Tugas Besar 1

IF3070 Dasar Inteligensi Artifisial

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

Disusun oleh:

Kelompok 7

Angelica Aliwinata / 18222113
Jason Jahja / 18222116
Melissa Trenggono/ 18222123
Anindita Widya S. / 18222128

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

Daftar Isi

Daftar Tabel

Pendahuluan

Bagian ini menjelaskan persoalan yang dipecahkan dalam dokumen ini serta spesifikasi dari tugas yang dikerjakan.

A. Deskripsi Persoalan

Tugas Besar I dalam kuliah IF3070 Dasar Inteligensi Artifisial bertujuan agar penulis memahami implementasi algoritma local search seperti Steepest Ascent Hill-Climbing, Hill-Climbing with Sideways Move, Random Restart Hill-Climbing, Stochastic Hill-Climbing, Simulated Annealing, dan Genetic Algorithm untuk menyelesaikan optimasi Diagonal Magic Cube. Melalui eksperimen, penulis akan mengevaluasi kinerja tiap algoritma dalam mendekati solusi optimal (global optimum), durasi proses, dan konsistensi hasil. Analisis ini juga mencakup pengaruh jumlah iterasi dan variasi populasi pada Genetic Algorithm, dan hasil eksperimen divisualisasikan agar memudahkan perbandingan efektivitas dan efisiensi tiap metode.

B. Spesifikasi Tugas

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 dilakukan (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 dibuat sendiri atau dari referensi lain untuk optimasi pencarian solusi, asalkan masih dalam lingkup local search. Jangan lupa jelaskan heuristik yang dipakai di laporan.

Selain itu, terdapat pula spesifikasi untuk bonus yang dapat dikerjakan ketika seluruh spesifikasi wajib sudah diselesaikan. Berikut merupakan beberapa spesifikasi bonus yang dapat dikerjakan:

- Buatlah implementasi untuk seluruh algoritma hill-climbing yang ada.
- Buatlah suatu ‘video player’ yang dapat menayangkan *replay* dari proses pencarian secara detail per tahap/iterasinya, berikut rincian fiturnya:
    - Play/pause
    - Progress bar (bisa *drag and drop*)
    - Playback speed
    - Load file hasil eksperimen (File eksperimen ini menyimpan tiap state/kondisi per iterasi dari proses pencarian)

Teori Dasar

Bagian ini akan menjelaskan teori dasar yang menjadi dasar pengembangan algoritma yang digunakan dalam tugas ini.

A. Objective Function

Objective function adalah fungsi yang digunakan untuk mengukur seberapa baik suatu solusi memenuhi kriteria yang diinginkan dalam konteks optimasi. Dalam kasus *Diagonal Magic Cube*, *objective function* bertujuan untuk mengukur seberapa dekat susunan angka dalam kubus mendekati nilai target yang disebut *magic number*. *Magic number* adalah jumlah yang harus dicapai oleh setiap baris, kolom, pilar, dan diagonal di dalam kubus agar solusi dianggap optimal.

Nilai *objective function* ditentukan dengan menghitung selisih antara jumlah angka pada setiap komponen kubus (baris, kolom, pilar, dan berbagai jenis diagonal) dengan *magic number*. Selisih ini dijumlahkan secara keseluruhan untuk memberikan ukuran yang menunjukkan seberapa jauh kubus dari kondisi optimal. Tujuan optimasi adalah meminimalkan nilai dari *objective function* ini, yang berarti mendekati atau mencapai kondisi di mana setiap komponen memenuhi nilai *magic number*.

Perhitungan magic number untuk kubus $n \, x \, n \, x \, n$ mengikuti rumus $M(n) = \frac{n(n^3+1)}{2}$, yang memberikan patokan target bagi seluruh komponen dalam kubus. Semakin kecil nilai *objective function*, semakin dekat kubus ke konfigurasi optimal. Sehingga ketika *objective function* bernilai nol, maka semua komponen kubus sudah memenuhi syarat *magic number*, menunjukkan bahwa kondisi optimal telah tercapai.

B. Algoritma

Berikut adalah penjelasan dari beberapa pendekatan algoritma local search dan implementasinya pada diagonal magic cube.

a. Steepest Ascent Hill-climbing

Steepest Ascent Hill-Climbing adalah algoritma optimasi berbasis *local search* yang berfungsi untuk mencari solusi terbaik dengan memperbaiki nilai heuristik dari suatu keadaan. Dalam konteks algoritma ini, heuristik merupakan strategi untuk memperkirakan seberapa baik sebuah solusi tanpa harus mengeksplorasi seluruh opsi yang ada.

Algoritma ini dimulai dari suatu keadaan awal, kemudian mengevaluasi tetangga-tetangganya, dan memilih tetangga dengan nilai heuristik terbaik yang meningkatkan objective function. Proses ini akan terus diulang hingga tidak ada lagi tetangga dengan nilai yang lebih baik. Meskipun algoritma ini efektif dan efisien, kelemahannya adalah kecenderungannya terjebak di local maximum, di mana tidak ada solusi tetangga yang lebih baik, meskipun mungkin masih jauh dari global maximum yang merupakan solusi terbaik.

Pada penerapannya untuk menyelesaikan *Diagonal Magic Cube*, *Steepest Ascent Hill-Climbing* dimulai dari keadaan awal kubus yang diacak, di mana setiap elemen dalam kubus memiliki nilai tertentu. Tujuan dari algoritma ini adalah meminimalkan perbedaan antara jumlah nilai elemen di setiap baris, kolom, tiang, dan diagonal dengan *magic constant*. *Magic constant* atau *magic number* adalah nilai total yang diharapkan pada setiap baris, kolom, dan diagonal untuk mencapai keadaan optimal. Algoritma akan melakukan pertukaran dua elemen di dalam kubus, mencari kombinasi terbaik yang meminimalkan selisih dari magic constant.

Proses iterasi *Steepest Ascent Hill-Climbing* terus berlangsung hingga mencapai keadaan di mana tidak ada lagi perbaikan yang bisa dilakukan tanpa menurunkan nilai heuristik. Namun, ada risiko terjebak di *local maximum*, di mana keadaan yang dicapai bukanlah keadaan optimal secara global, tetapi tidak ada tetangga yang lebih baik.

b. Hill-climbing with Sideways Move

Hill-Climbing with Sideways Move adalah algoritma yang dikembangkan untuk mengatasi kondisi *flat* atau kondisi di mana tidak ada tetangga (*neighbor*) yang memiliki nilai lebih baik dari keadaan saat ini. Mirip dengan *Steepest Ascent Hill-Climbing*, algoritma ini mencari tetangga terbaik untuk memperbaiki solusi saat ini. Namun, perbedaannya adalah bahwa *Hill-Climbing with Sideways Move* tetap akan bergerak ke tetangga dengan nilai yang sama baiknya, bahkan jika tidak ada perbaikan dalam *objective function*. Hal ini memberikan algoritma kesempatan untuk keluar dari *flat local maxima* dan meningkatkan peluang mencapai *global optimum*.

Pendekatan ini bermanfaat dalam penyelesaian masalah seperti *Diagonal Magic Cube*, di mana solusi optimal dicapai ketika jumlah nilai di setiap baris, kolom, tiang, dan diagonal mendekati atau sama dengan *magic constant*. Ketika berada pada kondisi *flat*, algoritma ini tetap mencoba berpindah ke tetangga dengan nilai yang sama baiknya, memungkinkan eksplorasi lebih lanjut untuk menemukan solusi yang lebih baik.

Kelebihan utama dari *Hill-Climbing with Sideways Move* adalah kemampuannya untuk menghindari terminasi prematur di flat *local maxima*, yang meningkatkan peluang mencapai solusi optimal dibandingkan *Steepest Ascent Hill-Climbing*. Namun, algoritma ini tetap memiliki kelemahan, yaitu kemungkinan terjebak di *local maximum*, meskipun dalam beberapa kasus, algoritma dapat memberikan hasil yang lebih optimal melalui perpindahan *sideways* di kondisi *flat*.

c. *Random Restart Hill-climbing*

Random Restart Hill-Climbing adalah varian dari algoritma *hill-climbing* yang dirancang untuk mengatasi masalah terjebak di *local maximum* dan *flat condition*. Algoritma ini bekerja mirip dengan *Steepest Ascent Hill-Climbing* dengan perbedaan signifikan yaitu adanya mekanisme *restart*. Ketika algoritma mencapai keadaan *local maximum* dan tidak ada perbaikan lebih lanjut, ia akan mengulang proses pencarian dengan keadaan awal yang diacak. Proses ini terus dilakukan hingga mencapai kondisi optimal atau jumlah restart yang telah ditentukan.

Pendekatan *Random Restart Hill-Climbing* efektif dalam menemukan solusi optimal dari permasalahan seperti *Diagonal Magic Cube*, di mana tujuan algoritma adalah mencapai *magic constant* dengan meminimalkan perbedaan di setiap baris, kolom, tiang, dan diagonal. Ketika algoritma terjebak dalam *local maximum*, ia akan melakukan *restart* pada kubus secara acak, memungkinkan algoritma untuk mengeksplorasi berbagai kemungkinan konfigurasi.

Keunggulan algoritma ini terletak pada kemampuannya mengeksplorasi ruang solusi yang lebih luas melalui restart acak, meningkatkan peluang mencapai *global optimum*. Namun, kekurangannya adalah waktu komputasi yang lebih lama karena proses diulang beberapa kali, dan *restart* acak tidak selalu menjamin akan mencapai solusi global.

d. *Stochastic Hill-climbing*

Stochastic Hill-Climbing adalah algoritma yang memilih tetangga secara acak dalam setiap langkahnya, berbeda dengan metode *hill-climbing* biasa yang mencari tetangga terbaik secara sistematis. Pada setiap iterasi, algoritma mengevaluasi tetangga acak dari keadaan saat ini dan menggantikannya hanya jika tetangga tersebut memiliki nilai objektif yang lebih baik. Pendekatan acak ini membantu algoritma mengeksplorasi lebih banyak kemungkinan solusi, sehingga mengurangi risiko terjebak di titik local maxima.

Dalam penerapan pada *Diagonal Magic Cube*, algoritma dimulai dengan menyusun angka secara acak di dalam kubus, yang sering kali jauh dari kondisi optimal. Pada setiap langkah, dua angka di dalam kubus dipilih secara acak untuk ditukar posisinya, dan nilai objektif dihitung. Jika pertukaran ini mengurangi selisih dari magic constant, tetangga baru diadopsi sebagai keadaan saat ini. Proses ini terus berlanjut hingga batas iterasi tercapai atau solusi optimal ditemukan.

Keunggulan utama dari *Stochastic Hill-Climbing* adalah kemampuannya mengeksplorasi solusi secara luas melalui pemilihan tetangga secara acak, yang bisa membantu menghindari jebakan pada solusi lokal yang kurang optimal. Namun, pemilihan tetangga secara acak juga bisa memperlambat proses pencarian, terutama jika pilihan tetangga yang lebih baik jarang ditemukan.

e. Simulated Annealing

Simulated Annealing adalah algoritma optimasi yang diilhami oleh proses *annealing* pada fisika material, di mana logam dipanaskan hingga suhu tinggi dan didinginkan secara bertahap untuk mencapai struktur kristal yang stabil. Dalam konteks algoritma, suhu (T) digunakan untuk mengatur probabilitas penerimaan solusi yang lebih buruk selama pencarian solusi, memungkinkan algoritma mengeksplorasi lebih luas di awal pencarian dan berfokus pada solusi terbaik saat suhu menurun.

Pada setiap iterasi, algoritma memilih tetangga secara acak dari keadaan saat ini. Jika tetangga tersebut lebih baik, maka langsung diterima sebagai keadaan baru. Namun, jika tetangga lebih buruk, penerimaannya bergantung pada probabilitas yang dihitung dari perbedaan kualitas solusi (ΔE) dan suhu (T) saat itu. Probabilitas ini dihitung sebagai $P = e^{\frac{\Delta E}{T}}$, di mana nilai yang lebih tinggi dari T meningkatkan peluang untuk menerima solusi yang lebih buruk. Seiring waktu, T menurun, mengurangi probabilitas menerima solusi yang tidak optimal, sehingga algoritma lebih cenderung mempertahankan solusi yang lebih baik.

Pendekatan ini cocok untuk mencari solusi *Diagonal Magic Cube*, karena dengan memilih tetangga secara acak dan menurunkan T secara bertahap, algoritma dapat menghindari jebakan pada *local maxima* dan tetap fokus pada pencarian solusi optimal. Proses berlanjut hingga mencapai suhu nol atau ditemukan solusi yang dianggap optimal, menjaga keseimbangan antara eksplorasi solusi yang luas dan eksploitasi solusi terbaik.

f. *Genetic Algorithm*

Genetic Algorithm adalah algoritma optimasi yang meniru proses evolusi biologis untuk menemukan solusi terbaik dari suatu permasalahan. Algoritma ini berawal dari pembentukan populasi awal yang terdiri dari beberapa kandidat solusi (individu) yang kemudian diseleksi, diikuti oleh proses *crossover* (perkawinan) dan mutasi, untuk menghasilkan generasi baru yang diharapkan memiliki solusi yang lebih baik. Tahapan evolusi ini dilakukan secara iteratif, memungkinkan algoritma mengeksplorasi berbagai kemungkinan solusi dan mengurangi risiko terjebak pada local optimum.

Proses seleksi bertujuan memilih individu-individu terbaik dari populasi berdasarkan suatu nilai *fitness*, yang mengukur seberapa baik solusi yang ditawarkan individu tersebut dalam menyelesaikan masalah. Individu-individu yang terpilih sebagai induk kemudian menjalani *crossover*, di mana sebagian karakteristik dari kedua induk dipadukan untuk membentuk keturunan yang baru. *Crossover* ini menghasilkan variasi solusi yang diharapkan dapat mendekati atau mencapai hasil optimal.

Selain itu, mutasi dilakukan dengan mengubah beberapa bagian dari individu secara acak, bertujuan untuk menjaga keragaman dalam populasi dan mencegah algoritma stagnan pada *local optimum*. Dengan mengevaluasi dan memperbarui populasi secara bertahap, *Genetic Algorithm* mampu mengeksplorasi ruang solusi secara luas, sehingga memiliki potensi besar untuk menemukan solusi yang optimal, meskipun memerlukan waktu komputasi yang lebih lama.

Implementasi

Bagian ini menjelaskan penerapan teori menjadi fungsi-fungsi yang dikembangkan dalam kode untuk menyelesaikan kasus *Diagonal Magic Cube* pada tugas ini.

A. Struktur Folder

Berikut adalah struktur dari folder kode (src)

```
src
│
├── algorithms
│   ├── genetic_algorithm.py
│   ├── random_restart.py
│   ├── sideways_move.py
│   ├── simulated_annealing.py
│   ├── steepest_hill_climbing.py
│   └── stochastic.py
│
├── cube
│   └── magic_cube.py
│
└── main.py
```

B. Objective Function

Implementasi *Objective Function* terdapat pada *file* magic_cube.py

Tabel 3.1 Fungsi dan Kode *Objective Function*

Nama Fungsi	objective_function(self)
Deskripsi Fungsi	Menghitung skor yang menggambarkan seberapa jauh konfigurasi saat ini dari target nilai yang diinginkan, yaitu sebuah *magic number*.

Kode

```
def objective_function(self):  20 usages (17 dynamic)  jasonjahja
    score = 0

    # jumlah tiap baris, kolom, tiang
    for i in range(self.size):
        for j in range(self.size):
            row_sum = sum(self.cube[k][i][j] for k in range(self.size))
            col_sum = sum(self.cube[i][k][j] for k in range(self.size))
            pillar_sum = sum(self.cube[i][j][k] for k in range(self.size))

            score += abs(row_sum - self.magic_number)
            score += abs(col_sum - self.magic_number)
            score += abs(pillar_sum - self.magic_number)

    # jumlah tiap diagonal ruang
    diag1_sum = sum(self.cube[i][i][i] for i in range(self.size))
    diag2_sum = sum(self.cube[i][i][self.size - i - 1] for i in range(self.size))
    diag3_sum = sum(self.cube[i][self.size - i - 1][i] for i in range(self.size))
    diag4_sum = sum(self.cube[self.size - i - 1][i][i] for i in range(self.size))
    score += abs(diag1_sum - self.magic_number)
    score += abs(diag2_sum - self.magic_number)
    score += abs(diag3_sum - self.magic_number)
    score += abs(diag4_sum - self.magic_number)
```

```
# jumlah tiap diagonal bidang
for i in range(self.size):
    diag1_xy = sum(self.cube[j][j][i] for j in range(self.size))
    diag2_xy = sum(self.cube[j][self.size - j - 1][i] for j in range(self.size))
    score += abs(diag1_xy - self.magic_number)
    score += abs(diag2_xy - self.magic_number)

    diag1_xz = sum(self.cube[j][i][j] for j in range(self.size))
    diag2_xz = sum(self.cube[self.size - j - 1][i][j] for j in range(self.size))
    score += abs(diag1_xz - self.magic_number)
    score += abs(diag2_xz - self.magic_number)

    diag1_yz = sum(self.cube[i][j][j] for j in range(self.size))
    diag2_yz = sum(self.cube[i][j][self.size - j - 1] for j in range(self.size))
    score += abs(diag1_yz - self.magic_number)
    score += abs(diag2_yz - self.magic_number)

return score
```

C. *Steepest Ascent Hill-climbing*

Implementasi *Steepest Ascent Hill-climbing* terdapat pada *file* steepest_hill_climbing.py

Tabel 3.2 Fungsi dan Kode __init__ pada *Steepest Ascent Hill-Climbing*

Nama Fungsi	__init__(self, cube)
Deskripsi Fungsi	Menginisialisasi objek SteepestHillClimbing , menyimpan hasil dari skor *objective function*, dan menginisialisasi *score* untuk menyimpan skor dari setiap iterasi.
Kode	

```
class SteepestHillClimbing:
    def __init__(self, cube):
        self.cube = cube
        self.best_score = self.cube.objective_function()
        self.scores = []
```

Tabel 3.3 Fungsi dan Kode swap_elements pada *Steepest Ascent Hill-Climbing*

Nama Fungsi	swap_elements(self, pos1, pos2)
Deskripsi Fungsi	Menukar dua elemen pada posisi 1 dan posisi 2 dalam kubus.
Kode	

```
def swap_elements(self, pos1, pos2):
    x1, y1, z1 = pos1
    x2, y2, z2 = pos2
    self.cube.cube[x1][y1][z1], self.cube.cube[x2][y2][z2] = self.cube.cube[x2][y2][z2], self.cube.cube[x1][y1][z1]
```

Tabel 3.4 Fungsi dan Kode find_best_neighbor pada *Steepest Ascent Hill-Climbing*

Nama Fungsi	find_best_neighbor(self)
Deskripsi Fungsi	Mencari posisi terbaik, yaitu posisi yang menghasilkan delta terbesar yang dinyatakan kondisi (delta > best_delta) antara skor setelah di *swap* dengan yang sebelumnya serta nilai delta yang dihasilkan.
Kode	

```
def find_best_neighbor(self):
    delta = 0
    best_delta = 0

    positions = [(x, y, z) for x in range (self.cube.size) for y in range (self.cube.size) for z in range (self.cube.size)]
    for pos1 in positions:
        for pos2 in positions:
            if pos2 > pos1:
                self.swap_elements(pos1, pos2)
                new_score = self.cube.objective_function()
                delta = self.best_score - new_score

                if delta > best_delta:
                    best_delta = delta
                    best_positions = (pos1, pos2)

                self.swap_elements(pos1, pos2)

    return best_positions, best_delta
```

