

# **Tugas Besar 1 IF3070 Dasar Inteligensi Artifisial**

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



**Disusun Oleh:**  
**Yoga Putra Pratama - 18222073**

**Program Studi Sistem dan Teknologi Informasi**  
**Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung**  
**Jl. Ganeshha 10, Bandung 40132**

# Daftar Isi

<b>Daftar Isi.....</b>	<b>1</b>
<b>Deskripsi Persoalan.....</b>	<b>3</b>
<b>Pembahasan.....</b>	<b>3</b>
1. Objective Function.....	3
2. Implementasi Algoritma Local Search.....	4
3. Hasil Eksperimen dan Analisis.....	6
<b>Kesimpulan dan Saran.....</b>	<b>28</b>
<b>Pembagian Tugas.....</b>	<b>29</b>
<b>Referensi.....</b>	<b>30</b>

# 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 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.

Laporan ini berisi tentang implementasi algoritma local search untuk mencari solusi Diagonal Magic Cube dan penyelesaian masalah Diagonal Magic Cube berukuran 5x5x5 dengan initial state dari suatu kubus adalah susunan angka 1 hingga  $5^3$  secara acak. Langkah yang boleh dilakukan untuk tiap iterasi pada algoritma local search adalah menukar posisi dari 2 angka pada kubus tersebut (2 angka yang ditukar tidak harus bersebelahan).

## Pembahasan

### 1. Objective Function

Pemilihan fungsi objektif pada implementasi algoritma local search untuk mencari solusi Diagonal Magic Cube ini dirancang untuk menghitung seberapa jauh konfigurasi saat ini dari kondisi ideal atau solusi yang diharapkan. Fungsi objektif ini berperan penting dalam mengevaluasi setiap konfigurasi kubus, karena hasilnya menjadi ukuran yang digunakan untuk membandingkan dan memilih konfigurasi terbaik pada tiap iterasi algoritma. Pada kasus Diagonal Magic Cube, nilai ideal untuk setiap baris, kolom, tiang, dan diagonal pada kubus harus berjumlah sama dengan magic number yang telah dihitung terlebih dahulu.

Di dalam kode, fungsi objektif dalam kelas `DiagonalMagicCube` bertugas menghitung selisih antara jumlah setiap baris, kolom, tiang, dan diagonal yang ada pada kubus dengan magic number. Selisih ini diakumulasi sebagai nilai penyimpangan (deviation) yang merepresentasikan seberapa jauh kondisi kubus dari solusi yang diharapkan. Semakin besar nilai deviation, semakin besar pula perbedaan antara kondisi kubus saat ini dan kondisi solusi, yang artinya kubus jauh dari susunan yang memenuhi syarat sebagai magic cube. Sebaliknya, semakin kecil nilai deviation, semakin dekat kubus ke solusi.

Untuk menghitung nilai deviation, fungsi objektif pertama-tama memeriksa setiap baris, kolom, dan tiang, serta menghitung selisih nilai dari hasil penjumlahan angka pada baris, kolom, dan tiang tersebut dengan magic number. Selain itu, fungsi objektif juga menghitung

selisih nilai pada setiap diagonal di tiap bidang (3x3) dari kubus dan pada diagonal ruang utama dari kubus (main space diagonals). Penyimpangan atau deviation dari masing-masing elemen ini kemudian dijumlahkan untuk mendapatkan nilai total deviation yang menunjukkan seberapa jauh konfigurasi saat ini dari kondisi ideal.

Dengan menggunakan nilai deviation sebagai fungsi objektif, algoritma local search dapat mengevaluasi konfigurasi kubus pada setiap langkah iterasi, baik itu dengan metode Hill Climbing, Simulated Annealing, atau Genetic Algorithm. Pemilihan ini memungkinkan algoritma untuk mencari solusi optimal secara efisien karena fungsi objektif ini cukup sederhana untuk dihitung namun mencakup semua syarat penting dari Diagonal Magic Cube. Nilai deviation yang rendah menunjukkan konfigurasi yang lebih mendekati magic cube, dan ini menjadi indikasi bahwa algoritma semakin dekat pada solusi yang diinginkan.

## 2. Implementasi Algoritma Local Search

Implementasi algoritma local search ini bertujuan untuk menemukan solusi terbaik dari masalah Diagonal Magic Cube berukuran  $5 \times 5 \times 5$ . Pada kubus ini, angka-angka 1 hingga  $n^3$  ditempatkan tanpa pengulangan sehingga jumlah angka di setiap baris, kolom, tiang, dan diagonal sama dengan magic number. Algoritma ini menggunakan tiga pendekatan local search, yaitu Hill Climbing, Simulated Annealing, dan Genetic Algorithm, untuk menemukan susunan yang memenuhi persyaratan magic cube. Berikut penjelasan dari setiap kelas dan fungsi yang digunakan:

### 1) Fungsi `run_experiment()`

Fungsi ini menjalankan eksperimen untuk algoritma yang diberikan. Parameter utamanya:

- `algorithm_class`: Kelas algoritma yang akan dijalankan (seperti `HillClimbing`, `SimulatedAnnealing`, atau `GeneticAlgorithm`).
- `name`: Nama algoritma untuk digunakan sebagai label selama eksperimen.
- `runs`: Jumlah percobaan yang akan dijalankan.
- `population_size` dan `generations`: Parameter khusus untuk Genetic Algorithm.

Dalam setiap percobaan, fungsi:

1. Menginisialisasi keadaan awal magic cube dan memvisualisasikannya menggunakan `Visualizer`.
2. Menjalankan algoritma yang dipilih dan melacak nilai fungsi objektif serta waktu eksekusi.
3. Menyimpan hasil akhir dari setiap percobaan.
4. Memvisualisasikan perkembangan nilai fungsi objektif selama iterasi.

### 2) Fungsi `run_genetic_algorithm_experiment()`

Fungsi ini mengatur eksperimen untuk Genetic Algorithm, menguji berbagai ukuran populasi dan jumlah generasi.

### 3) Fungsi `main`

Fungsi utama yang memanggil semua eksperimen dan menjalankan algoritma, mencatat hasil dari setiap percobaan, serta menampilkan hasilnya.

### 4) Kelas `HillClimbing`

Kelas ini berisi:

- `__init__(self, cube)`: Konstruktor yang menerima objek `DiagonalMagicCube` untuk diproses.
- `steepest_ascent(self, max_iterations=1000, record=None)`: Algoritma Hill Climbing dengan metode steepest ascent. Dalam setiap iterasi:
  1. Menentukan skor fungsi objektif.
  2. Menghasilkan tetangga (dengan menukar dua angka dalam kubus) dan memilih tetangga dengan skor terbaik.
  3. Jika tidak ada perbaikan, proses berhenti.
- `random_position(self)`: Menghasilkan posisi acak dalam kubus untuk melakukan pertukaran.

### 5) Kelas `SimulatedAnnealing`

Kelas ini berisi:

- `__init__(self, cube)`: Konstruktor yang menerima objek `DiagonalMagicCube`.
- `anneal(self, initial_temperature=1000, cooling_rate=0.95, max_iterations=1000, record=None)`: Implementasi algoritma Simulated Annealing.
  - Menghitung probabilitas penerimaan untuk solusi yang lebih buruk berdasarkan suhu dan perubahan energi.
  - Memperbarui suhu setiap iterasi sesuai dengan cooling rate.

### 6) Kelas `GeneticAlgorithm`

Kelas ini berisi:

- `__init__(self, cube)`: Konstruktor yang menerima objek `DiagonalMagicCube`.
- `run(self, population_size=50, generations=100, record=None)`: Metode untuk menjalankan algoritma genetika.
  - Menginisialisasi populasi awal, melakukan seleksi, crossover, dan mutation.
- `evaluate(self, cube)`: Mengevaluasi solusi berdasarkan fungsi objektif.
- `crossover(self, parent1, parent2)`: Melakukan crossover antara dua individu untuk menghasilkan keturunan.
- `mutate(self, cube, mutation_rate=0.01)`: Melakukan mutasi dengan menukar dua angka dalam kubus.

