

LAPORAN TUGAS BESAR 1

IF3170 INTELIGENSI ARTIFISIAL

Pencarian Solusi *Diagonal Magic Cube* dengan *Local Search*



Disusun oleh:

Agil Fadillah Sabri 13522006

Muhammad Althariq Fairuz 13522027

Bastian H Suryapratama 13522034

Haikal Assyauqi 13522052

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

DAFTAR ISI.....	1
BAB I.....	2
DESKRIPSI PERSOALAN.....	2
BAB II.....	3
PEMBAHASAN.....	3
2.1 Pemilihan Objective Function.....	3
2.2 Implementasi Algoritma Local Search.....	5
2.3 Hasil Percobaan dan Analisis.....	24
2.4 Analisis Kinerja Algoritma terhadap Pencarian Solusi Diagonal Magic Cube.....	42
BAB III.....	45
KESIMPULAN DAN SARAN.....	45
3.1 Kesimpulan.....	45
3.2 Saran.....	45
PEMBAGIAN TUGAS.....	46
REFERENSI.....	47

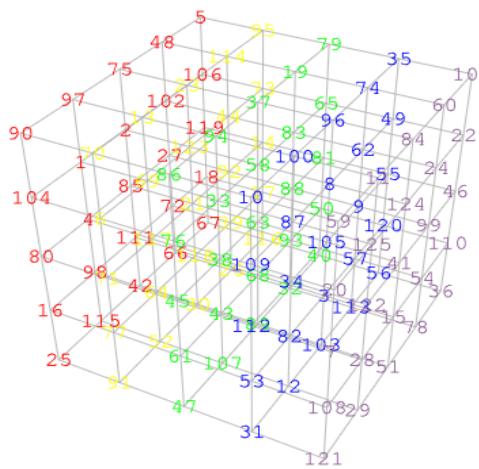
BAB I

DESKRIPSI PERSOALAN

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

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

Berikut ilustrasi dari *diagonal magic cube* berukuran 5:



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

Gambar 1.1 Diagonal Magic Cube 5x5x5

(Sumber: [Magic Cube](#))

Pada tugas kali ini, akan difokuskan pada *diagonal magic cube* $5 \times 5 \times 5$, sebuah bentuk khusus dari *magic cube* dengan panjang sisi lima. Ukuran ini menghadirkan tantangan yang lebih besar karena ketentuan *magic number* yang diperlukan untuk memvalidasi keseimbangan dalam seluruh dimensi dan arah pada kubus menjadi lebih kompleks seiring dengan peningkatan ukuran kubus.

BAB II

PEMBAHASAN

2.1 Pemilihan *Objective Function*

Fungsi objektif adalah suatu fungsi yang digunakan dalam persoalan optimasi untuk mengevaluasi seberapa baik solusi yang diberikan saat ini terhadap solusi optimal. Dalam persoalan *Diagonal Magic Cube*, fungsi ini akan bertujuan untuk mengukur seberapa baik solusi yang diberikan, yaitu seberapa baik kubus saat ini terhadap kondisi “*magic*”.

Dalam persoalan *magic cube*, *magic number* dari suatu *magic cube* berukuran n , diberikan oleh persamaan berikut:

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

Untuk kubus dengan $n = 5$, maka *magic number*-nya adalah:

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

$$M = 315$$

Itu artinya, untuk kubus dengan $n = 5$, dikatakan *magic cube* jika setiap baris, kolom, tiang, dan diagonalnya berjumlah 315. Oleh karena itu, fungsi objektif yang dibuat perlu menghitung bagaimana solusi saat ini menyimpang dari kondisi tersebut. Agar fungsi objektif mampu melakukan evaluasi sebaik mungkin, maka perhitungan haruslah mencakup semua arah pada kubus (baris, kolom, tiang, dan diagonal).

Dalam sebuah kubus dengan $n = 5$, akan terdapat total:

- 25 baris (5 untuk setiap lapisan).
- 25 kolom (5 untuk setiap lapisan).
- 25 tiang (5 untuk setiap potongan vertikal).
- 30 diagonal bidang (10 untuk setiap dimensi xy, xz, yz).
- 4 diagonal ruang.

Dari uraian di atas, maka jumlah total baris, kolom, tiang, dan diagonal untuk *magic cube* dengan $n = 5$ adalah $25 + 25 + 25 + 30 + 4 = 109$. Dalam persoalan ini, fungsi objektif yang digunakan adalah jumlah baris, kolom, tiang, dan diagonal yang masih belum berjumlah sama dengan 315.

Suatu *magic cube* dikatakan telah memenuhi properti “*magic*” jika jumlah baris, kolom, tiang, dan diagonal yang berjumlah tidak sama dengan 315 adalah 0. Adapun jika jumlah baris, kolom, tiang, dan diagonal yang berjumlah tidak sama dengan 315 adalah lebih dari 0, maka kubus dikatakan menyimpang dari *magic cube*. Penyimpangan terbesar adalah saat jumlah baris, kolom, tiang, dan diagonal yang berjumlah tidak sama dengan 315 ada 109 (tidak ada satupun properti kubus yang memenuhi sifat “*magic*”).

Karena algoritma harus memaksimumkan fungsi objektif, maka fungsi objektif dibuat berada pada rentang nilai 0 hingga -109 dengan 0 menyatakan nilai maksimum (kubus telah mencapai sifat “*magic*”) dan -109 menyatakan nilai minimum (kubus berada pada keadaan paling jauh dari sifat “*magic*”). Oleh karena itu, fungsi objektif persoalan *magic cube* ini dapat dirumuskan sebagai berikut:

Objective Function = – (banyak baris, kolom, tiang, dan diagonal yang berjumlah tidak sama dengan 315)

Ada beberapa alasan pemilihan fungsi objektif ini, yaitu:

1. Memasukkan setiap Properti dalam Proses Perhitungan

Karena sifat *magic cube* yang unik, dimana setiap baris, kolom, tiang, dan diagonal harus berjumlah sama dengan *magic number*, maka fungsi objektif yang digunakan haruslah memperhitungkan semua properti tersebut secara bersamaan.

2. Sederhana dan Mudah Diimplementasikan ke dalam Program

Menghitung jumlah baris, kolom, tiang, dan diagonal kubus yang berjumlah tidak sama dengan *magic number* adalah cara yang terbilang sederhana untuk melihat penyimpangan suatu kubus terhadap kondisi “*magic*”, sehingga akan memudahkan pada tahap implementasi program nantinya.

3. Mengarah ke Tujuan Utama

Persoalan *magic cube* menginginkan semua baris, kolom, tiang, dan diagonal berjumlah sama dengan *magic number*, dan fungsi objektif ini dapat secara langsung mengukur sejauh mana penyimpangan solusi saat ini terhadap kondisi ideal tersebut. Nilai fungsi yang semakin mendekati 0 menandakan kubus tersebut semakin dekat dengan kondisi “*magic*”.

4. *Reusable*

Fungsi objektif ini tidak hanya dapat digunakan untuk persoalan *magic cube* dengan $n = 5$, tapi juga dapat diterapkan ke semua persoalan *magic cube* lainnya dengan ukuran sisi berapapun.

2.2 Implementasi Algoritma *Local Search*

Program Pencarian Solusi *Diagonal Magic Cube* dengan *Local Search* ini diimplementasikan menggunakan bahasa pemrograman python (.py) dan memanfaatkan *library* Matplotlib untuk visualisasi hasil pencarian.

1. Kelas MagicCube

Class MagicCube	
<i>Atribut</i>	
+ cube: list	<i>State magic cube</i> (susunan kubus $5 \times 5 \times 5$ yang terdiri dari angka 1-125).
+ value: int	Nilai <i>objective value</i> dari <i>magic cube</i> .
+ fitness: int	Nilai <i>fitness value</i> dari <i>magic cube</i> .
<i>Method</i>	
+ MagicCube()	Konstruktor kelas MagicCube.
- __value() → int	Mengembalikan nilai objektif dari kubus.
- __fitness() → int	Mengembalikan nilai <i>fitness</i> dari kubus.
- refresh() → None	Mengupdate nilai atribut <i>value</i> dan <i>fitness</i> kubus.
+ swap(a: tuple, b: tuple) → None	Menukar dua elemen dalam kubus.
+ scramble(start: int, end: int) → None	Mengacak elemen kubus pada <i>range</i> tertentu.
+ inverse(start: int, end: int) → None	Membalik elemen kubus pada <i>range</i> tertentu.
+ highestSuccessor() → MagicCube	Mengembalikan <i>successor</i> dengan <i>objective value</i> tertinggi.
+ randomSuccessor() → MagicCube	Mengembalikan <i>successor</i> secara acak.
<i>Source Code</i>	

```
1  class MagicCube:
2      def __init__(self) -> None:
3          """
4              Constructor kelas Cube
5              membuat kubus magic berukuran 5x5x5 secara acak
6              """
7      self.cube = list(range(1, 126))
8      random.shuffle(self.cube)
9      self.cube = np.array(self.cube).reshape(5,5,5)
10
11     self.value = self.__value()
12     self.fitness = self.__fitness()
13
14     def __getitem__(self, key: tuple) -> int:
15         """
16             mengembalikan nilai elemen kubus pada indeks tertentu
17
18             Args:
19                 key (tuple): indeks elemen kubus
20
21             return:
22                 int: nilai elemen kubus
23                 """
24
25     return self.cube[key]
```

```

1  def __value(self) -> int:
2      """
3          mengembalikan nilai objektif dari kubus
4          value = -(banyaknya baris, kolom, tiang, dan diagonal yang belum berjumlah 315)
5          range value = (-109, 0)
6
7      return:
8          int: nilai objektif kubus
9      """
10     target = 315
11     count_mismatch = 0
12
13     # Mengecek baris, kolom, dan tiang
14     for i in range(5):
15         for j in range(5):
16             if np.sum(self[i, j, :]) != target: # Baris
17                 count_mismatch += 1
18             if np.sum(self[i, :, j]) != target: # Kolom
19                 count_mismatch += 1
20             if np.sum(self[:, i, j]) != target: # Tiang
21                 count_mismatch += 1
22
23     # Mengecek diagonal pada setiap potongan bidang
24     for i in range(5):
25         if np.sum(np.diagonal(self[i, :, :])) != target: # Diagonal bidang xy
26             count_mismatch += 1
27         if np.sum(np.diagonal(self[:, i, :])) != target: # Diagonal bidang xz
28             count_mismatch += 1
29         if np.sum(np.diagonal(self[:, :, i])) != target: # Diagonal bidang yz
30             count_mismatch += 1
31         if np.sum(np.diagonal(np.fliplr(self[i, :, :]))) != target: # Diagonal bidang xy (berLawanan)
32             count_mismatch += 1
33         if np.sum(np.diagonal(np.fliplr(self[:, i, :]))) != target: # Diagonal bidang xz (berLawanan)
34             count_mismatch += 1
35         if np.sum(np.diagonal(np.fliplr(self[:, :, i]))) != target: # Diagonal bidang yz (berLawanan)
36             count_mismatch += 1
37
38     # Mengecek diagonal pada kubus
39     if np.sum([self[i, i, i] for i in range(5)]) != target: # Diagonal ruang 1
40         count_mismatch += 1
41     if np.sum([self[i, i, 4 - i] for i in range(5)]) != target: # Diagonal ruang 2
42         count_mismatch += 1
43     if np.sum([self[i, 4 - i, i] for i in range(5)]) != target: # Diagonal ruang 3
44         count_mismatch += 1
45     if np.sum([self[4 - i, i, i] for i in range(5)]) != target: # Diagonal ruang 4
46         count_mismatch += 1
47
48     return -count_mismatch
49
50 def __fitness(self) -> int:
51     """
52         mengembalikan nilai fitness dari kubus
53         fitness = 109 + value
54         range fitness = (0, 109)
55
56     return:
57         int: nilai fitness kubus
58     """
59     return 109 + self.value

```

```
1 def refresh(self) -> None:
2     """
3     mengupdate nilai value dan fitness kubus
4     """
5     self.value = self.__value()
6     self.fitness = self.__fitness()
7
8 def swap(self, a: tuple, b: tuple) -> None:
9     """
10    menukar dua elemen dalam kubus
11
12 Args:
13     a (tuple): indeks elemen pertama
14     b (tuple): indeks elemen kedua
15 """
16     self(cube[a], self(cube[b]) = self(cube[b]), self(cube[a])
17     self.refresh()
18
19 def scramble(self, start: int, end: int) -> None:
20     """
21     mengacak elemen kubus pada range tertentu
22
23 Args:
24     start (int): indeks awal
25     end (int): indeks akhir
26 """
27     cube = self(cube).flatten() # ubah cube menjadi 1 dimensi
28     cube[start:end] = np.random.permutation(cube[start:end])
29     self(cube = cube.reshape(5,5,5)
30     self.refresh()
31
32 def inverse(self, start: int, end: int) -> None:
33     """
34     membalik elemen kubus pada range tertentu
35
36 Args:
37     start (int): indeks awal
38     end (int): indeks akhir
39 """
40     cube = self(cube).flatten() # ubah cube menjadi 1 dimensi
41     cube[start:end] = cube[start:end][::-1]
42     self(cube = cube.reshape(5,5,5)
43     self.refresh()
```



```
1  def highestSuccessor(self) -> 'MagicCube':
2      """
3          mengembalikan succussor dengan objective value tertinggi
4      """
5      return:
6      MagicCube: objek kubus successor
7      """
8      max_value = -109
9      max_cube = None
10     for x1 in range(5):
11         for y1 in range(5):
12             for z1 in range(5):
13                 for x2 in range(x1, 5):
14                     for y2 in range(y1, 5):
15                         for z2 in range(z1 + 1, 5):
16                             new_cube = MagicCube()
17                             new_cube.cube = self(cube).copy()
18                             new_cube.swap((x1, y1, z1), (x2, y2, z2))
19
20                             if new_cube.value > max_value:
21                                 max_value = new_cube.value
22                                 max_cube = new_cube
23
24     return max_cube
25
26     def randomSuccessor(self) -> 'MagicCube':
27         """
28             mengembalikan succussor secara random
29         """
30         return:
31         MagicCube: objek kubus successor
32         """
33         first_elmt = tuple(np.random.randint(5, size=3))
34         second_elmt = tuple(np.random.randint(5, size=3))
35         while first_elmt == second_elmt:
36             second_elmt = tuple(np.random.randint(5, size=3))
37
38         new_cube = MagicCube()
39         new_cube.cube = self(cube).copy()
40         new_cube.swap(first_elmt, second_elmt)
41
42     return new_cube
```