Tabel 3.5 Fungsi dan Kode run pada *Steepest Ascent Hill-Climbing*

Nama Fungsi	run(self, max_iterations=1000)
Deskripsi Fungsi	Menjalankan algoritma *Steepest Ascent Hill-Climbing* hingga mencapai batas maksimum iterasi dan mencatat setiap skor iterasi ke dalam *score*. Fungsi ini akan mengembalikan skor akhir, durasi, dan jumlah iterasi.
Kode	

```
def run(self, max_iterations=1000):
    start_time = time.perf_counter()
    last_best_delta = 0
    iteration = 0

    self.scores.append(self.cube.objective_function())

    while iteration < max_iterations:
        best_positions, best_delta = self.find_best_neighbor()

        if best_delta == last_best_delta:
            break

        self.swap_elements(*best_positions)
        self.scores.append(self.cube.objective_function())
        # print(f"Iteration {iteration}: objective score {self.cube.objective_function()}, best delta: {best_delta}\n")
        last_best_delta = best_delta
        iteration += 1

    final_score = self.cube.objective_function()
    end_time = time.perf_counter()
    duration = end_time - start_time
    self.plot_scores(iteration)

    return final_score, duration, iteration
```

D. Hill-climbing with Sideways Move

Implementasi *Hill-climbing with Sideways Move* terdapat pada *file* sideways_move.py

Tabel 3.6 Fungsi dan Kode __init__ pada Hill-climbing with Sideways Move

Nama Fungsi	__init__(self, cube)
Deskripsi Fungsi	Menginisialisasi objek di *Hill-climbing with Sideways Move* , menyimpan hasil dari skor *objective function*, dan menginisialisasi *score* untuk menyimpan skor dari setiap iterasi.
Kode	

```
class SidewaysMove:
    def __init__(self, cube):
        self.cube = cube
        self.best_score = self.cube.objective_function()
        self.scores = []
```

Tabel 3.7 Fungsi dan Kode swap_elements pada Hill-climbing with Sideways Move

Nama Fungsi	swap_elements(self, pos1, pos2)
Deskripsi Fungsi	Menukar dua elemen pada posisi 1 dan posisi 2 dalam kubus.
Kode	

```
def swap_elements(self, pos1, pos2):
    x1, y1, z1 = pos1
    x2, y2, z2 = pos2
    self.cube.cube[x1][y1][z1], self.cube.cube[x2][y2][z2] = self.cube.cube[x2][y2][z2], self.cube.cube[x1][y1][z1]
```

Tabel 3.8 Fungsi dan Kode find_best_neighbor pada Hill-climbing with Sideways Move

Nama Fungsi	find_best_neighbor(self)
Deskripsi Fungsi	Mencari posisi terbaik, yaitu posisi yang menghasilkan delta terbesar yang dinyatakan kondisi (delta >= best_delta) antara skor setelah di *swap* dengan yang sebelumnya serta nilai delta yang dihasilkan.
Kode	

```
def find_best_neighbor(self):
    delta = 0
    best_delta = 0

    positions = [(x, y, z) for x in range (self.cube.size) for y in range (self.cube.size) for z in range (self.cube.size)]
    for pos1 in positions:
        for pos2 in positions:
            if pos2 > pos1:
                self.swap_elements(pos1, pos2)
                new_score = self.cube.objective_function()
                delta = self.best_score - new_score

                if delta >= best_delta:
                    best_delta = delta
                    best_positions = (pos1, pos2)

                self.swap_elements(pos1, pos2)

    return best_positions, best_delta
```

Tabel 3.9 Fungsi dan Kode run pada Hill-Climbing with Sideways Move

Nama Fungsi	run(self, max_iterations=1000, max_sideways_iteration=100)
Deskripsi Fungsi	Menjalankan algoritma *Hill-climbing with Sideways Move* hingga mencapai batas iterasi maksimum atau batas iterasi *sideways* maksimum yang dinyatakan dengan max_sideways_iteration. Fungsi ini akan mengembalikan skor akhir, durasi, dan juga banyak iterasi.

Kode	

```
def run(self, max_iterations=1000, max_sideways_iteration=100):
    start_time = time.perf_counter()
    last_best_delta = 0
    iteration = 0
    sideways_iteration = 0

    self.scores.append(self.cube.objective_function())

    while iteration < max_iterations:
        best_positions, best_delta = self.find_best_neighbor()

        if best_delta == last_best_delta:
            sideways_iteration += 1
        else:
            sideways_iteration = 0

        if sideways_iteration >= max_sideways_iteration:
            break

        self.swap_elements(*best_positions)
        self.scores.append(self.cube.objective_function())
        # print(f"Iteration {iteration}: objective score {self.cube.objective_function()}\n")
        last_best_delta = best_delta
        iteration += 1

    final_score = self.cube.objective_function()
    end_time = time.perf_counter()
    duration = end_time - start_time
    self.plot_scores(iteration)

    return final_score, duration, iteration
```

E. Random Restart Hill-climbing

Implementasi *Random Restart Hill-climbing* terdapat pada *file* random_restart.py

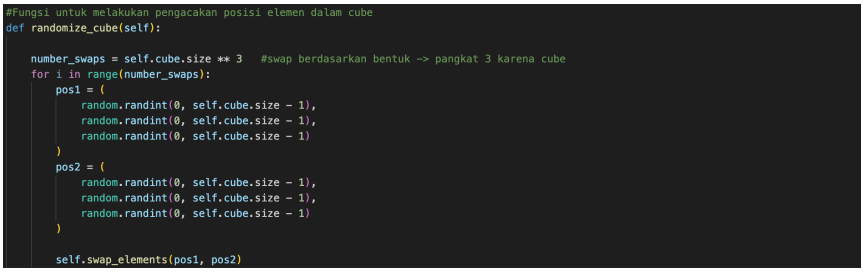
Tabel 3.10 Fungsi dan Kode __init__ pada Random Restart Hill Climbing

Nama Fungsi	__init__ (self, max_restarts, max_iterations_per_restart)
Deskripsi Fungsi	Menginisialisasi objek pada *Random Restart Hill-climbing* dengan mengambil objek *cube*, jumlah *restart* maksimal, dan jumlah iterasi maksimal untuk setiap *restart*. Fungsi ini juga akan menginisialisasi *scores* untuk menyimpan skor objektif pada setiap iterasi nantinya.
Kode	

```
#Class untuk Random Restart Hill Climbing
class RandomRestartHillClimbing:
    #Memulai tahap inisialisasi
    def __init__(self, cube, max_restarts, max_iterations_per_restart):
        self.cube = cube
        self.max_restarts = max_restarts
        self.max_iterations_per_restart = max_iterations_per_restart
        self.scores = [] #menyimpan objective score setiap iterasi
```

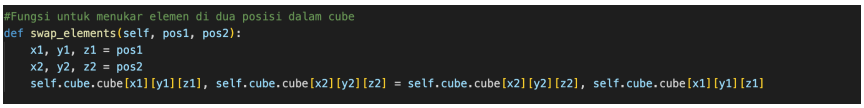
Tabel 3.11 Fungsi dan Kode randomize_cube pada Random Restart Hill Climbing

Nama Fungsi	randomize_cube(self)
Deskripsi Fungsi	Melakukan pengacakan terhadap posisi elemen yang berada dalam *cube*.

Kode	

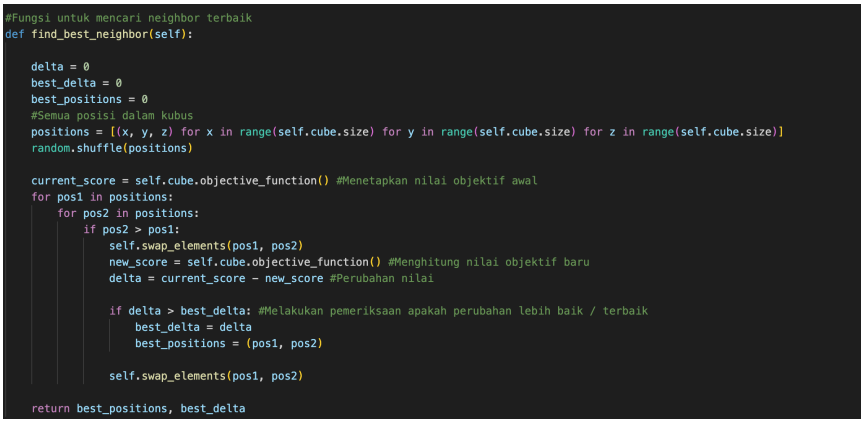
```
#Fungsi untuk melakukan pengacakan posisi elemen dalam cube
def randomize_cube(self):

    number_swaps = self.cube.size ** 3   #swap berdasarkan bentuk -> pangkat 3 karena cube
    for i in range(number_swaps):
        pos1 = (
            random.randint(0, self.cube.size - 1),
            random.randint(0, self.cube.size - 1),
            random.randint(0, self.cube.size - 1)
        )
        pos2 = (
            random.randint(0, self.cube.size - 1),
            random.randint(0, self.cube.size - 1),
            random.randint(0, self.cube.size - 1)
        )

        self.swap_elements(pos1, pos2)
```

Tabel 3.12 Fungsi dan Kode swap_elements pada Random Restart Hill Climbing

Nama Fungsi	swap_elements(self,pos1,pos2)
Deskripsi Fungsi	Menukar dua elemen pada posisi 1 dan posisi 2 dalam kubus.
Kode	

```
#Fungsi untuk menukar elemen di dua posisi dalam cube
def swap_elements(self, pos1, pos2):
    x1, y1, z1 = pos1
    x2, y2, z2 = pos2
    self.cube.cube[x1][y1][z1], self.cube.cube[x2][y2][z2] = self.cube.cube[x2][y2][z2], self.cube.cube[x1][y1][z1]
```

Tabel 3.13 Fungsi dan Kode find_best_neighbor pada Random Restart Hill Climbing

Nama Fungsi	find_best_neighbor(self)
Deskripsi Fungsi	Mencari posisi terbaik, yaitu posisi yang menghasilkan delta terbesar yang dinyatakan kondisi (delta > best_delta) antara skor setelah di *swap* dengan yang sebelumnya serta nilai delta yang dihasilkan.
Kode	

```
#Fungsi untuk mencari neighbor terbaik
def find_best_neighbor(self):

    delta = 0
    best_delta = 0
    best_positions = 0
    #Semua posisi dalam kubus
    positions = [(x, y, z) for x in range(self.cube.size) for y in range(self.cube.size) for z in range(self.cube.size)]
    random.shuffle(positions)

    current_score = self.cube.objective_function() #Menetapkan nilai objektif awal
    for pos1 in positions:
        for pos2 in positions:
            if pos2 > pos1:
                self.swap_elements(pos1, pos2)
                new_score = self.cube.objective_function() #Menghitung nilai objektif baru
                delta = current_score - new_score #Perubahan nilai

                if delta > best_delta: #Melakukan pemeriksaan apakah perubahan lebih baik / terbaik
                    best_delta = delta
                    best_positions = (pos1, pos2)

                self.swap_elements(pos1, pos2)

    return best_positions, best_delta
```

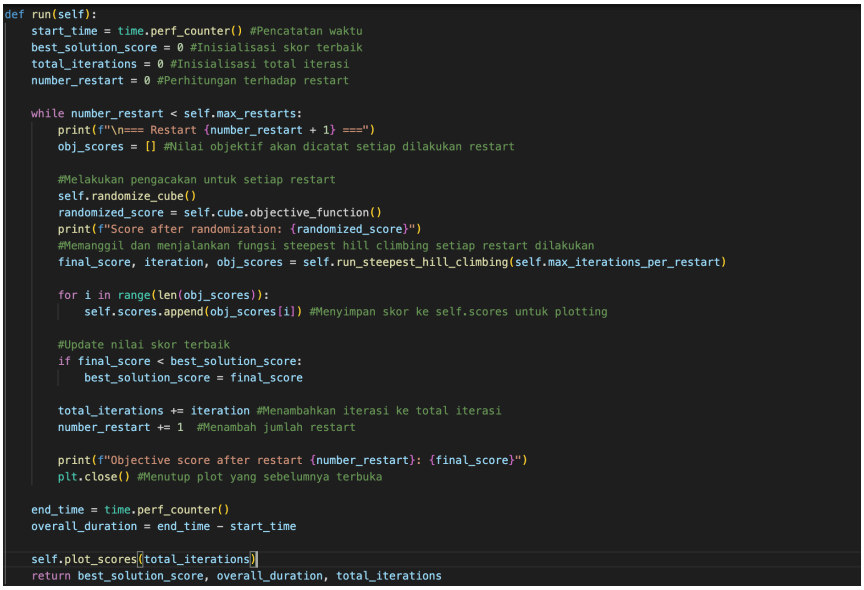
Tabel 3.14 Fungsi dan Kode run_steepest_hill_climbing pada Random Restart Hill Climbing

Nama Fungsi	run_steepest_hill_climbing(self,max_iterations)
Deskripsi Fungsi	Memanggil dan menjalankan algoritma *Steepest Ascent Hill-Climbing* hingga mencapai batas maksimum iterasi dan mencatat setiap skor iterasi ke dalam *score*. Fungsi ini akan mengembalikan skor dan total iterasi dan fungsi ini dipanggil setiap *restart* dilakukan.
Kode	

```python
#Fungsi Steepest Hill Climbing
def run_steepest_hill_climbing(self, max_iterations):
    iteration = 0
    last_best_delta = 0
    best_score = self.cube.objective_function()
    scores = []
    print(f"Initial score: {best_score}")

    while iteration < max_iterations:
        best_positions, best_delta = self.find_best_neighbor()


        if best_delta == last_best_delta: #Jika tidak ada peningkatan / perbaikan, maka proses dihentikan
            break

        self.swap_elements(*best_positions)
        best_score -= best_delta #Melakukan update untuk skor terbaik
        iteration += 1
        scores.append(best_score)
        print(f"Iteration {iteration}, New score: {best_score}")

    final_score = self.cube.objective_function()
    print(f"Final score after hill climbing: {final_score}")
    return final_score, iteration, scores
```

Tabel 3.15 Fungsi dan Kode run pada Random Restart Hill Climbing

Nama Fungsi	run(self)
Deskripsi Fungsi	Menjalankan algoritma *Random Restart Hill Climbing* yang akan melakukan *restart* hingga kondisi maksimum tercapai dan akan terus memanggil fungsi *steepest hill climbing* setiap kali melakukan *restart*.

Kode	

```
def run(self):
    start_time = time.perf_counter() #Pencatatan waktu
    best_solution_score = 0 #Inisialisasi skor terbaik
    total_iterations = 0 #Inisialisasi total iterasi
    number_restart = 0 #Perhitungan terhadap restart

    while number_restart < self.max_restarts:
        print(f"\n=== Restart {number_restart + 1} ===")
        obj_scores = [] #Nilai objektif akan dicatat setiap dilakukan restart

        #Melakukan pengacakan untuk setiap restart
        self.randomize_cube()
        randomized_score = self.cube.objective_function()
        print(f"Score after randomization: {randomized_score}")
        #Memanggil dan menjalankan fungsi steepest hill climbing setiap restart dilakukan
        final_score, iteration, obj_scores = self.run_steepest_hill_climbing(self.max_iterations_per_restart)

        for i in range(len(obj_scores)):
            self.scores.append(obj_scores[i]) #Menyimpan skor ke self.scores untuk plotting

        #Update nilai skor terbaik
        if final_score < best_solution_score:
            best_solution_score = final_score

        total_iterations += iteration #Menambahkan iterasi ke total iterasi
        number_restart += 1  #Menambah jumlah restart

        print(f"Objective score after restart {number_restart}: {final_score}")
        plt.close() #Menutup plot yang sebelumnya terbuka

    end_time = time.perf_counter()
    overall_duration = end_time - start_time

    self.plot_scores(total_iterations)
    return best_solution_score, overall_duration, total_iterations
```

F. Stochastic Hill-climbing

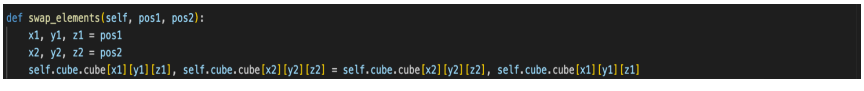
Implementasi *Stochastic Hill-climbing* terdapat pada *file* stochastic.py

Tabel 3.16 Fungsi dan Kode __init__ pada Stochastic Hill-climbing

Nama Fungsi	__init__(self, cube)
Deskripsi Fungsi	Menginisialisasi objek pada Stochastic Hill-climbing, menyimpan hasil dari skor *objective function*, dan menginisialisasi *score* untuk menyimpan skor dari setiap iterasi.
Kode	

```
class StochasticHillClimbing:
    def __init__(self, cube):
        self.cube = cube
        self.best_score = self.cube.objective_function()
        self.scores = []
```

Tabel 3.17 Fungsi dan Kode swap_elements pada Stochastic Hill-climbing

Nama Fungsi	swap_elements(self, pos1, pos2)
Deskripsi Fungsi	Menukar dua elemen pada posisi 1 dan posisi 2 dalam kubus.
Kode	

```
def swap_elements(self, pos1, pos2):
    x1, y1, z1 = pos1
    x2, y2, z2 = pos2
    self.cube.cube[x1][y1][z1], self.cube.cube[x2][y2][z2] = self.cube.cube[x2][y2][z2], self.cube.cube[x1][y1][z1]
```

Tabel 3.18 Fungsi dan Kode find_random_neighbor pada Stochastic Hill-climbing

Nama Fungsi	find_random_neighbor(self, best_delta)
Deskripsi Fungsi	Menukar dua elemen dengan posisi yang acak, jika dengan menukar kedua elemen tersebut, skor yang dihasilkan lebih baik, maka pertukaran tersebut akan dijalankan, namun jika tidak, maka pertukaran akan dibatalkan dan elemen akan dikembalikan ke posisi awal. Fungsi ini akan menghasilkan best_delta sebagai peningkatan skor terbaik.
Kode	

```
def find_random_neighbor(self, best_delta):
    pos1 = (random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1))
    pos2 = (random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1))
    while pos1 == pos2:
        pos2 = (random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1))

    self.swap_elements(pos1, pos2)
    new_score = self.cube.objective_function()
    delta = self.best_score - new_score

    if delta <= best_delta:
        self.swap_elements(pos1, pos2)
    else:
        best_delta = delta

    return best_delta
```

Tabel 3.19 Fungsi dan Kode run pada Stochastic Hill-climbing

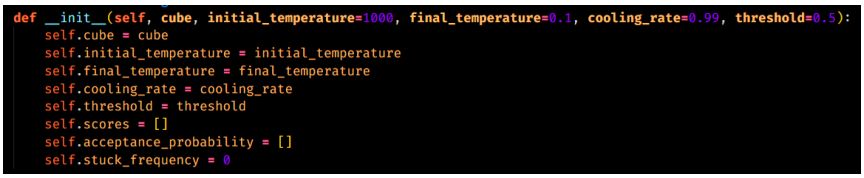
Nama Fungsi	run (self, max_iterations=1000)
Deskripsi Fungsi	Menjalankan algoritma *Stochastic Hill-climbing* dengan mencatat waktu mulai hingga mencapai batas iterasi maksimal yang ditentukan dan memanggil find_random_neighbor dan menyimpan skor ke dalam *scores*. Fungsi ini akan mengembalikan hasil akhir yaitu final_score, durasi (*duration*) dan *iteration* untuk menyatakan total iterasi.
Kode	

```
def run(self, max_iterations=1000):
    start_time = time.perf_counter()
    best_delta = 0
    iteration = 0

    self.scores.append(self.cube.objective_function())

    while iteration < max_iterations:
        best_delta = self.find_random_neighbor(best_delta)
        self.scores.append(self.cube.objective_function())
        iteration += 1

    final_score = self.cube.objective_function()
    end_time = time.perf_counter()
    duration = end_time - start_time
    self.plot_scores(iteration)

    return final_score, duration, iteration
```

G. *Simulated Annealing*

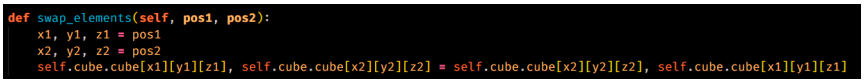
Implementasi *Simulated Annealing* terdapat pada *file* simulated_annealing.py

