19,  84,  97,  42,  18]

Layer 2:
[108,  54,  21, 112,  20]
[116,  26,  97,  83,  71]
[119,  78,  91, 120,  56]
[118,  55,  57,  58,  96]
[ 88, 102,  38,  70,  17]

Layer 3:
[119, 101,  18,  45,  93]
[ 13,  27,   8,  89,  57]
[ 32, 111,  12, 105,  55]
[ 98,  48,  34, 115,  20]
[125,  87,  42,  19,  42]

Layer 4:
[105,  67,  78,  64,  98]
[106,  23,  96,  57,  95]
[ 51,  60,  23,  10,  97]
[ 60,  90,  50,  72, 110]
[ 47,  43, 104,  22,  99]

Layer 5:
[ 41, 122, 101,  15, 103]
[123, 124,  53,  66, 101]
[ 65,  84,  36,  41,  89]
[ 37,  48,  84,  58,  40]
[ 62,  69,  41,  91,  56]
```



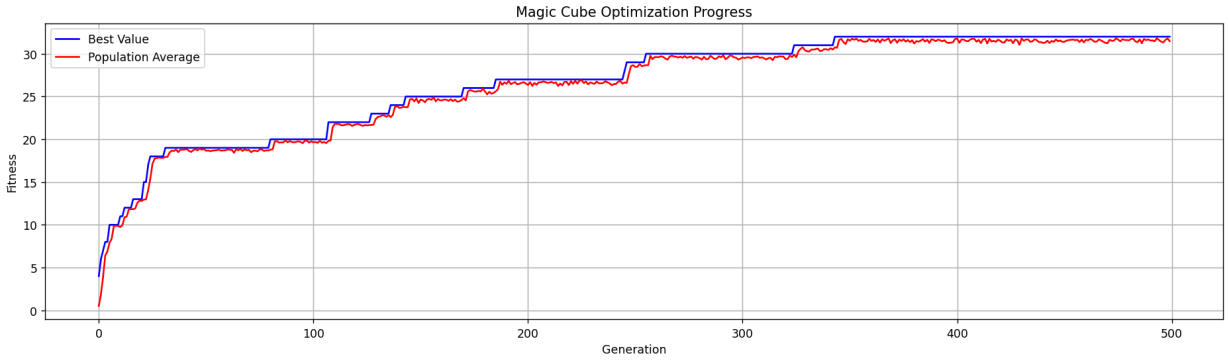
Nilai *objective function* awal: 1
Nilai *objective function* terakhir: 28
Jumlah populasi: 100
Banyak iterasi: 250
Durasi: 20.78 detik

3.3.2. 100 Populasi 500 Iterasi

3.3.2.1. Eksperimen 1

```
Layer 1:
[ 65,  41,  82,  48,  84]
[ 55, 117,  25,  52, 103]
[ 93,  30,  53,  21,  10]
[ 91,   5, 112, 115,  42]
[ 98, 123,  86,  63,  29]

Layer 2:
[ 33,  56,  28,  24,  60]
[ 40,  59, 107,   2,  16]
[ 51,  13,   3,  80,   4]
[ 54, 125,  99,  50,  83]
[ 71,   8, 118,  67, 114]

Layer 3:
[ 81,  96,  66,  89,  26]
[ 22,  64,  92,  76,  70]
[ 20, 121,  47,   9,  39]
[ 32, 102, 111,  88, 100]
[ 35,  94,  19,  15,  43]

Layer 4:
[122,  61,  62,  69,  12]
[ 11, 105,  74,  23,  97]
[ 37,  90,  87,   1,   6]
[110, 106, 113, 104, 109]
[ 31,  34, 101,   7,  73]

Layer 5:
[ 36,  58,  75,  49,  14]
[ 38,  27,  78, 108,  44]
[ 45,  79,  72, 116,  18]
[120,  57,  85,  68, 119]
[ 46,  77,  17,  95, 124]
```

```
Layer 1:
[103,  23,  74,  26,  89]
[ 68,  55, 114,  30,  98]
[ 48,  13,  33,  69,  45]
[ 80,  69,  98,  84, 102]
[ 94,  31,  44, 106,  40]

Layer 2:
[ 60,  59,  69,  23, 105]
[119,  67,  30, 114, 117]
[ 70, 115,  24, 110, 122]
[ 51,  57,  87,  75,   1]
[ 15,  55,  31,  98,  89]

Layer 3:
[ 98,  26,  49,  84,  82]
[  8,  77, 105,  12,  89]
[102,  92,  34,  67,  29]
[ 31,  78,  94,  35, 123]
[109,  16,  33,  81,  71]

Layer 4:
[ 25,  73, 119,  15,  78]
[ 80,  53,  19,  34, 102]
[ 88,  31,  35, 115,  29]
[116,  67,  57, 118,  30]
[ 46, 106, 110, 112,  84]

Layer 5:
[ 29,  53, 114,  30,  89]
[ 40,  83, 114, 120,  25]
[ 95,  87,  35,   4,  61]
[ 37,   1,   5,  93,  59]
[ 70,  24,  78,  68,  75]
```

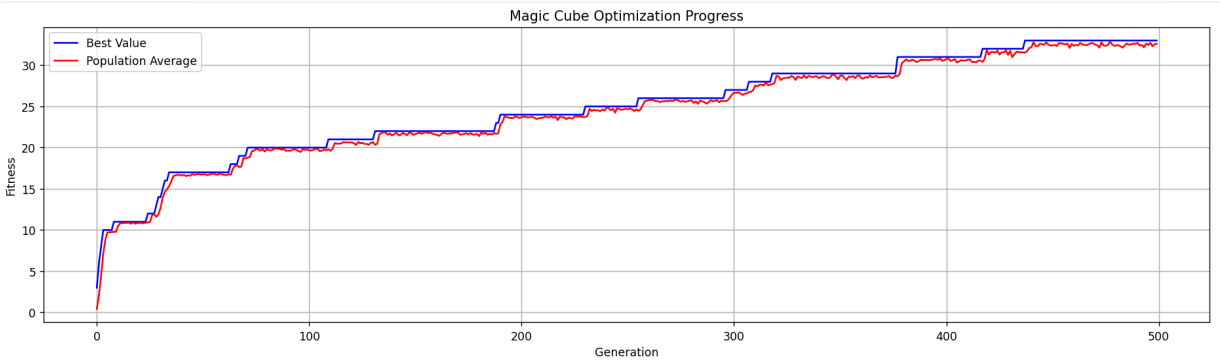


Nilai *objective function* awal: 0
Nilai *objective function* terakhir: 32
Jumlah populasi: 100
Banyak iterasi: 500
Durasi: 42.78 detik

3.3.2.2. Eksperimen 2

```
Layer 1:
[ 90,   3,  58, 122,  56]
[ 20,  44,  82, 102,  33]
[107,  68,   2,  65, 110]
[115,  30, 104, 125,  99]
[ 64,  70, 112,  78,   5]

Layer 2:
[ 14,  45,  26,  57,  16]
[ 89, 113,  23, 114, 117]
[ 47,  92,  77,  42,  12]
[ 43, 118,  96,  29,   6]
[ 53,  48, 109,  40,  19]

Layer 3:
[ 85,  97,  27,  13,  98]
[ 88,  80,  66,  81,  11]
[  9,  79,  75,  10,  31]
[ 93,  32,  36,  94,  67]
[ 63,  95,  17, 121, 108]

Layer 4:
[ 28, 100,  84,  61,  69]
[116,  15,  41,  60,  18]
[ 74,  51,  55,   8,  39]
[106, 111,  50,  76,  52]
[ 86,  72, 124,  49,  71]

Layer 5:
[  4,  34,  25,  24,   7]
[ 21, 123, 119,  83,  35]
[ 59, 120, 105, 101,  73]
[ 54,  91,   1,  37,  87]
[ 62,  22,  46, 103,  38]
```

```
Layer 1:
[ 52,  67,  50,  78,  68]
[ 32,  36,  41, 119,  55]
[124,  65, 107,  19,  58]
[ 61,  85, 111,  24,  91]
[ 23, 122,  94,  75,  96]

Layer 2:
[ 37, 111,  30, 111, 117]
[ 64,  75,  59, 107,  10]
[ 19,  20,  58, 110,  73]
[120, 125,  21,  26,  23]
[ 75,  13,  85, 124, 119]

Layer 3:
[ 17,  78,  67,  77,  23]
[ 45, 121, 106, 107,  50]
[ 40, 123,  92, 102, 114]
[ 58,  11,  27,  56,  43]
[ 82,  44,  23,  76,  29]

Layer 4:
[113,  50, 114,   4,  34]
[ 70,  60,  90,  16,  79]
[119,   2,  96,  63,  35]
[  5,  99, 121,  85,  52]
[ 70,  46,  22,  51,  33]

