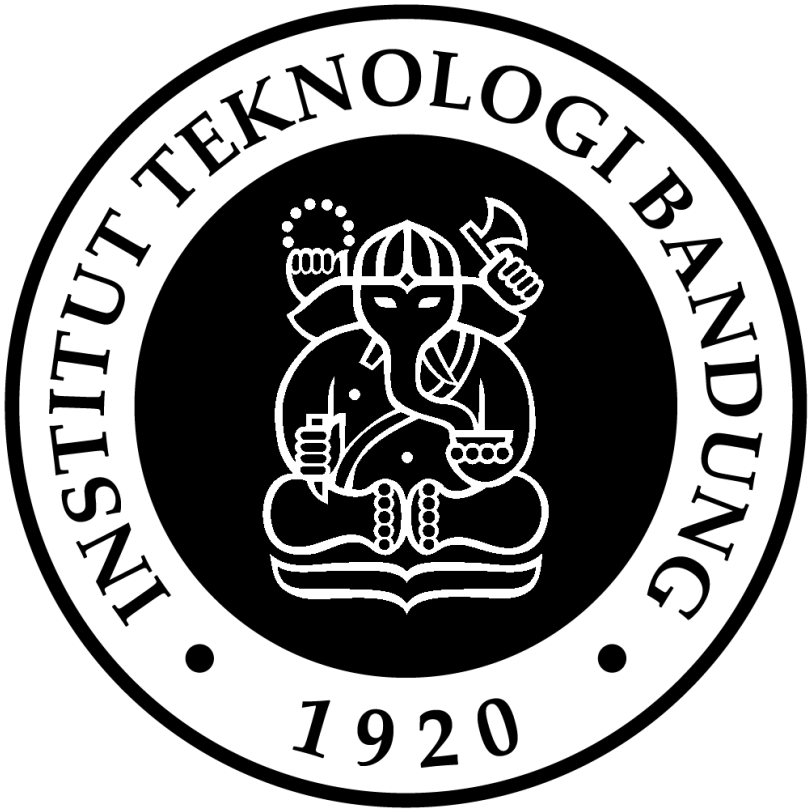
Laporan Tugas Besar 1 IF3170
Intelegensi Artifisial
Pencarian Solusi Diagonal Magic Cube dengan Local Search



Oleh:
Jimly Nur Arif / 13522123
Yosef Rafael Joshua / 13522133

DAFTAR ISI

DESKRIPSI PERSOALAN

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

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

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

PEMBAHASAN

Objective Function

- Steepest Ascent Hill Climbing
  Untuk algoritma ini digunakan objective function yang akan memberikan nilai terhadap state berdasarkan jumlah masing-masing row, column, pillar, diagonal sisi, dan diagonal ruang dari kubus 5x5x5. Setiap row, column, pillar, diagonal sisi, dan diagonal ruang akan dijumlahkan. Jika hasilnya tidak 315 (magic constant untuk kubus 5x5x5) maka nilai state dikurang ‘1’. Jika hasilnya sama dengan 315 maka nilai state ditambah ‘0’. Kasus paling baik adalah ketika semua row, column, pillar, diagonal sisi, dan diagonal ruang memiliki hasil penjumlahan sama dengan 315, sehingga nilai objective function untuk state tersebut adalah ‘0’. Kasus terburuk adalah ketika semua row, column, pillar, diagonal sisi, dan diagonal ruang tidak memiliki hasil penjumlahan sama dengan 315, sehingga nilai objective function untuk state tersebut adalah ‘-109’ (25 row, 25 column, 25 pillar, 30 diagonal sisi, 4 diagonal ruang)

- Simulated Annealing

Implementasi Algoritma

- Steepest Ascent Hill Climbing

  Objective function untuk steepest ascent

```
# the list must be a list of integer with length of exactly 125 elements
def rowValues(array:list) -> int:
    value = 0
    for i in range(0,25):
        j = i*5
        temp = array[j] + array[j+1] + array[j+2] + array[j+3] +
array[j+4]
        # print("temp: ", temp)
        if temp != 315:
            value -= 1

    return value
```

```
def pillarValues(array:list) -> int:
    value = 0
    for i in range(0,25):
        temp = array[i] + array[i + 25] + array[i + 50] + array[i + 75] +
array[i + 100]
        # print("temp:", temp)
        if temp != 315:
            value -= 1

    return value

def columnValues(array:list) -> int:
    value = 0
    for i in range(0, 101, 25):
        for j in range(i, i+5):
            temp = array[j] + array[j + 5] + array[j + 10] + array[j +
15] + array[j + 20]
            # print("temp:", temp)
            if temp != 315:
                value -= 1


    return value

def frontToBackSideDiagonalValues(array:list) -> int:
    value = 0
    for i in range(0, 101, 25):
        for j in range(i, i+5, 4):
            temp = 0
            if j % 5 == 0:
                temp = array[j] + array[j + 6] + array[j + 12] + array[j
+ 18] + array[j + 24]

            else:
                temp = array[j] + array[j + 4] + array[j + 8] + array[j +
12] + array[j + 16]
            # print("temp:", temp)
            if temp != 315:
```

```
                value -= 1

    return value

def leftToRightSideDiagonalValues(array:list) -> int:
    value = 0

    for i in range(0,5):
        for j in range(i, i+101, 100):
            # print(i,j)
            temp = 0
            if j == i:
                temp = array[j] + array[j + 30] + array[j + 60] + array[j
+ 90] + array[j + 120]
            else:
                temp = array[j] + array[j - 20] + array[j - 40] + array[j
- 60] + array[j - 80]
            # print("temp:", temp)
            if temp != 315:
                value -= 1

    return value

def upToDownSideDiagonalValues(array:list) -> int:
    value = 0

    for i in range(0, 21, 5):
        for j in range(i, i+5, 4):
            # print(i,j)
            temp = 0
            if j % 5 == 0:
                temp = array[j] + array[j + 26] + array[j + 52] + array[j
+ 78] + array[j + 104]
            else:
                temp = array[j] + array[j + 24] + array[j + 48] + array[j
+ 72] + array[j + 96]
            # print("temp:", temp)
            if temp != 315:
```

```
                value -= 1

    return value

def triagonalValues(array:list) -> int:
    value = 0
    first = array[0] + array[31] + array[62] + array[93] + array[124]
    second = array[4] + array[33] + array[62] + array[91] + array[120]
    third = array[20] + array[41] + array[62] + array[83] + array[104]
    fourth = array[24] + array[43] + array[62] + array[81] + array[100]



    if first != 315:
        value -=1

    if second != 315:
        value -=1

    if third != 315:
        value -=1

    if fourth != 315:
        value -=1

    return value

def objectiveFunctionSteepest(array:list) -> int:
    value = 0
    value += rowValues(array)
    value += pillarValues(array)
    value += columnValues(array)
    value += frontToBackSideDiagonalValues(array)
    value += leftToRightSideDiagonalValues(array)
    value += upToDownSideDiagonalValues(array)
    value += triagonalValues(array)
    return value
```

Di atas merupakan implementasi (source code) untuk objective function yang digunakan untuk steepest ascent hill climbing. Fungsi utama yang akan mengembalikan objective function dari sebuah state adalah **'objectiveFunctionSteepes**t' fungsi ini menerima list of integer (state kubus) dan mengembalikan integer (nilai objective function). Fungsi '**rowValues**', '**pillarValues**', '**columnValues**', '**frontToBackSideDiagonalValues**', '**leftToRightSideDiagonalValues**', '**upToDownSideDiagonalValues**', dan '**triagonalValues**' adalah fungsi untuk menghitung nilai untuk semua row, pilar, column, diagonal sisi, dan diagonal ruang. Fungsi utama akan menghitung nilai objective function dengan menjumlahkan hasil dari masing-masing fungsi di atas