2. Kelas HillClimbing

Class HillClimbing	
<i>Atribut</i>	
<i>Method</i>	
+ HillClimbing()	Konstruktor kelas HillClimbing.
+ steepestAscent(cube: MagicCube) → tuple[MagicCube, list]	Melakukan pencarian <i>steepest ascent hill-climbing</i> pada kubus <i>magic</i> .
+ sidewaysMove(cube: MagicCube, max_iterations: int) → tuple[MagicCube, list]	Melakukan pencarian <i>hill-climbing with sideways move</i> pada kubus <i>magic</i> .
+ randomRestart(cube: MagicCube, max_restarts: int) → tuple[MagicCube, list, list, int]	Melakukan pencarian <i>random restart hill-climbing</i> pada kubus <i>magic</i> .
+ stochastic(cube: MagicCube, max_iterations: int) → tuple[MagicCube, list, int]	Melakukan pencarian <i>stochastic hill-climbing</i> pada kubus <i>magic</i> .
<i>Source Code</i>	

```

1  class HillClimbing:
2      def steepestAscent(self, cube: MagicCube) -> tuple[MagicCube, List]:
3          """
4              melakukan pencarian steepest ascent hill-climbing pada kubus magic
5
6          Args:
7              cube (MagicCube): objek kubus magic
8
9          return:
10             tuple[MagicCube, list]: objek kubus hasil pencarian dan list nilai objektif
11             """
12
13     current = cube
14     objective_value = [current.value] # List of objective value of the cube
15
16     while True:
17         print(f"Current Value: {current.value}")
18         best_neighbor = current.highestSuccessor()
19         if best_neighbor.value <= current.value:
20             break
21         current = best_neighbor
22         objective_value.append(current.value)
23
24     return current, objective_value
25
26     def sidewaysMove(self, cube: MagicCube, max_iterations: int = 3) -> tuple[MagicCube, List]:
27         """
28             melakukan pencarian sideways move hill-climbing pada kubus magic
29
30         Args:
31             cube (MagicCube): objek kubus magic
32             max_iterations (int): maksimum sideways move
33
34         return:
35             tuple[MagicCube, list]: objek kubus hasil pencarian dan list nilai objektif
36             """
37
38     current = cube
39     value_count = {} # the count of the side way move
40     objective_value = [current.value] # List of objective value of the cube
41
42     while True:
43         print(f"Current Value: {current.value}")
44         best_neighbor = current.highestSuccessor()
45
46         if best_neighbor.value < current.value:
47             break
48
49         if best_neighbor.value == current.value:
50             # Add count of the value
51             value_count[best_neighbor.value] = value_count.get(best_neighbor.value, 0) + 1
52
53             # If the value has been reached max_iterations, break
54             if value_count[best_neighbor.value] >= max_iterations:
55                 break
56
57         current = best_neighbor
58         objective_value.append(current.value)
59
60     return current, objective_value

```

```

1  def randomRestart(self, cube: MagicCube, max_restarts: int = 3) -> tuple[MagicCube, List, List, int]:
2      """
3          melakukan pencarian random restart hill-climbing pada kubus magic
4
5          Args:
6              cube (MagicCube): objek kubus magic
7              max_restarts (int): maksimum restart
8
9          return:
10         tuple[MagicCube, list, list, int]: objek kubus hasil pencarian,
11                                         list nilai objektif, list iterasi per restart, dan jumlah restart
12         """
13
14     objective_values = []
15     iterations_per_restart = []
16
17     for i in range(max_restarts + 1):
18         print(f"Restart {i}")
19         steepest_ascent_result, objective_values_per_restart = self.steepestAscent(cube)
20         objective_values.extend(objective_values_per_restart)
21         iterations_per_restart.append(len(objective_values_per_restart))
22
23         if steepest_ascent_result.value == 0:
24             return steepest_ascent_result, objective_values, iterations_per_restart, i
25
26         if i < max_restarts:
27             cube = MagicCube()
28
29     return steepest_ascent_result, objective_values, iterations_per_restart, i
30
31 def stochastic(self, cube: MagicCube, max_iterations: int) -> tuple[MagicCube, List, int]:
32     """
33         melakukan pencarian stochastic hill-climbing pada kubus magic
34
35         Args:
36             cube (MagicCube): objek kubus magic
37             max_iterations (int): maksimum iterasi
38
39         return:
40         tuple[MagicCube, list, int]: objek kubus hasil pencarian,
41                                         list nilai objektif, dan jumlah iterasi
42         """
43
44     objective_values = []
45     objective_values.append(cube.value)
46
47     for i in range(max_iterations):
48         print(f"iterasi-{i} Current Value: {cube.value}")
49         neighbor = cube.randomSuccessor()
50         if neighbor.value > cube.value:
51             cube = neighbor
52
53         objective_values.append(cube.value)
54
55         if cube.value == 0:
56             return cube, objective_values, i
57
58     return cube, objective_values, i

```

3. Kelas SimulatedAnnealing

Class SimulatedAnnealing	
<i>Atribut</i>	
<i>Method</i>	
+ temperature: int	Menyimpan temperatur awal yang bernilai 1.
+ alpha_cold: float	Parameter untuk menurunkan suhu ketika terlalu sering pengulangan dengan nilai sama, bernilai 0,9999.
+ alpha: float	Parameter untuk menurunkan suhu secara normal, bernilai 0,99999.
+ min_temperature: float	Parameter yang menjadi batas akhir dari pencarian, bernilai 0,00001 (sangat sulit untuk mendapatkan nilai <i>temperature</i> tepat 0).
+ iteration: int	Jumlah iterasi dalam 1 kali proses pencarian.
+ currentValue: list	Menyimpan <i>objective value</i> tiap iterasi.
+ eulerValue: list	Menyimpan nilai fungsi euler pada tiap iterasi.
+ probability: float	Batas bawah dari probabilitas dimana perpindahan ke <i>state</i> yang lebih buruk diperbolehkan. Angka inisiasi adalah 0,3, namun berubah seiring waktu.
+ SimulatedAnnealing()	Konstruktor kelas SimulatedAnnealing.
+ getTemperature() → float	Mengambil nilai temperatur saat ini.
+ setTemperature()	Mengatur temperatur seiring waktu.
+ simulatedAnnealing (current: MagicCube) → MagicCube	Menjalankan program simulated annealing.
<i>Source Code</i>	

```
1  class SimulatedAnnealing:
2      def __init__(self):
3          """
4              Constructor kelas SimulatedAnnealing
5          """
6          self.temperature = 1
7          self.alpha_cold = 0.99999
8          self.alpha = 0.9999
9          self.min_temperature = 0.00001
10         self.iteration = 0
11         self.curretValue = []
12         self.eulerValue = []
13         self.probability = 0.3
14         self.stuck = 0
15
16     def getTemperature(self) -> float:
17         """
18             Mengembalikan suhu saat ini
19
20             Returns:
21                 float: suhu saat ini
22             """
23     return self.temperature
24
25     def setTemperature(self, temperature: float) -> None:
26         """
27             Mengatur suhu saat ini
28
29             Args:
30                 temperature (float): suhu yang akan diatur
31             """
32         self.temperature = temperature
```

```

1  def simulatedAnnealing(self, current: MagicCube) -> MagicCube:
2      random_successor = MagicCube()
3      no_improvement_steps = 0
4      probability = 0
5      self.currenValue.append(current.value)
6
7      while self.getTemperature() > self.min_temperature:
8          print(self.getTemperature())
9          random_successor = current.randomSuccessor()
10         delta = random_successor.value - current.value
11
12         if delta >= 0:
13             current = random_successor
14             probability = 1
15             if delta == 0:
16                 no_improvement_steps += 0
17             else :
18                 no_improvement_steps = 0
19         else:
20             probability = 2.71828 ** (delta / self.getTemperature())
21             self.stuck += 1
22             if probability > self.probability:
23                 current = random_successor
24                 no_improvement_steps = 0 # Reset counter on acceptance
25             else:
26                 no_improvement_steps += 1
27
28             self.eulerValue.append(probability)
29             self.currenValue.append(current.value)
30             self.iteration += 1
31
32             # self.temperature *= self.alpha
33
34             if no_improvement_steps // 2000 > 0:
35                 if no_improvement_steps // 2000 > 0:
36                     minus = no_improvement_steps // 2000
37                     elif no_improvement_steps // 2000 > 6:
38                         minus = 2.9
39                     if current.value < -52 and self.temperature > 0.001:
40                         self.probability = 0.3 + 0.05 * (1 - minus)
41                         self.temperature = self.temperature * self.alpha
42                     else:
43                         self.probability = 0.3
44                         self.temperature = self.temperature * self.alpha_cold
45
46                     if current.value == 0 :
47                         return current
48
49             return current

```

4. Kelas GeneticAlgorithm

Class GeneticAlgorithm	
<i>Atribut</i>	
+ value_max_history: list	Menyimpan daftar <i>objective value</i> tertinggi dari setiap generasi.
+ value_avg_history: list	Menyimpan daftar rata-rata <i>objective value</i> dari setiap generasi.
+ max_initial_cube: MagicCube	Menyimpan <i>cube</i> dengan <i>objective value</i> tertinggi dari populasi awal.
+ max_final_cube: MagicCube	Menyimpan <i>cube</i> dengan <i>objective value</i> tertinggi dari generasi terakhir saat proses pencarian berhenti.
<i>Method</i>	
+ GeneticAlgorithm()	Konstruktor kelas GeneticAlgorithm.
- __update_history(population: list) → None	Memperbarui <i>history objective value</i> maksimum dan rata-rata.
- __roulette_wheel(population: list, fitness_sum: int) → list	Membuat list <i>roulette wheel</i> berdasarkan populasi yang diterima.
- __selection(roulette_wheel: list) → tuple[MagicCube, MagicCube]	Melakukan proses seleksi dua individu berdasarkan <i>roulette wheel</i> yang diterima.
- __crossover(parent1: MagicCube, parent2: MagicCube) → tuple[MagicCube, MagicCube]	Melakukan proses <i>crossover</i> dua individu.
- __mutation(individu: MagicCube, type: int) → None	Melakukan proses mutasi pada individu. Terdapat tiga tipe mutasi, yaitu: <ul style="list-style-type: none"> - <i>Swap</i>, menukar posisi dua elemen secara acak. - <i>Scramble</i>, mengacak posisi sekumpulan elemen pada <i>range</i> tertentu. - <i>Inverse</i>, membalik urutan sekumpulan elemen pada posisi tertentu.
+ run(self, population: list, population_size: int, max_generation: int) → tuple[float, int]	Menjalankan <i>genetic algorithm</i> .

Source Code

```
1  class GeneticAlgorithm:
2      def __init__(self) -> None:
3          """
4              Constructor kelas GeneticAlgorithm
5          """
6          self.value_max_history = np.array([])
7          self.value_avg_history = np.array([])
8          self.max_initial_cube = None
9          self.max_final_cube = None
10
11     def __update_history(self, population: List) -> None:
12         """
13             Memperbarui histori objective value maksimum dan rata-rata
14
15             Args:
16                 population (list): populasi saat ini
17             """
18
19             total_value = sum([cube.value for cube in population])
20             self.value_avg_history = np.append(self.value_avg_history, total_value / len(population))
21             self.value_max_history = np.append(self.value_max_history, population[0].value)
22
23     def __roulette_wheel(self, population: List, fitness_sum: int) -> List:
24         """
25             Membuat list roulette wheel berdasarkan populasi
26
27             Args:
28                 population (list): daftar individu dalam populasi
29                     populasi terurut berdasarkan fitness dari yang terbaik
30                 fitness_sum (int): total fitness dari populasi
31
32             Returns:
33                 list: list roulette wheel
34             """
35
36             roulette_wheel = []
37             cumulative_probability = 0
38
39             for cube in population:
40                 probability = (cube.fitness / fitness_sum) if fitness_sum > 0 else (1 / len(population))
41                 cumulative_probability += probability
42                 roulette_wheel.append((cumulative_probability, cube))
43
44             return roulette_wheel
```

```

1  def __selection(self, roulette_wheel: list) -> tuple[MagicCube, MagicCube]:
2      """
3          Seleksi dua individu dari roulette wheel
4
5          Args:
6              roulette_wheel (list): list roulette wheel
7
8          Returns:
9              tuple[MagicCube, MagicCube]: dua individu yang terpilih
10         """
11     parent1 = parent2 = None
12
13     for _ in range(2):
14         random_number = random.random()
15         for probability, cube in roulette_wheel:
16             if random_number <= probability:
17                 if parent1 is None:
18                     parent1 = cube
19                 else:
20                     parent2 = cube
21                 break
22
23     return parent1, parent2
24
25 def __mutation(self, individu: MagicCube, type: int = 1) -> None:
26     """
27         Mutasi individu
28
29         Args:
30             individu (MagicCube): individu yang akan dimutasi
31             type (int): tipe mutasi
32                 1: swap
33                 2: scramble
34                 3: inverse
35         """
36
37         # mutasi swap
38         if type == 1:
39             a = tuple(np.random.randint(5, size=3))
40             b = tuple(np.random.randint(5, size=3))
41             individu.swap(a, b)
42
43         # mutasi scramble
44         elif type == 2:
45             start = np.random.randint(125)
46             end = np.random.randint(start, min(126, start+50))
47             individu.scramble(start, end)
48
49         # mutasi inverse
50         elif type == 3:
51             start = np.random.randint(125)
52             end = np.random.randint(start, min(126, start+50))
53             individu.inverse(start, end)
54
55     return