### 7) Kelas `DiagonalMagicCube`

Kelas yang mengatur representasi dari kubus dan fungsi objektif.

- `__init__(self, size=5)`: Menginisialisasi kubus dengan ukuran tertentu.
- `generate_initial_cube(self)`: Menghasilkan susunan awal angka secara acak dalam kubus.
- `calculate_magic_number(self)`: Menghitung magic number dari kubus.
- `swap_numbers(self, pos1, pos2)`: Menukar dua angka dalam kubus.
- `objective_function(self)`: Menghitung deviasi dari sum yang diharapkan pada setiap baris, kolom, dan diagonal.

## 8) Kelas `Visualizer`

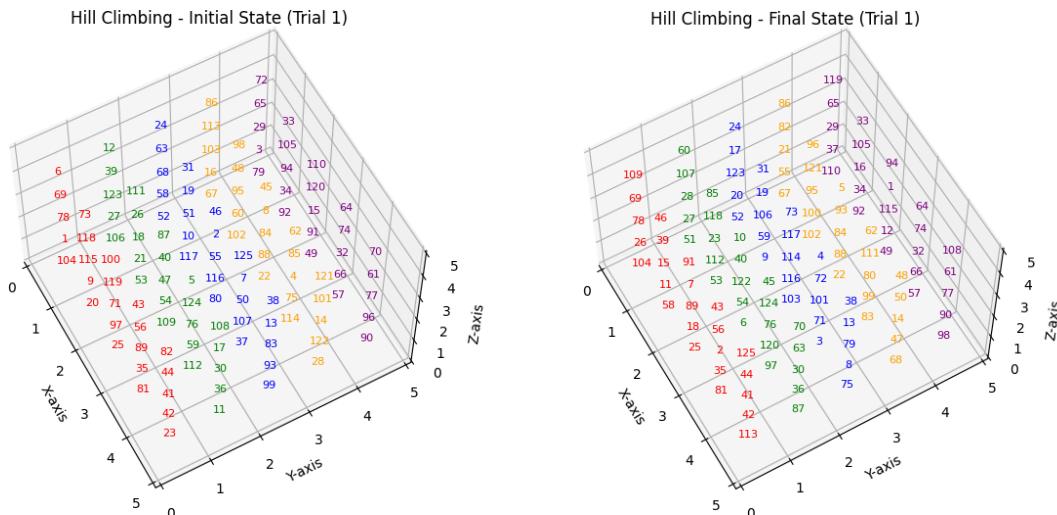
Kelas untuk memvisualisasikan keadaan kubus.

- `__init__(self, cube)`: Konstruktor yang menerima objek `DiagonalMagicCube`.
- `plot_cube(self, title="Magic Cube State")`: Memvisualisasikan kubus dalam grafik 3D.

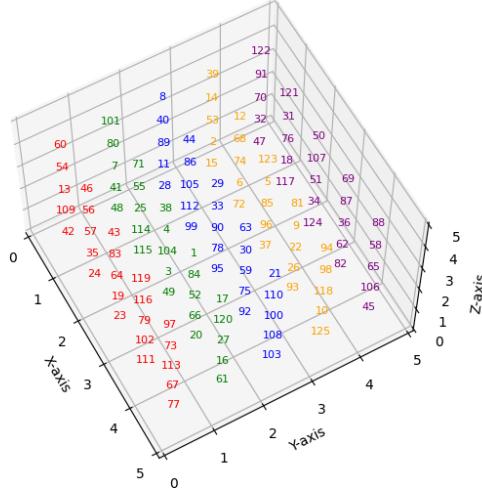
## 3. Hasil Eksperimen dan Analisis

### 1) Hasil Eksperimen

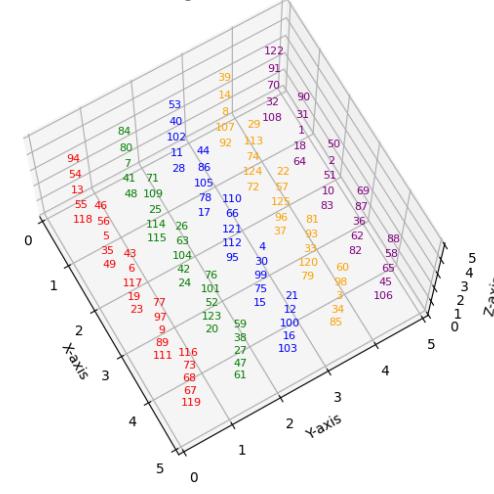
#### I. Steepest Ascent Hill-Climbing



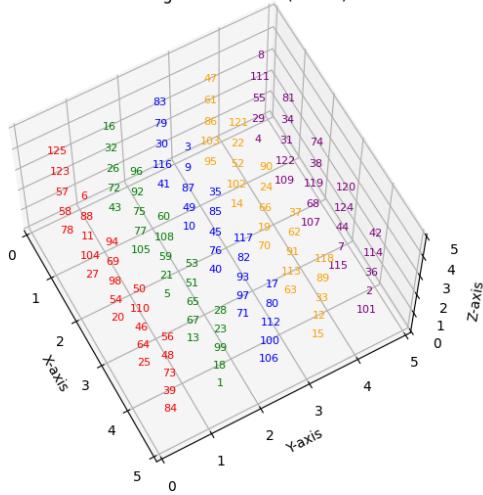
Hill Climbing - Initial State (Trial 2)



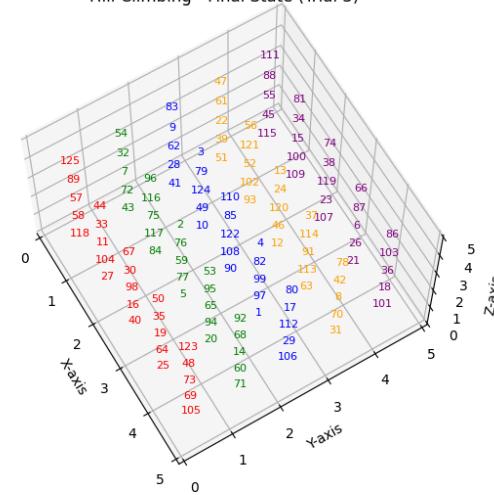
Hill Climbing - Final State (Trial 2)



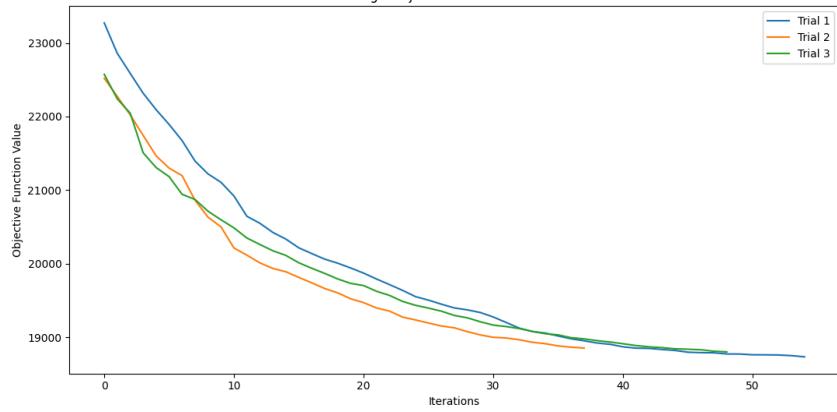
Hill Climbing - Initial State (Trial 3)



Hill Climbing - Final State (Trial 3)