Layer 5:
[ 56, 101,  54,  45,  59]
[104,   1, 106, 121,  14]
[ 53,   4, 103, 119,  16]
[ 71,  90,   8, 124, 111]
[ 78,  90,  91,  25,  31]
```

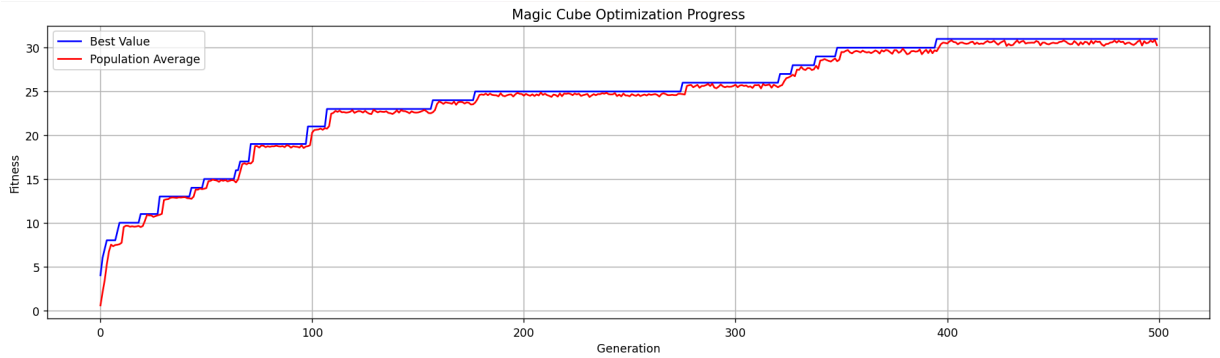


Nilai *objective function* awal: 1
Nilai *objective function* terakhir: 33
Jumlah populasi: 100
Banyak iterasi: 500
Durasi: 38.75 detik

3.3.2.3. Eksperimen 3

```
Layer 1:
[ 25,  27,  50,  24,   4]
[ 34,  37,  17, 102,   6]
[  5, 115,  78,   7,  58]
[ 87,  29,  97, 119, 111]
[ 85, 107,  79, 125,  15]

Layer 2:
[ 82,   9,  76,  19,  35]
[ 42,  71, 117,  98,  72]
[ 75,  32,  18,  99,  63]
[ 64,  94,  66,  49,  21]
[123,  62,  57,  31,  60]

Layer 3:
[ 38,  83, 112,  43,  53]
[105,  84,  33,  16,  93]
[  1,  12,  96,   3, 121]
[ 51,  30,  88, 114, 100]
[124,  54, 104,  56,  36]

Layer 4:
[ 14, 120,  81,   2, 116]
[109, 122,  65,  10,  40]
[ 68,  45,  80,  39, 108]
[ 67,  59,  55,  69,  22]
[ 52,  41, 103,  13,  61]

Layer 5:
[106,  95,  74,   8,  23]
[ 92,  91,  86,  11,  48]
[ 44, 110,  28,  26,  20]
[ 47,  90, 113, 118,  46]
[ 89,  70,  73,  77, 101]
```

```
Layer 1:
[ 66,   1,  27,  32, 124]
[ 12, 119,  63, 102,  19]
[105,  74, 110, 120,  38]
[ 72,  88,  93,  16,  34]
[115,  33,  43,  45,   4]

Layer 2:
[ 94,  15,  73,  60,  25]
[ 54,  42,  48,  53,  76]
[ 59,  81, 125,  43,   7]
[ 46,  65, 116,  80,   2]
[ 47, 122,  80,  94,  44]

Layer 3:
[ 81,  44,  10, 111,  69]
[ 90,  47,  55,  86,  17]
[116,  82,  35,  44,  80]
[105,  32,   5,  51, 122]
[ 93,  33, 115,  23, 101]

Layer 4:
[ 14,  12,   9, 100,  96]
[ 41,  16,  91,  60,  40]
[ 75,   4,  78,  82,  55]
[ 24,  17,  95,  23,  26]
[ 64,  90,   6,  50,  98]

Layer 5:
[109,  88,  71,  94, 124]
[ 21,  91,  58, 103,  90]
[ 61,  74,  23,  94,  12]
[ 68,   6,   6,  22,  24]
[ 12,  56,   5, 121,  70]
```



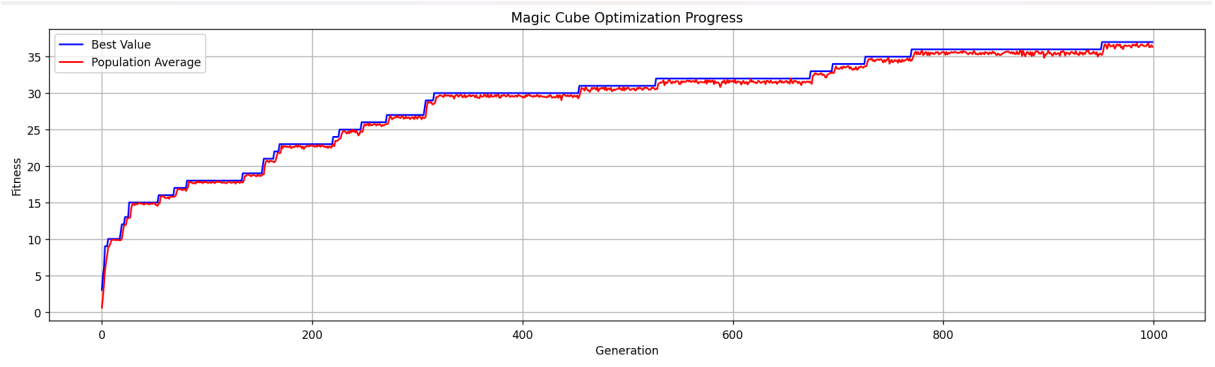
Nilai *objective function* awal: 1
Nilai *objective function* terakhir: 31
Jumlah populasi: 100
Banyak iterasi: 500
Durasi: 40.11 detik

3.3.3. 100 Populasi 1000 Iterasi

3.3.3.1. Eksperimen 1

```
Layer 1:
[ 75,  28, 116,  47,  41]
[106,  71,  66,  45, 111]
[109,  85,  62,  91,  48]
[ 31,   6,  46,  87,  30]
[ 73, 120, 123,  42,  20]

Layer 2:
[ 67,  23,  49,  44, 121]
[ 13,  80,  32,  59,  53]
[ 83, 115, 102, 112,  86]
[ 55,  15,  16,  18,  79]
[101, 107,  98,  64,  33]

Layer 3:
[ 54,  72,  78,  81,  40]
[ 29,  92,  90,  37,  51]
[ 26,  52,  50,  17,  34]
[119,  39,  82,  74,  76]
[108,  63,  11,   4,  95]

Layer 4:
[ 56,  43,  19, 122,  35]
[ 70,  97,  61,  77,  84]
[ 25, 124, 105,   5, 117]
[  1,   3,  93, 103,   2]
[ 69, 110,  12,  99,   9]

Layer 5:
[ 65,  21,  68, 118,  27]
[ 94,  96,  38,   7, 113]
[ 89,  22,  58,  24, 125]
[ 10, 104, 100,   8,  57]
[ 88,  60, 114,  14,  36]
```

```
Layer 1:
[ 75,  78,  28, 123, 116]
[ 25,  63, 124,  84,  19]
[  9,  11,   7,  46,  43]
[117,  26,  76,  16,  80]
[ 82,  55,  70,  46, 118]

Layer 2:
[122,   3, 123,  55, 114]
[ 31, 113, 100,   6,  65]
[ 12,  55, 124,  55,  44]
[108,  29,  99, 107,  49]
[ 42,   1, 124,  92,  56]

Layer 3:
[ 74,  29,   3, 111,  98]
[ 71, 112,  69,   7, 109]
[ 63,  25,  15,   9, 123]
[ 77, 100,  33,  88,  17]
[ 30,  55,  48, 100,  26]

Layer 4:
[  1,  37,  30,  63,  37]
[ 63,  96,  44,  34,  78]
[124, 108,  66,   4,  13]
[102,  69,  65,  41,  38]
[109,   5,  20,  70, 111]

Layer 5:
[ 50,  78,  10, 108,  11]
[125, 106, 123, 123, 102]
[ 94,  82,  30,  55,  54]
[ 22,  48, 116,  88,  41]
[ 52,  78,  36, 108,  41]
```

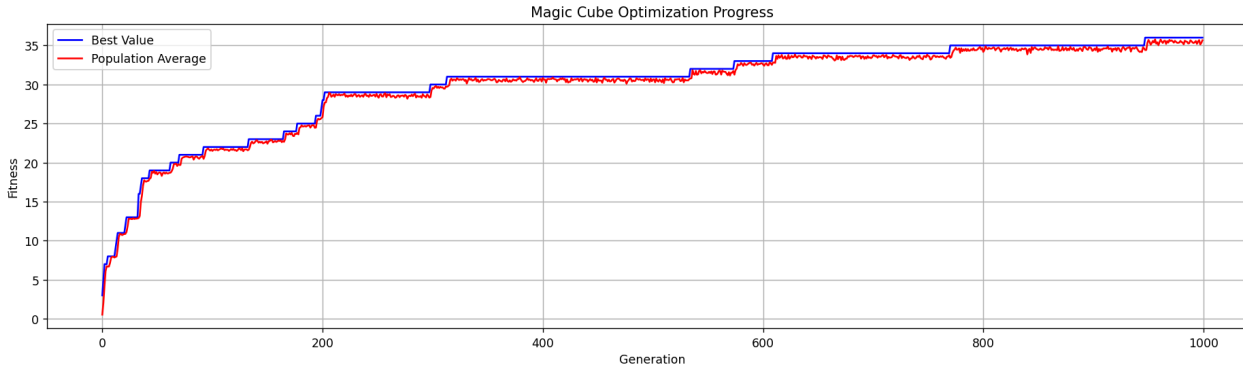


Nilai *objective function* awal: 2
Nilai *objective function* terakhir: 37
Jumlah populasi: 100
Banyak iterasi: 1000
Durasi: 77.75 detik

3.3.3.2. Eksperimen 2

