

Tugas Besar 1 IF3170 Inteligensi Artifisial

Pencarian Solusi Diagonal Magic Cube dengan Local Search



Disusun Oleh:

Shafiq Irvansyah	13522003
Raden Rafly Hanggaraksa Budiarto	13522014
Abdul Rafi Radityo Hutomo	13522089
Muhammad Dava Fathurrahman	13522114

Daftar Isi

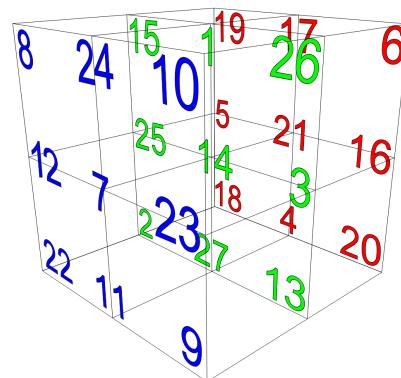
1. Deskripsi Persoalan	3
2. Pembahasan	4
2.1. Pemilihan Objective Function	4
2.2. Implementasi Kelas/Fungsi	7
2.2.1. Steepest Ascent Hill Climb	7
2.2.2. Hill Climb with Sideways Move	9
2.2.3. Random Restart Hill Climb	9
2.2.4. Stochastic Hill Climbing	10
2.2.5. Simulated Annealing	10
2.2.6. Genetic Algorithm	11
2.3. Hasil eksperimen	12
2.3.1. Steepest Ascent Hill Climb	12
2.3.2. Hill-Climbing with Sideways Move	13
2.3.3. Random Restart Hill Climbing	15
2.3.4. Stochastic Hill Climbing	18
2.3.5. Simulated Annealing	19
2.3.6. Genetic Algorithm	21
2.3.6.1. Variabel Kontrol Jumlah Populasi	21
2.3.6.2. Variabel Kontrol Banyak Iterasi	25
2.4. Analisis	29
3. Kesimpulan dan Saran	29
Pembagian tugas tiap anggota kelompok	29
Referensi	30

1. Deskripsi Persoalan

25	16	80	104	90				
115	98	4	1	97	90			
42	111	85	2	75	97	70		
66	72	27	102	48	75	13	70	
67	18	119	106	5	5	25	13	56
67	18	119	106	5	5	114	94	
116	17	14	73	95	95	37	10	
40	50	81	65	79	79	96	100	
56	120	55	49	35	35	74	11	59
36	110	46	22	101	101	60	84	

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. Jumlah angka-angka untuk setiap baris sama dengan magic number
- Jumlah angka-angka untuk setiap kolom sama dengan magic number
- Jumlah angka-angka untuk setiap tiang sama dengan magic number
- Jumlah angka-angka untuk seluruh diagonal ruang pada kubus sama dengan magic number
- Jumlah angka-angka untuk seluruh diagonal pada suatu potongan bidang dari kubus sama dengan magic number
 - Berikut ilustrasi dari potongan bidang yang ada pada suatu kubus berukuran 3:



- Terdapat 9 potongan bidang, yaitu:

8 24 10	15 1 26	19 17 6
12 7 23	25 14 3	5 21 16
22 11 9	2 27 13	18 4 20
19 17 6	5 21 16	18 4 20
15 1 26	25 14 3	2 27 13
8 24 10	12 7 23	22 11 9
8 15 19	12 25 5	22 2 18
24 1 17	7 14 21	11 27 4
10 26 6	23 3 16	9 13 20

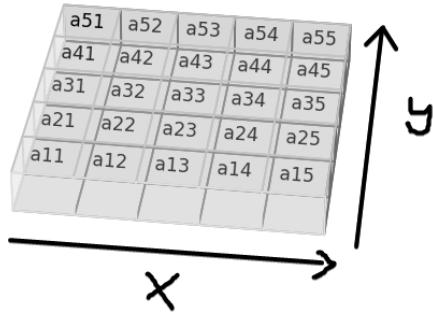
Pada tugas ini, dihadapkan dengan masalah pencarian solusi untuk **Diagonal Magic Cube**. Tantangan utamanya adalah menemukan susunan elemen pada kubus tersebut sehingga memenuhi aturan diatas. Untuk mencapai tujuan tersebut, digunakan berbagai algoritma **local search** yang bertujuan untuk menemukan solusi optimal berdasarkan fungsi objektif yang telah didefinisikan.

2. Pembahasan

2.1. Pemilihan Objective Function

Objective Function adalah fungsi yang digunakan sebagai parameter untuk menilai seberapa dekat suatu state pada state space dengan state tujuan. Berdasarkan persoalan yang telah dirincikan di atas, dapat ditarik sebuah kesimpulan.

Misal setiap angka pada kubus dimisalkan menjadi L_{yx} dengan $L \in \{a, b, c, d, e\}$ dimana a berkorelasi dengan level pertama, b berkorelasi dengan level kedua, dan seterusnya, x adalah koordinat x dari angka tersebut dan y adalah koordinat y dari angka tersebut.



Visualisasi dari sistem penomoran pada level 1 dari kubus

Berdasarkan ketentuan magic cube, yaitu jumlah semua baris pada setiap level harus sama, maka didapat persamaan

$$\sum_{x=1}^5 a_{1x} = \sum_{x=1}^5 a_{2x} = \sum_{x=1}^5 a_{3x} = \dots$$

Atau dapat ditulis dalam bentuk lain, yaitu

$$\forall L \in \{a, b, c, d, e\} (\forall y \in \{1, 2, 3, 4, 5\} (\sum_{x=1}^5 L_{yx} = m))$$

Dengan m adalah magic number.

Perhatikan ekspresi berikut yang merupakan jumlah dari seluruh angka pada kubus

$$\sum_{L=a}^e \sum_{y=1}^5 \sum_{x=1}^5 L_{yx}$$

dapat disederhanakan dengan proses berikut,

$$\sum_{L=a}^e \sum_{y=1}^5 (\sum_{x=1}^5 L_{yx})$$

ekspresi pada tanda kurung selalu bernilai m, maka

$$\sum_{L=a}^e \sum_{y=1}^5 (\sum_{x=1}^5 L_{yx}) = \sum_{L=a}^e \sum_{y=1}^5 m = 25m$$

Nilai dari ekspresi tersebut adalah jumlah dari seluruh angka pada kubus, yaitu jumlah semua bilangan asli hingga 125, sehingga nilai dari m didapat

$$25m = 1 + 2 + 3 + 4 + \dots + 124 + 125$$

$$25m = \frac{(1 + 125)(125)}{2}$$

$$25m = 7875$$

$$m = 315$$

Oleh karena itu, dapat disimpulkan bahwa untuk sebuah kubus $5 \times 5 \times 5$ dapat dikategorisasikan sebagai sebuah diagonal magic cube, jumlah semua angka-angka untuk setiap baris, setiap kolom, setiap tiang, setiap diagonal ruang, dan setiap diagonal pada perpotongan bidang harus sama dengan 315.

Namun, selain menghitung jumlah total nilai yang sama dengan 315, pada fungsi objektif juga akan ditambahkan nilai lain untuk membandingkan kubus dengan jumlah total 315 yang sama. Sebagai nilai pembanding ini, digunakan nilai standar deviasi dari besaran jumlah-jumlah pada kubus.

Dari kesimpulan tersebut, dirumuskan fungsi objektif yang ingin dimaksimalkan, yaitu

$$h(c) = n(c) + \text{std}(sums)$$

$$n(c) = \text{frekuensi dari } 315 \text{ pada multiset } S$$

$$\text{std}(sums) = \text{standar deviasi dari jumlah-jumlah pada kubus}$$

dengan

S = jumlah angka-angka pada setiap baris \cup jumlah angka-angka pada setiap kolom \cup jumlah semua angka-angka pada setiap tiang \cup jumlah angka-angka pada setiap

diagonal ruang U jumlah angka-angka pada setiap diagonal pada setiap perpotongan bidang.

2.2. Implementasi Kelas/Fungsi

2.2.1. Steepest Ascent Hill Climb

Untuk algoritma ini dibuat sebuah **Kelas Cube** dengan spesifikasi sebagai berikut.

Atribut	
Nama dan Tipe Atribut	Penjelasan
cube : List[5][5][5]	List yang merepresentasikan angka pada kubus
Prosedur/Fungsi Instance	
Procedure randomize()	Prosedur untuk mengganti state kubus menjadi sebuah konfigurasi acak
Function getH()	Fungsi yang mengembalikan nilai H berdasarkan state kubus saat ini
Function copyCube()	Fungsi yang mengembalikan copy dari kubus saat ini
Prosedur/Fungsi Kelas	
Function getAllCoordinatePairs()	Mengembalikan semua kombinasi dua koordinat yang ada pada kubus dengan urutan acak

Untuk algoritma utama Steepest ascent hill climb sesuai dengan yang telah dijelaskan dilakukan dengan pseudocode ini

```

procedure steepestAscentHillClimb
    done = False
    while not done do
        currentH = self.getH()
        pair, newH = self.findSteepestAscent()

        if (newH > currentH) then
            coord1, coord2 = pair
            x1, y1, z1 = coord1
            x2, y2, z2 = coord2
            self.cube[z1][y1][x1], self.cube[z2][y2][x2] = self.cube[z2][y2][x2], self.cube[z1][y1][x1]
        else
            done = True

```

Dengan fungsi `findSteepestAscent` adalah fungsi untuk mendapatkan pertukaran koordinat yang menghasilkan suksesor dengan value tertinggi dengan pseudocode sebagai berikut

```

function findSteepestAscent(self) -> (((int, int, int), (int, int, int)), double)
    dummy_cube = Cube.copyCube(self)

    coordinatePairs = Cube.allCoordinatePairs(self.dimension)

    maxCoordPairs = None
    maxH = -1
    traversal coord1, coord2 in coordinatePairs
        x1, y1, z1 = coord1
        x2, y2, z2 = coord2
        dummy_cube.cube[z1][y1][x1], dummy_cube.cube[z2][y2][x2] = dummy_cube.cube[z2][y2][x2], dummy_cube.cube[z1][y1][x1]

        currentH = dummy_cube.getH()

        if currentH > maxH then
            maxH = currentH
            maxCoordPairs = (coord1, coord2)

    dummy_cube.cube[z1][y1][x1], dummy_cube.cube[z2][y2][x2] = dummy_cube.cube[z2][y2][x2], dummy_cube.cube[z1][y1][x1]
    -> (maxCoordPairs, maxH)

```

2.2.2. Hill Climb with Sideways Move

Algoritma ini menggunakan Kelas Cube yang sama dengan sebelumnya dan dengan pseudocode sebagai berikut.

```

procedure sidewayAscentHillClimb
    done = False
    limit = 100
    currentStreak = 0
    while not done do
        currentH = self.getH()
        pair, newH = self.findsteepestAscent()

        if (newH > currentH or (newH == currentH and currentStreak < limit)) then
            coord1, coord2 = pair
            x1, y1, z1 = coord1
            x2, y2, z2 = coord2
            self(cube[z1][y1][x1], self(cube[z2][y2][x2] = self(cube[z2][y2][x2], self(cube[z1][y1][x1]

            if (newH > currentH) then
                currentStreak = 0
            else
                currentStreak += 1
            else
                done = True

```

2.2.3. Random Restart Hill Climb

Algoritma ini menggunakan kelas Cube yang sama dengan sebelumnya dan dengan pseudocode sebagai berikut.

```

procedure randomRestartHillclimb()
    while self.getH() != 109 :
        self.randomize()
        self.steepestAscentHillclimb()

```

2.2.4. Stochastic Hill Climbing

```

procedure stochasticHillClimb()
    nmax = 200000

    i traversal nmax

    x1, y1, z1, x2, y2, z2 = 0, 0, 0, 0, 0, 0
    while(x1 == x2 and y1 == y2 and z1 == z2) do
        x1, y1, z1 = randomInteger[0-4], randomInteger[0-4], randomInteger[0-4]
        x2, y2, z2 = randomInteger[0-4], randomInteger[0-4], randomInteger[0-4]

    currentH = self.getH()
    self(cube[z1][y1][x1], self(cube[z2][y2][x2] = self(cube[z2][y2][x2], self(cube[z1][y1][x1]

    newH = self.getH()

    if (currentH >= newH) then
        self(cube[z1][y1][x1], self(cube[z2][y2][x2] = self(cube[z2][y2][x2], self(cube[z1][y1][x1]

```

2.2.5. Simulated Annealing

```
function simulated_annealing(var problem, func schedule, func objective_function) -> a solution state
    current -> problem.initial
    for t=1 to inf do
        T <- schedule(t)

        if T = 0 then return current

        next <- get_random_successor(current)
        delta_E <- objective_function(next) - objective_function(current)
        if delta_E > 0 then current <- next
        else if get_probab(delta_E,T) > THRESHOLD then current <- next|
```

Fungsi Schedule

Fungsi ini memetakan waktu menjadi suhu. Semakin berjalananya waktu, suhu yang dihasilkan akan semakin kecil. Hal ini sesuai dengan konsep **Annealing**.

Fungsi Objective

Fungsi ini akan memberikan “jarak” yang dihasilkan oleh suatu state untuk menuju state solusi.

Fungsi Get Random Successor

Fungsi ini akan menggenerasi satu suksesor acak (tanpa memperhatikan nilai heuristiknya) yang akan dijadikan pertimbangan untuk dijadikan state selanjutnya.

Fungsi Get Probability

Fungsi ini akan menghasilkan nilai probabilitas . Fungsi ini bertujuan untuk memberikan nilai probabilitas yang akan dijadikan pertimbangan untuk penentuan state selanjutnya.

2.2.6. Genetic Algorithm

Algoritma genetik sesuai dengan penjelasan langkah-langkah sebelumnya memiliki pseudocode sebagai berikut,

```
function geneticAlgorithm() -> Cube
    population = initPopulation()

    for i in range(10000) :
        fitnesses = getFitnesses()
        buckets = getBucketsFromFitnesses(fitnesses)

        parents = []
        j traversal [0..popSize - 1]
            randomIdx = generateRandom(buckets)
            parents.append(population[randomIdx])

        j traversal [0..popSize//2 - 1]
            population[2*j], population[2*j + 1] = combine(parents[2*j], parents[2*j + 1])
            mutate(population[2*j])
            mutate(population[2*j + 1])

    -> maxHCube(population)
```

Fungsi **getFitnesses**

Fungsi ini akan menghasilkan list nilai fitness dari populasi

Fungsi **getBucket**

Fungsi ini akan memberikan seluruh list bobot state sesuai dengan list nilai fitness.

Fungsi **generateRandom**

Fungsi ini akan menghasilkan sejumlah n parent berdasarkan metode Random Choice melalui list bobot yang diberikan pada argumen.

Fungsi **combine**

Fungsi ini akan menghasilkan state baru berdasarkan kawin silang dari 2 parent state.

Fungsi **mutate**

Fungsi ini akan melakukan mutasi pada state dengan mengubah suatu nilai konfigurasi secara acak.

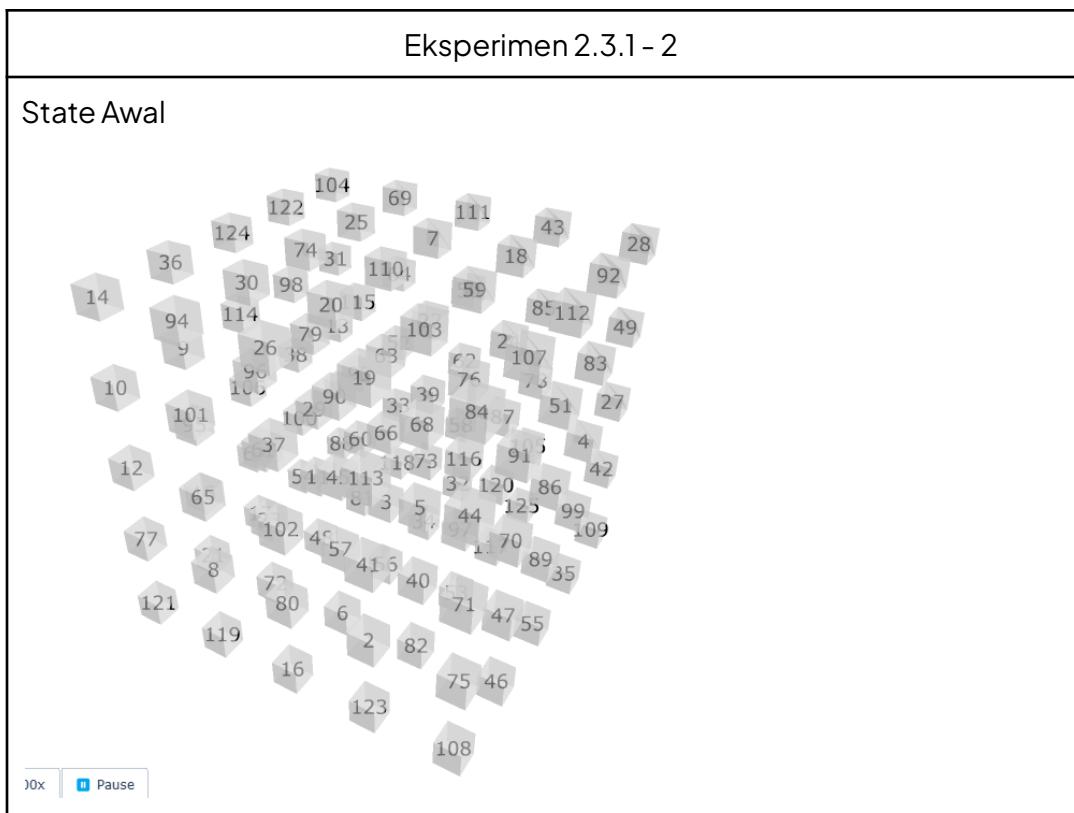
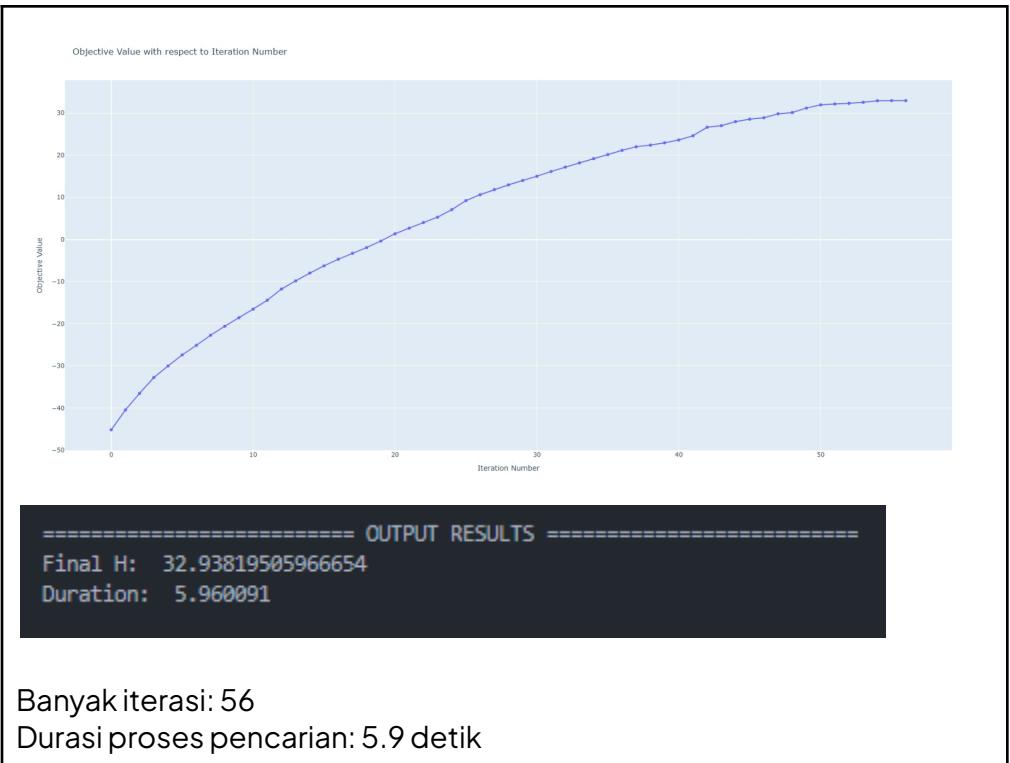
Fungsi **maxCube**

Fungsi ini akan mengembalikan state yang memiliki nilai fitness function terbesar dari populasinya.

