

IF3170 Inteligensi Artifisial

Pencarian Solusi Diagonal Magic Cube dengan Local Search

Tugas Besar 1

Disusun untuk memenuhi tugas mata kuliah Inteligensi Artifisial untuk
Semester 5 tahun ajaran 2024 / 2025



Oleh

Aland Mulia Pratama	13522124
Rizqika Mulia Pratama	13522126
Christian Justin Hendrawan	13522135
Ikhwan Al Hakim	13522147

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

DAFTAR ISI

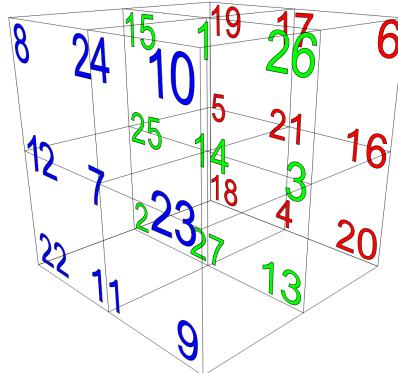
BAB I	
DESKRIPSI MASALAH.....	2
BAB II	
PEMBAHASAN.....	5
3.1. Pemilihan Objective Function.....	5
3.2. Implementasi Algoritma.....	6
2.2.1. Hill Climbing.....	6
2.2.1.1. Steepest Ascent Hill Climbing.....	6
2.2.1.2. Hill Climbing with Sideways Move.....	6
2.2.1.3. Stochastic Hill Climbing.....	6
2.2.1.4. Random Restart Hill Climbing.....	6
2.2.2. Simulated Annealing.....	10
2.2.3. Genetic Algorithm.....	13
BAB III	
HASIL DAN ANALISIS.....	21
3.1. Hasil Pengujian.....	21
3.1.1. Hill Climbing.....	21
3.1.1.1. Steepest Ascent Hill Climbing.....	21
3.1.1.2. Hill Climbing with Sideways Move.....	25
3.1.1.3. Random Restart Hill Climbing.....	29
3.1.1.4. Stochastic Hill Climbing.....	35
3.1.2. Simulated Annealing.....	40
3.1.3. Genetic Algorithm.....	44
3.1.3.1. Jumlah populasi sebagai kontrol dan iterasi yang bervariasi (Populasi = 1000).....	44
3.1.3.2. Jumlah iterasi sebagai kontrol dan populasi yang bervariasi (Iterasi = 150).....	64
3.2. Analisis Pengujian.....	85
3.2.1. Hill Climbing.....	85
3.2.1.1. Steepest Ascent Hill Climbing.....	85
3.2.1.2. Hill Climbing with Sideways Move.....	86
3.2.1.3. Random Restart Hill Climbing.....	86
3.2.1.4. Stochastic Hill Climbing.....	87
3.2.2. Simulated Annealing.....	87
3.2.3. Genetic Algorithm.....	88
BAB IV	
Kesimpulan dan Saran.....	90
5.1. Kesimpulan.....	90
5.2. Saran.....	90
BAB V	
Referensi.....	91
Lampiran.....	92

BAB I

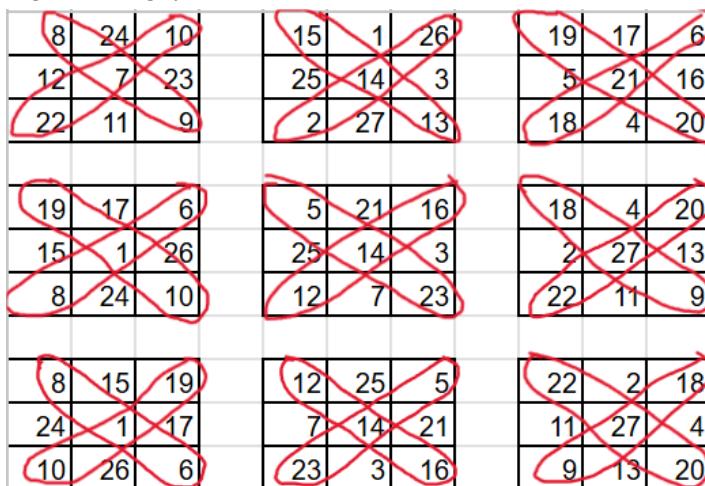
DESKRIPSI MASALAH

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
 - Berikut ilustrasi dari potongan bidang yang ada pada suatu kubus berukuran 3:



- Terdapat 9 potongan bidang, yaitu:



- Diagonal yang dimaksud adalah yang dilingkari warna merah saja

- Ilustrasi dan penjelasan lebih detail bisa anda lihat di link berikut: [Features of the magic cube - Magisch vierkant](#)

Pada tugas ini, peserta kuliah 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 (tetapi hanya menukar posisi 2 angka saja juga diperbolehkan).

Anda akan diminta untuk mengimplementasikan rencana yang telah Anda buat pada Tugas Kecil 1. Berikut merupakan hal-hal yang perlu dilakukan oleh setiap kelompok:

- Implementasikan 3 algoritma local search dengan rincian sebagai berikut:
 - Salah satu algoritma hill-climbing
 - Simulated Annealing
 - Genetic Algorithm
- Lakukan eksperimen dengan skema sebagai berikut:
 - Jalankan setiap algoritma sebanyak **3 kali**, kemudian catat beberapa hal berikut:
 - Berlaku untuk semua algoritma
 - State awal dan akhir dari kubus
 - Nilai objective function akhir yang dicapai
 - Plot nilai objective function terhadap banyak iterasi yang telah dilewati
 - Durasi proses pencarian
 - Berlaku hanya untuk Steepest Ascent Hill-Climbing dan Stochastic Hill-Climbing
 - Banyak iterasi hingga proses pencarian berhenti
 - Berlaku hanya untuk Hill-Climbing with Sideways Move
 - Banyak iterasi hingga proses pencarian berhenti
 - Note: Tambahkan parameter maximum sideways move, dimana ketika banyak sideways move yang dilakukan sudah mencapai maksimum, pencarian dihentikan
 - Berlaku hanya untuk Random Restart Hill-Climbing
 - Banyak restart
 - Banyak iterasi per restart
 - Note: Tambahkan parameter maximum restart, dimana ketika banyak restart sudah mencapai maksimum, pencarian dihentikan.
 - Berlaku hanya untuk Simulated Annealing
 - Plot $e^{\frac{\Delta E}{T}}$ terhadap banyak iterasi yang telah dilewati
 - Frekuensi ‘stuck’ di local optima
 - Khusus untuk Genetic Algorithm, lakukan beberapa hal berikut:
 - Terdapat 2 parameter yang dapat diubah, yaitu **Jumlah populasi** dan **banyak iterasi**.

- Jadikan jumlah populasi sebagai kontrol, kemudian pilih **3 variasi** banyak iterasi yang berbeda. Jalankan program sebanyak masing-masing **3 kali** untuk setiap konfigurasi parameter.
- Jadikan banyak iterasi sebagai kontrol, kemudian pilih **3 variasi** jumlah populasi yang berbeda. Jalankan program sebanyak masing-masing **3 kali** untuk setiap konfigurasi parameter.
- Untuk setiap eksperimen, catat beberapa hal berikut:
 - State awal dan akhir dari kubus
 - Nilai objective function akhir yang dicapai
 - Plot nilai objective function terhadap banyak iterasi yang telah dilewati (Cukup plot nilai objective function maksimum dan rata-rata dari populasi terhadap banyak iterasi yang telah dilewati. Jika sudah terlanjur membuat plot untuk tiap individu pada populasi tidak menjadi masalah, silahkan diberikan keterangan saja maksud plotnya apa)
 - Jumlah populasi
 - Banyak iterasi
 - Durasi proses pencarian
- Lakukan analisis terhadap hasil eksperimen, berikut merupakan beberapa pertanyaan yang dapat menjadi acuan untuk analisis yang Anda lakukan (Anda boleh menambahkan beberapa pertanyaan tambahan jika dirasa perlu untuk dijelaskan):
 - Seberapa dekat tiap-tiap algoritma bisa mendekati global optima dan mengapa hasilnya demikian?
 - Bagaimana perbandingan hasil pencarian tiap-tiap algoritma dengan algoritma local search yang lain?
 - Bagaimana perbandingan durasi proses pencarian tiap algoritma relatif terhadap algoritma lainnya?
 - Seberapa konsisten hasil akhir yang didapatkan dari tiap-tiap eksperimen yang dilakukan?
 - Bagaimana pengaruh banyak iterasi dan jumlah populasi terhadap hasil akhir pencarian pada Genetic Algorithm?
 - dst...
- Program yang dibuat harus bisa **memvisualisasikan** state awal kubus, state akhir kubus, dan juga hasil eksperimennya (sesuaikan informasi hasil eksperimen yang ditampilkan dengan ketentuan yang telah dijelaskan di poin sebelum ini).
- Cara visualisasi dibebaskan kepada kelompok masing-masing selama seluruh angka pada kubus dan juga hasil eksperimen terlihat dengan jelas.
- Dibebaskan menggunakan bahasa pemrograman apapun.
- Diperbolehkan untuk menggunakan heuristik yang Anda buat sendiri atau dari referensi lain untuk optimasi pencarian solusi, asalkan masih dalam lingkup local search. Jangan lupa jelaskan heuristik yang Anda pakai di laporan.

BAB II

PEMBAHASAN

3.1. Pemilihan Objective Function

Objective function pada umumnya adalah sebuah fungsi yang kita definisikan untuk mengukur seberapa “baik” atau “buruk” suatu solusi tertentu. Dalam kasus Diagonal Magic Cube, fungsi ini akan mengukur seberapa dekat konfigurasi dari *state* kubus saat ini dengan Diagonal Magic Cube yang sempurna. Objective Function dalam optimasi Diagonal Magic Cube memiliki tujuan utama untuk meminimalkan perbedaan dari konfigurasi ideal. Fungsi ini mengukur total perbedaan antara jumlah setiap baris, kolom, dan diagonal dengan Magic Number yang diharapkan. Semakin mendekati nol nilai Objective Function yang dihasilkan dari suatu konfigurasi, semakin baik konfigurasi tersebut, yang berarti semakin dekat dengan konfigurasi Diagonal Magic Cube yang sempurna. Berikut adalah representasi matematis dari objective function tersebut:

$$f(x) = - \sum_{z=0}^4 \sum_{x=0}^4 \left| 315 - \sum_{y=0}^4 \text{cube}[x][y][z] \right| - \sum_{z=0}^4 \sum_{y=0}^4 \left| 315 - \sum_{x=0}^4 \text{cube}[x][y][z] \right| - \sum_{y=0}^4 \sum_{x=0}^4 \left| 315 - \sum_{z=0}^4 \text{cube}[x][y][z] \right| \\ - \sum_{y=0}^4 \left| 315 - \sum_{i=0}^4 \text{cube}[i][y][i] \right| - \sum_{y=0}^4 \left| 315 - \sum_{i=0}^4 \text{cube}[4-i][y][i] \right| \\ - \sum_{x=0}^4 \left| 315 - \sum_{i=0}^4 \text{cube}[x][i][i] \right| - \sum_{x=0}^4 \left| 315 - \sum_{i=0}^4 \text{cube}[x][4-i][i] \right| \\ - \sum_{z=0}^4 \left| 315 - \sum_{i=0}^4 \text{cube}[i][i][z] \right| - \sum_{z=0}^4 \left| 315 - \sum_{i=0}^4 \text{cube}[4-i][i][z] \right| \\ - \left| 315 - \sum_{i=0}^4 \text{cube}[i][i][i] \right| - \left| 315 - \sum_{i=0}^4 \text{cube}[4-i][i][i] \right| - \left| 315 - \sum_{i=0}^4 \text{cube}[4-i][i][4-i] \right| - \left| 315 - \sum_{i=0}^4 \text{cube}[i][i][4-i] \right|$$

Perhitungan ini terbagi menjadi tiga bagian iterasi:

- Iterasi bagian pertama, yaitu baris pertama dari representasi matematis diatas. Bagian ini menghitung selisih penjumlahan baris, kolom, dan pilar terhadap magic number dari kubus.
- Iterasi bagian kedua, yaitu baris kedua hingga keempat dari representasi matematis diatas. Bagian ini menghitung selisih penjumlahan setiap diagonal sisi dari kubus terhadap magic number dari kubus.
- Iterasi bagian ketiga, yaitu baris kelima dari representasi matematis diatas. Bagian ini menghitung selisih semua diagonal ruang terhadap magic number dari kubus.

Dari perhitungan diatas, didapatkan bahwa sebuah kubus akan mencapai global optimum ketika memiliki objective function dengan nilai 0, yaitu ketika selisih setiap baris, kolom, pilar, dan semua diagonal dengan magic number nya bernilai 0, atau bernilai sama dengan magic number dari kubus.

3.2. Implementasi Algoritma

2.2.1. Hill Climbing

Algoritma *hill climbing* adalah metode pencarian heuristik di mana algoritma ini berusaha untuk menemukan solusi terbaik dengan melakukan perbaikan kecil terhadap state saat ini, bisa dengan mengambil random successor, atau bisa juga dengan mengambil successor terbaik. Algoritma ini bekerja dengan memulai dari solusi acak dan kemudian bergerak menuju solusi yang memiliki nilai objective function yang lebih baik, layaknya mendaki bukit (*hill climbing*). Setiap langkahnya, algoritma hanya mempertimbangkan pergerakan yang meningkatkan nilai solusi, sehingga cenderung cepat menemukan solusi lokal yang optimal. Namun, metode ini memiliki kelemahan karena mudah terjebak di puncak lokal dan mungkin tidak selalu mencapai solusi global terbaik tanpa teknik tambahan seperti *random restart* atau *simulated annealing* untuk mengatasi jebakan pada puncak lokal tersebut.

Hill climbing terdiri dari empat jenis utama, masing-masing dengan pendekatan berbeda untuk mencapai solusi optimal. Jenis tersebut adalah sebagai berikut:

2.2.1.1. Steepest Ascent Hill Climbing

Dalam metode ini, algoritma memilih langkah yang memberikan peningkatan terbesar pada nilai fungsi tujuan dalam setiap iterasi. Ini bertujuan untuk mempercepat pencarian solusi dengan mengutamakan pergerakan yang paling meningkatkan kualitas solusi. Namun, metode ini juga berisiko tinggi terjebak dalam puncak lokal.

2.2.1.2. Hill Climbing with Sideways Move

Teknik ini memungkinkan algoritma untuk tetap bergerak meskipun tidak ada peningkatan nilai, tetapi juga tidak mengalami penurunan, sehingga tetap di nilai yang sama. Sideways move digunakan untuk menghindari jebakan pada *plateau*, yaitu area solusi dengan nilai yang seragam, yang dapat menyebabkan algoritma berhenti tanpa menemukan solusi lebih baik.

2.2.1.3. Stochastic Hill Climbing

Dalam pendekatan ini, algoritma memilih langkah secara acak dari sekumpulan langkah yang meningkatkan nilai, bukan berdasarkan peningkatan tertinggi. Ini memberikan algoritma fleksibilitas lebih besar untuk mengeksplorasi berbagai arah dan berpotensi membantu menghindari puncak lokal dengan lebih baik.

2.2.1.4. Random Restart Hill Climbing

Pada jenis ini, algoritma akan mengulangi proses pencarian dari solusi awal yang berbeda setiap kali terjebak pada puncak lokal. Dengan mengulangi pencarian beberapa kali, random restart meningkatkan peluang menemukan solusi global yang optimal, karena algoritma dapat "memulai ulang" pencarian dari berbagai titik dalam ruang solusi.

Berikut adalah implementasi keempat algoritma hill climbing dalam bentuk pseudocode:

```
Function SteepestAscentHillClimbing(cube: array of array of
array of integer)
    → array of array of array of integer, integer, integer,
array of integer
```

KAMUS LOKAL

```

iterOF: array of integer
currentState: array of array of array of integer
currentValue: integer
newState: array of array of array of integer
newValue: integer

ALGORITMA
iterOF ← []
currentState ← cube
currentValue ← calculateObjectiveFunction(currentState)

WHILE currentValue < 0 DO
    newState ← copy3DArray(cube)
    newState ← generateMaximumSuccessor(newState)
    newValue ← calculateObjectiveFunction(newState)

    IF newValue ≤ currentValue THEN
        printCube(cube)
        PRINT "fail"
        BREAK

    cube ← newState
    currentValue ← newValue
    PRINT currentValue
    iterOF.APPEND(currentValue)

IF currentValue = 0 THEN
    printCube(cube)
    PRINT "success"

→ cube, currentValue, 0, iterOF

```

Function SidewaysMoveHillClimbing(cube: array of array of array of integer)
 → array of array of array of integer, integer, integer, array of integer

KAMUS LOKAL

```

iterOF: array of integer
currentState: array of array of array of integer
currentValue: integer
sameCounter: integer
newState: array of array of array of integer
newValue: integer

```

ALGORITMA

```

iterOF ← []
currentState ← cube
currentValue ← calculateObjectiveFunction(currentState)

```

```

WHILE currentValue < 0 DO
    sameCounter ← 0
    newState ← copy3DArray(cube)
    newState ← generateMaximumSuccessor(newState)
    newValue ← calculateObjectiveFunction(newState)

    IF newValue = currentValue THEN
        sameCounter ← sameCounter + 1
    IF newValue > currentValue THEN
        sameCounter ← 0

    IF newValue < currentValue OR sameCounter = 1 THEN
        printCube(cube)
        PRINT "fail"
        BREAK

    cube ← newState
    currentValue ← newValue
    PRINT currentValue
    iterOF.APPEND(currentValue)

    IF currentValue = 0 THEN
        printCube(cube)
        PRINT "success"

→ cube, currentValue, 0, iterOF

```

Function StochasticHillClimbing(cube: array of array of array of integer, NMax: integer)
→ array of array of array of integer, integer, integer, array of integer

KAMUS LOKAL

```

iterOF: array of integer
currentState: array of array of array of integer
currentValue: integer
proceed: boolean
i: integer
newState: array of array of array of integer
newValue: integer

```

ALGORITMA

```

iterOF ← []
currentState ← cube
currentValue ← calculateObjectiveFunction(currentState)
proceed ← true

WHILE currentValue < 0 AND proceed DO
    FOR i ← 0 TO NMax - 1 DO
        IF i MOD 100000 = 0 AND i ≠ 0 THEN

```

```

        PRINT "Iterasi ke", i

        newState ← copy3DArray(currentState)
        newState ← swapRandom(currentState)
        newValue ← calculateObjectiveFunction(newState)

        IF newValue - currentValue > 0 THEN
            currentState ← copy3DArray(newState)
            currentValue ← newValue
            BREAK

        PRINT currentValue
        iterOF.APPEND(currentValue)

        IF i = NMax THEN
            proceed ← false
            printCube(cube)
            PRINT "fail"

        IF currentValue = 0 THEN
            printCube(cube)
            PRINT "success"

→ currentState, currentValue, 0, iterOF

```

Function RandomRestartHillClimbing(cube: array of array of array of integer, p: real, n: integer)
→ array of array of array of integer, integer, integer, array of integer

KAMUS LOKAL

```

currentValue: integer
iterOF: array of integer
temp: integer
param: real
i: integer

```

ALGORITMA

```

iterOF ← []
temp ← 0
PRINT p
i ← 0

REPEAT
    cube, currentValue, temp, iterOF ←
SteepestAscentHillClimbing(cube)
    param ← random()

    IF param ≥ p OR i = n - 1 THEN
        PRINT "not restarting"

```

```

        BREAK

        i ← i + 1
        PRINT "restarting"
        PRINT param
        cube ← generateRandom5x5x5Array()

→ cube, currentValue, temp, iterOF

```

2.2.2. Simulated Annealing

Simulated Annealing adalah algoritma local search dalam kecerdasan buatan yang terinspirasi dari proses pendinginan logam dalam ilmu metalurgi. Dalam proses tersebut, logam dipanaskan hingga mencapai suhu tertentu yang sangat tinggi, untuk kemudian didinginkan secara perlahan-lahan. Pada algoritma simulated annealing, prinsip ini digunakan untuk menemukan solusi optimal dari suatu masalah dengan cara mengeksplorasi berbagai solusi secara acak sambil menurunkan kemungkinan menerima neighbor state yang memiliki value lebih jelek seiring waktu.

Pada dasarnya, simulated annealing bekerja dengan:

1. Memulai sebuah state awal secara acak.
2. Mengevaluasi state tersebut menggunakan fungsi objektif yang mengukur kualitas neighbor state.
3. Menghasilkan neighbor state secara acak dari state saat ini dan mengevaluasi kualitasnya.
4. Memutuskan apakah akan menerima neighbor state ini atau tidak berdasarkan seberapa baik kualitasnya dan suhu sistem saat ini menggunakan rumus probabilitas.

Neighbor state yang lebih baik dari state saat ini selalu diterima, sedangkan neighbor state yang lebih buruk dari state saat ini dipertimbangkan untuk diterima berdasarkan suatu probabilitas yang dihitung dengan persamaan Boltzmann:

$$probabilitas = e^{\Delta/T}$$

Di mana Δ adalah perubahan nilai fungsi objektif antara solusi tetangga dan state saat ini, dan suhu mengontrol kemungkinan penerimaan neighbor state yang kurang baik. Semakin tinggi suhunya, semakin besar kemungkinan untuk menerima neighbor state yang lebih buruk untuk meningkatkan eksplorasi state solusi.

Berikut adalah langkah-langkah algoritma simulated annealing yang diterapkan dalam konteks masalah magic cube, di mana kita ingin menemukan konfigurasi kubus yang mencapai nilai objektif dari state global optimum.

1. Inisiasi
 - Mulai dengan konfigurasi awal dari cube yang mungkin dihasilkan secara acak.

- Terapkan suhu awal yang tinggi (dalam kasus ini digunakan 10000) dan tingkat penurunan suhu (dalam kasus ini 0.995), yang menentukan seberapa cepat suhu sistem menurun seiring waktu.
- Tetapkan kondisi berhenti jika nilai objektif target tercapai atau suhu sudah mendekati nol.

2. Definisikan Fungsi Objektif

- Fungsi objektif mengevaluasi kualitas atau nilai dari konfigurasi cube saat ini, yang mengukur seberapa dekat nilai state saat ini dengan state solusi optimum global. Semakin dekat nilai ini dengan nilai optimum global, semakin baik solusi tersebut.