```
Layer 1:
[ 39,  34,  68, 106,  62]
[  5,  69, 100,  44, 112]
[125,  49,  45,  75, 103]
[ 76,  91,  89, 107,  55]
[ 51, 119,  18, 115,  46]

Layer 2:
[120, 121, 114,  42,  36]
[ 32,  78,  48,  24, 108]
[ 52,  26,  82,  87,  22]
[109,  54,  47,  41, 102]
[ 63, 104, 122,  84,  21]

Layer 3:
[ 59,  77,  86,  80,  25]
[ 12,   3,  15,  27,  97]
[ 29,  56, 124,   6, 105]
[ 38,  83,  95,  17,  70]
[ 61,  64,  90,  81,  19]

Layer 4:
[ 88,  94,  96, 117,  50]
[  4,  40,  30,   9,  72]
[ 23,  85,  43,  60,  13]
[ 35, 118, 110, 101,  92]
[ 73,  71, 116,  31,  57]

Layer 5:
[  7,  28,  16,  79,  66]
[ 33,   8,  99, 123, 111]
[ 58,   2,  98,  65,  10]
[113,   1,  20,  67,  74]
[ 53,  14,  11,  37,  93]
```

```
Layer 1:
[ 47,  27,  23,  24,  14]
[ 10, 123,  29,  63,  90]
[ 91,  32,  11,  15,  66]
[ 46,  16,  31,  28, 101]
[ 99, 117,  59, 105, 106]

Layer 2:
[ 41,  66,  79,  71,  49]
[115, 114,  25,  46,  15]
[111,  13,  62,  59,  70]
[ 68,  18,  69,  56,  95]
[ 53, 104,  80,  21,  42]

Layer 3:
[ 68, 119,  50, 122,  15]
[ 71,  15, 123,  81,  25]
[ 48,  95, 109,  20,  43]
[ 86,  68,  98,  17,  46]
[ 42,  69,  13, 107,  87]

Layer 4:
[ 81,  11, 125,  74,   1]
[ 26,  41,  15,  94, 103]
[ 27,  27,  48,  31, 111]
[  4,  56,  44,  58,  13]
[116,  22,  83,  65,  87]

Layer 5:
[ 88,  49,  38,  24,  65]
[ 93,  95,  11, 110,  95]
[ 54,  84,  20,  76,  81]
[ 14, 115, 100,  26,  60]
[  5,   3, 119,  17,  86]
```

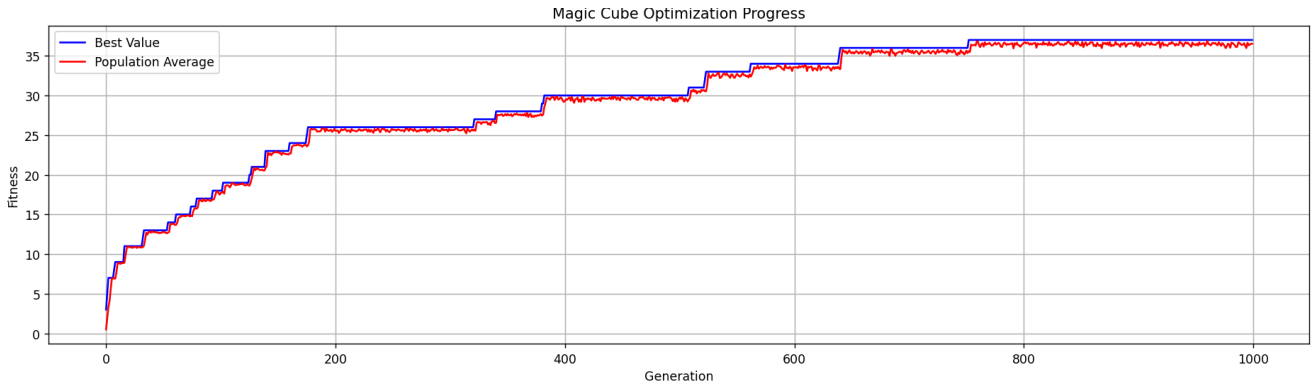


Nilai *objective function* awal: 1
Nilai *objective function* terakhir: 36
Jumlah populasi: 100
Banyak iterasi: 1000
Durasi: 80.01 detik

3.3.3.3. Eksperimen 3

```
Layer 1:
[ 75, 118,  16,  66,  23]
[122, 125, 124, 103,  74]
[ 14, 110,  62,  68, 117]
[ 93, 119,  64,  40,  67]
[ 12,   7,  37, 121,  15]

Layer 2:
[ 36,  59,  65,  98, 113]
[105,  45, 114,   3,  21]
[ 78, 106,  69, 107,  73]
[  6,  82,  92,  13,   8]
[ 70, 100, 102,  47,  63]

Layer 3:
[ 90, 109,  22,  94, 104]
[ 29,  27,  88,   4,  35]
[ 44,  24,  33,  76,  38]
[101, 112,  42,  53,  57]
[123,  17,  84, 116,  34]

Layer 4:
[108,   1,  31,  60,  51]
[ 50,  83,  49,  11,  46]
[ 86,  26,  96,  58,  41]
[ 28, 111,  80,  39, 120]
[ 56,  18,   2,  61,  32]

Layer 5:
[ 43,  97,  91, 115,  89]
[ 85,  55,  30,  95,  25]
[ 99,  79,  71,   9,  77]
[ 81,  48,  52,  10,  19]
[ 72,   5,  20,  54,  87]
```

```
Layer 1:
[ 19,  77, 119,  27,  86]
[ 62,  84,  24,  32,  42]
[  4,  50, 122,  39, 100]
[ 63,  53,  41,  33,  30]
[ 22,  43,   9,  34,  57]

Layer 2:
[ 43, 101,   7,  31,  59]
[ 86,  41,  74,  53, 108]
[ 30,  81,  84, 118,   2]
[ 80,  69, 125,  15,  26]
[ 50, 125,  25,  98, 117]

Layer 3:
[ 38,  26, 119,  52,  80]
[ 66,  37, 123,  43,  43]
[ 67,  86,  81, 100,  29]
[ 57,  41,  59,  17,  43]
[ 70, 125,  78, 103,  55]

Layer 4:
[ 83,   2, 115,  93,  78]
[  4,  36,  24,  28,  81]
[106,  31,  18,  90,  70]
[  5,  93,  93,  55,  48]
[ 98,  87,  43,  49,  38]

Layer 5:
[ 34,  93,   4,  28, 105]
[ 97, 109,  70,  13,  41]
[108,  36,  16,  41, 114]
[ 34,  96,  56, 115,  14]
[ 85,  90,  84,  38,  41]
```



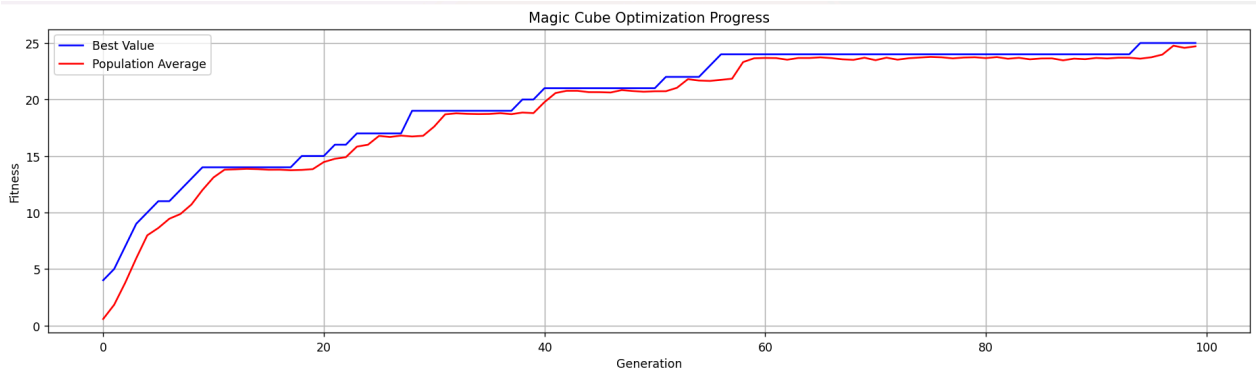
Nilai *objective function* awal: 0
Nilai *objective function* terakhir: 37
Jumlah populasi: 100
Banyak iterasi: 1000
Durasi: 78.91 detik

3.3.4. 250 Populasi 100 Iterasi

3.3.4.1. Eksperimen 1

```
Layer 1:
[102,  62,  67,  87,  81]
[ 59,  75,  82,  22,  13]
[ 80,  88, 111, 124,   5]
[  3,  89,  39,  69, 125]
[ 40,  16,  78,  20,  24]

Layer 2:
[ 55,  50,  92, 105,  70]
[ 29,  57,  93, 108,  66]
[ 98, 122,  30,  26, 104]
[ 95, 121,  33, 119,  84]
[ 79,  49,  64, 123, 110]

Layer 3:
[ 44,  43, 114,  74,  51]
[ 14,   7,  68, 118,  86]
[ 35,  45,   9,  97, 103]
[112,  38,  61,  23, 101]
[ 12,  17, 116,  41,  90]

Layer 4:
[ 65, 109,  37,  46,  56]
[ 42,  96,  19, 115,  10]
[106,  72,  36,  48,  63]
[ 94,  53,  77, 117,   2]
[ 99,  47,  21,  25,   1]