2.3. Hasil eksperimen

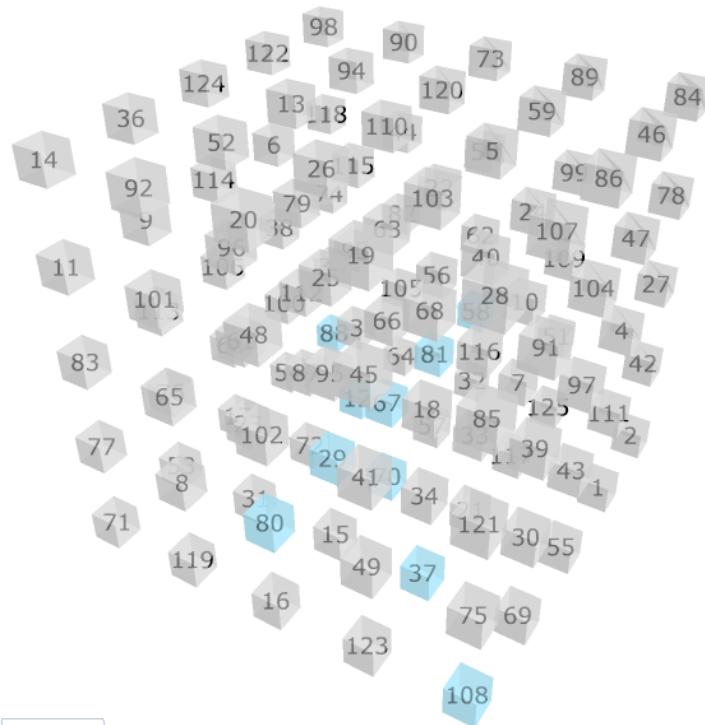
2.3.1. Steepest Ascent Hill Climb





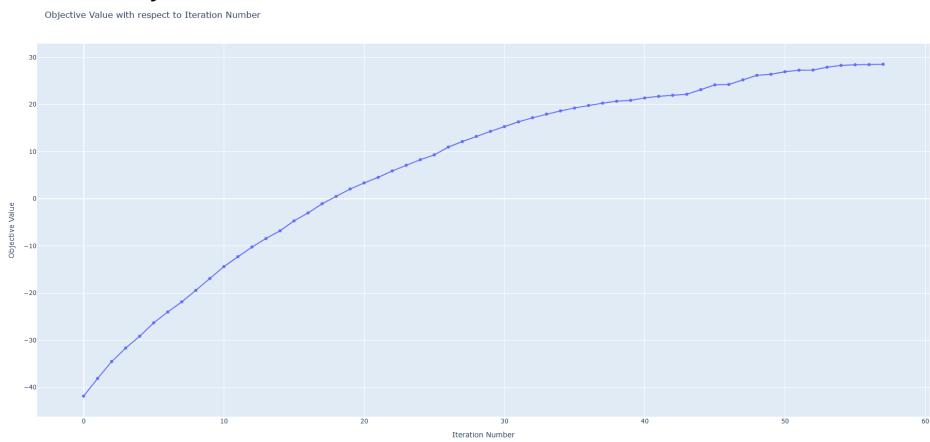
Nilai objektif = -41.8

State Akhir



Nilai objektif= 28.55

Plot nilai objective function



===== OUTPUT RESULTS =====

Final H: 28.553636617958617

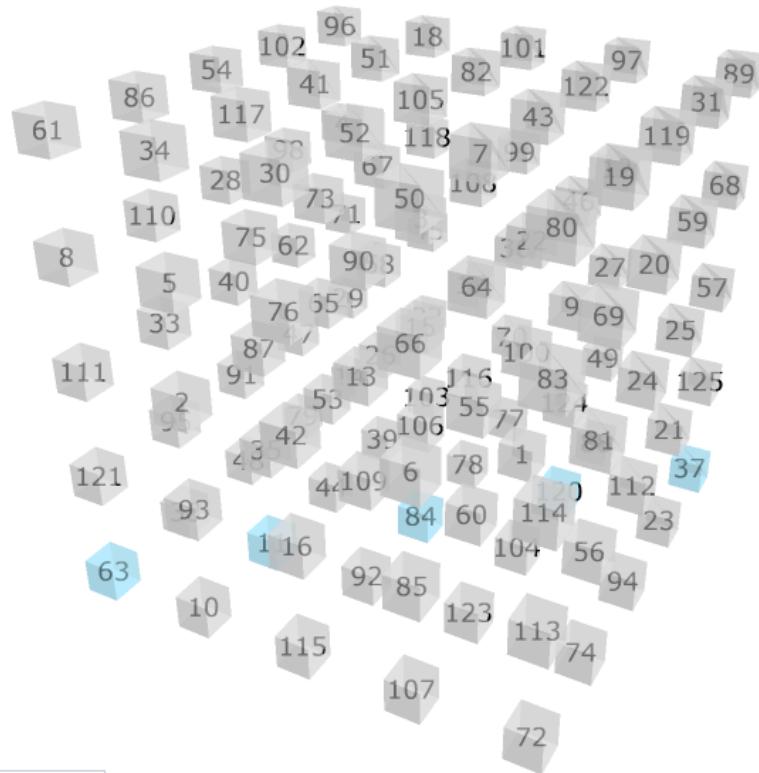
Duration: 5.928606

Banyak iterasi: 57

Durasi proses pencarian: 5.92 detik

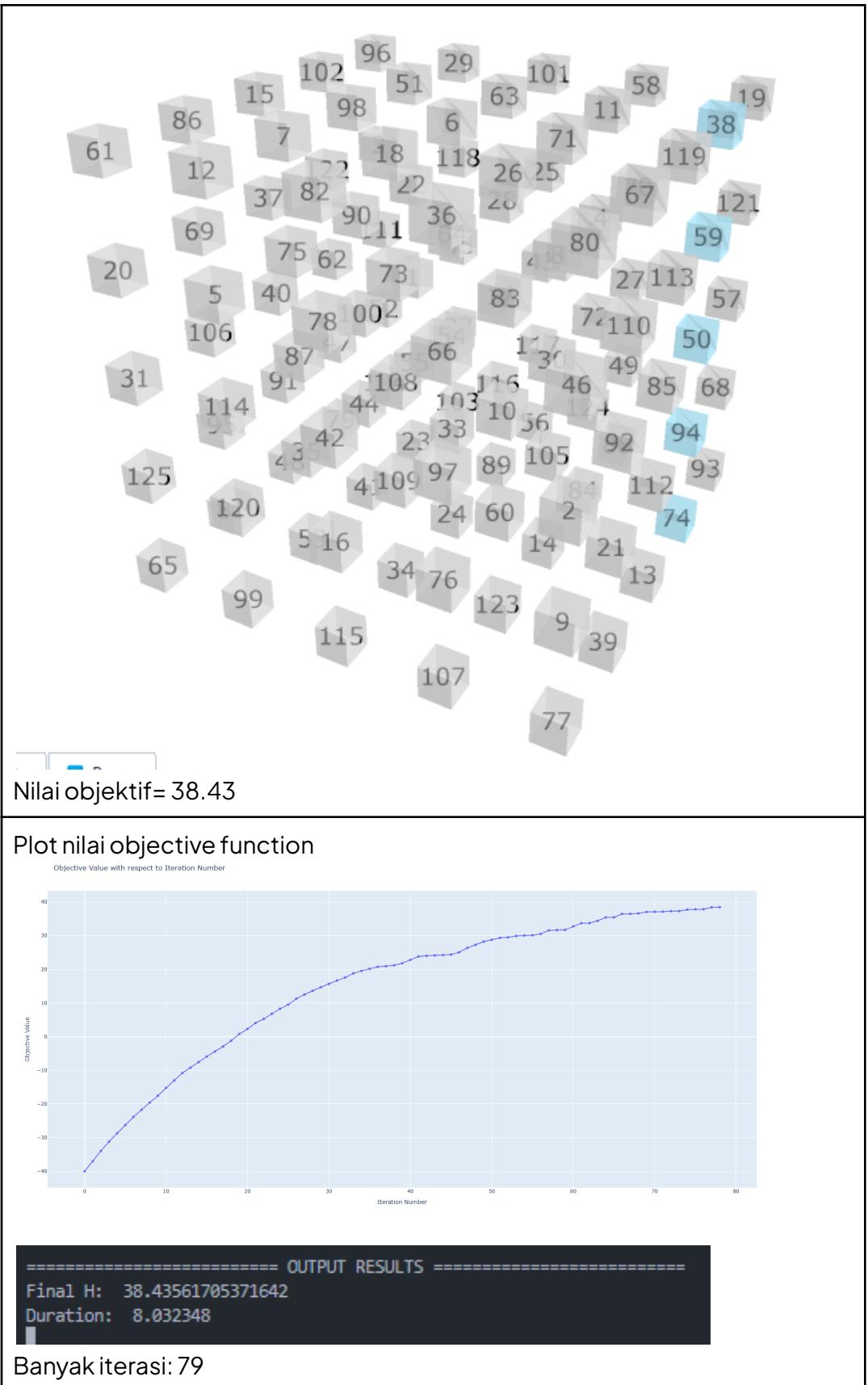
Eksperimen 2.3.1 - 3

State Awal



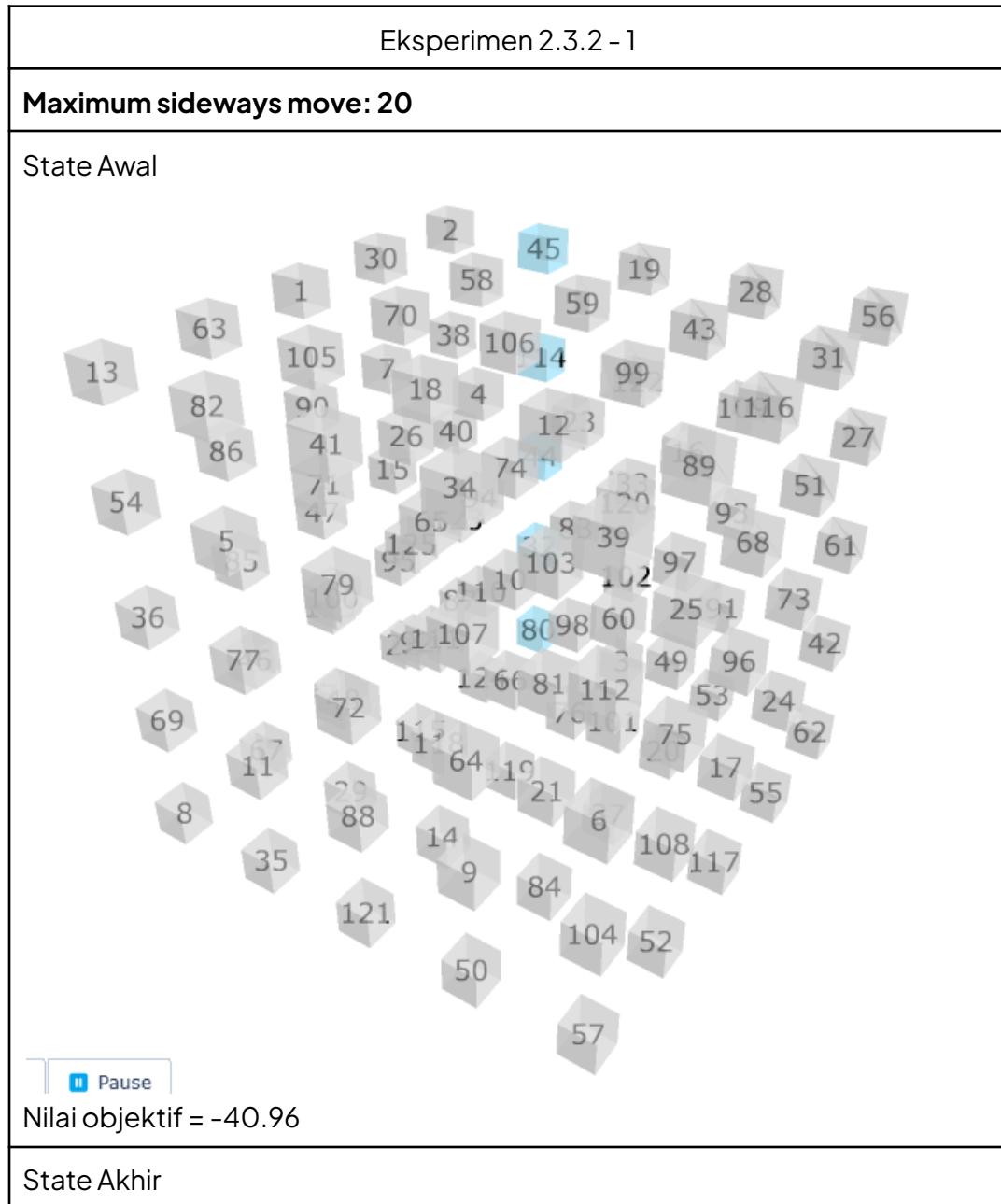
Nilai objektif = -39.99

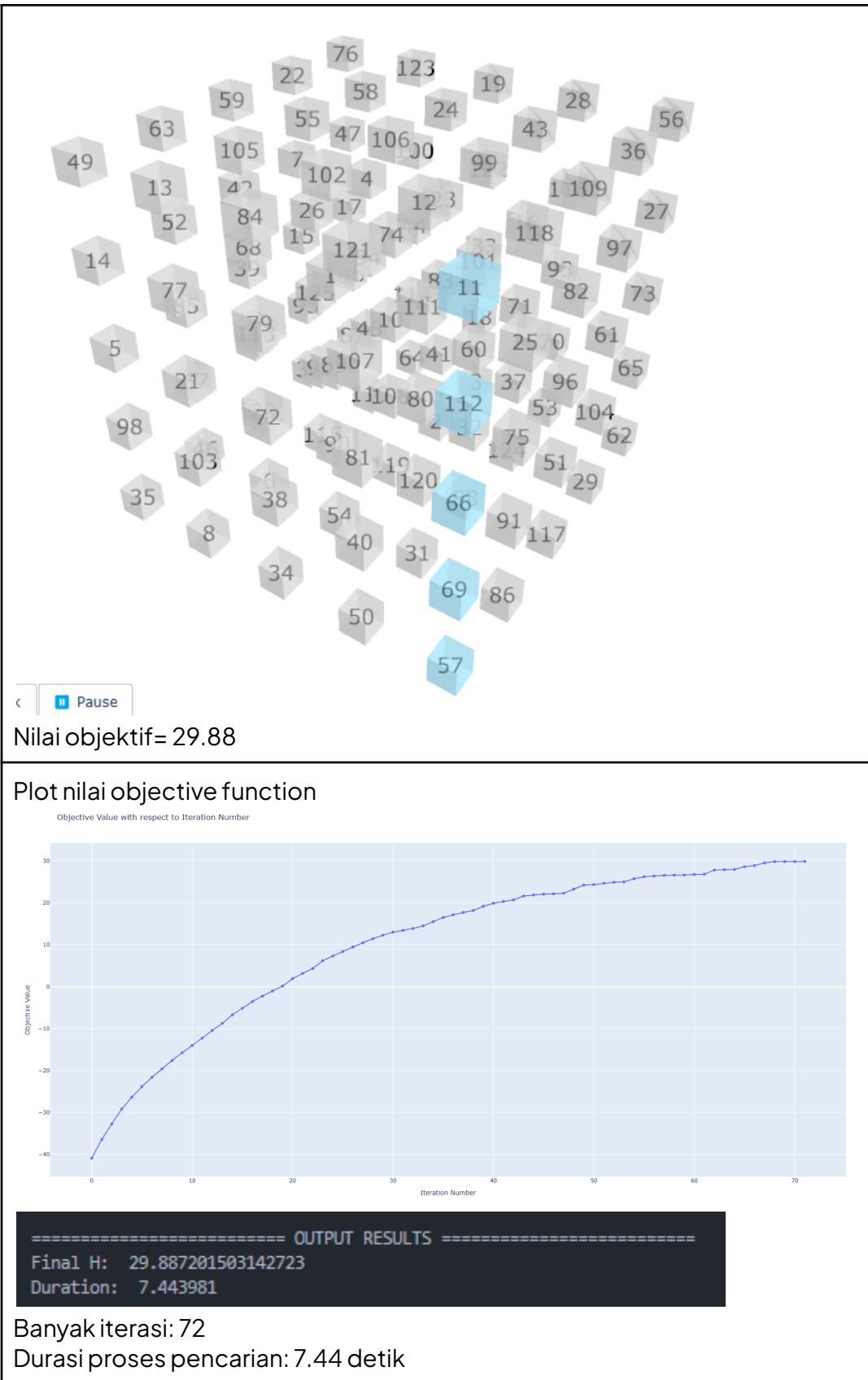
State Akhir



Durasi proses pencarian: 8 detik

2.3.2. Hill-Climbing with Sideways Move

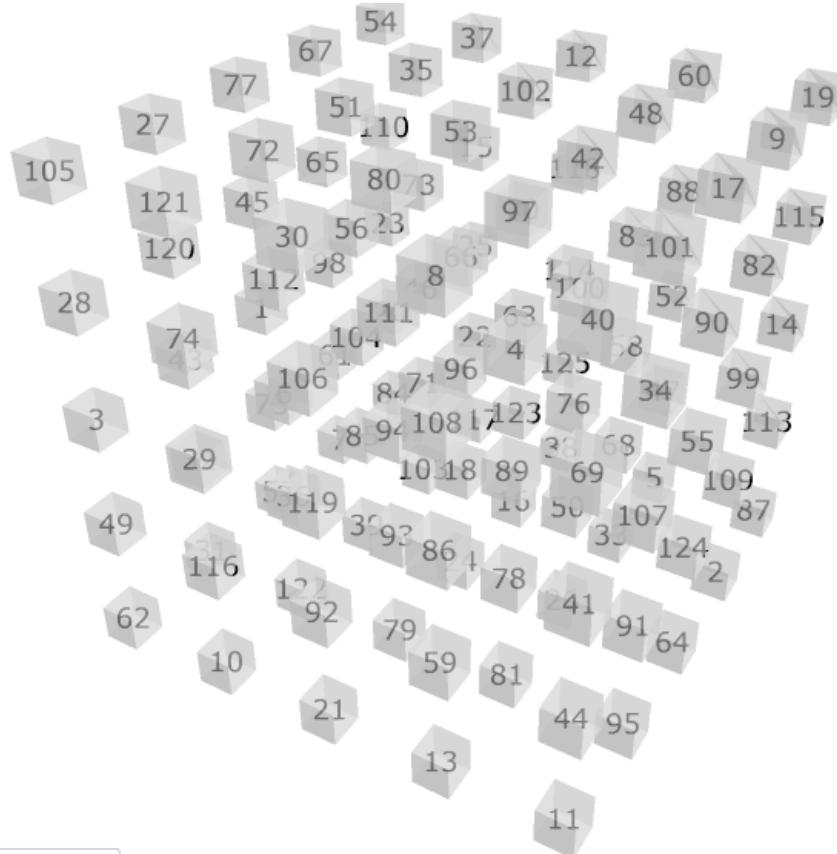




Eksperimen 2.3.2 - 2

Maximum sideways move: 30

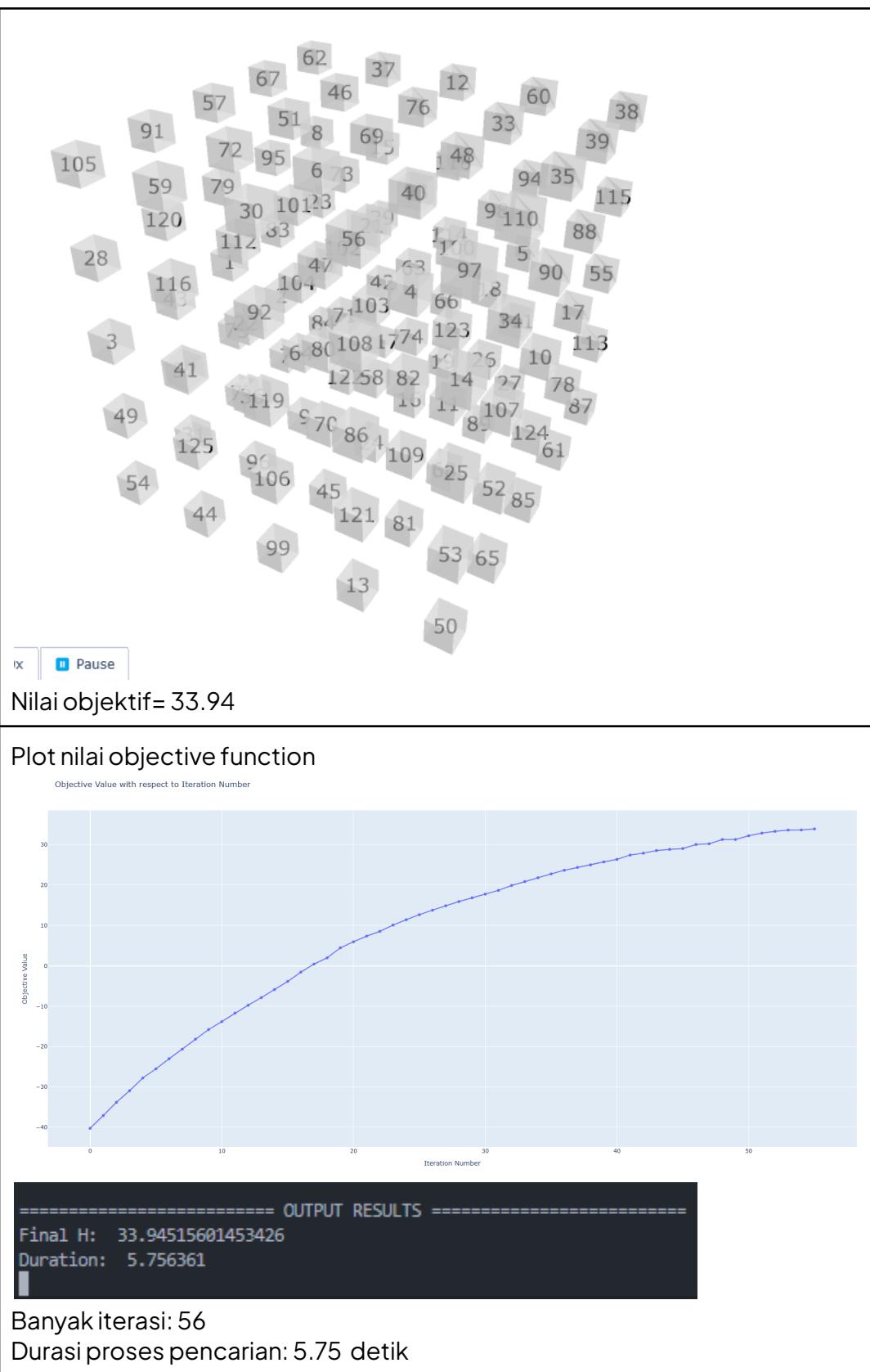
State Awal



Dx Pause