3. Membangkitkan *Neighbor State*

- Pada setiap iterasi, dilakukan pembangkitan neighbor state secara acak berdasarkan state saat ini, yaitu menukar dua buah block dalam cube secara acak. Tujuannya adalah untuk menjelajahi konfigurasi yang mungkin lebih optimal.

4. Evaluasi dan Penerimaan *Neighbor State*

- Hitung fungsi objektif untuk neighbor state.
- Hitung perubahan nilai fungsi objektif (delta) antara state value tetangga dengan state value saat ini.
- Jika delta positif (value neighbor state lebih baik), langsung terima neighbor state sebagai state saat ini.
- Jika delta negatif (value neighbor state lebih buruk), terima neighbor state berdasarkan probabilitas penerimaan yang dihitung menggunakan persamaan Boltzmann. Langkah ini memungkinkan algoritma untuk menerima neighbor state yang lebih buruk, sehingga mampu menghindari optimum lokal.

5. *Cooling*

- Setelah setiap iterasi, suhu sistem diturunkan dengan mengalikan suhu dengan tingkat cooling rate yang dimasukkan user. *Cooling* ini mengurangi kemungkinan menerima *neighbor state* yang lebih buruk seiring berjalananya waktu, sehingga pencarian solusi lebih difokuskan pada solusi terbaik.

6. Terminating

- Proses ini berlanjut hingga suhu mendekati nol atau fungsi objektif target tercapai. Jika nilai target sudah ditemukan sebelum suhu habis, algoritma dapat dihentikan lebih awal.

7. Mengembalikan *State* Terbaik

- Setelah kondisi berhenti, algoritma mengembalikan *state* terbaik yang ditemukan selama pencarian. Diharapkan konfigurasi ini sudah mendekati atau mencapai nilai objektif target.

Untuk mengimplementasikan langkah-langkah algoritma simulated annealing, seluruh langkah di atas diimplementasikan ke dalam program dengan bahasa pemrograman go dengan rincian berikut:

```
Function SimulatedAnnealing(cube: array of array of
```

```
integer, initialTemp: real, coolingRate: real, maxIterations:  
integer)  
→ array of array of array of integer, integer, integer,  
array of integer
```

KAMUS LOKAL

```
currentState: array of array of array of integer  
currentValue: integer  
temperature: real  
iter: integer  
stuckCount: integer  
delta: integer  
probability: real  
newState: array of array of array of integer  
newValue: integer  
iterOF: array of integer
```

ALGORITMA

```
currentState ← cube  
currentValue ← calculateObjectiveFunction(currentState)  
temperature ← initialTemp  
iter ← 0  
stuckCount ← 0  
iterOF ← []  
  
FOR i ← 0 TO maxIterations - 1 DO  
    if currentValue ≥ 0 THEN  
        BREAK  
  
        newState ← copy3DArray(currentState)  
        newState ← swapRandom(newState)  
        newValue ← calculateObjectiveFunction(newState)  
  
        delta ← newValue - currentValue  
        probability ← exp(delta / temperature)  
  
        IF delta > 0 OR random() < probability THEN  
            currentState ← copy3DArray(newState)  
            currentValue ← newValue  
        ELSE  
            stuckCount ← stuckCount + 1  
  
        temperature ← temperature * coolingRate  
        PRINT "Iteration:", i, "Current value:", currentValue,  
        "Temperature:", temperature, "Stuck Count:", stuckCount  
  
        IF temperature < 1e-1000 OR currentValue = 0 THEN  
            BREAK
```

```

        iter <- iter + 1
        iterOF.APPEND(currentValue)

        PRINT "Final value:", currentValue
        PRINT "Final solution:"
        printCube(currentState)

        IF currentValue = 0 THEN
            PRINT "success"
        ELSE
            PRINT "fail"

    → currentState, currentValue, stuckCount, iterOF

```

Dengan cara ini, simulated annealing menggunakan proses yang mirip dengan pendinginan fisik untuk menemukan solusi optimal bagi masalah magic cube, menjaga keseimbangan antara eksplorasi solusi baru dan konvergensi ke solusi terbaik.

2.2.3. Genetic Algorithm

Algoritma genetika adalah metode pencarian dan optimasi yang terinspirasi oleh proses seleksi alam dalam teori evolusi. Algoritma ini menggunakan mekanisme yang mirip dengan proses biologis seperti seleksi, crossover (rekombinasi), dan mutasi untuk menemukan solusi optimal atau mendekati optimal dalam ruang pencarian yang besar. Pendekatan Genetic Algorithm untuk mencari solusi Diagonal Magic Cube secara umum dimulai dengan inisialisasi populasi yang terdiri dari beberapa solusi acak, di mana setiap solusi adalah representasi individu dari konfigurasi Cube. Setelah populasi awal terbentuk, algoritma mengevaluasi setiap individu menggunakan fungsi objektif.

Selanjutnya, algoritma menerapkan seleksi, di mana individu dengan fitness terbaik (solusi yang mendekati Magic Cube ideal) dipilih untuk reproduksi. Reproduksi dilakukan melalui mekanisme crossover, di mana dua individu (parent) dipilih dan digabungkan untuk menghasilkan individu baru (offspring), yang mewarisi sebagian dari gen (angka-angka dalam kubus) dari kedua parent. Crossover ini memungkinkan eksplorasi kombinasi baru dalam solusi. Selain itu, mutasi diterapkan untuk memperkenalkan variasi acak dalam offspring. Setelah crossover dan mutasi, populasi baru terbentuk, dan proses evaluasi, seleksi, crossover, dan mutasi diulang hingga mencapai kondisi penghentian, seperti jumlah iterasi atau tercapainya solusi optimal.

Langkah-langkah implementasi Genetic Algorithm untuk mencari solusi Diagonal Magic Cube (deskripsi fungsi/kelas beserta source codenya) :

1. Inisialisasi Populasi

Fungsi : initializePopulation

Fungsi ini bertujuan untuk membuat populasi awal yang terdiri dari sejumlah individu. Setiap individu dalam populasi mewakili solusi potensial untuk masalah Diagonal Magic

Cube. Populasi awal diinisialisasi dengan satu individu yang merupakan kubus awal yang diberikan, dan sisanya dihasilkan secara acak.

Parameter:

- cube *[][][]int: Pointer ke array 3D yang mewakili kubus awal atau kubus input
- size int: Ukuran populasi yang diinginkan.

Mengembalikan:

- [][][]int: Populasi awal yang terdiri dari sejumlah individu.

Source code dalam Pseudocode:

```
Function initializePopulation(cube: array of array of array of integer, size: integer)
    → array of array of array of integer

KAMUS LOKAL
population: array of array of array of array of integer
i: integer

ALGORITMA
population ← array of array of array of array of integer(size)
population[0] ← cube

FOR i ← 1 TO size - 1 DO
    population[i] ← generateRandom5x5x5Array()

→ population
```

2. Seleksi Orang Tua (Parent Selection)

Fungsi : selectParent

Fungsi ini bertujuan untuk memilih individu-individu terbaik sebagai orang tua (parent) berdasarkan nilai fitness menggunakan metode seleksi turnamen. Dalam seleksi turnamen, sejumlah individu dipilih secara acak dari populasi, dan individu dengan nilai fitness terbaik dipilih sebagai orang tua. Metode ini membantu memastikan bahwa individu dengan nilai fitness yang lebih tinggi memiliki peluang lebih besar untuk dipilih, namun tetap memberikan kesempatan bagi individu lain untuk dipilih, menjaga keragaman dalam populasi.

Parameter:

- population [][][]int: Populasi saat ini yang terdiri dari sejumlah individu.

Mengembalikan:

- [][]int: Individu terbaik yang dipilih sebagai orang tua.

Source code dalam Pseudocode:

```
Function selectParent(population: array of array of array of array of integer)
```

```

→ array of array of array of integer

KAMUS LOKAL
tournamentSize: integer
best: array of array of array of integer
bestFitness: integer
individual: array of array of array of integer
individualFitness: integer
i: integer

ALGORITMA
tournamentSize ← 20
best ← population[random(0, len(population) - 1)]
bestFitness ← calculateObjectiveFunction(best)

FOR i ← 0 TO tournamentSize - 1 DO
    individual ← population[random(0, len(population) - 1)]
    individualFitness ←
calculateObjectiveFunction(individual)
    IF individualFitness > bestFitness THEN
        best ← individual
        bestFitness ← individualFitness

→ best

```

3. Crossover (Rekombinasi)

Fungsi : crossover

Fungsi crossover bertujuan untuk menghasilkan dua keturunan (offspring) dari dua individu orang tua (parent) dengan cara melakukan pertukaran sebagian elemen kubus antara kedua orang tua. Proses ini dikenal sebagai rekombinasi atau crossover. Crossover dilakukan pada titik tertentu dalam kubus, dan bagian kubus setelah titik tersebut dipertukarkan antara dua orang tua. Proses ini menciptakan variasi dalam populasi dan memungkinkan kombinasi konfigurasi kubus yang berbeda untuk dieksplorasi, yang dapat meningkatkan peluang menemukan solusi optimal.

Parameter:

- parent1 [][][]int: Individu pertama yang dipilih sebagai orang tua.
- parent2 [][][]int: Individu kedua yang dipilih sebagai orang tua.

Mengembalikan:

- [][][]int: Keturunan pertama yang dihasilkan dari crossover.
- [][][]int: Keturunan kedua yang dihasilkan dari crossover.

Source code dalam Pseudocode:

```
Function crossover(parent1: array of array of array of integer,
parent2: array of array of array of integer)
```

```
→ array of array of array of integer, array of array of  
array of integer
```

KAMUS LOKAL

```
child1: array of array of array of integer  
child2: array of array of array of integer  
x: integer  
i, j, k: integer
```

ALGORITMA

```
child1 ← copy3DArray(parent1)  
child2 ← copy3DArray(parent2)  
  
x ← 3  
  
FOR i ← 0 TO len(parent1) - 1 DO  
    FOR j ← 0 TO len(parent1[i]) - 1 DO  
        FOR k ← 0 TO len(parent1[i][j]) - 1 DO  
            IF i ≤ x THEN  
                child1[i][j][k] ← parent1[i][j][k]  
                child2[i][j][k] ← parent2[i][j][k]  
            ELSE  
                child1[i][j][k] ← parent2[i][j][k]  
                child2[i][j][k] ← parent1[i][j][k]  
  
→ child1, child2
```

4. Mutasi

Fungsi : mutate

Fungsi ini bertujuan untuk melakukan mutasi pada individu dengan perubahan acak pada elemen-elemen kubus. Mutasi membantu menjaga keragaman agar memenuhi *constraint uniqueness* dalam populasi dan mencegah algoritma terjebak pada solusi lokal. Proses mutasi mencakup identifikasi dan penggantian nilai duplikat, serta pertukaran baris dan kolom dengan probabilitas tertentu.

Langkah-langkah:

- Identifikasi Nilai Duplikat: Cari dan identifikasi nilai-nilai duplikat dalam individu dengan cara menyimpannya dalam map.
- Ganti Nilai Duplikat: Ganti nilai-nilai duplikat dengan nilai acak baru dari 1-125 yang belum muncul dalam konfigurasi kubus.
- Pertukaran Baris: Lakukan pertukaran dua elemen acak dalam kubus dengan probabilitas 7%.

- Pertukaran Kolom: Lakukan pertukaran dua kolom acak dalam kubus dengan probabilitas 3%.

Parameter:

- individual [][]int: Individu yang akan mengalami mutasi.

Mengembalikan:

- [][]int: Individu yang telah mengalami mutasi.

Source code dalam Pseudocode:

```
Function mutate(individual: array of array of array of integer)
→ array of array of integer

KAMUS LOKAL
usedNumbers: map of integer to boolean
duplicates: array of array of integer
newNum: integer
i1, j1, k1, i2, j2, k2: integer
col1, col2: integer
i, j, k: integer

ALGORITMA
usedNumbers ← map of integer to boolean
duplicates ← array of array of integer

FOR i ← 0 TO len(individual) - 1 DO
    FOR j ← 0 TO len(individual[i]) - 1 DO
        FOR k ← 0 TO len(individual[i][j]) - 1 DO
            num ← individual[i][j][k]
            IF usedNumbers[num] THEN
                duplicates ← append(duplicates, [i, j, k])
            ELSE
                usedNumbers[num] ← true

FOR each pos IN duplicates DO
    newNum ← random(1, 125)
    WHILE usedNumbers[newNum] DO
        newNum ← random(1, 125)
    individual[pos[0]][pos[1]][pos[2]] ← newNum
    usedNumbers[newNum] ← true

    IF random() < 0.07 THEN
        i1, j1, k1 ← random(0, len(individual) - 1), random(0,
len(individual[0]) - 1), random(0, len(individual[0][0]) - 1)
        i2, j2, k2 ← random(0, len(individual) - 1), random(0,
len(individual[0]) - 1), random(0, len(individual[0][0]) - 1)
        individual[i1][j1][k1], individual[i2][j2][k2] ←
individual[i2][j2][k2], individual[i1][j1][k1]
```

```

IF random() < 0.03 THEN
    col1 ← random(0, len(individual[0]) - 1)
    col2 ← random(0, len(individual[0]) - 1)
    FOR i ← 0 TO len(individual) - 1 DO
        FOR j ← 0 TO len(individual[i]) - 1 DO
            individual[i][j][col1], individual[i][j][col2]
← individual[i][j][col2], individual[i][j][col1]

→ individual

```

5. Evolusi Populasi

Fungsi : evolvePopulation

Fungsi ini bertujuan untuk membentuk populasi baru dari keturunan yang dihasilkan melalui proses seleksi, crossover, dan mutasi. Populasi baru dibentuk dengan memilih orang tua, melakukan crossover, dan mutasi pada keturunan yang dihasilkan. Proses ini diulang hingga populasi baru terbentuk sepenuhnya.

Parameter:

- population [][][]int: Populasi saat ini yang terdiri dari sejumlah individu.

Mengembalikan:

- [][][]int: Populasi baru yang terdiri dari keturunan yang dihasilkan.

Source code dalam Pseudocode:

```

Function evolvePopulation(population: array of array of array
of array of integer)
    → array of array of array of integer

```

KAMUS LOKAL

```

newPopulation: array of array of array of array of integer
parent1, parent2: array of array of array of integer
child1, child2: array of array of array of integer

```

ALGORITMA

```

newPopulation ← array of array of array of integer

WHILE len(newPopulation) < len(population) DO
    parent1 ← SelectParent(population)
    parent2 ← SelectParent(population)
    child1, child2 ← Crossover(parent1, parent2)
    newPopulation ← append(newPopulation, Mutate(child1))
    IF len(newPopulation) < len(population) THEN
        newPopulation ← append(newPopulation,
Mutate(child2))

→ newPopulation

```

6. Algoritma Genetika

Fungsi : geneticAlgorithm

Fungsi ini menggabungkan semua langkah sebelumnya untuk mencari solusi Diagonal Magic Cube. Algoritma ini melibatkan inisialisasi populasi, seleksi orang tua, crossover, mutasi, dan evolusi populasi untuk menemukan solusi optimal atau mendekati optimal. Proses ini diulang hingga kondisi penghentian tercapai, seperti jumlah generasi maksimum atau nilai fitness yang memadai.

Parameter:

- cube *[][][]int: Pointer ke array 3D yang mewakili kubus awal.
- populationSize int: Ukuran populasi yang diinginkan.
- maxGenerations int: Jumlah generasi maksimum yang diizinkan.

Mengembalikan:

- [][][]int: Solusi terbaik yang ditemukan.
- int: Nilai fitness terbaik yang ditemukan.
- []int: Daftar nilai fitness terbaik untuk setiap generasi.

Source code dalam Pseudocode:

```
Function geneticAlgorithm(cube: array of array of array of integer, populationSize: integer, maxGenerations: integer)
    → array of array of array of integer, integer, array of integer

KAMUS LOKAL
population: array of array of array of array of integer
generation: integer
iterOF: array of integer
bestState: array of array of array of integer
bestFitness: integer
individual: array of array of array of integer
fitness: integer

ALGORITMA
population ← InitializePopulation(cube, populationSize)
generation ← 0
iterOF ← []
bestFitness ← -50000

FOR generation ← 0 TO maxGenerations - 1 DO
    population ← EvolvePopulation(population)

    FOR each individual IN population DO
        fitness ← CalculateObjectiveFunction(individual)
        IF fitness > bestFitness THEN
            bestFitness ← fitness
```

```
bestState ← individual  
iterOF ← append(iterOF, bestFitness)  
→ bestState, bestFitness, iterOF
```

BAB III

HASIL DAN ANALISIS

3.1. Hasil Pengujian

3.1.1. Hill Climbing

3.1.1.1. Steepest Ascent Hill Climbing

3.1.1.1.1. Percobaan Pertama

TEST 1

Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Konfigurasi awal kubus:

Layer 1:

80	107	119	53	109
105	41	29	67	12
23	55	4	78	110
89	22	90	123	34
26	98	19	6	8

Layer 2:

74	79	101	16	70
39	57	91	87	72
5	7	43	3	104
51	92	15	111	47
103	83	88	122	115

Layer 3:

77	40	100	96	50
84	60	28	112	73
106	94	85	71	44
81	108	21	20	58
56	37	62	93	13

Layer 4:

10	49	52	118	75
36	97	18	24	76
46	33	113	11	114

124	38	54	116	32
27	9	35	120	64

Layer 5:

65	125	117	102	61
121	17	48	31	95
86	45	42	66	69
1	99	63	82	59
14	25	2	68	30

HASIL

RESULTS

25	120	2	68	89
7	118	97	47	56
95	45	90	66	19
122	22	48	31	92
63	10	78	103	59

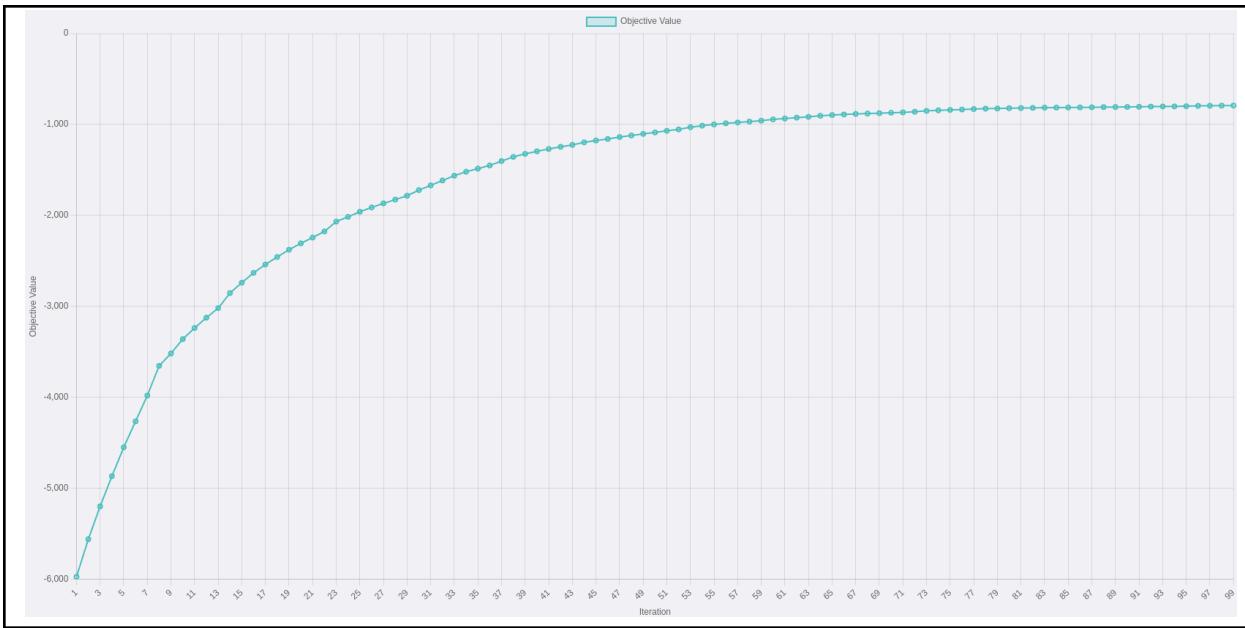
Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Steepest Ascent Hill Climbing** solution in **20.96 seconds!**

Final State Objective Value: **-792**

Iterations needed: **99**



3.1.1.1.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS

25	120	2	68	89
7	118	97	47	56
95	45	90	66	19
122	22	48	31	92
63	10	78	103	59

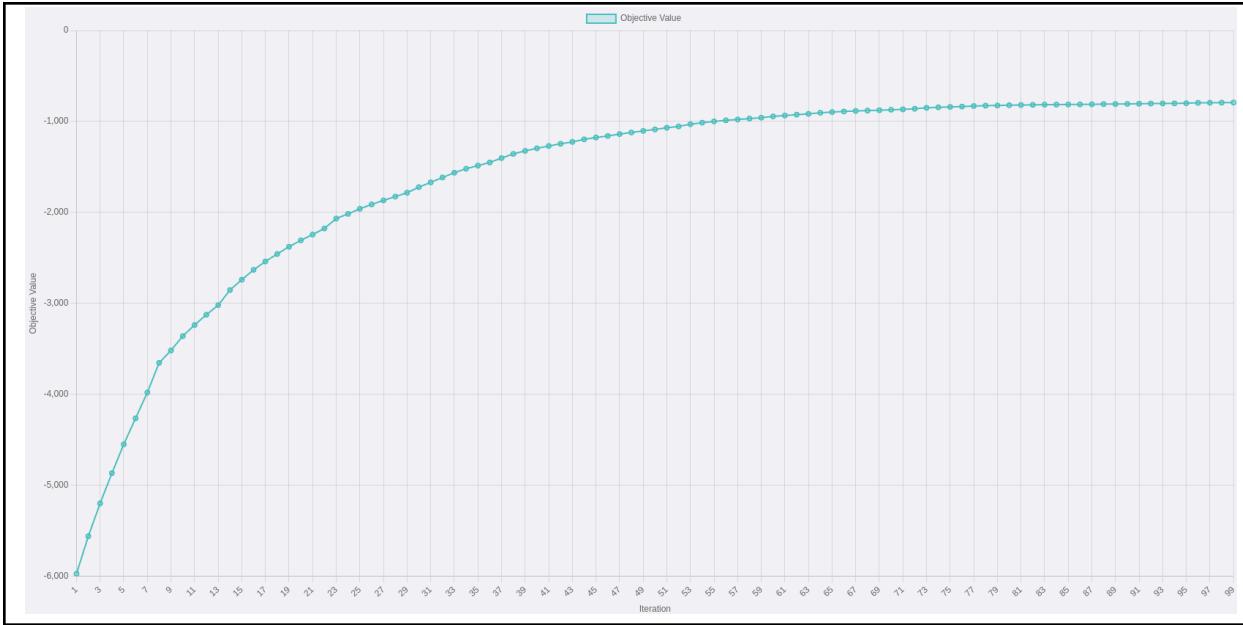
Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Steepest Ascent Hill Climbing** solution in **21.26 seconds!**