Hill Climbing - Objective Function Over Iterations

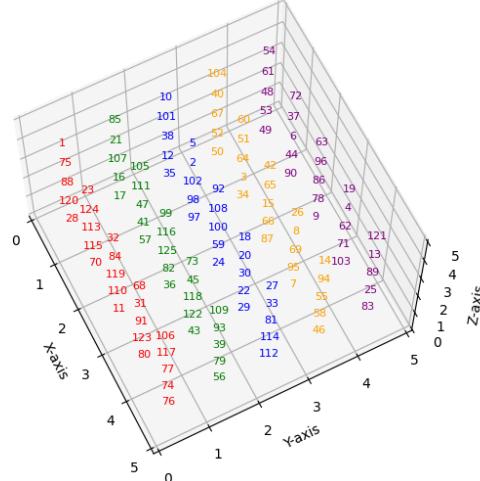


Output program:

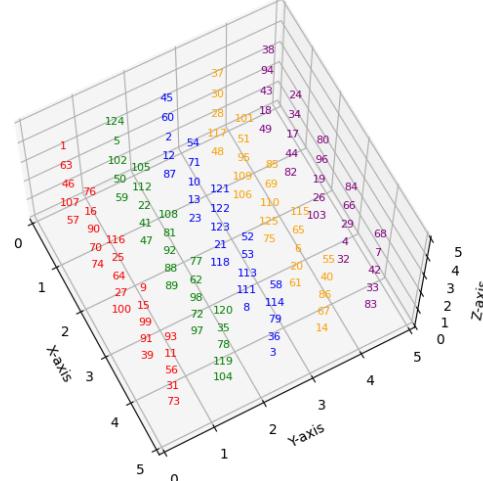
```
Running Hill Climbing, Trial 1...
Final Objective: 18736, Duration: 1.50 seconds
Total Iterations: 55
Running Hill Climbing, Trial 2...
Final Objective: 18855, Duration: 1.04 seconds
Total Iterations: 38
Running Hill Climbing, Trial 3...
Final Objective: 18803, Duration: 1.32 seconds
Total Iterations: 49
```

## II. Simulated Annealing

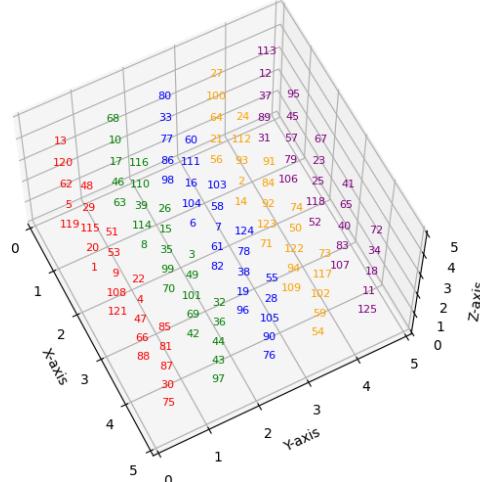
Simulated Annealing - Initial State (Trial 1)



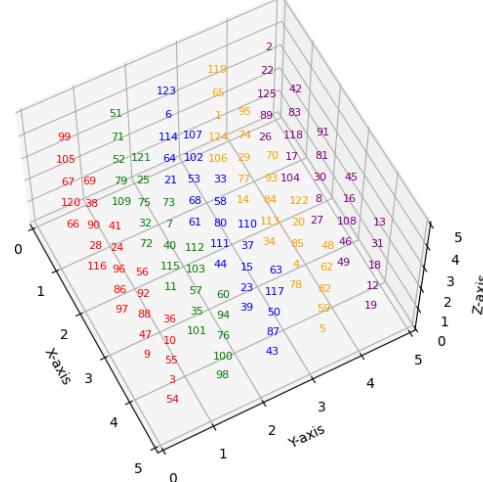
Simulated Annealing - Final State (Trial 1)



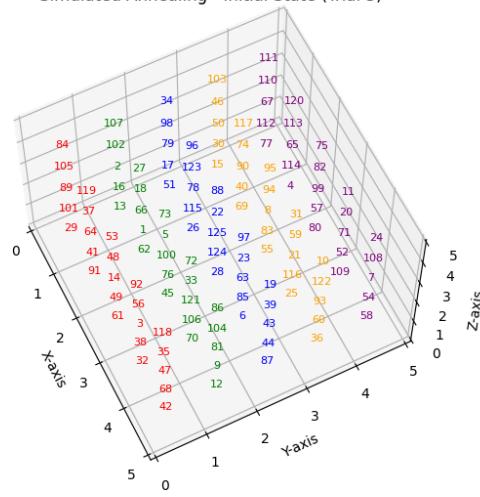
Simulated Annealing - Initial State (Trial 2)



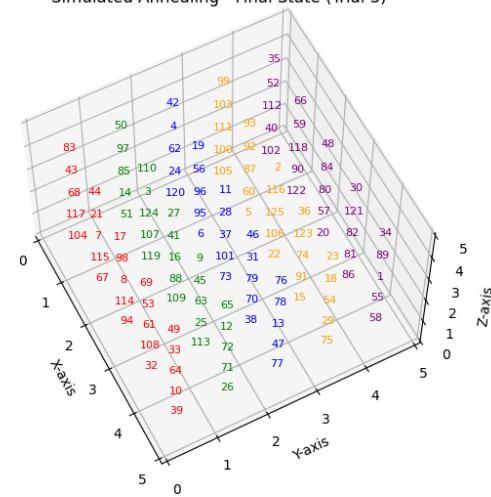
Simulated Annealing - Final State (Trial 2)



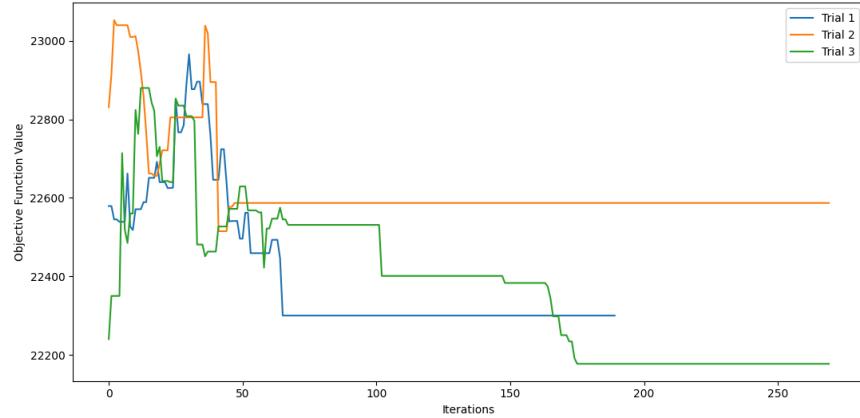
Simulated Annealing - Initial State (Trial 3)



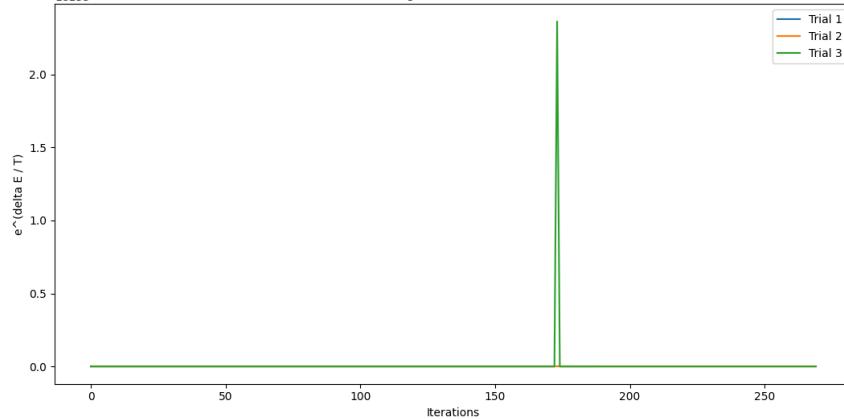
Simulated Annealing - Final State (Trial 3)