Nilai objektif = -40.26

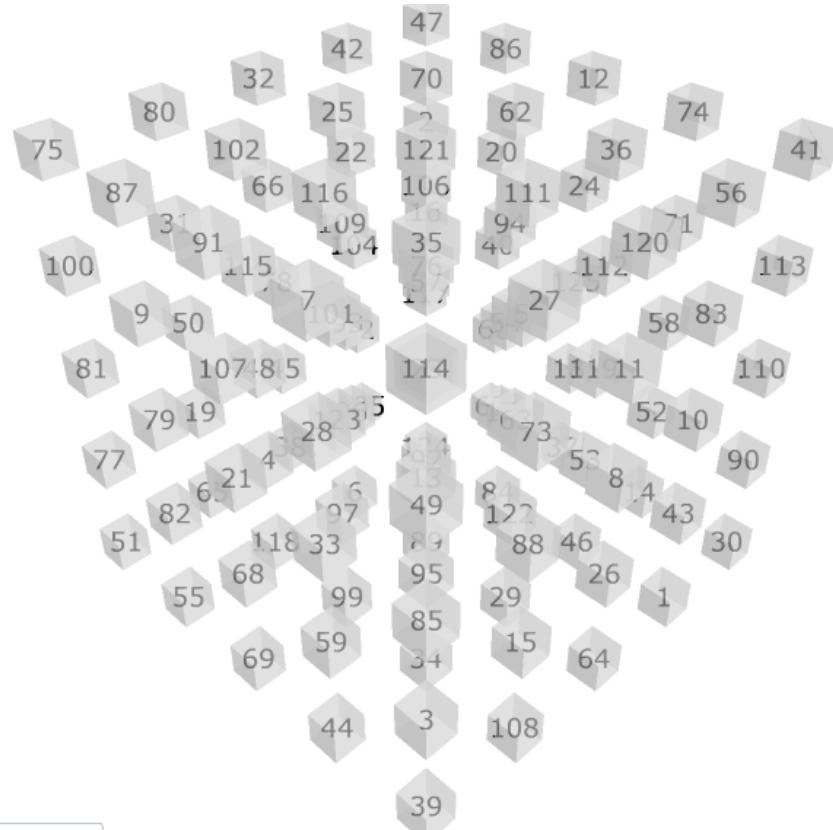
State Akhir



Eksperimen 2.3.2 - 3

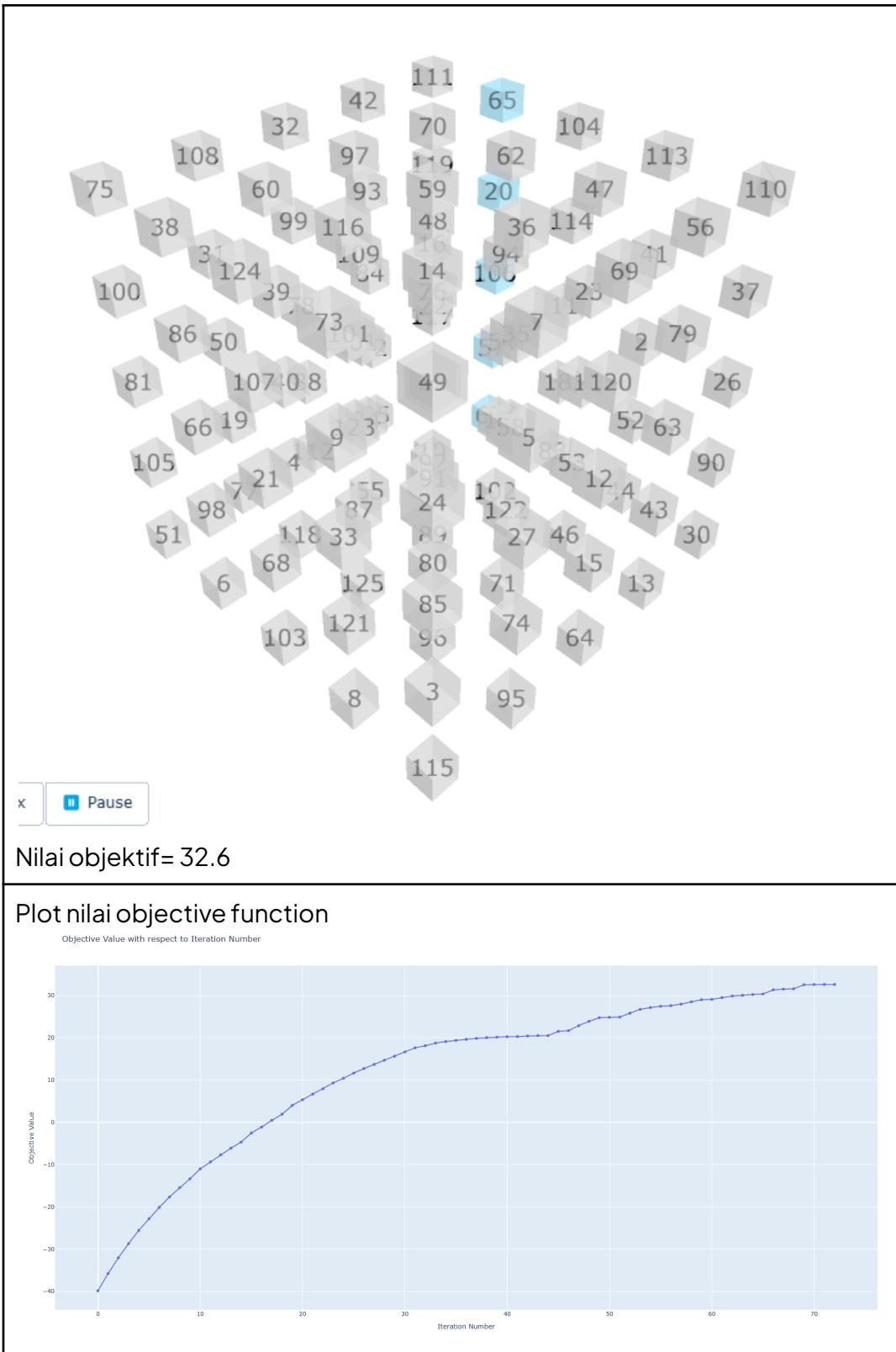
Maximum sideways move: 40

State Awal



Nilai objektif = -39.86

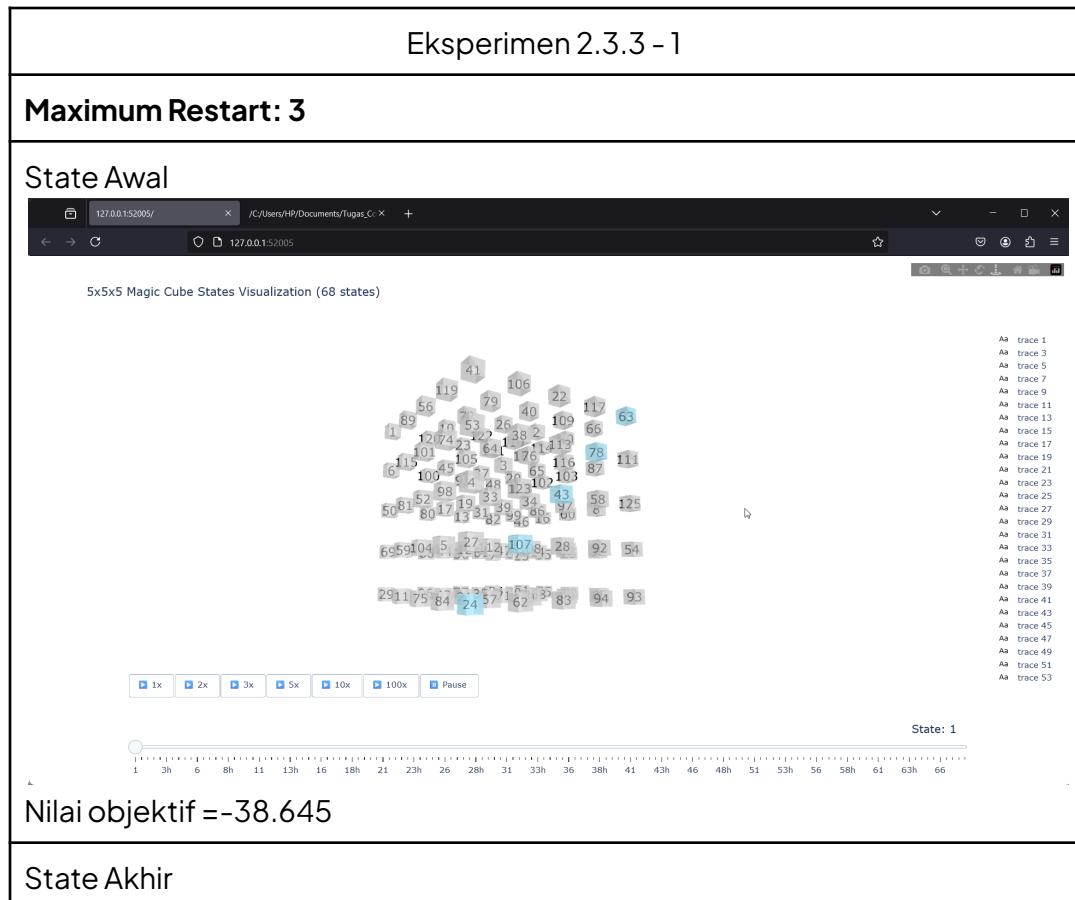
State Akhir



```
===== OUTPUT RESULTS =====
Final H: 32.60290795563482
Duration: 7.408585
```

Banyak iterasi: 73
Durasi proses pencarian: 7.4 detik

2.3.3. Random Restart Hill Climbing

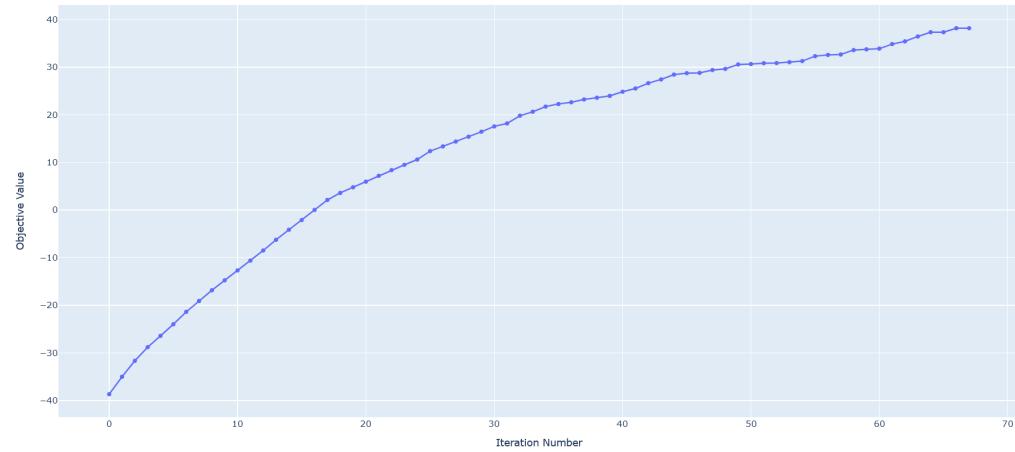




Nilai objektif=38.158

Plot nilai objective function

Objective Value with respect to Iteration Number



Banyak restart: 3

Banyak iterasi per restart: 77, 55, 68

Durasi proses pencarian: 13.84

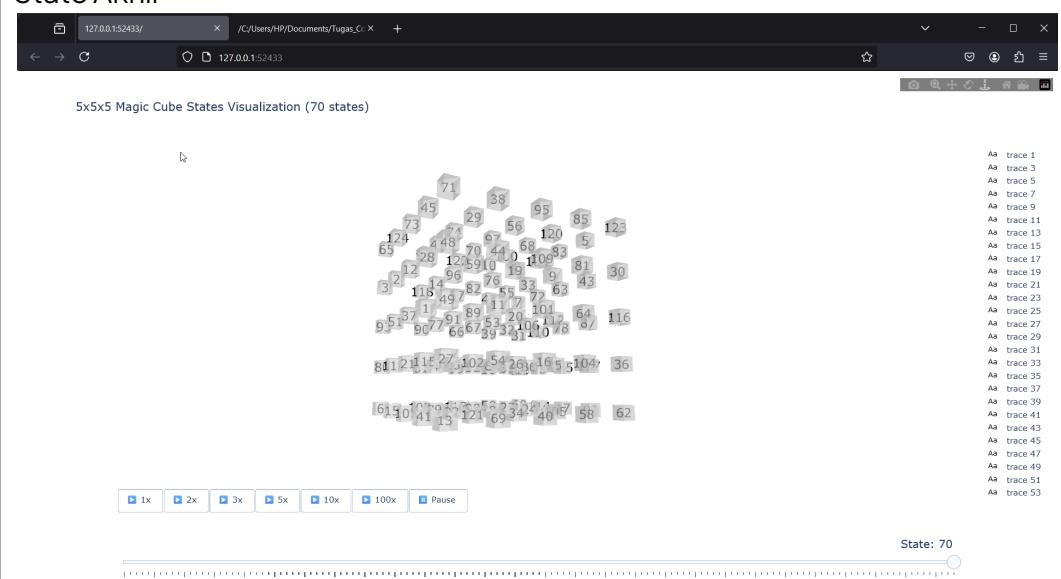
Eksperimen 2.3.3 - 2

Maximum Restart:

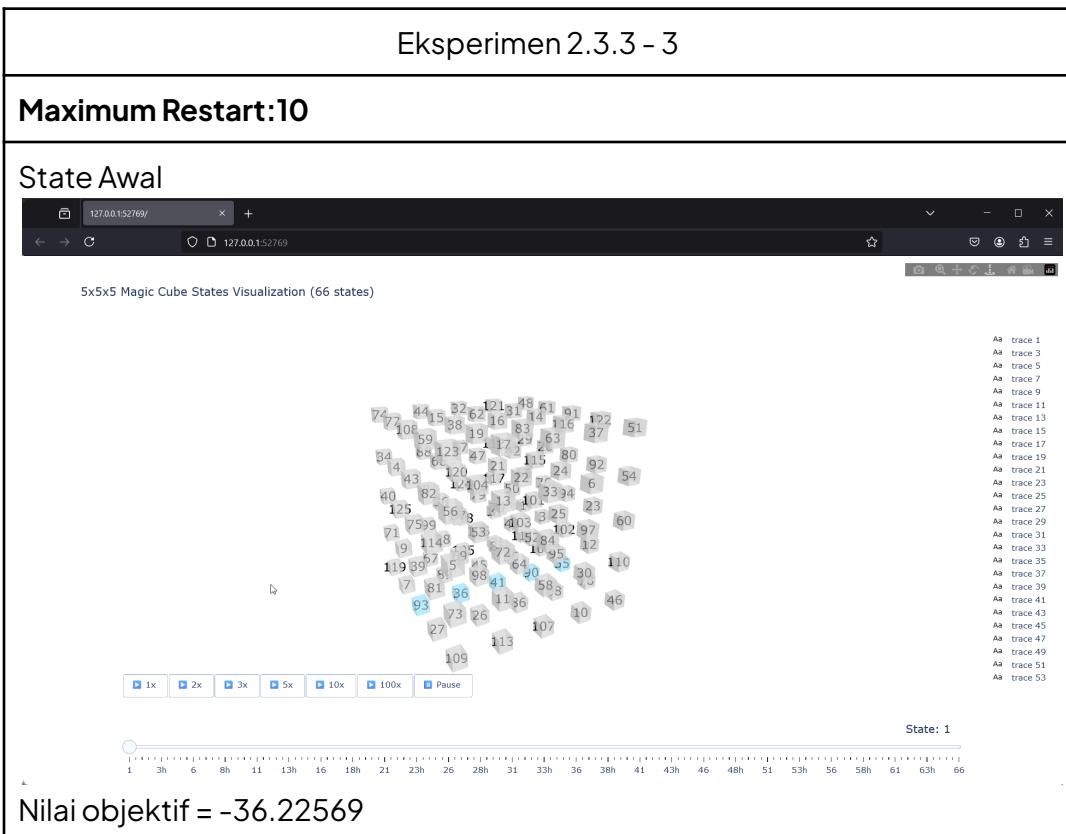
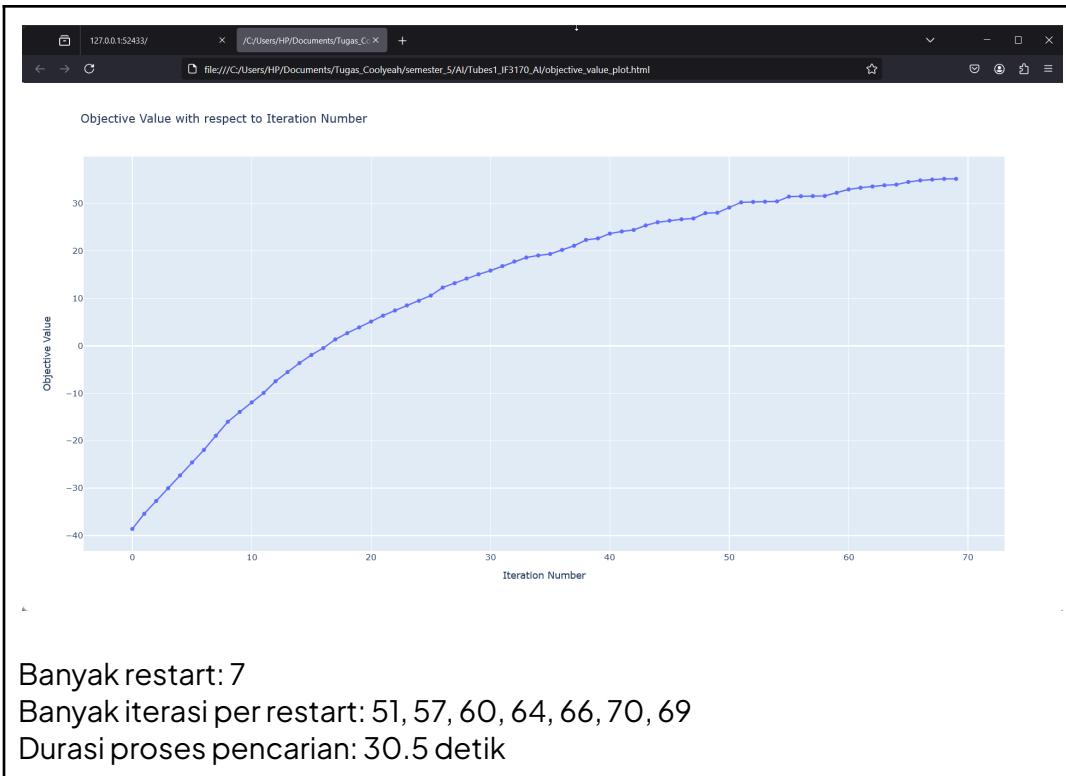
State Awal