Tabel 3.20 Fungsi dan Kode __init__ pada *Simulated Annealing*

Nama Fungsi	__init__(self, cube, initial_temperature=1000, final_temperature=0.1, cooling_rate=0.99, threshold=0.5)
Deskripsi Fungsi	Menginisialisasi objek pada algoritma *Simulated Annealing*, menetapkan parameter suhu awal, suhu akhir, laju penurunan suhu, dan ambang batas untuk menentukan probabilitas penerimaan solusi buruk.
Kode	

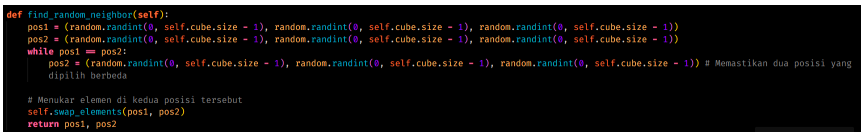
```
def __init__(self, cube, initial_temperature=1000, final_temperature=0.1, cooling_rate=0.99, threshold=0.5):
    self.cube = cube
    self.initial_temperature = initial_temperature
    self.final_temperature = final_temperature
    self.cooling_rate = cooling_rate
    self.threshold = threshold
    self.scores = []
    self.acceptance_probability = []
    self.stuck_frequency = 0
```

Tabel 3.21 Fungsi dan Kode swap_elements pada *Simulated Annealing*

Nama Fungsi	swap_elements(self, pos1, pos2)
Deskripsi Fungsi	Menukar dua elemen pada posisi 1 dan posisi 2 dalam kubus.
Kode	

```
def swap_elements(self, pos1, pos2):
    x1, y1, z1 = pos1
    x2, y2, z2 = pos2
    self.cube.cube[x1][y1][z1], self.cube.cube[x2][y2][z2] = self.cube.cube[x2][y2][z2], self.cube.cube[x1][y1][z1]
```

Tabel 3.22 Fungsi dan Kode find_random_neighbor pada *Simulated Annealing*

Nama Fungsi	find_random_neighbor(self)
Deskripsi Fungsi	Menemukan tetangga acak dengan menukar dua elemen dalam kubus secara acak dan mengembalikan kedua posisi elemen tersebut yang ditukar.
Kode	

```
def find_random_neighbor(self):
    pos1 = (random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1))
    pos2 = (random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1))
    while pos1 == pos2:
        pos2 = (random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1), random.randint(0, self.cube.size - 1)) # Memastikan dua posisi yang
        dipilih berbeda

    # Menukar elemen di kedua posisi tersebut
    self.swap_elements(pos1, pos2)
    return pos1, pos2
```

Tabel 3.23 Fungsi dan Kode run pada *Simulated Annealing*

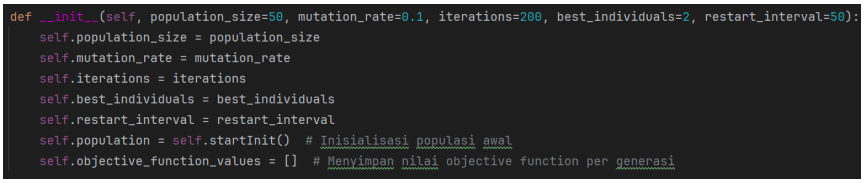
Nama Fungsi	run(self)
Deskripsi Fungsi	Menjalankan algoritma *simulated annealing* untuk mencari solusi optimal. Fungsi ini berfungsi untuk menemukan solusi terbaik dengan memulai pencarian solusi dari suhu tinggi dan menurunkannya secara bertahap. Selama proses berlangsung, fungsi ini dapat menerima solusi yang lebih baik secara langsung atau solusi yang lebih buruk berdasarkan probabilitas yang bergantung pada nilai delta dan suhu.
Kode	

```
def run(self):
    start_time = time.perf_counter()
    temperature = self.initial_temperature
    self.best_score = self.cube.objective_function()
    current_value = self.best_score

    iteration = 0

    # List untuk menyimpan probabilitas penerimaan solusi buruk
    acc_probability_list = []

    # Berhenti saat temperature mencapai final_temperature
    while temperature > self.final_temperature:
        pos1, pos2 = self.find_random_neighbor()
        neighbor_value = self.cube.objective_function()
        delta = current_value - neighbor_value

        # Jika solusi baru lebih buruk
        if delta < 0:
            acceptance_probability = math.exp(delta / temperature) # Hitung probabilitas penerimaan solusi buruk
            acc_probability_list.append(acceptance_probability)


            # Menerima solusi buruk jika probabilitasnya lebih besar dari threshold
            if self.threshold < acceptance_probability:
                current_value = neighbor_value

            # Solusi yang buruk tidak diambil jika probabilitasnya tidak lebih besar dari threshold
            else:
                self.swap_elements(pos1, pos2) # Menukar kembali kedua elemen di posisi tersebut: posisi elemen tidak jadi berubah
                self.stuck_frequency += 1 # Jumlah stuck di local optima bertambah

        # Jika solusi baru lebih baik, maka langsung diterima tanpa syarat
        else:
            current_value = neighbor_value
            acc_probability_list.append(1) # Probabilitas penerimaannya adalah 1

        self.scores.append(current_value)


        # Suhu berkurang berdasarkan cooling_rate
        temperature *= self.cooling_rate
        iteration += 1

    # Memperbarui best_score dengan nilai solusi saat ini jika solusi ini lebih baik daripada yang sebelumnya
    self.best_score = current_value

    end_time = time.perf_counter()
    duration = end_time - start_time

    return self.best_score, duration, self.stuck_frequency, iteration
```

H. *Genetic Algorithm*

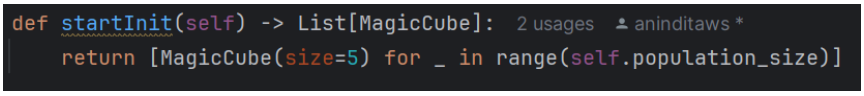
Implementasi *Genetic Algorithm* terdapat pada *file* genetic_algorithm.py

Tabel 3.24 Fungsi dan Kode __init__pada *Genetic Algorithm*

Nama Fungsi	__init__(self, population_size=50, mutation_rate=0.1, iterations=200, best_individuals=2, restart_interval=50)
Deskripsi Fungsi	Menginisialisasi populasi awal *Magic Cube*.
Kode	

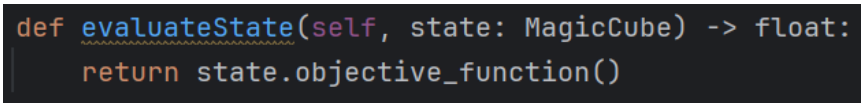
```
def __init__(self, population_size=50, mutation_rate=0.1, iterations=200, best_individuals=2, restart_interval=50):
    self.population_size = population_size
    self.mutation_rate = mutation_rate
    self.iterations = iterations
    self.best_individuals = best_individuals
    self.restart_interval = restart_interval
    self.population = self.startInit()  # Inisialisasi populasi awal
    self.objective_function_values = []  # Menyimpan nilai objective function per generasi
```

Tabel 3.25 Fungsi dan Kode startInit pada *Genetic Algorithm*

Nama Fungsi	startInit(self) → List[MagicCube]
Deskripsi Fungsi	Menginisialisasi populasi awal dengan sejumlah *Magic Cube* acak yang mengembalikan daftar individu dalam populasi awal.
Kode	

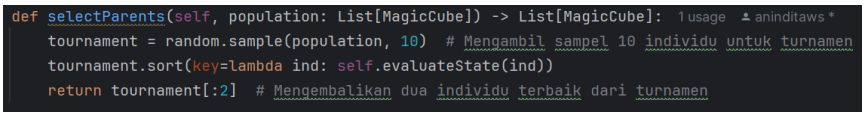
```
def startInit(self) -> List[MagicCube]:  2 usages  aninditaws *
    return [MagicCube(size=5) for _ in range(self.population_size)]
```

Tabel 3.26 Fungsi dan Kode evaluateState pada *Genetic Algorithm*

Nama Fungsi	evaluateState(self, state: MagicCube) → float
Deskripsi Fungsi	Mengevaluasi kualitas sebuah *Magic Cube* menggunakan *objective function*. Fungsi ini menerima sebuah *Magic Cube* yang akan dievaluasi dan mengembalikan nilai skor evaluasi, di mana semakin rendah skor, semakin baik kualitas solusi yang mendekati kondisi optimal.
Kode	

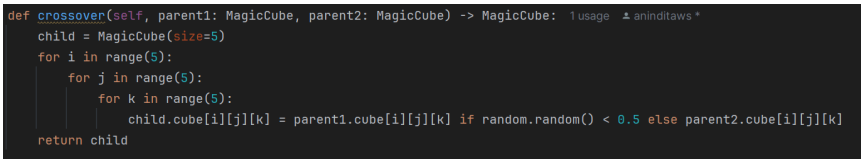
```
def evaluateState(self, state: MagicCube) -> float:
    return state.objective_function()
```

Tabel 3.27 Fungsi dan Kode selectParents pada *Genetic Algorithm*

Nama Fungsi	selectParents(self, population: List[MagicCube]) → List[MagicCube]
Deskripsi Fungsi	Memilih dua individu terbaik dari populasi sebagai induk menggunakan metode turnamen. Fungsi ini menerima daftar individu dalam populasi dan mengembalikan dua individu terpilih sebagai induk untuk proses reproduksi.
Kode	

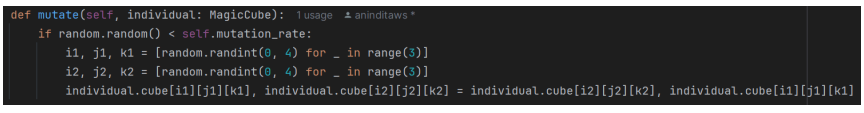
```
def selectParents(self, population: List[MagicCube]) -> List[MagicCube]:  1 usage  aninditaws *
    tournament = random.sample(population, 10)  # Mengambil sampel 10 individu untuk turnamen
    tournament.sort(key=lambda ind: self.evaluateState(ind))
    return tournament[:2]  # Mengembalikan dua individu terbaik dari turnamen
```

Tabel 3.28 Fungsi dan Kode crossover pada *Genetic Algorithm*

Nama Fungsi	crossover(self, parent1: MagicCube, parent2: MagicCube) → MagicCube
Deskripsi Fungsi	Melakukan *crossover* antara dua individu induk untuk menghasilkan keturunan baru. Fungsi ini menerima dua *Magic Cube* sebagai induk dan menghasilkan satu *Magic Cube* sebagai keturunan baru hasil dari proses *crossover*.
Kode	

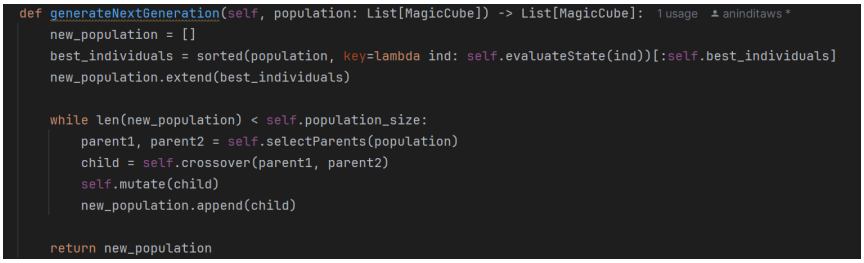
```
def crossover(self, parent1: MagicCube, parent2: MagicCube) -> MagicCube:  1 usage  aninditaws *
    child = MagicCube(size=5)
    for i in range(5):
        for j in range(5):
            for k in range(5):
                child.cube[i][j][k] = parent1.cube[i][j][k] if random.random() < 0.5 else parent2.cube[i][j][k]
    return child
```

Tabel 3.29 Fungsi dan Kode mutate pada *Genetic Algorithm*

Nama Fungsi	mutate(self, individual: MagicCube)
Deskripsi Fungsi	Melakukan mutasi pada *Magic Cube* dengan menukar dua elemen acak berdasarkan *Magic Cube* yang akan dimutasi.
Kode	

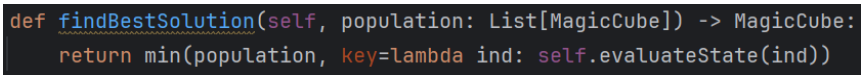
```
def mutate(self, individual: MagicCube):  1 usage  aninditaws *
    if random.random() < self.mutation_rate:
        i1, j1, k1 = [random.randint(0, 4) for _ in range(3)]
        i2, j2, k2 = [random.randint(0, 4) for _ in range(3)]
        individual.cube[i1][j1][k1], individual.cube[i2][j2][k2] = individual.cube[i2][j2][k2], individual.cube[i1][j1][k1]
```

Tabel 3.30 Fungsi dan Kode generateNextGeneration pada *Genetic Algorithm*

Nama Fungsi	generateNextGeneration(self, population: List[MagicCube]) → List[MagicCube]
Deskripsi Fungsi	Menghasilkan generasi baru dengan menggunakan seleksi, *crossover*, dan mutasi berdasarkan daftar individu dalam populasi saat ini dan menghasilkan populasi baru.
Kode	

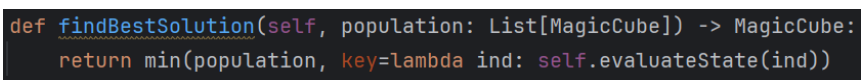
```
def generateNextGeneration(self, population: List[MagicCube]) -> List[MagicCube]:  1 usage  aninditaws *
    new_population = []
    best_individuals = sorted(population, key=lambda ind: self.evaluateState(ind))[:self.best_individuals]
    new_population.extend(best_individuals)

    while len(new_population) < self.population_size:
        parent1, parent2 = self.selectParents(population)
        child = self.crossover(parent1, parent2)
        self.mutate(child)
        new_population.append(child)

    return new_population
```

Tabel 3.31 Fungsi dan Kode findBestSolution pada *Genetic Algorithm*

Nama Fungsi	findBestSolution(self, population: List[MagicCube]) → MagicCube
Deskripsi Fungsi	Menemukan individu terbaik dalam populasi berdasarkan nilai evaluasi dari daftar individu dalam populasi dan mengembalikan individu terbaik dalam populasi.
Kode	

```
def findBestSolution(self, population: List[MagicCube]) -> MagicCube:
    return min(population, key=lambda ind: self.evaluateState(ind))
```

Tabel 3.32 Fungsi dan Kode run pada *Genetic Algorithm*

Nama Fungsi	run(self) → MagicCube
Deskripsi Fungsi	Menjalankan algoritma genetika untuk mencari solusi optimal yang menghasilkan solusi terbaik yang ditemukan berdasarkan seluruh generasi.
Kode	

```
def findBestSolution(self, population: List[MagicCube]) -> MagicCube:
    return min(population, key=lambda ind: self.evaluateState(ind))
```

I. Pendukung

Implementasi algoritma pendukung terdapat pada *file* magic_cube.py dan main.py

Tabel 3.33 Fungsi dan Kode __init__ pada *Magic Cube*

Nama Fungsi	__init__(self, size=5)
Deskripsi Fungsi	Menginisialisasi objek *Magic Cube* dengan ukuran 5×5×5 dengan nilai awal 0. Fungsi ini mengisi kubus dengan nilai acak dan menghitung nilai *magic number* yang diharapkan.

Kode	`def __init__(self, size=5):  jasonjahja`
	`    self.size = size`
	`    self.cube = [[[0 for _ in range(size)] for _ in range(size)] for _ in range(size)]`
	`    self.initialize_random_state()`
	`    self.magic_number = self.calculate_magic_number()`

Tabel 3.34 Fungsi dan Kode initialize_random_state pada *Magic Cube*

Nama Fungsi	initialize_random_state(self)
Deskripsi Fungsi	Mengisi *Magic Cube* dengan angka acak yang unik dari 1 hingga size³ sehingga setiap posisi dalam kubus memiliki nilai acak tanpa pengulangan.

Kode	`def initialize_random_state(self):  1 usage  jasonjahja`
	`    numbers = list(range(1, self.size ** 3 + 1))`
	`    random.shuffle(numbers)`
	
	`    index = 0`
	`    for i in range(self.size):`
	`        for j in range(self.size):`
	`            for k in range(self.size):`
	`                self.cube[i][j][k] = numbers[index]`
	`                index += 1`

Tabel 3.35 Fungsi dan Kode calculate_magic_number pada *Magic Cube*

Nama Fungsi	calculate_magic_number(self)
Deskripsi Fungsi	Menghitung nilai magic_number sebagai nilai target untuk penjumlahan komponen dalam kubus agar memenuhi kondisi magic cube.
Kode	`def calculate_magic_number(self):` 1 usage jasonjahja `    return (self.size * (self.size ** 3 + 1)) // 2`

Tabel 3.36 Fungsi dan Kode display pada *Magic Cube*

Nama Fungsi	display(self)
Deskripsi Fungsi	Menampilkan setiap layer cube ke layar

Kode	`def display(self):` 10 usages jasonjahja
	`    for layer in range(self.size):`
	`        print(f"Layer {layer + 1}:")`
	`        for element in self.cube[layer]:`
	`            print(" ".join(f"{num:3}" for num in element))`
	`        print("-" * 25)`