Final State Objective Value: **-792**

Iterations needed: **99**



3.1.1.1.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS

25	120	2	68	89
7	118	97	47	56
95	45	90	66	19
122	22	48	31	92
63	10	78	103	59

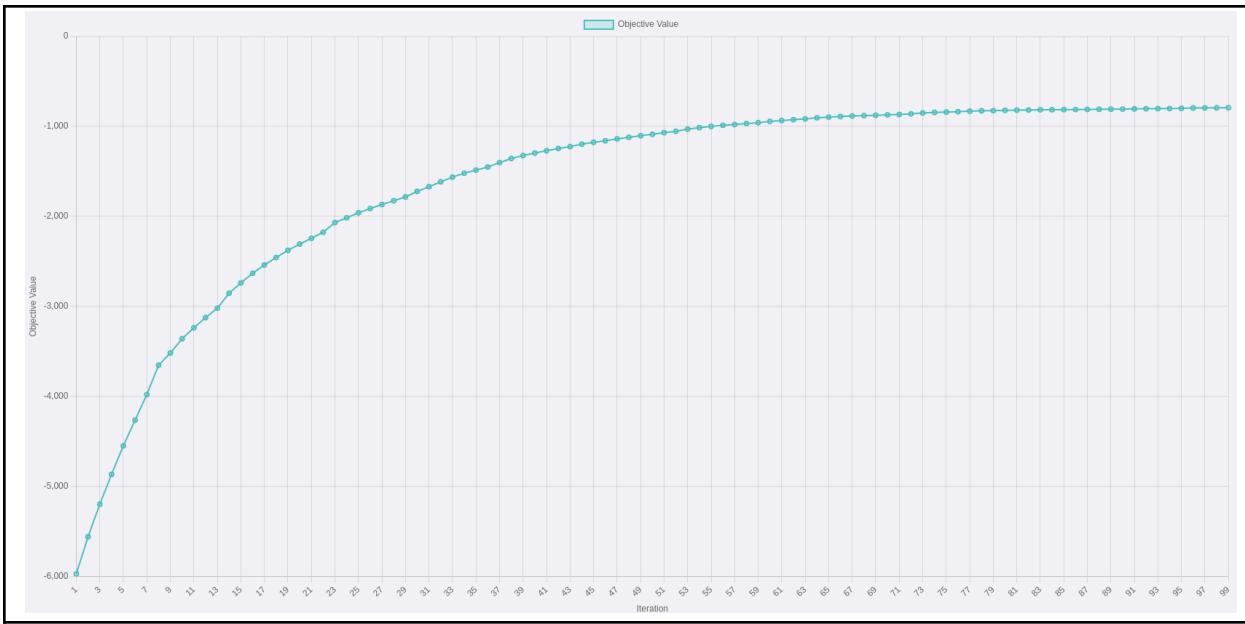
Separate by X:
 Separate by Y:
 Separate by Z:
 Select Z Level:
 Current Level: 4

Drag to pan. Scroll to zoom.

Found **Steepest Ascent Hill Climbing** solution in **20.75 seconds!**

Final State Objective Value: **-792**

Iterations needed: **99**



3.1.1.2. Hill Climbing with Sideways Move

3.1.1.2.1. Percobaan Pertama

TEST 1

22	43	74	102	16
9	33	67	99	70
124	111	76	57	97
5	15	106	28	54
42	41	36	109	52

Drag to pan. Scroll to zoom.

Konfigurasi awal kubus:

Layer 1:

85	13	65	98	66
92	10	100	4	78
35	26	116	17	1
122	12	56	104	114
49	50	108	87	110

Layer 2:

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level: Current Level: 4

83	123	89	40	20
7	101	45	80	55
94	62	29	77	79
46	27	25	51	34
3	86	38	90	119

Layer 3:

105	82	19	24	69
73	125	72	113	8
31	14	95	115	61
91	58	75	2	23
64	32	88	120	107

Layer 4:

21	112	63	37	48
68	53	81	47	103
60	6	117	11	71
96	44	84	118	121
30	93	59	39	18

Layer 5:

42	41	36	109	52
5	15	106	28	54
124	111	76	57	97
9	33	67	99	70
22	43	74	102	16

HASIL

RESULTS

76	109	77	14	39
7	75	56	124	52
79	112	22	6	97
71	8	103	66	70
82	13	58	105	57

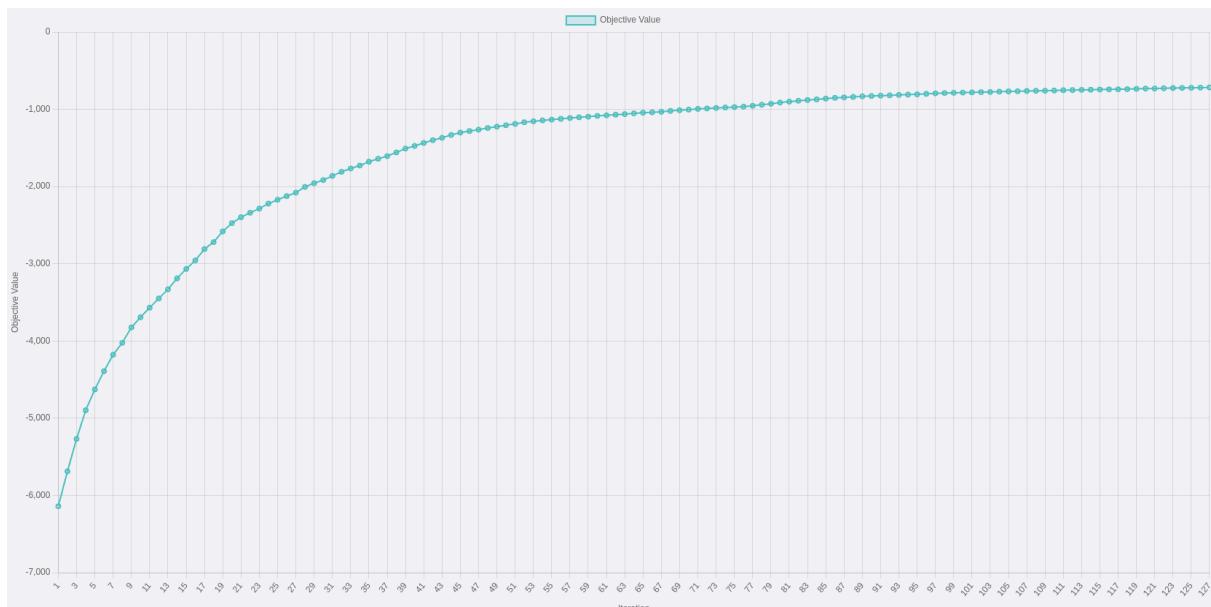
Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Drag to pan. Scroll to zoom.

Found **Sideways Move Hill Climbing** solution in **25.61 seconds!**

Final State Objective Value: **-716**

Iterations needed: **127**



3.1.1.2.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS

76	109	77	14	39
7	75	56	124	52
79	112	22	6	97
71	8	103	66	70
82	13	58	105	57

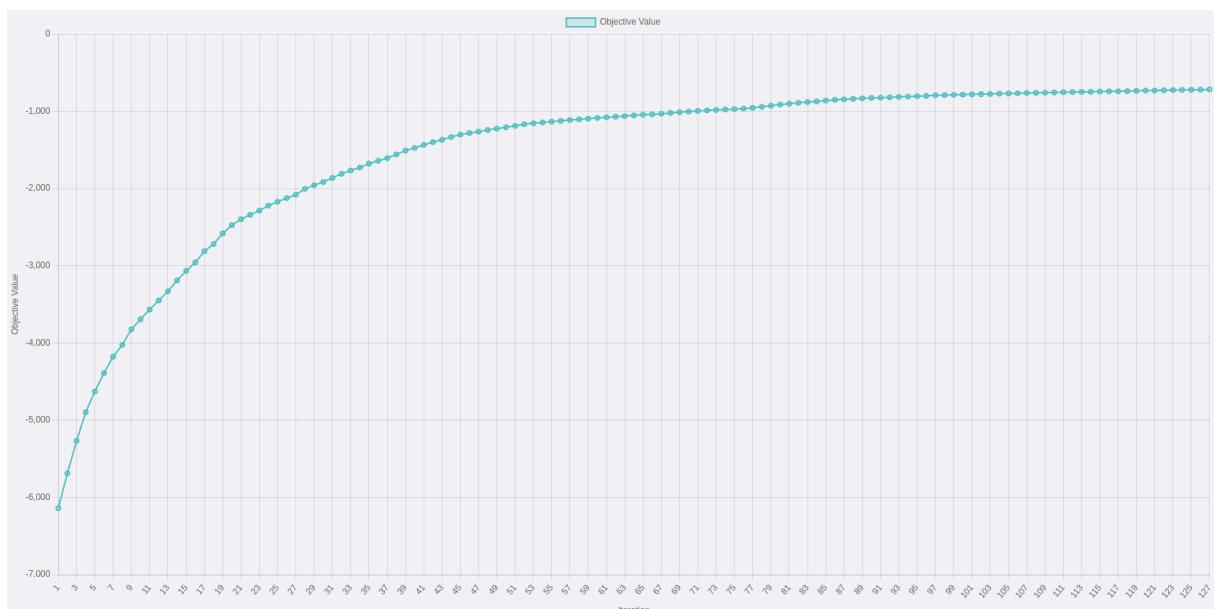
Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Drag to pan. Scroll to zoom.

Found **Sideways Move Hill Climbing** solution in **26.71 seconds!**

Final State Objective Value: **-716**

Iterations needed: **127**



3.1.1.2.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS

76	109	77	14	39
7	75	56	124	52
79	112	22	6	97
71	8	103	66	70
82	13	58	105	57

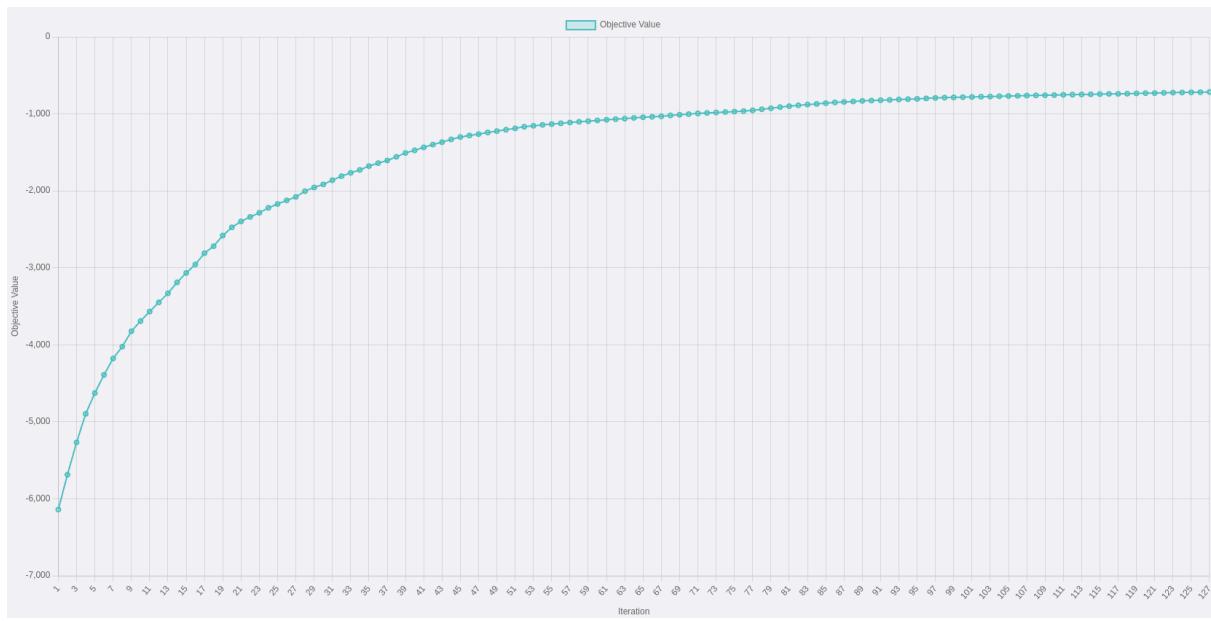
Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Drag to pan. Scroll to zoom.

Found **Sideways Move Hill Climbing** solution in **27.21 seconds!**

Final State Objective Value: **-716**

Iterations needed: **127**



3.1.1.3. Random Restart Hill Climbing

3.1.1.3.1. Percobaan Pertama

TEST 1

121	37	32	31	71
104	62	29	40	66
116	79	15	51	107
102	92	69	23	35
91	53	76	34	55

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Drag to pan. Scroll to zoom.

Restart Chance (in %):

70

Maximum Restart:

3

Konfigurasi awal kubus:

Layer 1:

94	59	85	117	4
67	70	26	110	18
39	105	50	38	24
9	77	64	52	120
86	56	72	49	111

Layer 2:

33	44	20	99	57
82	45	42	87	17
43	123	46	61	80
12	2	1	41	47
25	74	125	101	11

Layer 3:

22	96	113	60	98
112	10	93	84	73
13	83	90	7	36
114	48	14	54	68
27	19	88	100	89

IF3170 Inteligensi Artifisial - Tugas Besar 1 | 30

Layer 4:

21	75	58	30	115
6	95	63	118	108
119	65	109	103	78
16	122	8	5	106
3	28	124	97	81

Layer 5:

91	53	76	34	55
102	92	69	23	35
116	79	15	51	107
104	62	29	40	66
121	37	32	31	71

HASIL

RESULTS

85	21	95	12	103
42	57	72	58	86
105	114	2	73	26
15	84	45	113	60
68	39	101	48	61

Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

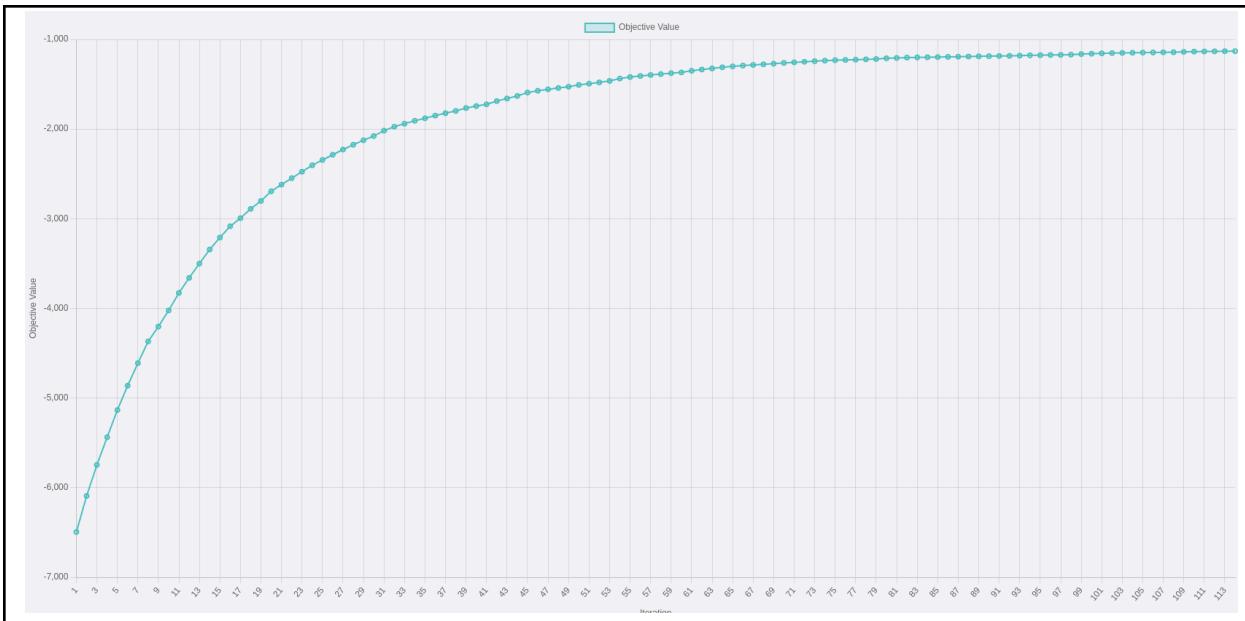
Select Z Level:

Current Level: 4

Found **Random Restart Hill Climbing** solution in **49.17 seconds!**

Final State Objective Value: **-1131**

Restarts: **1**



3.1.1.3.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS

12	79	108	52	64
22	118	24	62	89
91	20	70	32	100
116	44	57	76	21
74	54	56	94	42

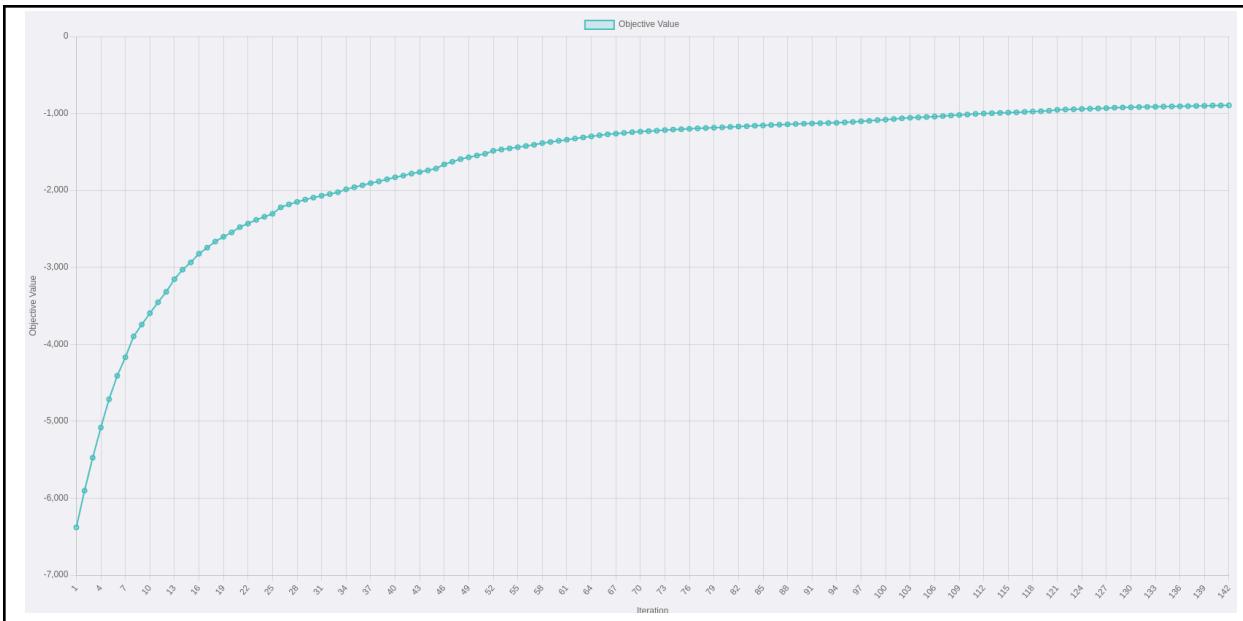
Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Random Restart Hill Climbing** solution in **114.15 seconds!**

Final State Objective Value: **-895**

Restarts: **3**



3.1.1.3.3. Percobaan Ketiga

TEST 3

43	73	1	78	82
51	105	14	16	19
41	89	103	62	49
80	108	22	115	98
87	72	12	35	52

Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Restart Chance (in %):

70

Maximum Restart:

3

Konfigurasi awal kubus:

Layer 1:

9	123	8	58	119
125	27	40	60	96

55	57	47	33	85
118	91	34	18	109
65	88	97	75	5

Layer 2:

110	48	124	53	114
77	79	76	24	107
20	26	17	83	56
106	32	46	59	121
112	31	11	90	13

Layer 3:

122	86	25	54	10
2	36	102	74	39
104	81	63	29	61
7	117	37	42	30
99	4	120	116	100

Layer 4:

92	50	71	93	66
3	84	45	70	64
28	68	44	21	94
95	15	38	111	101
69	113	23	67	6

Layer 5:

87	72	12	35	52
80	108	22	115	98
41	89	103	62	49
51	105	14	16	19
43	73	1	78	82

HASIL

RESULTS

43	76	27	82	87
51	109	120	16	19
24	21	88	124	52
113	40	22	32	94
84	71	60	26	73

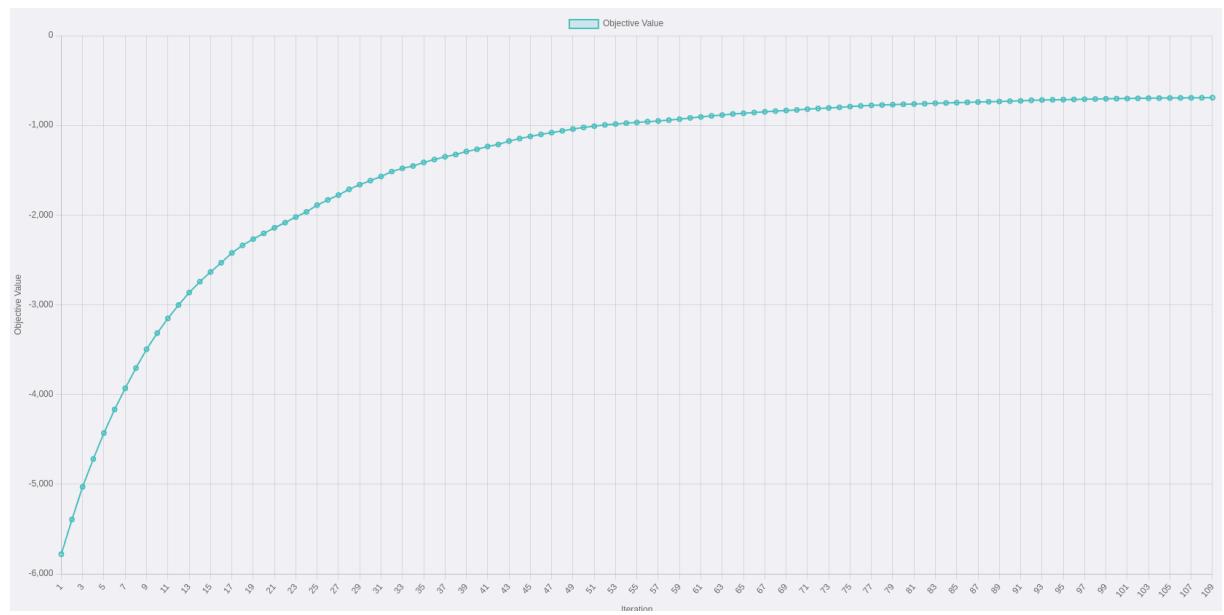
Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Random Restart Hill Climbing** solution in **22.98 seconds!**