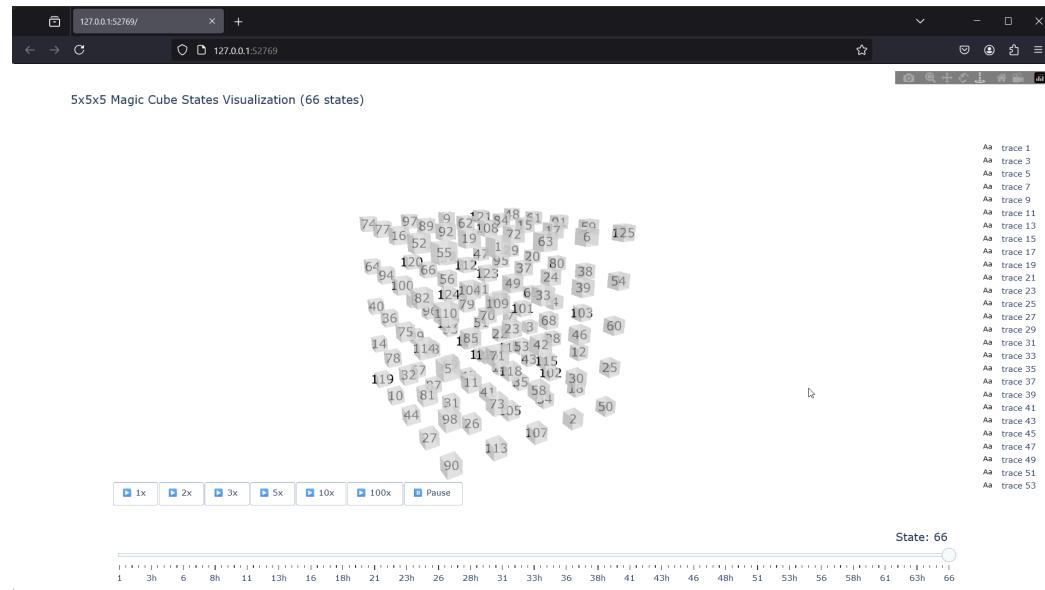
State Akhir



Plot nilai objective function

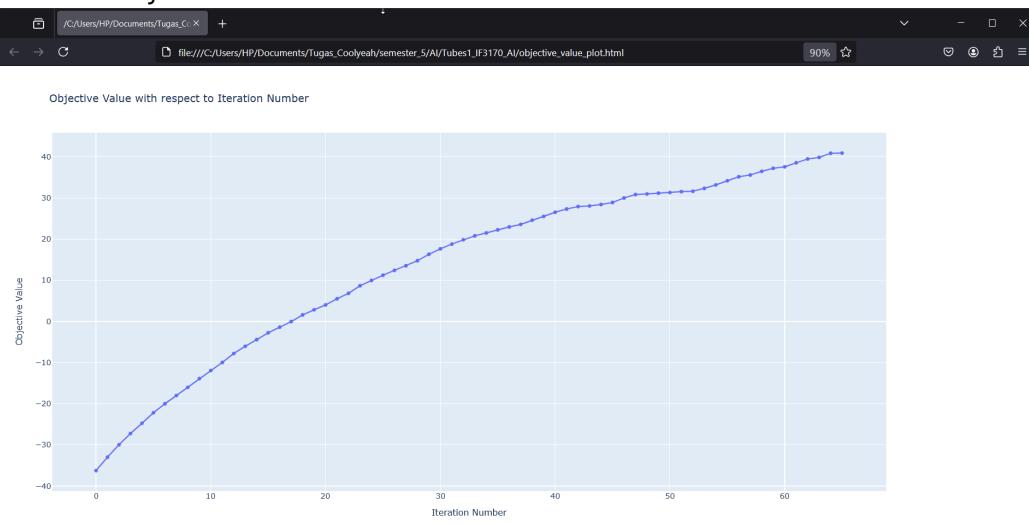


State Akhir



Nilai objektif= 32.440839346590174

Plot nilai objective function



Banyak restart: 10

Banyak iterasi per restart: 86, 63, 65, 56, 69, 73, 66, 69, 65, 77

Durasi proses pencarian: 41.8 detik

Eksperimen 2.3.3 - 4

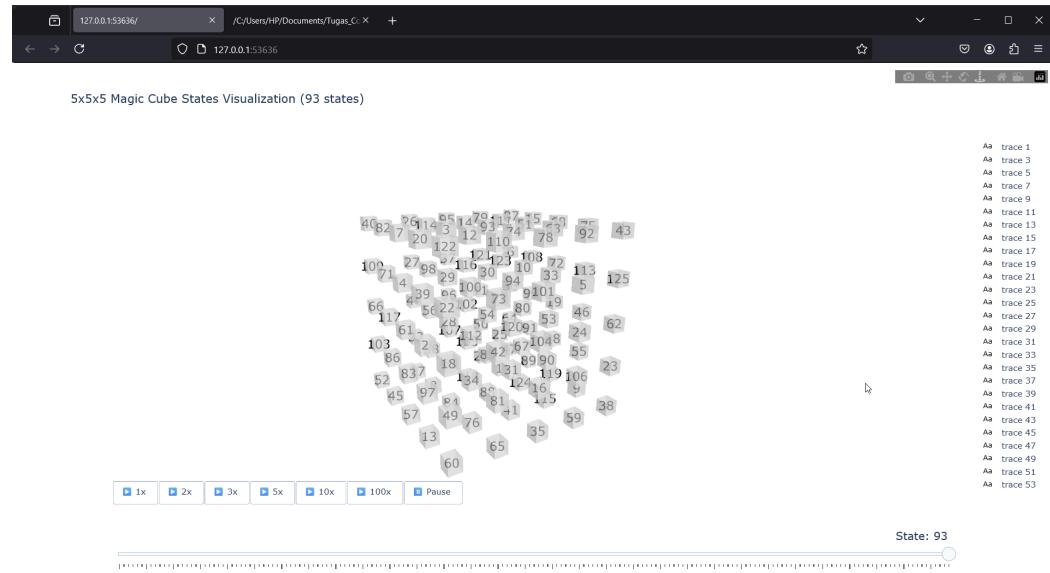
Maximum restart: 15

State Awal



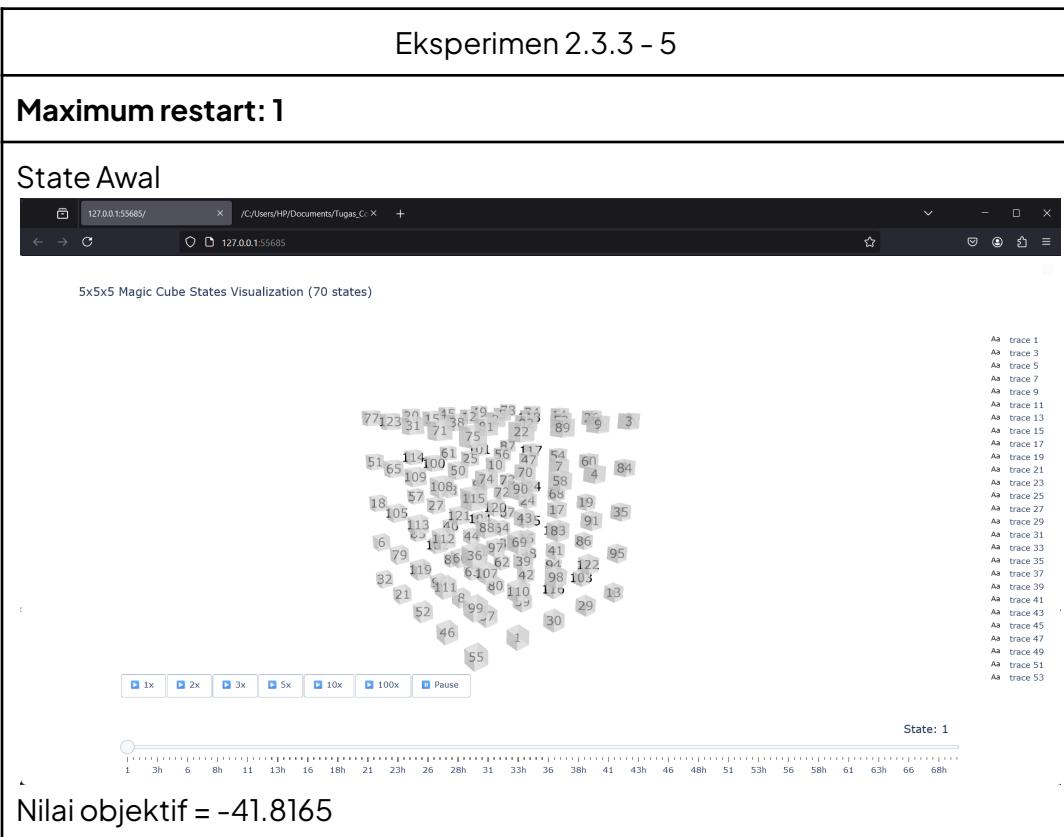
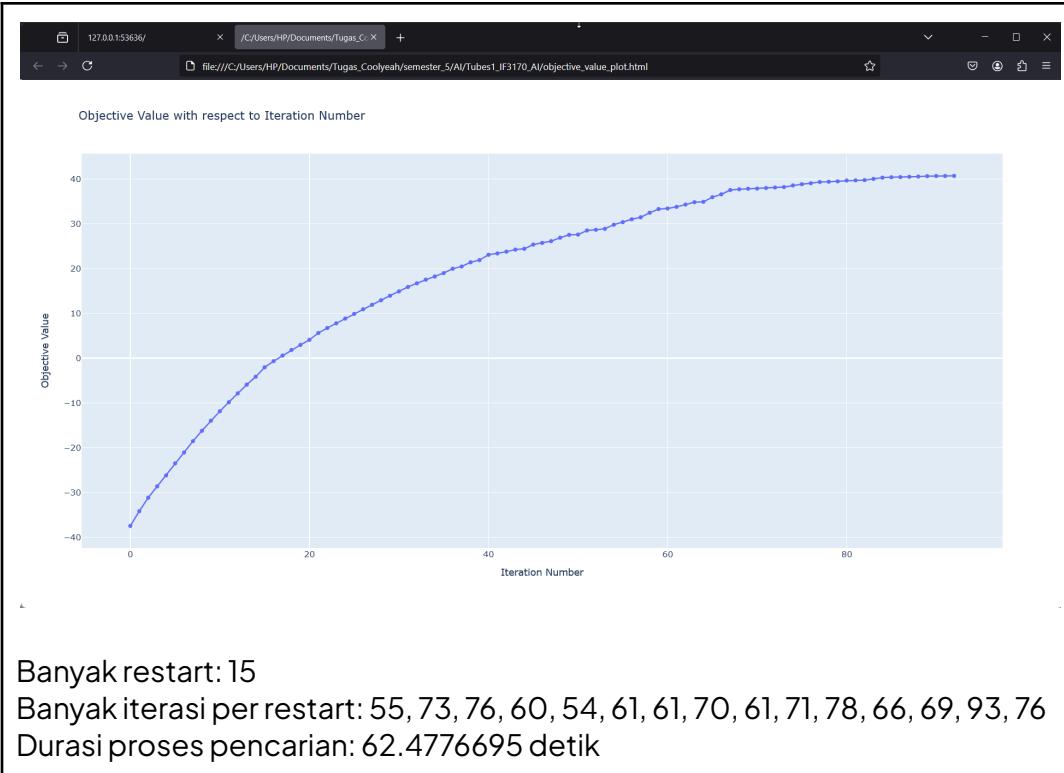
Nilai objektif = -37.42221