Tabel 3.37 Kode pada *file* main.py

Kode

```
from cube.magic_cube import MagicCube
from algorithms.steepest_hill_climbing import SteepestHillClimbing
from algorithms.sideways_move import SidewaysMove
from algorithms.stochastic import StochasticHillClimbing
from algorithms.random_restart import RandomRestartHillClimbing
from algorithms.genetic_algorithm import GeneticAlgorithmMagicCube
from algorithms.simulated_annealing import SimulatedAnnealing

# Variabel global
size = 5
algorithms = []

# Mengosongkan terminal
def clear_screen():  1 usage  jasonjahja
    os.system('cls' if os.name == 'nt' else 'clear')

# Pendefinisian warna
LIME_GREEN = "\033[38;5;149m"
TEAL = "\033[38;2;42;157;143m"
GOLDEN_YELLOW = "\033[38;2;233;196;106m"
ORANGE = "\033[38;2;244;162;97m"
RED_ORANGE = "\033[38;2;231;111;81m"
RESET_COLOR = "\033[0m"
```

```
# Shortcut text dengan warna
def separator(title):  11 usages  jasonjahja
    print("\n" + "=" * 50)
    print(TEAL + f"===== {title} =====" + RESET_COLOR)
    print("=" * 50 + RESET_COLOR)

def colorText(text, color):  36 usages  jasonjahja
    print(color + f"{text}" + RESET_COLOR)

# Fungsi pemanggilan algoritma
def run_steepest_hill_climbing():  1 usage  jasonjahja +1
    max_iterations = int(input("Masukkan jumlah maksimal iterasi yang diinginkan: "))
    magic_cube = MagicCube(size)
    steepest_initial_score = magic_cube.objective_function()
    colorText( text: "First state of magic cube:", LIME_GREEN)
    magic_cube.display()
    steepest_hill = SteepestHillClimbing(magic_cube)
    steepest_final_score, steepest_move_duration, steepest_move_iteration = steepest_hill.run(max_iterations)
    colorText( text: "\nState of magic cube after Steepest Hill Climbing:", LIME_GREEN)
    magic_cube.display()
    algorithms.append({
        "algorithm": "Steepest Hill Climbing",
        "initial_score": steepest_initial_score,
        "final_score": steepest_final_score,
        "delta": steepest_initial_score - steepest_final_score,
        "duration": steepest_move_duration,
        "iteration": steepest_move_iteration
    })
```

```python
def run_sideways_move_hill_climbing():  1 usage  jasonjahja +1
    max_iterations = int(input("Masukkan jumlah maksimal iterasi yang diinginkan: "))
    max_sideways_iteration = int(input("Masukkan jumlah maksimal iterasi sideways yang diinginkan: "))
    magic_cube = MagicCube(size)
    sideways_initial_score = magic_cube.objective_function()
    colorText( text: "First state of magic cube:", LIME_GREEN)
    magic_cube.display()
    sideways_climber = SidewaysMove(magic_cube)
    sideways_final_score, sideways_move_duration, sideways_move_iteration, most_sideways_move_iteration = sideways_climber.run(
    colorText( text: "\nState of magic cube after Sideways Move Hill Climbing:", LIME_GREEN)
    magic_cube.display()
    algorithms.append({
        "algorithm": "Sideways Move Hill Climbing",
        "initial_score": sideways_initial_score,
        "final_score": sideways_final_score,
        "delta": sideways_initial_score - sideways_final_score,
        "duration": sideways_move_duration,
        "iteration": sideways_move_iteration,
        "most_sideways_iteration": most_sideways_move_iteration
    })
```

```python
def run_stochastic_hill_climbing():  1 usage  jasonjahja +1
    max_iterations = int(input("Masukkan jumlah maksimal iterasi yang diinginkan: "))
    magic_cube = MagicCube(size)
    stochastic_initial_score = magic_cube.objective_function()
    colorText( text: "First state of magic cube:", LIME_GREEN)
    magic_cube.display()
    stochastic_climber = StochasticHillClimbing(magic_cube)
    stochastic_final_score, stochastic_duration, stochastic_iteration = stochastic_climber.run(max_iterations)
    colorText( text: "\nState of magic cube after Stochastic Hill Climbing:", LIME_GREEN)
    magic_cube.display()
    algorithms.append({
        "algorithm": "Stochastic Hill Climbing",
        "initial_score": stochastic_initial_score,
        "final_score": stochastic_final_score,
        "delta": stochastic_initial_score - stochastic_final_score,
        "duration": stochastic_duration,
        "iteration": stochastic_iteration
    })
```

```python
def run_random_restart():  1 usage  jasonjahja +1
    max_restart = int(input("Masukkan jumlah restart yang diinginkan: "))
    max_iterations_per_restart = int(input("Masukkan jumlah maksimal iterasi per restart: "))
    magic_cube = MagicCube(size)
    randomr_initial_score = magic_cube.objective_function()
    colorText( text: "First state of magic cube:", LIME_GREEN)
    magic_cube.display()
    random_restart = RandomRestartHillClimbing(magic_cube, max_restart, max_iterations_per_restart)
    randomr_final_score, randomr_duration, randomr_restart = random_restart.run(randomr_initial_score)
    colorText( text: "\nState of magic cube after Random Restart Hill Climbing:", LIME_GREEN)
    magic_cube.display()
    algorithms.append({
        "algorithm": "Random Restart Hill Climbing",
        "initial_score": randomr_initial_score,
        "final_score": randomr_final_score,
        "delta": randomr_initial_score - randomr_final_score,
        "duration": randomr_duration,
        "iteration": randomr_restart
    })
```

```python
def run_genetic_algorithm():  1 usage  jasonjahja
    magic_cube = MagicCube(size)
    population_size = int(input("Enter Population Size: "))
    iteration = int(input("Enter Number of Iterations: "))
    genetic_algorithm = GeneticAlgorithmMagicCube(population_size=population_size, mutation_rate=0.1,
                                                  iterations=iteration)
    colorText( text: "First state of magic cube:", LIME_GREEN)
    magic_cube.display()
    genetic_initial_score, genetic_final_score, duration = genetic_algorithm.run()
    colorText( text: "\nState of magic cube after Genetic Algorithm:", LIME_GREEN)
    magic_cube.display()
    algorithms.append(
        {
            "algorithm": "Genetic Algorithm",
            "initial_score": genetic_initial_score,
            "final_score": genetic_final_score,
            "delta": genetic_initial_score - genetic_final_score,
            "duration": duration,
            "population": population_size,
            "iteration": iteration
        }
    )
```

```python
def run_simulated_annealing():  1 usage  jasonjahja +1
    initial_temp = int(input("Masukkan temperatur awal yang diinginkan: "))
    final_temp = int(input("Masukkan temperatur akhir yang diinginkan: "))
    cooling_rate = int(input("Masukkan laju penurunan suhu yang diinginkan: "))
    threshold = int(input("Masukkan nilai threshold yang diinginkan: "))
    magic_cube = MagicCube(size)
    sim_initial_score = magic_cube.objective_function()
    colorText( text: "First state of magic cube:", LIME_GREEN)
    magic_cube.display()
    simulated_annealing = SimulatedAnnealing(magic_cube, initial_temp, final_temp, cooling_rate, threshold)
    sim_final_score, sim_duration, stuck_frequency, sim_iteration = simulated_annealing.run()
    colorText( text: "\nState of magic cube after Simulated Annealing Algorithm:", LIME_GREEN)
    magic_cube.display()
    algorithms.append(
        {
            "algorithm": "Simulated Annealing Algorithm",
            "initial_score": sim_initial_score,
            "final_score": sim_final_score,
            "delta": sim_initial_score - sim_final_score,
            "duration": sim_duration,
            "iteration": sim_iteration,
            "stuck frequency": stuck_frequency
        }
    )
```

```python
def run_comparison_algorithms():  1 usage  jasonjahja
    # Perbandingan tiap algoritma
    best_algorithm_delta = max(algorithms, key=lambda algo: algo["delta"])
    worst_algorithm_delta = min(algorithms, key=lambda algo: algo["delta"])
    fastest_algorithm_duration = min(algorithms, key=lambda algo: algo["duration"])
    slowest_algorithm_duration = max(algorithms, key=lambda algo: algo["duration"])

    separator("Algoritma dengan perkembangan paling besar")
    colorText( text: f"\n🔹 Algoritma: {best_algorithm_delta['algorithm']}", GOLDEN_YELLOW)
    colorText( text: f"   • Delta: {best_algorithm_delta['delta']}", ORANGE)
    colorText( text: f"   • Skor Final: {best_algorithm_delta['final_score']}", ORANGE)

    separator("Algoritma dengan perkembangan paling kecil")
    colorText( text: f"\n🔹 Algoritma: {worst_algorithm_delta['algorithm']}", GOLDEN_YELLOW)
    colorText( text: f"   • Delta: {worst_algorithm_delta['delta']}", ORANGE)
    colorText( text: f"   • Skor Final: {worst_algorithm_delta['final_score']}", ORANGE)

    separator("Algoritma dengan perkembangan paling cepat")
    colorText( text: f"\n🔹 Algoritma: {fastest_algorithm_duration['algorithm']}", GOLDEN_YELLOW)
    colorText( text: f"   • Durasi: {fastest_algorithm_duration['duration']:.4f} seconds", ORANGE)

    separator("Algoritma dengan perkembangan paling lambat")
    colorText( text: f"\n🔹 Algoritma: {slowest_algorithm_duration['algorithm']}", GOLDEN_YELLOW)
    colorText( text: f"   • Durasi: {slowest_algorithm_duration['duration']:.4f} seconds", ORANGE)
```

```
clear_screen()
colorText( text: "Welcome to the Magic Cube Solver!\n", GOLDEN_YELLOW)
colorText( text: "Pilihan algoritma yang tersedia:", TEAL)

algo_names = ["Steepest Hill Climbing", "Sideways Move Hill Climbing", "Stochastic Hill Climbing", "Random Restart Hill Climbing"
i = 1
for algon in algo_names:
    colorText( text: f"{i}. {algon}", LIME_GREEN)
    i += 1

colorText( text: f"{i}. Semua Algoritma (Tambahan Info Perbandingan)", LIME_GREEN)

choice = input(TEAL + "Masukkan algoritma yang ingin dicoba (1/2/3/4/5/6/7): " + RESET_COLOR)
```

```
# Pemilihan algoritma
if "1" in choice or "7" in choice:
    separator("Menjalankan Steepest Hill Climbing")
    run_steepest_hill_climbing()

if "2" in choice or "7" in choice:
    separator("Menjalankan Sideways Move Hill Climbing")
    run_sideways_move_hill_climbing()

if "3" in choice or "7" in choice:
    separator("Menjalankan Stochastic Hill Climbing")
    run_stochastic_hill_climbing()

if "4" in choice or "7" in choice:
    separator("Menjalankan Random Restart Hill Climbing")
    run_random_restart()

if "5" in choice or "7" in choice:
    separator("Menjalankan Genetic Algorithm")
    run_genetic_algorithm()

if "6" in choice or "7" in choice:
    separator("Menjalankan Simulated Annealing")
    run_simulated_annealing()
```

```
# Pemaparan informasi setiap algoritma
separator("Informasi Algoritma Terpilih")
for algo in algorithms:
    colorText( text: f"\n 🔹 Algoritma: {algo['algorithm']}", GOLDEN_YELLOW)
    colorText( text: f"   • Skor inisial: {algo['initial_score']}", ORANGE)
    colorText( text: f"   • Skor final: {algo['final_score']}", ORANGE)
    colorText( text: f"   • Delta: {algo['delta']}", ORANGE)
    colorText( text: f"   • Durasi: {algo['duration']:.4f} seconds", ORANGE)
    if algo['algorithm'] == "Genetic Algorithm":
        colorText( text: f"   • Populasi: {algo['population']}", ORANGE)

    colorText( text: f"   • Iterasi: {algo['iteration']}", ORANGE)
    if algo['algorithm'] == "Sideways Move Hill Climbing":
        colorText( text: f"   • Iterasi sideways terbanyak: {algo['most_sideways_iteration']}", ORANGE)
    elif algo['algorithm'] == "Genetic Algorithm":
        colorText( text: f"   • Populasi: {algo['population']}", ORANGE)
    elif algo['algorithm'] == "Simulated Annealing":
        colorText( text: f"   • Stuck Frequency: {algo['stuck_frequency']}", ORANGE)

# Menampilkan informasi perbandingan jika 4 dipilih
if "7" in choice:
    run_comparison_algorithms()
```

Hasil Eksperimen dan Analisis

Bagian ini menjelaskan hasil eksperimen dan analisis dari penerapan teori ke dalam fungsi-fungsi yang dikembangkan dalam kode untuk menyelesaikan kasus *Diagonal Magic Cube* pada tugas ini.

A. Eksperimen

Berikut ini adalah hasil eksperimen dari setiap algoritma yang diterapkan pada kasus *Diagonal Magic Cube*.

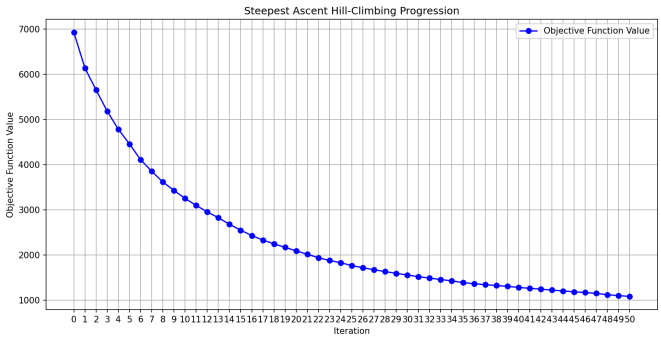
a. Steepest Ascent Hill-climbing

Berikut merupakan hasil eksperimen *Steepest Ascent Hill-climbing.*

Tabel 4.1 Eksperimen 1 *Steepest Ascent Hill-climbing*

Jumlah Iterasi	50
***State Cube* Awal**	Layer 1: 61 31 119 60 39 100 103 86 80 102 38 75 69 29 12 77 91 63 88 96 70 43 44 110 4 ---------- Layer 2: 22 122 26 125 121 76 42 33 7 30 93 118 27 8 71 28 74 87 82 6 114 67 112 81 23 ---------- Layer 3: 9 59 89 54 120 49 20 16 72 73 48 109 37 95 40 108 101 124 32 113 3 35 18 98 123 ---------- Layer 4: 45 111 17 107 58 99 34 62 79 65 11 66 1 84 97 25 116 115 46 47 56 64 13 92 105 ----------

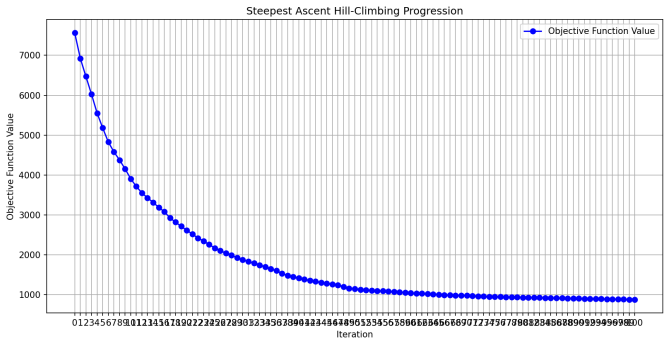
	Layer 5: 53 117 94 2 78 36 106 85 104 10 15 19 68 52 14 41 83 55 21 50 90 51 5 24 57 ----------
***Objective Function* Awal**	6929
***State Cube* Akhir**	Layer 1: 74 20 119 60 39 47 75 12 80 102 48 103 76 13 78 77 24 31 88 96 70 89 81 72 4 ---------- Layer 2: 22 15 41 125 112 69 100 9 19 114 93 118 42 8 62 28 61 99 117 6 104 18 121 44 23 ---------- Layer 3: 120 64 43 54 33 49 1 124 110 26 38 52 66 95 63 108 101 16 32 58 3 98 79 21 113 ---------- Layer 4: 53 111 17 82 50 116 34 71 30 65 11 37 83 84 97 73 87 115 35 27 56 51 29 85 105 ---------- Layer 5: 45 107 94 2 86 36 106 92 67 10 122 7 68 109 14 25 40 55 46 123 90 59 5 91 57 ----------
Objective	1079

***Function* Akhir**	
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	126.4842 *seconds*

Tabel 4.2 Eksperimen 2 *Steepest Ascent Hill-climbing*

Jumlah Iterasi	100
***State Cube* Awal**	Layer 1: 81 102 68 31 16 90 110 24 6 62 99 46 121 32 76 9 79 123 51 104 7 59 33 106 58 --- Layer 2: 114 72 117 96 40 83 35 118 15 47 37 48 84 49 50 87 74 29 103 41 22 67 20 3 2 --- Layer 3: 69 60 30 75 5 78 38 4 56 42 124 86 8 120 10 53 113 57 19 1 63 54 125 71 34 --- Layer 4: 61 95 116 92 93 14 66 13 36 88 108 28 55 89 107 112 109 11 21 111 45 98 52 43 26 ---

	Layer 5: 73 85 17 115 27 12 100 101 80 91 25 82 44 70 119 64 122 18 23 39 97 105 94 77 65 ----------
***Objective Function* Awal**	7563
***State Cube* Akhir**	Layer 1: 81 102 64 29 39 119 16 1 117 68 85 47 121 6 56 23 31 101 52 108 8 120 34 106 44 ---------- Layer 2: 2 76 80 122 35 93 33 105 11 65 24 48 86 98 59 88 74 27 84 42 109 69 20 3 114 ---------- Layer 3: 62 60 26 75 96 41 95 87 58 36 73 38 37 90 77 124 112 49 21 9 12 22 115 71 97 ---------- Layer 4: 61 5 82 78 91 30 66 111 19 89 104 99 55 50 7 13 45 15 125 113 107 100 51 43 14 ---------- Layer 5: 116 72 63 10 54 28 103 17 110 57 25 83 18 70 118 67 53 123 32 40 79 4 94 92 46 ----------
Objective	878

***Function* Akhir**	
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	151.1143 *seconds*

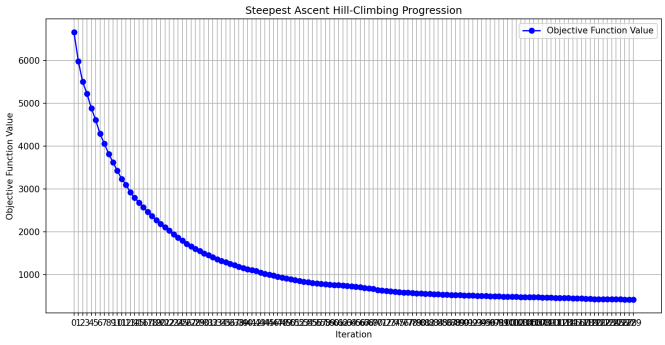
Tabel 4.3 Eksperimen 3 *Steepest Ascent Hill-climbing*

Jumlah Iterasi	150
***State Cube* Awal**	Layer 1: 99 71 5 10 50 8 81 104 54 23 110 118 117 83 94 21 44 125 15 46 57 20 30 12 78 --- Layer 2: 19 75 70 18 27 40 11 13 26 103 31 93 116 42 32 111 100 36 73 89 101 53 122 48 35 --- Layer 3: 28 2 65 68 17 87 76 91 38 6 79 1 34 80 95 108 90 113 25 64 61 58 74 4 16 --- Layer 4: 55 107 119 86 105 7 115 84 37 41 43 88 102 51 120 92 98 72 114 24 123 39 3 45 56 ---

	Layer 5: 106 97 60 121 29 112 67 66 47 96 62 52 14 124 9 33 85 59 77 49 63 109 22 82 69 ----------
***Objective Function* Awal**	6653

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

Objective	423

***Function* Akhir**	
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	343.1270 *seconds*

b. Hill-climbing with Sideways Move

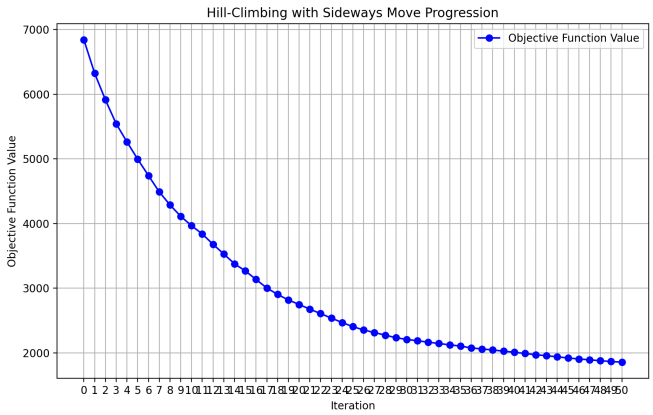
Berikut merupakan hasil eksperimen *Hill-climbing with Sideways Move*.