```

```
1 def __crossover(self, parent1: MagicCube, parent2: MagicCube) -> tuple[MagicCube, MagicCube]:
2     """
3         Crossover dua individu
4         Crossover yang dilakukan adalah Order Crossover
5         Crossover dilakukan dengan mengubah parent menjadi array 1 dimensi dan melakukan crossover pada array tersebut
6
7     Args:
8         parent1 (MagicCube): individu pertama
9         parent2 (MagicCube): individu kedua
10
11    Returns:
12        tuple[MagicCube, MagicCube]: dua individu hasil crossover
13        """
14    # Pilih titik potong secara acak
15    point = np.random.randint(125)
16
17    # Ubah ke 1 dimensi
18    parent1_cube = parent1(cube.flatten())
19    parent2_cube = parent2(cube.flatten())
20    child1_cube = np.zeros(125)
21    child2_cube = np.zeros(125)
22
23    # Crossover
24    child1_cube[:point] = parent1_cube[:point]
25    child2_cube[:point] = parent2_cube[:point]
26
27    # Cari elemen yang belum ada pada child
28    idx1 = 0
29    idx2 = 0
30    for i in range(point, 125):
31        while parent2_cube[idx2] in child1_cube[:i]:
32            idx2 += 1
33        while parent1_cube[idx1] in child2_cube[:i]:
34            idx1 += 1
35        child1_cube[i] = parent2_cube[idx2]
36        child2_cube[i] = parent1_cube[idx1]
37        idx1 += 1
38        idx2 += 1
39
40    # Ubah kembali ke 3 dimensi
41    child1 = MagicCube()
42    child1(cube = child1_cube.reshape(5,5,5))
43    child1.refresh()
44
45    child2 = MagicCube()
46    child2(cube = child2_cube.reshape(5,5,5))
47    child2.refresh()
48
49    return child1, child2
```

```

1  def run(self,
2         population: list,
3         population_size: int,
4         max_generation: int
5         ) -> tuple[float, int]:
6     """
7     Menjalankan algoritma genetika
8
9     Args:
10        population (list): populasi awal
11        population_size (int): ukuran populasi
12        max_generation (int): maksimum generasi
13
14    Returns:
15        tuple: waktu eksekusi, generasi terakhir
16    """
17    current_generation = population
18    generation = 0
19    patient = 0 # jumlah generasi tanpa perubahan fitness terbaik
20    total_fitness = sum([cube.fitness for cube in current_generation])
21    self._update_history(current_generation)
22    self.max_initial_cube = current_generation[0]
23
24    max_stuck = 0.1 * max_generation
25    batas_mutasi_rendah = 0.05 if population_size >= 100 else 0.01
26    batas_mutasi_tinggi = 0.1 if population_size >= 100 else 0.05
27    batas_elitisme = 0.05
28
29    start = time.time()
30    while (current_generation[0].fitness < 109) and (generation < max_generation):
31        print(f"Generasi ke-{generation}, fitness terbaik: {current_generation[0].fitness}, patient: {patient}")
32
33        next_generation = []
34
35        # Roulette wheel
36        roulette_wheel = self._roulette_wheel(current_generation, total_fitness)
37
38        # Elitisme
39        if patient < max_stuck:
40            next_generation.extend(current_generation[:int(batas_elitisme * population_size)])
41
42        # Seleksi dan Crossover
43        for _ in range((population_size - len(next_generation)) // 2):
44            parent1, parent2 = self._selection(roulette_wheel)
45            child1, child2 = self._crossover(parent1, parent2)
46            next_generation.extend([child1, child2])
47
48        # Mutasi
49        if patient < max_stuck: # mutasi rendah
50            for i in range(len(next_generation)): # Lewati individu terbaik
51                if np.random.random() < batas_mutasi_rendah:
52                    self._mutation(next_generation[i], np.random.randint(1, 4))
53                else: # mutasi tinggi
54                    for i in range(len(next_generation)): # Lewati individu terbaik
55                        if np.random.random() < batas_mutasi_tinggi:
56                            self._mutation(next_generation[i], np.random.randint(1, 4))
57
58        # Update generasi
59        current_generation = next_generation
60        current_generation.sort(key=lambda x: x.fitness, reverse=True)
61        total_fitness = sum([cube.fitness for cube in current_generation])
62        self._update_history(current_generation)
63
64        # Update patient
65        if self.value_max_history[-1] == self.value_max_history[-2]:
66            patient += 1
67        else:
68            patient = 0
69
70        generation += 1
71    end = time.time()
72
73    self.max_final_cube = current_generation[0]
74    return end - start, generation

```

5. Kelas Visualization

Class Visualization	
<i>Atribut</i>	
<i>Method</i>	
plot(objective_value: list, message: str, initial_cube: MagicCube, final_cube: MagicCube, avg_objective_value: list) → None	Membuat plot hasil pencarian.
visualize_3d_cube(ax: plt.Axes3D, magic_cube: MagicCube, message: str) → None	Membuat visualisasi 3D dari <i>magic cube</i> .
<i>Source Code</i>	

```

1 class Visualization:
2
3     @staticmethod
4     def plot(objective_value: List, message: str,
5             initial_cube: MagicCube, final_cube: MagicCube,
6             avg_objective_value: List = None, euler_value: List = None) -> None:
7         """
8             membuat plot hasil pencarian
9
10            Args:
11                objective_value (list): list nilai objektif
12                message (str): keterangan plot
13                initial_cube (MagicCube): kubus awal
14                final_cube (MagicCube): kubus akhir
15                avg_objective_value (list): list rata-rata nilai objektif
16
17            # Membuat figure dengan gridspec dan mengatur rasio
18            fig = plt.figure(figsize=(15, 12))
19            if euler_value is None:
20                gs = fig.add_gridspec(3, 2, height_ratios=[30, 5, 65], width_ratios=[70, 30])
21                print("Euler Value is None")
22            else:
23                gs = fig.add_gridspec(3, 2, height_ratios=[30, 5, 65], width_ratios=[50, 50])
24
25            # Baris 1, Kolom 1: Grafik garis nilai objektif
26            ax1 = fig.add_subplot(gs[0, 0], zorder=3)
27            if avg_objective_value is not None:
28                ax1.plot(objective_value, label="Max Objective Value")
29                ax1.plot(avg_objective_value, label="Average Objective Value")
30            else:
31                ax1.plot(objective_value, label="")
32            ax1.set_title("Objective Value")
33            ax1.set_xlabel("Iteration")
34            ax1.set_ylabel("Objective Value")
35            ax1.legend()
36
37            if euler_value is not None:
38                axEuler = fig.add_subplot(gs[0, 1], zorder=3)
39                axEuler.plot(euler_value, 'o', linestyle='None', markersize=1)
40                axEuler.set_title("Euler Value")
41                axEuler.set_xlabel("Iteration")
42                axEuler.set_ylabel("Euler Value")
43                axEuler.legend()
44
45            # Baris 1, Kolom 2: Statistik algoritma genetika
46            array_stats_text = message
47            if euler_value is None:
48                ax2 = fig.add_subplot(gs[0, 1])
49                ax2.axis("off")
50                ax2.text(-0.5, 0.75, array_stats_text, ha="left", va="center", fontsize=10, fontdict={"family": "monospace"})
51            else:
52                ax2 = fig.add_subplot(gs[1, 0], zorder=2)
53                ax2.axis("off")
54                ax2.text(0, -1, array_stats_text, ha="left", va="center", fontsize=10, fontdict={"family": "monospace"})
55
56            # Baris 3: Visualisasi 3D kubus
57            gs_bottom = gs[2, :].subgridspec(1, 2, width_ratios=[50, 50])
58
59            # Baris 3, Kolom 1 dan 2: Plot 3D kubus
60            ax3 = fig.add_subplot(gs_bottom[0, 0], projection='3d')
61            Visualization.visualize_3d_cube(ax3, initial_cube, "Initial Cube")
62
63            ax4 = fig.add_subplot(gs_bottom[0, 1], projection='3d')
64            Visualization.visualize_3d_cube(ax4, final_cube, "Final Cube")
65
66            plt.tight_layout()
67            plt.subplots_adjust(hspace=0, wspace=0.2)
68            plt.show()

```

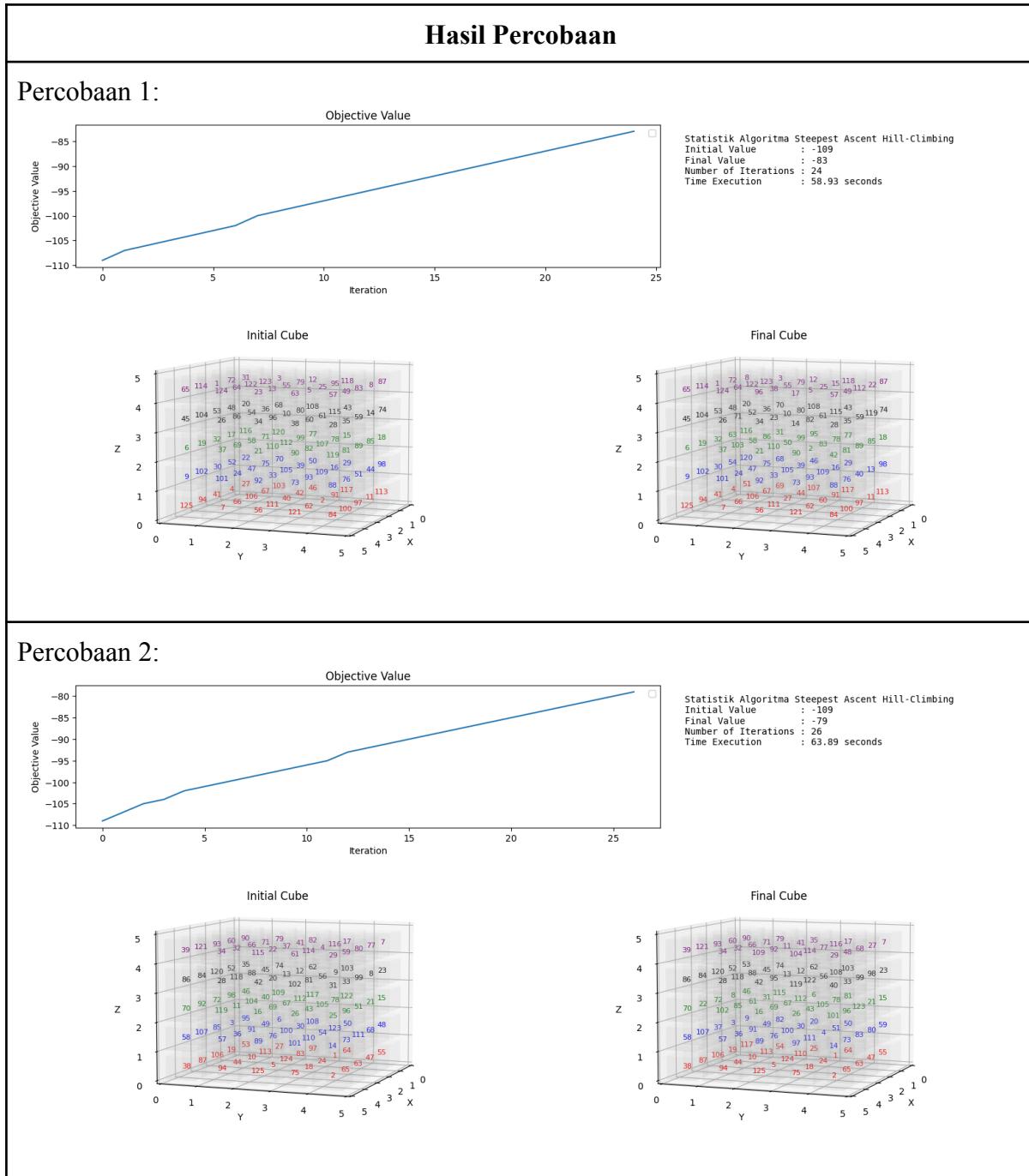
```
● ● ●
```

```
1  @staticmethod
2  def visualize_3d_cube(ax: plt.Axes, magic_cube: MagicCube, message: str) -> None:
3      """
4          Create 3D visualization of a magic cube 5x5x5
5
6      Params:
7          ax (matplotlib.axes._subplots.Axes3DSubplot): 3D axes
8          magic_cube (MagicCube): 5x5x5 array of numbers
9          message (str): title of the plot
10         """
11     for i in range(5):
12         for j in range(5):
13             for k in range(5):
14                 # Create white cube
15                 ax.bar3d(i, j, k, 0.8, 0.8, 0.8, color='white', alpha=0.05)
16
17                 # Add number in the middle of the cube
18                 value = str(int(magic_cube[i, j, k]))
19                 x_center, y_center, z_center = i + 0.4, j + 0.4, k + 0.4
20                 ax.text(x_center, y_center, z_center,
21                         value, color=colors[k],
22                         ha='center', va='center',
23                         fontsize=8)
24
25                 # Set labels and title
26                 ax.set_xlabel('X')
27                 ax.set_ylabel('Y')
28                 ax.set_zlabel('Z')
29                 ax.text(2, 2, 6, message, color='black', fontsize=12, ha='center')
30
31                 # Set axis limits
32                 ax.set_xlim([0, 5])
33                 ax.set_ylim([0, 5])
34                 ax.set_zlim([0, 5])
35
36                 # Set viewing angle
37                 ax.view_init(elev=7, azim=21)
38
39                 # Add grid
40                 ax.grid(True)
```