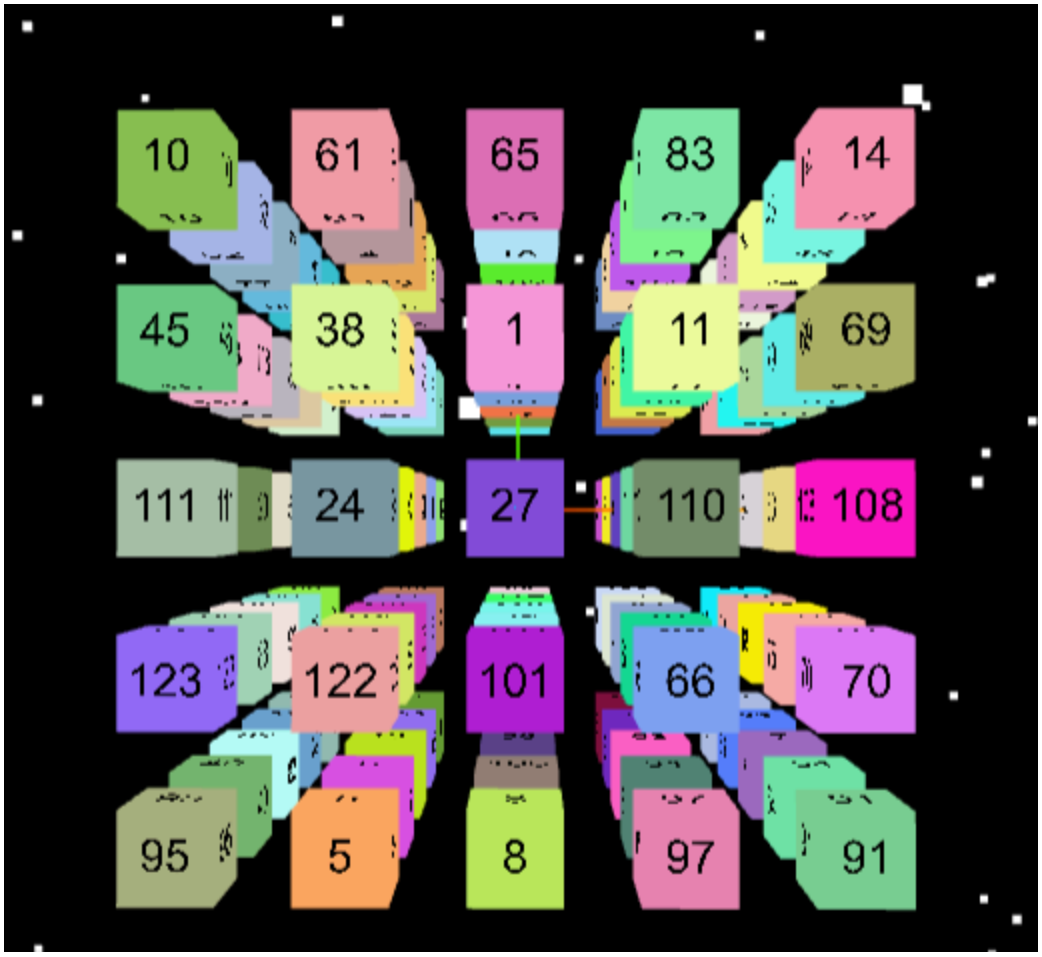
Steepest ascent hill climbing

```
from objectiveFunctionSteepest import objectiveFunctionSteepest
from typing import Tuple

def findSteepestNeighbor(array:list, objectiveValue:int) -> list:
    currentList = list(array)
    currentObjective = objectiveValue

    for i in range(125):
        for j in range(i+1, 125):
            tempArray = list(array)
            tempArray[i], tempArray[j] = tempArray[j], tempArray[i]
            tempObjective = objectiveFunctionSteepest(tempArray)

            # print("tempArray:", tempArray)
            # print("tempObj:", tempObjective)

            if tempObjective > currentObjective:
                currentList = tempArray
                currentObjective =tempObjective

    return currentList

def steepestAscent(array:list, objectiveValue:int, iteration:list) ->
tuple[list[int] , list[int]]: # the first element of tuple is the actual
states, the second is the iteration
```

```
    neighbor = findSteepestNeighbor(array, objectiveValue)
    iteration.append(objectiveFunctionSteepest(neighbor))
    # print("neighbor", neighbor)
    if neighbor == array:
        return neighbor, iteration
    else:

return(steepestAscent(neighbor,objectiveFunctionSteepest(neighbor),
iteration))
```

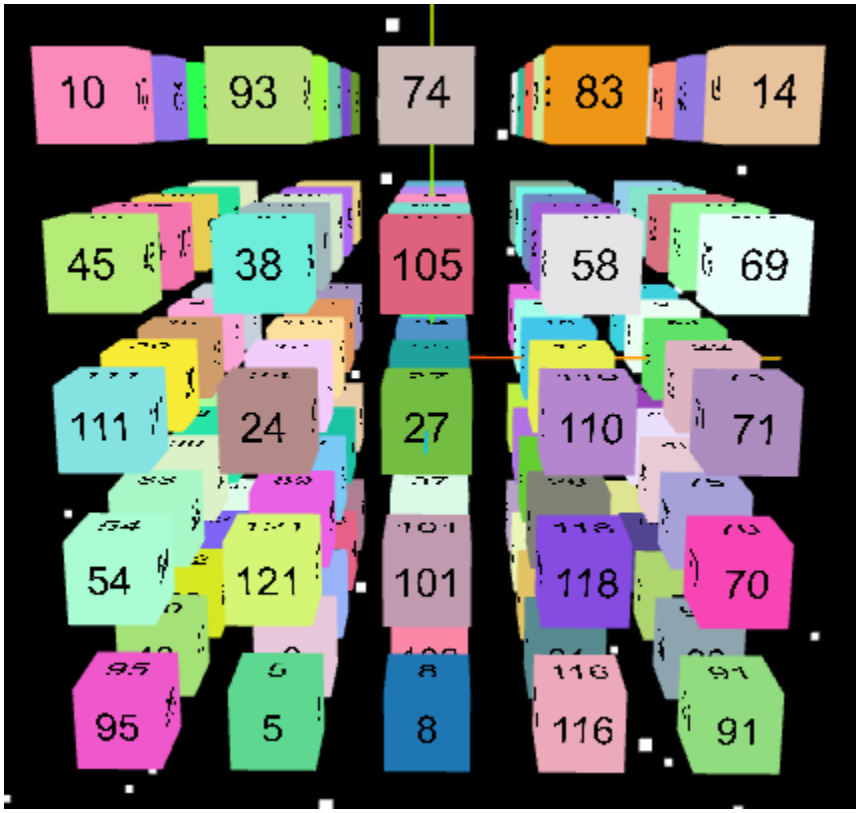
Di atas merupakan implementasi (source code) dari steepest ascent hill climbing. Implementasi di atas terdiri dari dua fungsi, yaitu '**steepestAscent**' dan '**findSteepestNeighbor**'. '**findSteepestNeighbor**' adalah fungsi untuk mencari neighbor dengan nilai objective function terbaik dari state awal. Pertama-tama akan dibuat salinan dari state masukan dan nilai objective function state masukan. Kemudian akan dicari semua neighbor dari state tersebut dengan cara menyalin state awal dan melakukan looping untuk swap 2 elemen yang ada. Jika nilai objective function dari hasil neighbor lebih baik, maka neighbor tersebut yang akan disimpan. Looping akan terus dilakukan hingga semua kemungkinan neighbor untuk state awal sudah ditemukan.
'**steepestAscent**' adalah fungsi rekursif untuk melakukan local search. Pertama-tama akan dilakukan pencarian neighbor dari state awal. Basis dari fungsi ini adalah jika neighbor yang ditemukan sama dengan state masukan. Ini menandakan bahwa tidak ada neighbor dengan nilai objective function yang lebih baik. Jika kasus ini tercapai maka dikembalikan neighbor dan iteration. Bagian rekursi adalah jika neighbor tidak sama dengan masukan awal. Ini menandakan bahwa ditemukan neighbor dengan nilai objective function yang lebih baik. Jika kasus ini tercapai maka akan dicari lagi neighbor selanjutnya dari neighbor sekarang (pemanggilan fungsi rekursif lagi). Fungsi ini mengembalikan tuple, elemen pertama adalah state hasil pencarian dan elemen kedua adalah array yang berisi nilai objective function di setiap iterasi (digunakan untuk plotting)

Eksperimen 1:
(awal)

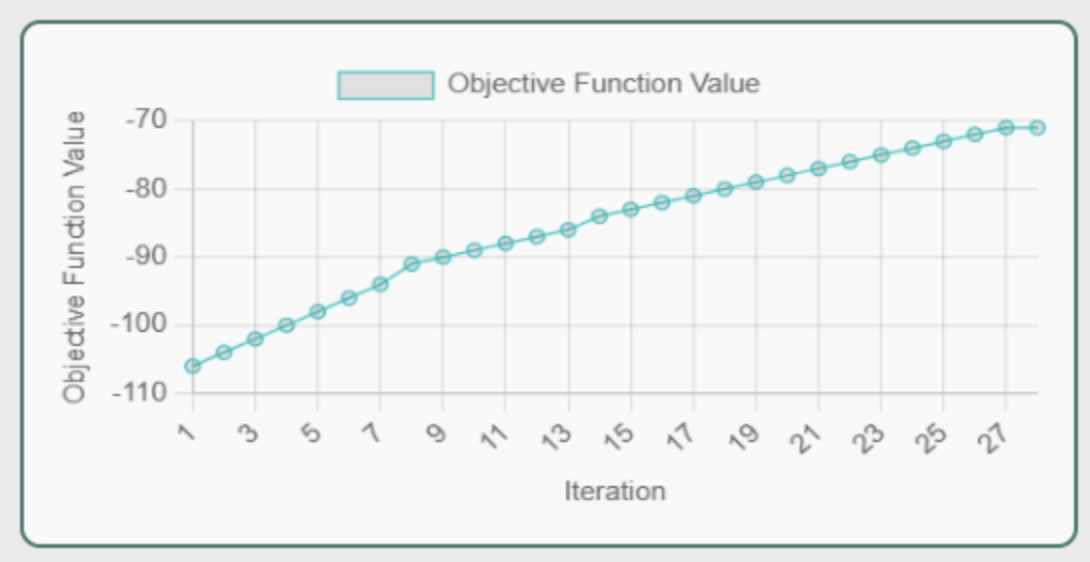


State:
[10, 61, 65, 83, 14, 45, 38, 1, 11, 69, 111, 24, 27, 110, 108, 123, 122, 101, 66, 70, 95, 5, 8, 97, 91, 62, 4, 18, 19, 43, 118, 100, 99, 81, 79, 20, 42, 55, 17, 63, 88, 89, 59, 26, 75, 40, 6, 106, 31, 54, 77, 113, 102, 107, 74, 86, 125, 49, 23, 114, 52, 121, 34, 16, 94, 96, 105, 119, 84, 48, 82,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