Layer 5:
[ 11,  27,   4,  18,  32]
[ 60,  54,  52, 107,  91]
[ 34,  28,  71,  31,  83]
[  6,  15,  58, 100,  73]
[120,  85,  76,   8, 113]
```

```
Layer 1:
[121,   7,  97, 100,  21]
[ 69,  72,  47,  51,  35]
[ 39,  75,  83,  54,  17]
[ 18,  92, 110,  13,  82]
[ 68,  96,  26,  49,  94]

Layer 2:
[ 32,  62,   2, 104,  83]
[ 41,  46,  34,  24,  37]
[ 75, 124, 120,  65,  81]
[ 50,  70, 104,   6,  85]
[ 55,  16,  90,  43, 111]

Layer 3:
[ 12,  68,  36,  74, 125]
[ 77,  94,  41,  57,  46]
[ 34,  19,  70,   3, 112]
[116,  15,  49, 101,   8]
[ 88,  64,  99,  40,  24]

Layer 4:
[120,  19,  32,  48,  96]
[ 44,   1,  14,  23,  26]
[ 93,  41,  87,  21,  27]
[ 30, 107,  15,  99,  64]
[  2,  46,   3,  17,   8]

Layer 5:
[  6,  16,  86,  22,  48]
[ 30,  51,  71,  72,  47]
[ 99,  18,  55,  82,  61]
[117,  47,  68,  24, 100]
[ 93,  10,  90, 123,  59]
```

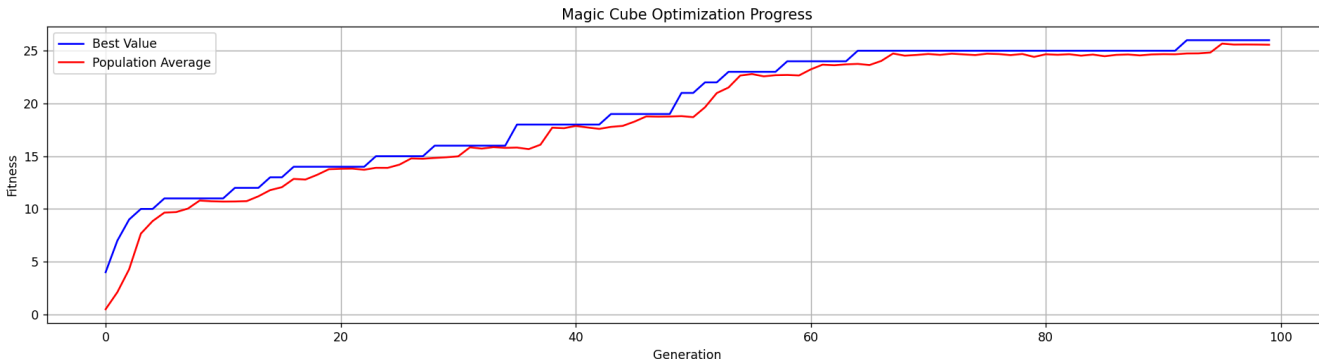


Nilai *objective function* awal: 2
Nilai *objective function* terakhir: 25
Jumlah populasi: 250
Banyak iterasi: 100
Durasi: 25.03 detik

3.3.4.2. Eksperimen 2

```
Layer 1:
[105,  80,  45,  64,  48]
[  6,  27,  44,  65,  35]
[120,  84,  54,  36, 108]
[ 31,  32,  43,  90,  42]
[ 26,   1, 112,  22, 104]

Layer 2:
[ 46,  47,   8,  15, 102]
[  7,  67,  57,  86,  83]
[ 70,  82, 121, 109,  12]
[ 56,  93, 107, 114, 117]
[122, 106,  53,  71,   3]

Layer 3:
[111,  89,  91,  55,  50]
[ 81, 119,  99, 124,  20]
[125,  96,  17,  73,  98]
[ 59,  77,  94,  40,  34]
[ 11, 115,  79,  97, 118]

Layer 4:
[100, 116,  13,  75,  24]
[ 58,  14,  66,  21,  30]
[ 16,  76,  33,  85,  18]
[ 74, 113,  37,  78,  28]
[  4,  38,   2,  62,  19]

Layer 5:
[ 95,  92,  23,  25,  29]
[ 61,   9,  10, 110,  52]
[ 69,  88, 101,  72,  63]
[ 51, 123,  87,   5,  68]
[ 41,  39,  49,  60, 103]
```

```
Layer 1:
[  8,  13,  66,  95, 116]
[  7,  12,  41,  87, 114]
[ 60, 120,  74,  11,  38]
[ 29,  32, 107, 121,   8]
[  6,  37,  50,   1, 100]

Layer 2:
[ 57, 100,  12,  71,  68]
[ 26, 110,  55,  50,  23]
[108,  24,   9,  63,  83]
[ 44, 119,  20,  41, 106]
[ 80, 115,  18,  59,  43]

Layer 3:
[ 46, 121,  13, 117,  42]
[ 36,  19,  59,  17,  44]
[ 32,   6, 103,   2,  27]
[ 71,  84, 112,  37, 107]
[ 69, 115,  28,  29, 110]

Layer 4:
[112,  76,  58,  19,  33]
[ 15,  63,  46, 100,  73]
[ 98, 113,  46,  62,  67]
[ 43,  28,  14,  64,  32]
[108,  29,  13, 106,  59]

Layer 5:
[  8,   5,  45, 112, 118]
[  1, 124,  47,  78,  61]
[ 86,  58,  67, 114,   1]
[ 54,  49,  62, 116,  34]
[  3,  19,  94, 120,  60]
```

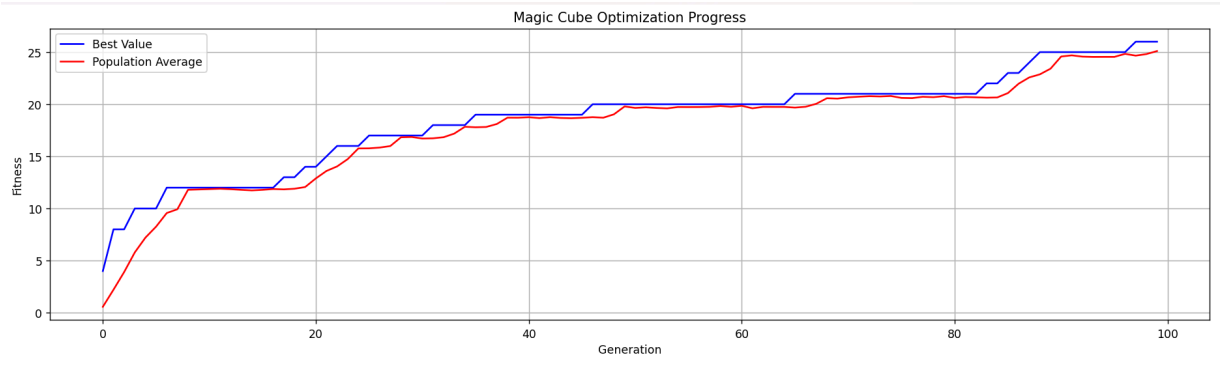


Nilai *objective function* awal: 1
Nilai *objective function* terakhir: 26
Jumlah populasi: 250
Banyak iterasi: 100
Durasi: 23.77 detik

3.3.4.3. Eksperimen 3

```
Layer 1:
[103, 113,  82,  67,   8]
[ 17,  53,  46,  25,  85]
[ 87,  37,  83,  40,  21]
[ 33,  69,  71,  26,  36]
[ 18,  15,  42,  34,  91]

Layer 2:
[ 89, 117,  20,  68,  39]
[  5, 105,  38,  45, 125]
[109,  49, 124,  61, 110]
[ 29,  77,  24,  66, 100]
[111,  12, 118,  97, 116]

Layer 3:
[114,  92,  64, 120,  74]
[ 10,  99,  62,  81,  31]
[ 44,  80,   1,  79, 108]
[ 23,  78,  98,  63,   6]
[ 50,  41,   9,  60,   2]

Layer 4:
[ 70,  48,  43,  16,  75]
[ 28,  13,  56, 106,  94]
[ 47,  35, 119, 123,  58]
[121,   3,  54,  55,  22]
[ 95,  32,  76,  14, 122]

Layer 5:
[104,   4,  88,  19,  65]
[ 84, 115,  72,  59,  96]
[ 52, 102,  30,  11,  86]
[  7,  27, 101, 107,  73]
[ 93,  51,  57,  90, 112]
```

```
Layer 1:
[ 98,   3,  24,   9,  24]
[111,  85,  54,  42,  23]
[ 22,  88, 122, 116,  87]
[ 33,  19, 103,  95,  65]
[108,  90,  12,  55, 116]

Layer 2:
[117,  51,  44,  94, 108]
[ 83,  43, 124,  87,  81]
[ 63,  90,  12,  71,  79]
[ 38, 112,  74,   6,  85]
[ 14,  19,  89,  37, 121]

Layer 3:
[ 24,  40,  76,  23,  87]
[ 19,  48,  43,  58,  39]
[109,  81,  83,  17, 100]
[ 71,  24,  12,  98, 110]
[ 63,  41, 101,  54,  62]

Layer 4:
[  1, 104,  96,  22,  92]
[119,  39,  15,  10,   6]
[ 23,  47,  69,  11,  68]
[ 83,  43,  69,  48,  76]
[  7,  82,  68,  56,  24]