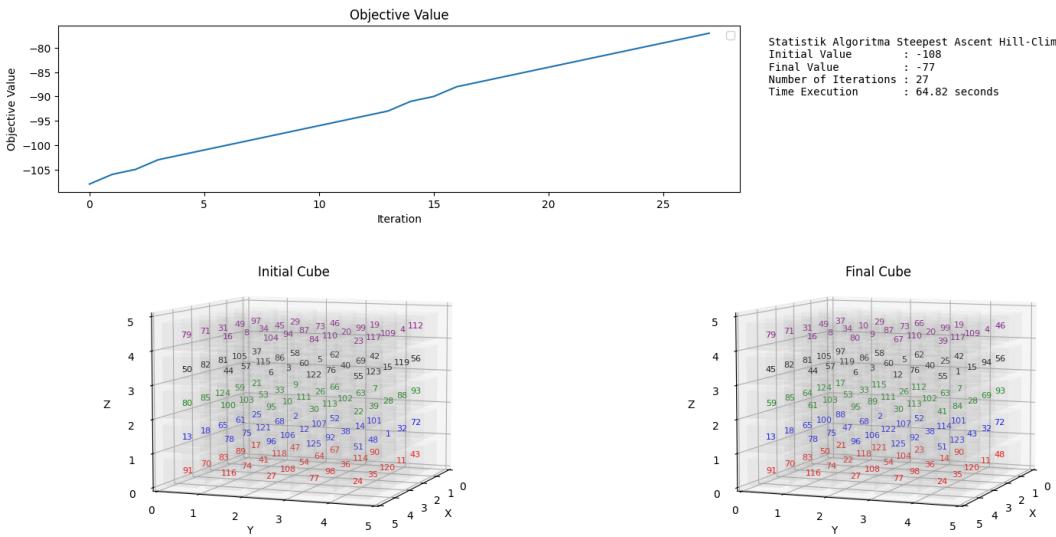
2.3 Hasil Percobaan dan Analisis

1. Steepest Ascent Hill-climbing

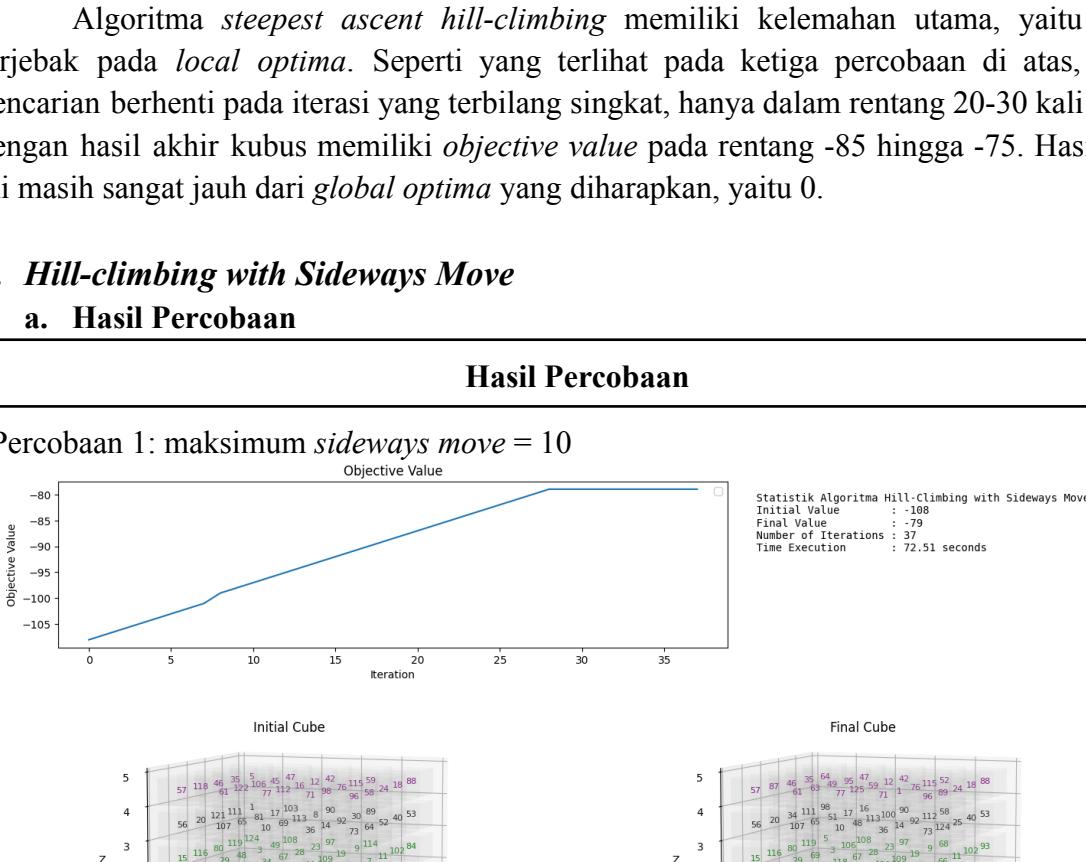
a. Hasil Percobaan



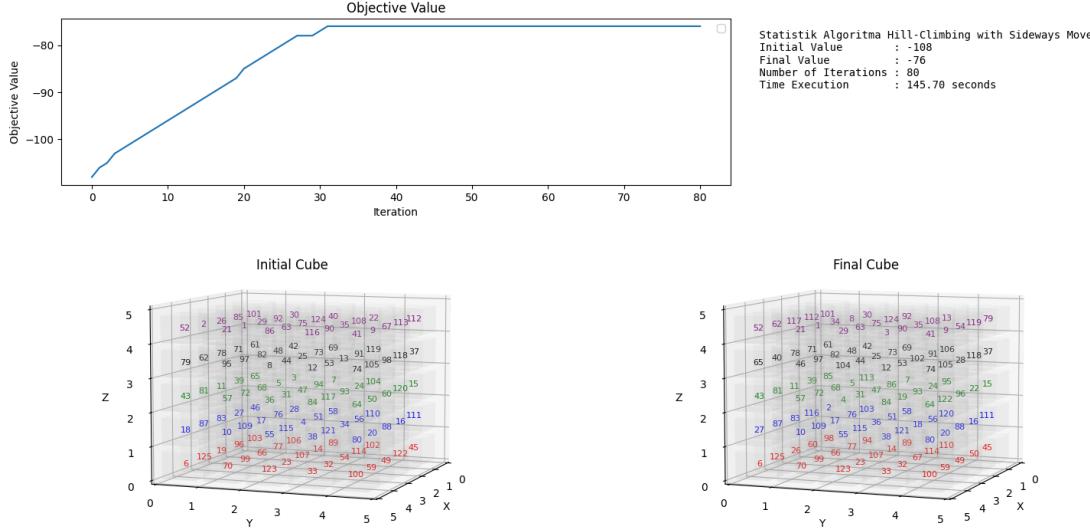
Percobaan 3:



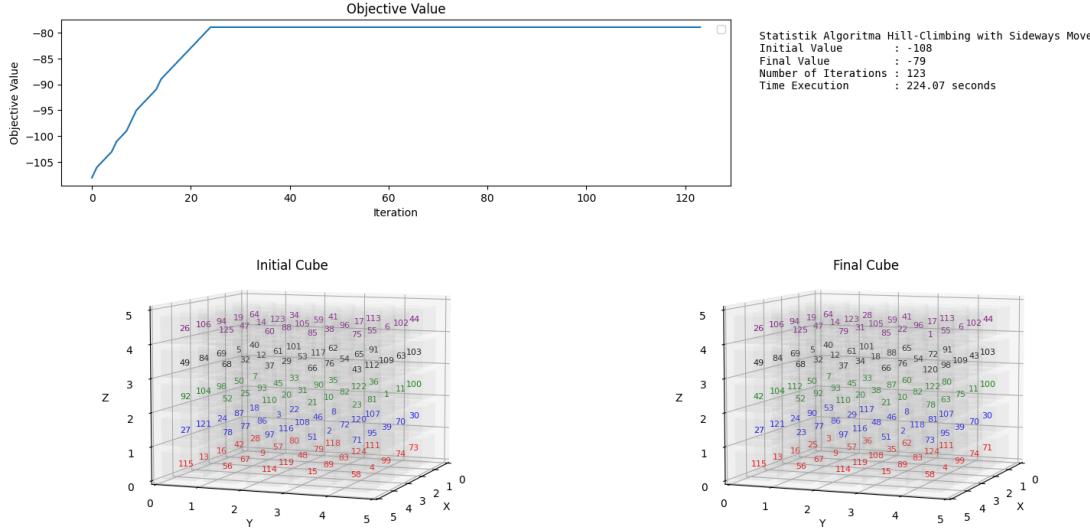
Percobaan 1: maksimum sideways move = 10



Percobaan 2: maksimum sideways move = 50



Percobaan 3: maksimum sideways move = 100

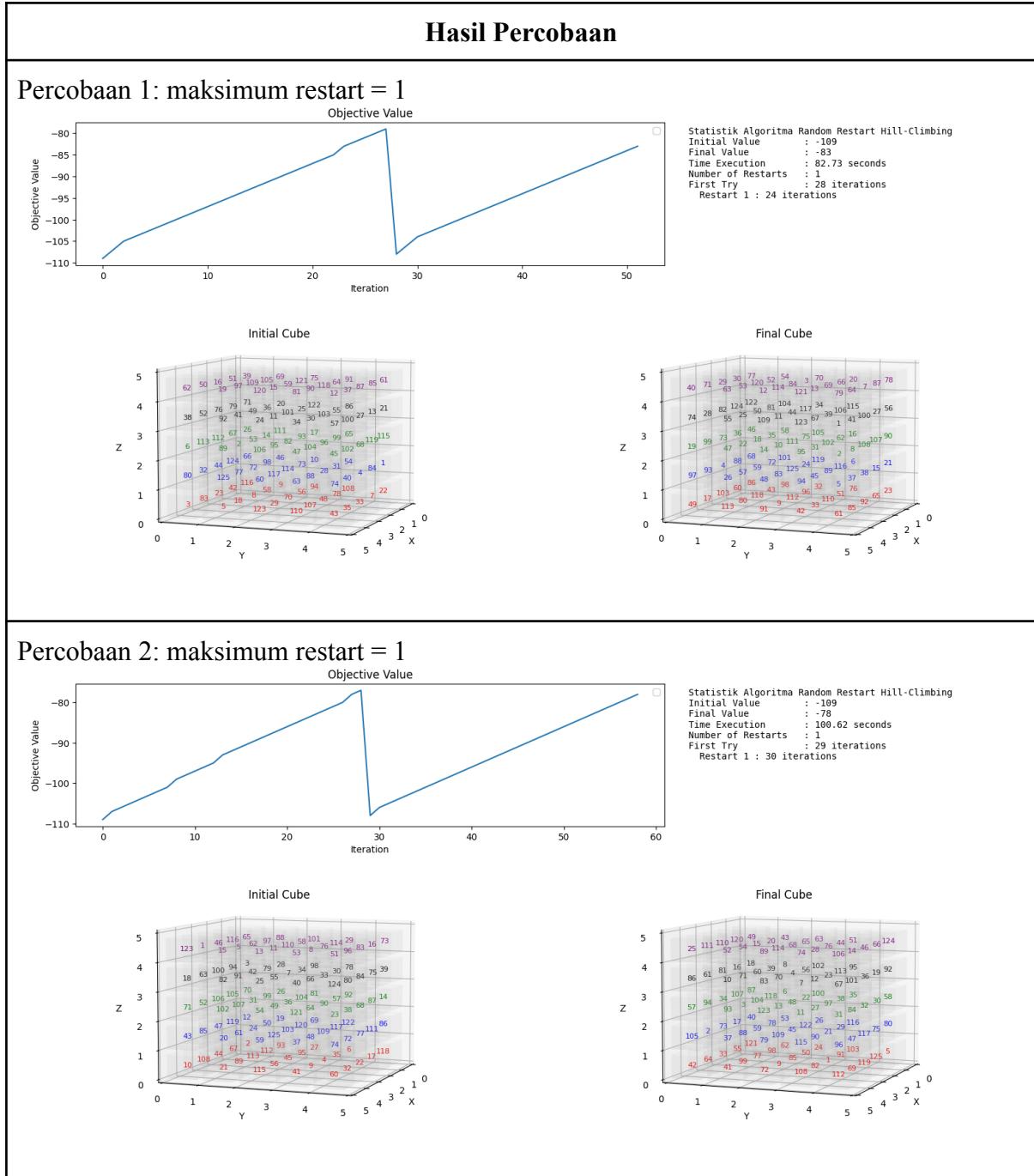


b. Analisis Hasil

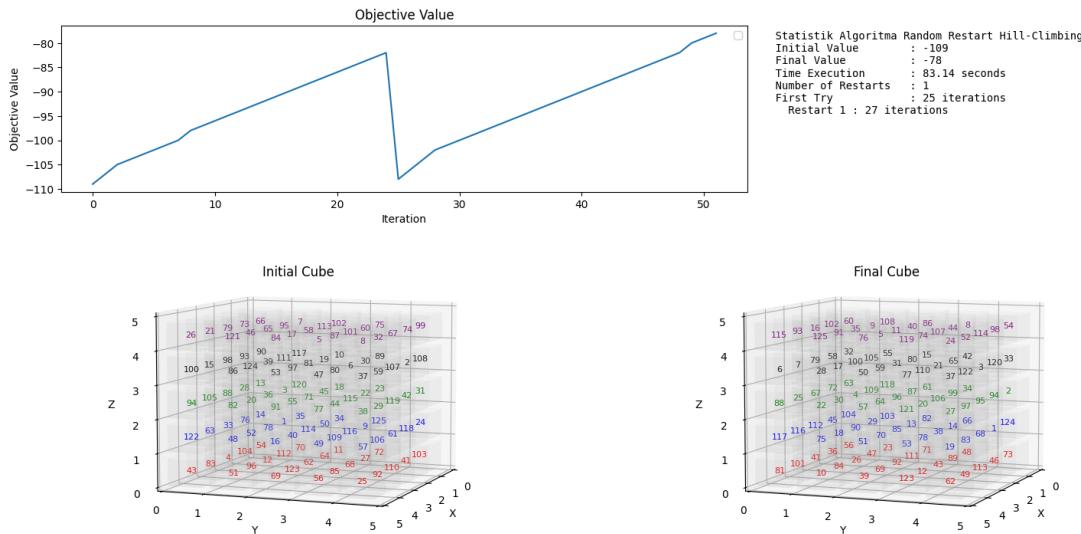
Algoritma *hill-climbing with sideways move* memiliki fleksibilitas yang sedikit lebih baik dibandingkan *steepest ascent hill-climbing* karena mengizinkan pergerakan ke nilai yang sama. Akan tetapi, sama seperti *steepest ascent hill-climbing*, algoritma ini rentan terjebak pada *local optima* sehingga gagal menemukan solusi persoalan. Adapun jika dilihat dari hasil percobaan di atas, dapat terlihat bahwa hasil percobaan *hill-climbing with sideways move* menghasilkan hasil yang tidak jauh berbeda dengan algoritma *steepest ascent hill-climbing*, hanya jumlah iterasi yang bertambah berdasarkan banyaknya *sideways move* maksimal yang diperbolehkan. Adapun objective value akhir yang diperoleh berada pada rentang -85 hingga -75, yang juga masih sangat jauh dari *global optima* yang diharapkan.