Final State Objective Value: **-691**

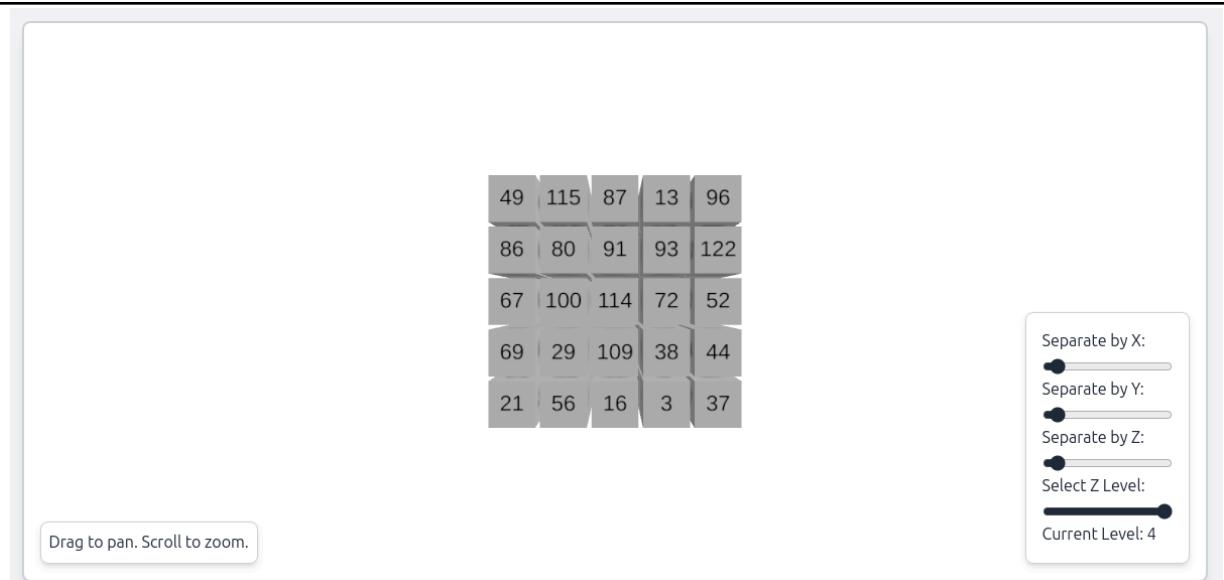
Restarts: **0**



3.1.1.4. Stochastic Hill Climbing

3.1.1.4.1. Percobaan Pertama

TEST 1



Max State Generation:

1000000

Konfigurasi awal kubus:

Layer 1:

84	43	119	79	54
31	14	112	77	36
125	58	94	124	106
55	81	46	23	105
98	11	30	62	18

Layer 2:

33	48	83	118	108
82	88	74	8	24
40	1	85	60	42
71	70	76	9	97
35	63	19	5	47

Layer 3:

120	32	123	51	26
89	104	111	12	22
17	57	10	116	92
27	39	90	6	75
2	34	61	95	50

Layer 4:

7	15	110	25	113
64	41	59	101	28
20	102	53	121	66

4	117	73	99	68
45	78	103	65	107

Layer 5:

21	56	16	3	37
69	29	109	38	44
67	100	114	72	52
86	80	91	93	122
49	115	87	13	96

HASIL

RESULTS

45	80	50	15	29
97	19	121	79	56
102	112	58	110	43
122	111	2	53	67
68	101	89	117	46

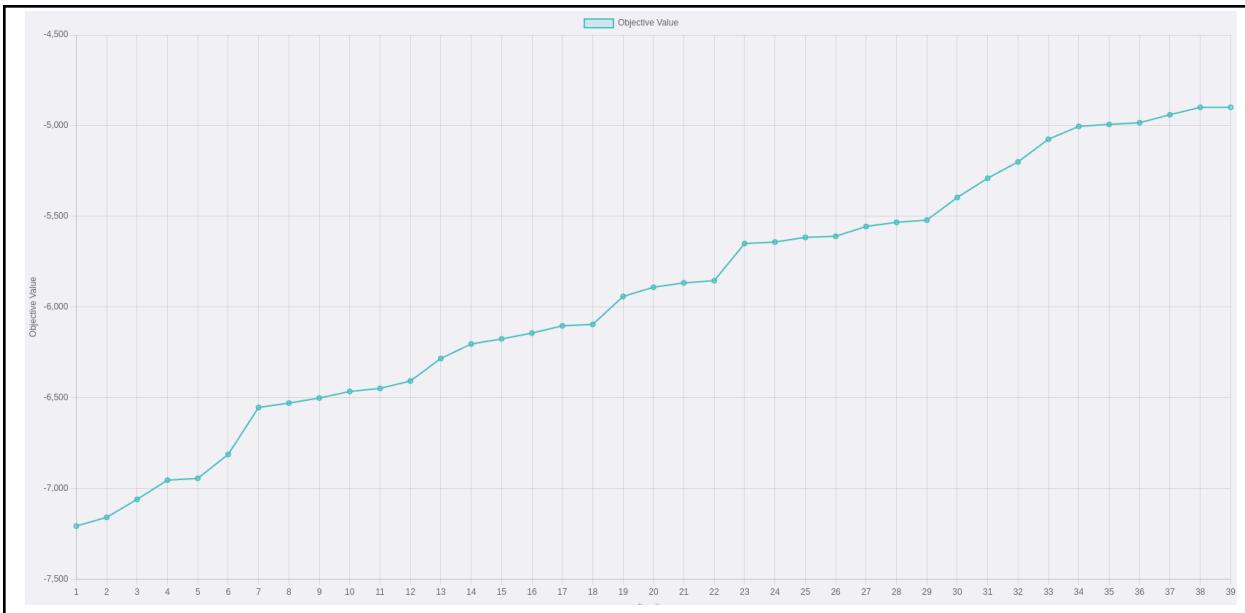
Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Stochastic Hill Climbing** solution in **19.06 seconds!**

Final State Objective Value: **-4899**

Iterations needed: **39**



3.1.1.4.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS

14	15	40	29	63
105	117	70	102	85
96	55	51	6	26
24	19	39	101	41
109	61	69	65	20

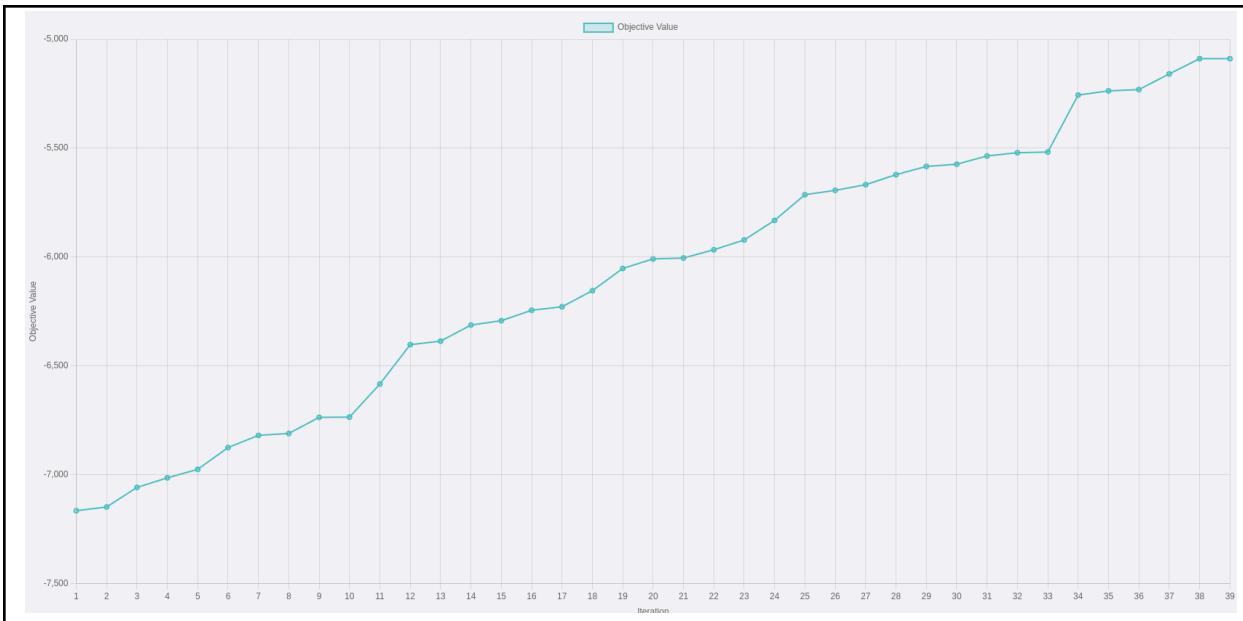
Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Drag to pan. Scroll to zoom.

Found **Stochastic Hill Climbing** solution in **9.95 seconds!**

Final State Objective Value: **-5089**

Iterations needed: **39**



3.1.1.4.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS

33	26	91	86	124
7	13	52	53	48
60	55	36	50	42
99	66	107	81	100
47	80	22	20	64

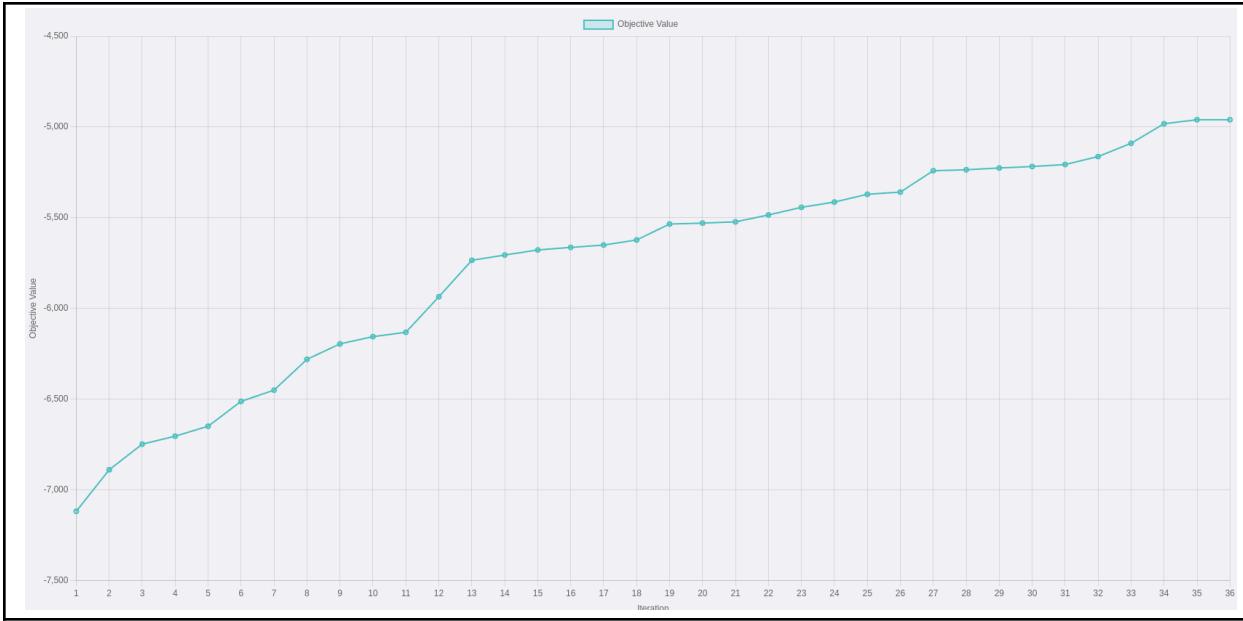
Separate by X:
 Separate by Y:
 Separate by Z:
 Select Z Level:
 Current Level: 4

Drag to pan. Scroll to zoom.

Found **Stochastic Hill Climbing** solution in **14.30 seconds!**

Final State Objective Value: **-4960**

Iterations needed: **36**



3.1.2. Simulated Annealing

3.1.2.1. Percobaan Pertama

TEST 1

49	115	87	13	96
86	80	91	93	122
67	100	114	72	52
69	29	109	38	44
21	56	16	3	37

Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Temperature:

100

Cooling Rate:

0.995

Max Iterations:

30000

Konfigurasi awal kubus:

Layer 1:

84	43	119	79	54
31	14	112	77	36
125	58	94	124	106
55	81	46	23	105
98	11	30	62	18

Layer 2:

33	48	83	118	108
82	88	74	8	24
40	1	85	60	42
71	70	76	9	97
35	63	19	5	47

Layer 3:

120	32	123	51	26
89	104	111	12	22
17	57	10	116	92
27	39	90	6	75
2	34	61	95	50

Layer 4:

7	15	110	25	113
64	41	59	101	28
20	102	53	121	66
4	117	73	99	68
45	78	103	65	107

Layer 5:

21	56	16	3	37
69	29	109	38	44
67	100	114	72	52
86	80	91	93	122

49 115 87 13 96

HASIL

RESULTS

49	125	86	16	32
41	37	102	107	24
109	15	119	33	36
90	29	4	74	117
28	94	6	79	106

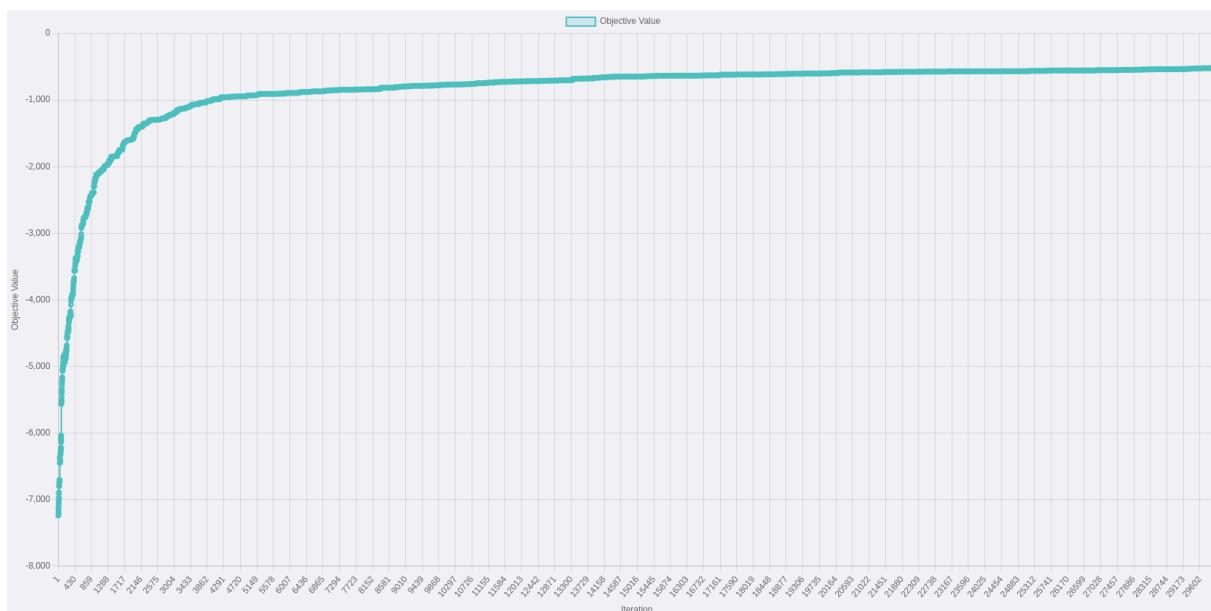
Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Drag to pan. Scroll to zoom.

Found **Simulated Annealing** solution in **0.57 seconds!**

Final State Objective Value: **-522**

Stucks Count: **29082**



3.1.2.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS

91	71	73	33	47
41	39	109	98	29
66	25	67	122	30
75	85	5	32	118
42	95	61	27	89

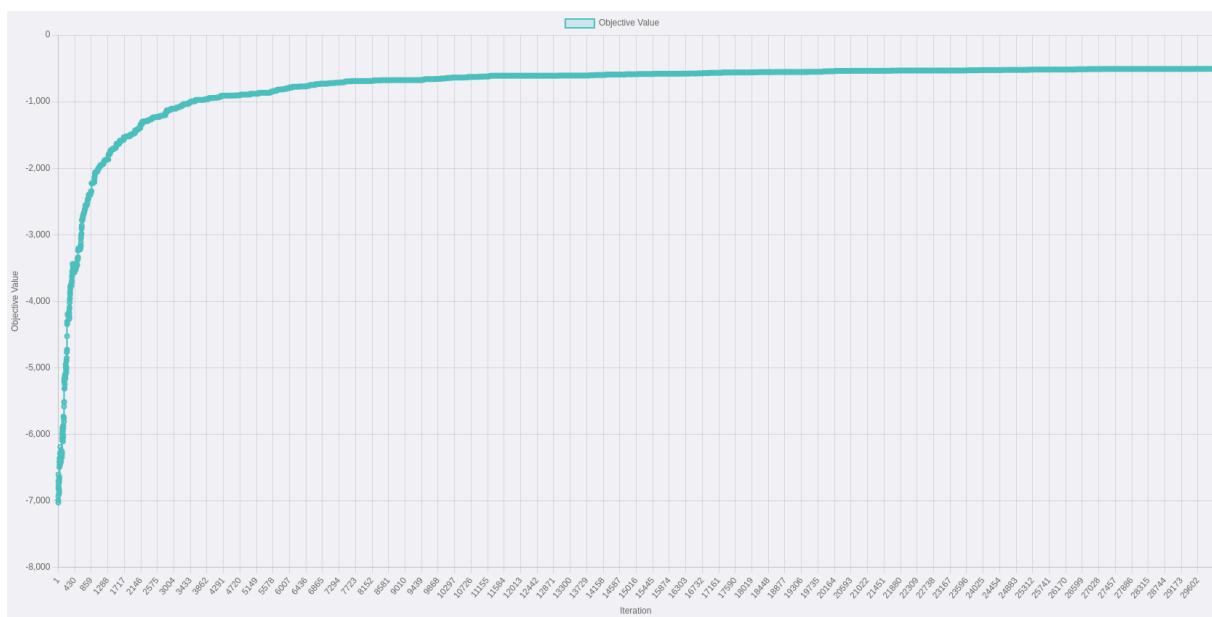
Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Drag to pan. Scroll to zoom.