Layer 5:
[101,  26,  37,  67,  84]
[ 95,  10,  88,  74, 104]
[ 98,  42,  90, 100,  20]
[110,  36,  56,  71,  64]
[ 94, 105,  45,   1,  43]
```



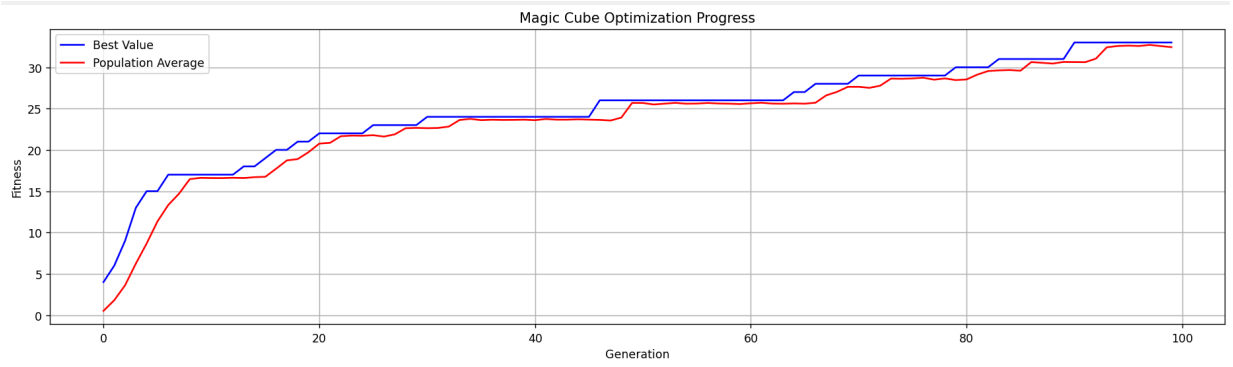
Nilai *objective function* awal: 1
Nilai *objective function* terakhir: 26
Jumlah populasi: 250
Banyak iterasi: 100
Durasi: 24.13

3.3.5. 500 Populasi 100 Iterasi

3.3.5.1. Eksperimen 1

```
Layer 1:
[ 26, 110,  17,  71,  32]
[109,  16,  79,  61,  49]
[ 67, 116, 106,  13,  12]
[101,  25,   2,  75,  62]
[ 66,  39, 115,  97,  95]

Layer 2:
[ 56, 117,  38,  40, 122]
[ 99,  68,  20,  15,  74]
[ 65,  37, 124,  52,   1]
[  3,   5,  81, 112,  14]
[ 60,  73,  93,  44,  70]

Layer 3:
[107,  63,  77,  98,  55]
[102,  48,  53, 120, 108]
[ 24,  41,   8,  87, 104]
[113,  28,  57,  82,  11]
[105,  27,  31,   9,   6]

Layer 4:
[ 22,  78, 114,  33,  58]
[ 46, 118,  36,  30,  91]
[ 92,  94,  43, 121, 103]
[ 45,  72,   4,  96, 100]
[ 51,  88,  19,  42,  89]

Layer 5:
[ 83,  47,  18,  69,  10]
[ 59,  84, 111,  34,  85]
[ 35,  21, 125,  90, 119]
[ 86,  76,  50, 123,  54]
[ 80,  23,   7,  29,  64]
```

```
Layer 1:
[  5, 106,  97, 103,   4]
[ 93, 118,  68,  84,  42]
[ 83,  88,  92,  79,  81]
[121, 122,  50,  83,  70]
[ 13,  20,  74, 109,  17]

Layer 2:
[ 31,  66,  27, 116,  75]
[  3,  88,  82,  44,   8]
[ 10,  77,  39,  15,  71]
[110, 119,  63,  86, 117]
[ 38,  20, 104,  54,  71]

Layer 3:
[ 53, 108, 117,  47,  70]
[ 55,  79,  23,  45,  35]
[ 78,  19, 125,  26,  87]
[121,  40,  46,  28,  93]
[ 35,  72,  82,  42,  30]

Layer 4:
[ 82, 100, 125,  47, 110]
[ 83,  96,  55,  29,  85]
[ 94,   2,  44,  68, 121]
[ 76, 119, 116,  90,  48]
[ 13,  35,  61,  81,   3]

Layer 5:
[123,  34, 124,   5,  29]
[110,  74,  11,  89,  31]
[ 92, 114,  24,  14,  35]
[  2,  65,  40,   3,  95]
[108,  88,  50,  21,  91]
```

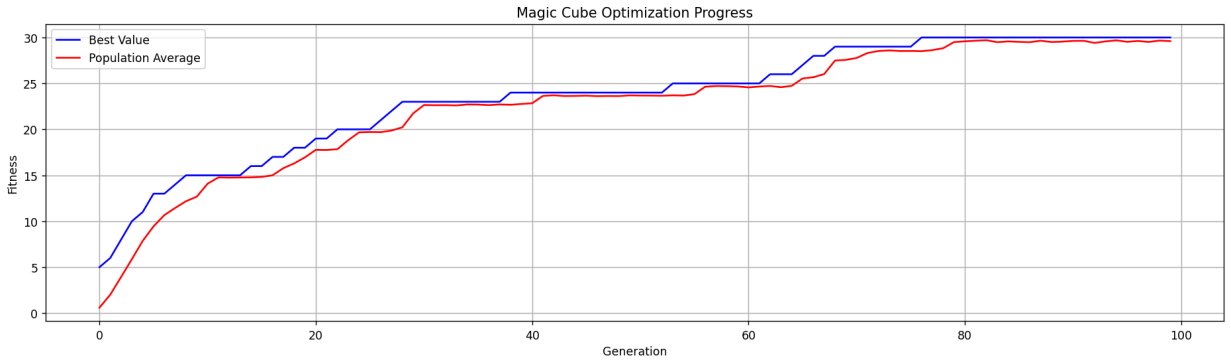


Nilai *objective function* awal: 0
Nilai *objective function* terakhir: 33
Jumlah populasi: 500
Banyak iterasi: 100
Durasi: 58.66 detik

3.3.5.2. Eksperimen 2

```
Layer 1:
[ 79,   9,  49,  18,  38]
[ 29,  94, 111,  63,  81]
[ 25, 101,  68,  52,  21]
[ 12,  78,  77,  17, 102]
[110,  73,  34,  39,  71]

Layer 2:
[ 47, 125,  89, 123,  93]
[ 26,  96,  48,  57, 122]
[120,  72,  82,  86,   7]
[  1,  80,  85, 114,  91]
[121,  64,  97, 104, 112]

Layer 3:
[117,  53,  65,  54,   8]
[113, 106,  11,  30,   5]
[ 31,  32,  99,  40,  16]
[ 92, 116,  19,  95,  24]
[103,  33, 107,   6, 109]

Layer 4:
[ 42, 118,  43,  90,  69]
[ 27,   4,  66,  60,  58]
[ 88,  67, 124,  37,  84]
[100,  51,  13,  10,   3]
[ 55,  56, 115,  62,   2]

Layer 5:
[ 22,  15,  98,  35,  70]
[ 87,  59,  23,  28,  20]
[105,  44,  83, 108,  50]
[119,  41,  75,  46,  45]
[ 76,  61,  74,  36,  14]
```

```
Layer 1:
[102,  19,  79,  22,  93]
[ 81, 125,  43,  38,   2]
[105,  18,  20,  32, 119]
[ 25,  69, 115,  15,  91]
[ 95,   6,  67,  94,  53]

Layer 2:
[ 85, 113,  72,   8,  37]
[ 52,  47,  16,  92, 108]
[ 22, 125,  45,  94,  81]
[105,  64,  65,  32,  75]
[ 70,  10, 120,   2, 106]

Layer 3:
[ 50, 117, 101,  65,  92]
[ 45,  12,  20,  53,  27]
[100, 101, 116,  74,  77]
[ 90,  48, 110,  16,  11]
[  6,  37, 105,  31,  68]

Layer 4:
[ 32,   7, 101, 102, 107]
[ 52,   2,  78,  92,  91]
[  4,  93,  42,  82,  94]
[ 28,  50,   1,  14,  20]
[ 24, 119,  81,  25,  12]

Layer 5:
[ 46,  43,  14, 118,  94]
[  6,  26,  15,  20,  52]
[ 84,  13,  94,  45,  53]
[ 67,   6, 113,  27, 102]
[ 30,  51,   7, 105, 122]
```

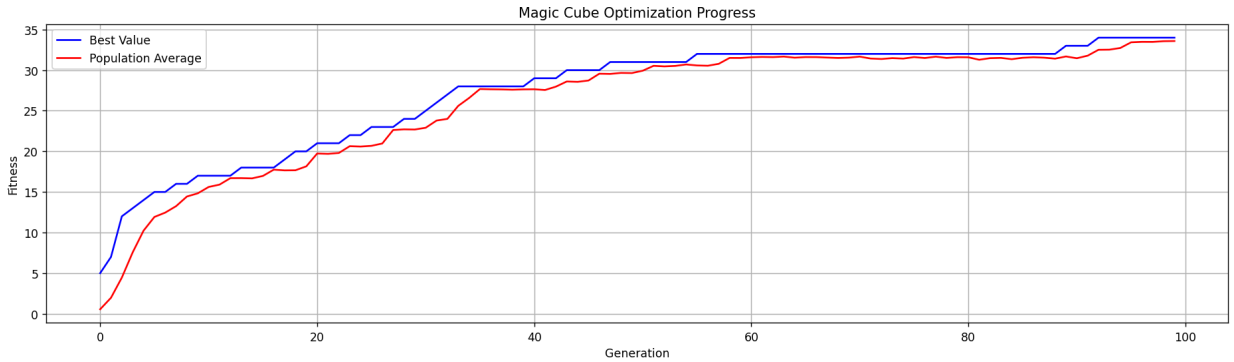


Nilai *objective function* awal: 1
Nilai *objective function* terakhir: 30
Jumlah populasi: 500
Banyak iterasi: 100
Durasi: 57.74 detik

3.3.5.3. Eksperimen 3