3. Random Restart Hill-climbing

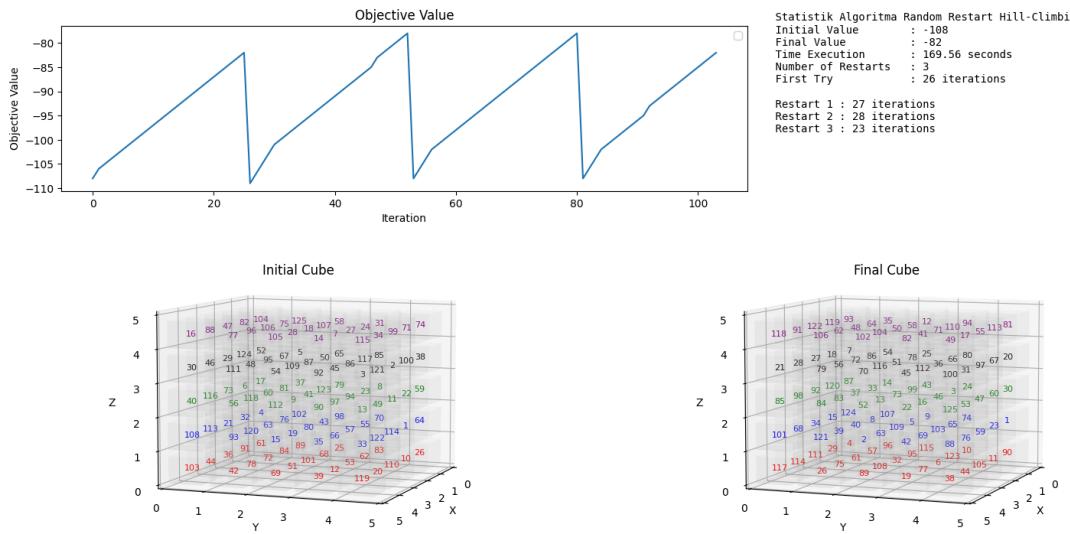
a. Hasil Percobaan



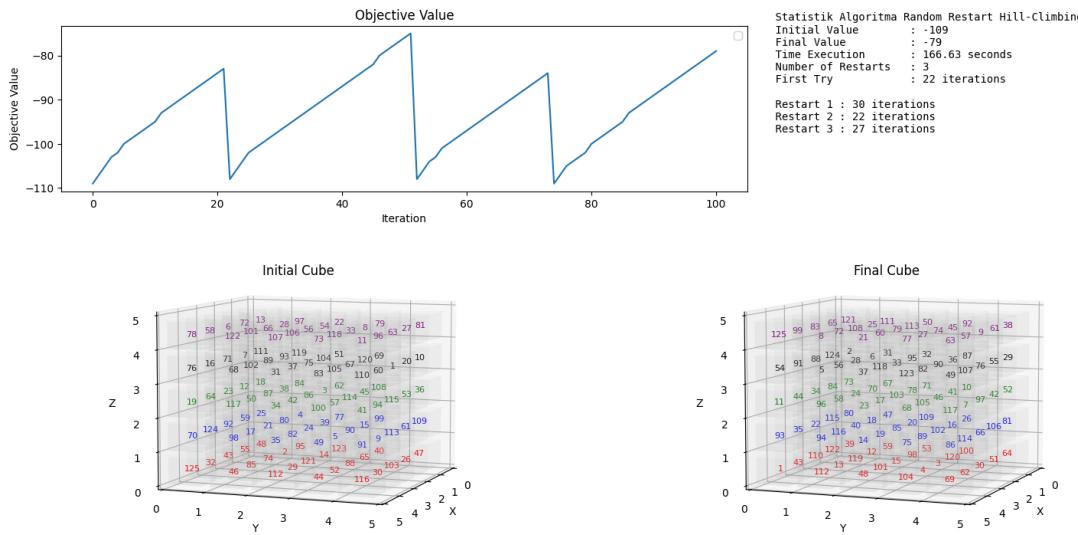
Percobaan 3: maksimum restart = 1



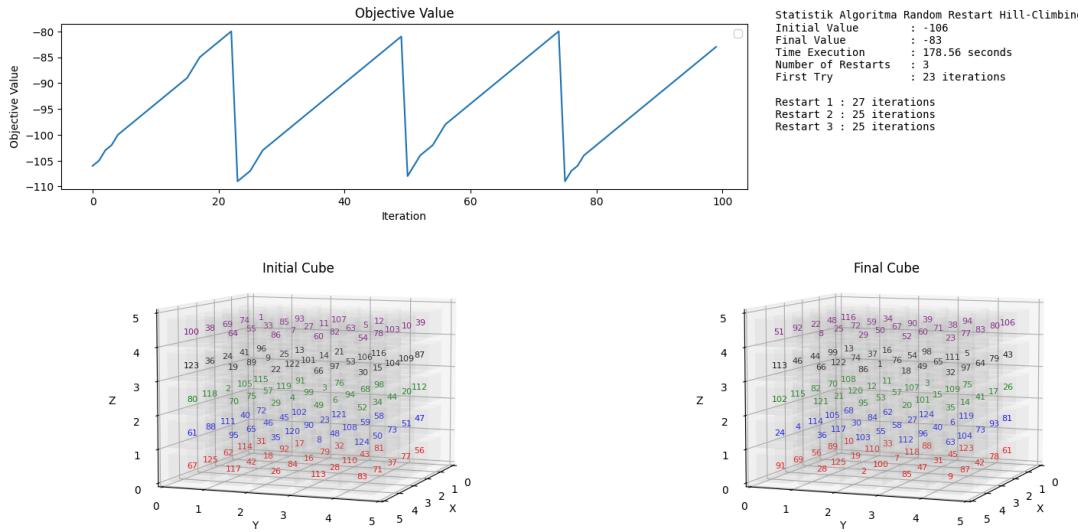
Percobaan 4: maksimum restart = 3



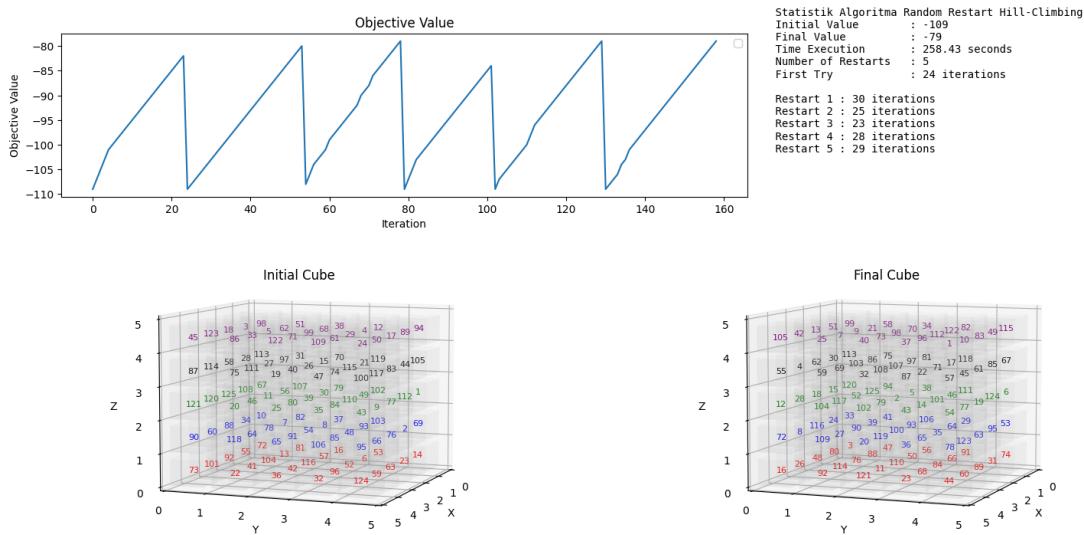
Percobaan 5: maksimum restart = 3



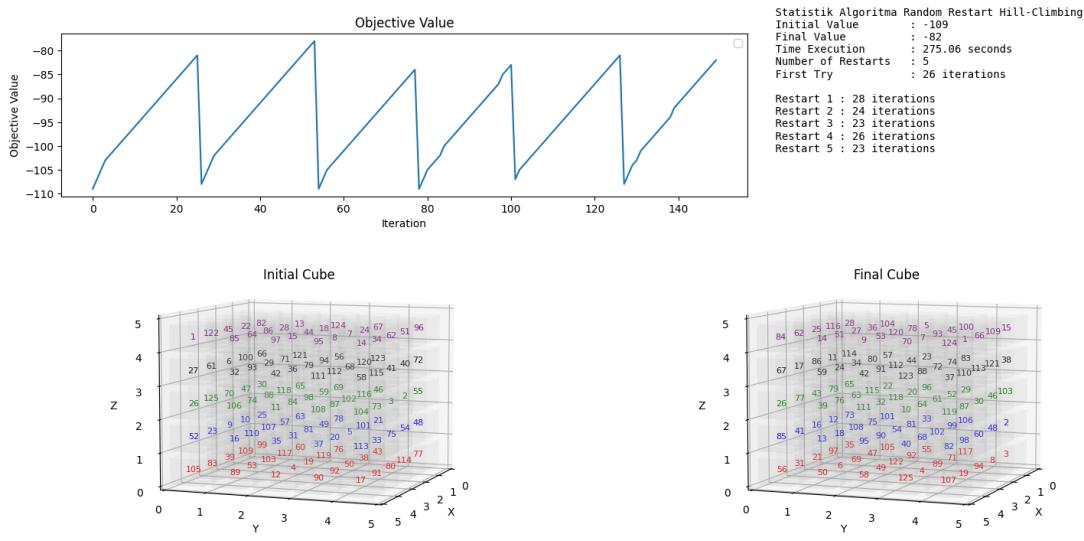
Percobaan 6: maksimum restart = 3



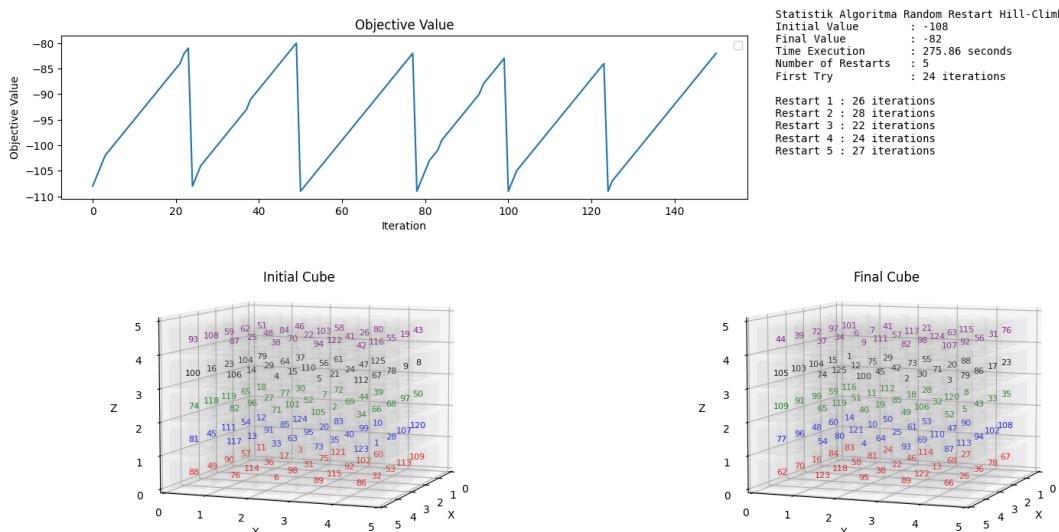
Percobaan 7: maksimum restart = 5



Percobaan 8: maksimum restart = 5



Percobaan 9: maksimum restart = 5



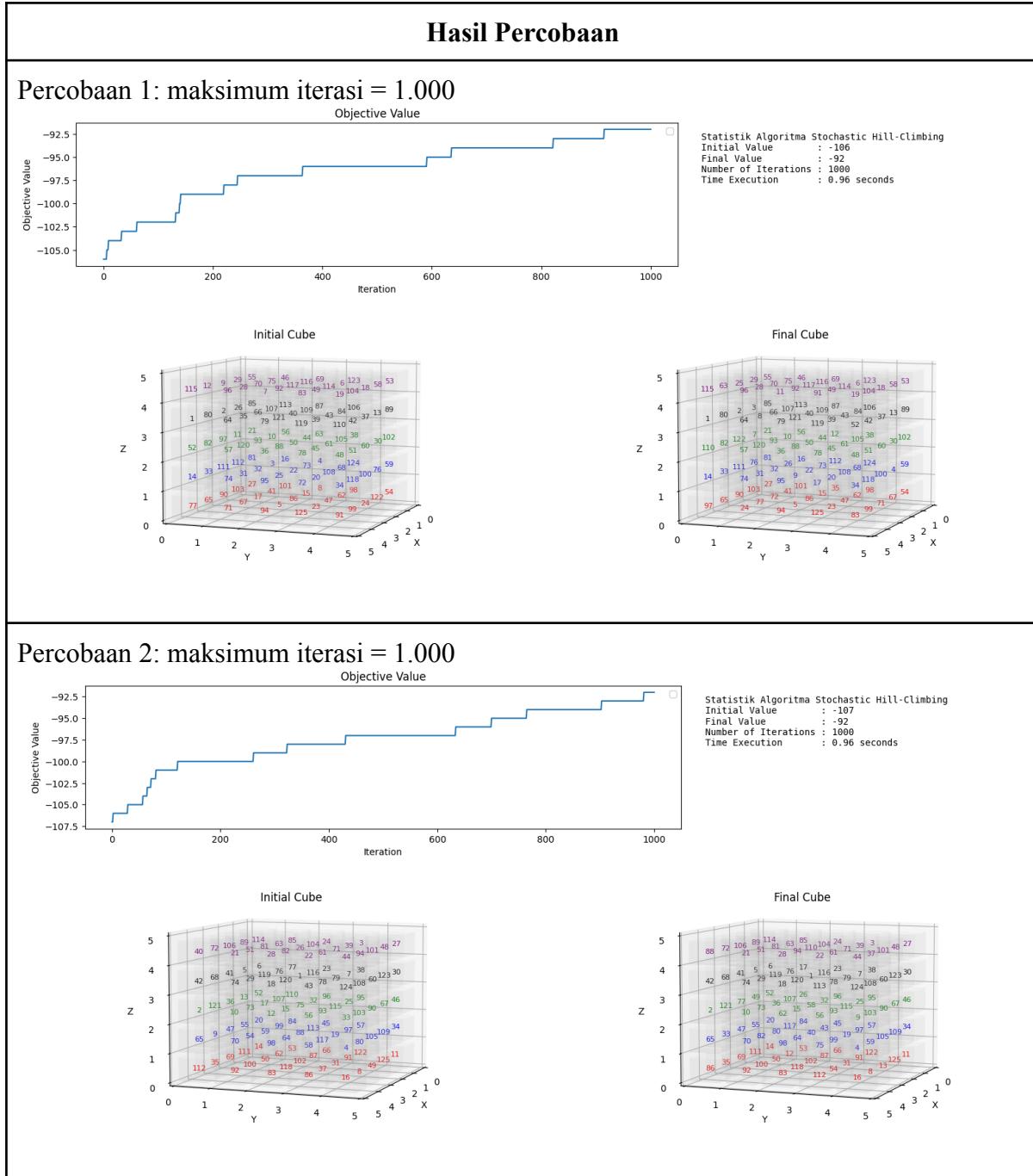
b. Analisis Hasil

Algoritma *random restart hill-climbing* tidak memberikan hasil yang cukup dekat dengan *global optima*. Hal ini disebabkan oleh kecilnya kemungkinan untuk mendapatkan konfigurasi susunan angka kubus yang mendekati *magic cube*. Meskipun terdapat elemen *restart* dalam algoritma ini, seringkali *state* awal setelah *restart* merupakan *state* yang memiliki *objective value* yang rendah. Akibatnya, hasil pencarian dengan algoritma ini sama saja dengan algoritma *steepest ascent hill-climbing*. Meskipun hasil pencarinya sama, waktu eksekusi algoritma ini lebih lama dari *steepest ascent hill-climbing* karena perlu melakukan *restart*.

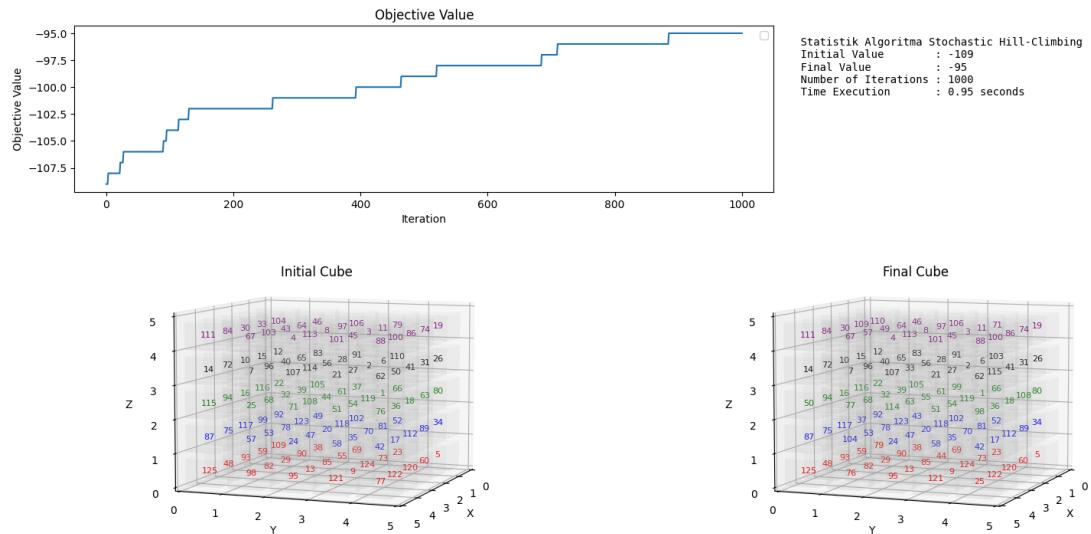
Dari beberapa kali percobaan yang telah dilakukan, hasil pencarinya cukup konsisten antara satu percobaan dengan percobaan lainnya, dengan rentang *objective value* berada pada nilai -85 hingga -75. Tidak ada penyimpangan yang signifikan.