State Akhir

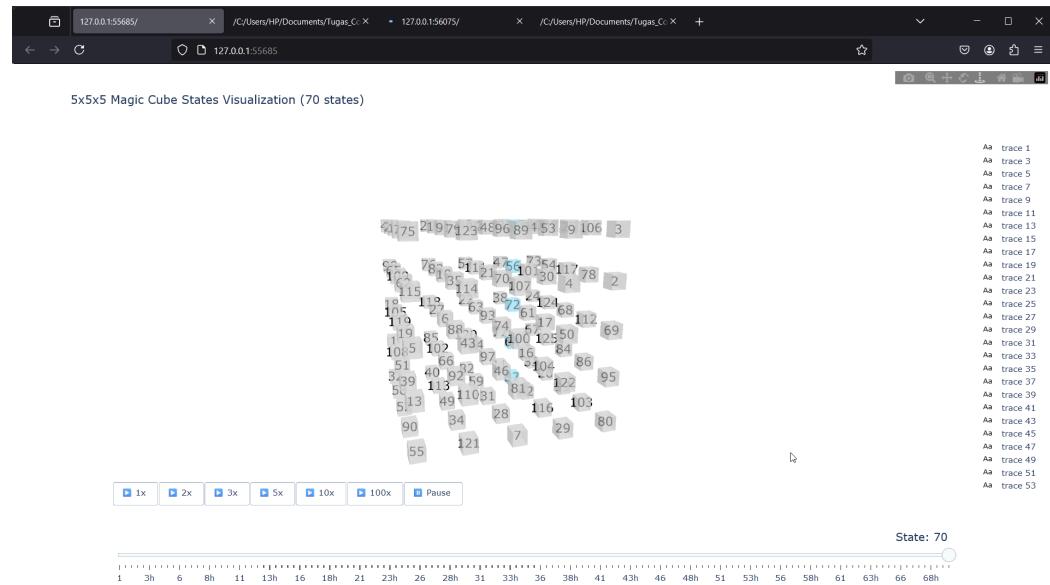


Nilai objektif=40.71202

Plot nilai objective function

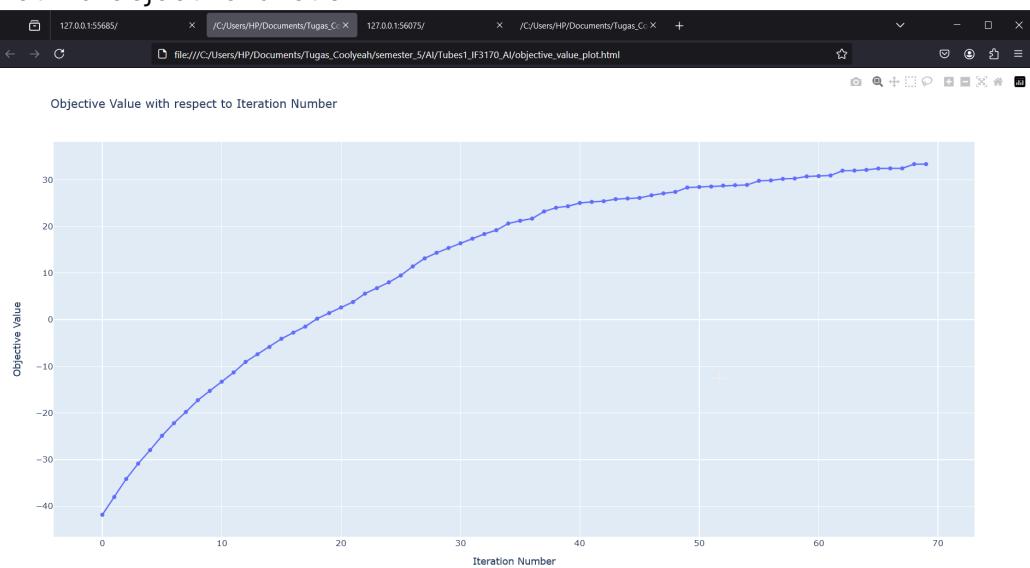


State Akhir



Nilai objektif= 33.3434

Plot nilai objective function



Banyak restart: 1

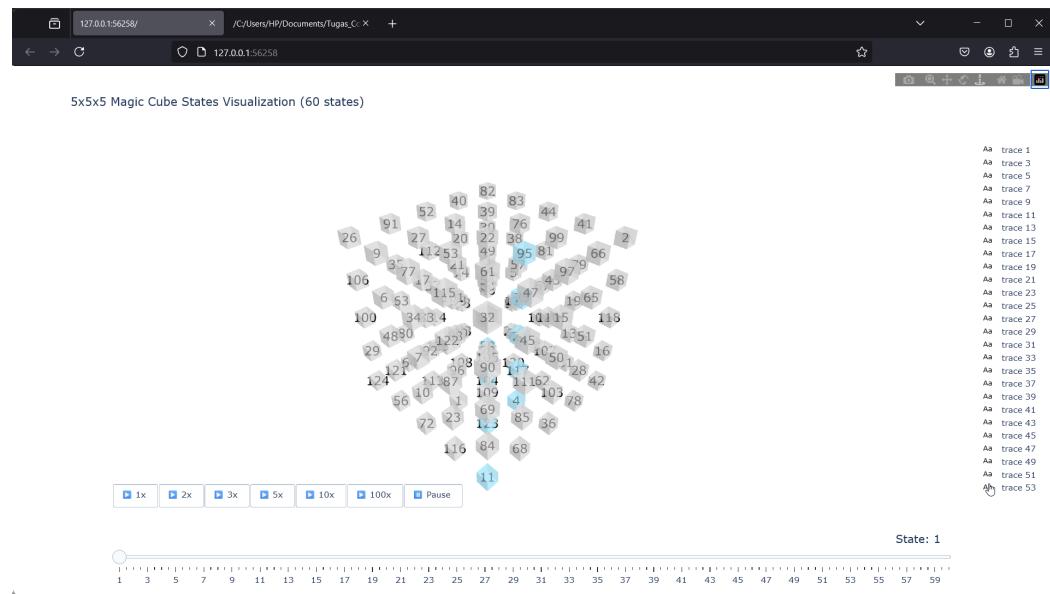
Banyak iterasi per restart: 67

Durasi proses pencarian: 4.231 detik

Eksperimen 2.3.3 - 6

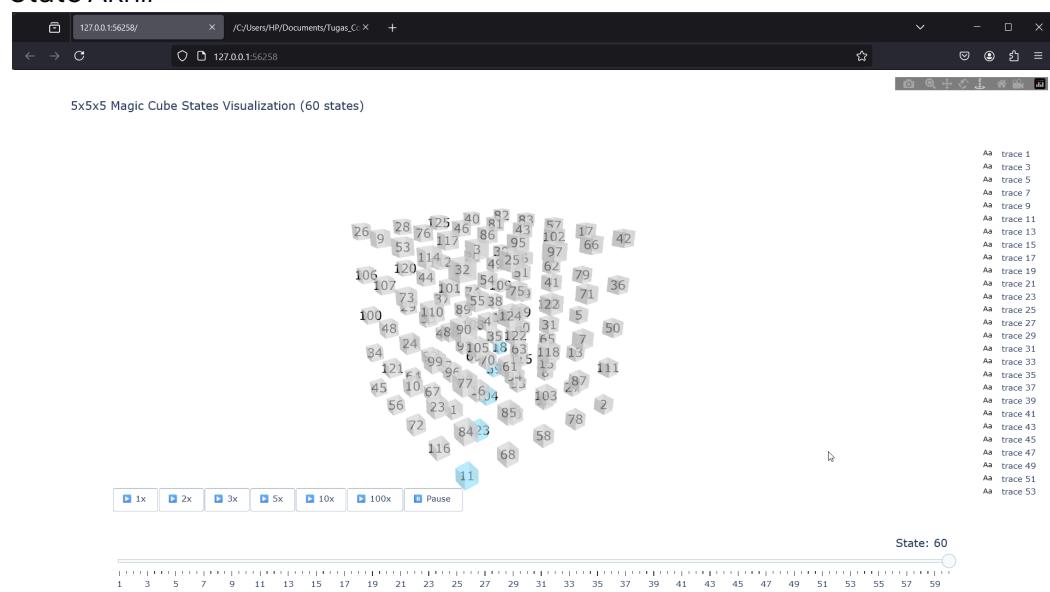
Maximum restart: 3

State Awal



Nilai objektif = -38.7204

State Akhir



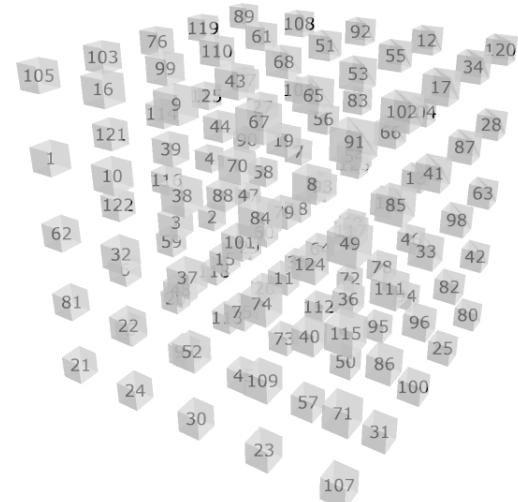
Nilai objektif= 34.867

Plot nilai objective function



2.3.4. Stochastic Hill Climbing

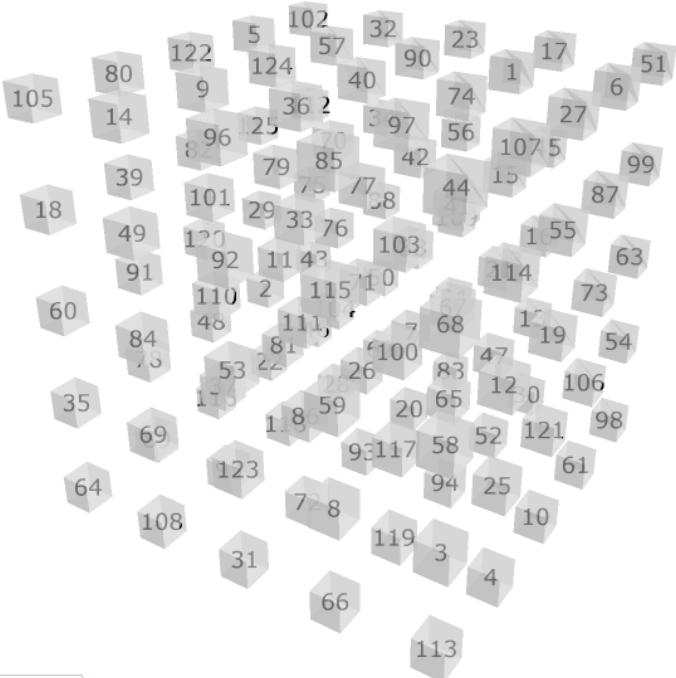
Eksperimen 2.3.4 - 1
State Awal



100x Pause

Nilai objektif = -39,16

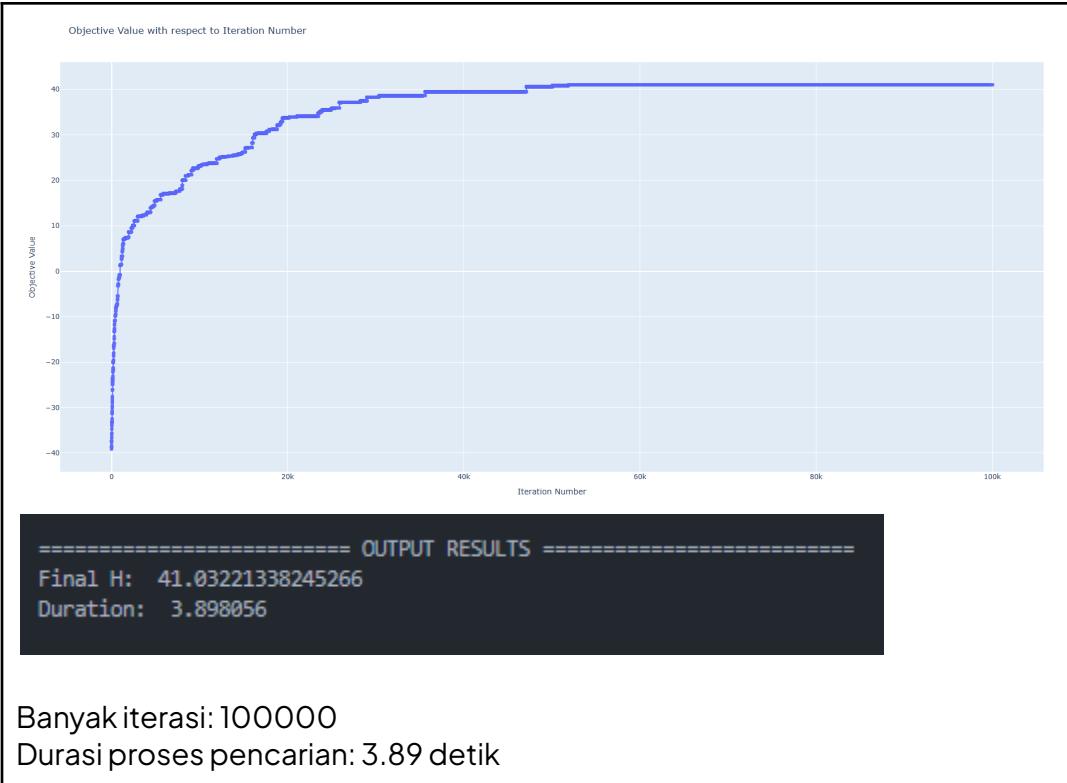
State Akhir



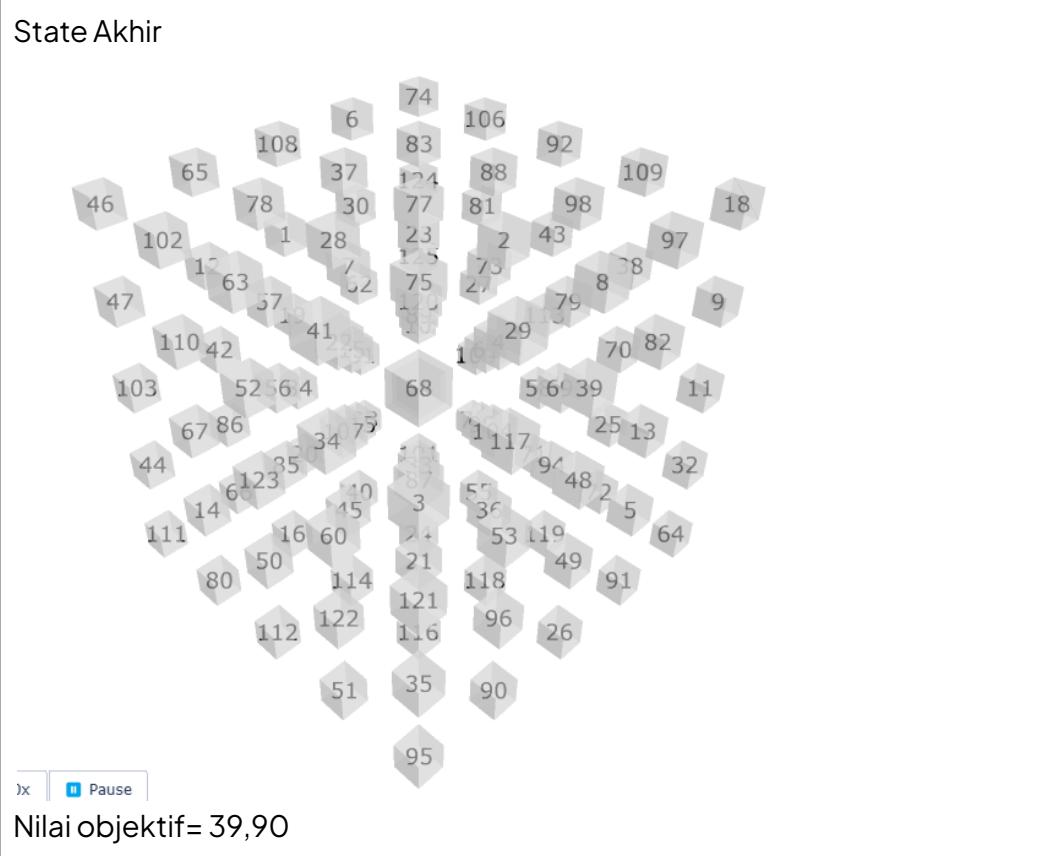
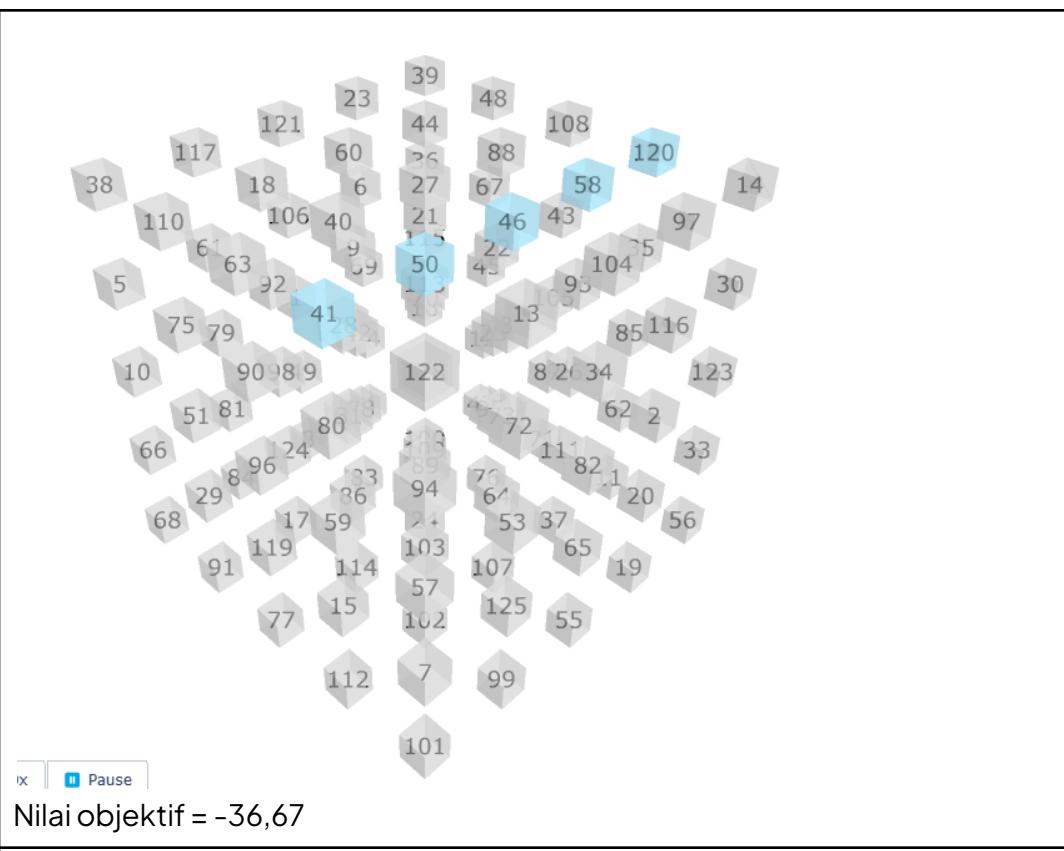
100x Pause

Nilai objektif= 41.03

Plot nilai objective function

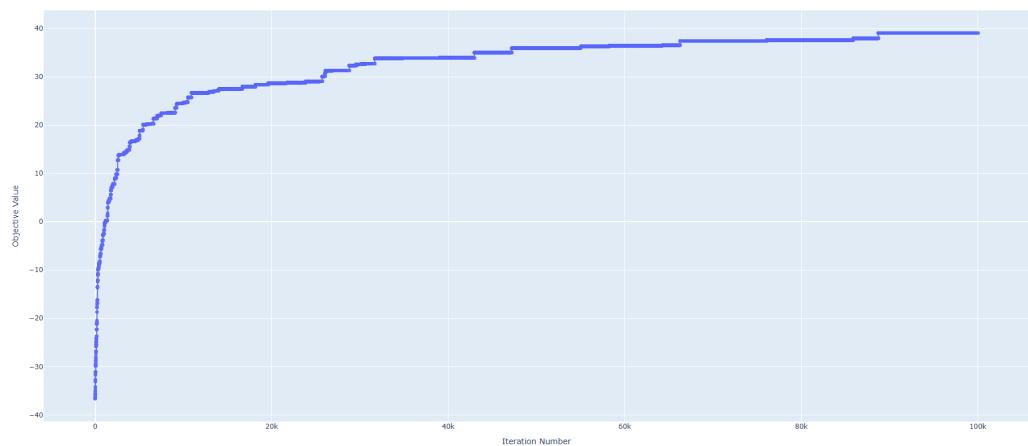


Eksperimen 2.3.4 - 2
State Awal



Plot nilai objective function

Objective Value with respect to Iteration Number



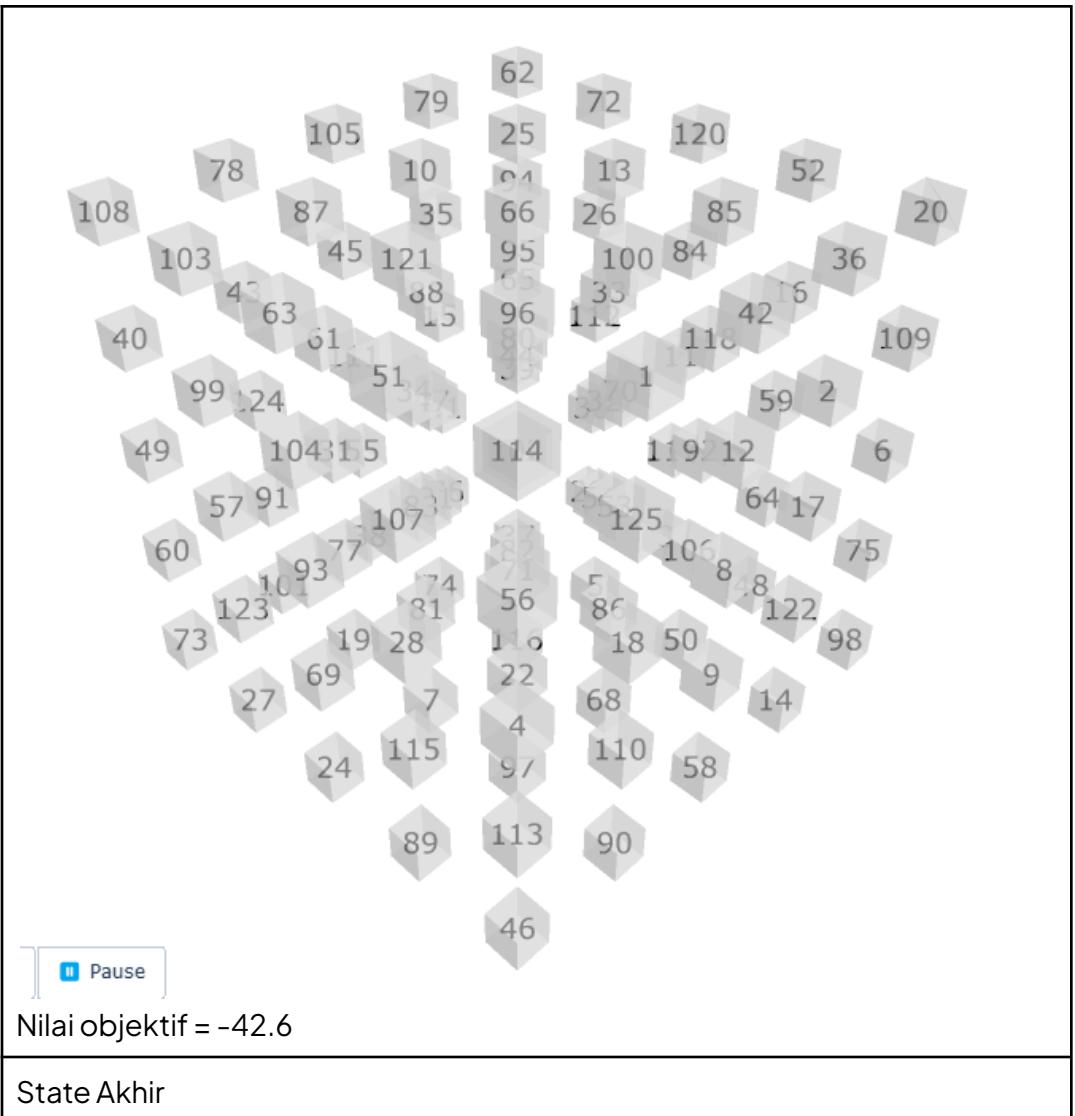
```
===== OUTPUT RESULTS =====
Final H: 39.073056454886604
Duration: 3.948323
[]
```