```
Layer 1:
[102,   4,  47,  12, 100]
[ 17,  97, 106,  68,  83]
[124,   3, 105, 108,  92]
[ 52,  79, 122,  55,  36]
[  9, 103,  99,  73,  65]

Layer 2:
[ 59,  24,  31,  53,  27]
[110,  80,  41, 111,  57]
[ 70,  77,  66, 121,  46]
[ 13, 125,  93,  39, 119]
[ 35,  76,  21,  10,  14]

Layer 3:
[ 32, 114,  11,  87,  62]
[  2,   8,  28,  49,  96]
[ 37,  60, 116,  40,  90]
[ 86,   5,  91,  72,  50]
[104,  82,  44,  58,  26]

Layer 4:
[ 19,   7,  69,  78,  95]
[ 84, 123, 101,  85,  33]
[120,  20, 117,   1,  75]
[ 45,   6,  42,  22, 109]
[ 38,  16,  23,  81,  34]

Layer 5:
[ 43,  89,  74, 118, 112]
[ 94,  67,  61,  54,  18]
[113,  88,  30,  15,  63]
[ 25, 107, 115,  56,  48]
[ 29,  64,  71,  98,  51]
```

```
Layer 1:
[  2, 114,  75,  71,  13]
[ 63,  15,  34,  28,  47]
[ 33,  73, 106,   7,  74]
[ 24,  70,  58,  99,  19]
[  8,  53, 115,  46,  93]

Layer 2:
[ 87,  23,  33, 117,   5]
[ 59,   6, 120,  99,  31]
[ 76, 114, 118,   7,  47]
[ 74,  41, 111,  69,  58]
[ 19,  63,  73,  23,  35]

Layer 3:
[ 16,  13,  88,  28, 112]
[  6, 110,  59,  53,   8]
[ 24,  15,  64,  32,  95]
[ 79,  84,  34,  98,  73]
[  2,  93,  89, 104,  27]

Layer 4:
[  8,  87,  23,  52,  99]
[ 23, 124,  17,  55,  96]
[123,  69,  12,  16, 112]
[100, 111,  84, 103,  15]
[ 38,  84, 101,  89,  68]

Layer 5:
[ 10,  34,  21,  41, 105]
[ 44,  59,  94, 103,  39]
[101,  44,  56, 120,  66]
[ 38,   9,  28,  86, 100]
[ 42,  22, 116,   5, 104]
```



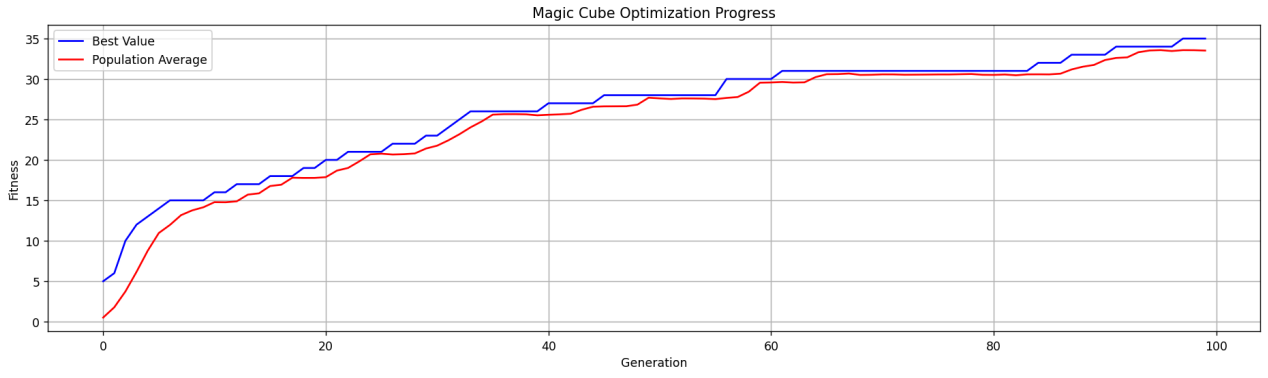
Nilai *objective function* awal: 2
Nilai *objective function* terakhir: 34
Jumlah populasi: 500
Banyak iterasi: 100
Durasi: 60.19 detik

3.3.6. 1000 Populasi 100 Iterasi

3.3.6.1. Eksperimen 1

```
Layer 1:
[ 51, 124, 106,  87,  47]
[ 98,  74,  67, 112, 109]
[ 62,  34, 122, 116,  88]
[ 27,  29,  94, 108,  12]
[ 35, 117,  25,  11,   4]

Layer 2:
[ 73,  79,  19, 105,  69]
[  3,  26,  68, 121,  96]
[ 89,  63,  86,  71, 120]
[102,  77,  72, 115,  10]
[ 53,  17,  85, 113,  22]

Layer 3:
[ 32, 101,   6,  57, 104]
[ 50,  81,  28,   8, 111]
[ 75,  16,  92,  43,  37]
[ 38,   2, 103,  58,  95]
[ 23,  97, 119,  42,  84]

Layer 4:
[ 48,   5,  55,   7, 100]
[ 99,  49,  70,  46,  61]
[ 13,  83,  40,  78,   9]
[118,  56,  15,  91,  54]
[ 76,   1,  24, 123,  59]

Layer 5:
[ 39,  18,  41,  30, 114]
[107,  14,  31,  36,  20]
[ 44,  33,  65, 125,  93]
[ 21,  90,  64,  66,  82]
[ 80,  52,  60,  45, 110]
```

```
Layer 1:
[ 86,  90,  23,  19, 109]
[108, 123,  46,  13,  77]
[  6,  62,  86,  67,  47]
[ 17,   9,  66,  13,  89]
[ 98, 125,  55, 111,   7]

Layer 2:
[ 20,  56,  72, 114,  65]
[ 34,  36,  46, 101,   5]
[ 67,  31,  14,  98,  37]
[110,  51,  83,  91,  97]
[ 84,  43, 100,  70, 111]

Layer 3:
[ 52, 125,  17,  71,  50]
[ 45,  62,  69, 115,  42]
[ 88,  68,  65,  12,  82]
[105,  60,  10, 120,  20]
[ 25,  26, 110,  13, 102]

Layer 4:
[120,  19,  61,  45,  70]
[ 20,  72,  35,  67,  57]
[ 49,  89,  26,  64, 102]
[ 80, 106,  66, 123,  41]
[ 46,  16, 110,  43, 125]

Layer 5:
[ 32,  97,  86,  66,  34]
[ 87,  22, 119,  67,  20]
[125,  65,  76,  66,  46]
[ 19,  26,  90, 122, 114]
[ 52, 105, 125,  62,  63]
```

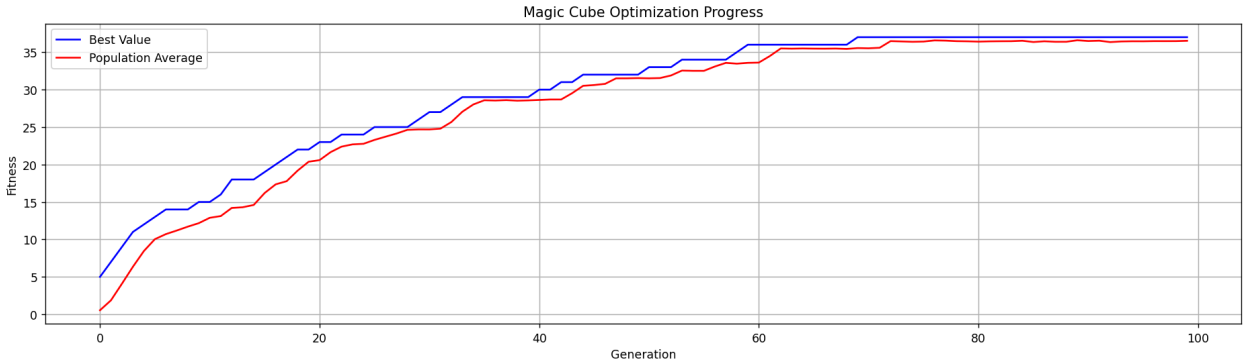


Nilai *objective function* awal: 0
Nilai *objective function* terakhir: 35
Jumlah populasi: 1000
Banyak iterasi: 100
Durasi: 155.28

3.3.6.2. Eksperimen 2

```
Layer 1:
[105,  51,  47,  94,  97]
[  7,  43, 111,  99,  92]
[123,  76,  31,  80,   2]
[  5, 104,  62,  11,  70]
[ 15, 109,   9,  93, 124]

Layer 2:
[ 54,  32,  61, 116,  48]
[ 19,  85, 120,  81,  17]
[ 20,  67,  18,  14,  22]
[100,  29,   3,  57,  84]
[122,  21, 125,  40,  44]