Found **Simulated Annealing** solution in **0.44 seconds!**

Final State Objective Value: **-507**

Stucks Count: **29121**



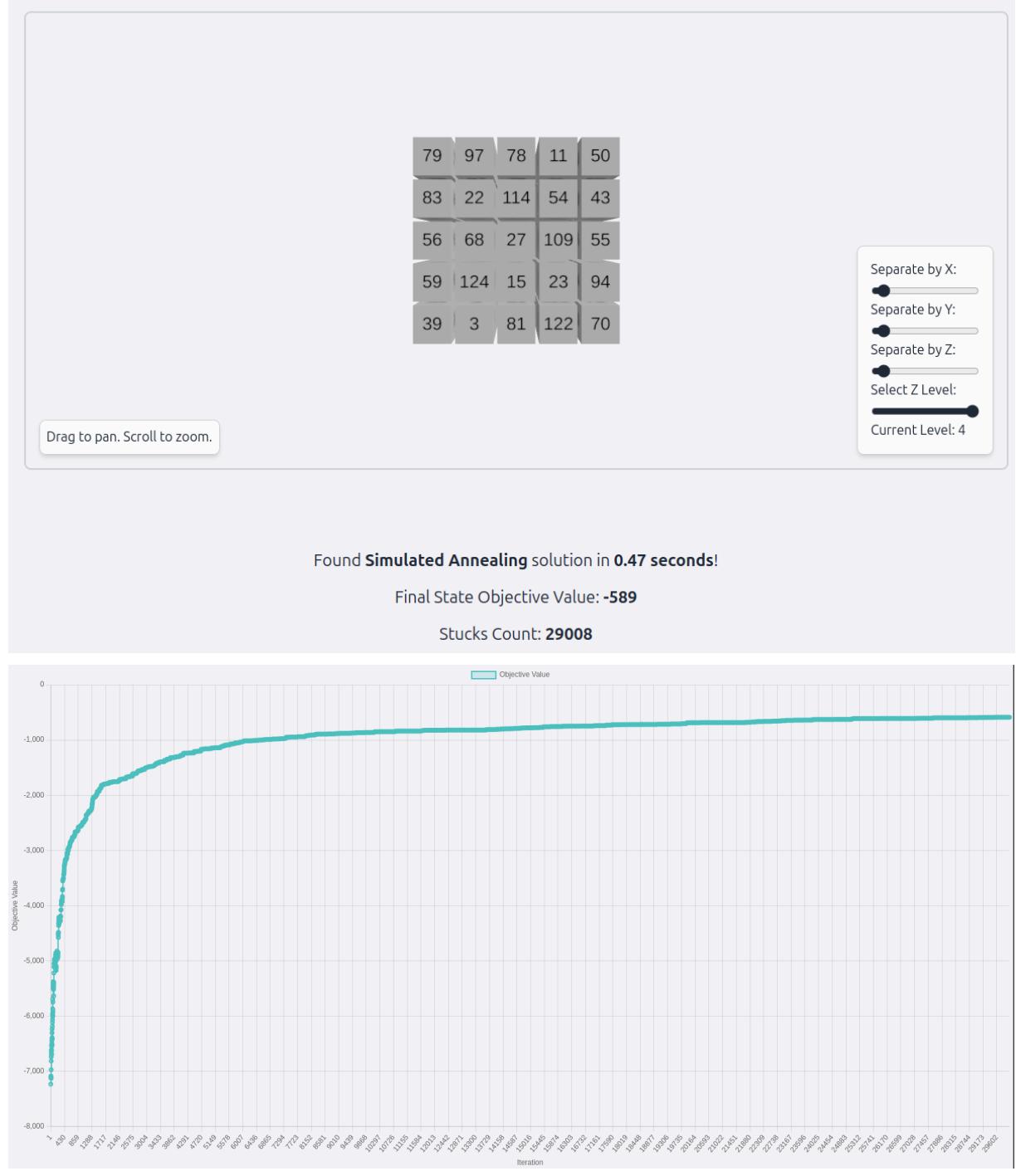
3.1.2.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



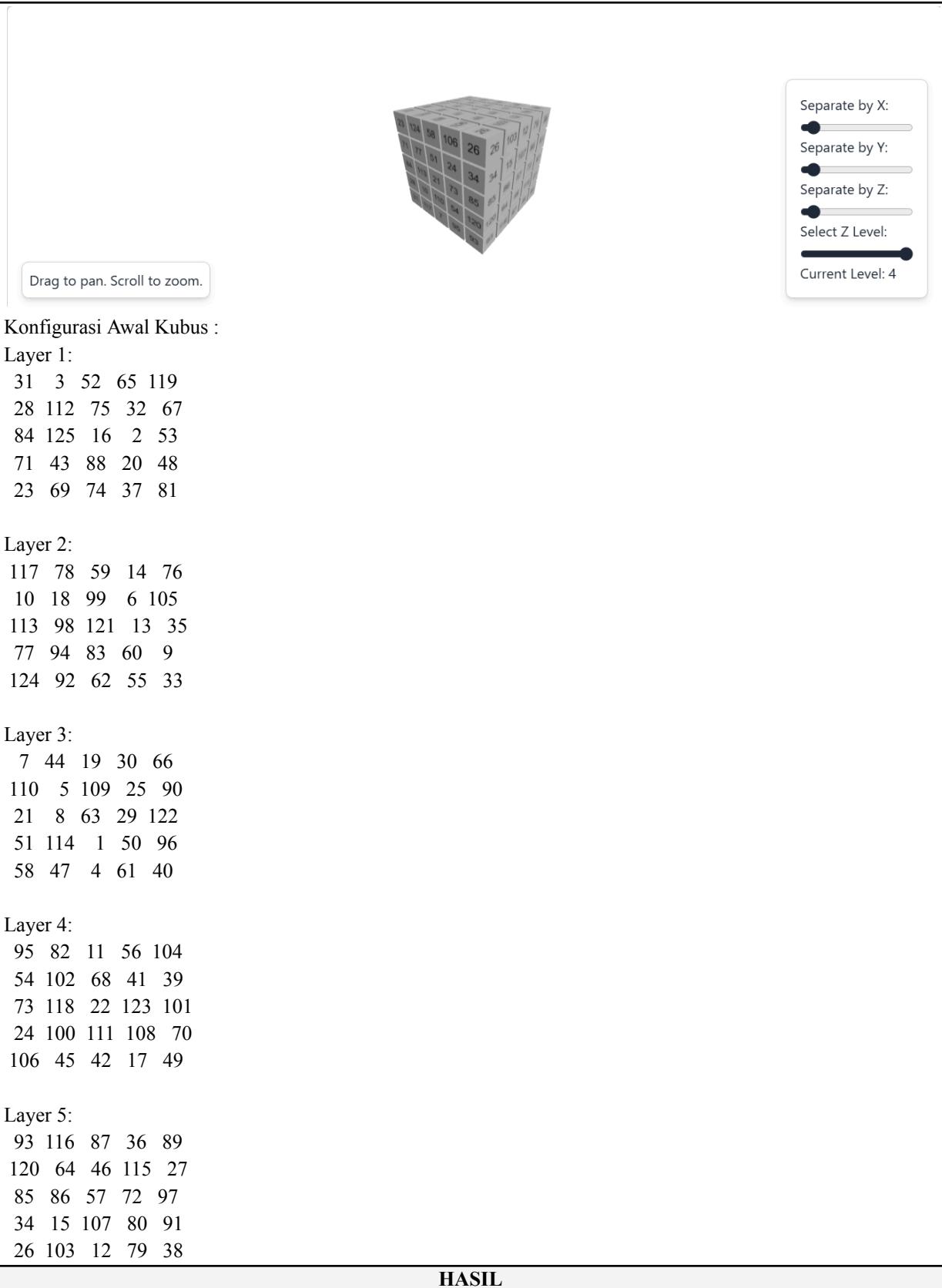
3.1.3. Genetic Algorithm

3.1.3.1. Jumlah populasi sebagai kontrol dan iterasi yang bervariasi (Populasi = 1000)

3.1.3.1.1. Variasi Iterasi 1 = 10

3.1.1.1. Percobaan Pertama

TEST 1



RESULTS



Drag to pan. Scroll to zoom.

- Separate by X:
- Separate by Y:
- Separate by Z:
- Select Z Level:
Current Level: 4

Found **Genetic Algorithm** solution in **0.59 seconds!**

Final State Objective Value: **-4613**

Iterations needed: **10**

Population Size: **1000**

Layer 1:

98	73	16	82	120
36	34	66	87	4
57	27	29	65	7
37	84	117	94	124
83	114	54	32	64

Layer 2:

106	91	76	20	52
103	86	108	74	30
119	11	49	31	59
10	112	63	23	71
44	3	123	80	79

Layer 3:

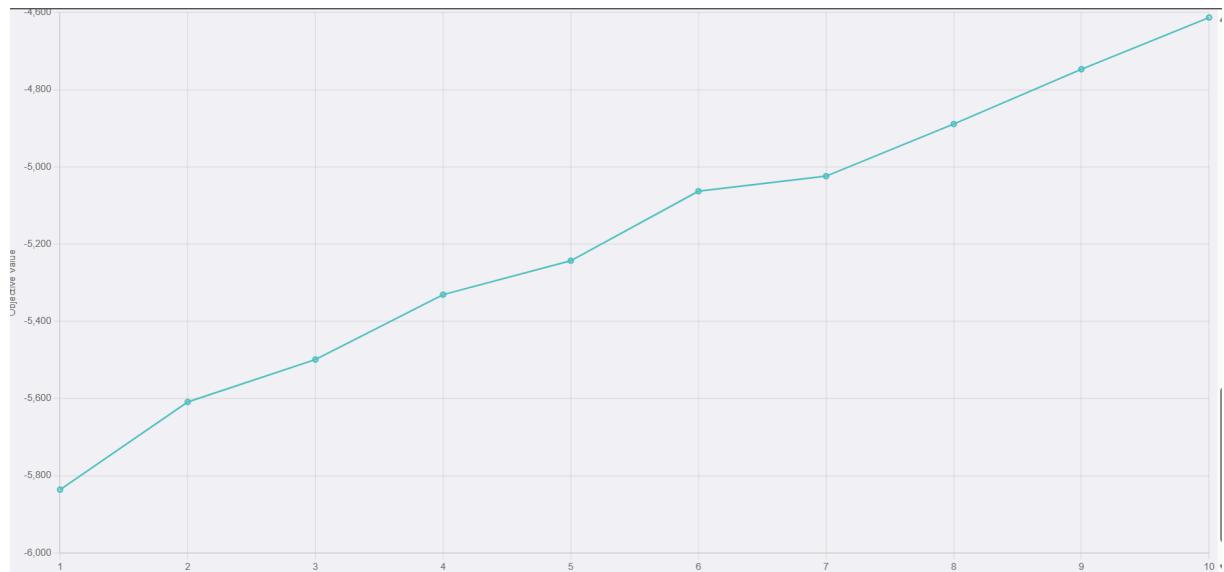
78	38	118	15	101
45	1	107	41	17
12	35	58	96	43
100	110	48	92	42
13	40	9	85	72

Layer 4:

61	5	81	89	47
53	67	111	26	90
75	125	109	2	99
88	39	8	60	18
68	97	77	102	33

Layer 5:

95	46	6	113	25
51	50	21	115	62
70	116	55	93	122
56	22	105	28	24
19	69	121	14	104



3.1.1.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Found **Genetic Algorithm** solution in **0.46 seconds!**

Final State Objective Value: **-4530**

Iterations needed: **10**

Population Size: **1000**

Layer 1:

97	121	2	22	111
117	40	110	9	83
86	44	94	76	12
84	119	34	82	64
42	63	87	67	24

Layer 2:

74	38	108	17	23
18	43	118	57	41
8	13	112	125	91
123	14	73	3	25
101	115	16	21	122

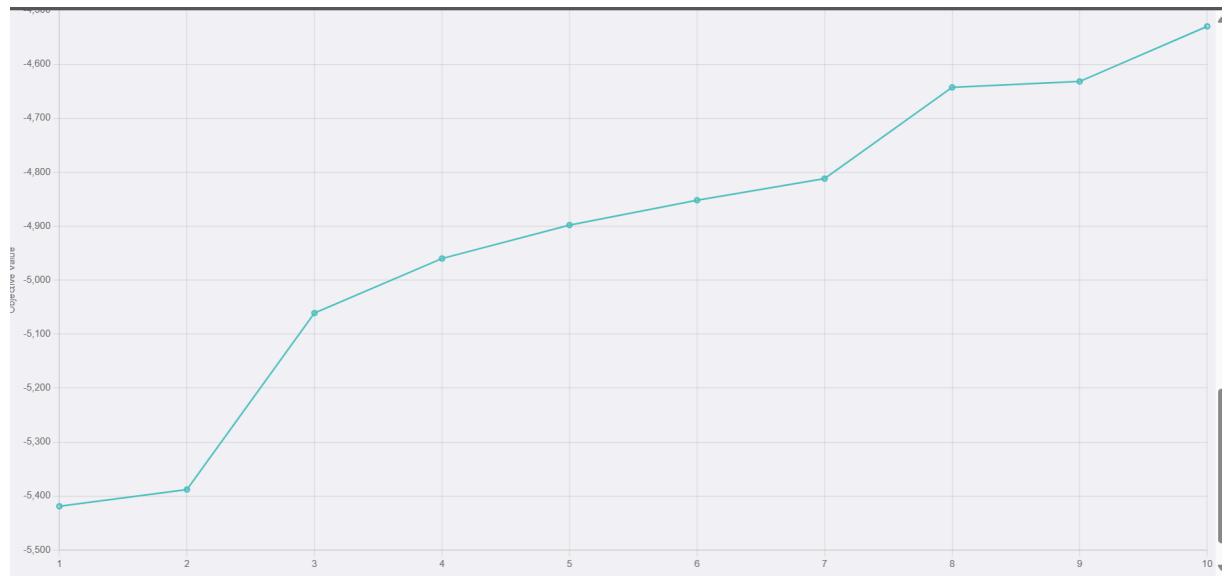
Layer 3:

47	68	54	71	28
36	50	31	75	80
11	89	72	95	4
113	39	53	15	98
105	46	58	70	88

Layer 4:

120	48	109	116	66
1	69	29	90	104
6	100	10	78	20
107	62	5	51	92
55	114	124	61	32

Layer 5:
52 81 27 65 103
106 56 45 60 26
85 99 30 7 93
37 77 33 102 79
35 19 96 59 49



3.1.1.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:
 Current Level: 4

Found **Genetic Algorithm** solution in **0.44 seconds!**

Final State Objective Value: **-4377**

Iterations needed: **10**

Population Size: **1000**

Layer 1:

43	35	88	52	1
110	112	33	79	23
46	73	92	15	113
50	38	34	87	55
84	114	63	11	48

Layer 2:

51	57	123	13	36
27	41	108	28	70
62	111	54	78	10
118	104	103	98	95
37	40	4	105	77

Layer 3:

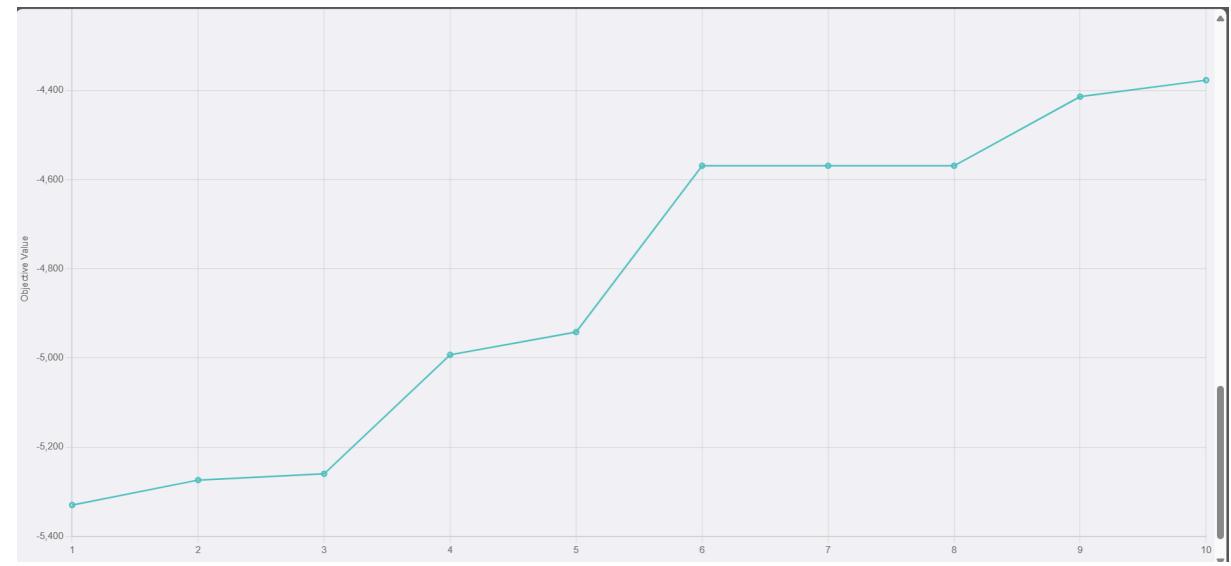
74	72	21	86	121
19	59	66	31	106
119	76	17	75	39
107	116	68	32	7
24	2	115	93	96

Layer 4:

69	60	30	89	26
22	82	102	56	14
100	71	18	53	101
85	61	16	44	109
117	81	20	42	49

Layer 5:

5	90	25	58	122
120	45	8	67	9
29	6	64	99	94
80	3	97	124	12
65	125	91	47	83



3.1.3.1.2. Variasi Iterasi 2 = 100

3.1.2.1. Percobaan Pertama

TEST 1



Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Konfigurasi awal kubus:

Layer 1:

73	47	69	11	88
48	72	65	39	61
121	17	112	100	58
102	95	4	119	122

80 107 15 98 45

Layer 2:

118 99 30 96 101
87 62 91 64 59
90 71 123 34 26
36 70 2 82 56
24 31 9 23 50

Layer 3:

57 49 105 37 10
76 22 63 77 51
68 42 27 110 21
16 40 43 44 74
32 67 25 120 115

Layer 4:

12 35 117 41 38
28 103 66 14 20
8 92 84 75 125
33 60 54 104 111
85 78 29 6 13

Layer 5:

52 86 3 55 109
89 108 113 94 5
83 97 1 106 19
7 116 114 81 46
18 53 93 79 124

HASIL

RESULTS



Found **Genetic Algorithm** solution in **3.71 seconds!**

Final State Objective Value: **-1703**

Iterations needed: **100**

Layer 1:

65	17	29	83	66
33	12	87	96	54
125	108	25	71	2
20	45	42	114	91
74	75	76	13	100

Layer 2:

89	112	28	52	32
63	68	117	47	6
5	69	41	122	99
78	31	86	24	93
88	36	48	67	84

Layer 3:

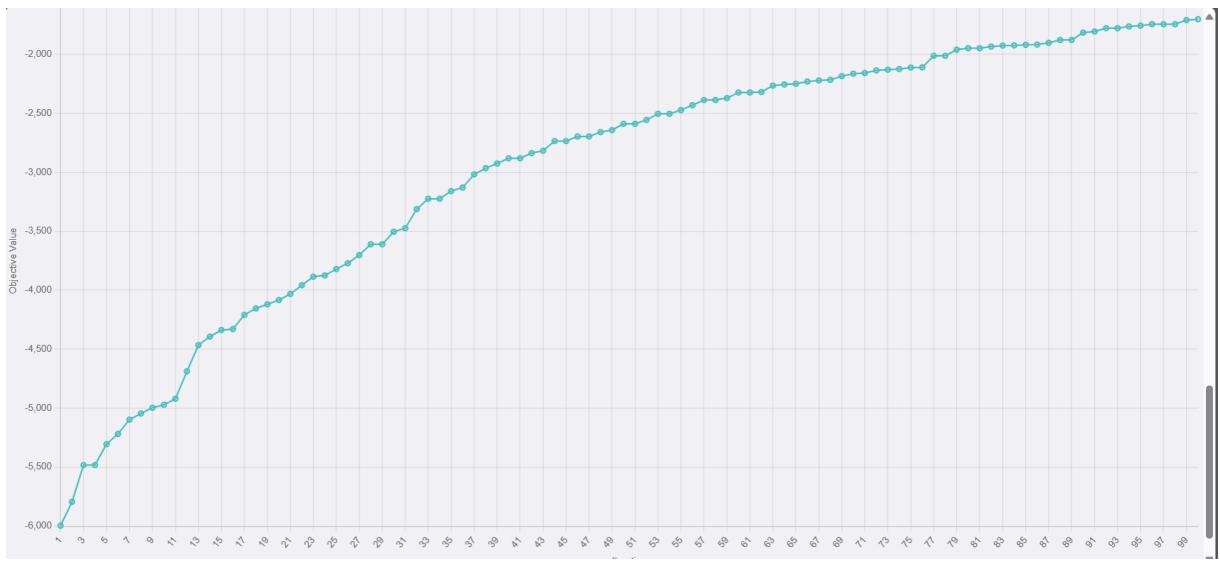
59	53	82	44	80
21	51	35	73	123
9	97	79	50	105
111	22	77	120	1
106	94	61	27	19

Layer 4:

11	56	119	16	113
85	62	37	104	34
98	30	110	4	70
115	101	3	57	18
10	46	49	124	72

Layer 5:

58	64	40	121	26
116	118	15	23	102
107	8	92	60	39
7	103	90	14	109
43	55	81	95	38



3.1.2.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Found **Genetic Algorithm** solution in **5.20 seconds!**

Final State Objective Value: **-2027**

Iterations needed: **100**

Population Size: **1000**

Layer 1:

68	38	15	125	47
113	27	7	37	122
82	124	112	4	5
34	14	121	53	97
33	114	90	61	40

Layer 2:

72	28	74	22	108
85	9	62	58	73
2	96	11	117	66
101	119	20	89	19
16	84	110	36	91

Layer 3:

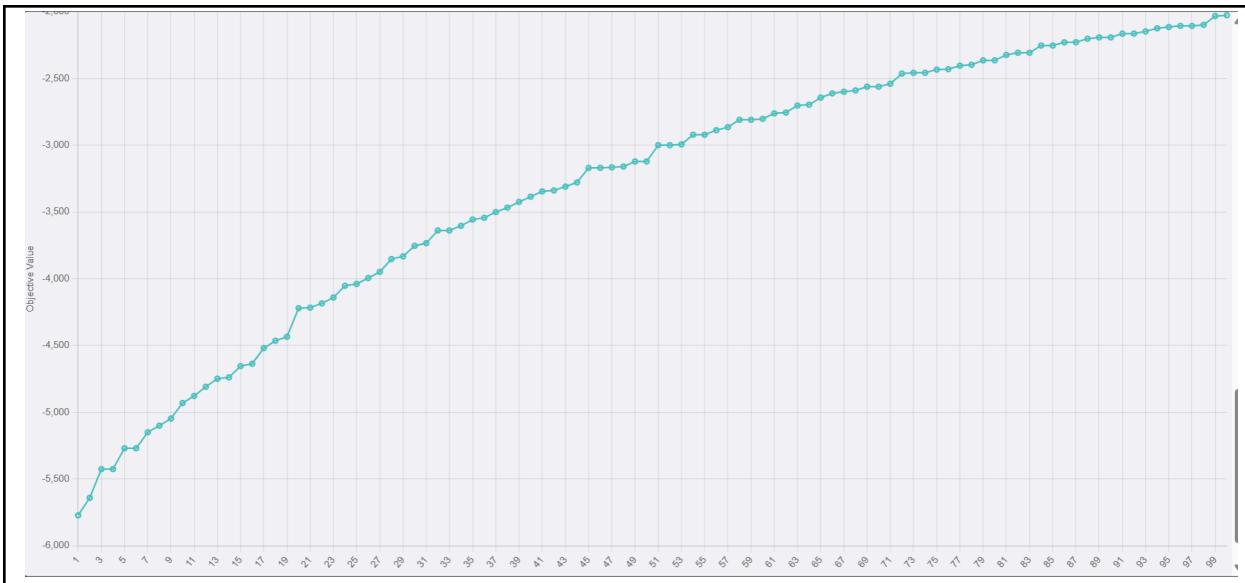
104	80	88	3	70
18	94	106	76	10
39	24	69	57	109
79	81	1	65	93
8	103	43	123	31

Layer 4:

32	120	25	87	46
116	12	64	50	63
95	23	105	26	100
30	60	115	71	45
54	92	6	77	78

Layer 5:

41	51	107	29	44
13	118	75	98	42
99	35	21	111	48
67	49	56	52	83
102	59	55	17	86



3.1.2.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

- Separate by X:
 - Separate by Y:
 - Separate by Z:
 - Select Z Level:
- Current Level: 4

Found **Genetic Algorithm** solution in **5.14 seconds!**

Final State Objective Value: **-1345**

Iterations needed: **100**

Population Size: **1000**

Layer 1:

78	85	56	5	53
38	101	34	116	33
25	13	72	120	89
102	1	54	49	117

76 81 90 17 14

Layer 2:

39 31 106 124 18
122 26 42 30 110
37 118 114 10 55
50 73 8 100 91
87 61 58 47 40

Layer 3:

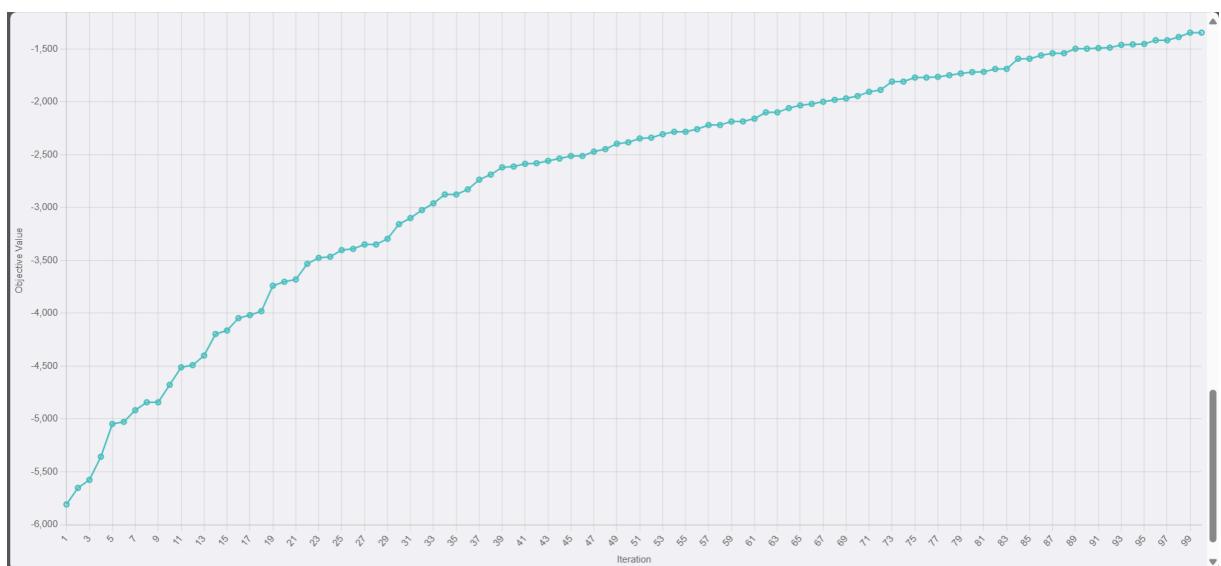
105 123 9 15 57
62 4 115 107 27
24 80 59 83 64
95 66 35 52 60
19 44 79 104 109

Layer 4:

23 63 67 103 69
77 84 108 2 45
97 21 36 70 94
41 71 92 99 20
75 119 7 32 86

Layer 5:

74 3 82 93 96
43 112 6 51 113
121 88 28 46 11
22 98 125 16 29
48 12 68 111 65



3.1.3.1.3. Variasi 3 iterasi = 200

3.1.3.1. Percobaan Pertama

TEST 1



Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Konfigurasi awal kubus:

Layer 1:

57	53	6	54	52
48	58	81	34	95
37	43	91	125	22
113	119	71	2	73
90	24	70	15	45

Layer 2:

40	23	116	44	87
80	41	39	59	107
5	63	65	106	101
33	115	11	69	16
114	55	14	29	25

Layer 3:

64	27	103	51	38
28	9	82	66	121
92	13	110	85	17
122	67	74	89	7
68	96	1	18	112

Layer 4:

4	46	105	123	78
117	79	98	19	20
3	75	109	56	118
86	35	102	31	120
32	108	61	124	83

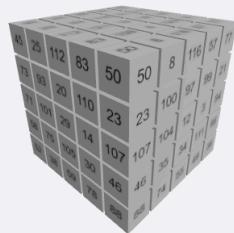
Layer 5:

10	47	50	77	8
----	----	----	----	---

62	12	94	72	49
104	36	84	111	76
42	100	93	30	97
99	26	88	60	21

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Genetic Algorithm** solution in **9.63 seconds!**

Final State Objective Value: **-943**

Iterations needed: **200**

Population Size: **1000**

Layer 1:

52	125	9	54	66
58	72	81	56	43
71	40	91	117	1
73	60	67	2	113
45	24	68	87	92

Layer 2:

38	17	120	39	96
75	82	32	65	61
101	64	15	106	11
93	115	10	69	27
25	53	118	31	114

Layer 3:

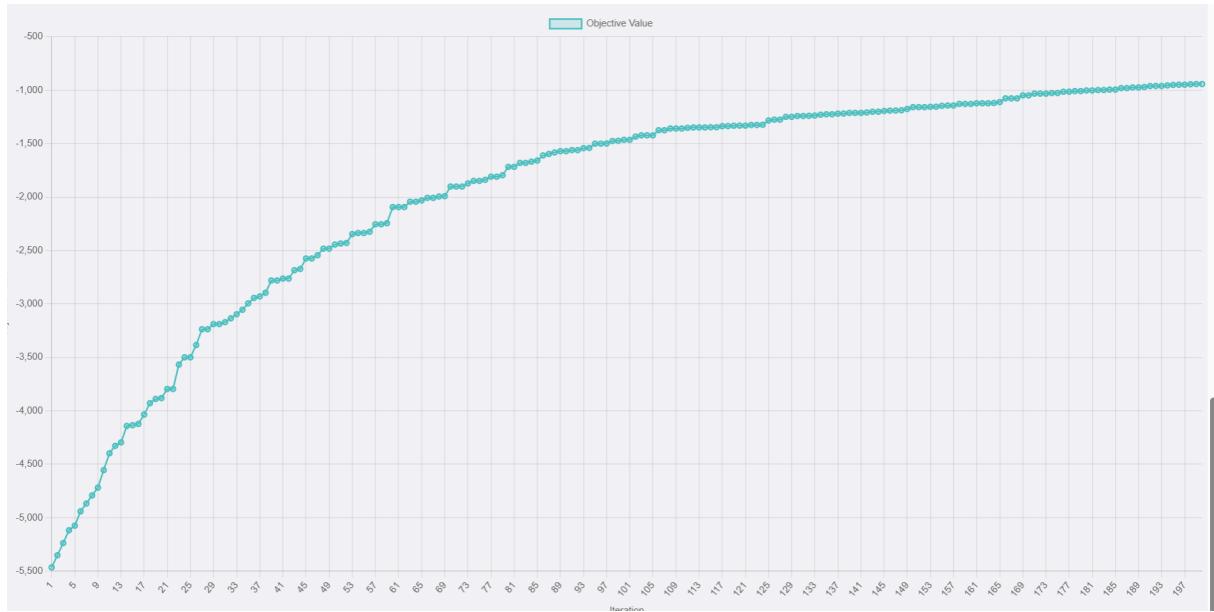
59	86	103	47	41
105	48	70	62	28
29	63	80	85	84
20	7	51	102	122
112	121	13	18	33

Layer 4:

```
78 16 22 123 76
30 79 98 19 90
14 42 109 5 119
110 37 89 44 26
83 108 4 124 6
```

Layer 5:

```
88 74 55 49 36
46 35 34 111 95
107 104 12 3 94
23 100 97 99 21
50 8 116 57 77
```



3.1.3.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Genetic Algorithm** solution in **9.92 seconds!**

Final State Objective Value: **-996**

Iterations needed: **200**

Population Size: **1000**

Layer 1:

79	54	6	107	63
9	75	83	58	95
24	112	88	48	22
113	34	16	60	92
90	52	110	37	45

Layer 2:

68	44	125	21	59
100	57	39	41	82
13	106	65	103	27
17	38	11	101	120
114	50	72	55	25

Layer 3:

70	66	31	115	35
4	51	108	43	121
64	85	67	56	36
124	109	78	8	7
61	18	30	96	116

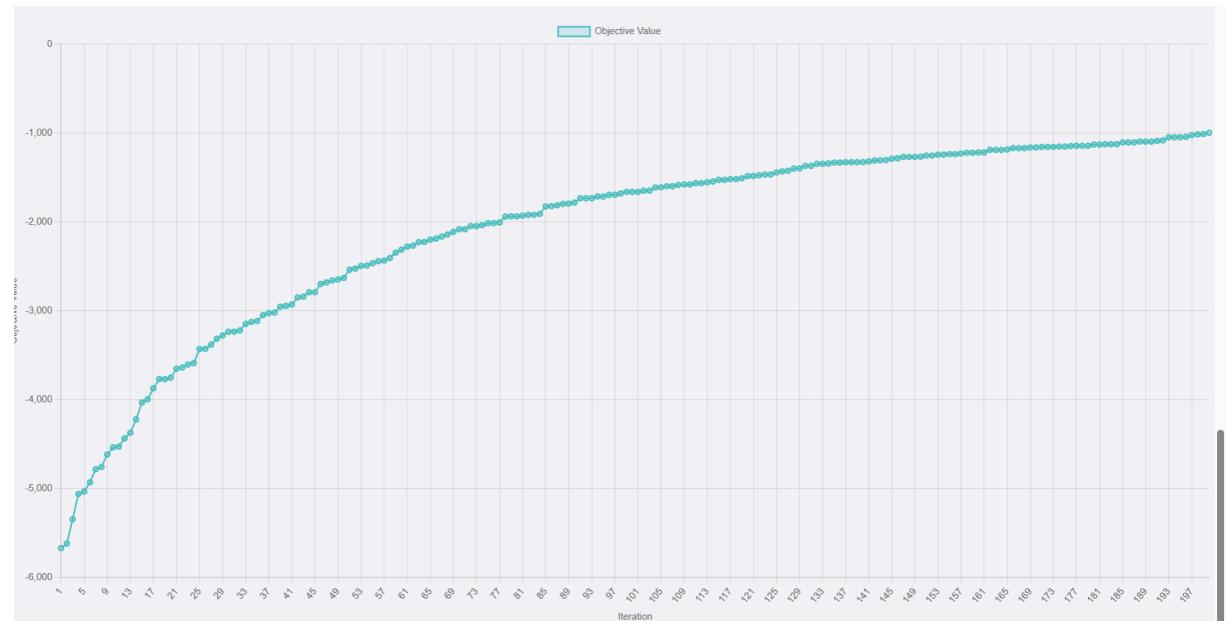
Layer 4:

19	123	73	46	53
117	33	47	99	2
91	3	89	15	118

49	40	102	69	71
32	119	5	87	74

Layer 5:

76	29	80	20	111
86	98	28	93	12
122	10	1	84	97
14	94	104	77	26
23	81	105	42	62



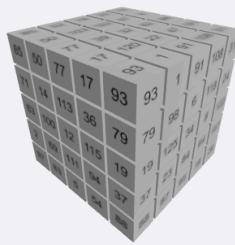
3.1.3.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Found **Genetic Algorithm** solution in **9.78 seconds!**

Final State Objective Value: **-951**

Iterations needed: **200**

Population Size: **1000**

Layer 1:

97	104	27	41	53
2	55	119	106	35
63	46	21	80	96
71	59	65	67	49
85	52	78	20	76

Layer 2:

83	28	16	114	72
69	81	29	11	123
100	30	110	90	7
14	75	62	43	121
50	103	102	57	9

Layer 3:

5	99	58	33	124
111	109	73	15	4
12	61	66	112	44
113	10	86	42	60
77	39	32	82	89

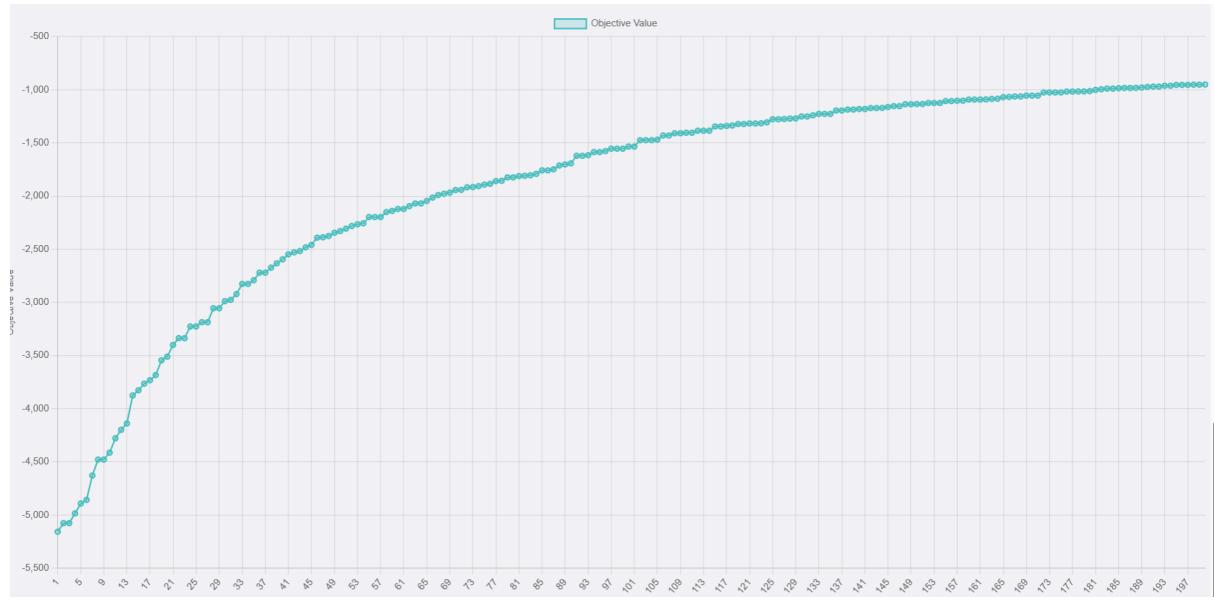
Layer 4:

54	3	116	105	38
94	40	13	117	48
115	51	74	25	56
36	70	92	45	68

17 120 22 47 101

Layer 5:

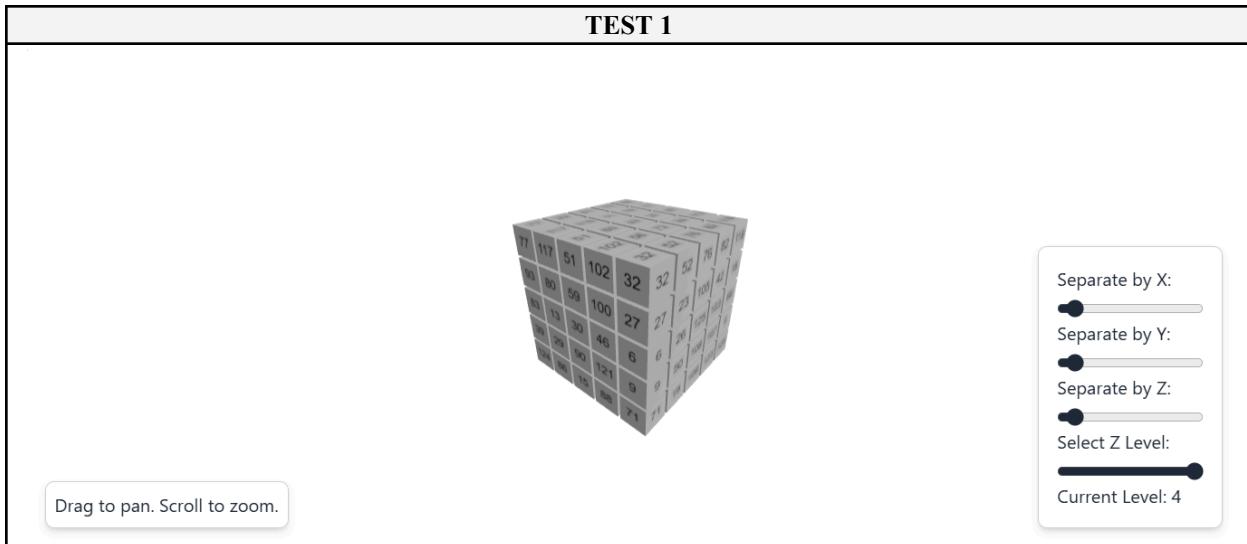
88	87	95	18	26
37	23	84	64	107
19	125	34	8	122
79	98	6	118	24
93	1	91	108	31



3.1.3.2. Jumlah iterasi sebagai kontrol dan populasi yang bervariasi (Iterasi = 150)

3.1.3.2.1. Variasi 1 Populasi = 500

3.2.1.1. Percobaan Pertama



Konfigurasi awal kubus:

Layer 1:

124	28	41	81	62
39	96	65	97	91
83	112	40	104	20
93	16	78	54	63
77	22	61	75	58

Layer 2:

86	12	70	68	94
29	110	14	64	57
13	73	7	25	114
80	48	21	33	109
117	115	11	79	17

Layer 3:

15	24	36	10	111
90	85	3	98	1
30	2	92	74	89
59	8	69	122	44
51	60	43	37	87

Layer 4:

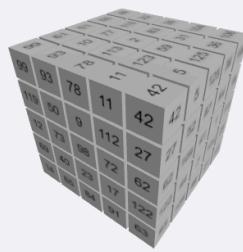
88	31	67	35	101
121	45	47	49	53
46	99	38	95	84
100	4	116	119	55
102	56	72	34	113

Layer 5:

71	19	106	123	120
9	50	108	107	5
6	26	125	103	66
27	23	105	42	18
32	52	76	82	118

HASIL

RESULTS



Drag to pan. Scroll to zoom.

- Separate by X:
- Separate by Y:
- Separate by Z:
- Select Z Level:
Current Level: 4

Found **Genetic Algorithm** solution in **4.00 seconds!**

Final State Objective Value: **-1513**

Iterations needed: **150**

Population Size: **500**

Layer 1:

14	102	33	95	80
69	121	81	29	18
12	39	53	110	108
119	4	92	32	70
99	61	51	55	49

Layer 2:

65	101	71	74	8
40	67	66	30	117
73	54	111	26	58
50	76	28	68	96
93	10	77	115	22

Layer 3:

84	37	89	15	88
23	60	118	94	25
98	45	46	100	24
9	47	57	90	116
78	113	2	83	35

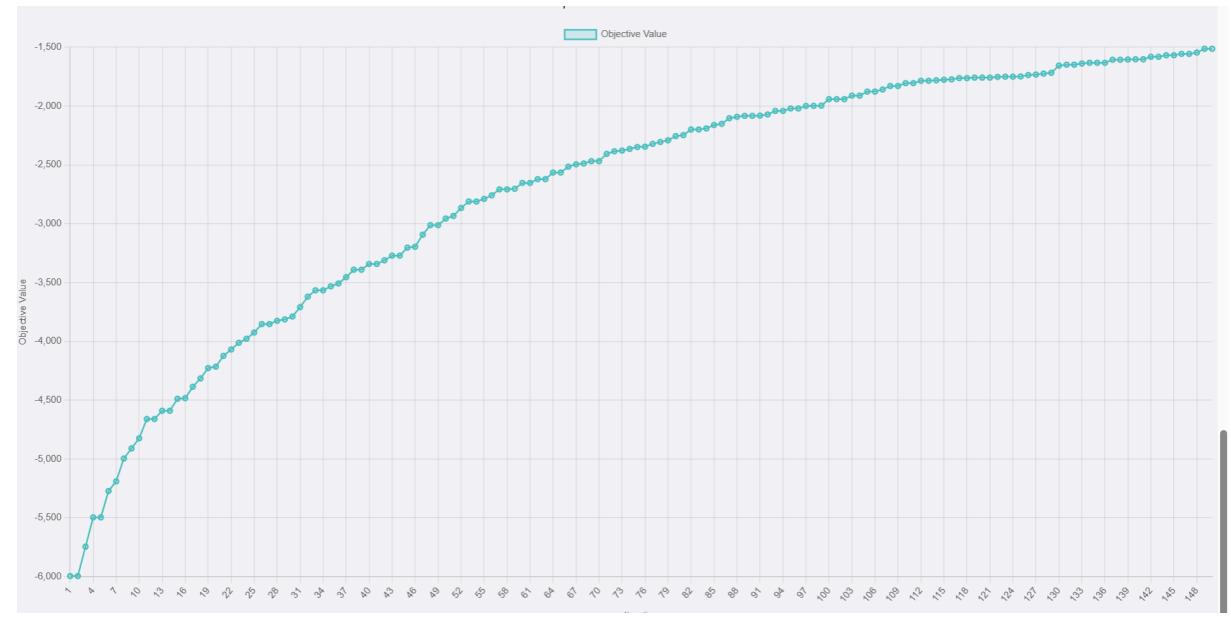
Layer 4:

91	3	34	105	85
17	16	103	38	107
72	75	79	82	6

112	97	48	56	13
11	123	59	31	104

Layer 5:

63	120	86	21	43
122	44	7	124	20
62	109	41	1	114
27	52	87	64	19
42	5	125	36	106



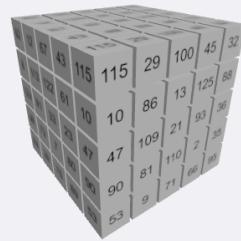
3.2.1.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Drag to pan. Scroll to zoom.