4. Stochastic Hill-climbing

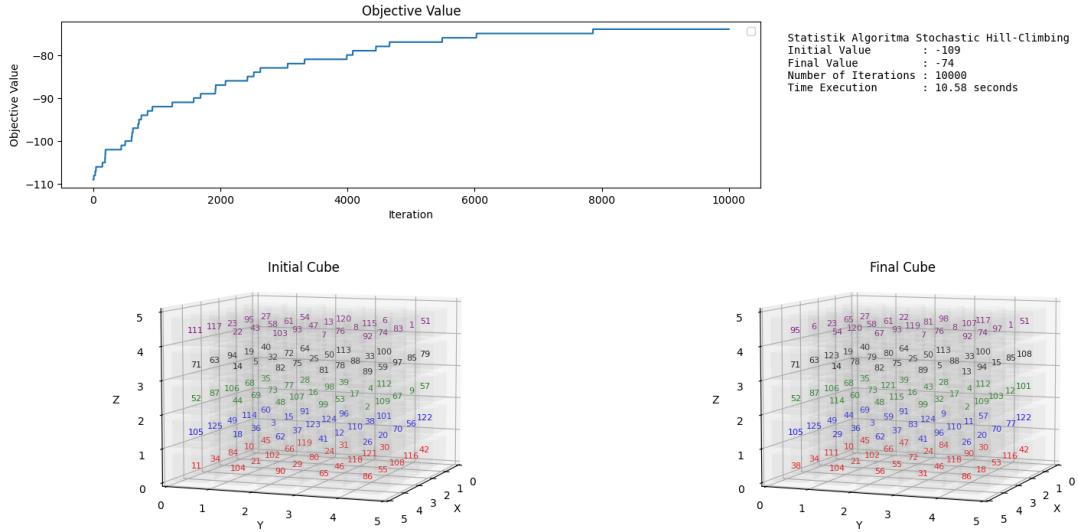
a. Hasil Percobaan



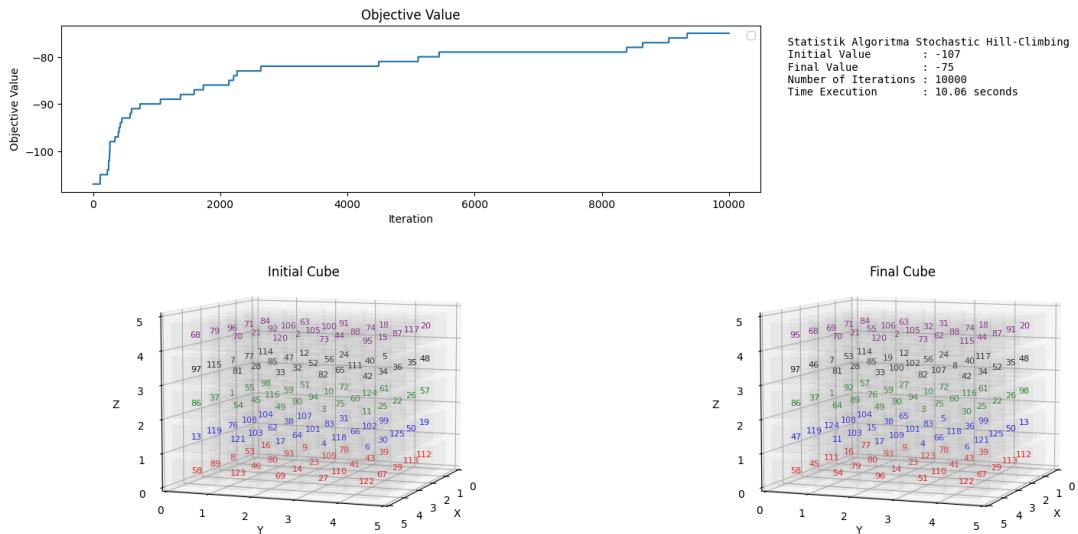
Percobaan 3: maksimum iterasi = 1.000



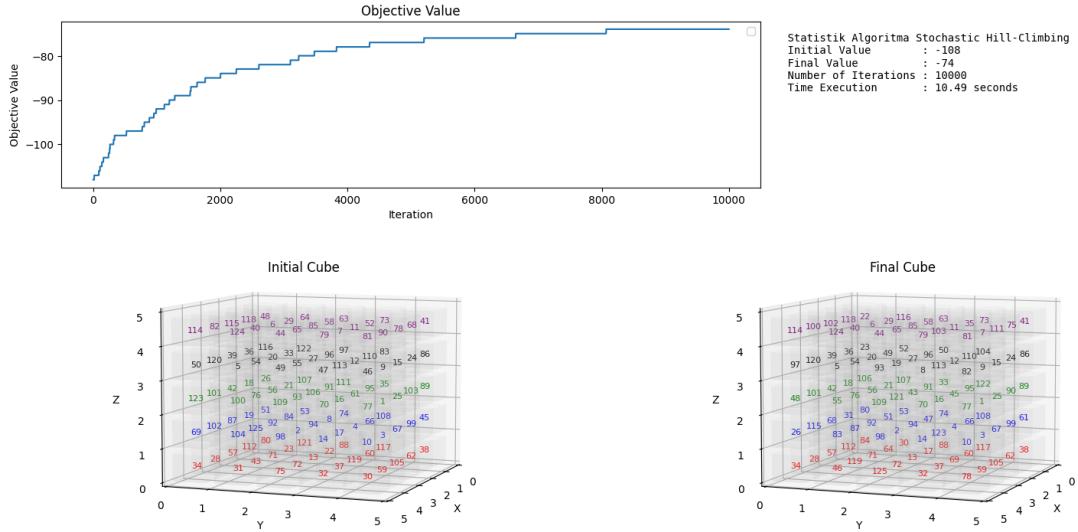
Percobaan 4: maksimum iterasi = 10.000



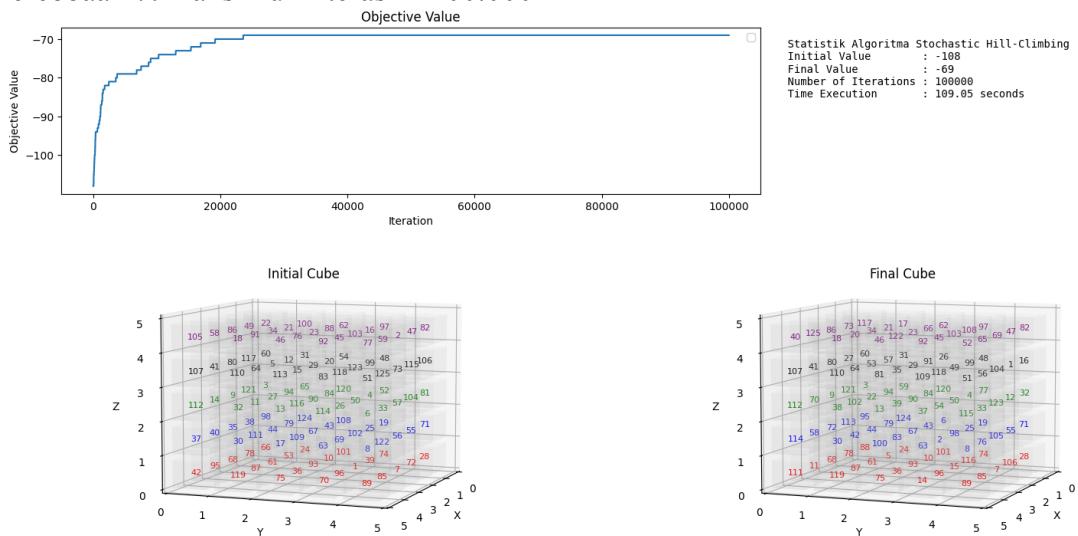
Percobaan 5: maksimum iterasi = 10.000



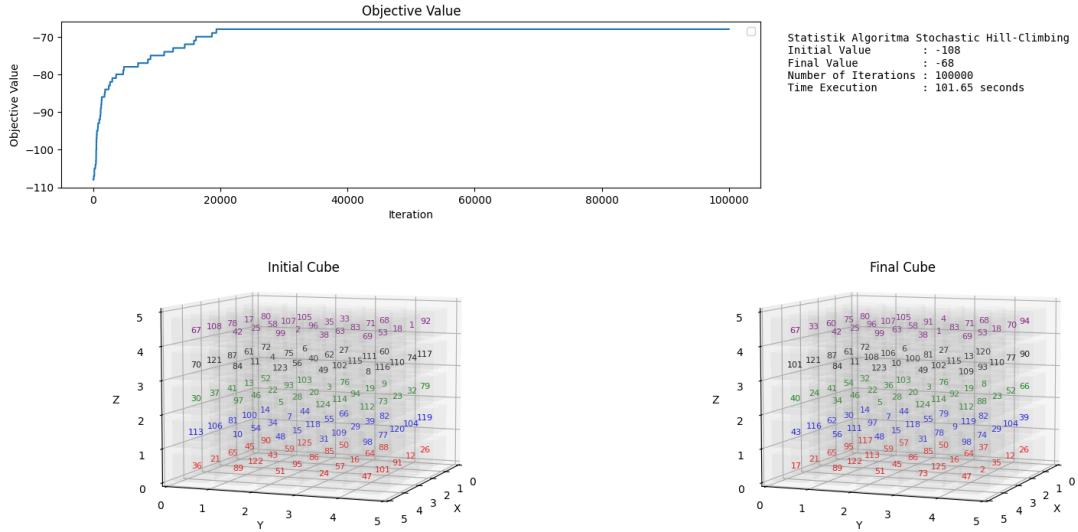
Percobaan 6: maksimum iterasi = 10.000



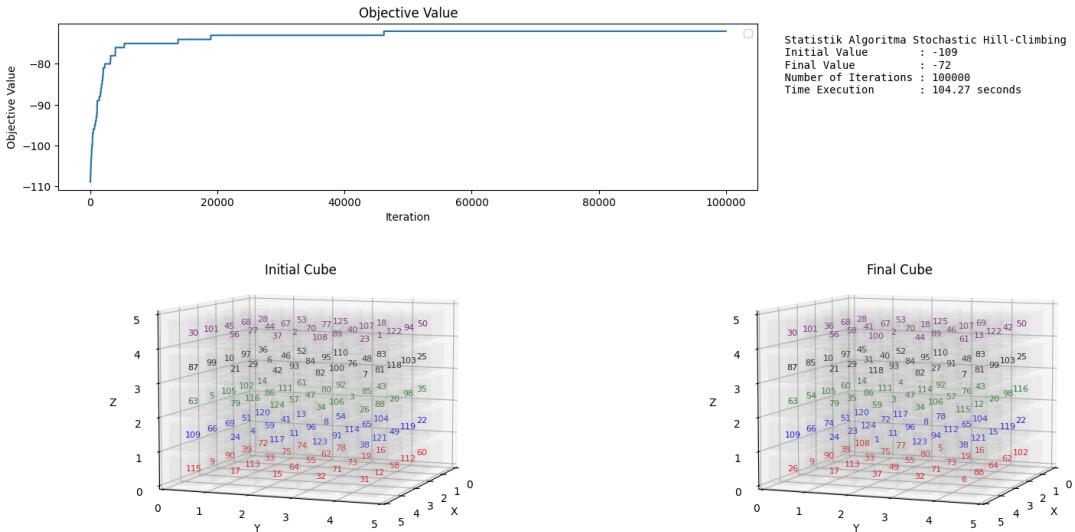
Percobaan 7: maksimum iterasi = 100.000



Percobaan 8: maksimum iterasi = 100.000



Percobaan 9: maksimum iterasi = 100.000



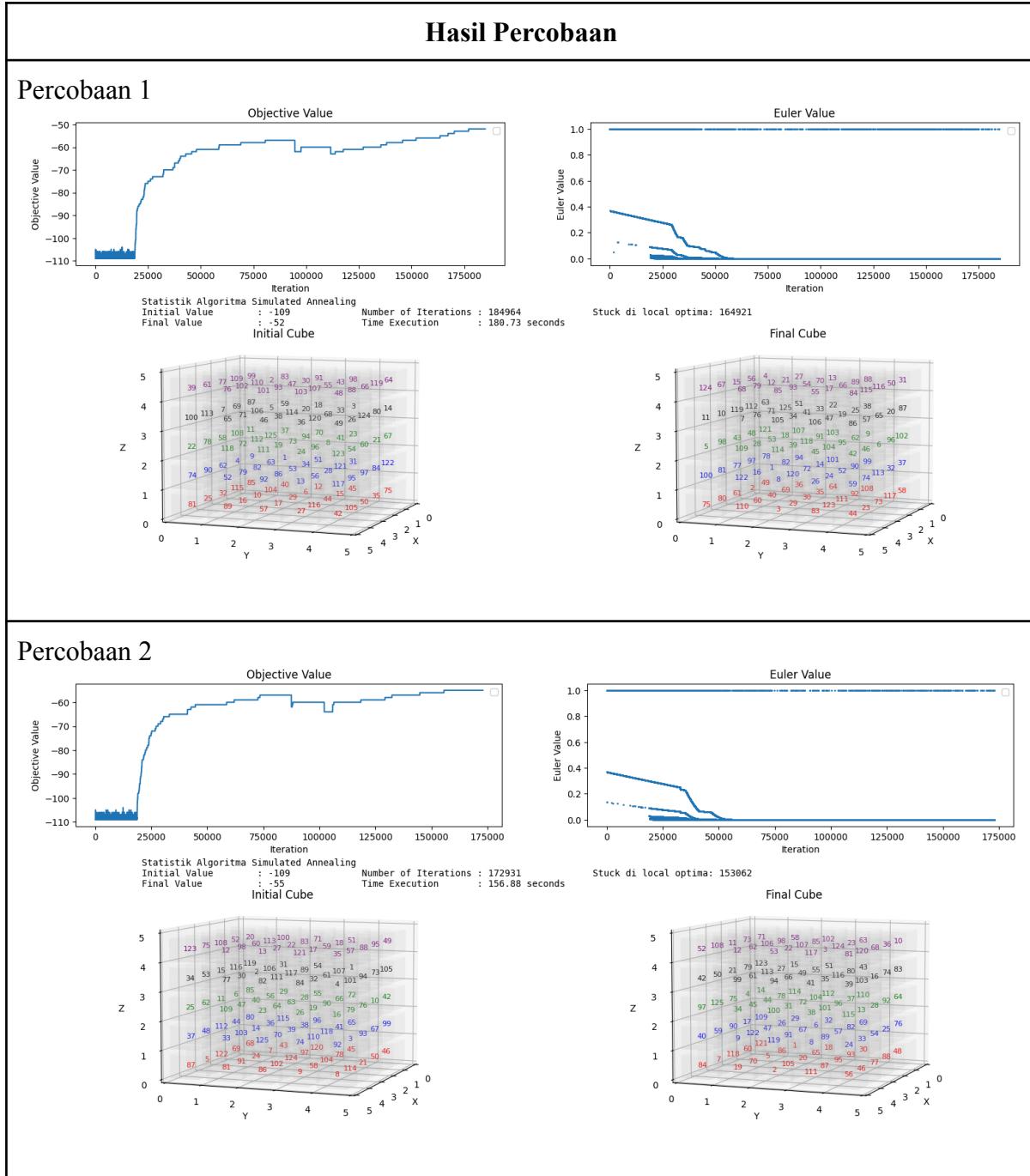
b. Analisis Hasil

Algoritma *stochastic hill-climbing* tidak memberikan hasil yang cukup dekat dengan *global optima*. Sama seperti *steepest ascent hill-climbing*, hal ini disebabkan oleh kecilnya kemungkinan untuk mendapatkan konfigurasi susunan angka kubus yang mendekati *magic cube*. Hasil pencarian dengan algoritma ini sedikit lebih baik dibandingkan dengan algoritma *steepest ascent hill-climbing*. Selain itu, waktu eksekusi algoritma ini juga lebih baik daripada *steepest ascent hill-climbing* karena pemilihan *neighbor* yang acak, sehingga tidak perlu mencari nilai maksimum dari seluruh *successor* yang ada.