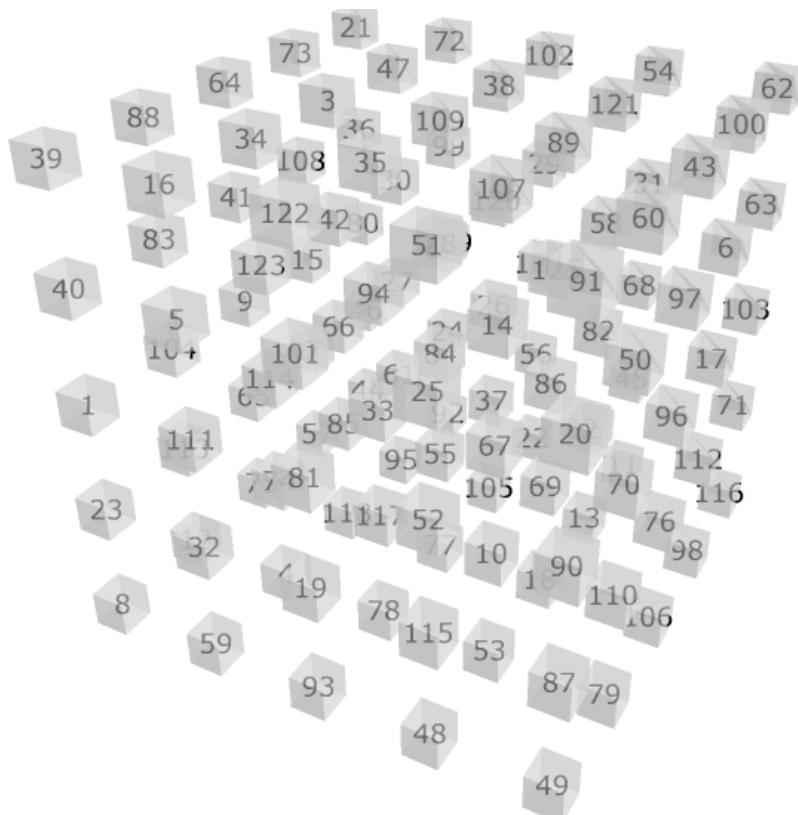
Banyak iterasi: 100000

Durasi proses pencarian: 3.94 detik

Eksperimen 2.3.4 - 3

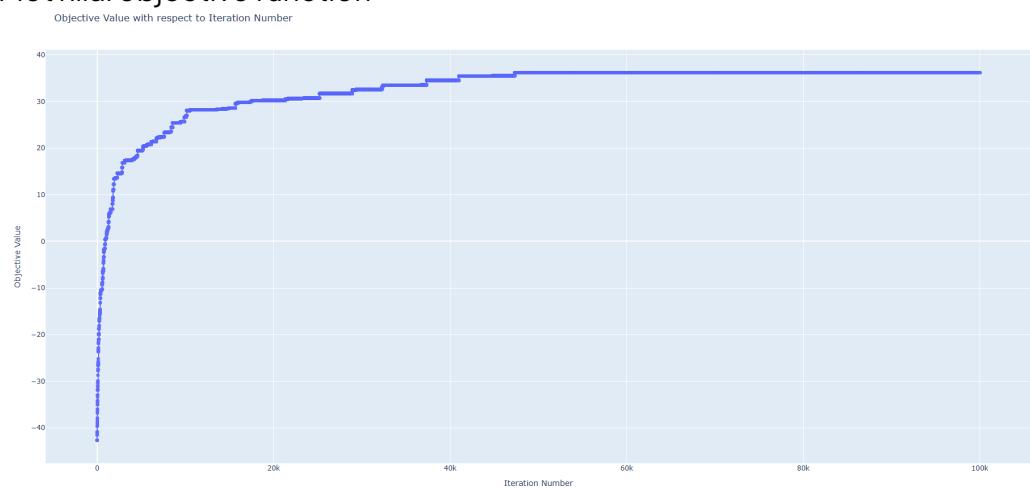
State Awal





Nilai objektif = 36.17

Plot nilai objective function



===== OUTPUT RESULTS =====

Final H: 36.17624808847797

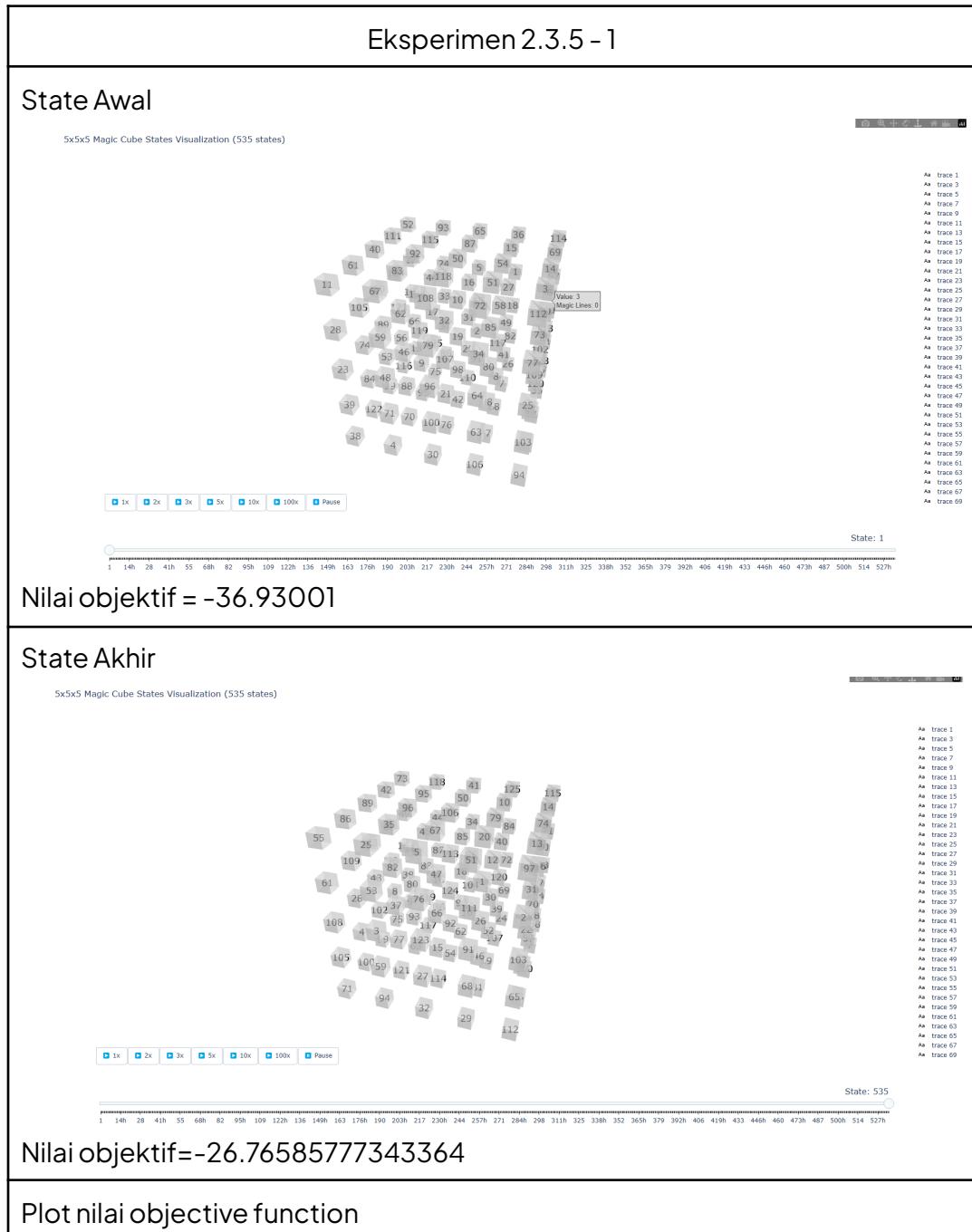
Duration: 3.895872

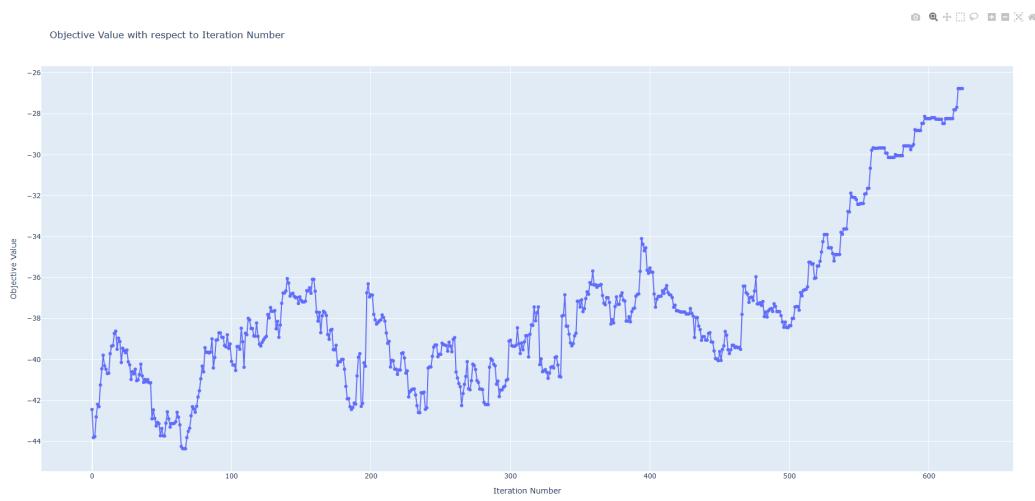
1

Banyak iterasi: 100000

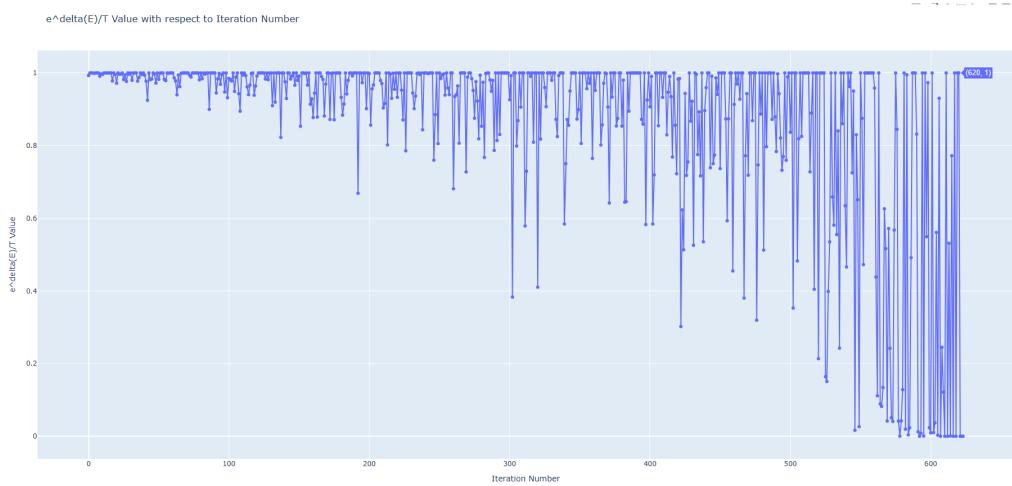
Durasi proses pencarian: 3.89 detik

2.3.5. Simulated Annealing





Plot e^{delta(E)/T}

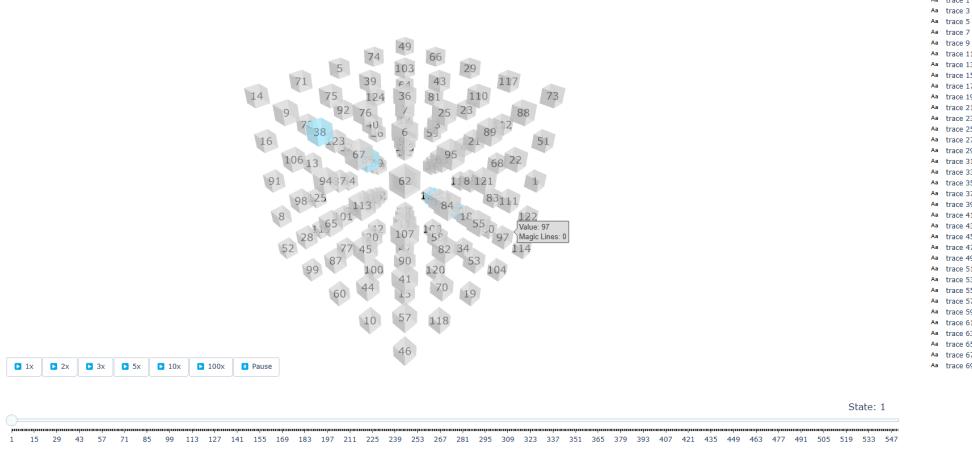


Frekuensi 'stuck' di local optima: 354
Durasi proses pencarian: 0.044076 detik

Eksperimen 2.3.5 - 2

State Awal

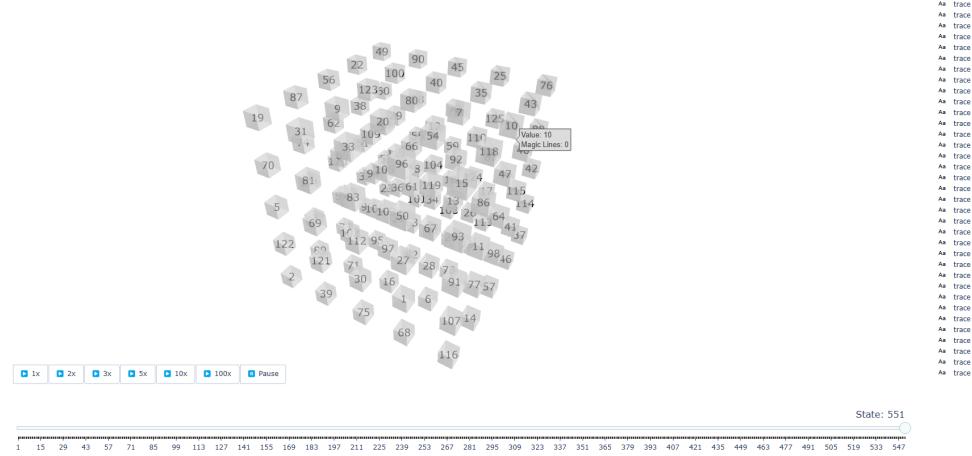
5x5x5 Magic Cube States Visualization (551 states)



Nilai objektif =

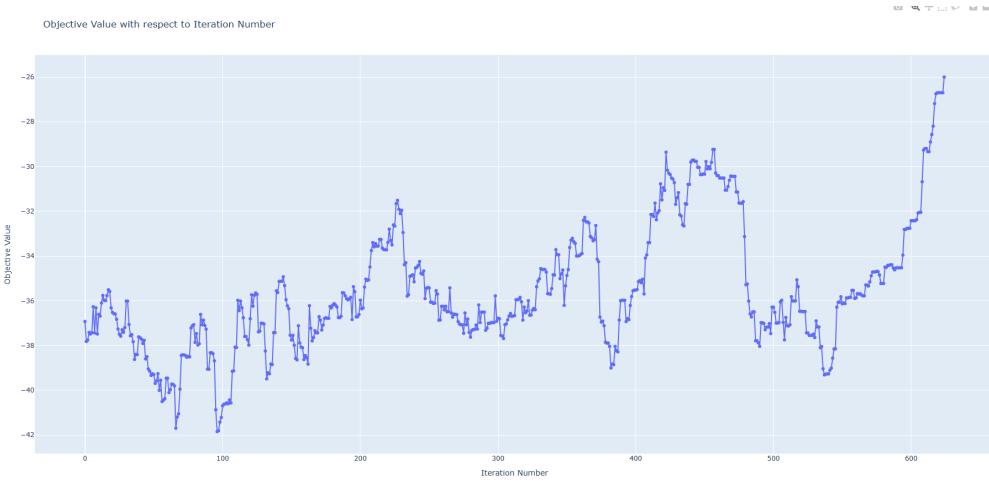
State Akhir

5x5x5 Magic Cube States Visualization (551 states)

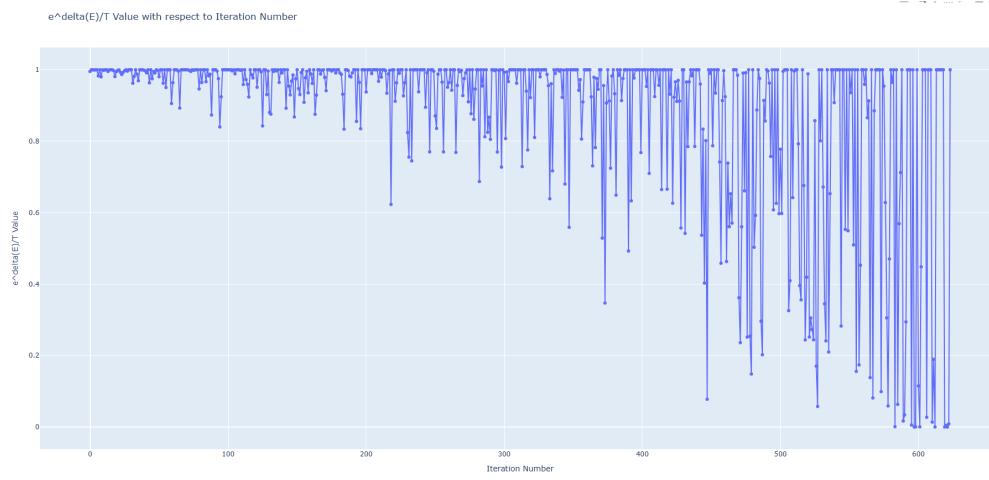


Nilai objektif= -25.9980318356287

Plot nilai objective function



Plot $e^{\delta(E)/T}$

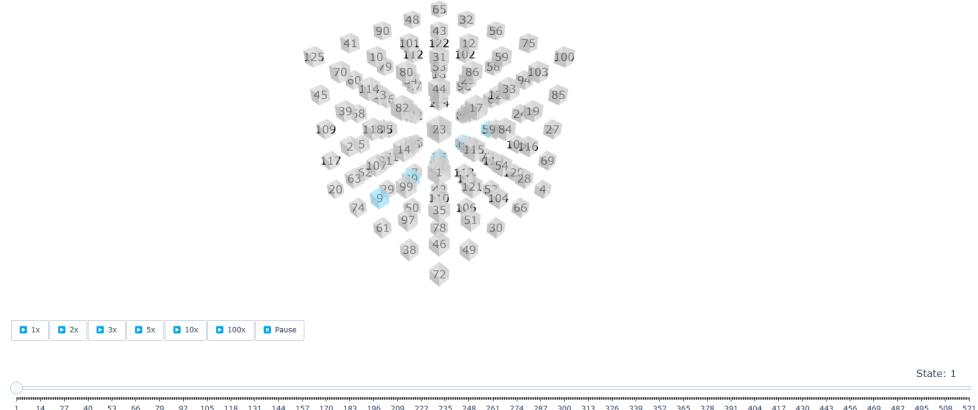


Frekuensi 'stuck' di local optima: 343
Durasi proses pencarian: 0.042056detik

Eksperimen 2.3.5 - 3

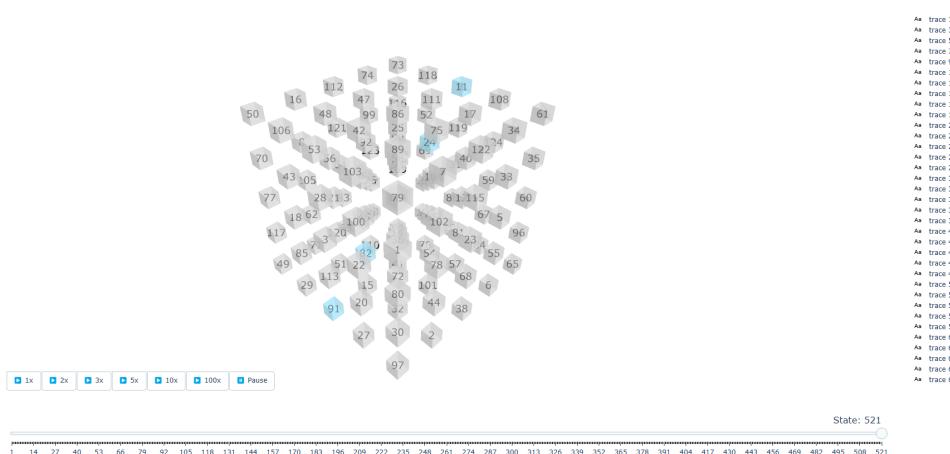
State Awal

5x5x5 Magic Cube States Visualization (521 states)

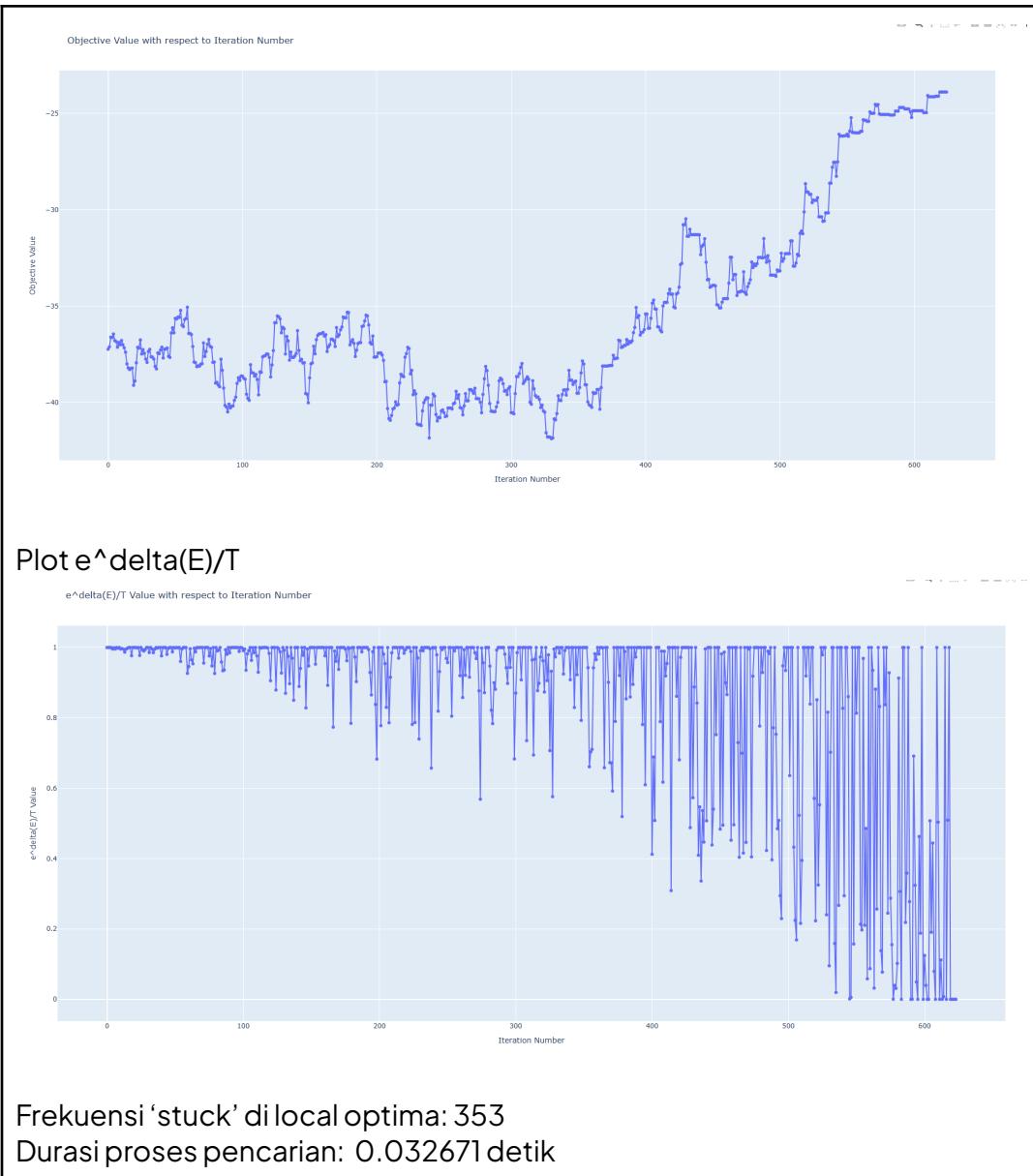


State Akhir

5x5x5 Magic Cube States Visualization (521 states)



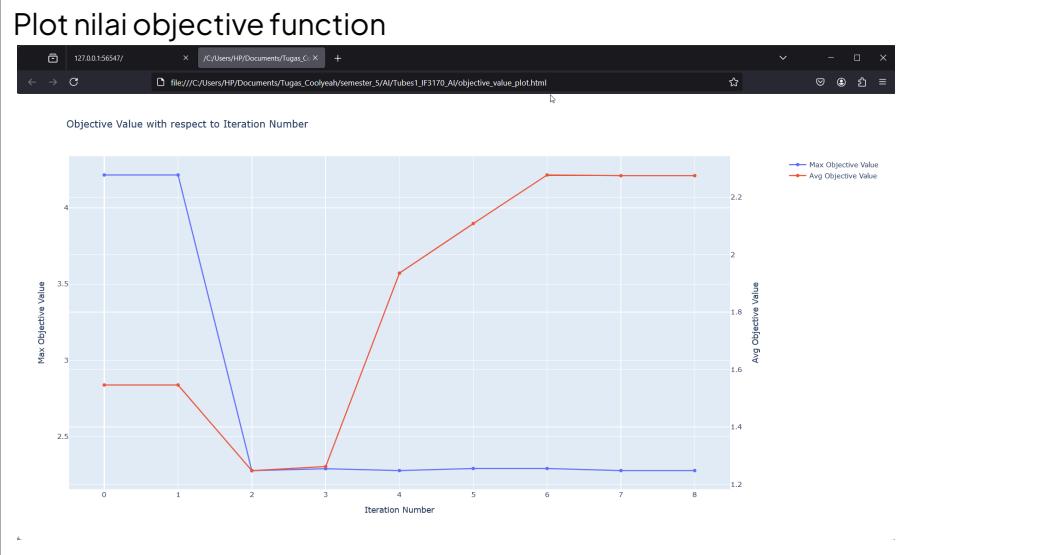
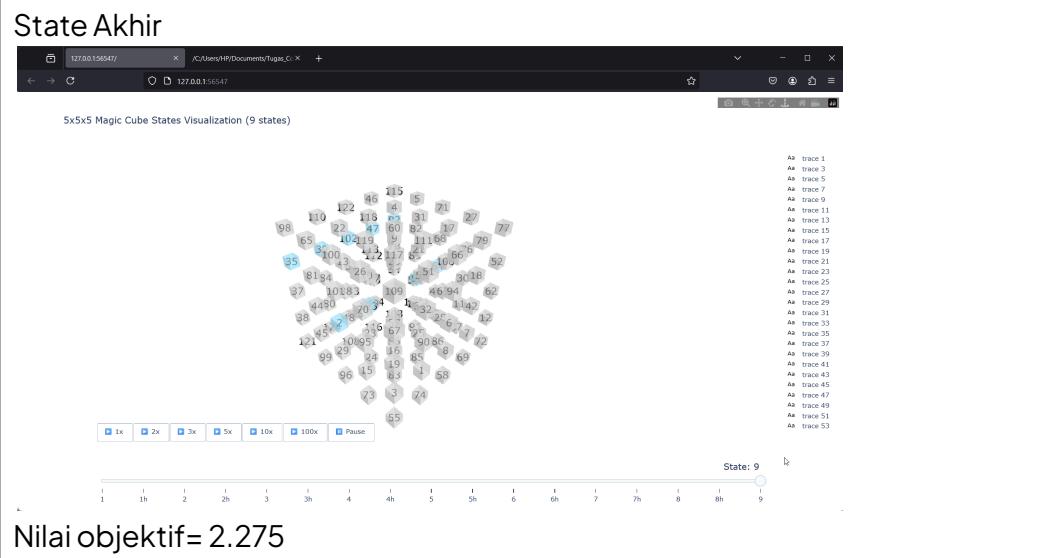
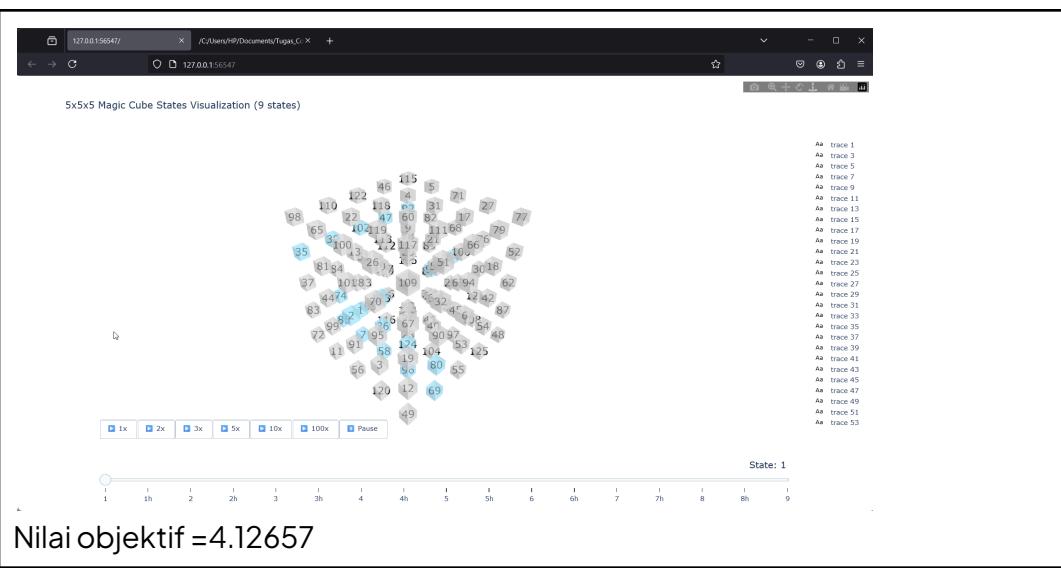
Plot nilai objective function