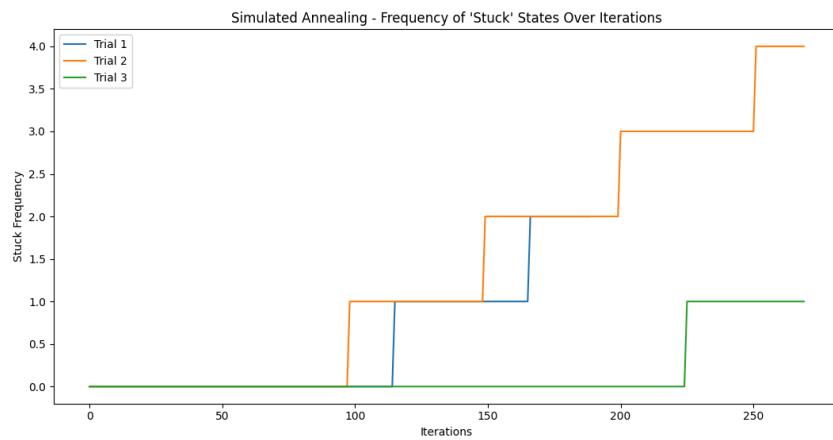


Simulated Annealing - Objective Function Over Iterations



Simulated Annealing -  $e^{(\delta E / T)}$  Over Iterations





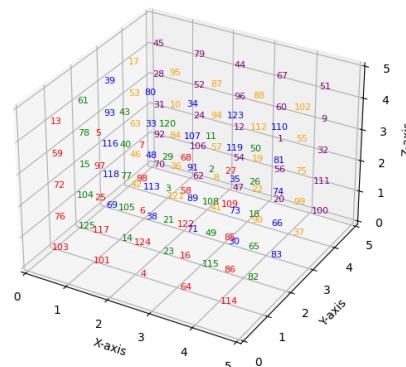
Output program:

```
Running Simulated Annealing, Trial 1...
Overflow encountered in calculation at iteration 189.
Final Objective: 22248, Duration: 0.05 seconds
Running Simulated Annealing, Trial 2...
Final Objective: 23815, Duration: 0.10 seconds
Running Simulated Annealing, Trial 3...
Final Objective: 22915, Duration: 0.08 seconds
```

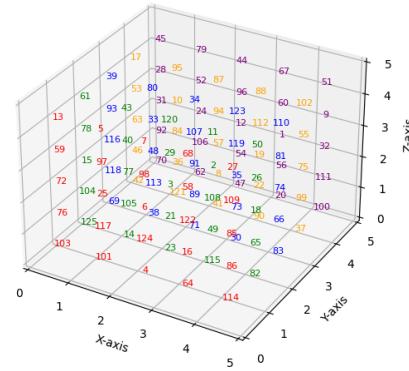
### III. Genetic Algorithm

1. Jumlah populasi sebagai kontrol (Populasi = 100)
  - 1.1. Variasi 50 Generasi

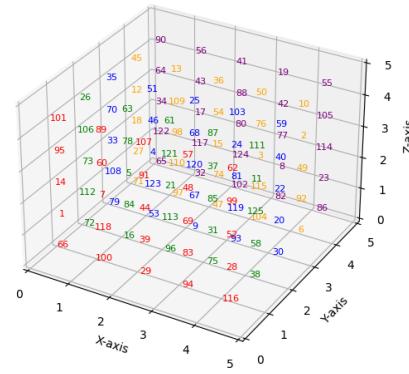
Genetic Algorithm - Initial State (Trial 1)



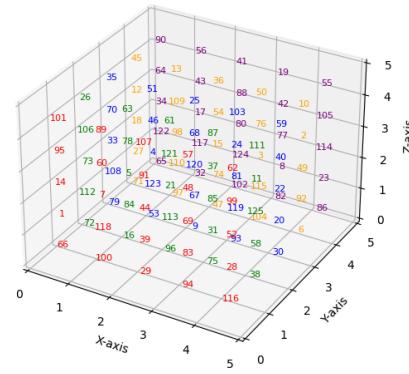
Genetic Algorithm - Final State (Trial 1)



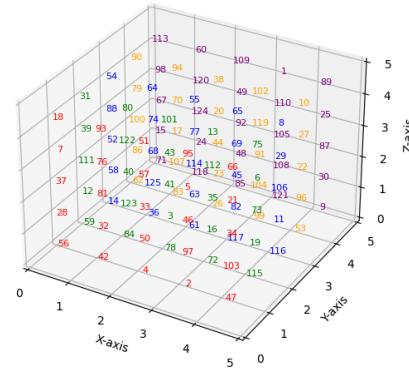
Genetic Algorithm - Initial State (Trial 2)



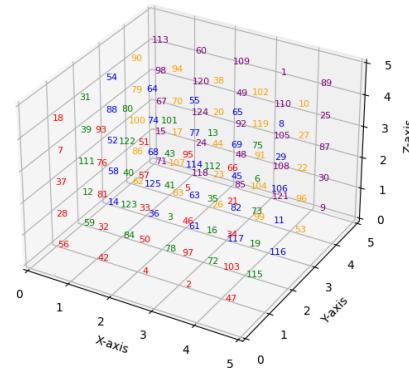
Genetic Algorithm - Final State (Trial 2)



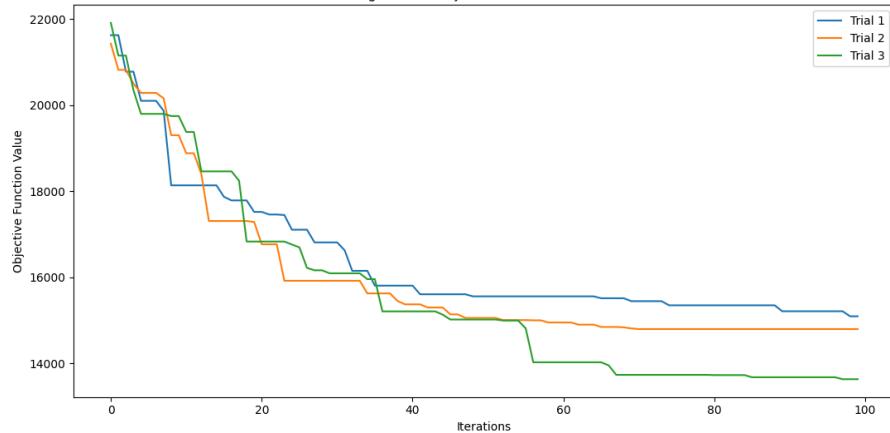
Genetic Algorithm - Initial State (Trial 3)



Genetic Algorithm - Final State (Trial 3)



Genetic Algorithm - Objective Function Over Iterations

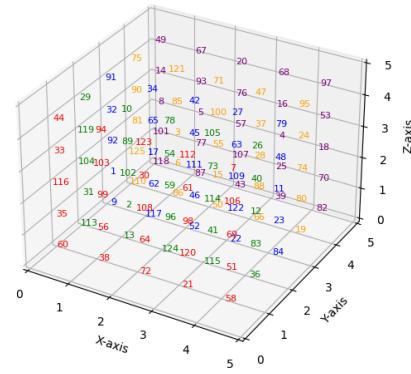


Output program:

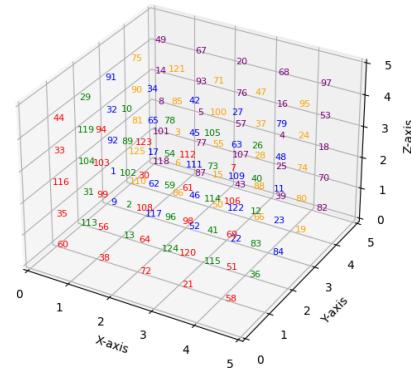
```
Running Genetic Algorithm with Population Size 100 and Generations 50
Running Genetic Algorithm, Trial 1...
Final Objective: 23172, Duration: 3.12 seconds
Running Genetic Algorithm, Trial 2...
Final Objective: 21942, Duration: 3.12 seconds
Running Genetic Algorithm, Trial 3...
Final Objective: 21813, Duration: 3.12 seconds
Experiment with Population Size 100 and Generations 50 completed.
```