Found **Genetic Algorithm** solution in **2.71 seconds!**

Final State Objective Value: **-1745**

Iterations needed: **150**

Population Size: **500**

Layer 1:

64	112	44	117	3
49	52	4	102	108
123	1	106	14	69
6	41	119	92	57
62	105	34	54	74

Layer 2:

20	31	97	85	103
73	56	30	17	124
91	18	99	82	11
113	84	51	38	24
12	120	28	55	107

Layer 3:

75	50	63	19	111
7	121	79	96	8
33	76	83	60	104
122	25	98	5	65
67	42	59	116	27

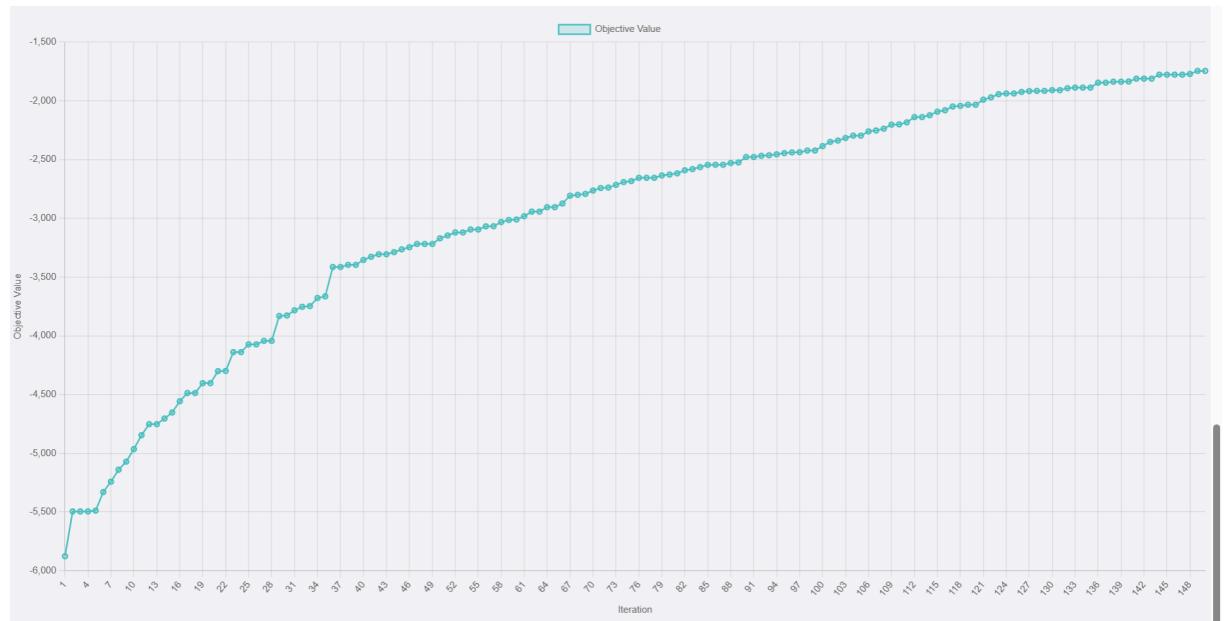
Layer 4:

89	114	39	26	16
80	15	77	101	40
23	22	70	78	118

61	87	37	68	58
43	48	94	46	72

Layer 5:

53	9	71	66	95
90	81	110	2	35
47	109	21	93	36
10	86	13	125	88
115	29	100	45	32



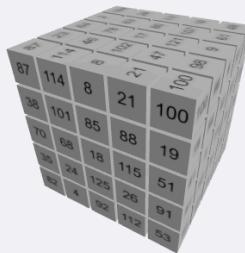
3.2.1.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Found **Genetic Algorithm** solution in **3.90 seconds!**

Final State Objective Value: **-1447**

Iterations needed: **150**

Population Size: **500**

Layer 1:

82	67	56	110	29
35	48	58	105	69
70	93	30	66	57
38	71	106	62	44
87	23	64	2	104

Layer 2:

4	36	33	113	123
24	95	108	5	83
68	99	31	39	79
101	45	65	94	12
114	46	78	63	28

Layer 3:

92	89	116	1	27
125	11	7	90	84
18	22	76	120	73
85	86	103	25	14
8	102	17	80	111

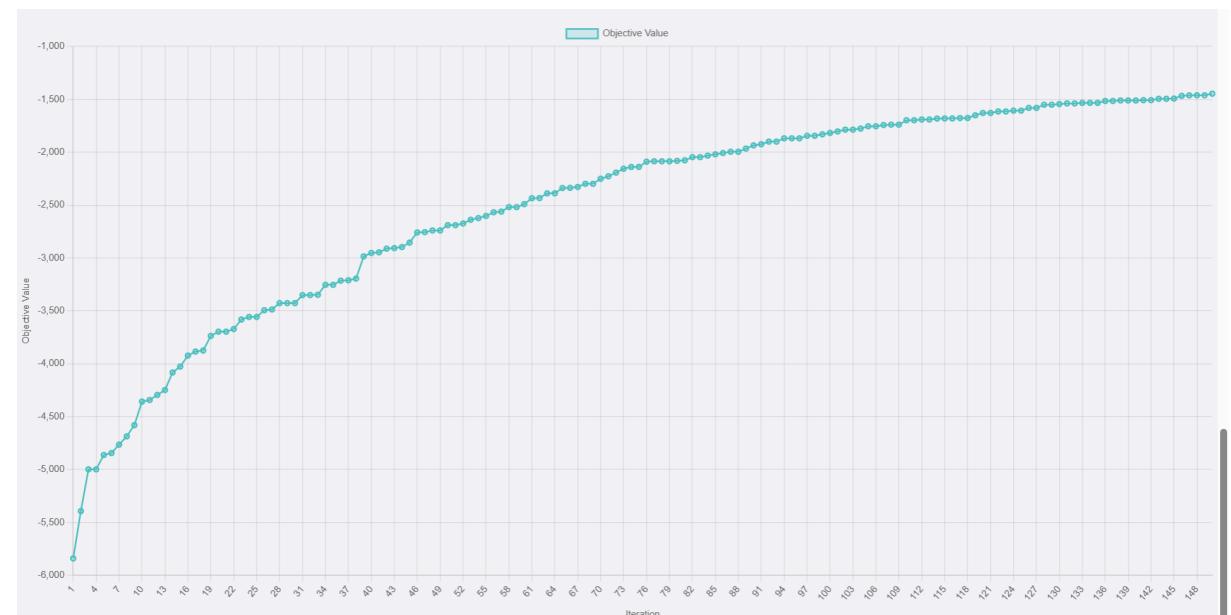
Layer 4:

112	6	37	55	119
26	124	60	75	15

115	96	43	13	52
88	54	40	16	117
21	47	121	107	10

Layer 5:

53	122	72	41	20
91	42	81	34	77
51	3	118	74	59
19	50	32	97	109
100	98	9	61	49



3.1.3.2.2. Variasi 2 Populasi = 1000

3.2.2.1. Percobaan Pertama

TEST 1

Drag to pan. Scroll to zoom.

Konfigurasi awal kubus:

Layer 1:

60	72	29	5	30
125	77	97	4	39
75	67	34	90	69
116	11	106	96	109
66	23	56	15	83

Layer 2:

37	50	102	80	108
122	121	17	65	103
53	86	9	74	54
59	43	63	95	31
19	52	70	68	79

Layer 3:

71	28	101	117	35
3	51	12	26	27
113	45	42	94	1
84	21	81	57	18
87	100	82	32	20

Layer 4:

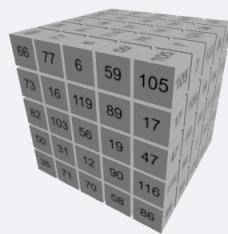
46	85	114	13	10
33	120	107	8	36
55	25	7	38	105
62	104	40	124	47
111	41	78	99	58

Layer 5:

115	119	48	24	16
88	92	44	110	98
89	73	22	76	14
118	6	112	64	123
61	2	49	93	91

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Genetic Algorithm** solution in **7.88 seconds!**

Final State Objective Value: **-1986**

Iterations needed: **150**

Population Size: **1000**

Layer 1:

35	88	42	64	85
50	34	92	27	91
82	107	75	4	36
73	45	43	104	67
66	39	61	106	38

Layer 2:

71	120	32	51	41
31	97	46	78	63
103	29	115	55	14
16	5	60	112	113
77	81	48	23	84

Layer 3:

70	3	52	69	122
12	94	101	95	8
56	21	28	108	99
119	87	98	2	10
6	114	37	44	110

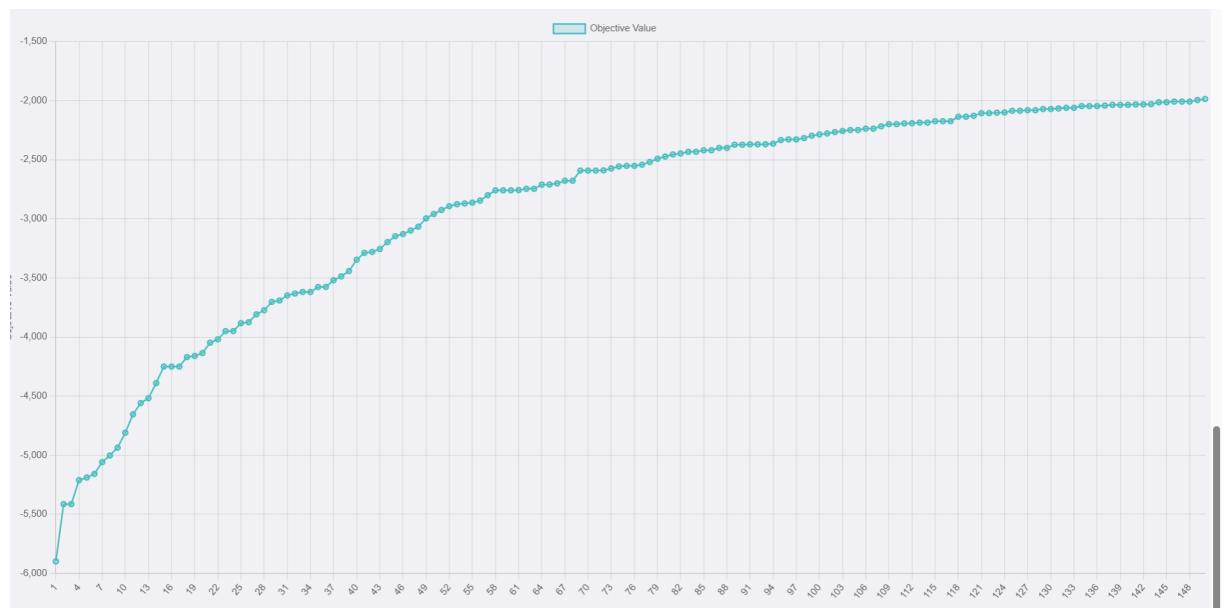
Layer 4:

58	30	68	102	57
90	11	65	25	121
19	124	74	62	40
89	72	96	49	13

59 76 15 79 100

Layer 5:

86 83 125 20 9
116 80 1 93 24
47 33 22 117 111
17 109 18 54 118
105 7 123 26 53



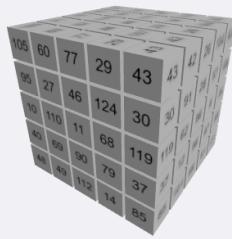
3.2.2.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:
 Separate by Y:
 Separate by Z:
 Select Z Level:
Current Level: 4

Found **Genetic Algorithm** solution in **7.78 seconds!**

Final State Objective Value: **-1232**

Iterations needed: **150**

Population Size: **1000**

Layer 1:

48	32	35	114	81
40	125	5	56	100
10	115	92	84	15
95	4	117	2	78
105	31	75	50	47

Layer 2:

49	122	8	58	76
69	67	93	19	55
110	53	54	99	24
27	22	45	102	106
60	52	116	38	44

Layer 3:

112	33	83	39	25
90	13	64	65	70
11	71	74	20	123
46	113	17	73	59
77	86	80	111	36

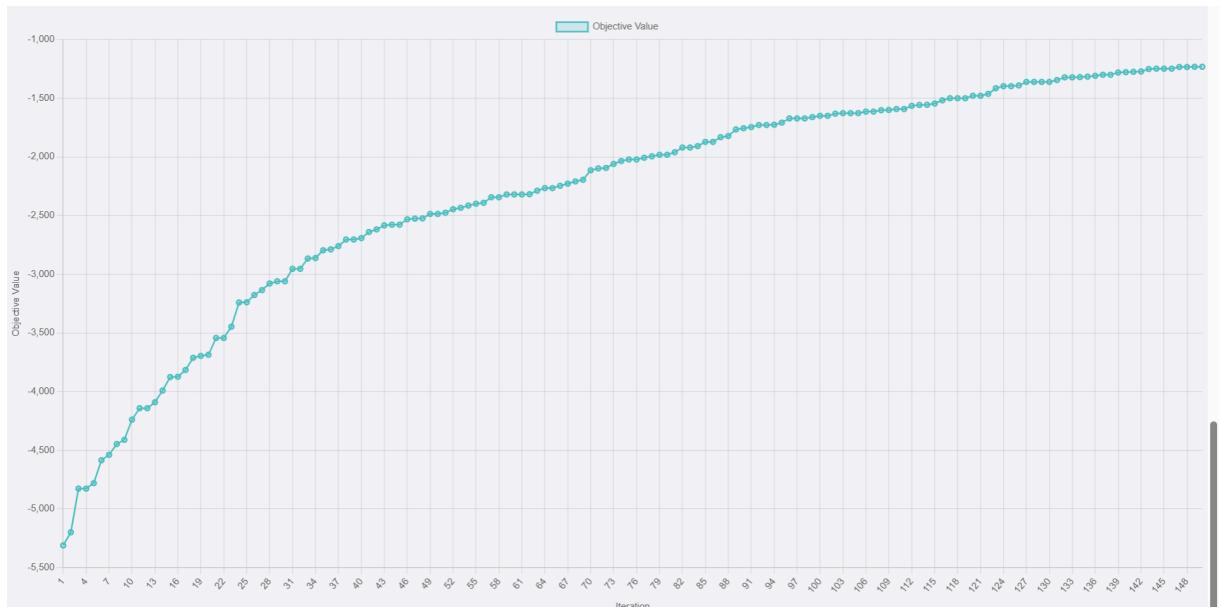
Layer 4:

14	16	121	41	103
79	97	51	82	1
68	9	7	109	120

124	94	88	72	3
29	107	23	18	118

Layer 5:

85	101	66	61	21
37	12	98	87	89
119	62	96	6	34
30	91	28	57	108
43	42	26	104	63



3.2.2.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Found **Genetic Algorithm** solution in **5.46 seconds!**

Final State Objective Value: **-1317**

Iterations needed: **150**

Population Size: **1000**

Layer 1:

74	75	43	80	44
31	98	52	108	30
70	61	50	32	113
33	12	123	35	81
102	84	36	47	45

Layer 2:

26	2	117	29	112
64	101	24	34	92
114	40	20	121	23
58	91	41	119	21
53	77	111	14	66

Layer 3:

86	83	59	68	65
78	5	120	39	79
15	71	56	110	55
124	125	3	11	6
22	10	62	85	116

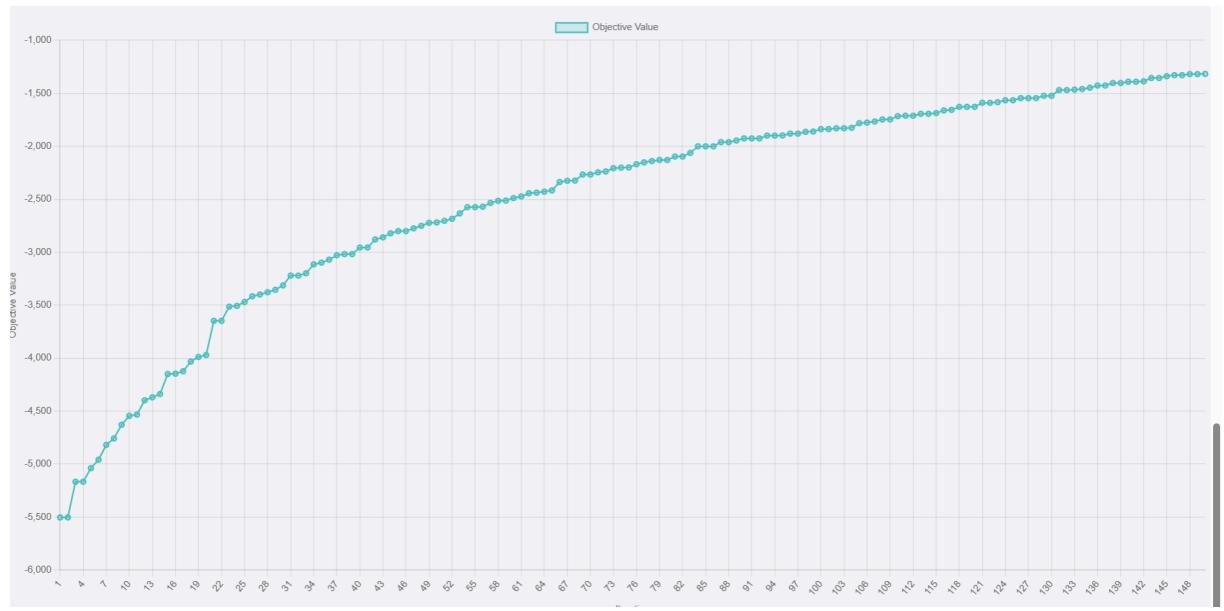
Layer 4:

63	107	17	109	19
73	28	94	25	99
103	13	118	48	38

49	57	90	67	95
96	106	4	69	42

Layer 5:

72	46	87	18	82
93	89	27	105	7
16	122	51	1	115
54	8	60	88	104
76	37	97	100	9



3.1.3.2.3. Variasi 3 Populasi = 2000

3.2.3.1. Percobaan Pertama

TEST 1

Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:

Current Level: 4

Konfigurasi awal kubus:

Layer 1:

5	34	63	69	88
41	18	24	26	89
120	111	52	78	22
49	33	19	92	100
61	113	6	7	10

Layer 2:

96	56	68	62	75
107	123	45	4	99
53	84	3	72	11
97	12	76	20	60
47	14	116	40	94

Layer 3:

70	23	91	110	64
25	109	17	124	1
30	44	79	118	16
43	122	29	80	55
117	112	82	38	35

Layer 4:

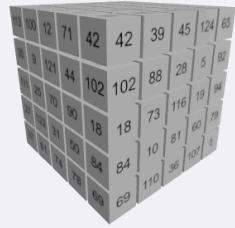
59	8	27	36	108
93	101	48	2	115
66	9	32	81	125
13	98	71	102	31
119	58	21	54	50

Layer 5:

95	104	77	90	86
74	42	28	83	67
121	39	65	37	46
103	73	85	106	57
15	114	51	105	87

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Genetic Algorithm** solution in **15.79 seconds!**

Final State Objective Value: **-1436**

Iterations needed: **150**

Population Size: **2000**

Layer 1:

37	95	13	85	117
27	120	34	26	106
111	58	29	93	32
35	33	119	86	15
113	7	118	30	46

Layer 2:

61	41	75	17	109
122	67	72	43	14
25	38	63	125	59
9	82	21	101	114
100	87	99	2	22

Layer 3:

74	53	105	4	80
31	40	104	115	6
70	52	56	16	96
121	49	57	76	8
12	98	3	103	123

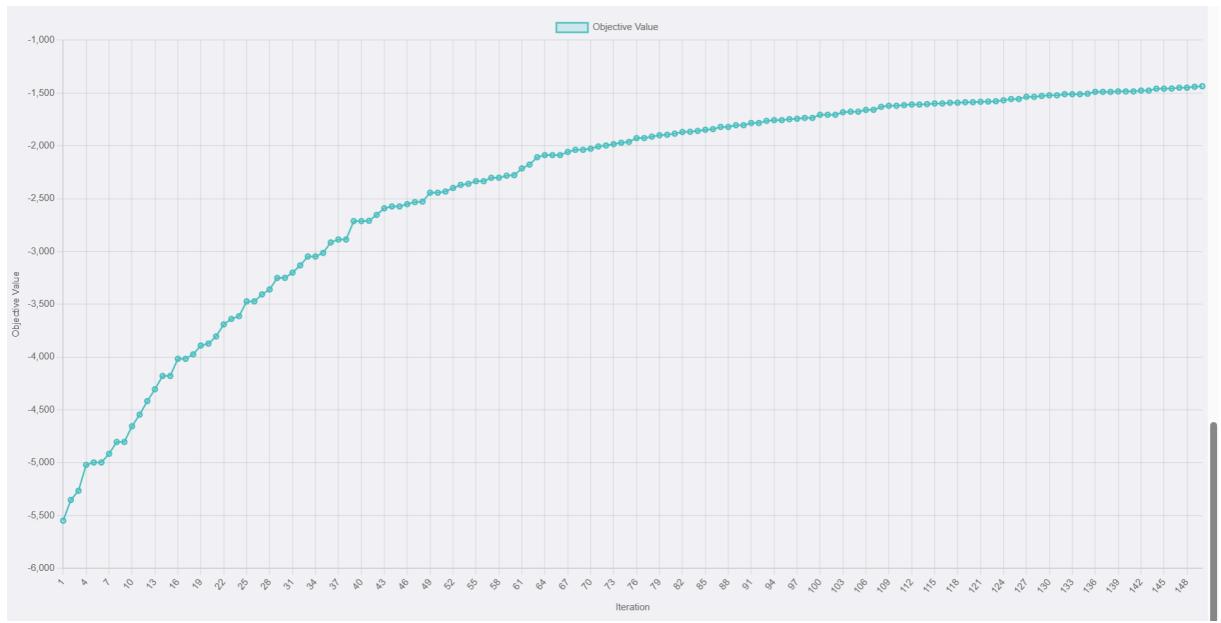
Layer 4:

78	11	97	112	20
50	66	24	62	108
90	89	55	54	23

44	64	91	48	77
71	83	51	47	68

Layer 5:

69	110	36	107	1
84	10	81	60	79
18	73	116	19	94
102	88	28	5	92
42	39	45	124	65



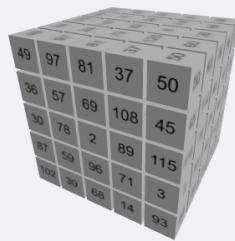
3.2.3.2. Percobaan Kedua

TEST 2

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:

Separate by Y:

Separate by Z:

Select Z Level:
 Current Level: 4

Found **Genetic Algorithm** solution in **15.21 seconds!**

Final State Objective Value: **-1247**

Iterations needed: **150**

Population Size: **2000**

Layer 1:

102 84 27 5 86
87 72 11 33 111
30 90 103 98 1
36 35 52 118 74
49 34 122 64 46

Layer 2:

39 112 19 121 10
59 32 106 66 56
78 18 67 28 119
57 85 7 94 41
97 24 107 9 88

Layer 3:

68 61 12 105 63
96 77 92 43 4
2 91 70 100 76
69 54 79 40 109
81 73 62 29 58

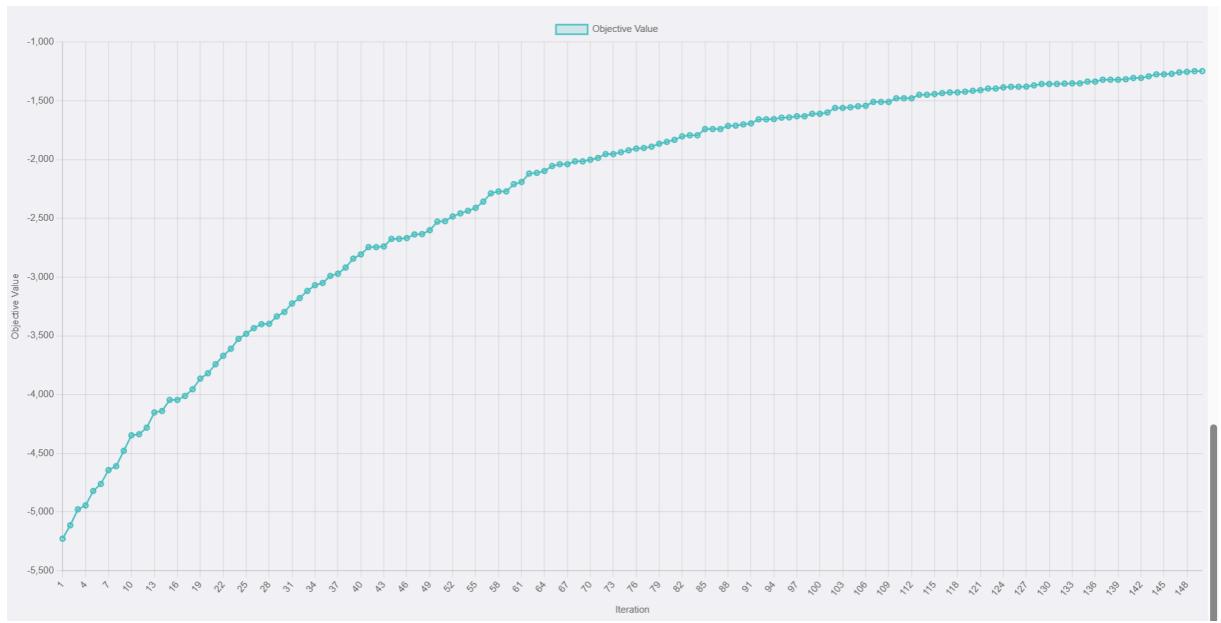
Layer 4:

14 13 123 48 117
71 47 8 82 120
89 101 53 51 22

108	25	113	44	17
37	125	6	116	31

Layer 5:

93	55	104	42	21
3	83	95	110	23
115	15	20	38	124
45	114	65	16	75
50	60	26	99	80



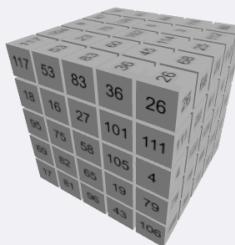
3.2.3.3. Percobaan Ketiga

TEST 3

Sesuai dengan konfigurasi Test 1

HASIL

RESULTS



Drag to pan. Scroll to zoom.