(akhir)

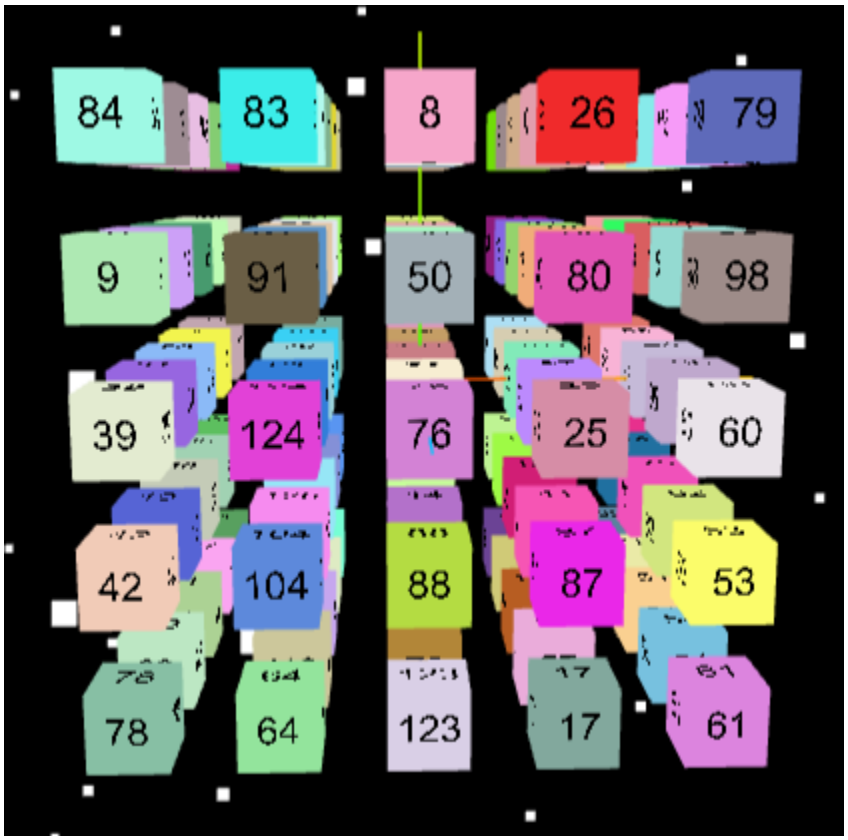


State:
[10, 93, 74, 83, 14, 45, 38, 105, 58, 69, 111, 24, 27, 110, 71, 54, 121, 101, 118, 70, 95, 5, 8, 116, 91, 63, 4, 18, 33, 43, 104, 68, 99, 123, 79, 20, 42, 55, 17, 22, 88, 89, 37, 26, 75, 40, 6, 106, 31, 96, 65, 113, 102, 107, 114, 86, 125, 49, 23, 25, 46, 122, 34, 16, 94, 36, 62, 119, 84, 48,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



Durasi: 8.300967693328857 detik
Banyak iterasi: 28

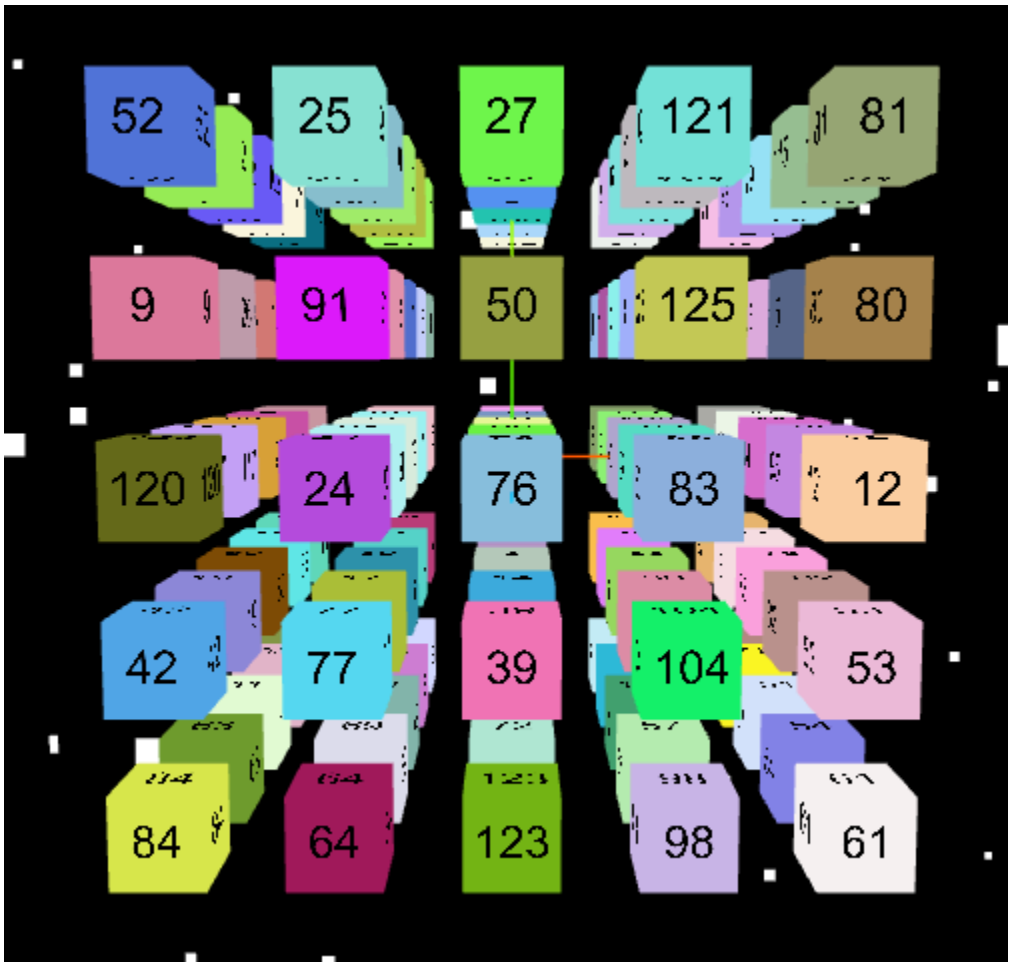
Eksperimen 2:
(awal)



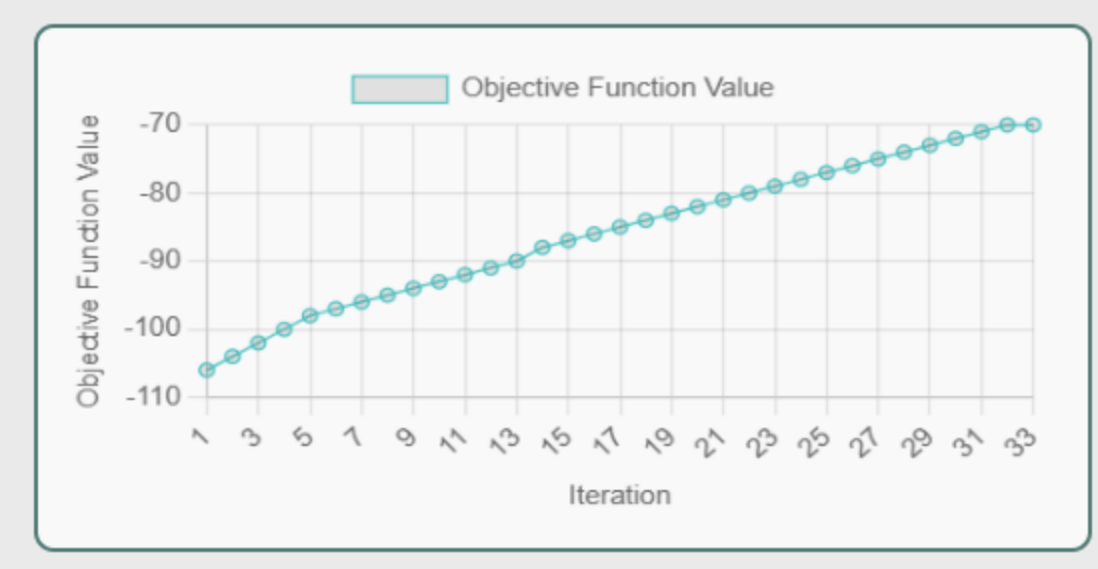
State:

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

(akhir)