## 1.2. Variasi 100 Generasi

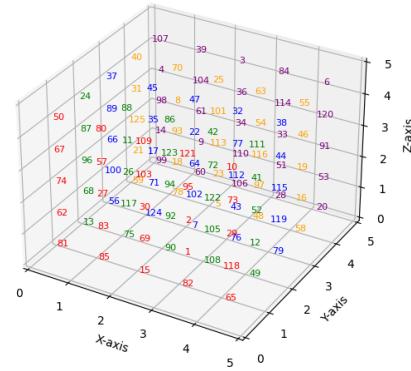
Genetic Algorithm - Initial State (Trial 1)



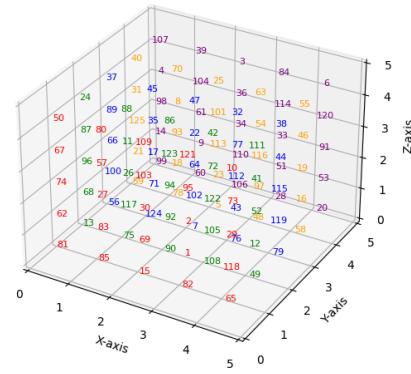
Genetic Algorithm - Final State (Trial 1)



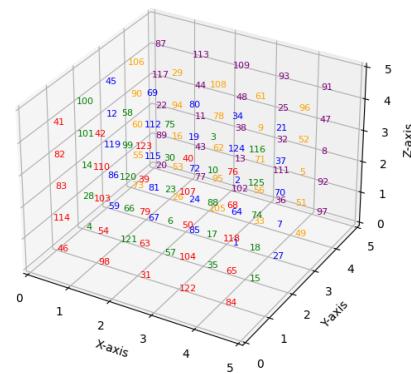
Genetic Algorithm - Initial State (Trial 2)



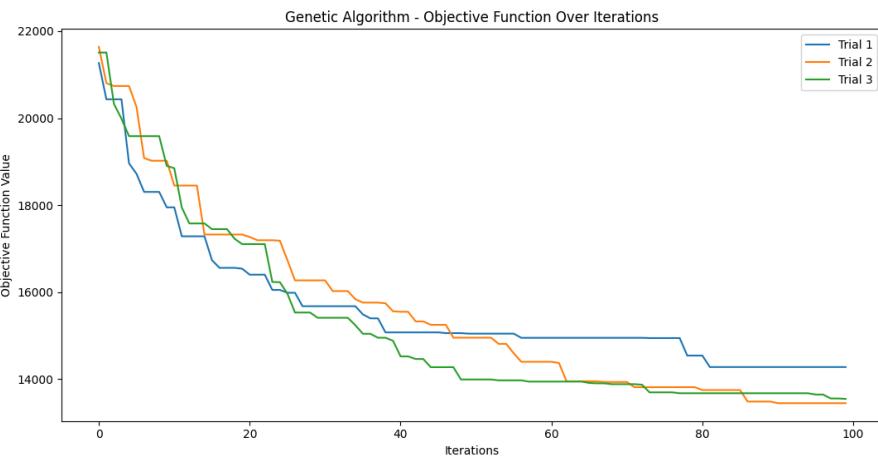
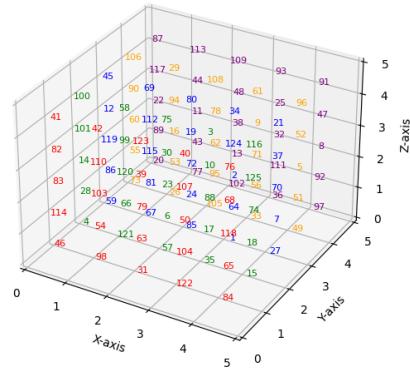
Genetic Algorithm - Final State (Trial 2)



Genetic Algorithm - Initial State (Trial 3)



Genetic Algorithm - Final State (Trial 3)



### Output program:

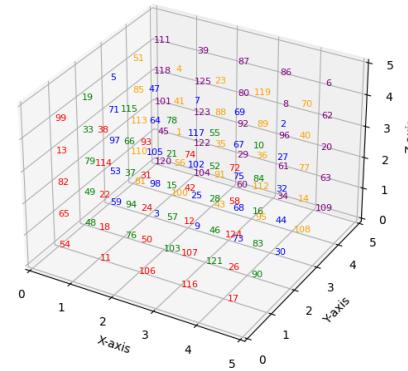
```

Running Genetic Algorithm with Population Size 100 and Generations 100
Running Genetic Algorithm, Trial 1...
Final Objective: 22606, Duration: 3.12 seconds
Running Genetic Algorithm, Trial 2...
Final Objective: 23198, Duration: 3.01 seconds
Running Genetic Algorithm, Trial 3...
Final Objective: 22693, Duration: 3.12 seconds
Experiment with Population Size 100 and Generations 100 completed.

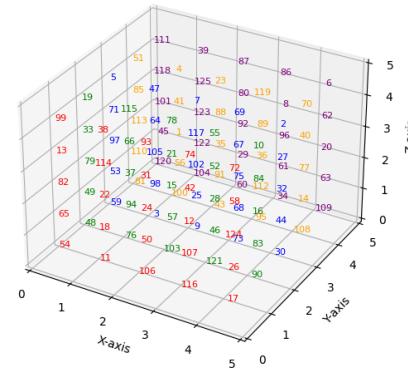
```

### 1.3. Variasi 200 Generasi

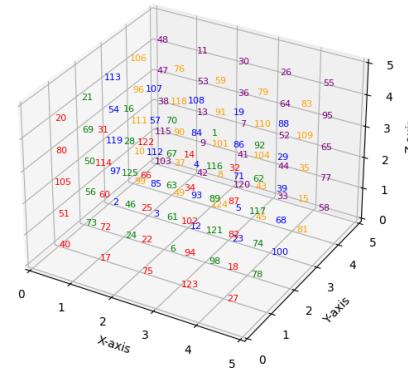
Genetic Algorithm - Initial State (Trial 1)



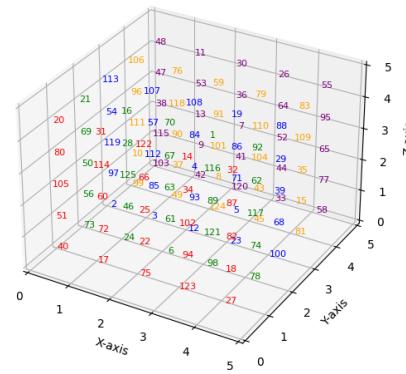
Genetic Algorithm - Final State (Trial 1)



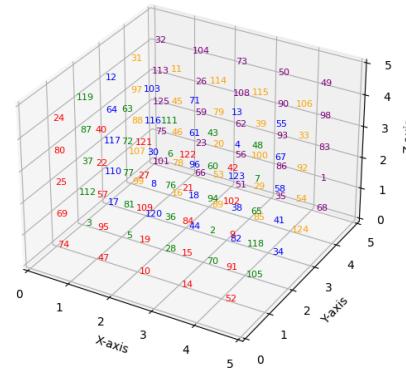
Genetic Algorithm - Initial State (Trial 2)



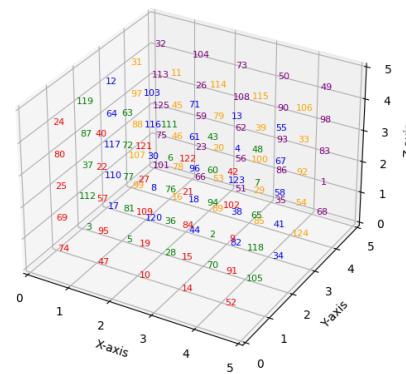
Genetic Algorithm - Final State (Trial 2)