Separate by X:
Separate by Y:
Separate by Z:
Select Z Level:
Current Level: 4

Found **Genetic Algorithm** solution in **15.35 seconds!**

Final State Objective Value: **-1352**

Iterations needed: **150**

Population Size: **2000**

Layer 1:

17	40	109	24	108
69	124	56	50	8
95	31	32	46	123
18	2	110	121	62
117	104	9	67	15

Layer 2:

81	118	22	11	91
82	59	94	38	42
75	29	57	76	92
16	74	71	66	52
53	23	70	125	41

Layer 3:

96	55	72	93	30
65	33	63	48	107
58	49	37	86	87
27	116	34	84	51
83	60	102	5	64

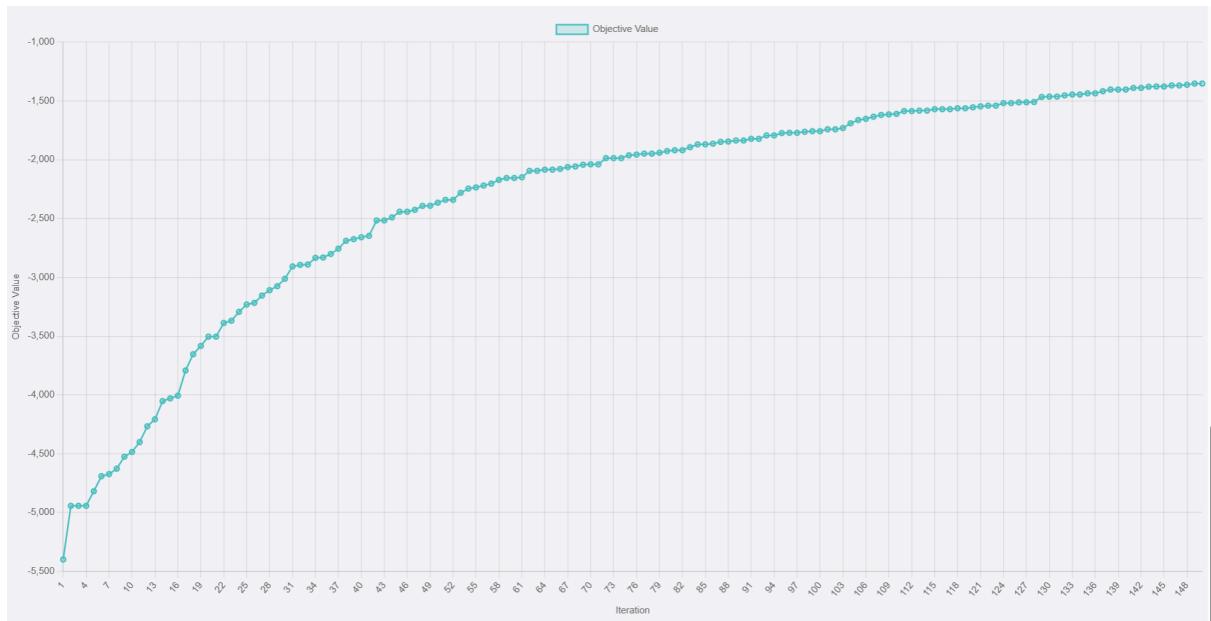
Layer 4:

43	12	97	98	73
19	85	21	77	113
105	78	35	99	1

101	103	39	44	28
36	45	114	3	115

Layer 5:

106	90	14	88	13
79	10	89	100	47
4	122	120	7	54
111	20	61	6	119
26	68	25	112	80



3.2. Analisis Pengujian

3.2.1. Hill Climbing

3.2.1.1. Steepest Ascent Hill Climbing

Steepest Ascent Hill Climbing cenderung mendekati optimum global karena selalu memilih *successor* terbaik sebagai *neighbor* pada setiap langkah. Ini berarti setiap iterasi mendorong solusi semakin dekat ke konfigurasi *Magic Cube* yang hampir sempurna. Karena pendekatan ini hanya mempertimbangkan peningkatan terbaik di antara semua *neighbor*, algoritma mampu mencapai hasil yang sangat dekat dengan solusi optimal.

Jika dibandingkan dengan algoritma *local search* lainnya, *Steepest Ascent* lebih cepat dalam mendekati solusi berkualitas tinggi karena tidak mengulang atau mengandalkan pemilihan acak. Hal ini menjadikan algoritma ini lebih efektif dibandingkan varian seperti *Stochastic Hill Climbing*, yang memilih solusi secara acak dan lebih lambat dalam mendekati optimum.

Dalam hal kecepatan, *Steepest Ascent Hill Climbing* adalah yang tercepat. Karena selalu memilih *neighbor* dengan nilai terbaik, algoritma ini tidak membuang waktu dengan solusi yang tidak meningkatkan hasil. Ini menjadikannya algoritma yang efisien dalam waktu pencarian dibandingkan varian lain yang mungkin membutuhkan lebih banyak iterasi.

Dari segi konsistensi, Steepest Ascent Hill Climbing menunjukkan hasil yang stabil dalam berbagai eksperimen. Karena pilihan neighbor selalu berdasarkan nilai terbaik, solusi yang dihasilkan cenderung serupa di berbagai percobaan, menghasilkan hasil akhir yang konsisten.

3.2.1.2. Hill Climbing with Sideways Move

Hill Climbing with Sideways Move mendekati optimum global dengan baik dan menghasilkan solusi yang hampir sempurna pada *Magic Cube*. Fitur *sideways move* memungkinkan algoritma bergerak horizontal di dataran tinggi, mengatasi hambatan tanpa perlu mundur. Ini membantu algoritma untuk terus maju menuju solusi yang optimal meskipun tidak secepat *Steepest Ascent*.

Jika dibandingkan dengan algoritma lain, *Hill Climbing with Sideways Move* lebih efektif daripada algoritma acak seperti *Stochastic Hill Climbing* dan lebih konsisten daripada *Random Restart*. Tambahan *sideways moves* meningkatkan peluang algoritma untuk keluar dari dataran tinggi tanpa memulai ulang pencarian, sehingga lebih efisien dalam menemukan solusi berkualitas.

Dari segi kecepatan, *Hill Climbing with Sideways Move* hampir secepat *Steepest Ascent Hill Climbing*. Namun, jika dataran tinggi luas, *sideways moves* dapat memperpanjang waktu pencarian karena ada langkah horizontal yang dilakukan, meski tetap lebih cepat dari *Random Restart* yang harus memulai ulang dari awal.

Hasil yang dihasilkan oleh *Hill Climbing with Sideways Move* cukup konsisten di berbagai percobaan. Karena algoritma tetap memilih *successor* terbaik atau yang sama nilainya, hasil akhirnya tetap stabil. Ini menjadikannya algoritma yang andal untuk mendapatkan hasil yang serupa di setiap eksperimen.

3.2.1.3. Random Restart Hill Climbing

Random Restart Hill Climbing mampu mendekati optimum global dengan baik karena setiap kali menemui puncak lokal, algoritma akan memulai ulang dari titik acak yang baru. Dengan cara ini, algoritma memiliki peluang untuk mengeksplorasi lebih banyak area solusi, sehingga peluang mencapai solusi global optimal meningkat meskipun waktu komputasinya lebih panjang.

Dibandingkan dengan algoritma lain, *Random Restart* lebih efektif dalam menemukan solusi optimal karena ia mengatasi jebakan optimum lokal dengan memulai ulang pencarian. Dalam kasus seperti *Magic Cube*, yang memiliki banyak titik optimum lokal, metode ini lebih unggul dalam menghasilkan solusi yang optimal secara konsisten.

Dalam hal durasi, *Random Restart Hill Climbing* adalah yang paling lambat. Setiap kali algoritma mencapai titik buntu atau optimum lokal, pencarian dimulai dari awal, yang memakan waktu. Karena algoritma ini mengulangi proses dari banyak titik acak, waktu pencarian totalnya cenderung lebih panjang daripada varian lainnya.

Meskipun prosesnya lama, *Random Restart Hill Climbing* sangat konsisten dalam menghasilkan solusi berkualitas tinggi. Karena algoritma ini selalu melakukan pencarian ulang dari posisi baru, kemungkinan menemukan solusi optimal lebih tinggi, dan hasil akhirnya lebih stabil di antara berbagai eksperimen.

3.2.1.4. Stochastic Hill Climbing

Stochastic Hill Climbing kurang efektif dalam mendekati optimum global dibandingkan varian lain karena algoritma ini memilih *successor* secara acak, bukan berdasarkan nilai terbaik. Ini berarti pada setiap langkah, tidak ada jaminan bahwa *successor* yang terpilih akan membawa solusi lebih dekat ke optimum, sehingga solusi akhir biasanya tidak mendekati konfigurasi *Magic Cube* yang sempurna.

Dibandingkan dengan algoritma lain, Stochastic Hill Climbing lebih lambat dalam menemukan solusi berkualitas tinggi. Karena pilihan *successor* bersifat acak, hasil akhirnya cenderung berada jauh dari optimum dibandingkan Steepest Ascent atau Random Restart, yang memiliki strategi pencarian lebih terarah.

Dari segi durasi, Stochastic Hill Climbing lebih lambat daripada Steepest Ascent dan Sideways Move Hill Climbing karena algoritma ini dapat memilih solusi yang kurang optimal, yang memperlambat kemajuan pencarian. Namun, ia tetap lebih cepat daripada Random Restart Hill Climbing, yang mengulangi proses pencarian dari awal jika menemui jalan buntu.

Hasil akhir yang dihasilkan oleh Stochastic Hill Climbing cenderung kurang konsisten. Karena algoritma memilih *neighbor* secara acak, variasi solusi yang dihasilkan cukup tinggi, sehingga hasil akhir bisa berbeda secara signifikan di berbagai eksperimen.

3.2.2. Simulated Annealing

Pada kasus magic cube, algoritma simulated annealing menunjukkan hasil yang mendekati optimum global dengan cukup baik jika dibandingkan dengan algoritma local search lainnya. Algoritma ini berhasil mendekati nilai optimum global dengan kemampuannya yang bisa menerima state *successor* yang lebih buru di awal, namun secara bertahap mengurangi kemungkinan tersebut seiring dengan menurunnya suhu. Pendekatan ini memungkinkan simulated annealing untuk mengeksplorasi lebih banyak ruang solusi dan terhindar dari jebakan optimum lokal, sehingga lebih efektif untuk mendekati state yang merupakan optimum global. Sebagai perbandingan, algoritma hill climbing atau genetic algorithm cenderung lebih mudah terjebak di optimum lokal. Hill climbing biasanya lebih cepat menemukan solusi yang lebih baik namun lebih sering terjebak di optimum lokal, sementara genetic algorithm lebih baik dalam eksplorasi solusi secara keseluruhan, namun memerlukan waktu yang lebih lama untuk mencapai solusi terbaik.

Dari segi durasi pencarian, simulated annealing berada di tengah-tengah. Meskipun lebih lambat dibandingkan dengan hill climbing, yang langsung bergerak maju menuju solusi terbaik yang terdeteksi, simulated annealing secara signifikan lebih cepat dibandingkan dengan genetic algorithm. Durasi simulated annealing dipengaruhi oleh pengaturan suhu awal, tingkat penurunan suhu, serta keberuntungan atas hasil acak *neighbor* state. Jika suhu awalnya tinggi dan penurunannya lambat, waktu komputasi akan lebih panjang tetapi juga memungkinkan eksplorasi state yang lebih optimal. Oleh karena itu, simulated annealing membutuhkan waktu yang bervariasi, tergantung pada pengaturan parameter dan *neighbor* state acak yang dihasilkan.

Dari segi konsistensi, simulated annealing mampu menghasilkan hasil akhir yang lebih stabil dan mendekati nilai optimum global dalam beberapa eksperimen yang berbeda, meskipun hasilnya bisa bervariasi. Hal ini bergantung pada keacakan saat pembangkitan *neighbor* state dan pengaturan parameter di awal, yang dapat menyebabkan hasil berbeda dalam eksperimen terpisah.

Dibandingkan dengan hill climbing, yang memberikan hasil yang cenderung sama pada setiap eksperimen karena pendekatannya yang deterministik, simulated annealing lebih fleksibel. Sementara itu, genetic algorithm juga memiliki konsistensi yang bervariasi karena proses seleksi dan mutasi acak, tetapi lebih lambat mencapai solusi terbaik dibandingkan dengan simulated annealing. Kesimpulannya, simulated annealing menunjukkan keseimbangan yang baik antara kecepatan pencarian, kedekatan dengan solusi optimum global, dan konsistensi hasilnya, membuatnya efektif dalam masalah seperti Magic Cube dimana solusi optimum global sulit dicapai dengan metode local search yang lain.

3.2.3. Genetic Algorithm

Pada kasus Magic Cube, Genetic Algorithm (GA) menunjukkan kemampuan yang baik untuk mendekati solusi optimum global. Hal ini disebabkan oleh pendekatan berbasis populasi, di mana GA memulai pencarian dengan sekelompok individu atau solusi yang dievaluasi berdasarkan fungsi fitness. Setiap individu kemudian melalui seleksi, persilangan (crossover), dan mutasi untuk membentuk generasi baru. Dalam setiap iterasi, individu yang lebih "fit" memiliki peluang lebih besar untuk berkembang, sementara mutasi menambahkan variasi acak. Proses ini memungkinkan GA untuk mengeksplorasi ruang solusi yang luas, menghindari jebakan optimum lokal yang sering dialami algoritma lain seperti hill climbing. Hasil akhirnya, GA mampu menemukan solusi yang mendekati optimum global, meskipun konvergensinya cenderung lebih lambat karena membutuhkan banyak generasi.

Dibandingkan algoritma local search lainnya, GA menunjukkan hasil pencarian yang lebih efektif dalam menjelajahi ruang solusi. Algoritma seperti hill climbing, misalnya, hanya bergerak menuju solusi yang lebih baik pada setiap langkah, yang membuatnya cepat namun rentan terjebak di optimum lokal. Simulated annealing memiliki kemampuan yang lebih baik dalam menghindari optimum lokal dibanding hill climbing karena kemampuannya untuk sementara menerima solusi yang kurang baik, tetapi tetap terbatas pada satu jalur solusi. Sementara itu, GA melakukan pencarian pada banyak solusi dalam setiap generasi. Pendekatan ini menjadikan GA unggul dalam kasus yang memerlukan eksplorasi ruang solusi secara luas, seperti Magic Cube, di mana solusi global sulit ditemukan melalui pendekatan lokal.

Dari segi durasi pencarian, GA cenderung memerlukan waktu komputasi yang lebih lama dibandingkan hill climbing dan simulated annealing. Pada hill climbing, pencarian bergerak maju hanya pada satu solusi, tanpa evaluasi populasi atau generasi baru, sehingga durasinya lebih singkat. Simulated annealing lebih lambat dari hill climbing karena mengizinkan transisi ke solusi yang lebih buruk, tetapi tetap lebih cepat dibanding GA. Waktu komputasi GA diperpanjang oleh ukuran populasi dan jumlah generasi yang besar, di mana setiap generasi baru memerlukan evaluasi dan penciptaan individu melalui seleksi, persilangan, dan mutasi. Durasi pencarian ini sangat bergantung pada pengaturan parameter dan jumlah generasi yang diperlukan untuk mencapai solusi terbaik.

Seberapa konsisten hasil GA dalam eksperimen juga bervariasi karena adanya elemen acak dalam proses mutasi dan seleksi individu untuk persilangan, yang menyebabkan hasil akhirnya bisa berbeda pada setiap percobaan. Dalam pendekatan yang lebih deterministik seperti hill climbing, solusi yang sama cenderung dihasilkan pada setiap percobaan. Simulated annealing mirip dengan GA dalam variasi hasil, meskipun GA cenderung menunjukkan lebih banyak perbedaan antara satu eksperimen dengan yang lain. Meski hasil GA tidak selalu sama dalam

setiap percobaan, fleksibilitasnya dalam menjelajahi ruang solusi dan menemukan solusi yang baik secara konsisten membuatnya berguna dalam masalah optimisasi kompleks.

Terakhir, jumlah iterasi dan ukuran populasi memainkan peran besar dalam kualitas hasil akhir GA. Semakin banyak generasi yang digunakan, semakin lama GA memiliki waktu untuk berevolusi dan mendekati solusi optimal. Populasi yang lebih besar memungkinkan lebih banyak variasi individu dalam satu generasi, memperluas ruang solusi yang dapat dijelajahi. Namun, peningkatan populasi dan generasi secara berlebihan memperpanjang waktu komputasi, sehingga perlu keseimbangan parameter untuk mendapatkan hasil optimal tanpa mengorbankan efisiensi. Jika dirancang dengan parameter yang seimbang, GA sangat efektif untuk mencari solusi global, meskipun memerlukan waktu komputasi yang lebih panjang dibandingkan dengan algoritma local search lainnya.

BAB IV

Kesimpulan dan Saran

5.1. Kesimpulan

Berdasarkan hasil eksperimen yang dilakukan ditemukan fakta bahwa dari segi kedekatan terhadap solusi optimum global, urutan algoritma dari yang terdekat adalah simulated annealing, genetic algorithm, hill climbing with sideways move, steepest ascent hill climbing, random restart hill climbing, dan stochastic hill climbing. Dari segi durasi pencarian, urutan algoritma dari yang tercepat adalah simulated annealing, genetic algorithm, stochastic hill climbing, steepest ascent hill climbing, hill climbing with sideways move, dan random restart hill climbing. Dari segi konsistensi, urutan algoritma dari yang paling konsisten adalah steepest ascent hill climbing, hill climbing with sideways move, simulated annealing, genetic algorithm, random restart hill climbing, dan stochastic hill climbing. Sehingga dari fakta-fakta diatas dapat disimpulkan bahwa algoritma paling optimal untuk menyelesaikan problem Magic Cube ini adalah **Simulated Annealing**.

5.2. Saran

Berdasarkan pengalaman penulis dalam menyelesaikan problem Magic Cube dengan menggunakan Local Search, ada beberapa saran yang penulis sarankan:

1. Gunakan *random float* sebagai pembanding *cooling rate* pada algoritma *simulated annealing*, tujuannya agar nilai *randomness*-nya lebih baik.
2. Gunakan iterasi maksimum untuk algoritma simulated annealing, hal ini bertujuan agar komputer tidak overload dan menyebabkan *lagging* saat menampilkan solusi.

BAB V

Referensi

- [1] Dokumen [!\[\]\(a9dddcb0dc218fcdcf59425eb24f0579_img.jpg\) Tucil1_13522124_13522135](#)
- [2] Dokumen [!\[\]\(ee3a3e123cb8b0b1aba51607bdd2878a_img.jpg\) Tucil1_13522126_13522147](#)
- [3] Tim Pengajar IF3170. 2024/2025. Slide perkuliahan Local Search.
- [4] Tim Pengajar IF3170. 2024/2025. Slide perkuliahan Genetic Algorithm.
- [5] Tim Pengajar IF3170. 2024/2025. Slide perkuliahan Simulated Annealing.
- [6] Tim Pengajar IF3170. 2024/2025. Slide perkuliahan Hill Climbing Algorithm.

Lampiran

Pembagian Kerja Kelompok

NIM	Nama	Pekerjaan
13522124	Aland Mulia Pratama	Frontend Developer
13522126	Rizqika Mulia Pratama	Simulated Annealing Algorithm dan Backend Developer
13522135	Christian Justin Hendrawan	Genetic Algorithm
13522147	Ikhwan Al Hakim	Hill Climbing Algorithm