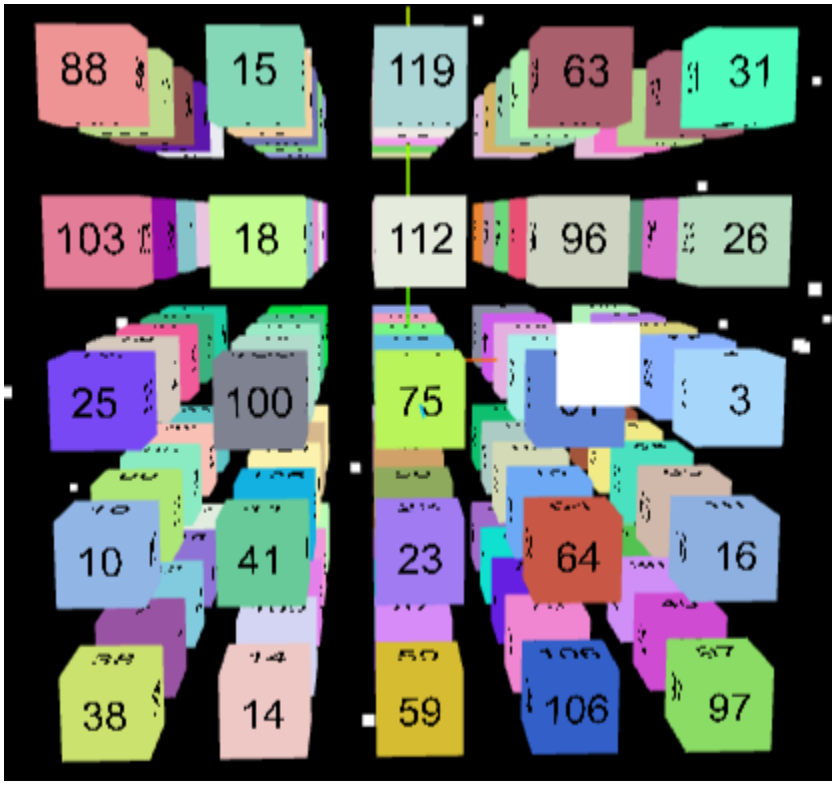


State:
[52, 25, 27, 121, 81, 9, 91, 50, 125, 80, 120, 24, 76, 83, 12, 42, 77, 39, 104, 53, 84, 64, 123, 98, 61, 2, 90, 8, 100, 115, 26, 6, 40, 18, 5, 117, 21, 68, 97, 95, 19, 17, 14, 30, 82, 63, 69, 72, 57, 54, 102, 10, 108, 16, 79, 118, 70, 78, 23, 105, 28, 73, 59, 103, 114, 60, 89, 3, 88, 75, 11, 49,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



Durasi: 10.447726011276245 detik
Banyak iterasi: 33

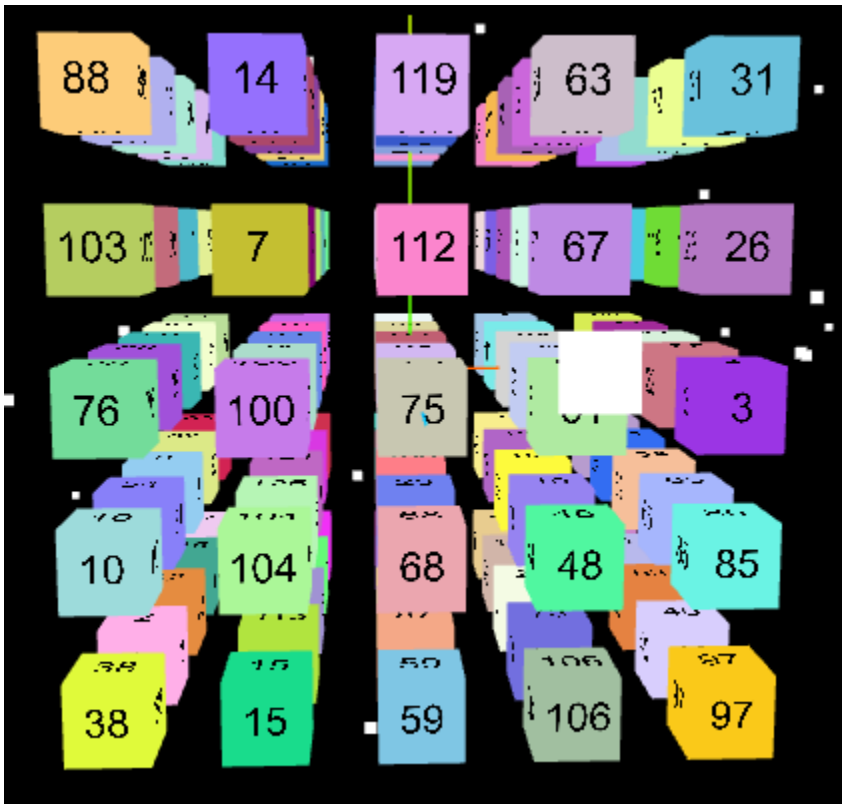
Eksperimen 3:
(awal)



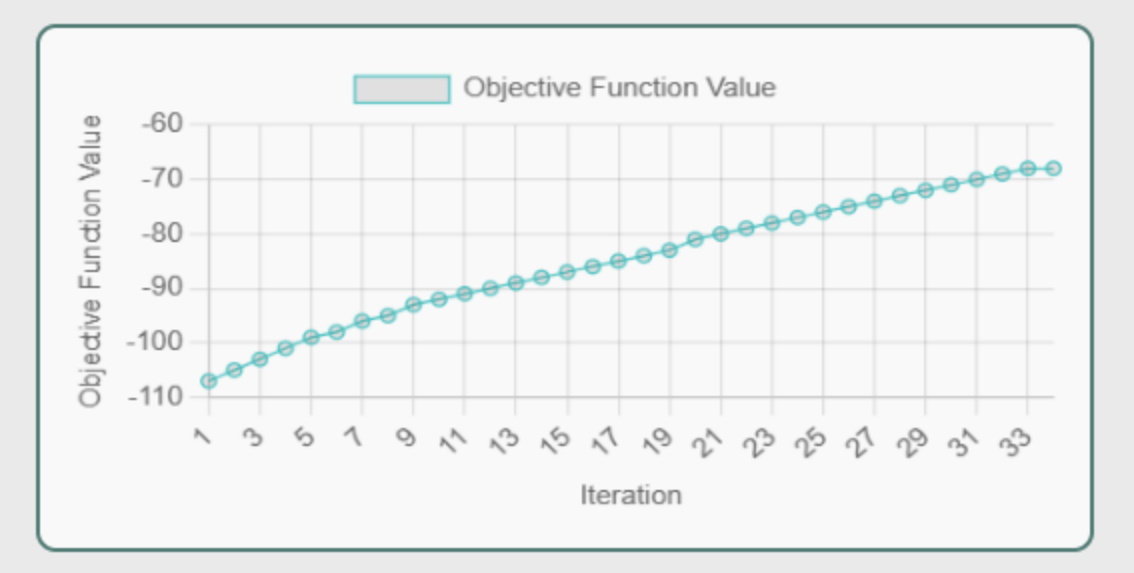
State:
[88, 15, 119, 63, 31, 103, 18, 112, 96, 26, 25, 100, 75, 61, 3, 10, 41, 23, 64, 16, 38, 14, 59, 106, 97, 111, 43, 80, 54, 72, 68, 79, 107, 4, 91, 12, 71, 98, 67, 22, 66, 125, 50, 19, 65, 7, 109, 87, 73, 40, 56, 30, 2, 82, 29, 1, 108, 33, 27, 81, 39, 55, 122, 35, 86, 85, 124,

37, 92, 69, 42, 95, 11, 76, 36, 44, 84, 5, 32, 116, 114, 120, 17, 89, 101, 51, 113, 53, 117, 90, 62, 94, 104, 13, 24, 115, 123, 49, 77, 9, 8, 48, 58, 20, 93, 118, 105, 121, 60, 45, 74, 78, 70, 6, 99, 83, 47, 34, 28, 102, 57, 21, 52, 46, 110]

(akhir)



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



Durasi: 7.336933851242065 detik
Banyak iterasi: 34

Analisis:
Dari ketiga hasil eksperimen, didapatkan bahwa nilai objective function di akhir adalah -71, -70, -68. Ini menandakan bahwa hasil pencarian dari algoritma Steepest Ascent Hill Climbing masih jauh dari global optima dan masih terjebak di local optima. Dibandingkan dengan algoritma lain, algoritma ini adalah algoritma yang paling cepat.