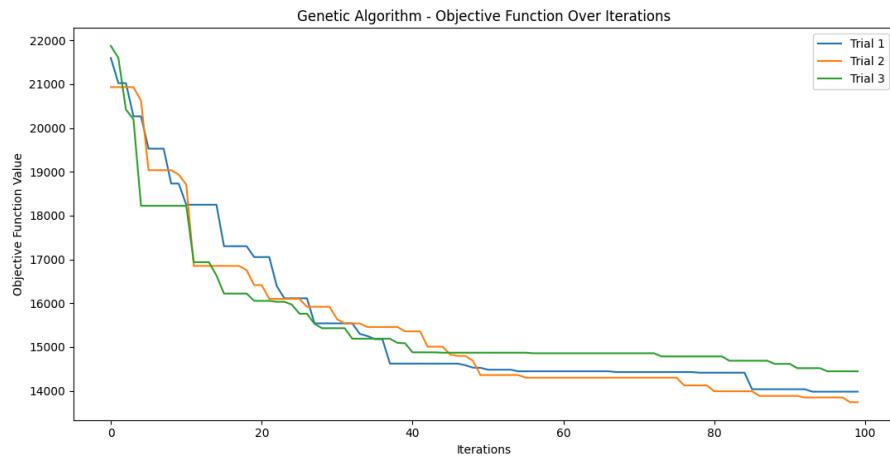


Genetic Algorithm - Initial State (Trial 3)



Genetic Algorithm - Final State (Trial 3)





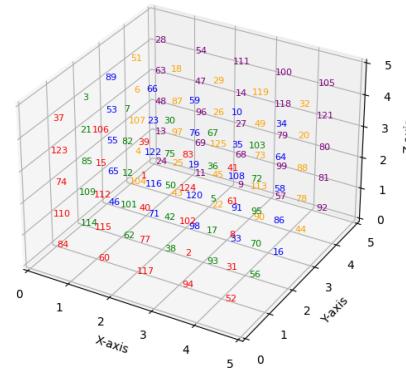
### Output program:

```
Running Genetic Algorithm with Population Size 100 and Generations 200
Running Genetic Algorithm, Trial 1...
Final Objective: 22104, Duration: 3.04 seconds
Running Genetic Algorithm, Trial 2...
Final Objective: 23213, Duration: 3.11 seconds
Running Genetic Algorithm, Trial 3...
Final Objective: 23255, Duration: 3.14 seconds
Experiment with Population Size 100 and Generations 200 completed.
```

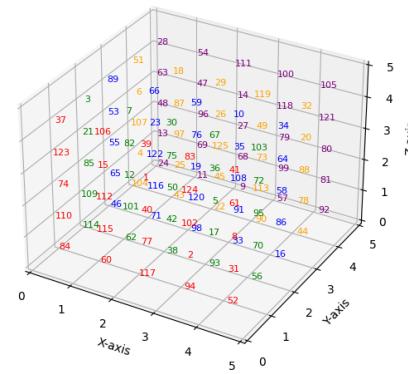
## 2. Banyak iterasi sebagai kontrol (Generasi = 100)

### 2.1. Variasi 50 Populasi

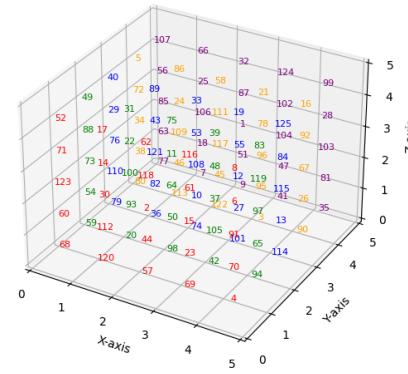
Genetic Algorithm - Initial State (Trial 1)



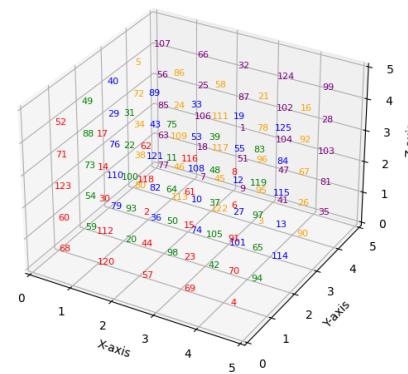
Genetic Algorithm - Final State (Trial 1)



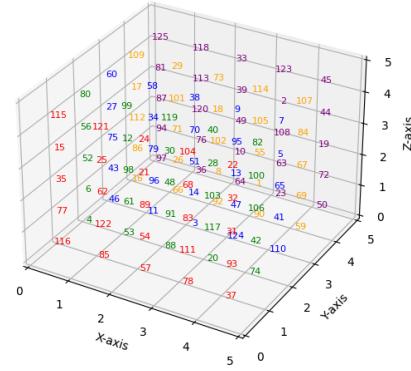
Genetic Algorithm - Initial State (Trial 2)



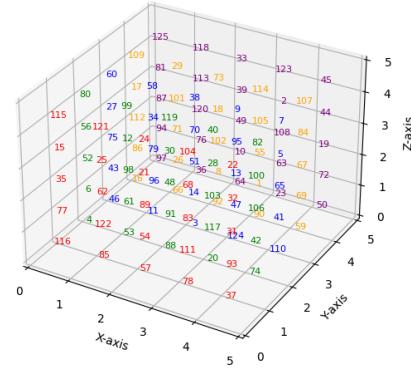
Genetic Algorithm - Final State (Trial 2)



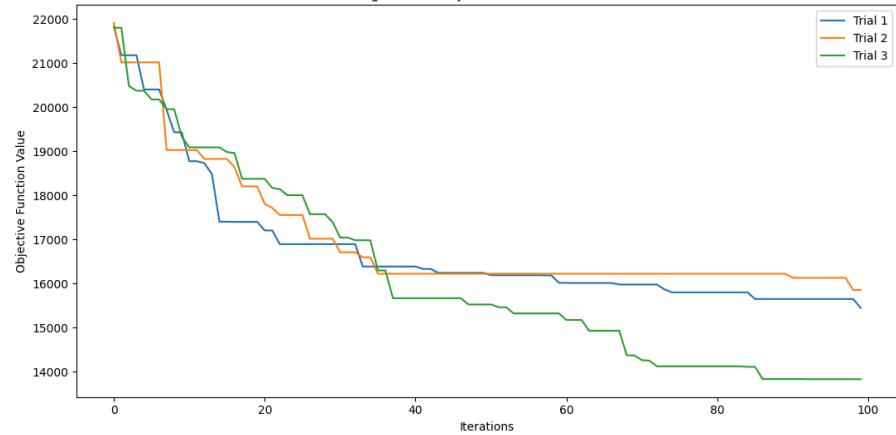
Genetic Algorithm - Initial State (Trial 3)



Genetic Algorithm - Final State (Trial 3)



Genetic Algorithm - Objective Function Over Iterations

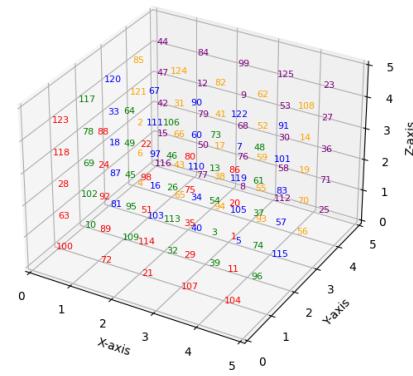


Output program:

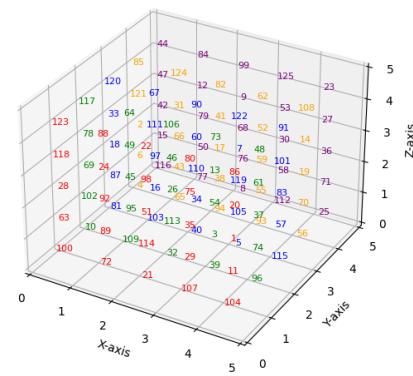
```
Running Genetic Algorithm with Population Size 50 and Generations 100
Running Genetic Algorithm, Trial 1...
Final Objective: 23188, Duration: 3.06 seconds
Running Genetic Algorithm, Trial 2...
Final Objective: 22862, Duration: 3.03 seconds
Running Genetic Algorithm, Trial 3...
Final Objective: 22971, Duration: 3.08 seconds
Experiment with Population Size 50 and Generations 100 completed.
```