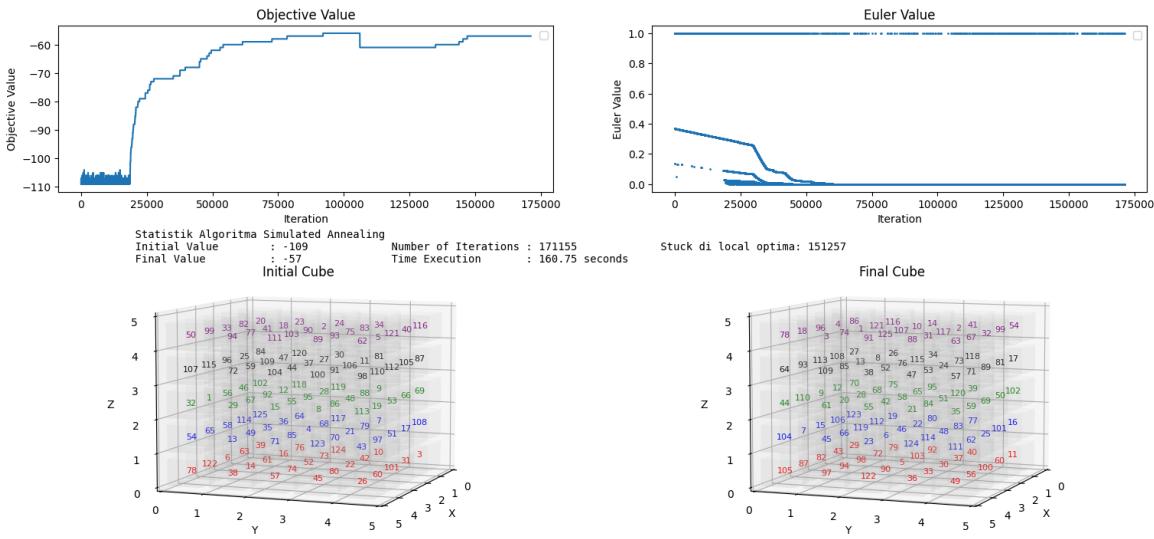
Dari beberapa kali percobaan yang telah dilakukan, hasil pencarinya cukup konsisten untuk nilai maksimum iterasi yang sama. Adapun saat nilai maksimum iterasi ditambah, terjadi kenaikan nilai *objective value* hasil pencarian. Akan tetapi, pada percobaan 100.000 iterasi, mulai dari iterasi 20.000 ke atas, pencarian selalu terjebak di *local optima*.

5. Simulated Annealing

a. Hasil Percobaan



Percobaan 3



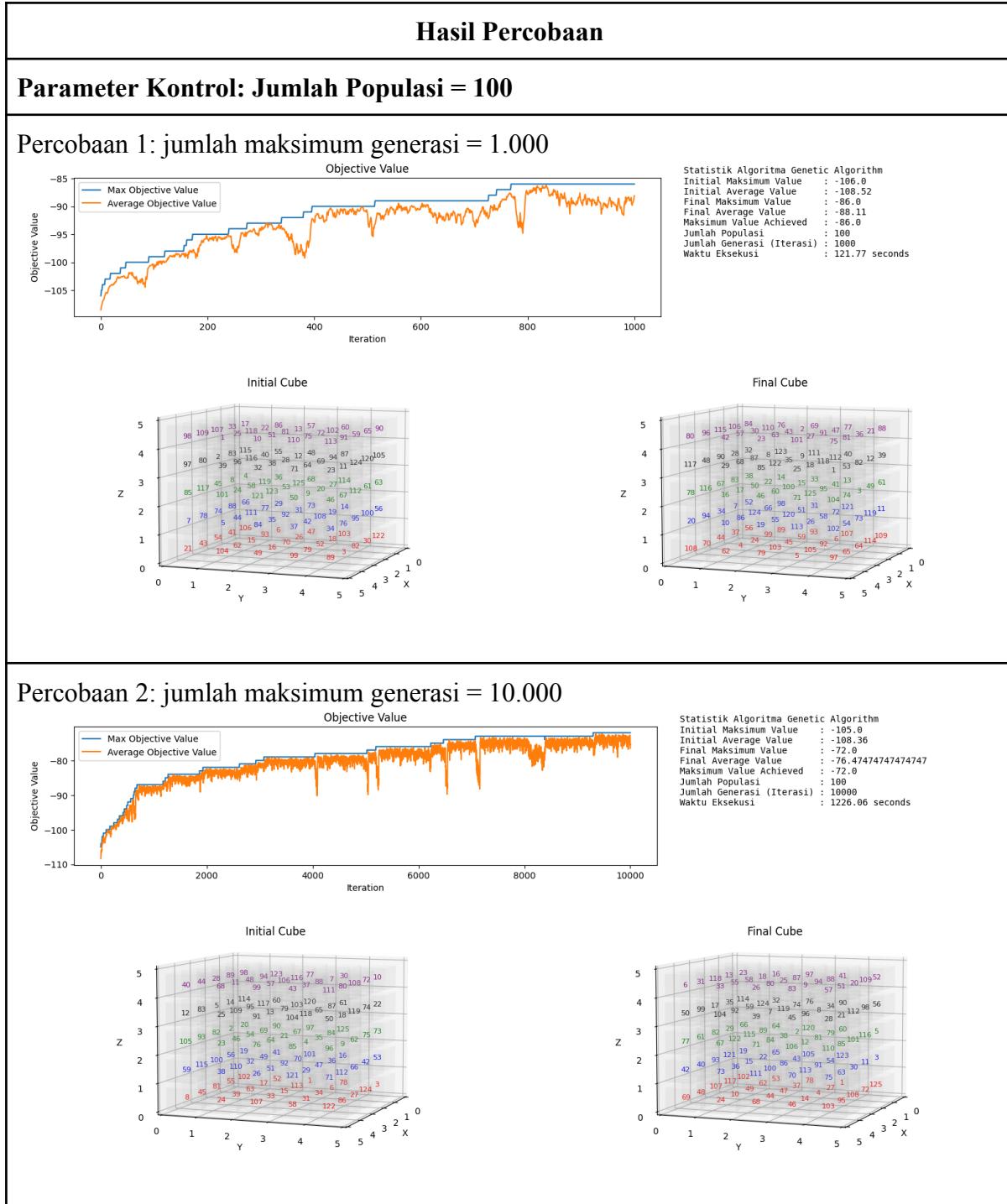
b. Analisis Hasil

Dari hasil percobaan, *simulated annealing* memiliki performa yang lebih baik jika dibandingkan dengan algoritma-algoritma sebelumnya. Namun pada 20.000 iterasi awal dari ketiga percobaan, terlihat bahwa nilai *objective value* mengalami fluktuasi di nilai -109 dan -105. Hal ini menandakan bahwa pada periode tersebut, pencarian masih memperbolehkan berpindah ke state yang lebih buruk. Hal ini dibuktikan dari *plot* probabilitas euler, dimana nilai probabilitas *state* untuk iterasi di bawah 20.000 berada di atas batas yang ditetapkan, sehingga perpindahan ke *state* yang lebih buruk dimungkinkan. Namun, di atas iterasi 20.000, nilai probabilitas euler berada di bawah batas yang ditetapkan, dan proses pencarian pun mulai melarang berpindah ke *state* yang lebih buruk, mengakibatkan peningkatan tajam pada *plot objective value*.

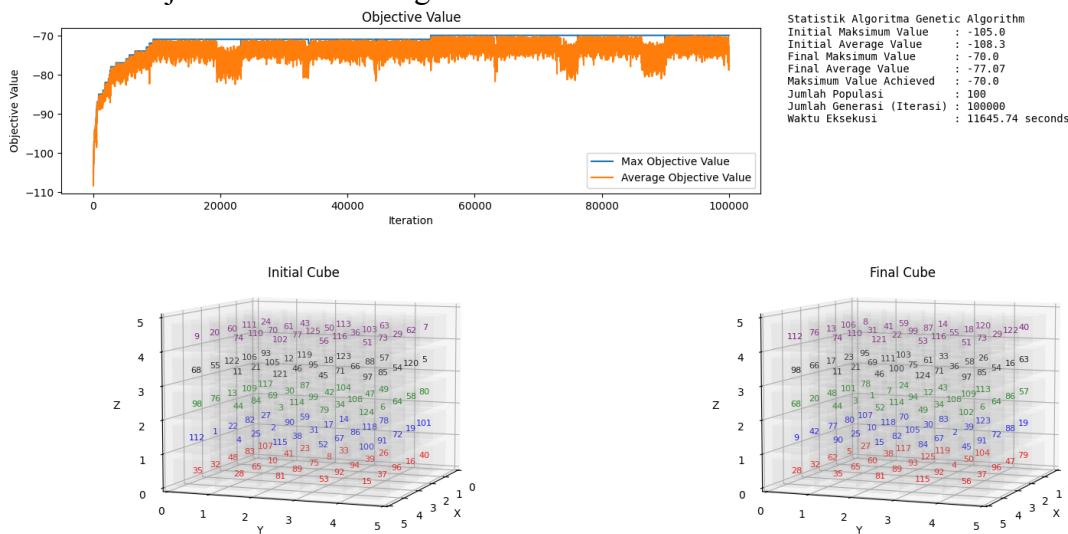
Dari ketiga percobaan di atas, terlihat bahwa nilai *objective value* dari hasil akhir pencarian menggunakan *simulated annealing* konsisten pada rentang -60 hingga -50, yang masih cukup jauh dari nilai *global optima*.

6. Genetic Algorithm

a. Hasil Percobaan

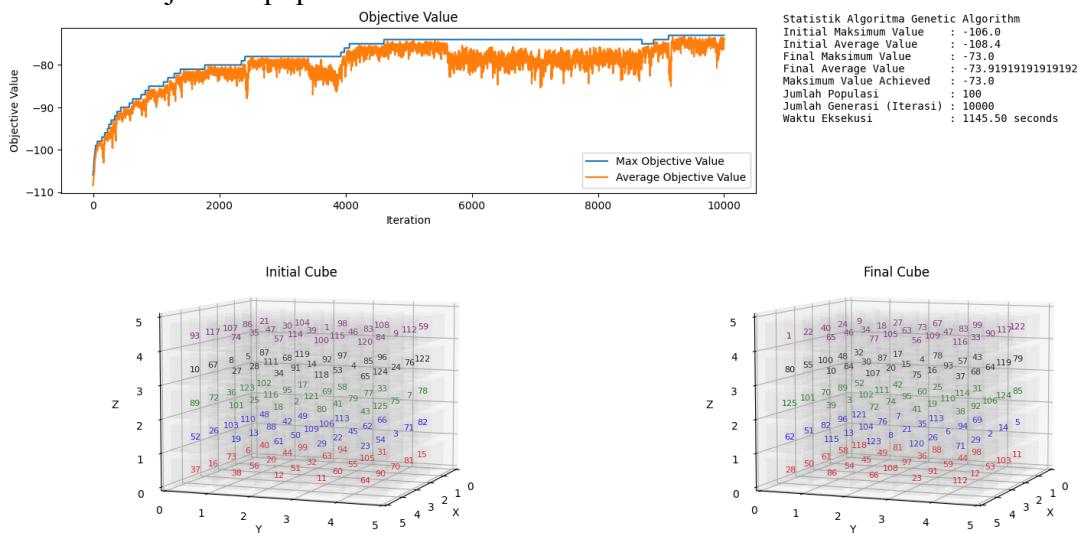


Percobaan 3: jumlah maksimum generasi = 100.000

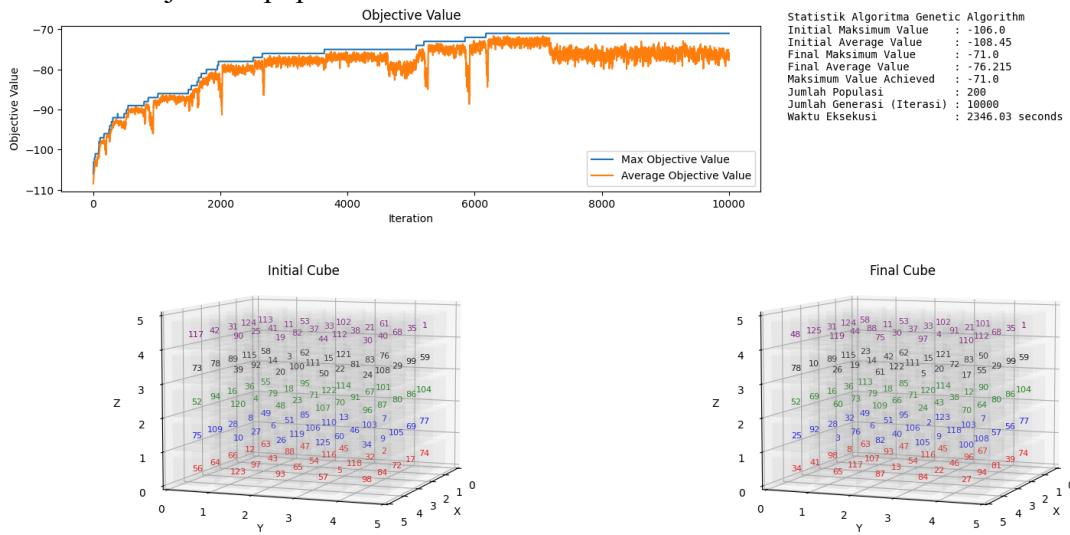


Parameter Kontrol: Jumlah Maksimum Generasi = 10.000

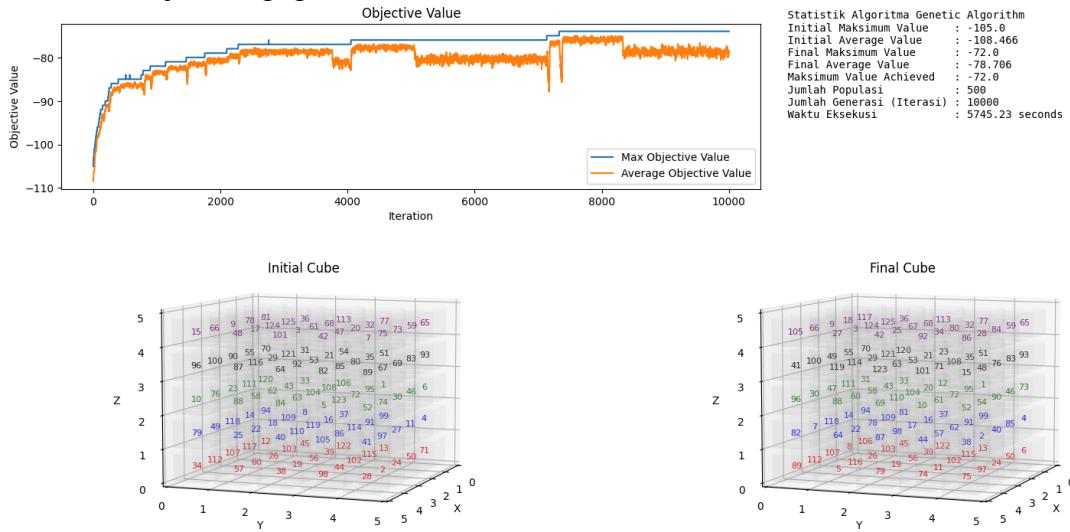
Percobaan 1: jumlah populasi = 100



Percobaan 2: jumlah populasi = 200



Percobaan 3: jumlah populasi = 500



b. Analisis Hasil

Berdasarkan hasil percobaan di atas, bisa dilihat nilai akhir terbaik yang diperoleh adalah sekitar -70, yang masih sangat jauh dari *global optima* yang diharapkan. Hal ini terjadi karena *genetic algorithm* sering kali tidak mampu untuk mencapai *global optima* dan sering terjebak di *local optima* dalam masalah dengan ruang pencarian yang sangat besar. Hal ini dibuktikan dari *plot* maksimum *objective value* dan rata-rata *objective value* per generasi yang lebih banyak menghabiskan waktu pada daerah datar (mengalami stagnasi) untuk jumlah generasi yang besar.