- Simulated Annealing

```
def objective_function(cube, x=None, y=None, z=None, old_val=None, new_val=None):
    """Compute score for deviation from the magic number for affected rows, columns, pillars, and diagonals."""
    total_difference = 0
    n = 5

    # If no specific cell provided, compute full objective function
    if x is None:
        # Sum rows, columns, pillars, and diagonals
        for z in range(n):
            for y in range(n):
                row_sum = sum(cube[z][y][x] for x in range(n))
                total_difference += abs(magic_number - row_sum)
```

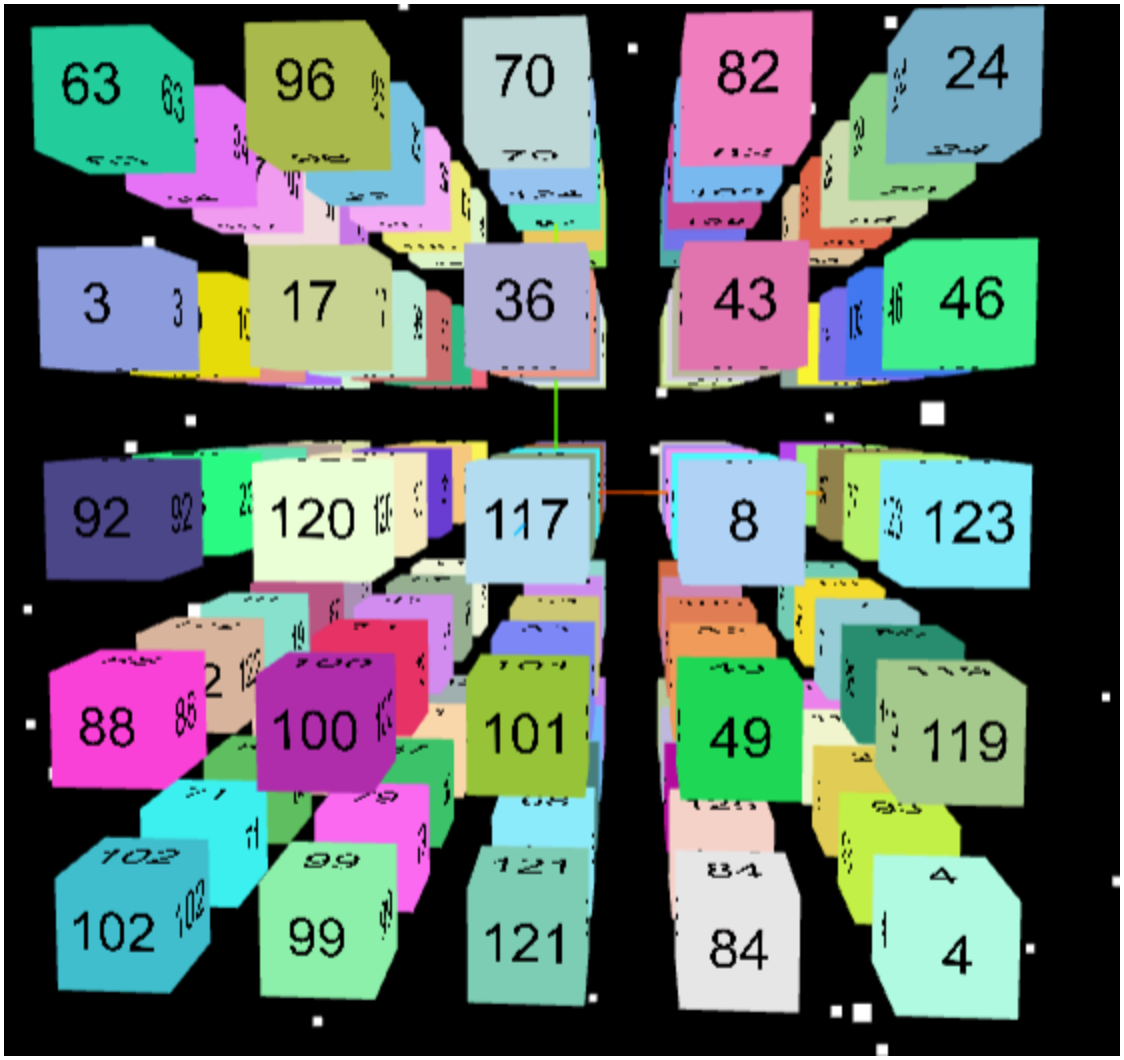
```
    for y in range(n):
        for x in range(n):
            col_sum = sum(cube[z][y][x] for z in range(n))
            total_difference += abs(magic_number - col_sum)

    for x in range(n):
        for z in range(n):
            pillar_sum = sum(cube[z][y][x] for y in range(n))
            total_difference += abs(magic_number - pillar_sum)

    # Diagonals for each plane
    for z in range(n):
        main_diag_xy = sum(cube[z][i][i] for i in range(n))
        anti_diag_xy = sum(cube[z][i][n - 1 - i] for i in range(n))
        total_difference += abs(magic_number - main_diag_xy)
        total_difference += abs(magic_number - anti_diag_xy)

    for y in range(n):
        main_diag_xz = sum(cube[i][y][i] for i in range(n))
        anti_diag_xz = sum(cube[i][y][n - 1 - i] for i in range(n))
        total_difference += abs(magic_number - main_diag_xz)
        total_difference += abs(magic_number - anti_diag_xz)

    for x in range(n):
        main_diag_yz = sum(cube[i][i][x] for i in range(n))
        anti_diag_yz = sum(cube[n - 1 - i][i][x] for i in range(n))
        total_difference += abs(magic_number - main_diag_yz)
        total_difference += abs(magic_number - anti_diag_yz)

    # 3D diagonals
    diag1_sum = sum(cube[i][i][i] for i in range(n))
    diag2_sum = sum(cube[i][i][n - 1 - i] for i in range(n))
    diag3_sum = sum(cube[i][n - 1 - i][i] for i in range(n))
    diag4_sum = sum(cube[n - 1 - i][i][i] for i in range(n))

    total_difference += abs(magic_number - diag1_sum)
    total_difference += abs(magic_number - diag2_sum)
    total_difference += abs(magic_number - diag3_sum)
```

```
        total_difference += abs(magic_number - diag4_sum)

    return total_difference

# Tambahkan variabel global untuk menyimpan nilai objective function per
iterasi


def simulated_annealing(nums):
    global iteration_data
    iteration_data = []  # Reset data iterasi

    T = 100.0      # Initial temperature
    T_min = 0.001  # Minimum temperature
    alpha = 0.9999  # Cooling rate
    n = 5          # Cube dimension
    local_optima_count = 0  # Counter for stuck in local optima

    current_state = [nums[i * 25:(i + 1) * 25] for i in range(5)]
    current_state = [[current_state[z][y * 5:(y + 1) * 5] for y in
range(5)] for z in range(5)]
    current_cost = objective_function(current_state)

    # Simpan nilai objective function awal
    iteration_data.append(current_cost)

    while T > T_min:
        z1, y1, x1 = random.randint(0, n - 1), random.randint(0, n - 1),
random.randint(0, n - 1)
        z2, y2, x2 = random.randint(0, n - 1), random.randint(0, n - 1),
random.randint(0, n - 1)

        old_val_1, old_val_2 = current_state[z1][y1][x1],
current_state[z2][y2][x2]
        current_state[z1][y1][x1], current_state[z2][y2][x2] = old_val_2,
old_val_1

        new_cost = objective_function(current_state)
```

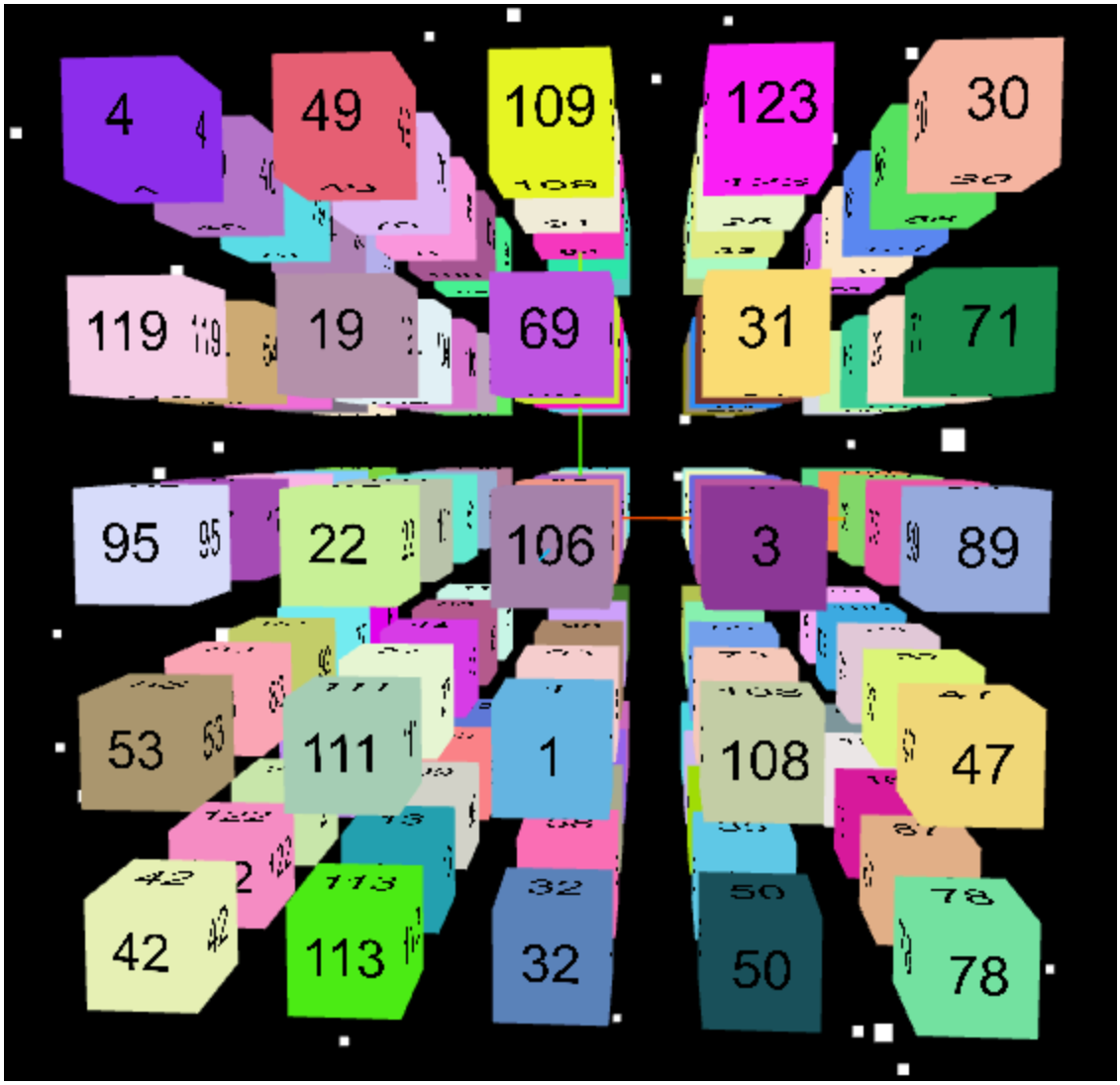
```
        if new_cost < current_cost or random.random() <
math.exp((current_cost - new_cost) / T):
            current_cost = new_cost
        else:
            current_state[z1][y1][x1], current_state[z2][y2][x2] =
old_val_1, old_val_2
            local_optima_count += 1  # Increment counter for local optima

        T *= alpha

        # Simpan nilai objective function setelah setiap iterasi
        iteration_data.append(current_cost)

    final_state = [num for layer in current_state for row in layer for
num in row]

    # Save the local optima count
    global stuck_local_optima
    stuck_local_optima = local_optima_count

    return final_state
```