## 2.2. Variasi 100 Populasi

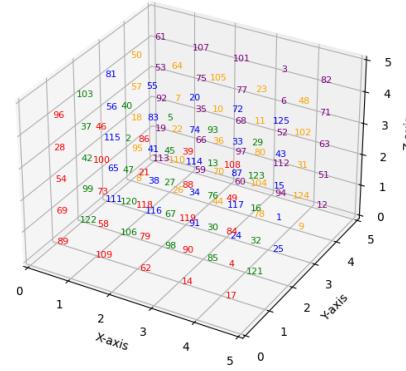
Genetic Algorithm - Initial State (Trial 1)



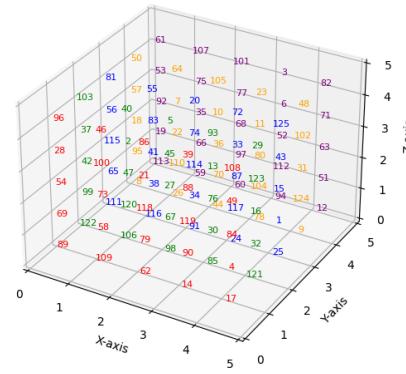
Genetic Algorithm - Final State (Trial 1)



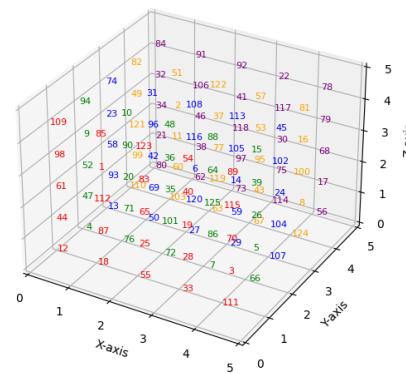
Genetic Algorithm - Initial State (Trial 2)



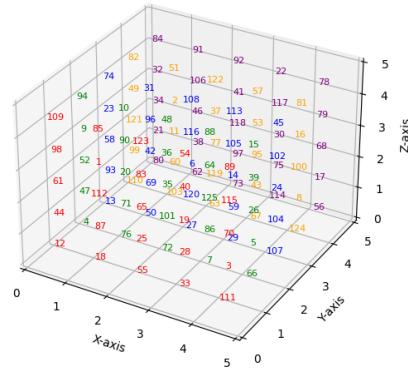
Genetic Algorithm - Final State (Trial 2)



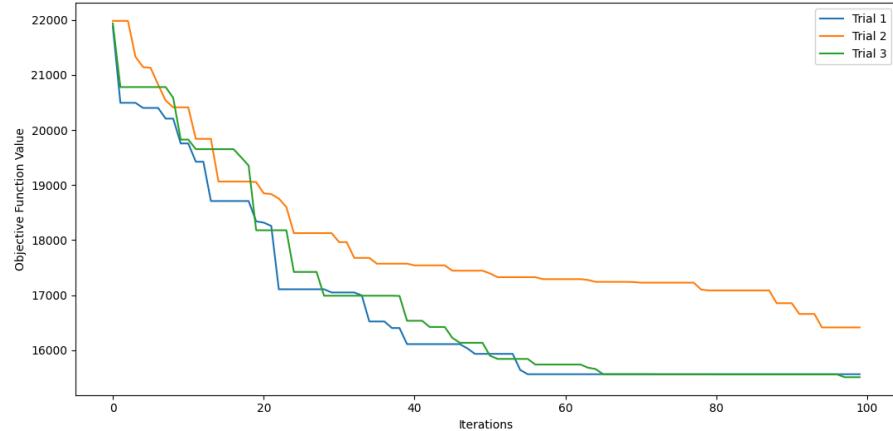
Genetic Algorithm - Initial State (Trial 3)



Genetic Algorithm - Final State (Trial 3)



Genetic Algorithm - Objective Function Over Iterations



### Output program:

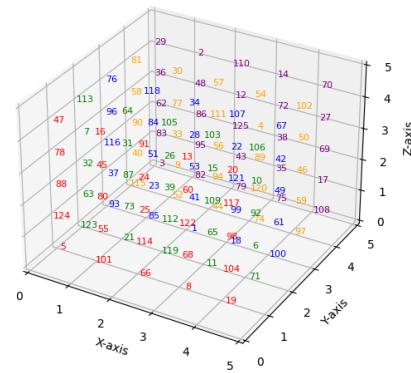
```

Running Genetic Algorithm with Population Size 100 and Generations 100
Running Genetic Algorithm, Trial 1...
Final Objective: 22767, Duration: 3.08 seconds
Running Genetic Algorithm, Trial 2...
Final Objective: 22406, Duration: 3.12 seconds
Running Genetic Algorithm, Trial 3...
Final Objective: 23205, Duration: 3.06 seconds
Experiment with Population Size 100 and Generations 100 completed.

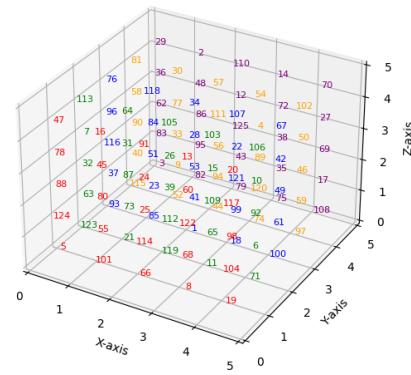
```

### 2.3. Variasi 200 Populasi

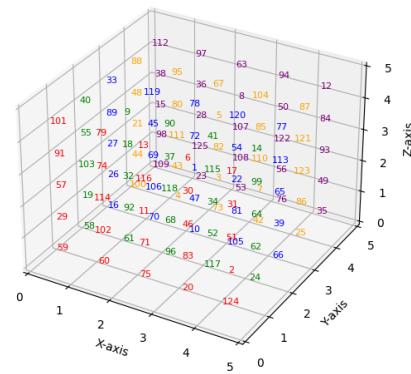
Genetic Algorithm - Initial State (Trial 1)



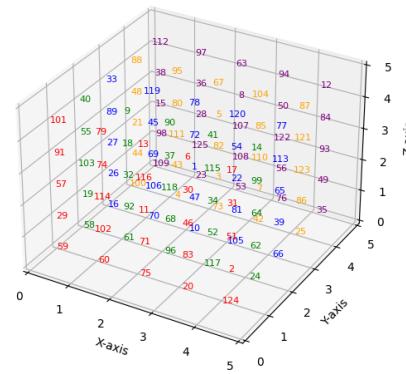
Genetic Algorithm - Final State (Trial 1)



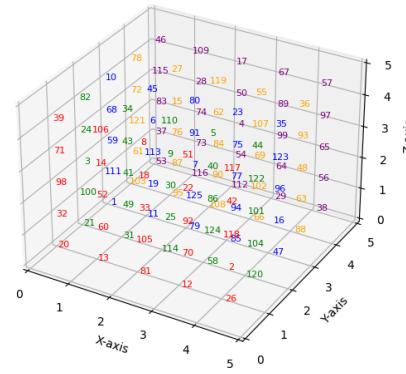
Genetic Algorithm - Initial State (Trial 2)



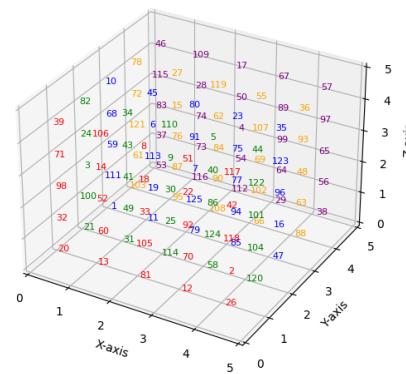
Genetic Algorithm - Final State (Trial 2)