Tabel 4.4 Eksperimen 1 *Hill-climbing with Sideways Move*

Jumlah Iterasi	50
Max Sideways Move	5
***State Cube* Awal**	Layer 1: 75 45 49 16 40 39 68 97 28 108 59 43 34 84 103 94 93 31 7 112 105 19 64 98 4 -------------------- Layer 2: 85 60 11 83 14 12 22 114 86 122 33 125 41 26 78 99 35 66 15 17 124 95 74 53 25 -------------------- Layer 3: 73 13 104 118 24 23 89 32 72 80 10 106 9 109 111 101 107 57 52 87 123 30 81 117 55 --------------------

	Layer 4: 6 90 2 37 3 110 76 50 67 69 44 18 88 46 77 71 115 92 70 116 61 62 113 119 54 ---------- Layer 5: 100 82 63 47 27 36 91 21 65 96 48 58 20 8 56 38 102 51 5 1 29 79 120 42 121 ----------
***Objective Function* Awal**	6842

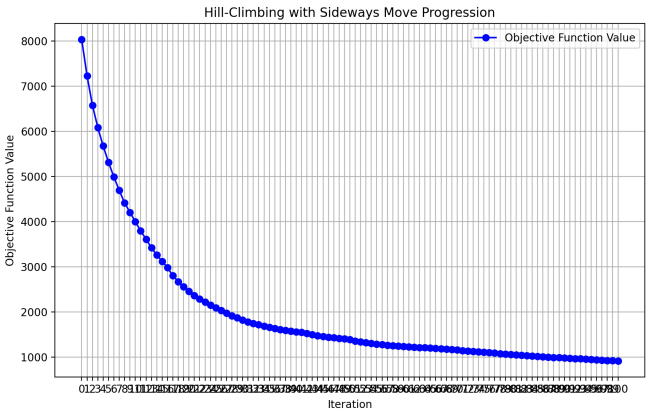
***State Cube* Akhir**	Layer 1:
	 75 67 23 32 115
	108 68 97 28 14
	 59 43 77 107 34
	 94 93 9 7 112
	  4 19 122 98 89
	---------- 	
	Layer 2:
	 85 44 11 83 111
	 12 22 121 100 39
	 33 120 25 58 78
	 99 35 73 20 87
	 90 95 74 53 3
	---------- 	
	Layer 3:
	 66 10 104 118 24
	 41 114 16 72 80
	 13 106 31 109 54
	 15 45 103 52 101
	123 60 70 5 55
	---------- 	
	Layer 4:
	  6 117 113 37 49
	110 76 21 50 61
	 84 18 88 36 91
	 71 40 92 81 17
	 69 62 2 116 64
	---------- 	
	Layer 5:

	86 82 63 47 27 46 30 57 65 119 125 26 96 8 56 38 102 51 124 1 29 79 48 42 105 ----------
***Objective Function* Akhir**	1857
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	67.788 *seconds*

Tabel 4.5 Eksperimen 2 *Hill-climbing with Sideways Move*

Jumlah Iterasi	100
Max Sideways Move	10
***State Cube* Awal**	Layer 1: 70 2 71 88 16 18 38 87 111 86 47 48 34 54 59 96 85 58 61 32 103 118 104 110 121 ---------- Layer 2: 65 50 77 99 80 46 124 112 92 74 37 23 115 75 1 52 91 33 13 19

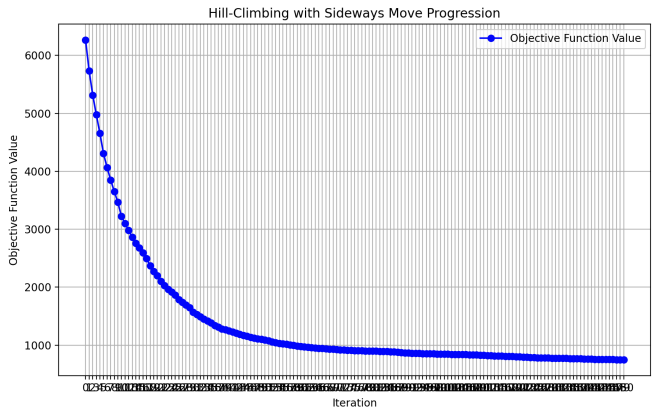
	39 119 97 93 7 --- Layer 3: 64 5 67 83 12 15 51 125 36 68 76 117 62 55 14 11 84 100 78 69 6 109 25 45 3 --- Layer 4: 28 4 29 98 57 73 123 43 24 22 27 82 9 89 30 35 90 42 102 8 20 44 94 116 53 --- Layer 5: 120 114 106 31 21 17 101 26 40 107 66 63 108 79 41 49 105 81 95 113 122 72 60 10 56 ---
***Objective Function* Awal**	8035
***State Cube* Akhir**	Layer 1: 70 118 59 51 16 18 17 89 107 85 65 111 71 14 55 96 66 75 38 40 68 2 26 105 117 --- Layer 2: 121 35 1 104 67 43 41 103 39 95 92 28 113 45 37 52 91 33 27 110 3 119 97 90 7 --- Layer 3: 58 21 82 32 124 125 50 64 24 49 76 23 62 100 54 48 99 84 73 8 6 122 25 83 80

	---------- Layer 4: 34 19 86 98 78 112 116 46 36 5 20 93 9 77 106 12 72 42 102 87 123 15 120 13 44 ---------- Layer 5: 30 109 115 31 29 11 88 10 108 101 61 56 60 79 69 94 4 81 74 53 114 57 47 22 63 ----------
***Objective Function* Akhir**	913
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	128.2789 *seconds*

Tabel 4.5 Eksperimen 3 *Hill-climbing with Sideways Move*

Jumlah Iterasi	150
Max Sideways Move	15
***State Cube* Awal**	Layer 1: 118 46 70 7 34 91 8 24 69 88 108 19 104 100 107

	63 114 57 1 12 50 53 48 121 28 --- Layer 2: 98 13 21 123 106 54 68 117 51 78 40 27 122 6 119 44 9 61 89 124 81 59 125 90 66 --- Layer 3: 25 84 73 111 72 29 105 76 2 15 35 92 65 67 62 11 83 52 97 64 96 109 87 16 4 --- Layer 4: 36 23 31 26 93 82 38 85 74 55 37 22 45 43 17 86 113 115 80 39 5 56 102 3 77 --- Layer 5: 20 112 33 101 75 116 60 18 94 120 47 71 49 110 14 103 32 30 42 10 58 99 95 79 41 ---
***Objective Function* Awal**	6266
***State Cube* Akhir**	Layer 1: 88 65 89 6 67 8 111 34 72 90 108 1 20 98 95 61 102 123 19 10 50 46 47 118 52 --- Layer 2: 81 96 11 97 30 29 83 119 49 32 40 28 115 7 122 64 21 37 73 120

	100 87 27 92 9 -------------------- Layer 3: 59 15 69 103 70 105 24 71 5 110 56 78 63 58 66 3 84 54 125 44 93 114 62 22 25 -------------------- Layer 4: 36 23 41 109 106 91 38 75 76 35 85 124 43 45 18 86 77 33 80 39 17 53 121 4 117 -------------------- Layer 5: 51 116 104 2 42 82 60 12 113 48 26 94 74 107 14 101 31 68 16 99 55 13 57 79 112 --------------------
***Objective Function* Akhir**	748
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	182.1754 *seconds*

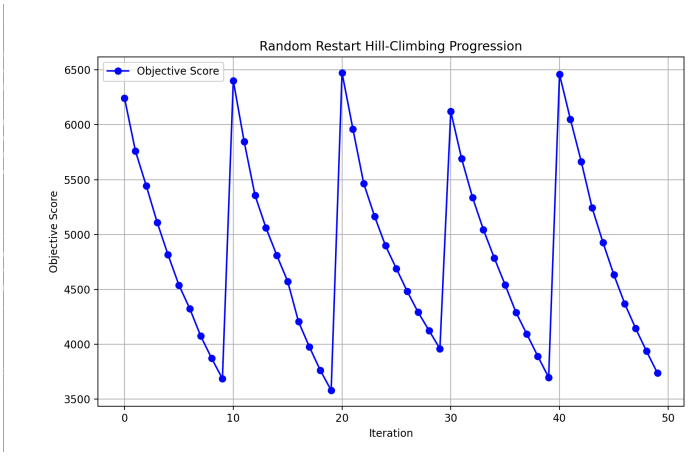
c. *Random Restart Hill-climbing*

Berikut merupakan hasil eksperimen *Random Restart Hill-climbing*.

Tabel 4.7 Eksperimen 1 *Random-Restart Hill-climbing*

Jumlah *Restart*	5
Jumlah Iterasi per *Restart*	10
***State Cube* Awal**	Layer 1: 46 67 89 16 77 100 25 50 8 62 124 76 61 73 19 123 80 111 121 79 29 78 69 118 112 ---------- Layer 2: 48 87 119 26 24 13 96 99 85 53 60 70 33 35 18 56 104 32 68 113 40 108 92 36 1 ---------- Layer 3: 94 71 39 44 64 116 83 59 102 115 43 114 122 54 15 101 21 22 2 72 103 57 17 10 105 ---------- Layer 4: 38 37 75 88 125 95 5 6 63 65 93 81 47 66 27 9 30 49 55 98 82 51 4 91 20 ---------- Layer 5: 12 110 106 34 97 109 117 28 14 84 42 90 58 120 3 45 7 11 86 31 107 23 52 41 74 ----------
***Objective Function* Awal**	6590
***State Cube* Akhir**	Layer 1:

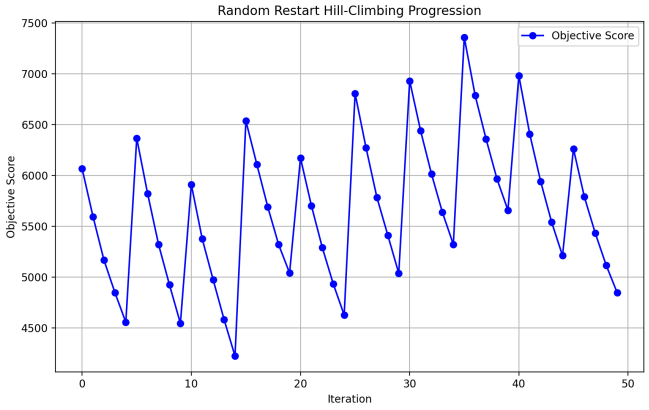
	123 105 9 54 102 50 73 107 42 66 94 32 48 124 13 30 59 91 20 72 29 89 52 56 40 ---------- Layer 2: 119 125 78 24 15 4 3 69 67 99 80 77 44 10 109 106 70 1 114 16 8 41 85 100 88 ---------- Layer 3: 90 65 46 108 63 84 37 110 6 61 112 49 83 19 76 2 111 17 64 113 36 87 117 82 21 ---------- Layer 4: 26 7 71 101 92 115 51 31 45 62 28 122 68 38 39 55 86 98 79 12 43 22 5 121 118 ---------- Layer 5: 25 34 116 81 97 14 96 11 95 104 93 18 58 120 75 27 47 74 53 103 35 60 57 33 23 ----------
***Objective Function* Akhir**	3578

***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	85.5827 *seconds*

Tabel 4.7 Eksperimen 2 *Random-Restart Hill-climbing*

Jumlah *Restart*	10
Jumlah Iterasi per *Restart*	5
***State Cube* Awal**	Layer 1: 74 4 113 120 58 62 73 9 80 23 98 11 46 49 122 53 38 72 99 92 67 100 87 104 71 --- Layer 2: 1 21 18 12 25 60 89 96 16 3 107 19 56 81 114 15 108 7 24 118 48 41 37 40 44 --- Layer 3: 111 57 26 119 82 5 102 39 36 83 28 97 124 42 117 50 63 125 47 13 32 61 86 8 43 --- Layer 4: 30 79 35 112 64 84 70 54 14 33

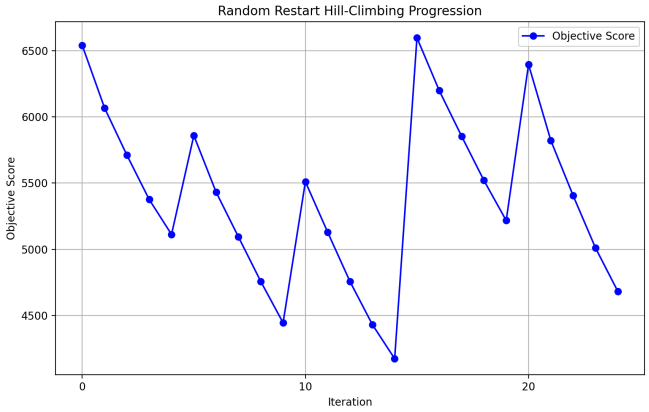
	78 34 110 52 75 2 88 115 68 94 69 51 66 27 90 ---------- Layer 5: 109 95 6 45 20 85 105 101 65 106 116 17 10 91 76 123 77 22 93 29 59 121 31 103 55 ----------
***Objective Function* Awal**	7005
***State Cube* Akhir**	Layer 1: 69 47 56 57 107 37 34 75 16 82 46 87 28 91 21 48 19 97 109 7 94 27 39 67 100 ---------- Layer 2: 93 2 83 73 71 106 31 110 30 44 17 108 25 32 90 121 77 49 85 35 12 41 88 64 103 ---------- Layer 3: 111 20 120 55 10 5 81 43 86 45 122 98 79 9 58 36 105 23 22 124 59 115 38 116 15 ---------- Layer 4: 11 8 104 6 53 80 50 62 1 101 119 123 66 51 74 63 89 13 65 92 96 4 42 125 61 ---------- Layer 5: 14 76 26 84 118 113 95 70 60 114

	40 72 102 18 78 68 29 99 117 3 52 33 54 112 24 ----------
***Objective Function* Akhir**	4226
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	64.9209 *seconds*

Tabel 4.8 Eksperimen 3 *Random-Restart Hill-climbing*

Jumlah *Restart*	5
Jumlah Iterasi per *Restart*	5
***State Cube* Awal**	Layer 1: 76 121 64 54 22 9 43 11 84 78 8 61 112 40 44 81 69 15 23 41 63 51 67 29 110 ---------- Layer 2: 107 96 13 74 37 36 114 99 100 117 122 113 103 88 82 2 59 83 106 90 35 70 91 52 89 ---------- Layer 3:

	3 109 115 32 34 86 45 33 93 4 118 55 124 105 65 87 62 14 12 7 27 38 39 75 1 ---------- Layer 4: 71 94 77 53 60 125 6 120 31 46 50 104 21 48 79 58 68 101 17 47 85 97 28 49 20 ---------- Layer 5: 5 111 66 98 95 57 42 92 16 80 30 72 26 123 108 10 73 25 18 24 19 56 102 119 116 ----------
***Objective Function* Awal**	7153
***State Cube* Akhir**	Layer 1: 66 110 1 84 10 96 68 13 112 59 60 34 70 32 119 106 20 90 56 51 75 30 85 43 36 ---------- Layer 2: 78 54 87 57 118 65 38 120 2 19 50 97 77 45 11 31 23 83 116 113 35 61 4 74 91 ---------- Layer 3: 117 14 123 53 79 33 99 3 107 49 62 98 41 21 40 15 94 72 46 105 80 37 81 92 17 ---------- Layer 4:

	5 8 73 86 39 63 25 44 104 100 103 88 42 18 52 58 24 76 124 48 115 47 111 71 114 -------------------- Layer 5: 108 122 26 16 22 82 55 64 93 101 9 27 89 125 67 109 29 12 6 102 95 121 28 69 7 --------------------
***Objective Function* Akhir**	4174
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	41.5853 *seconds*

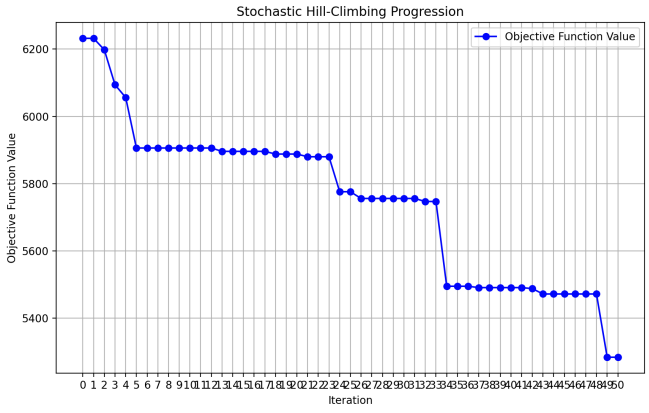
d. Stochastic Hill-climbing

Berikut merupakan hasil eksperimen *Stochastic Hill-climbing.*

Tabel 4.10 Eksperimen 1 *Stochastic Hill-climbing*

Jumlah Iterasi	50
***State Cube* Awal**	Layer 1: 22 64 105 65 98 20 109 41 5 100 68 35 56 87 89

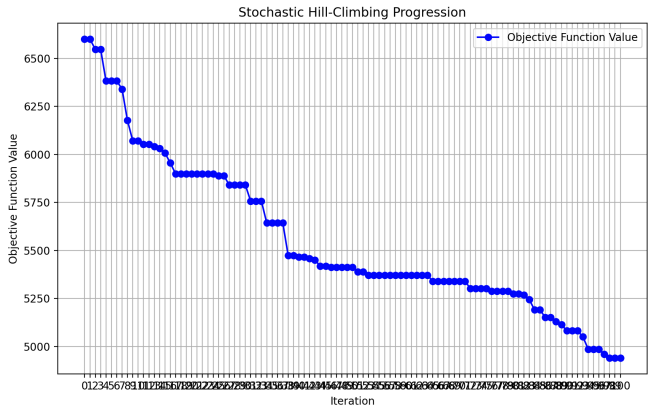
	76 99 50 40 14 101 117 26 37 13 --- Layer 2: 121 118 77 102 63 97 58 49 24 19 92 80 42 6 88 67 112 54 46 115 59 30 113 27 43 --- Layer 3: 120 53 125 85 94 23 110 86 73 74 4 111 45 70 29 103 11 72 52 9 16 75 3 93 71 --- Layer 4: 62 7 47 1 96 51 61 119 34 123 114 28 60 90 81 84 69 39 15 91 116 108 32 78 36 --- Layer 5: 25 66 21 38 2 8 17 18 83 57 95 10 107 124 33 12 48 104 79 55 31 82 106 44 122 ---
***Objective Function* Awal**	6232
***State Cube* Akhir**	Layer 1: 22 64 105 65 98 20 109 47 5 100 68 35 56 87 125 29 78 50 110 14 101 117 26 37 13 --- Layer 2: 121 118 77 102 67 97 58 49 76 19 92 80 42 6 88 63 112 54 46 115

	116 30 113 27 43 -------------------- Layer 3: 120 53 89 85 94 23 15 86 73 74 75 111 45 70 25 103 11 124 12 61 16 4 3 93 71 -------------------- Layer 4: 62 7 41 1 96 51 9 119 34 123 114 28 60 90 81 84 69 39 40 91 59 108 48 99 36 -------------------- Layer 5: 24 66 21 38 2 8 17 18 83 57 95 10 107 72 33 52 32 104 79 55 31 82 106 44 122 --------------------
***Objective Function* Akhir**	5284
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	0.0543 *seconds*

Tabel 4.11 Eksperimen 2 *Stochastic Hill-climbing*

Jumlah Iterasi	100

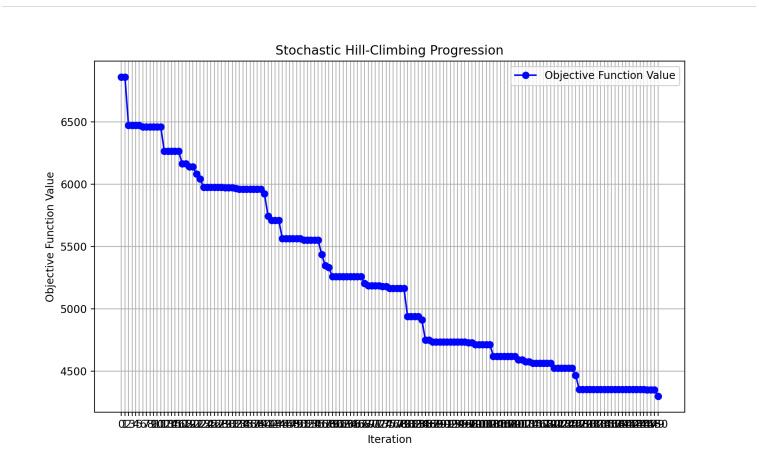
***State Cube* Awal**	Layer 1: 19 44 35 123 1 114 111 38 46 84 62 117 50 87 83 40 45 90 27 13 72 58 92 28 113 --- Layer 2: 11 70 55 49 65 33 47 86 3 66 81 103 26 24 2 107 60 122 31 59 53 5 16 20 29 --- Layer 3: 85 6 106 93 79 10 42 56 91 120 14 96 74 95 82 37 109 99 67 54 89 61 41 124 101 --- Layer 4: 25 88 64 43 23 110 4 115 12 68 15 108 100 32 119 63 57 98 118 39 73 52 105 112 34 --- Layer 5: 30 71 36 51 116 7 97 125 104 9 76 75 78 22 8 48 18 102 17 69 121 94 21 77 80 ---
***Objective Function* Awal**	6600
***State Cube* Akhir**	Layer 1: 19 44 101 123 16 114 111 38 46 9 62 117 50 87 91 8 100 41 27 13 98 49 92 11 113 --- Layer 2:

	34 122 55 112 65 33 24 77 36 116 81 103 63 47 2 107 60 58 31 26 53 5 1 37 29 --- Layer 3: 72 6 106 93 79 120 42 67 73 10 14 96 74 95 23 20 76 99 56 54 89 61 35 51 82 --- Layer 4: 115 88 64 15 90 110 4 125 12 57 43 108 45 17 102 59 68 85 118 39 40 52 105 84 28 --- Layer 5: 30 71 3 124 66 7 97 25 104 70 109 75 78 22 83 48 18 119 32 69 121 94 21 86 80 ---
***Objective Function* Akhir**	4940
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	0.0876 *seconds*