2.3.6. Genetic Algorithm

2.3.6.1. Variabel Kontrol Jumlah Populasi

Eksperimen 2.3.6.1 - 1
Jumlah Populasi: 6 Jumlah Iterasi: 8
State Awal



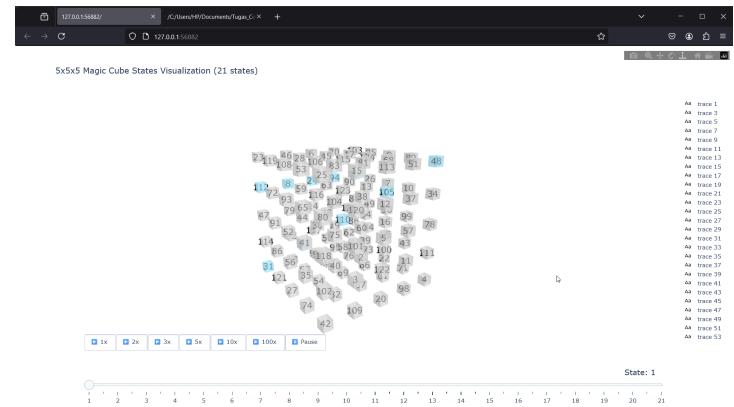
Durasi proses pencarian: 0.015000104904174805 detik

Eksperimen 2.3.6.1 - 2

Jumlah Populasi: 6

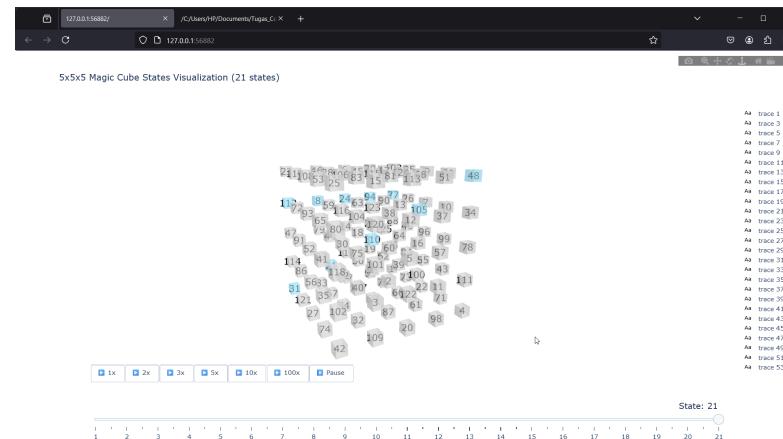
Jumlah Iterasi: 20

State Awal



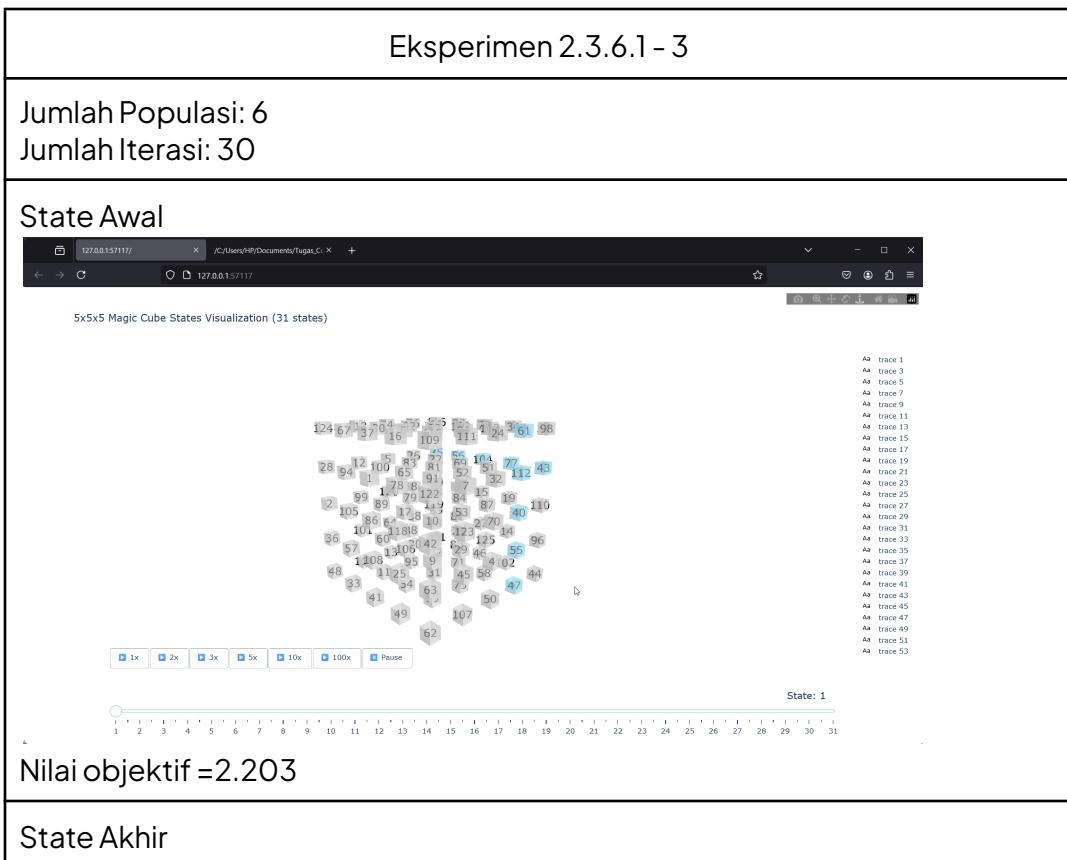
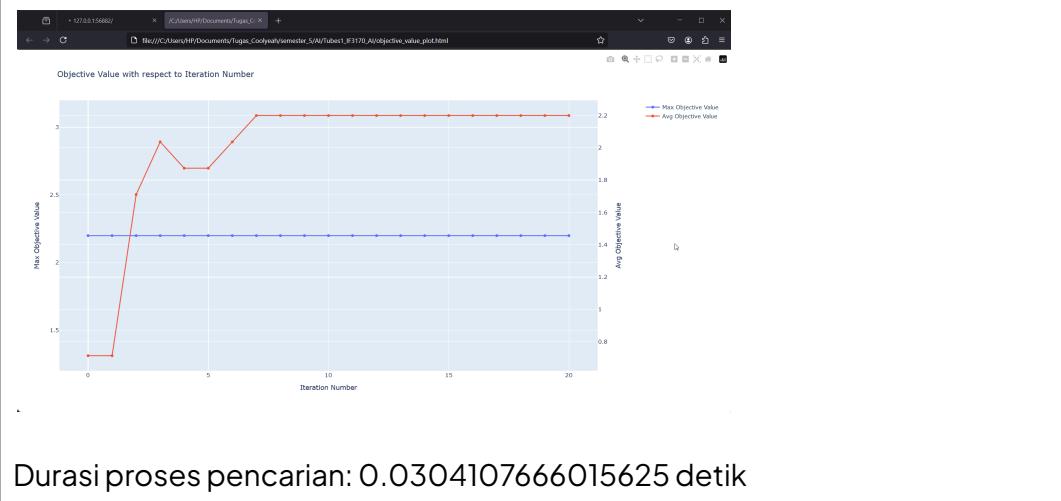
Nilai objektif = 2.1992354740061164

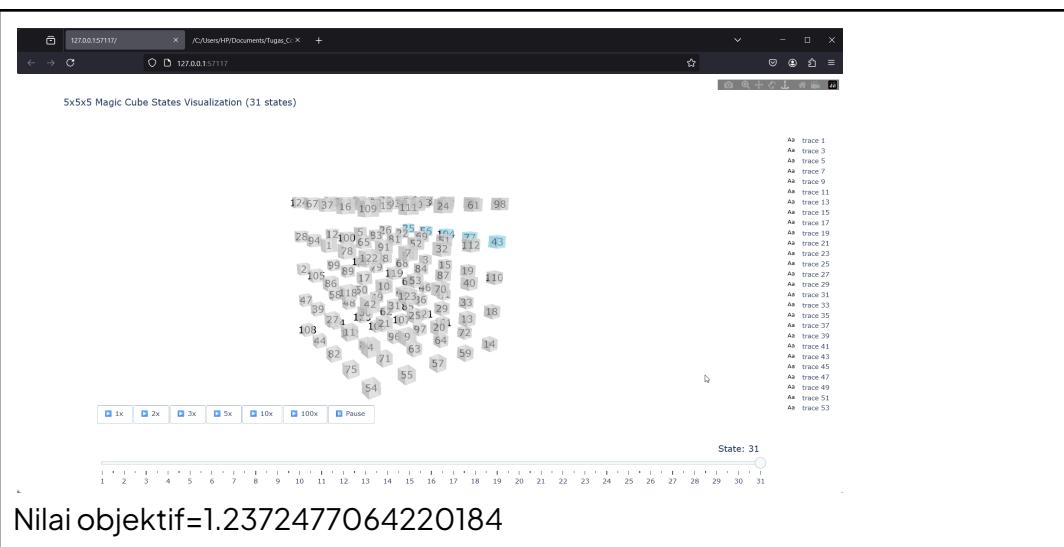
State Akhir



Nilai objektif= 2.1992354740061164

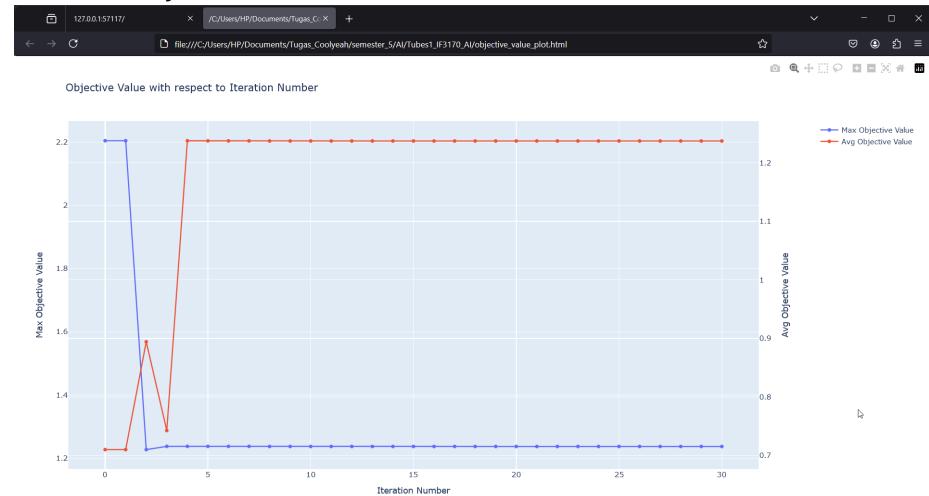
Plot nilai objective function





Nilai objektif=1.2372477064220184

Plot nilai objective function

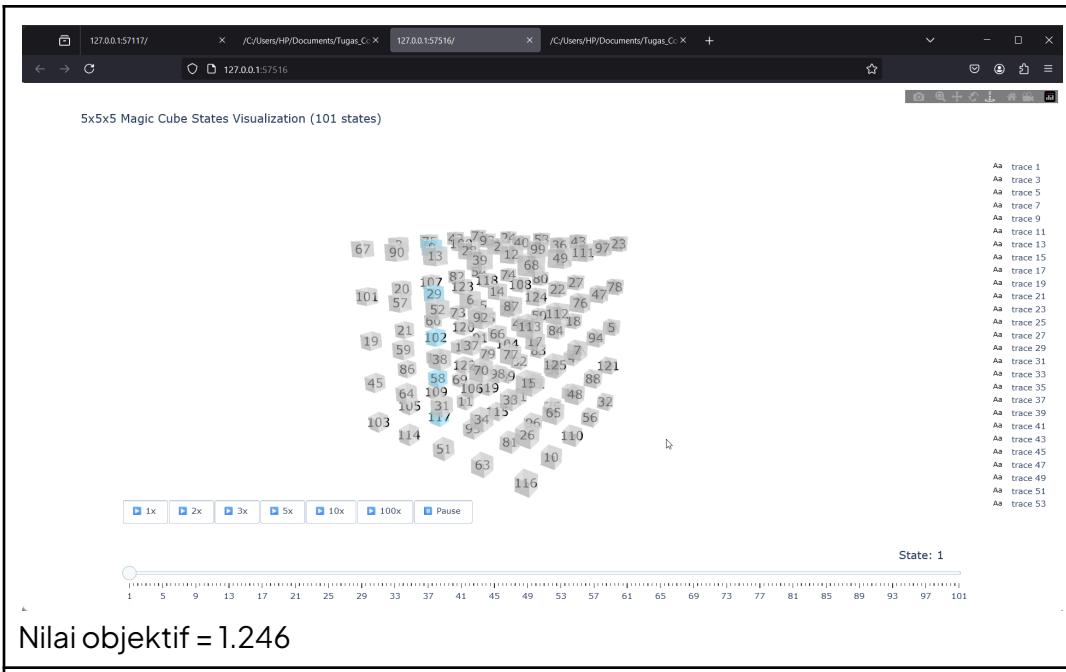


Durasi proses pencarian: 0.03402066230773926 detik

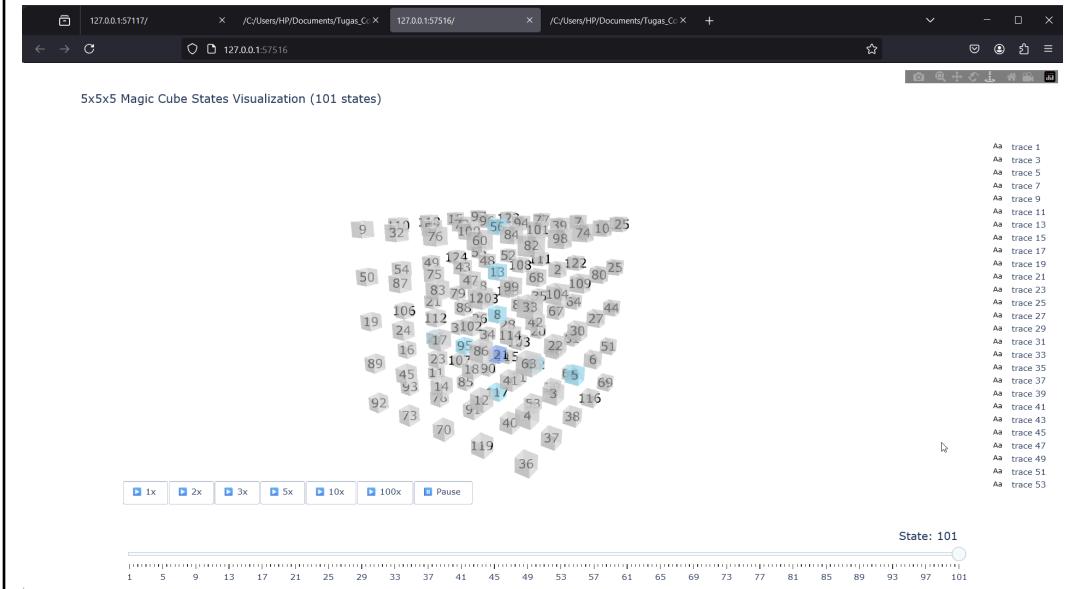
Eksperimen 2.3.6.1 - 4

Jumlah Populasi: 6
Jumlah Iterasi: 100

State Awal



State Akhir



Plot nilai objective function



Eksperimen 2.3.6.1 - 5
Jumlah Populasi: 6 Jumlah Iterasi: 100
State Awal [gambar] Nilai objektif =
State Akhir [gambar] Nilai objektif=2.2237003058103975
Plot nilai objective function [gambar]
Durasi proses pencarian: 0.14633679389953613 detik

Eksperimen 2.3.6.1 - 6
Jumlah Populasi: 6 Jumlah Iterasi: 75

<p>State Awal [gambar] Nilai objektif =</p>
<p>State Akhir [gambar] Nilai objektif=</p>
<p>Plot nilai objective function [gambar]</p> <p>Durasi proses pencarian: 53 detik</p>

Eksperimen 2.3.6.1 - 7
Jumlah Populasi: 6 Jumlah Iterasi: 100
<p>State Awal [gambar] Nilai objektif =</p>
<p>State Akhir [gambar] Nilai objektif=</p>
<p>Plot nilai objective function [gambar]</p> <p>Durasi proses pencarian: 53 detik</p>

Eksperimen 2.3.6.1 - 8
Jumlah Populasi: 6 Jumlah Iterasi: 4
<p>State Awal [gambar] Nilai objektif =</p>
<p>State Akhir [gambar] Nilai objektif=</p>

Plot nilai objective function
[gambar]

Durasi proses pencarian: 53 detik

Eksperimen 2.3.6.1 - 9

Jumlah Populasi: 6
Jumlah Iterasi: 120

State Awal
[gambar]
Nilai objektif =

State Akhir
[gambar]
Nilai objektif =

Plot nilai objective function
[gambar]

Durasi proses pencarian: 53 detik

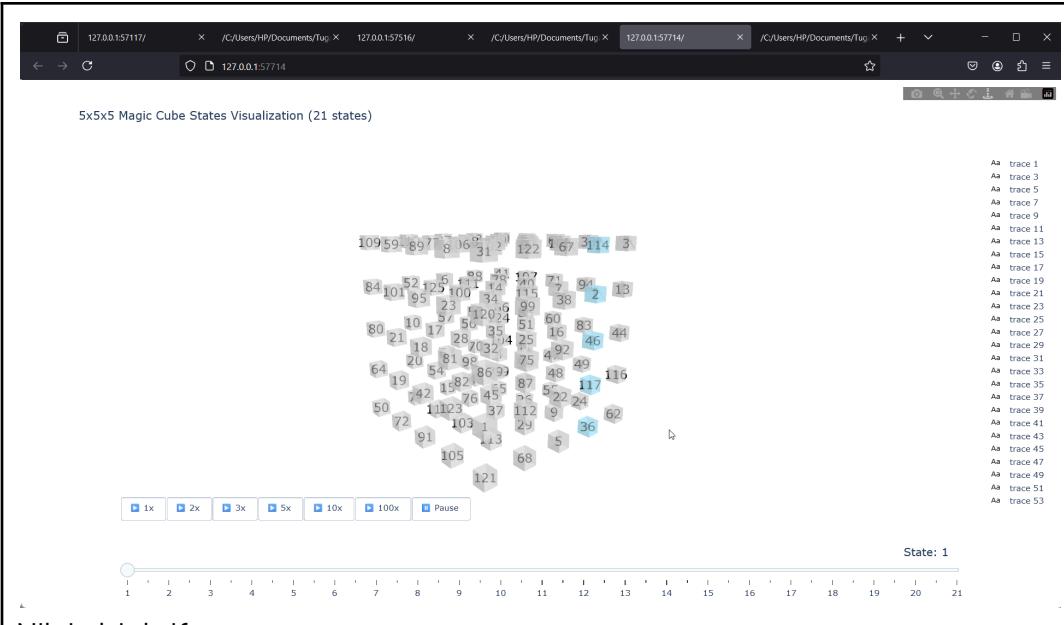
2.3.6.2. Variabel Kontrol Banyak Iterasi

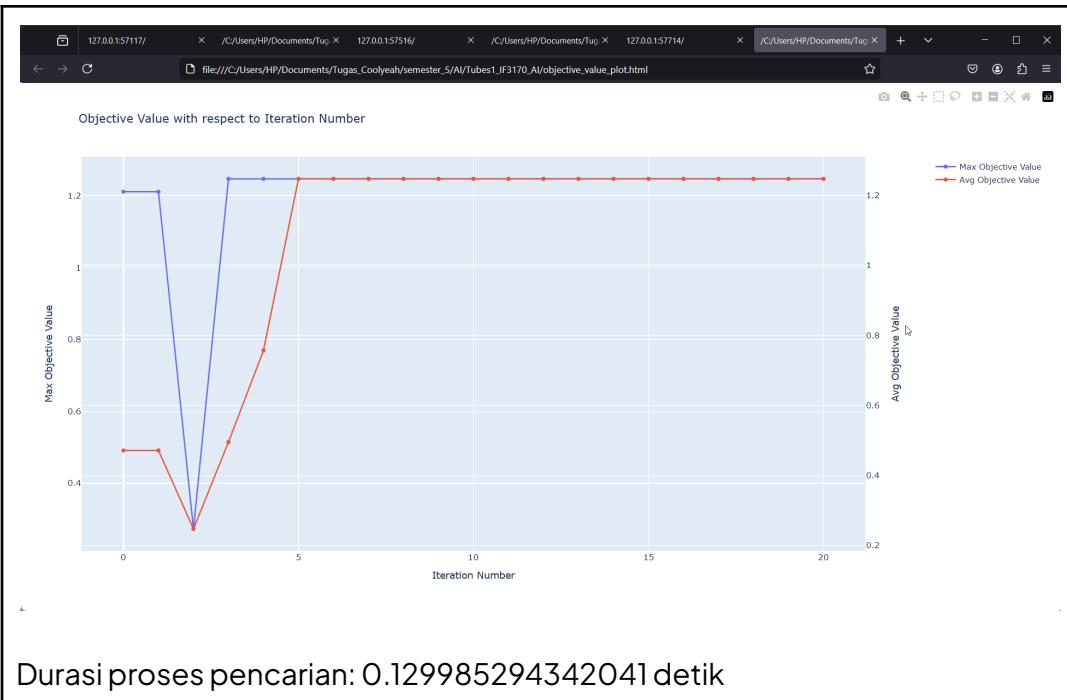
Banyak iterasi samain, jumlah populasi bedain