Di atas merupakan implementasi (source code) dari algoritma Simulated Annealing. Pertama-tama dilakukan inisialisasi untuk T, T_min, alpha, n, dan local optima count. Kemudian dilakukan looping untuk pencarian. Loop akan berjalan selama T > T_min. Pada setiap iterasi dari loop tersebut, akan dilakukan swap dua elemen secara random. State yang didapatkan akan dicari nilai objective functionnya. Kalau nilai objective function yang didapatkan lebih kecil maka state tersebut yang akan diambil. Jika nilai objective function yang didapatkan lebih besar maka tergantung pada temperatur dan perbedeaan cost yang ada. Kemudian di bagian bawah loop ada pengurangan temperatur. Kemudian ada variabel-variabel yang digunakan untuk eksperimen.

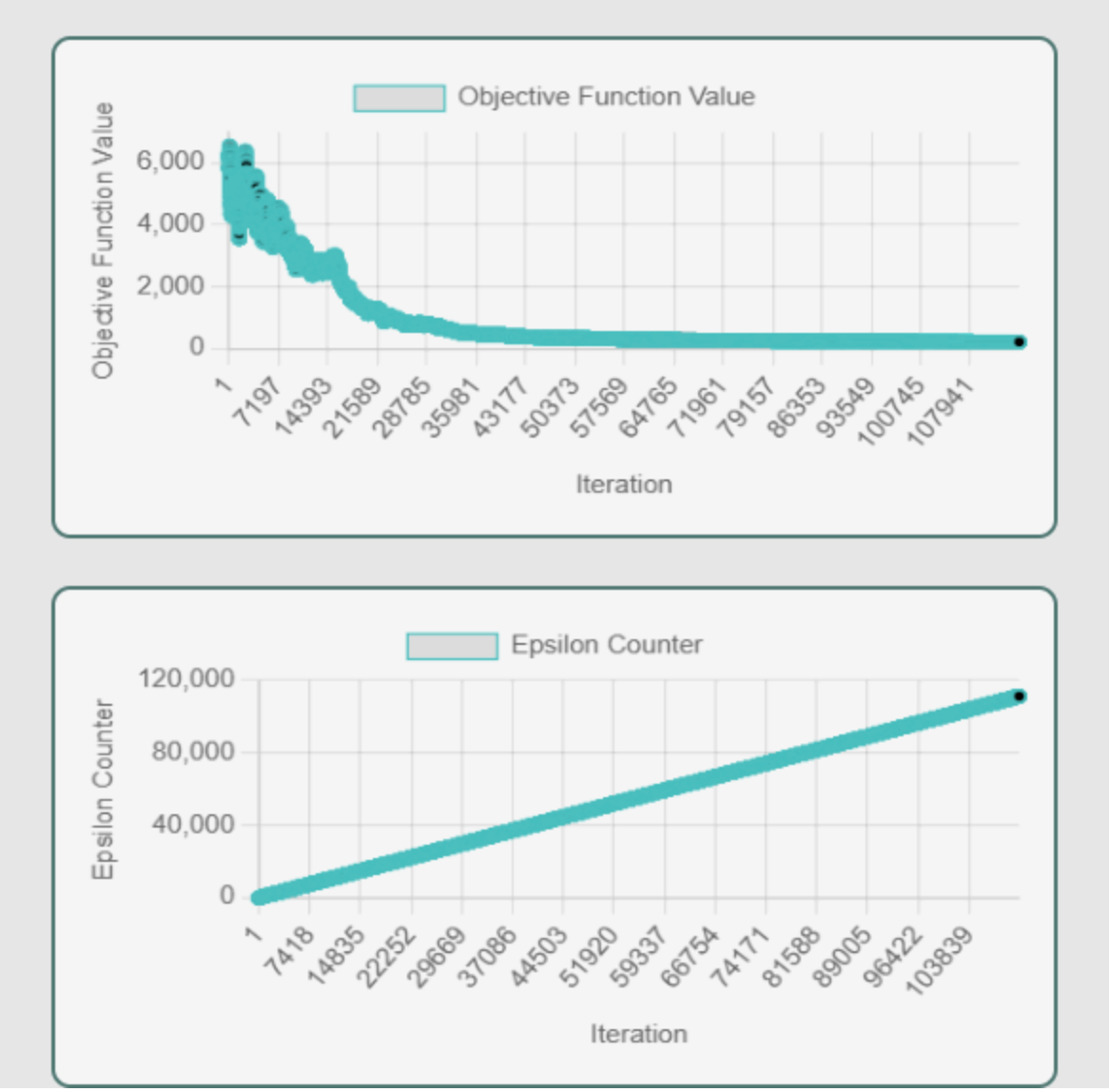
Eksperimen 1:
(awal)


63
96
70
82
24
3
17
36
43
46
92
120
117
8
123
88
100
101
49
119
102
99
121
84
4

(akhir)

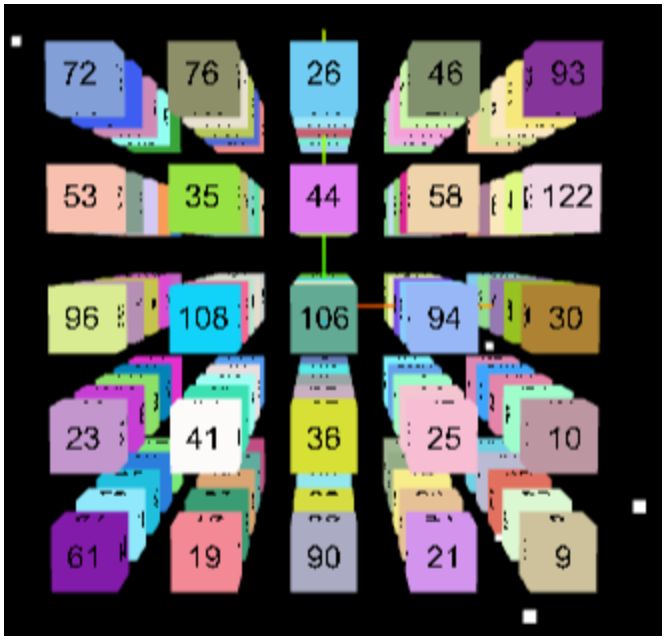


Nilai objective function akhir: 139
Plot:

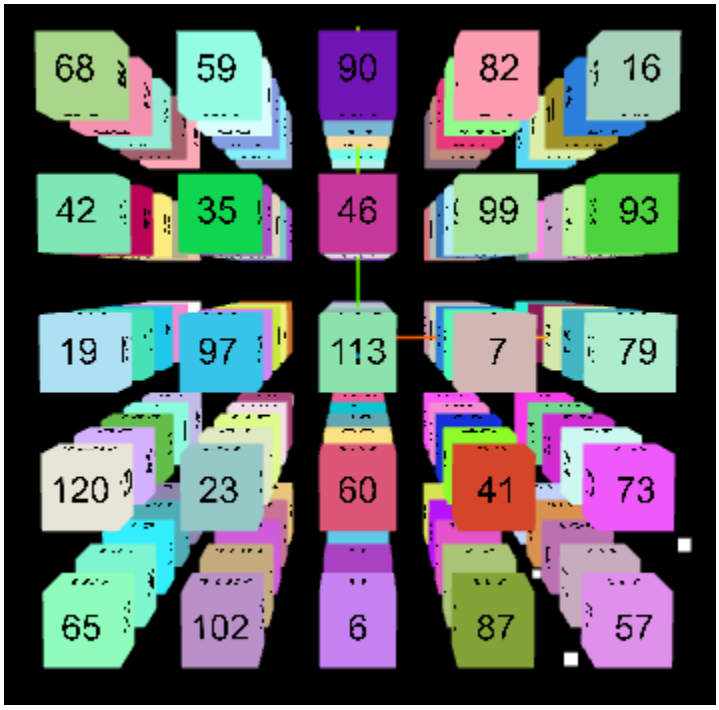


Durasi: 20.817386627197266 detik
Frekuensi stuck: 106046

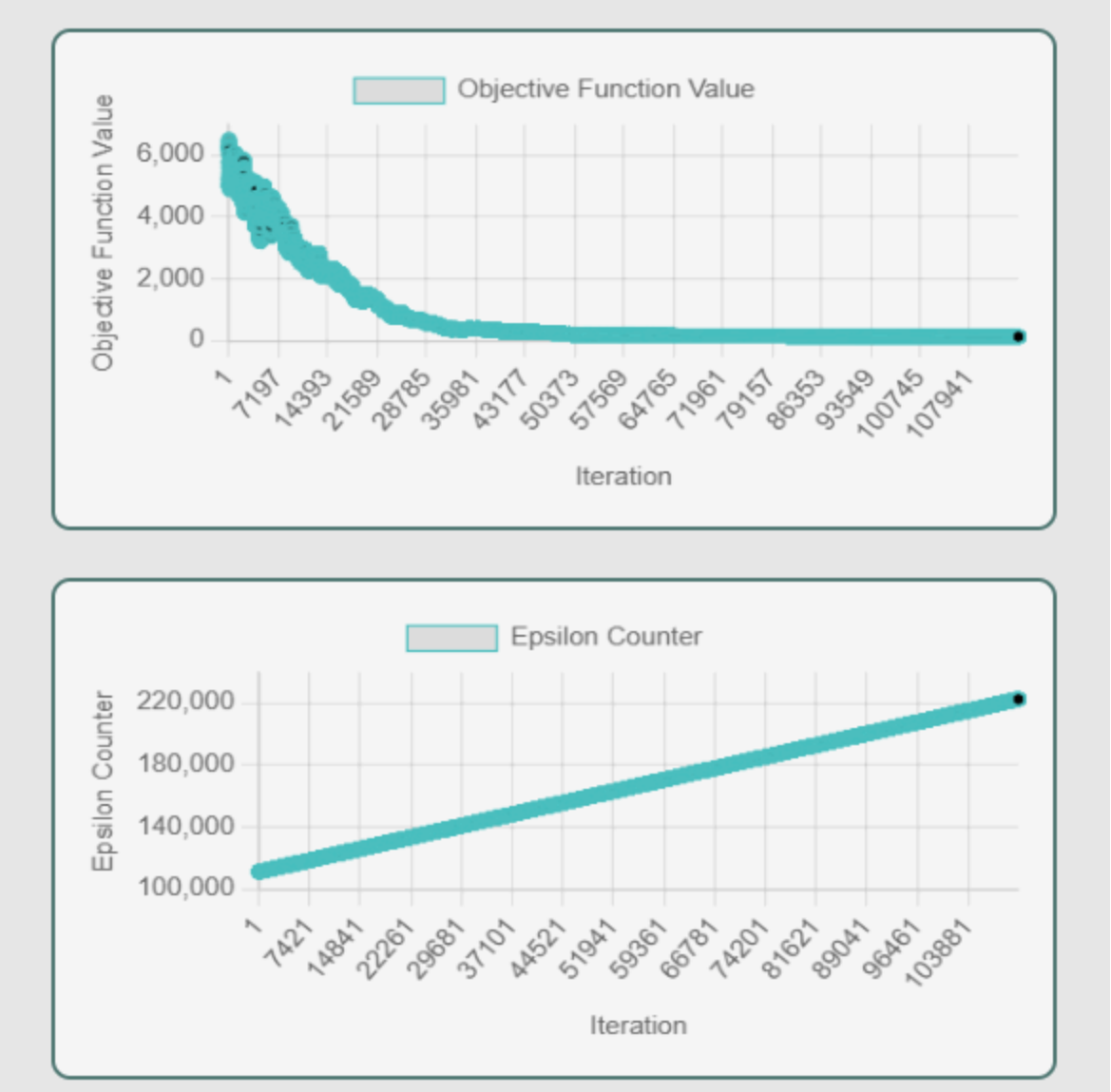
Eksperimen 2:
(awal)



(akhir)

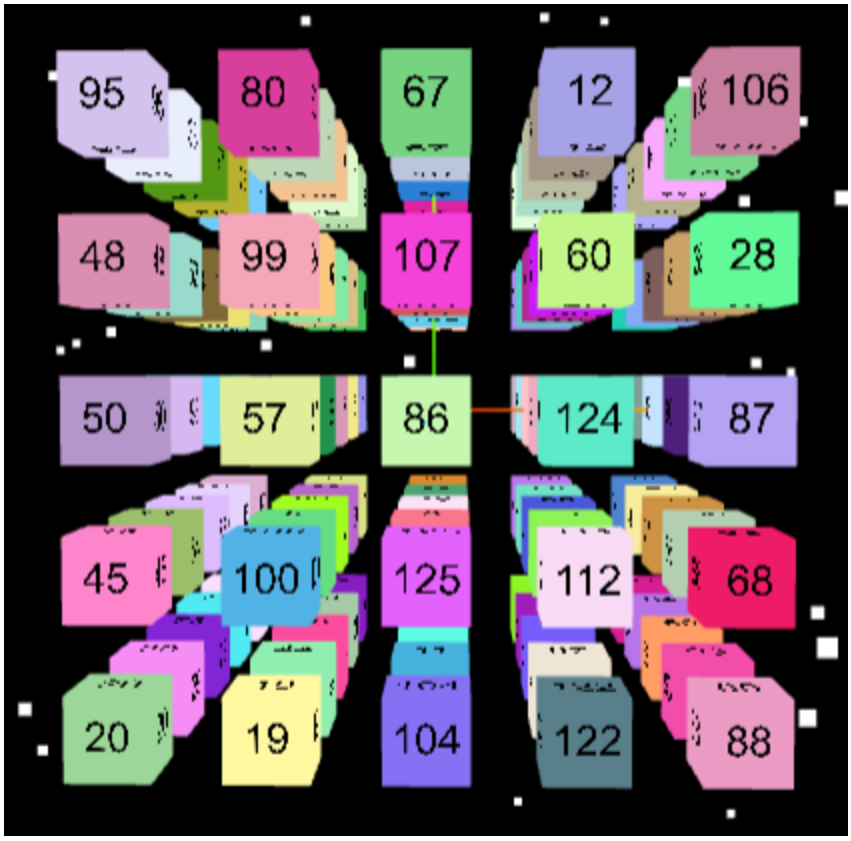


Nilai objective function akhir: 140
Plot:

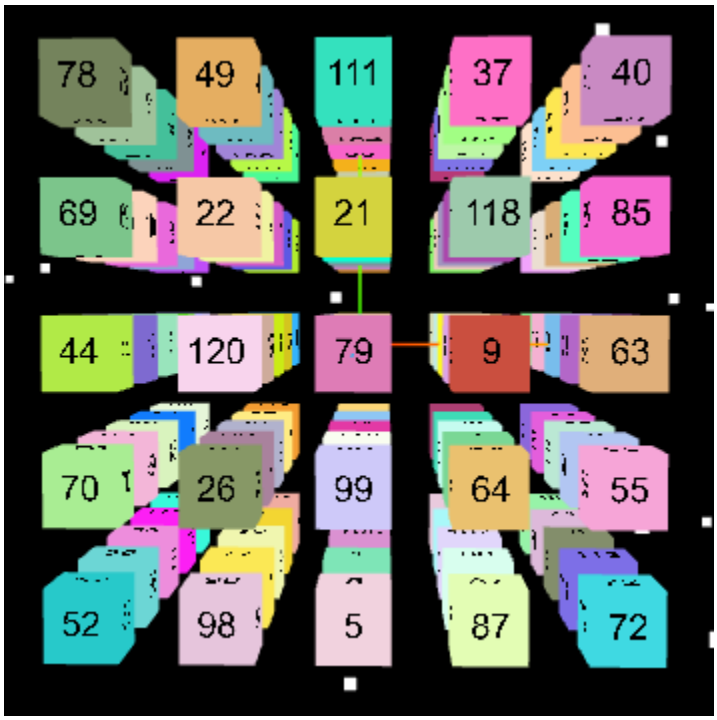


Durasi: 20.929354667663574 detik
Frekuensi stuck: 106007

Eksperimen 3:
(awal)