Layer 3:
[ 37,   4, 110,  16,  66]
[ 78,  68, 118,  90,  12]
[ 52,  23,  65, 101, 103]
[ 60,  77, 102,  79,  13]
[113,  73, 117,   8,  86]

Layer 4:
[ 74,  42,  34,  10, 107]
[ 64,   1,  45,  59,  53]
[ 69,  89,  83,  24,  25]
[ 33,  98,  26,  49,  56]
[ 50, 119, 112, 108,   6]

Layer 5:
[ 95,  91,  58,  38,  96]
[ 75,  71, 115,  27,  72]
[ 55,  63,  46, 121, 106]
[ 82,  41,  36,  30, 114]
[ 87,  28,  88,  39,  35]
```

```
Layer 1:
[ 55, 112, 123, 122,  99]
[ 16,  43,  96,  33, 119]
[ 79, 104,  76,  22,  34]
[ 33,  41,  32,  39,  56]
[ 66,  15,  29,  81, 102]

Layer 2:
[ 64,  77,  46,  70,  22]
[101,   4, 100, 121,  30]
[100, 122,  75,   6,  85]
[ 32,  10,  52,  65,  81]
[ 87, 102, 119,  67, 107]

Layer 3:
[112, 123,  73,  29,  56]
[ 36,  25, 115,  75,  64]
[ 31,  39,  28, 108,  53]
[ 95,  80,  45,  36,  22]
[ 76,  48,  54,  57, 114]

Layer 4:
[ 91,  14,   3,  11,  43]
[ 39,  33,  54,   8,  56]
[ 93,  64,  93,  46, 107]
[119,  83,  60,  21,  32]
[ 88,  21,  87,  59,  77]

Layer 5:
[  5,  36,  70,  83, 121]
[123,  28,  65,  29,  46]
[ 12, 119,  59, 109,  16]
[112,  93,   2, 125,  34]
[ 71,  25, 119,  51,  98]
```

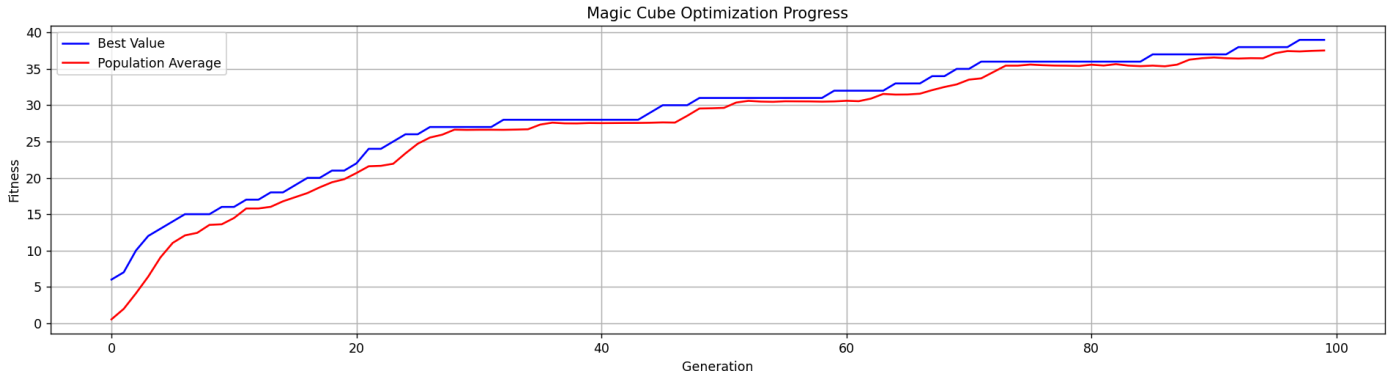


Nilai *objective function* awal: 1
Nilai *objective function* terakhir: 37
Jumlah populasi: 1000
Banyak iterasi: 100
Durasi: 154.68 detik

3.3.6.3. Eksperimen 3

```
Layer 1:
[100, 103,  51, 115, 119]
[ 41,  72,   4, 105,   2]
[110,  90,  75,  59,  54]
[ 44,  42,  11, 118, 109]
[ 76,  55, 108,   3,  46]

Layer 2:
[ 82, 116,  27,  12,  47]
[112,  97,  10,  63, 102]
[ 25,  28, 121,  73,  61]
[ 37, 125, 124,  15,  95]
[  8,  87,  34,  65,  17]

Layer 3:
[ 32,  60,  16, 120,  13]
[  5,  38,  62, 104,  84]
[ 21,  29, 106,  92,  81]
[ 18,  83,  36,  14,  48]
[  9,  58,  40, 111,  79]

Layer 4:
[113,  57,   1, 117,  68]
[ 71,  69,  91,  96,  26]
[ 24,  99,  85,  52, 107]
[ 33,  20,  43,  22,  64]
[ 98,   7,  78,  35,  39]

Layer 5:
[122,  93,  89,  88,  31]
[ 49,  45,  74,  23,  80]
[ 67, 123,  30,  56,  66]
[ 70,  77, 101,  53,   6]
[ 19,  50,  86, 114,  94]

Current Value: 3
```

```
Layer 1:
[108,  61,   2, 118,  51]
[ 15,  20,  15,  26,  14]
[115,  92,  82, 113,   6]
[ 12,  76,  55,  81,  23]
[ 80,  66, 117,  54,  24]

Layer 2:
[ 27,  74,  39,  24,  66]
[  3, 113,  55,  44, 100]
[ 96,   7,  12,  21,  46]
[ 20,  30,  23,  86,  38]
[ 18,  93,  51,  76,  77]

Layer 3:
[ 10,  71,  64,  15,  66]
[ 61,  65,  15,  83,  29]
[117, 107,  26,  11,  13]
[ 27, 105, 115, 120, 100]
[ 35,  47,  95,  38,  94]

Layer 4:
[ 58,  15,  63,  42, 105]
[119, 125, 117,  81,  11]
[121,  24,  35,  46,  89]
[ 17,  47,  55,  57,   8]
[ 47,   6,  45,   2,  40]

Layer 5:
[100,  55,  72,  12,  27]
[117,  17,  79,  81, 105]
[ 25,  85,  90, 124, 100]
[ 44, 103,  67,  63,  38]
[ 14, 124,   7,  35,  45]