Eksperimen 2.3.6.2 - 1

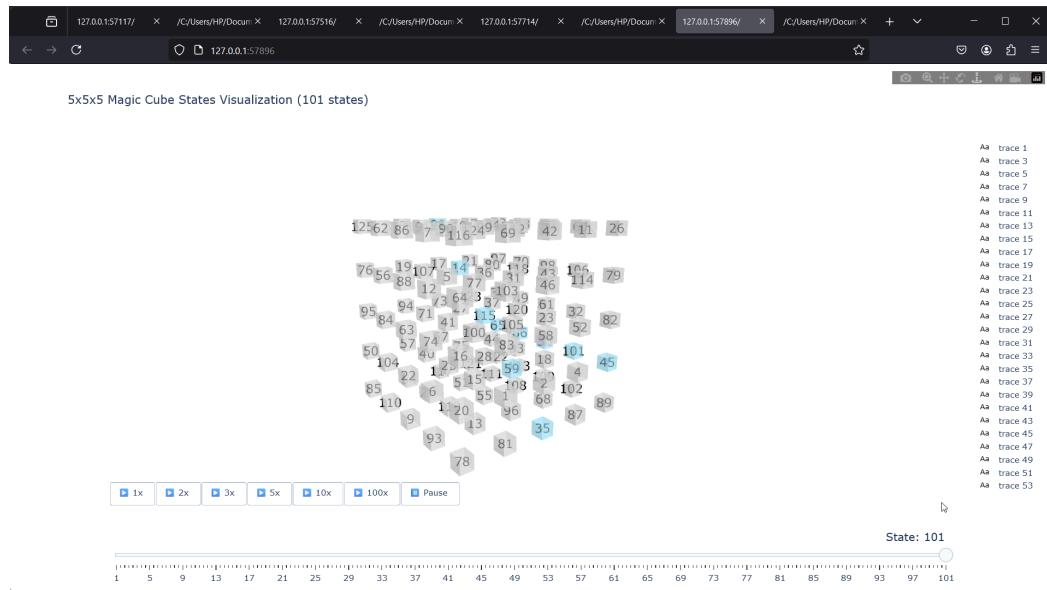
Jumlah Iterasi: 100
Jumlah Populasi: 4

State Awal



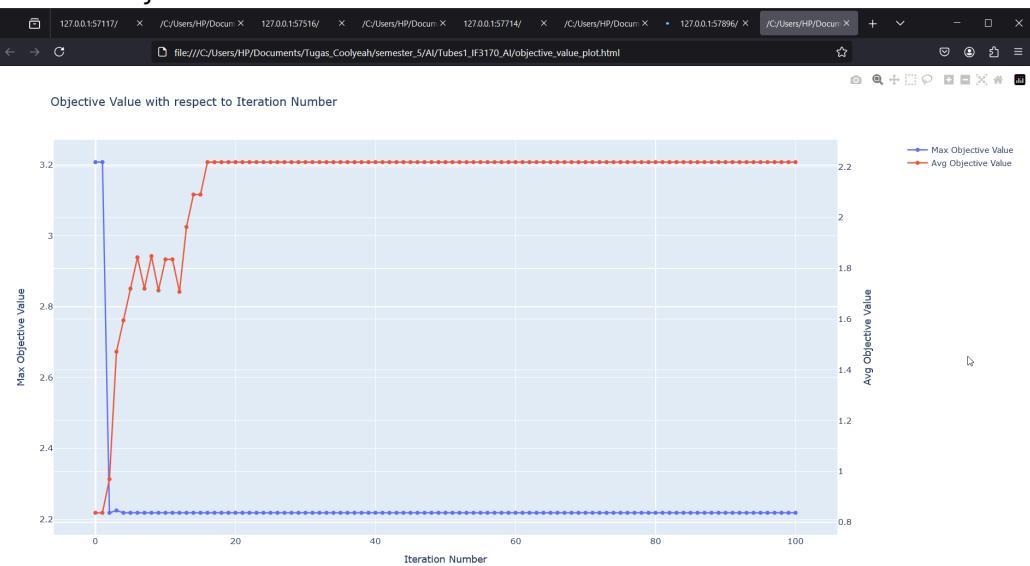


State Akhir



Nilai objektif=2.218532110091743

Plot nilai objective function



Durasi proses pencarian: 0.14196181297302246 detik

Eksperimen 2.3.6.2 - 3

Jumlah Iterasi: 100

Jumlah Populasi: 16

State Awal

[gambar]

Nilai objektif =

State Akhir

[gambar]

Nilai objektif=2.2440672782874618

Plot nilai objective function

[gambar]

Durasi proses pencarian: 0.28330087661743164 detik

Eksperimen 2.3.6.2 - 4

Jumlah Iterasi: 12

Jumlah Populasi: 32

State Awal

[gambar]

Nilai objektif =

State Akhir

[gambar]

Nilai objektif=

Plot nilai objective function

[gambar]

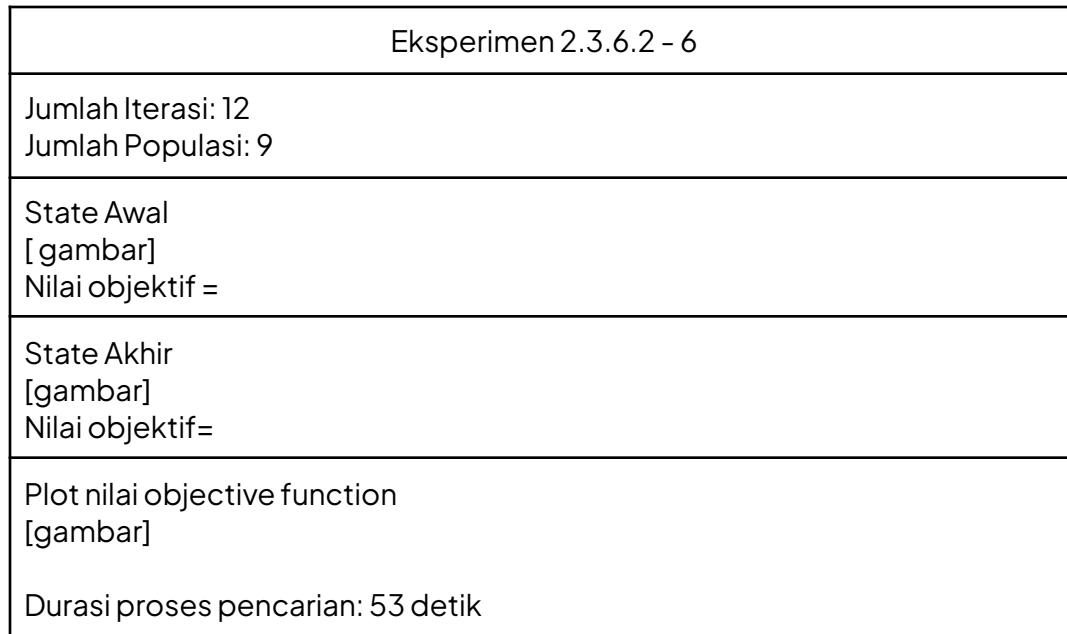
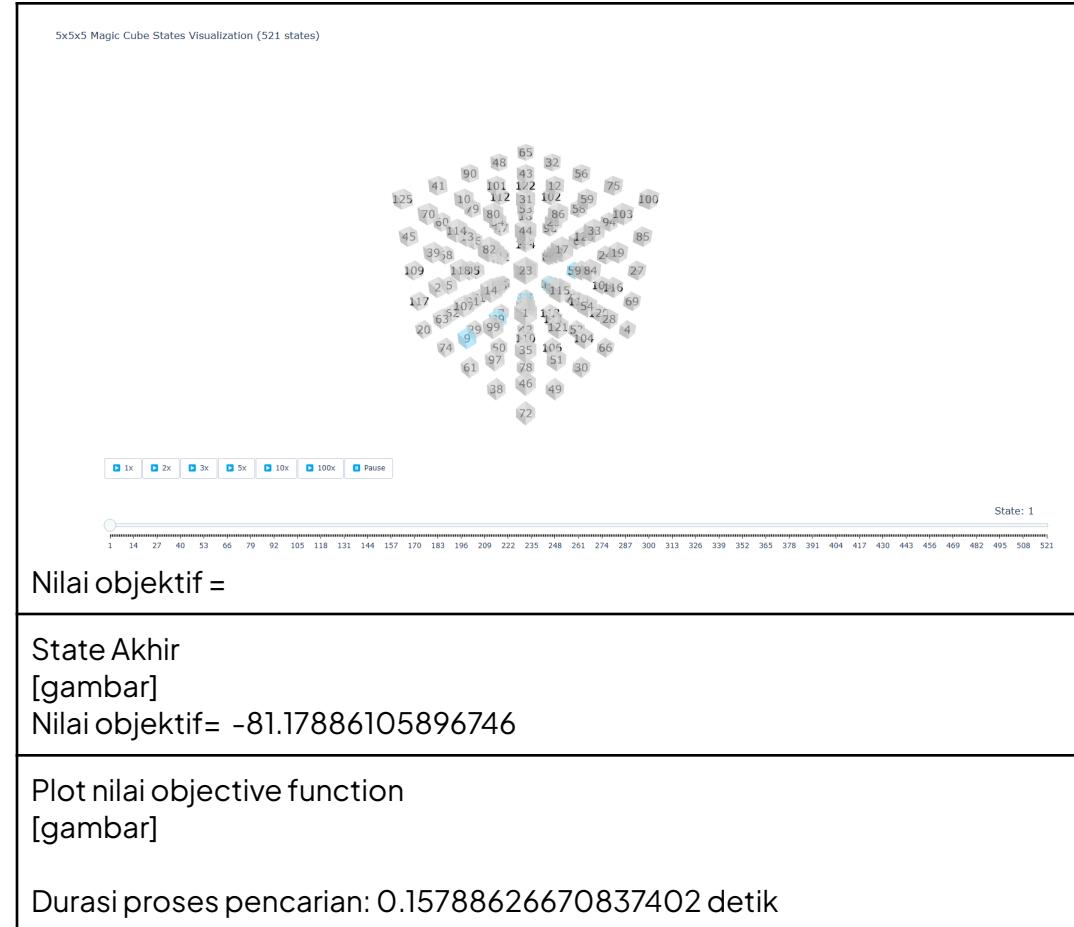
Durasi proses pencarian: 53 detik

Eksperimen 2.3.6.2 - 5

Jumlah Iterasi: 12

Jumlah Populasi: 9

State Awal



Eksperimen 2.3.6.2 - 7
Jumlah Iterasi: 12 Jumlah Populasi: 10
State Awal [gambar] Nilai objektif =
State Akhir [gambar] Nilai objektif=
Plot nilai objective function [gambar]
Durasi proses pencarian: 53 detik

Eksperimen 2.3.6.2 - 8
Jumlah Iterasi: 12 Jumlah Populasi: 10
State Awal [gambar] Nilai objektif =
State Akhir [gambar] Nilai objektif=
Plot nilai objective function [gambar]
Durasi proses pencarian: 53 detik

Eksperimen 2.3.6.2 - 9
Jumlah Iterasi: 12 Jumlah Populasi: 10

State Awal [gambar] Nilai objektif =
State Akhir [gambar] Nilai objektif=
Plot nilai objective function [gambar] Durasi proses pencarian: 53 detik

2.4. Analisis

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

Steepest Ascent Hill Climb akan memilih peningkatan nilai objektif yang terbesar dari state saat ini. Akan tetapi, algoritma ini sangat besar kemungkinannya untuk terjebak dalam lokal optima karena tidak mempertimbangkan kemungkinan langkah alternatif yang mungkin mengarah ke solusi yang lebih baik.

Hill Climb with Sideways Move mirip dengan Steepest Ascent HC, tetapi memperbolehkan gerakan atau transisi “flat”, yaitu berpindah ke state dengan nilai objektif yang sama. Namun, karena pada permasalahan ini fungsi objektif yang digunakan bersifat tidak diskret, akibatnya hill climb with sideways move memiliki performa yang secara efektif sama dengan steepest ascent.

Random Restart Hill Climb akan memilih suksesor secara acak. Jika terjebak dalam lokal optima, algoritma ini akan memulai kembali dari initial state sehingga dapat lebih mendekati global optima dengan cukup banyak percobaan, tetapi tidak menjamin. Dengan cara tersebut, eksplorasi ruang pencarian menjadi lebih luas, tetapi tetap bergantung pada jumlah restart dan distribusi initial state.

Stochastic Hill Climbing akan memilih suksesor secara acak dan memilih suksesor tersebut jika nilai objektifnya lebih baik. Ini lebih baik dari algoritma deterministik seperti steepest ascent, tetapi masih berisiko terjebak di lokal optima.

Simulated Annealing menggunakan konsep “annealing” (proses pendinginan) untuk menghindari lokal optima dengan kemungkinan menerima suksesor dengan nilai objektif yang lebih buruk, yang berkurang seiring waktu.

Algoritma ini sangat baik dalam mendekati global optima jika parameter T disetel dengan benar .

Genetic Algorithm dalam kasus penyelesaian Magic Cube 5x5x5 sangat tidak efektif karena algoritma ini terlalu acak. Jika tidak diatur dengan baik, proses pencarian terlihat tidak terarah atau terjebak di solusi yang buruk. Algoritma ini sangat bergantung terhadap parameternya yang bisa dijadikan variabel kontrol.

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

Steepest Ascent Hill Climb hasilnya cenderung kurang optimal dibandingkan algoritma lain yang lebih eksploratif karena hanya memilih langkah yang memberikan peningkatan terbesar tanpa mempertimbangkan alternatif.

Hill Climb with Sideways Move memiliki hasil yang serupa dengan Steepest Ascent karena seperti yang telah dijelaskan sebelumnya, secara efektif keduanya sama.

Random Restart Hill Climb hasil pencarinya bisa sangat bervariasi. Dalam beberapa kasus, algoritma ini dapat menemukan solusi yang lebih baik dan lebih cepat daripada algoritma pencarian lokal lainnya, tetapi tidak selalu menjamin hasil yang konsisten.

Stochastic Hill Climbing lebih efektif dibandingkan metode deterministik seperti Steepest Ascent karena memilih suksesor secara acak sehingga meningkatkan peluang menemukan solusi yang lebih baik.

Simulated Annealing sangat baik daripada metode algoritma lainnya dalam menemukan solusi yang optimal karena mekanisme “annealing” yang membantu menghindari lokal optima.

Genetic Algorithm sangat buruk dalam kasus ini.

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

Steepest Ascent Hill Climb relatif cepat dan yang paling singkat karena hanya melakukan evaluasi pada langkah yang memberikan peningkatan nilai objektif terbesar.

Hill Climb with Sideways Move memiliki hasil yang serupa dengan Steepest Ascent karena seperti yang telah dijelaskan sebelumnya, secara efektif keduanya sama.

Random Restart Hill Climb memiliki durasi yang sangat bervariasi tergantung pada jumlah restart yang dilakukan. Secara waktu komputasi, algoritma ini tentunya serupa dengan steepest ascent, tetapi membutuhkan waktu yang lebih lama karena melakukannya berulang kali. Untuk persoalan ini state space nya sangat luas, sehingga akan membutuhkan waktu yang sangat-sangat lama untuk menemukan solusi yang optimal.

Stochastic Hill Climbing sering lebih singkat dibandingkan Steepest Ascent. Namun, tidak menjamin hasilnya jauh lebih baik daripada Steepest Ascent.

Simulated Annealing memerlukan waktu pencarian yang lebih lama daripada algoritma lain. Namun, menawarkan potensi lebih untuk menemukan solusi yang lebih baik.

Genetic Algorithm memerlukan waktu paling lama karena melibatkan populasi yang besar dan proses seleksi, crossover, dan mutasi.

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

Steepest Ascent Hill Climb memberikan hasil akhir yang konsisten. Jika memulai dari initial state yang sama, hasilnya hampir selalu serupa, tetapi tidak selalu optimal.

Hill Climb with Sideways Move sedikit lebih bervariasi dibandingkan Steepest Ascent karena adanya gerakan flat yang bisa menghasilkan transisi pencarian yang berbeda. Namun, masih bisa dibilang konsistensinya tetap tinggi.

Random Restart Hill Climb memberikan hasil akhir yang sangat tidak konsisten. Hasilnya sangat bervariasi tergantung pada jumlah restart dan initial state yang dipilih.

Stochastic Hill Climbing memiliki konsistensi yang lebih baik dibandingkan Random Restart tetapi masih bervariasi tergantung pada langkah acak yang diambil.

Simulated Annealing cenderung lebih konsisten dalam menemukan solusi yang baik, apalagi dengan parameter (seperti laju pendinginan) diatur dengan baik. Hasil dapat bervariasi, tetapi sering kali tetap dalam rentang yang wajar.

Genetic Algorithm yang banyak menggunakan sifat acak dari populasi awal hingga mutasi memberikan hasil yang sangat tidak konsisten.

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

Semakin banyak individu dalam populasi, waktu yang dibutuhkan untuk melakukan evaluasi dan seleksi juga meningkat. Hal ini karena algoritma harus menghitung dan membandingkan lebih banyak solusi dalam setiap generasi.

Dengan populasi yang lebih besar, ada lebih banyak keragaman genetik yang memungkinkan algoritma menemukan solusi yang lebih baik. Ini dapat menghasilkan nilai fungsi objektif yang lebih tinggi, karena algoritma dapat mengeksplorasi ruang solusi yang lebih luas.

Banyak iterasi memungkinkan algoritma untuk melakukan lebih banyak seleksi dan penyilangan, yang bisa meningkatkan konvergensi ke solusi optimal. Namun, jika terlalu banyak iterasi dilakukan tanpa variasi yang cukup, algoritma bisa terjebak dalam solusi lokal.

Terdapat trade-off antara ukuran populasi, jumlah iterasi, dan waktu komputasi. Populasi yang sangat besar dengan iterasi yang banyak bisa memproduksi hasil yang lebih baik, tetapi juga membutuhkan lebih banyak waktu dan sumber daya komputasi.

Jumlah iterasi yang lebih tinggi memberi kesempatan bagi individu yang lebih baik untuk mendominasi populasi, tetapi jika tidak diimbangi dengan mekanisme untuk mempertahankan keragaman (seperti mutasi), ini dapat mengurangi kemampuan algoritma untuk menemukan solusi baru.

3. Kesimpulan dan Saran

Berdasarkan analisis yang dilakukan, algoritma Steepest Ascent Hill Climb dan Hill Climb with Sideways Move cenderung memiliki konsistensi yang tinggi namun sering terjebak dalam solusi lokal, yang mengakibatkan hasil akhir yang tidak optimal. Sementara itu, Random Restart Hill Climb dan Stochastic Hill Climbing menawarkan pendekatan yang lebih eksploratif, tetapi hasilnya bisa sangat bervariasi tergantung pada strategi pemilihan langkah dan kondisi awal. Simulated Annealing menunjukkan potensi yang kuat dalam menemukan solusi optimal berkat mekanisme pendinginannya, meskipun memerlukan waktu lebih lama untuk mencari solusi dibandingkan algoritma lainnya.

Di sisi lain, Genetic Algorithm terbukti kurang efektif dalam persoalan penyelesaian Magic Cube 5×5×5 karena sifatnya yang acak dan bergantung pada pengaturan parameter. Meskipun Genetic Algorithm memiliki keunggulan dalam eksplorasi ruang pencarian yang lebih luas, waktu komputasinya jauh lebih lama dibandingkan algoritma lainnya. Secara keseluruhan, Simulated Annealing muncul sebagai algoritma yang paling menjanjikan dalam mendekati global optima, sementara Steepest Ascent dan Hill Climb with Sideways Move menawarkan efisiensi waktu tetapi kurang dalam eksplorasi solusi yang optimal.

Pembagian tugas tiap anggota kelompok

GitHub Repository: https://github.com/abdulrafirh/Tubes1_IF3170_AI/

13522003	Visualisasi 3D kubus dan Laporan
13522014	Implementasi Kelas Cube dan Laporan
13522089	Implementasi Algoritma Local Search & Laporan
13522114	Visualisasi plot nilai dan Laporan

Referensi

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. 3rd ed. New Jersey: Pearson.

Magisch vierkant. (2014). *Features of the magic cube*. [online] Available at:
<https://www.magischvierkant.com/three-dimensional-eng/magic-features/>.

Trump.de. (2024). *Perfect Magic Cubes*. [online] Available at:
<https://www.trump.de/magic-squares/magic-cubes/cubes-1.html>.

Wikipedia Contributors (2024). *Magic cube*. [online] Wikipedia. Available at:
https://en.wikipedia.org/wiki/Magic_cube.