Tabel 4.12 Eksperimen 3 *Stochastic Hill-climbing*

Jumlah Iterasi	150
***State Cube* Awal**	Layer 1: 104 102 118 5 87 33 119 115 110 43 7 35 120 69 117 83 46 48 39 52 54 49 45 50 79 ---------- Layer 2: 103 42 60 19 124 22 58 21 65 37 9 71 55 73 78 113 98 116 32 77 15 2 44 89 90 ---------- Layer 3: 11 3 25 6 57 125 80 70 100 85 76 17 64 26 23 122 86 51 27 74 109 4 108 92 121 ---------- Layer 4: 106 111 38 75 91 12 95 93 59 94 40 107 97 112 16 105 56 30 62 114 81 68 14 101 13 ---------- Layer 5: 63 82 84 8 47 72 67 41 1 24 28 88 66 18 123 36 10 53 34 96 29 61 99 20 31 ----------
***Objective Function* Awal**	6859
***State Cube* Akhir**	Layer 1: 97 54 118 16 87 33 119 115 110 59 18 113 4 69 8

	71 46 48 84 52 70 38 45 50 79 ---------- Layer 2: 103 42 68 30 98 22 102 2 65 37 120 9 21 73 78 35 116 124 32 72 15 55 44 89 92 ---------- Layer 3: 11 88 25 80 57 125 6 56 100 85 76 17 64 58 60 67 114 51 83 14 109 34 108 23 121 ---------- Layer 4: 81 111 3 75 91 12 95 93 43 94 36 107 104 112 5 105 62 19 26 86 106 27 74 101 13 ---------- Layer 5: 63 82 39 117 47 77 122 41 1 24 28 49 66 7 123 40 10 53 90 96 29 61 99 20 31 ----------
***Objective Function* Akhir**	4300

***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	0.0870 *seconds*

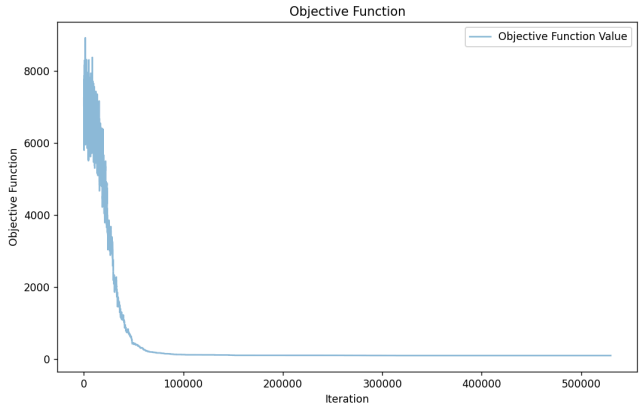
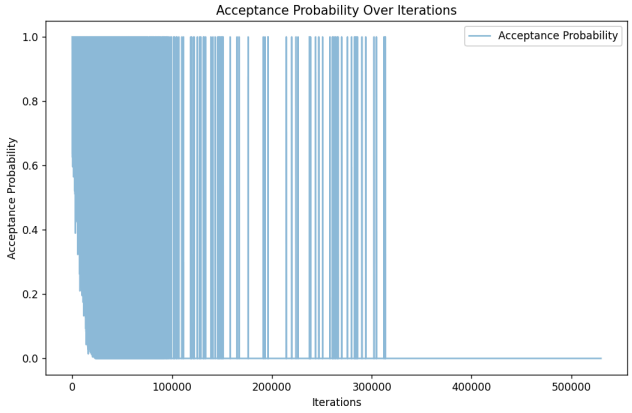
e. *Simulated Annealing*

Berikut merupakan hasil eksperimen *Simulated Annealing.*

Tabel 4.13 Eksperimen 1 *Simulated Annealing*

Temperatur Awal	1000
Temperatur Akhir	1e-20
Cooling Rate	0.9999
Threshold	0.5
***State Cube* Awal**	Layer 1: 48 104 28 13 87 40 55 42 91 37 125 20 23 89 86 62 38 83 75 56 22 80 74 82 95 --- Layer 2: 53 63 68 5 109 29 47 121 52 94 31 19 58 61 21 54 105 14 33 36 85 71 44 117 4 --- Layer 3: 124 16 6 92 60 84 15 9 90 59

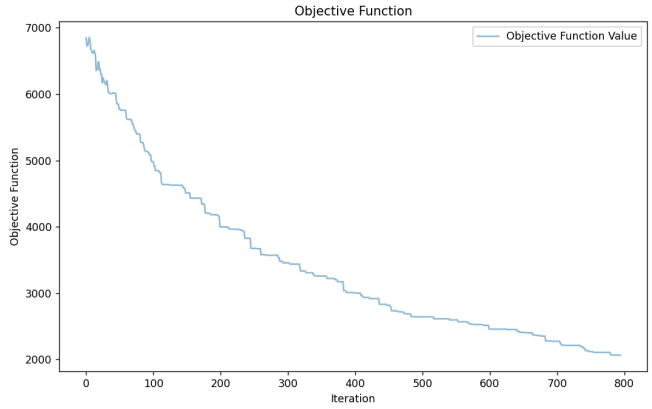
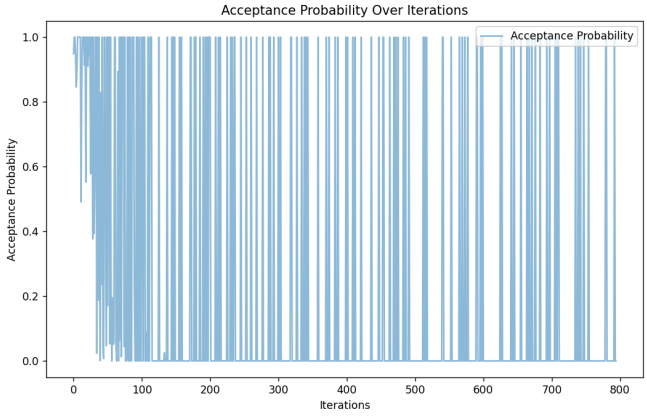
	78 101 79 110 43 35 17 69 93 77 107 114 81 10 106 ---------- Layer 4: 64 111 122 2 50 65 3 73 112 118 103 113 88 72 25 100 76 32 102 57 7 45 99 98 49 ---------- Layer 5: 39 97 108 1 96 119 116 115 120 51 30 26 34 66 27 8 12 70 46 67 41 11 18 24 123 ----------
***Objective Function* Awal**	7237
***State Cube* Akhir**	Layer 1: 69 85 14 110 37 118 81 16 1 100 25 11 112 56 111 44 105 90 29 50 60 33 83 119 20 ---------- Layer 2: 38 18 122 40 97 31 47 124 74 39 70 114 48 5 77 121 54 15 103 22 55 82 6 93 80 ---------- Layer 3: 88 23 49 61 94 109 46 53 71 36 35 99 62 27 92 17 21 73 115 89 66 125 78 41 4 ---------- Layer 4: 32 102 68 101 13 12 43 91 63 106 72 24 84 107 28 123 79 19 42 51

	76 67 52 2 116 --- Layer 5: 87 86 64 3 75 45 98 30 108 34 113 65 9 120 7 10 58 117 26 104 57 8 96 59 95 ---
***Objective Function* Akhir**	90
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
***Plot* $e^{\frac{\Delta E}{T}}$ terhadap Banyak Iterasi yang Telah Dilewati**	
Jumlah Iterasi	529569
Frekuensi *Stuck* di *Local Optima*	503738
Durasi	52.203319 *seconds*

Tabel 4.14 Eksperimen 2 *Simulated Annealing*

Temperatur Awal	500
Temperatur Akhir	1e-15
Cooling Rate	0.95
Threshold	0.5
***State Cube* Awal**	Layer 1: 17 21 103 62 104 44 67 73 114 33 93 117 53 110 86 120 63 109 91 46 48 119 122 87 125 --- Layer 2: 8 100 77 96 78 50 37 107 115 3 42 11 70 54 71 85 35 29 84 60 68 12 23 112 106 --- Layer 3: 26 97 10 57 101 121 82 41 99 124 27 34 80 64 56 1 15 79 113 6 61 89 13 24 111 --- Layer 4: 43 58 20 88 52 95 2 90 76 108 75 83 74 39 40 30 81 36 98 19 102 59 66 105 116 --- Layer 5: 38 22 32 72 28 69 94 7 92 47 51 9 25 31 49 4 118 45 65 14 18 123 16 5 55 ---
Objective	6822

***Function* Awal**	
***State Cube* Akhir**	Layer 1: 79 7 60 62 93 18 72 26 107 17 119 103 15 84 68 44 54 96 27 104 50 82 110 14 37 ---------- Layer 2: 25 108 113 9 10 33 85 16 97 114 53 61 120 5 95 99 36 13 88 69 125 12 52 112 11 ---------- Layer 3: 31 20 42 117 105 121 80 90 19 6 64 55 51 98 56 24 91 73 40 35 59 67 46 30 111 ---------- Layer 4: 65 92 38 58 81 32 2 94 76 115 77 124 78 8 28 123 70 86 71 1 29 43 22 116 109 ---------- Layer 5: 75 101 45 63 34 102 48 41 21 89 66 3 74 122 49 4 118 47 83 106 57 87 100 23 39 ----------
***Objective Function* Akhir**	2064

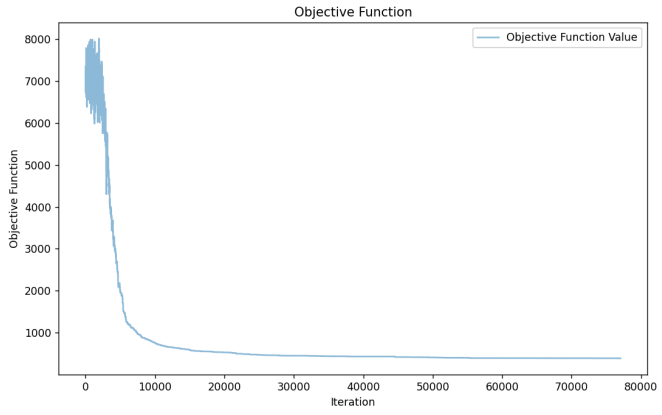
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Plot $e^{\frac{\Delta E}{T}}$ **terhadap Banyak Iterasi yang Telah Dilewati**	
Jumlah Iterasi	795
Frekuensi *Stuck* di *Local Optima*	626
Durasi	0.090063 *seconds*

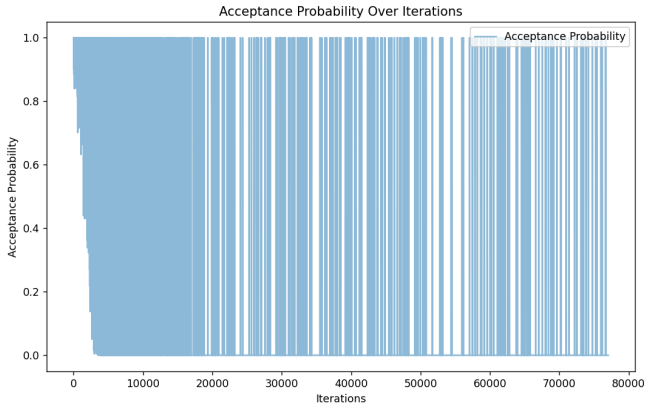
Tabel 4.15 Eksperimen 3 *Simulated Annealing*

Temperatur Awal	3000
Temperatur Akhir	1e-30
Cooling Rate	0.999
Threshold	0.5
***State Cube* Awal**	Layer 1: 52 102 101 98 25

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

***Objective Function* Awal**	6783
***State Cube* Akhir**	Layer 1: 88 78 4 56 89 55 36 124 13 76 20 35 72 113 75 115 108 45 46 6 34 57 68 87 70 ---------- Layer 2: 110 63 17 67 58 37 65 114 81 18 107 10 71 28 102

	23 66 59 50 117 39 111 52 94 19 ---------- Layer 3: 2 98 93 119 3 32 106 1 80 100 122 40 61 44 48 31 47 84 69 85 123 27 77 9 79 ---------- Layer 4: 82 24 83 22 104 99 101 11 90 15 42 120 8 125 16 25 41 116 38 95 73 30 97 29 86 ---------- Layer 5: 33 51 118 49 62 92 7 60 53 105 26 109 103 5 74 121 54 14 112 12 43 91 21 96 64 ----------
***Objective Function* Akhir**	254
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	

Plot $e^{\frac{\Delta E}{T}}$ terhadap Banyak Iterasi yang Telah Dilewati	
Jumlah Iterasi	77046
Frekuensi *Stuck* di *Local Optima*	72808
Durasi	7.184037 *seconds*

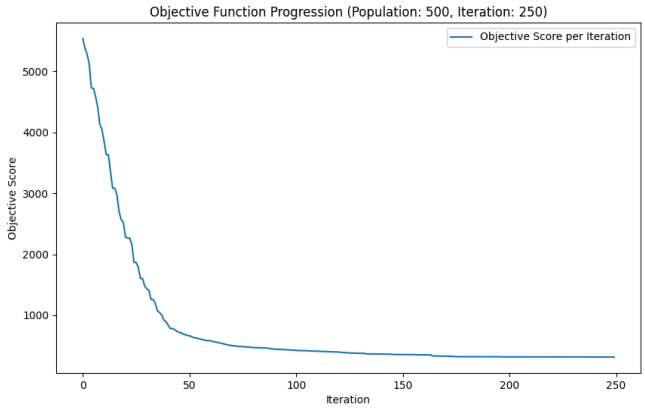
f. *Genetic Algorithm*

Berikut merupakan hasil eksperimen *Genetic Algorithm.*

Tabel 4.16 Eksperimen 1 Variasi Populasi 1 *Genetic Algorithm*

Jumlah Populasi	500
Jumlah Iterasi	250
***State Cube* Awal**	Layer 1: 92 103 116 99 113 93 5 91 102 49 72 37 33 38 56 117 61 13 66 94 88 62 10 59 19 --- Layer 2: 26 30 24 20 104 78 76 69 81 54 8 100 14 118 50 39 108 71 86 16 97 106 58 105 85 --- Layer 3: 9 82 124 114 47 101 119 74 29 28

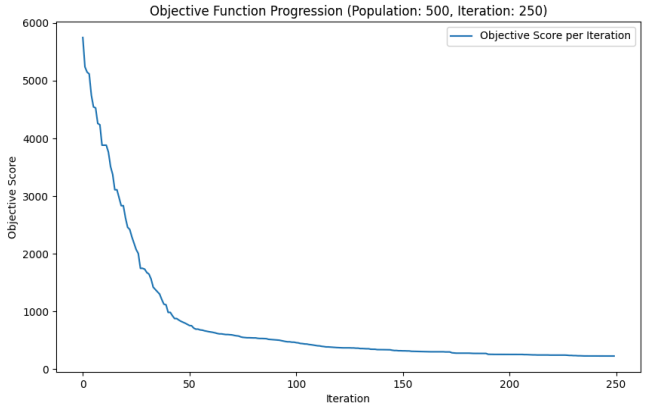
	60 52 1 65 57 98 35 84 67 77 110 42 87 115 73 ---------- Layer 4: 109 21 70 122 36 53 63 121 43 89 96 4 41 40 107 32 64 18 2 75 125 80 22 55 34 ---------- Layer 5: 31 79 44 90 17 51 11 83 123 111 12 46 27 25 15 3 23 48 45 112 6 68 95 120 7 ----------
***Objective Function* Awal**	7013
***State Cube* Akhir**	Layer 1: 92 103 116 99 113 93 5 91 102 49 72 37 33 38 56 117 61 13 66 94 88 62 10 59 19 ---------- Layer 2: 26 30 24 20 104 78 76 69 81 54 8 100 14 118 50 39 108 71 86 16 97 106 58 105 85 ---------- Layer 3: 9 82 124 114 47 101 119 74 29 28 60 52 1 65 57 98 35 84 67 77 110 42 87 115 73 ---------- Layer 4: 109 21 70 122 36 53 63 121 43 89 96 4 41 40 107 32 64 18 2 75

	125 80 22 55 34 -------------------- Layer 5: 31 79 44 90 17 51 11 83 123 111 12 46 27 25 15 3 23 48 45 112 6 68 95 120 7 --------------------
***Objective Function* Akhir**	316
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	143.9388 *seconds*

Tabel 4.17 Eksperimen 2 Variasi Populasi 1 *Genetic Algorithm*

Jumlah Populasi	500
Jumlah Iterasi	250
***State Cube* Awal**	Layer 1: 62 75 43 72 79 18 89 69 120 102 121 113 8 111 31 49 106 4 108 66 99 101 1 5 103 -------------------- Layer 2: 88 25 27 44 123 87 67 3 37 86 39 59 60 48 45 40 93 19 63 50

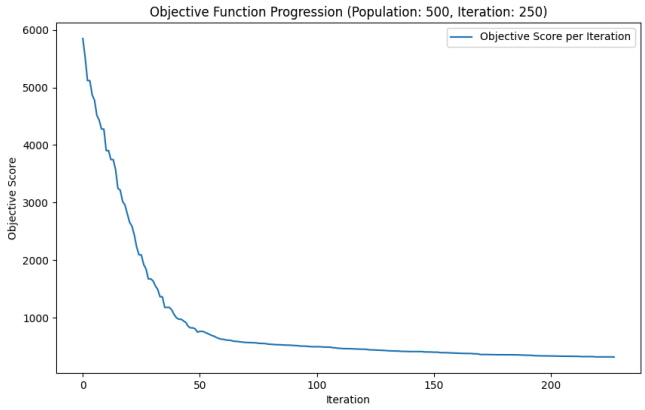
	81 33 46 57 58 --- Layer 3: 10 117 112 29 24 16 68 114 100 109 51 77 125 116 98 90 70 115 83 52 6 105 95 22 17 --- Layer 4: 36 97 78 26 41 74 119 21 53 107 64 11 91 23 20 92 12 84 13 61 104 32 96 9 42 --- Layer 5: 34 47 7 85 124 82 76 80 15 14 28 73 55 118 94 30 54 122 56 110 35 65 38 2 71 ---
***Objective Function* Awal**	7582
***State Cube* Akhir**	Layer 1: 62 75 43 72 79 18 89 69 120 102 121 113 8 111 31 49 106 4 108 66 99 101 1 5 103 --- Layer 2: 88 25 27 44 123 87 67 3 37 86 39 59 60 48 45 40 93 19 63 50 81 33 46 57 58 --- Layer 3: 10 117 112 29 24 16 68 114 100 109 51 77 125 116 98 90 70 115 83 52 6 105 95 22 17 ---

	Layer 4: 36 97 78 26 41 74 119 21 53 107 64 11 91 23 20 92 12 84 13 61 104 32 96 9 42 ---------- Layer 5: 34 47 7 85 124 82 76 80 15 14 28 73 55 118 94 30 54 122 56 110 35 65 38 2 71 ----------
***Objective Function* Akhir**	226
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	146.0335 *seconds*

Tabel 4.18 Eksperimen 3 Variasi Populasi 1 *Genetic Algorithm*

Jumlah Populasi	500
Jumlah Iterasi	250
***State Cube* Awal**	Layer 1: 104 47 22 84 86 63 103 39 82 70 72 93 31 118 119 61 96 85 25 87 106 78 80 51 62 ----------