Current Value: 39
```



Nilai *objective function* awal: 3
Nilai *objective function* terakhir: 39
Jumlah populasi: 1000
Banyak iterasi: 100
Durasi: 155.95 detik

3.4. Analisis Perbandingan

Dikarenakan algoritma hill-climbing memiliki hasil dan proses yang cukup identik antara satu algoritma dengan algoritma lainnya, maka pada beberapa bagian di sub bab ini, algoritma hill-climbing akan dijelaskan secara bersama-sama.

3.4.1. Kedekatan Algoritma Terhadap Global Optima

Berikut merupakan kedekatan tiap-tiap algoritma untuk bisa mencapai global optima dan penjelasan alasannya.

1. Hill-Climbing

   Hasil eksperimen menunjukkan bahwa algoritma hill-climbing mencapai nilai *function objective* yang lebih dekat dibandingkan beberapa algoritma lainnya. Walaupun hanya dapat mencapai *local optima*, algoritma hill-climbing dapat memperoleh nilai *function objective* yang lebih baik dibandingkan dengan simulated annealing dan genetic algorithm dalam waktu yang singkat dan dengan sedikit iterasi. Hal ini disebabkan oleh strategi algoritma hill-climbing yang sederhana dan langsung menuju perbaikan, meskipun perlu diwaspadai risiko terjebak di *local optima*.

2. Random Restart Hill-Climbing

   Pada algoritma random restart hill-climbing, hasil yang diperoleh adalah yang terbaik pada durasi yang sama dibandingkan algoritma lain yang dijelaskan. Hal ini karena algoritma ini menerapkan hill-climbing secara berulang, sehingga dapat mencapai hasil terbaik dibandingkan algoritma lain pada durasi yang sama.

3. Simulated Annealing

   Jika durasi tidak terbatas dan pengaturan parameter dilakukan dengan tepat, algoritma simulated annealing dapat menghasilkan kedekatan yang baik dengan *global optima*. Namun, hasil eksperimen menunjukkan performa yang kurang baik dibandingkan hill-climbing, serta membutuhkan durasi yang lama. Hal ini disebabkan oleh ketidakseimbangan eksplorasi dan eksploitasi, serta pemilihan nilai suhu (temperatur) yang kurang tepat.

4. Genetic Algorithm

   Pada penerapan algoritma genetic algorithm ini, berdasarkan hasil eksperimen yang telah dilakukan, performa yang diperoleh menunjukkan hasil yang kurang memuaskan dibandingkan dengan metode hill-climbing. Nilai yang diperoleh cenderung jauh dari nilai *objective function*. Hasil ini disebabkan oleh keterbatasan fungsi yang digunakan dalam algoritma, yang berdampak signifikan terhadap hasil akhirnya.

3.4.2. Perbandingan Hasil Pencarian Antar Algoritma

Berdasarkan eksperimen yang telah dilakukan, diperoleh hasil bahwa algoritma hill-climbing, terutama random restart hill-climbing, adalah yang paling mendekati *local optima*. Pada random restart hill-climbing, nilai *functional objective* yang diperoleh adalah sebesar 51/109.

Untuk algoritma simulated annealing, berdasarkan tiga kali eksperimen, nilai *functional objective* tertinggi yang diperoleh adalah 37 dengan durasi 533 detik. Sementara itu, algoritma genetic algorithm memperoleh nilai *functional objective* tertinggi sebesar 39 untuk populasi 1000 dan iterasi 100 dengan durasi 156 detik.

Jika tujuan hanya sebatas menentukan nilai *functional objective* terbesar dengan durasi yang pendek, maka algoritma hill-climbing, genetic algorithm, dan simulated annealing adalah urutan untuk yang terbaik. Namun, perlu diperhatikan bahwa jika tujuannya adalah untuk mencapai *global optima*, maka penggunaan hill-climbing tidak disarankan. Hal ini dikarenakan algoritma hill-climbing rentan terjebak di *local optima*.

3.4.3. Perbandingan Durasi Pencarian Antar Algoritma

Berdasarkan durasi yang ditampilkan pada hasil eksperimen, jika tujuannya hanya sebatas mencapai local optima, maka algoritma yang memiliki durasi tercepat secara berurutan adalah hill-climbing, genetic algorithm (dengan populasi 1000 dan iterasi 100), dan simulated annealing.

Namun, apabila tujuannya adalah untuk memperoleh global optimum, tidak dapat ditarik kesimpulan yang jelas untuk algoritma simulated annealing dan genetic algorithm. Hal ini dikarenakan tidak ada hasil eksperimen yang menunjukkan bahwa kedua algoritma tersebut dapat mencapai global optimum. Sementara itu, hill-climbing memiliki kemungkinan yang sangat kecil untuk dapat mencapai global optimum karena rentan terjebak di optimum lokal.

3.4.4. Konsistensi Hasil Akhir

Berikut merupakan konsistensi dari hasil akhir yang diperoleh oleh masing-masing algoritma.

a. Steepest-ascent hill-climbing
   Hasil yang ditunjukkan konsisten berada pada nilai *objective function* sekitar 40.

b. Sideways-moves hill-climbing
   Hasil yang ditunjukkan konsisten berada pada nilai *objective function* sekitar 40.

c. Stochastic hill-climbing
   Hasil yang ditunjukkan konsisten berada pada nilai *objective function* sekitar 40.

d. Random restart hill-climbing

Hasil yang ditunjukkan konsisten pada nilai *objective function* 51. Hal ini disebabkan karena algoritma ini merupakan pengembangan dari algoritma steepest-ascent hill-climbing yang dijalankan berulang kali, sehingga dapat disimpulkan terdapat local optima pada nilai *objective function* 51.

e. Simulated annealing
   Nilai *objective function* untuk algoritma ini berdasarkan tiga eksperimen yang dilakukan konsisten berada pada kisaran 30.

f. Genetic algorithm
   Nilai *objective function* untuk algoritma ini berdasarkan eksperimen yang dilakukan menunjukkan hasil yang beragam. Keberagaman ini dipengaruhi oleh jumlah populasi dan iterasi yang digunakan. Bahkan untuk parameter populasi dan iterasi yang sama, hasil yang diperoleh tetap beragam. Hal ini disebabkan oleh ukuran populasi dalam eksperimen yang minimal 100 individu, yang menghasilkan kemungkinan munculnya individu yang beragam dalam populasi. Akibatnya, nilai *objective function* akhir sangat bergantung pada keberagaman individu dalam suatu populasi.

3.4.5. Pengaruh Jumlah Populasi dan Iterasi pada *Genetic Algorithm*

Berdasarkan eksperimen yang dilakukan, semakin tinggi jumlah populasi maupun iterasi, maka hasil yang diperoleh bisa menjadi lebih baik. Hal ini dikarenakan jumlah populasi yang lebih besar dan jumlah iterasi yang lebih tinggi memberikan keberagaman individu yang lebih tinggi dalam populasi, meningkatkan probabilitas mendapatkan solusi yang lebih optimal, memperluas ruang pencarian *global optima*, mengurangi risiko mengalami local optima, dan memberikan lebih banyak kesempatan untuk mutasi dan *crossover* populasi.

Kombinasi dari jumlah populasi dan iterasi yang tinggi dapat mengoptimalkan proses pencarian solusi melalui mekanisme mutasi dan *crossover*, meningkatkan kemampuan algoritma dalam mengeksplorasi dan mengeksploitasi ruang pencarian, dan memaksimalkan potensi algoritma genetika dalam menemukan solusi optimal global. Namun perlu diperhatikan bahwa peningkatan jumlah populasi dan iterasi juga akan berdampak pada durasi yang lebih lama.

Bab 4
Kesimpulan dan Saran

4.1 Kesimpulan

Berdasarkan implementasi dan eksperimen yang telah dilakukan pada penyelesaian Diagonal Magic Cube menggunakan tiga algoritma local search (Hill-Climbing, Simulated Annealing, dan Genetic Algorithm), dapat ditarik beberapa kesimpulan:

1. Perbandingan Performa Algoritma:
    a. Random Restart Hill-Climbing menunjukkan performa terbaik dengan nilai objective function mencapai 51/109
    b. Simulated Annealing mencapai nilai 35-37/109 dengan konsistensi yang baik
    c. Genetic Algorithm mencapai nilai maksimal 39/109 pada konfigurasi 1000 populasi dan 100 iterasi
    d. Masing-masing algoritma memiliki trade-off antara kualitas solusi dan waktu komputasi
2. Karakteristik Algoritma
    a. Hill-Climbing:
        i. Cepat mencapai local optima
        ii. Memiliki durasi eksekusi singkat (14-18 detik untuk Steepest-Ascent), kecuali untuk random restart
        iii. Konsisten mencapai nilai sekitar 40-50
        iv. Random Restart meningkatkan eksplorasi dan hasil akhir, akan tetapi durasi eksekusi yang lama
    b. Simulated Annealing:
        i. Durasi eksekusi lebih lama (~530 detik)
        ii. Konsisten mencapai nilai 35-37
        iii. Memiliki mekanisme escape dari local optima yang efektif (stuck rate rendah)
        iv. Balance yang baik antara eksplorasi dan eksploitasi
    c. Genetic Algorithm:
        i. Performa bergantung pada jumlah populasi dan iterasi
        ii. Hasil lebih baik dengan populasi dan iterasi yang lebih besar
        iii. Waktu eksekusi bervariasi (19-156 detik)
        iv. Kurang konsisten dibanding algoritma lainnya
3. Efektivitas dan Efisiensi
    a. Tidak ada algoritma yang mencapai global optima (109)
    b. Hill-Climbing paling efisien dalam hal waktu komputasi
    c. Simulated Annealing paling konsisten hasilnya
    d. Genetic Algorithm paling fleksibel dengan parameter yang bisa disesuaikan

4.2 Saran

1. Pengembangan Algoritma
    a. Implementasikan hybrid approach yang menggabungkan kelebihan dari masing-masing algoritma
    b. Kembangkan heuristik khusus untuk Diagonal Magic Cube
    c. Optimalkan parameter algoritma menggunakan meta-optimization
2. Optimasi Performa
    a. Implementasikan parallelisasi untuk mempercepat komputasi
    b. Gunakan struktur data yang lebih efisien untuk representasi kubus
    c. Optimalkan fungsi objektif untuk evaluasi yang lebih cepat
3. Parameter Tuning
    a. Simulated Annealing: Sesuaikan cooling schedule dan initial temperature
    b. Genetic Algorithm: Eksperimen dengan berbagai operator crossover dan mutasi
    c. Hill-Climbing: Implementasikan adaptive step size
4. Metodologi Eksperimen
    a. Lakukan eksperimen dengan jumlah iterasi yang lebih besar
    b. Tambahkan metrik evaluasi tambahan
    c. Analisis statistik yang lebih mendalam untuk hasil eksperimen

Pembagian Tugas

Nama Mahasiswa	:	Habib Akhmad Al Farisi
NIM	:	18222029
No	**Tugas yang dikerjakan**	
1	Membuat implementasi algoritma genetic algorithm dan melakukan eksperimen pada genetic algorithm	
2	Melakukan analisis perbandingan, hasilnya berupa laporan analisis perbandingan pada dokumen	

Nama Mahasiswa	:	Alfandito Rais Akbar
NIM	:	18222037
No	**Tugas yang dikerjakan**	
1	Membuat implementasi magic cube, steepest ascent sideways move, stochastic, dan visualizer ‘video player’ dan melakukan eksperimen pada steepest ascent sideways move, dan stochastic	
2	Mengerjakan objective function dan implementasi algoritma magic cube, steepest ascent sideways move, stochastic, dan visualizer ‘video player’ pada dokumen	

Nama Mahasiswa	:	Winata Tristan
NIM	:	18222061
No	**Tugas yang dikerjakan**	
1	Membuat implementasi algoritma Random Restart Hill Climbing dan melakukan eksperimen pada Random Restart Hill Climbing	
2	Mengerjakan implementasi algoritma random restart pada dokumen	

Nama Mahasiswa	:	Muhammad Rafi Dhiyaulhaq
NIM	:	18222069
No	**Tugas yang dikerjakan**	
1	Membuat deskripsi persoalan, membuat implementasi algoritma simulated annealing, hasil eksperimen simulated annealing, kesimpulan, dan saran pada dokumen	
2	Membuat algoritma dan visualisasi untuk simulated annealing, membuat struktur project, membuat video player dalam js, membuat main.py, memperbaiki file yang sudah dibuat (MagicCube, genetic_algorithm, visualizer), dan README	