Konsistensi hasil dapat diukur dari pola nilai akhir yang dihasilkan berdasarkan variasi parameter. Pada percobaan dengan maksimum generasi yang meningkat (1.000, 10.000, 100.000) untuk populasi tetap (100), hasilnya menunjukkan bahwa penambahan generasi membantu dalam mencapai nilai akhir yang lebih baik. Begitu pula pada percobaan dengan populasi yang meningkat (100, 200, 500) untuk generasi tetap (10.000), terlihat bahwa hasil akhir terdapat peningkatan, namun peningkatan yang terjadi tidak terlalu signifikan dibandingkan dengan saat jumlah generasi yang ditingkatkan dan juga terdapat fluktuasi kecil.

2.4 Analisis Kinerja Algoritma terhadap Pencarian Solusi *Diagonal Magic Cube*

1. Analisis Solusi Persoalan *Diagonal Magic Cube* $5 \times 5 \times 5$

Persoalan *diagonal magic cube* $5 \times 5 \times 5$, merupakan persoalan dengan ruang pencarian yang sangat besar. Mengingat terdapat 125 bilangan berbeda yang bisa dipertukarkan, akan terdapat hingga $125!$ susunan kubus yang mungkin. Adapun hingga saat ini, baru terdapat satu solusi susunan kubus yang berhasil ditemukan. Dengan 4 rotasi pada sumbu-z dan sisi kubus yang berjumlah 6, satu solusi tersebut dapat dilakukan rotasi sehingga memberikan representasi yang berbeda untuk solusi yang sama ketika direpresentasikan sebagai *array* di dalam program komputer. Dengan memilih 1 sisi yang menghadap ke atas, maka banyaknya kemungkinan rotasi yang mungkin adalah:

$$6 (\text{sisi yang menghadap ke atas}) \times 4 (\text{rotasi di sekitar sumbu z}) = 24 \text{ solusi}$$

Jadi jika dilakukan pencarian dengan menggunakan program komputer, kita hanya akan memiliki 24 solusi berbeda yang merujuk ke susunan kubus yang sama. jadi dari $125!$ susunan kubus yang mungkin, hanya terdapat 24 susunan kubus yang merupakan solusi *diagonal magic cube*.

Pada persoalan ini, pencarian memiliki rentang *nilai objective* yang sempit relatif terhadap banyaknya susunan kubus, yaitu -109 hingga 0. Akibatnya, akan terdapat begitu banyak kubus dengan nilai yang sama, yang akan memunculkan begitu banyaknya *local optima* pada proses pencarian.

2. Analisis Kinerja Algoritma

Algoritma *steepest ascent hill-climbing*, *hill-climbing with sideways move*, dan *random restart hill-climbing* menghasilkan solusi yang sama, dimana solusi pencarian memiliki *objective value* pada rentang -80-an hingga -70-an. Hal ini terjadi karena algoritma yang digunakan untuk ketiga algoritma tersebut pada dasarnya sama, dimana *hill-climbing with sideways move* hanyalah *steepest ascent hill-climbing* yang memperbolehkan berpindah ke *state* yang sama kualitasnya, sedangkan *random restart hill-climbing* hanyalah *steepest ascent hill-climbing* yang diulang berkali-kali. Akibatnya solusi yang dihasilkan pun tidak jauh berbeda, terutama saat suatu persoalan memiliki begitu banyak *local optima*.

Adapun dari segi waktu pencarian, *steepest ascent hill-climbing* menjadi yang tercepat diantara ketiganya. Algoritma *hill-climbing with sideways move* sedikit lebih lambat, tergantung pada batas maksimum *sideways move* yang diperbolehkan. Adapun algoritma

random restart hill-climbing memakan waktu yang sama dengan waktu pencarian *steepest ascent hill-climbing* dikali dengan banyaknya *restart* yang dilakukan.

Algoritma selanjutnya, *stochastic hill-climbing*, memberikan hasil pencarian yang lebih baik dari tiga algoritma sebelumnya, terutama saat jumlah iterasi yang dilakukan lebih dari 10.000, dimana *objective value* hasil pencarian berada pada rentang -70-an hingga -60-an. Hasil yang lebih baik ini terjadi karena *neighbor state* yang dipilih bukanlah *successor* dengan *value* tertinggi, tetapi dipilih secara acak. Perpindahan yang terjadi tetap hanya ke *neighbor* yang lebih baik, namun tidak harus yang paling optimal, sehingga dapat membuka ruang solusi yang berbeda dibandingkan jika hanya berpindah ke *neighbor* dengan *value* tertinggi. Adapun dari segi waktu pencarian, sangat bergantung pada banyaknya iterasi yang dilakukan, dengan 100.000 iterasi memerlukan waktu rata-rata sekitar 100 detik.

Algoritma *simulated annealing* menghasilkan kinerja terbaik dari seluruh algoritma yang diimplementasikan dalam program pencarian solusi persoalan *diagonal magic cube* $5 \times 5 \times 5$ ini, dengan *objective value* hasil pencarian berada pada rentang -50-an, yang jauh lebih baik dari 4 algoritma sebelumnya. Hal ini terjadi karena algoritma *simulated annealing* memperbolehkan berpindah ke *state* yang lebih buruk di awal-awal proses pencarian guna membuka seluas mungkin ruang pencarian yang mungkin bisa mengarah ke solusi yang lebih baik. Perpindahan ini bergantung pada suatu fungsi peluang tertentu yang nilainya akan semakin kecil ketika jumlah iterasi bertambah. Jika peluang suatu *state* lebih baik dari suatu batas bawah yang ditetapkan, maka pencarian akan berpindah ke *state* tersebut. Sebaliknya, pencarian tidak akan berpindah ke suatu *state* jika peluangnya lebih kecil dari batas yang ditetapkan. Dari segi waktu pencarian, algoritma *simulated annealing* memakan waktu 3 hingga 4 menit, yang lebih lama dari algoritma sebelumnya. Waktu pencarian ini bergantung pada seberapa cepat penurunan “*temperature*” terjadi.

Algoritma yang terakhir adalah *genetic algorithm*. Dari segi solusi yang dihasilkan, rentang nilai *objective value* hasil pencarian sangat bergantung pada parameter banyaknya iterasi/generasi dan populasi yang diberikan. Semakin banyak iterasi yang dilakukan, dan semakin banyak populasi yang diberikan, proses pencarian akan semakin baik, namun waktu komputasi yang diperlukan semakin lama. Secara rata-rata, untuk 10.000 iterasi dengan 100 populasi, nilai *objective value* hasil pencarian berada pada rentang -70-an, yang lebih baik dari algoritma *steepest ascent hill-climbing*, *hill-climbing with sideways move*, dan *random restart hill-climbing*, hampir sama dengan algoritma *stochastic hill-climbing*, dan lebih buruk dari algoritma *simulated annealing*.

Dalam program yang dibuat, dimanfaatkan suatu prinsip yang disebut dengan elitisme, yaitu prinsip mempertahankan sekumpulan individu terbaik pada generasi saat ini ke generasi selanjutnya. Tujuannya adalah agar populasi tidak mengalami penurunan kualitas dalam setiap iterasinya. Namun saat pencarian terjebak terlalu lama dalam *local optima*, proses elitisme akan dilewatkan hingga pencarian kembali mengalami peningkatan.

Dari keseluruhan algoritma, *genetic algorithm* adalah algoritma yang memakan waktu paling lama, bahkan perbedaannya sangat signifikan dibandingkan 5 algoritma sebelumnya. Sebagai contoh, untuk jumlah populasi 100 individu dengan iterasi hingga 100.000, waktu pencarian yang diperlukan hingga 11.645 detik (3 jam lebih). Waktu komputasi yang lama ini terjadi karena *genetic algorithm* memiliki algoritma yang lebih rumit dibandingkan algoritma

sebelumnya. Selain itu, algoritma ini juga mengevaluasi banyak individu sekaligus, tidak hanya satu seperti 5 algoritma sebelumnya.

Untuk setiap iterasi/generasi, harus dilakukan proses-proses *selection*, *crossover*, dan *mutation*, yang masing-masing harus melakukan iterasi ke seluruh individu yang ada. Proses-proses ini sebenarnya dapat dilakukan secara paralel/*concurrent* untuk setiap individu, namun *library concurrent python* memiliki keterbatasan untuk melakukan hal ini. Saat suatu proses memerlukan waktu yang terbilang cepat, proses pembuatan dan pengaturan *thread* akan jauh lebih mahal dibandingkan operasi itu sendiri, sehingga akan mengakibatkan *overhead*. Satu proses *selection* hanya melakukan pemilihan secara acak (berdasarkan probabilitas tertentu) dua individu pada populasi yang memiliki kompleksitas waktu $O(1)$. Adapun proses *crossover* memiliki kompleksitas waktu $O(n)$, bergantung pada ukuran individu yang disilangkan, dalam persoalan kali ini $n = 125$, yang terbilang masih rendah untuk program komputer. Akibatnya, alih-alih meningkatkan kinerja program, *overhead* dari *multiprogramming* justru malah memperburuk kinerja program. Hal ini terjadi saat kami menggunakan *library concurrent python*, seperti *concurrent.futures*, *threading*, dan *multiprocessing*, dimana kinerja program tidak lebih baik dibandingkan saat tidak menggunakan library tersebut.

BAB III

KESIMPULAN DAN SARAN

3.1 Kesimpulan

1. Algoritma *local search* tidak mampu/sulit untuk mencapai *global optima* secara sempurna dalam masalah yang memiliki banyak puncak lokal, seperti persoalan *diagonal magic cube*, yang membuat algoritma mudah untuk terjebak ke dalam solusi suboptimal.
2. Dalam program penyelesaian persoalan *diagonal magic cube* $5 \times 5 \times 5$ yang dibuat, dari segi hasil pencarian, algoritma *simulated annealing* memberikan hasil pencarian terbaik, sedangkan algoritma *steepest ascent hill-climbing*, *hill-climbing with sideways move*, dan *random restart hill-climbing* memberikan hasil pencarian terburuk.
3. Dalam program penyelesaian persoalan *diagonal magic cube* $5 \times 5 \times 5$ yang dibuat, dari segi waktu pencarian, algoritma *stochastic hill-climbing* memiliki waktu pencarian yang paling cepat, sedangkan *genetic algorithm* memiliki waktu pencarian yang paling lama.

3.2 Saran

1. Gunakan bahasa pemrograman lain yang memungkinkan *multiprogramming* dilakukan dengan lebih baik untuk mempercepat proses pencarian dalam algoritma *genetic algorithm*.
2. Lakukan efisiensi pada kode program untuk mengefisienkan kinerja program.
3. Gunakan teknik pencarian lainnya untuk dapat menyelesaikan persoalan *diagonal magic cube* $5 \times 5 \times 5$.

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522006	Agil Fadillah Sabri	<i>Genetic Algorithm</i> , Kelas MagicCube.
13522027	Muhammad Althariq Fairuz	<i>Steepest Ascent Hill-climbing</i> , <i>Hill-climbing with Sideways Move</i> , Visualisasi.
13522034	Bastian H. Suryapratama	<i>Random Restart Hill-climbing</i> , <i>Stochastic Hill-climbing</i> .
13522052	Haikal Assyauqi	<i>Simulated Annealing</i> .

REFERENSI

- Breedijk, A. (n.d.). *Half Pandiagonal 5x5x5 Magic Cube (Shift Method)*. Magisch Vierkant. <https://www.magischvierkant.com/three-dimensional-eng/5x5x5/half-pandiagonal-shift/>.
- Guariso, G., & Sangiorgio, M. (2020). *Improving the Performance of Multiobjective Genetic Algorithms: An Elitism-Based Approach*. MPDI Open Access Journals, 11(12). <https://www.mdpi.com/2078-2489/11/12/587#:~:text=Most%20algorithms%20use%20%E2%80%9Celitism%E2%80%9D,%20which.>
- Patil, V. P., & Pawar, D. D. (2015). *THE OPTIMAL CROSSOVER OR MUTATION RATES IN GENETIC ALGORITHM: A REVIEW*. CIBTech, 5(3), 38-41. <https://www.cibtech.org/J-ENGINEERING-TECHNOLOGY/PUBLICATIONS/2015/VOL-5-NO-3/05-JET-006-PATIL-MUTATION.pdf#:~:text=ABSTRACT%20Choice%20of%20crossover%20and/or.>
- Rani, S., Suri, B., & Goyal, R. (2019). *On the Effectiveness of Using Elitist Genetic Algorithm in Mutation Testing*. MPDI Open Access Journals, 11(9). <https://www.mdpi.com/2073-8994/11/9/1145#:~:text=This%20study%20implements%20an%20elitist%20Genetic.>
- Taylor, Y. (2019, Maret 17). *Genetic Algorithms*. SlideServe. <https://www.slideserve.com/yonah/genetic-algorithms-powerpoint-ppt-presentation#:~:text=Content%20%E2%80%A2%20Evolutional%20Algorithms%20%E2%80%A2.>
- Wolfram. (n.d.). *Perfect Magic Cube*. Wolfram. <https://mathworld.wolfram.com/PerfectMagicCube.html>.