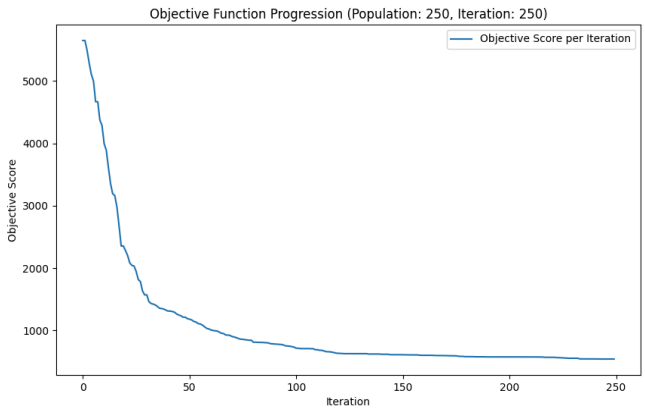
	Layer 2: 92 65 90 116 71 111 124 27 69 55 44 58 109 11 74 81 16 4 38 45 24 75 34 18 67 ---------- Layer 3: 88 73 53 46 41 76 113 7 48 114 68 57 91 9 110 60 98 59 125 94 99 105 107 5 14 ---------- Layer 4: 26 32 77 97 100 37 83 50 42 102 89 10 101 49 66 52 108 36 64 19 115 56 117 79 8 ---------- Layer 5: 12 2 29 6 35 3 20 112 28 13 120 17 40 30 23 54 33 95 121 21 15 1 122 123 43 ----------
***Objective Function* Awal**	6899
***State Cube* Akhir**	Layer 1: 104 47 22 84 86 63 103 39 82 70 72 93 31 118 119 61 96 85 25 87 106 78 80 51 62 ---------- Layer 2: 92 65 90 116 71 111 124 27 69 55 44 58 109 11 74 81 16 4 38 45 24 75 34 18 67 ---------- Layer 3: 88 73 53 46 41

	76 113 7 48 114 68 57 91 9 110 60 98 59 125 94 99 105 107 5 14 -------------------- Layer 4: 26 32 77 97 100 37 83 50 42 102 89 10 101 49 66 52 108 36 64 19 115 56 117 79 8 -------------------- Layer 5: 12 2 29 6 35 3 20 112 28 13 120 17 40 30 23 54 33 95 121 21 15 1 122 123 43 --------------------
***Objective Function* Akhir**	315
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	134.1040 *seconds*

Tabel 4.19 Eksperimen 1 Variasi Populasi 2 *Genetic Algorithm*

Jumlah Populasi	250
Jumlah Iterasi	250
***State Cube* Awal**	Layer 1: 53 21 3 34 87

	106 5 62 15 120 29 38 100 99 111 96 12 102 98 113 101 52 61 122 97 --- Layer 2: 83 46 58 84 1 63 32 85 20 108 36 55 24 16 51 7 67 9 59 88 41 107 6 72 90 --- Layer 3: 23 54 110 11 109 28 95 119 103 42 81 44 114 37 77 10 40 49 65 104 76 4 19 74 45 --- Layer 4: 56 80 91 123 22 112 70 73 60 121 8 57 43 93 17 18 89 118 69 105 125 35 31 66 50 --- Layer 5: 64 30 27 2 94 48 47 117 124 14 79 78 82 33 75 115 68 71 39 92 13 86 25 116 26 ---
***Objective Function* Awal**	6969
***State Cube* Akhir**	Layer 1: 53 21 3 34 87 106 5 62 15 120 29 38 100 99 111 96 12 102 98 113 101 52 61 122 97 --- Layer 2: 83 46 58 84 1 63 32 85 20 108 36 55 24 16 51

	7 67 9 59 88 41 107 6 72 90 ---------- Layer 3: 23 54 110 11 109 28 95 119 103 42 81 44 114 37 77 10 40 49 65 104 76 4 19 74 45 ---------- Layer 4: 56 80 91 123 22 112 70 73 60 121 8 57 43 93 17 18 89 118 69 105 125 35 31 66 50 ---------- Layer 5: 64 30 27 2 94 48 47 117 124 14 79 78 82 33 75 115 68 71 39 92 13 86 25 116 26 ----------
***Objective Function* Akhir**	544
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	72.6665 *seconds*

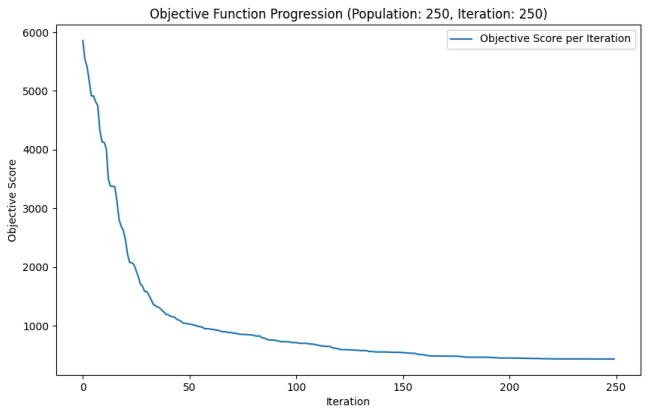
Tabel 4.20 Eksperimen 2 Variasi Populasi 2 *Genetic Algorithm*

Jumlah Populasi	250

Jumlah Iterasi	250

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

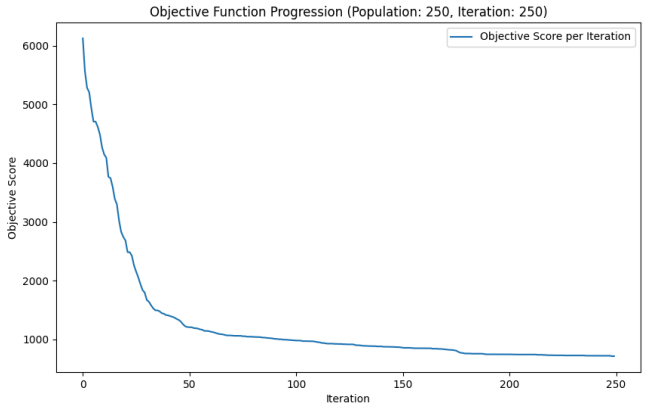
***Objective Function* Awal**	6943
***State Cube* Akhir**	Layer 1: 5 101 84 77 111 25 106 57 27 10 33 12 35 82 90 112 109 117 45 72 79 24 1 110 43 --------------------

	Layer 2: 55 4 49 125 28 59 38 114 62 81 37 58 29 74 51 30 54 89 70 41 122 103 16 60 19 --- Layer 3: 94 20 17 52 42 83 22 11 14 124 32 78 87 48 113 92 99 76 119 71 91 34 31 39 7 --- Layer 4: 97 36 50 67 2 8 21 69 46 120 118 15 23 9 40 123 86 121 64 75 13 108 73 3 26 --- Layer 5: 88 6 107 18 68 100 105 63 116 85 104 56 47 44 102 96 80 95 93 61 53 115 98 65 66 ---
***Objective Function* Akhir**	433
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	72.8765 *seconds*

Tabel 4.21 Eksperimen 3 Variasi Populasi 2 *Genetic Algorithm*

Jumlah Populasi	250
Jumlah Iterasi	250
***State Cube* Awal**	Layer 1: 27 69 67 115 59 110 83 30 62 55 53 14 73 39 46 89 5 121 35 104 38 71 45 36 96 ---------- Layer 2: 70 17 56 102 113 119 32 124 37 80 49 120 114 29 58 3 50 9 2 100 51 87 34 72 111 ---------- Layer 3: 16 66 105 11 40 85 97 88 33 75 47 65 31 108 94 44 92 60 76 6 42 64 93 82 116 ---------- Layer 4: 91 25 77 123 23 63 109 7 4 117 8 95 18 118 13 52 125 41 103 26 99 106 78 79 19 ---------- Layer 5: 112 1 61 98 68 15 28 86 22 20 90 81 84 43 54 24 74 57 21 48 12 122 10 107 101 ----------
***Objective Function* Awal**	6552
***State Cube* Akhir**	Layer 1: 27 69 67 115 59 110 83 30 62 55

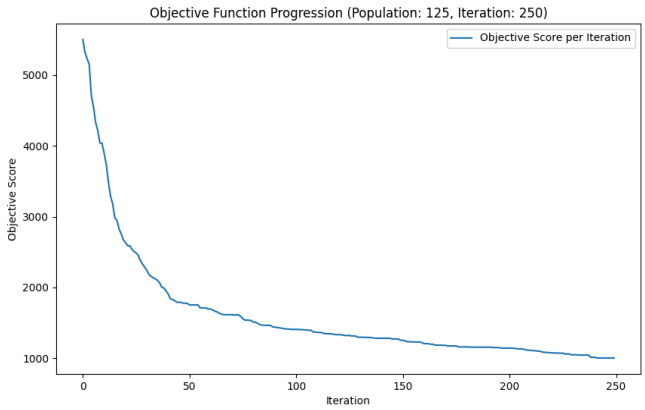
	53 14 73 39 46 89 5 121 35 104 38 71 45 36 96 ---------- Layer 2: 70 17 56 102 113 119 32 124 37 80 49 120 114 29 58 3 50 9 2 100 51 87 34 72 111 ---------- Layer 3: 16 66 105 11 40 85 97 88 33 75 47 65 31 108 94 44 92 60 76 6 42 64 93 82 116 ---------- Layer 4: 91 25 77 123 23 63 109 7 4 117 8 95 18 118 13 52 125 41 103 26 99 106 78 79 19 ---------- Layer 5: 112 1 61 98 68 15 28 86 22 20 90 81 84 43 54 24 74 57 21 48 12 122 10 107 101 ----------
***Objective Function* Akhir**	716

***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	75.0159 *seconds*

Tabel 4.22 Eksperimen 1 Variasi Populasi 3 *Genetic Algorithm*

Jumlah Populasi	125
Jumlah Iterasi	250
***State Cube* Awal**	Layer 1: 54 78 18 9 71 82 101 21 73 38 53 13 25 76 55 2 121 90 30 44 81 59 19 4 118 ---------- Layer 2: 98 65 108 35 125 17 84 113 120 114 7 51 109 29 34 87 41 106 77 1 6 28 67 97 85 ---------- Layer 3: 61 42 37 79 91 23 26 31 52 70 15 36 122 107 62 110 112 5 56 24 96 39 60 88 32 ---------- Layer 4: 103 94 16 117 75 119 83 50 14 93 102 63 99 72 105

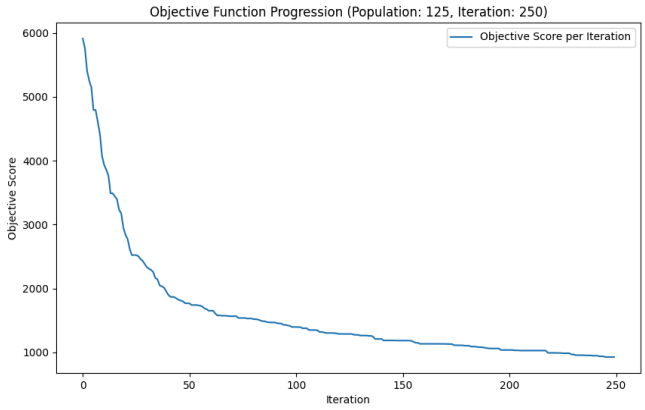
	47 115 12 92 100 74 3 48 43 64 --- Layer 5: 80 116 57 89 86 58 95 8 124 20 22 27 40 123 10 33 45 111 104 49 11 46 66 68 69 ---
***Objective Function* Awal**	6669
***State Cube* Akhir**	Layer 1: 54 78 18 9 71 82 101 21 73 38 53 13 25 76 55 2 121 90 30 44 81 59 19 4 118 --- Layer 2: 98 65 108 35 125 17 84 113 120 114 7 51 109 29 34 87 41 106 77 1 6 28 67 97 85 --- Layer 3: 61 42 37 79 91 23 26 31 52 70 15 36 122 107 62 110 112 5 56 24 96 39 60 88 32 --- Layer 4: 103 94 16 117 75 119 83 50 14 93 102 63 99 72 105 47 115 12 92 100 74 3 48 43 64 --- Layer 5: 80 116 57 89 86 58 95 8 124 20 22 27 40 123 10 33 45 111 104 49 11 46 66 68 69

***Objective Function* Akhir**	1003
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	36.9171 *seconds*

Tabel 4.23 Eksperimen 2 Variasi Populasi 3 *Genetic Algorithm*

Jumlah Populasi	125
Jumlah Iterasi	250
***State Cube* Awal**	Layer 1: 121 69 32 21 118 70 65 96 15 10 112 82 49 80 85 120 24 68 52 37 97 113 58 57 42 ---------- Layer 2: 72 122 59 51 123 19 93 119 67 18 31 125 102 108 23 41 55 11 56 116 30 1 117 84 44 ---------- Layer 3: 62 107 94 92 2 20 105 79 35 46 7 22 103 77 64 98 75 110 45 36 16 14 29 114 4

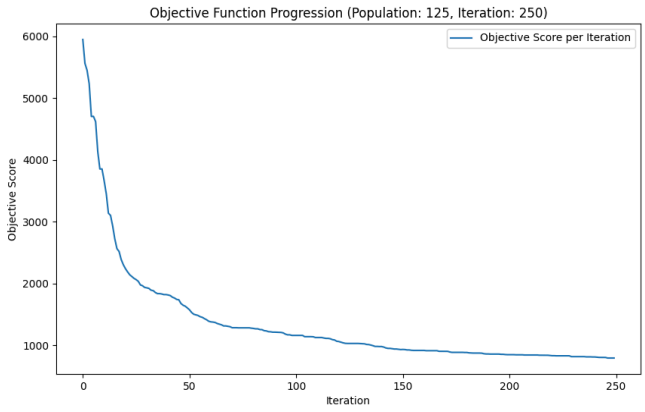
	--- Layer 4: 38 5 91 61 99 88 86 28 54 95 100 87 66 43 63 26 9 73 124 12 81 104 60 106 74 --- Layer 5: 101 6 90 33 71 17 109 39 111 47 53 27 48 13 3 50 83 76 115 8 40 34 25 89 78 ---
***Objective Function* Awal**	6620
***State Cube* Akhir**	Layer 1: 121 69 32 21 118 70 65 96 15 10 112 82 49 80 85 120 24 68 52 37 97 113 58 57 42 --- Layer 2: 72 122 59 51 123 19 93 119 67 18 31 125 102 108 23 41 55 11 56 116 30 1 117 84 44 --- Layer 3: 62 107 94 92 2 20 105 79 35 46 7 22 103 77 64 98 75 110 45 36 16 14 29 114 4 --- Layer 4: 38 5 91 61 99 88 86 28 54 95 100 87 66 43 63 26 9 73 124 12 81 104 60 106 74 --- Layer 5:

	101 6 90 33 71 17 109 39 111 47 53 27 48 13 3 50 83 76 115 8 40 34 25 89 78 ----------
***Objective Function* Akhir**	927
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	36.2130 *seconds*

Tabel 4.24 Eksperimen 3 Variasi Populasi 3 *Genetic Algorithm*

Jumlah Populasi	125
Jumlah Iterasi	250
***State Cube* Awal**	Layer 1: 48 57 27 6 47 68 83 9 41 98 93 108 118 111 123 12 59 110 115 61 56 34 22 95 28 ---------- Layer 2: 84 72 71 77 21 75 31 8 2 64 91 25 37 7 10 89 1 17 66 19 86 33 112 4 26 ---------- Layer 3:

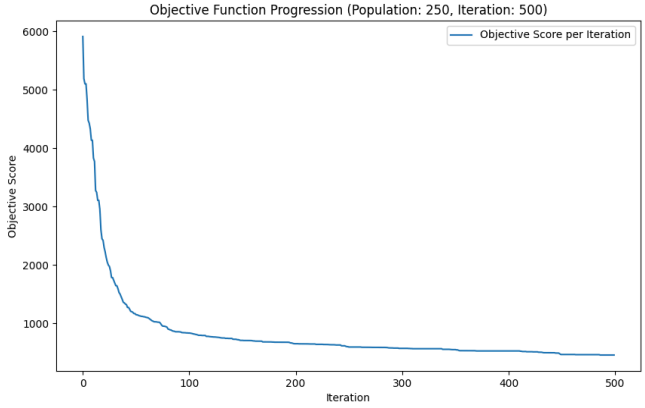
	119 40 100 99 94 30 96 69 65 11 60 58 5 53 13 18 92 24 15 39 101 116 14 88 90 ---------- Layer 4: 125 36 35 55 80 76 29 49 43 87 82 113 104 54 42 78 23 20 45 105 79 121 74 114 120 ---------- Layer 5: 81 97 106 63 70 103 62 122 85 67 124 16 46 50 3 51 38 52 32 109 102 117 107 73 44 ----------
***Objective Function* Awal**	7098
***State Cube* Akhir**	Layer 1: 48 57 27 6 47 68 83 9 41 98 93 108 118 111 123 12 59 110 115 61 56 34 22 95 28 ---------- Layer 2: 84 72 71 77 21 75 31 8 2 64 91 25 37 7 10 89 1 17 66 19 86 33 112 4 26 ---------- Layer 3: 119 40 100 99 94 30 96 69 65 11 60 58 5 53 13 18 92 24 15 39 101 116 14 88 90 ---------- Layer 4: 125 36 35 55 80 76 29 49 43 87

	82 113 104 54 42 78 23 20 45 105 79 121 74 114 120 --- Layer 5: 81 97 106 63 70 103 62 122 85 67 124 16 46 50 3 51 38 52 32 109 102 117 107 73 44 ---
***Objective Function* Akhir**	793
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	36.1924 *seconds*

Tabel 4.25 Eksperimen 1 Variasi Iterasi 1 *Genetic Algorithm*

Jumlah Populasi	250
Jumlah Iterasi	500
***State Cube* Awal**	Layer 1: 103 25 19 21 91 83 110 68 119 113 76 42 114 98 88 28 13 122 10 60 99 55 7 97 31 --- Layer 2: 123 6 54 59 41 112 107 3 20 9

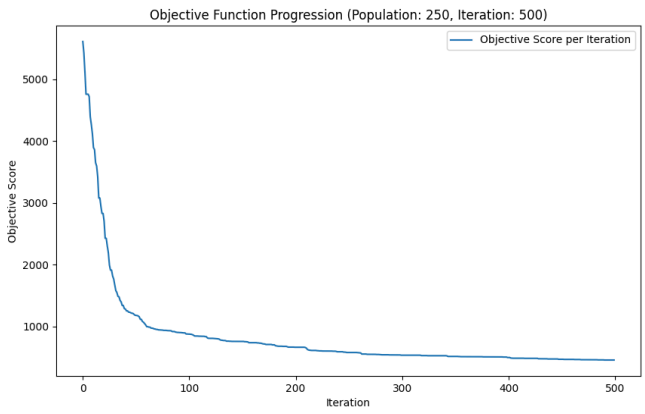
	111 70 53 36 51 17 63 64 46 92 115 44 120 102 109 --- Layer 3: 62 74 72 5 32 29 82 52 38 30 58 116 23 95 49 8 100 117 34 65 61 105 47 96 90 --- Layer 4: 77 75 84 69 4 86 87 35 18 85 33 40 11 78 1 15 27 125 50 45 26 118 79 39 93 --- Layer 5: 12 67 121 16 94 43 48 101 66 56 73 89 80 104 2 124 106 14 24 108 57 22 81 71 37 ---
***Objective Function* Awal**	7057
***State Cube* Akhir**	Layer 1: 103 25 19 21 91 83 110 68 119 113 76 42 114 98 88 28 13 122 10 60 99 55 7 97 31 --- Layer 2: 123 6 54 59 41 112 107 3 20 9 111 70 53 36 51 17 63 64 46 92 115 44 120 102 109 --- Layer 3: 62 74 72 5 32 29 82 52 38 30 58 116 23 95 49 8 100 117 34 65

	61 105 47 96 90 -------------------- Layer 4: 77 75 84 69 4 86 87 35 18 85 33 40 11 78 1 15 27 125 50 45 26 118 79 39 93 -------------------- Layer 5: 12 67 121 16 94 43 48 101 66 56 73 89 80 104 2 124 106 14 24 108 57 22 81 71 37 --------------------
***Objective Function* Akhir**	457
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	141.4125 *seconds*

Tabel 4.26 Eksperimen 2 Variasi Iterasi 1 *Genetic Algorithm*

Jumlah Populasi	250
Jumlah Iterasi	500
***State Cube* Awal**	Layer 1: 123 65 74 78 16 105 3 76 6 116 100 84 7 33 46 34 15 86 114 5