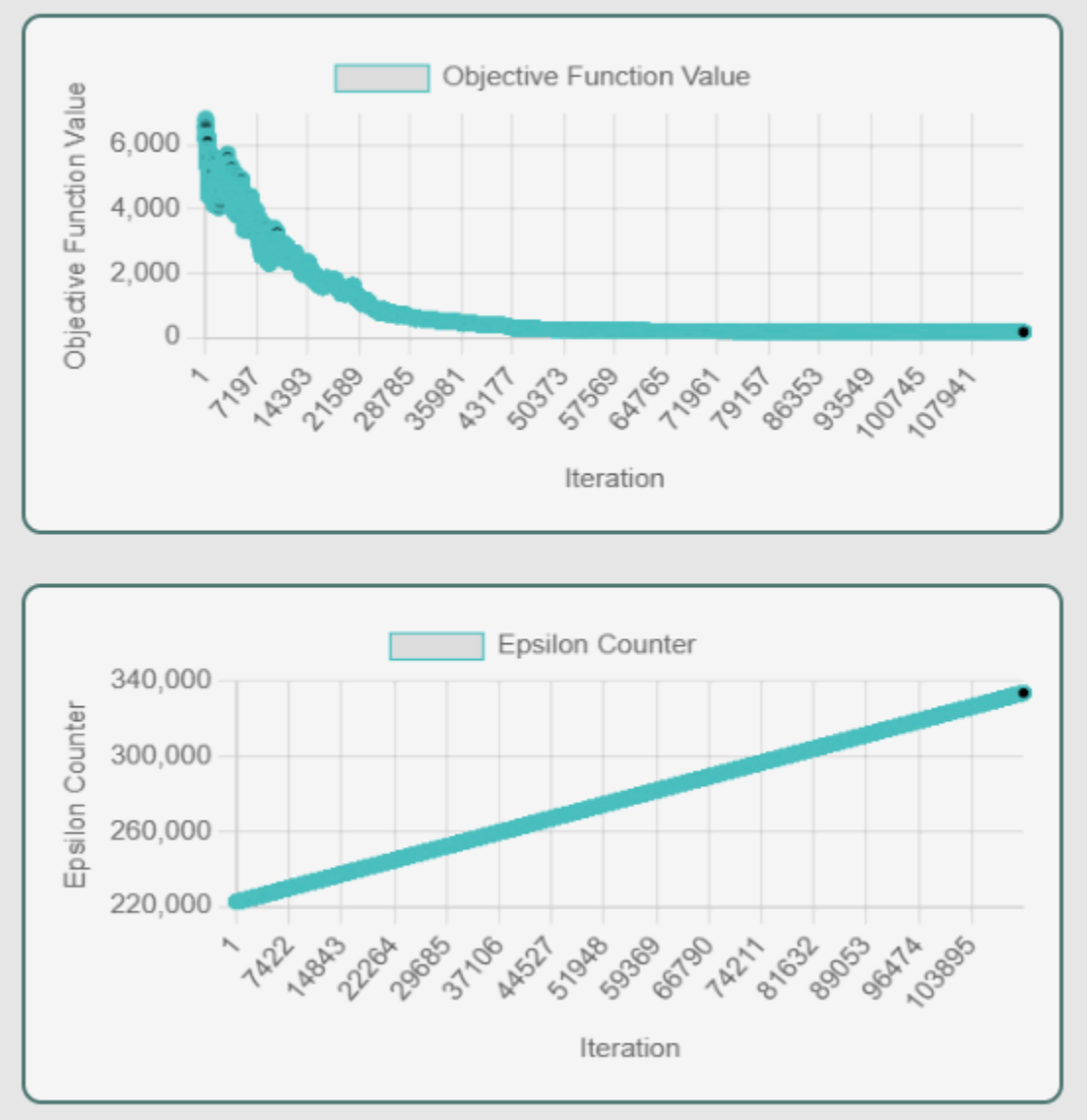


(akhir)



Nilai objective function akhir: 177

Plot:



Durasi: 21.828585147857666 detik
Frekuensi stuck: 106172

- Genetic Algorithm
  Genetic Algorithm (GA) adalah metode optimasi dan pencarian solusi yang terinspirasi oleh proses seleksi alam dalam evolusi biologis. Algoritma ini bekerja dengan cara membentuk populasi awal dari solusi acak, kemudian melalui proses seleksi, crossover (pertukaran gen antar individu), dan mutasi, menghasilkan populasi baru yang semakin mendekati solusi optimal. Setiap individu dalam populasi direpresentasikan sebagai kromosom yang mengkodekan solusi dalam bentuk gen, dan kualitas solusi diukur oleh fungsi kebugaran (fitness function). Individu dengan kebugaran lebih tinggi memiliki peluang lebih besar untuk dipilih sebagai "orang tua" yang akan menghasilkan

"keturunan" di generasi berikutnya. Dengan iterasi bertahap, populasi berevolusi menuju solusi optimal global. Genetic algorithm terkenal karena fleksibilitasnya dan kemampuannya mengeksplorasi ruang solusi yang luas dan kompleks, meskipun memerlukan biaya komputasi tinggi dan penyesuaian parameter yang cermat.

KESIMPULAN DAN SARAN

Dari hasil eksperimen dan analisis yang telah kami lakukan, dapat disimpulkan bahwa

1. Steepest Hill Climbing
    Kelebihan:
    Sederhana dan cepat: Algoritma ini mudah diimplementasikan dan dapat dengan cepat menemukan solusi lokal optimal dalam ruang pencarian yang sederhana.
    Efisiensi komputasi: Karena hanya perlu mengevaluasi solusi tetangga yang lebih baik, algoritma ini cukup ringan dalam penggunaan sumber daya.
    Kekurangan:
    Terjebak di puncak lokal: Steepest hill climbing cenderung terhenti pada solusi optimal lokal, terutama dalam ruang pencarian yang kompleks.
    Tidak memiliki mekanisme eksplorasi: Algoritma ini hanya mengejar solusi yang meningkatkan nilai fungsi tujuan, sehingga sering gagal dalam menemukan solusi optimal global.
    Tidak ideal untuk masalah non-linear kompleks: Untuk masalah dengan banyak puncak atau perbukitan (multi-modal), performa algoritma ini kurang memuaskan.
2. Simulated Annealing
    Kelebihan:
    Dapat keluar dari puncak lokal: Simulated annealing memiliki mekanisme untuk menerima solusi yang lebih buruk secara acak (tergantung pada suhu), sehingga memiliki peluang lebih besar untuk menemukan solusi optimal global.
    Cocok untuk ruang pencarian kompleks: Dengan memanfaatkan suhu yang menurun secara bertahap, algoritma ini dapat menghindari jebakan di solusi lokal.
    Fleksibel: Bisa diterapkan pada berbagai jenis masalah optimasi, termasuk yang bersifat diskrit maupun kontinu.
    Kekurangan:
    Proses konvergensi lambat: Algoritma ini memerlukan banyak iterasi untuk mencapai hasil akhir, terutama jika pengaturan suhu tidak tepat.
    Pemilihan parameter yang rumit: Pengaturan awal, seperti suhu awal, laju pendinginan, dan fungsi jadwal, sangat mempengaruhi performa algoritma.
    Tidak menjamin solusi optimal global: Meskipun peluangnya lebih besar daripada hill climbing, simulated annealing tetap bisa berhenti di solusi suboptimal.
3. Genetic Algorithm
    Kelebihan:
    Eksplorasi luas pada ruang solusi: Dengan menggunakan populasi solusi yang diperbarui setiap generasi, genetic algorithm memiliki kemampuan eksplorasi yang lebih luas dan bisa keluar dari puncak lokal.

Dapat menemukan solusi optimal global: Dengan kombinasi seleksi, mutasi, dan crossover, algoritma ini dapat menemukan solusi optimal global dalam ruang pencarian yang kompleks.
Fleksibilitas tinggi: Genetic algorithm bisa diterapkan pada berbagai jenis masalah tanpa banyak modifikasi, baik itu masalah optimasi linear, non-linear, maupun masalah multi-modal.
Kekurangan:
Biaya komputasi tinggi: Karena melibatkan populasi besar dan iterasi banyak, genetic algorithm memerlukan sumber daya komputasi yang lebih besar.
Parameter tuning yang kompleks: Pengaturan parameter seperti ukuran populasi, tingkat mutasi, dan persentase crossover sangat mempengaruhi hasil dan memerlukan penyesuaian yang cermat.
Konvergensi lambat: Algoritma ini cenderung membutuhkan banyak generasi sebelum mencapai solusi yang stabil atau optimal.
Kesimpulan Umum
Pemilihan algoritma tergantung pada kebutuhan spesifik dan karakteristik masalah yang dihadapi:

Untuk kasus sederhana dan kebutuhan komputasi cepat, steepest hill climbing bisa menjadi pilihan.
Untuk kasus yang membutuhkan fleksibilitas dengan ruang pencarian yang kompleks, simulated annealing dapat membantu menemukan solusi lebih baik.
Pada masalah yang luas dan kompleks, genetic algorithm biasanya menghasilkan hasil terbaik, meskipun membutuhkan sumber daya yang lebih banyak.

PEMBAGIAN TUGAS

- 13522123:
  Visualisasi Three JS, Simulated Annealing, Laporan, Plot Chart Steepest Ascent dan Simulated Annealing

- 13522133:
  Visualisasi kubus, Steepest Ascent Hill Climbing, Laporan

REFERENSI

- Features of the magic cube - Magisch vierkant
- Perfect Magic Cubes (trump.de)