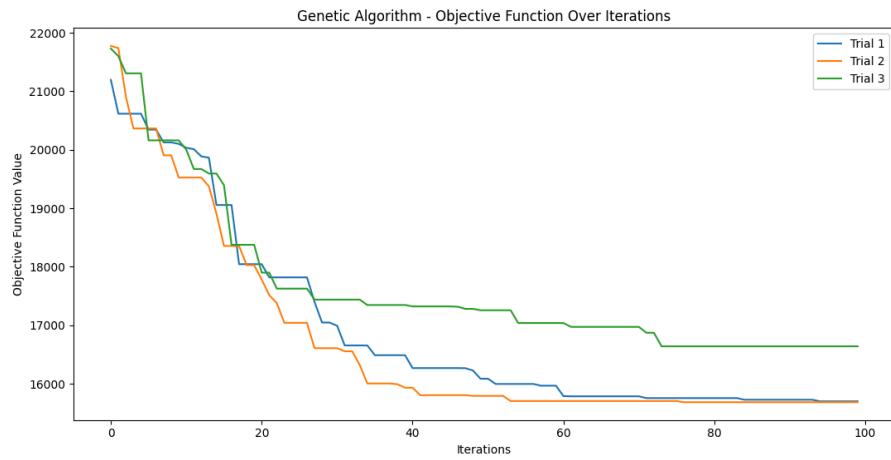


Genetic Algorithm - Initial State (Trial 3)



Genetic Algorithm - Final State (Trial 3)





Output program:

```
Running Genetic Algorithm with Population Size 200 and Generations 100
Running Genetic Algorithm, Trial 1...
Final Objective: 24145, Duration: 3.06 seconds
Running Genetic Algorithm, Trial 2...
Final Objective: 23756, Duration: 3.03 seconds
Running Genetic Algorithm, Trial 3...
Final Objective: 23652, Duration: 3.10 seconds
Experiment with Population Size 200 and Generations 100 completed.
```

## 2) Analisis

Algoritma Local Search yang diterapkan, yaitu Hill Climbing, Simulated Annealing, dan Genetic Algorithm, menunjukkan variasi tingkat keberhasilan dalam mendekati solusi global optimum. Genetic Algorithm cenderung lebih mendekati solusi optimal dibandingkan Hill Climbing dan Simulated Annealing. Hal ini dikarenakan Genetic Algorithm memanfaatkan seleksi populasi dan crossover, yang memungkinkan eksplorasi lebih luas di ruang solusi dibandingkan dua algoritma lainnya yang berfokus pada eksplorasi solusi lokal. Algoritma Hill Climbing sering terjebak pada solusi lokal karena hanya mengejar perbaikan dari kondisi saat ini tanpa probabilitas menerima solusi lebih buruk untuk eksplorasi. Sementara itu, Simulated Annealing memperkenalkan probabilitas untuk menerima solusi yang lebih buruk pada suhu tinggi, sehingga memiliki kemampuan eksplorasi lebih baik daripada Hill Climbing.

Dari hasil eksperimen, terlihat bahwa Genetic Algorithm menghasilkan solusi yang lebih baik dan mendekati kondisi ideal dibandingkan dengan Hill Climbing dan Simulated Annealing. Hill Climbing, khususnya pada metode steepest ascent, menunjukkan keterbatasan dalam ruang solusi yang cenderung datar dan sering mengalami local optima, sementara Simulated Annealing menunjukkan hasil lebih baik karena kemampuannya menghindari jebakan solusi lokal dengan penurunan suhu secara bertahap. Genetic

Algorithm menonjol dalam pencarian solusi dengan berbagai variasi individu di setiap generasi, memungkinkan peningkatan nilai solusi lebih signifikan dibandingkan algoritma lainnya.

Dari segi efisiensi waktu, durasi pencarian pada Simulated Annealing lebih singkat dibandingkan Hill Climbing dan Genetic Algorithm. Simulated Annealing mencapai keseimbangan antara eksplorasi dan efisiensi waktu, dengan memanfaatkan mekanisme penurunan suhu yang mengendalikan probabilitas penerimaan solusi yang kurang optimal. Meskipun memiliki pendekatan untuk menghindari jebakan local optima, prosesnya tetap lebih cepat dibandingkan Genetic Algorithm, yang memerlukan lebih banyak waktu untuk seleksi, crossover, dan mutasi populasi dalam beberapa generasi. Durasi yang singkat ini menunjukkan bahwa Simulated Annealing dapat menawarkan efisiensi yang lebih baik dalam konteks tertentu, terutama bila kualitas solusi cukup mendekati optimum tanpa perlu eksplorasi ruang solusi yang ekstensif seperti dalam Genetic Algorithm.

Dari segi konsistensi, Hill Climbing cenderung menunjukkan hasil akhir yang cukup seragam karena sifatnya yang deterministik, namun sering kali jauh dari solusi optimal. Simulated Annealing lebih bervariasi, terutama tergantung pada nilai awal suhu dan cooling rate, sehingga pada beberapa eksperimen hasilnya dapat mendekati optimum. Genetic Algorithm menunjukkan konsistensi yang lebih tinggi dalam mencapai solusi optimal, terutama saat jumlah populasi dan generasi yang diuji lebih banyak, sehingga memungkinkan algoritma untuk lebih eksploratif.

Berdasarkan eksperimen, peningkatan jumlah iterasi dan populasi berpengaruh signifikan pada hasil Genetic Algorithm. Semakin banyak generasi dan populasi yang diterapkan, semakin baik Genetic Algorithm mendekati solusi optimal. Hal ini karena jumlah iterasi dan populasi yang lebih besar memungkinkan lebih banyak eksplorasi dan kombinasi solusi, sehingga meningkatkan peluang untuk menemukan solusi yang lebih dekat dengan global optima.

# Kesimpulan dan Saran

## 1. Kesimpulan

Berdasarkan hasil eksperimen pada Diagonal Magic Cube 5x5x5, implementasi metode Local Search menunjukkan perbedaan kinerja antara Hill Climbing, Simulated Annealing, dan Genetic Algorithm dalam mencapai solusi optimal. Genetic Algorithm terbukti lebih efektif dalam mendekati solusi optimal karena mampu mengeksplorasi ruang solusi lebih luas melalui mekanisme seleksi dan crossover, sehingga dapat menghindari jebakan local optima. Simulated Annealing memiliki keunggulan dalam efisiensi waktu, dengan mekanisme penurunan suhu yang memungkinkan penerimaan solusi kurang optimal pada suhu tinggi, memberikan fleksibilitas lebih dibandingkan Hill Climbing dalam eksplorasi ruang solusi. Meski demikian, kualitas solusinya bervariasi tergantung pada parameter suhu dan cooling rate. Sebaliknya, Hill Climbing menunjukkan keterbatasan dalam mencapai solusi optimal karena sifat deterministiknya yang sering terjebak pada local optima tanpa peluang eksplorasi lebih luas.

## 2. Saran

- 1) Untuk peningkatan performa algoritma, disarankan untuk meningkatkan parameter populasi dan jumlah generasi pada Genetic Algorithm, karena hasil eksperimen menunjukkan bahwa variasi yang lebih besar dari kedua parameter ini mampu menghasilkan solusi yang lebih baik.
- 2) Penggunaan algoritma hibrida, seperti mengombinasikan Genetic Algorithm dan Simulated Annealing, dapat dipertimbangkan untuk memanfaatkan kekuatan eksplorasi dari Genetic Algorithm dan efisiensi waktu dari Simulated Annealing.
- 3) Mengoptimalkan parameter suhu dan cooling rate pada Simulated Annealing dapat meningkatkan efisiensi serta kualitas solusi yang dicapai, khususnya dalam pencarian solusi optimal dalam waktu yang lebih singkat.

# Pembagian Tugas

NIM	Nama	Tugas
18222073	Yoga Putra Pratama	Source code, laporan

# **Referensi**

Stuart J Russell & Peter Norvig, Artificial Intelligence: A Modern Approach, 4th Edition, Prentice-Hall International, Inc, 2022.