	12 94 111 112 120 -------------------- Layer 2: 72 91 77 10 2 89 71 62 67 11 101 8 43 37 32 24 64 9 54 56 110 60 66 90 118 -------------------- Layer 3: 97 68 96 83 20 39 17 117 47 108 119 14 87 55 57 58 1 49 53 59 40 19 50 45 36 -------------------- Layer 4: 42 4 35 27 52 124 30 85 79 51 38 63 69 104 92 18 81 115 75 61 93 70 121 109 95 -------------------- Layer 5: 29 107 113 26 25 73 102 125 48 13 122 23 80 41 44 103 106 99 31 22 28 88 98 21 82 --------------------
***Objective Function* Awal**	7045
***State Cube* Akhir**	Layer 1: 123 65 74 78 16 105 3 76 6 116 100 84 7 33 46 34 15 86 114 5 12 94 111 112 120 -------------------- Layer 2: 72 91 77 10 2 89 71 62 67 11 101 8 43 37 32 24 64 9 54 56 110 60 66 90 118 --------------------

	Layer 3: 97 68 96 83 20 39 17 117 47 108 119 14 87 55 57 58 1 49 53 59 40 19 50 45 36 ---------- Layer 4: 42 4 35 27 52 124 30 85 79 51 38 63 69 104 92 18 81 115 75 61 93 70 121 109 95 ---------- Layer 5: 29 107 113 26 25 73 102 125 48 13 122 23 80 41 44 103 106 99 31 22 28 88 98 21 82 ----------
***Objective Function* Akhir**	455
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	140.3368 *seconds*

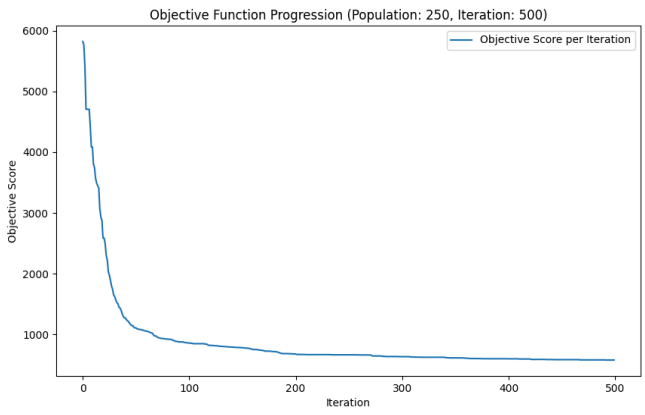
Tabel 4.27 Eksperimen 3 Variasi Iterasi 1 *Genetic Algorithm*

Jumlah Populasi	250
Jumlah Iterasi	500

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

***Objective Function* Awal**	7167

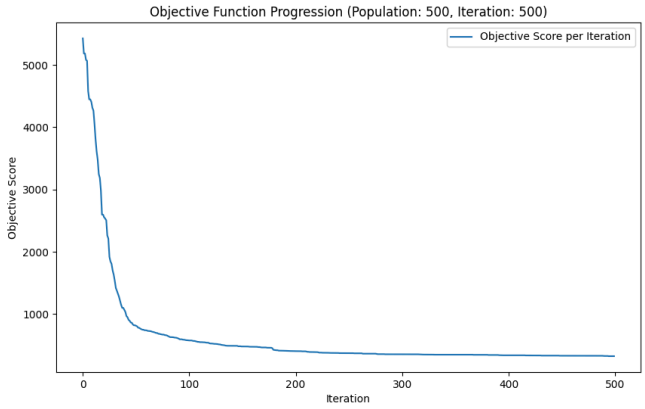
***State Cube* Akhir**	Layer 1:
	  7 29 9 34 99
	  3 96 58 26 70
	117 43 91 57 114
	 18 72 35 87 62
	 24 42 71 123 40
	-------------------- 	
	Layer 2:
	102 74 44 60 101

	36 79 50 32 122 88 13 16 11 55 46 107 84 12 39 4 63 1 82 6 --- Layer 3: 83 120 89 105 38 41 100 48 49 54 65 22 68 108 30 59 116 90 104 10 66 23 125 93 15 --- Layer 4: 118 111 110 61 75 33 73 98 80 27 86 21 109 51 17 20 76 92 53 8 19 103 14 81 52 --- Layer 5: 112 124 121 56 85 69 45 97 37 67 78 106 31 47 28 5 115 94 113 77 64 119 95 2 25 ---
***Objective Function* Akhir**	580
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	140.6737 *seconds*

Tabel 4.28 Eksperimen 1 Variasi Iterasi 2 *Genetic Algorithm*

Jumlah Populasi	500
Jumlah Iterasi	500
***State Cube* Awal**	Layer 1: 105 84 14 116 38 68 47 35 45 51 19 65 76 9 48 13 87 101 89 118 49 50 40 110 63 ---------- Layer 2: 74 61 17 52 7 44 104 72 75 57 67 83 10 60 29 37 108 58 25 85 59 73 125 20 30 ---------- Layer 3: 123 4 33 96 27 106 70 117 42 71 95 23 18 26 28 11 32 100 113 66 31 81 3 1 103 ---------- Layer 4: 109 122 2 80 99 15 124 6 39 12 69 82 36 115 8 77 24 86 16 64 56 112 78 79 92 ---------- Layer 5: 55 91 97 93 22 21 62 119 5 114 121 46 102 90 107 120 43 54 34 53 41 98 88 94 111 ----------
***Objective Function* Awal**	6423
***State Cube* Akhir**	Layer 1: 105 84 14 116 38 68 47 35 45 51

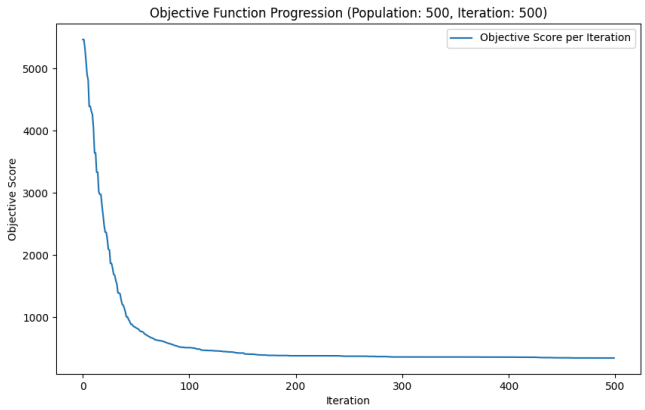
	19 65 76 9 48 13 87 101 89 118 49 50 40 110 63 ---------- Layer 2: 74 61 17 52 7 44 104 72 75 57 67 83 10 60 29 37 108 58 25 85 59 73 125 20 30 ---------- Layer 3: 123 4 33 96 27 106 70 117 42 71 95 23 18 26 28 11 32 100 113 66 31 81 3 1 103 ---------- Layer 4: 109 122 2 80 99 15 124 6 39 12 69 82 36 115 8 77 24 86 16 64 56 112 78 79 92 ---------- Layer 5: 55 91 97 93 22 21 62 119 5 114 121 46 102 90 107 120 43 54 34 53 41 98 88 94 111 ----------
***Objective Function* Akhir**	323

***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	276.7321 *seconds*

Tabel 4.29 Eksperimen 2 Variasi Iterasi 2 *Genetic Algorithm*

Jumlah Populasi	500
Jumlah Iterasi	500
***State Cube* Awal**	Layer 1: 103 58 97 69 82 26 49 44 22 18 7 5 94 72 29 90 119 36 55 1 62 43 20 110 34 ---------- Layer 2: 50 85 3 95 12 100 93 23 25 75 61 109 106 10 45 118 120 112 86 31 21 63 38 15 78 ---------- Layer 3: 48 13 102 124 4 70 79 83 6 71 123 87 2 117 40 65 67 74 96 108 105 33 114 88 80 ---------- Layer 4: 46 116 99 59 51 41 111 92 122 8 39 125 73 101 30

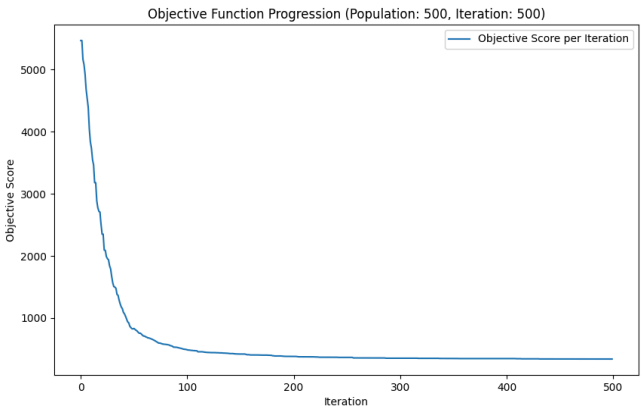
	24 42 9 104 60 53 91 115 11 14 --- Layer 5: 54 47 16 84 66 98 32 57 19 56 113 107 37 76 27 52 17 35 89 64 28 121 81 77 68 ---
***Objective Function* Awal**	7173
***State Cube* Akhir**	Layer 1: 103 58 97 69 82 26 49 44 22 18 7 5 94 72 29 90 119 36 55 1 62 43 20 110 34 --- Layer 2: 50 85 3 95 12 100 93 23 25 75 61 109 106 10 45 118 120 112 86 31 21 63 38 15 78 --- Layer 3: 48 13 102 124 4 70 79 83 6 71 123 87 2 117 40 65 67 74 96 108 105 33 114 88 80 --- Layer 4: 46 116 99 59 51 41 111 92 122 8 39 125 73 101 30 24 42 9 104 60 53 91 115 11 14 --- Layer 5: 54 47 16 84 66 98 32 57 19 56 113 107 37 76 27 52 17 35 89 64 28 121 81 77 68

***Objective Function* Akhir**	343
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	276.5618 *seconds*

Tabel 4.30 Eksperimen 3 Variasi Iterasi 2 *Genetic Algorithm*

Jumlah Populasi	500
Jumlah Iterasi	500
***State Cube* Awal**	Layer 1: 100 112 58 94 67 42 60 44 102 53 110 118 124 35 111 47 120 116 1 41 88 31 92 99 51 ---------- Layer 2: 56 48 77 38 70 107 71 13 65 87 21 55 40 10 76 90 66 113 122 17 45 19 18 50 32 ---------- Layer 3: 105 64 4 106 12 121 108 36 86 49 97 72 14 57 96 75 52 89 98 119 104 39 81 85 80

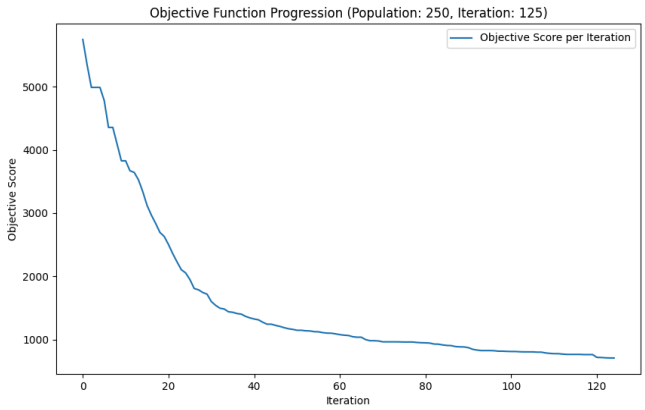
	--- Layer 4: 54 11 83 25 8 27 114 123 61 37 29 78 115 23 84 109 28 9 33 93 95 79 26 117 2 --- Layer 5: 43 6 5 46 103 74 91 24 7 30 62 22 125 16 82 68 69 73 34 20 15 59 3 101 63 ---
***Objective Function* Awal**	6723
***State Cube* Akhir**	Layer 1: 100 112 58 94 67 42 60 44 102 53 110 118 124 35 111 47 120 116 1 41 88 31 92 99 51 --- Layer 2: 56 48 77 38 70 107 71 13 65 87 21 55 40 10 76 90 66 113 122 17 45 19 18 50 32 --- Layer 3: 105 64 4 106 12 121 108 36 86 49 97 72 14 57 96 75 52 89 98 119 104 39 81 85 80 --- Layer 4: 54 11 83 25 8 27 114 123 61 37 29 78 115 23 84 109 28 9 33 93 95 79 26 117 2 --- Layer 5:

	43 6 5 46 103 74 91 24 7 30 62 22 125 16 82 68 69 73 34 20 15 59 3 101 63 ----------
***Objective Function* Akhir**	340
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	278.0833 *seconds*

Tabel 4.31 Eksperimen 1 Variasi Iterasi 3 *Genetic Algorithm*

Jumlah Populasi	250
Jumlah Iterasi	125
***State Cube* Awal**	Layer 1: 38 9 31 15 22 43 122 46 104 70 19 1 116 114 92 33 96 93 123 85 41 64 45 86 52 ---------- Layer 2: 53 54 30 87 73 23 71 10 106 80 42 65 57 5 51 98 13 27 68 125 97 89 76 107 81 ---------- Layer 3:

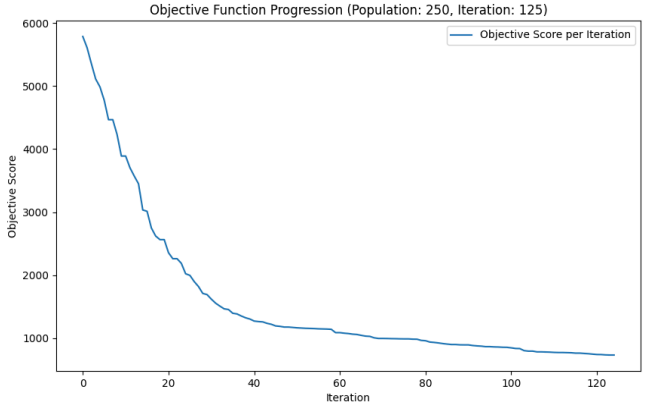
	69 3 26 74 39 6 47 61 56 77 40 44 78 59 99 110 121 113 94 67 60 108 29 34 21 ---------- Layer 4: 72 120 111 75 91 117 105 112 118 88 119 90 100 17 102 50 83 7 55 4 95 32 103 66 49 ---------- Layer 5: 25 58 37 63 62 2 14 24 35 11 101 48 84 82 124 16 36 12 109 18 115 8 20 28 79 ----------
***Objective Function* Awal**	7110
***State Cube* Akhir**	Layer 1: 38 9 31 15 22 43 122 46 104 70 19 1 116 114 92 33 96 93 123 85 41 64 45 86 52 ---------- Layer 2: 53 54 30 87 73 23 71 10 106 80 42 65 57 5 51 98 13 27 68 125 97 89 76 107 81 ---------- Layer 3: 69 3 26 74 39 6 47 61 56 77 40 44 78 59 99 110 121 113 94 67 60 108 29 34 21 ---------- Layer 4: 72 120 111 75 91 117 105 112 118 88

	119 90 100 17 102 50 83 7 55 4 95 32 103 66 49 --- Layer 5: 25 58 37 63 62 2 14 24 35 11 101 48 84 82 124 16 36 12 109 18 115 8 20 28 79 ---
***Objective Function* Akhir**	709
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	37.3456 *seconds*

Tabel 4.32 Eksperimen 2 Variasi Iterasi 3 *Genetic Algorithm*

Jumlah Populasi	250
Jumlah Iterasi	125
***State Cube* Awal**	Layer 1: 123 88 8 57 12 86 5 116 118 31 22 107 16 97 6 102 95 89 101 37 43 4 17 2 30 --- Layer 2: 53 104 11 39 3 63 80 29 50 41

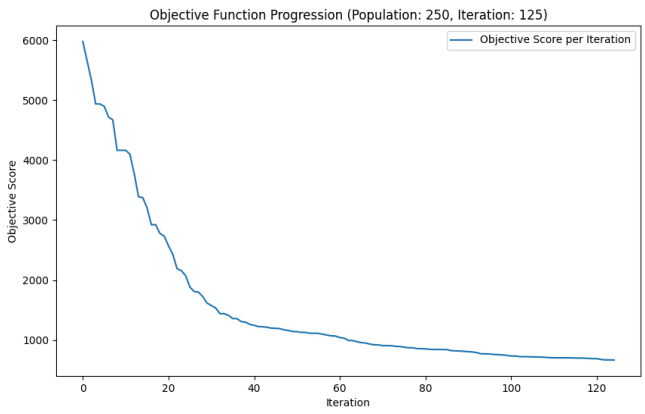
	49 35 38 75 10 78 7 76 79 60 45 103 55 119 73 --- Layer 3: 56 47 36 51 93 42 32 105 21 81 110 72 112 96 120 61 109 27 18 20 48 85 65 122 70 --- Layer 4: 87 124 117 64 99 1 34 100 106 108 52 84 44 77 90 46 58 115 74 62 23 83 91 26 13 --- Layer 5: 69 40 15 24 82 59 9 19 68 111 67 94 92 98 121 54 71 28 125 25 113 66 33 14 114 ---
***Objective Function* Awal**	7742
***State Cube* Akhir**	Layer 1: 123 88 8 57 12 86 5 116 118 31 22 107 16 97 6 102 95 89 101 37 43 4 17 2 30 --- Layer 2: 53 104 11 39 3 63 80 29 50 41 49 35 38 75 10 78 7 76 79 60 45 103 55 119 73 --- Layer 3: 56 47 36 51 93 42 32 105 21 81 110 72 112 96 120 61 109 27 18 20

	48 85 65 122 70 -------------------- Layer 4: 87 124 117 64 99 1 34 100 106 108 52 84 44 77 90 46 58 115 74 62 23 83 91 26 13 -------------------- Layer 5: 69 40 15 24 82 59 9 19 68 111 67 94 92 98 121 54 71 28 125 25 113 66 33 14 114 --------------------
***Objective Function* Akhir**	732
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	36.3131 *seconds*

Tabel 4.33 Eksperimen 3 Variasi Iterasi 3 *Genetic Algorithm*

Jumlah Populasi	250
Jumlah Iterasi	125
***State Cube* Awal**	Layer 1: 7 103 3 4 49 91 44 72 16 28 33 14 123 20 41 55 47 106 80 115

	23 89 50 15 97 --- Layer 2: 61 36 8 86 62 117 54 42 67 102 46 32 105 10 94 57 11 35 92 121 66 29 95 51 75 --- Layer 3: 73 77 96 22 114 122 71 90 87 69 68 88 38 34 109 26 37 5 17 39 83 6 108 101 120 --- Layer 4: 19 85 65 98 118 60 70 99 81 104 27 76 79 63 48 93 84 107 58 116 43 82 113 18 111 --- Layer 5: 59 110 119 112 25 53 12 100 31 13 30 9 74 56 125 1 24 2 64 124 21 45 40 78 52 ---
***Objective Function* Awal**	6436
***State Cube* Akhir**	Layer 1: 7 103 3 4 49 91 44 72 16 28 33 14 123 20 41 55 47 106 80 115 23 89 50 15 97 --- Layer 2: 61 36 8 86 62 117 54 42 67 102 46 32 105 10 94 57 11 35 92 121 66 29 95 51 75 ---

	Layer 3: 73 77 96 22 114 122 71 90 87 69 68 88 38 34 109 26 37 5 17 39 83 6 108 101 120 --- Layer 4: 19 85 65 98 118 60 70 99 81 104 27 76 79 63 48 93 84 107 58 116 43 82 113 18 111 --- Layer 5: 59 110 119 112 25 53 12 100 31 13 30 9 74 56 125 1 24 2 64 124 21 45 40 78 52 ---
***Objective Function* Akhir**	669
***Plot* Nilai *Objective Function* terhadap Banyak Iterasi yang Telah Dilewati**	
Durasi	37.0023 *seconds*

B. Analisis

Berdasarkan hasil eksperimen berbagai algoritma *Local Search* untuk menyelesaikan *Diagonal Magic Cube*, didapatkan beberapa kesimpulan tentang efektivitas dan kinerja masing-masing algoritma. Dalam hal kedekatan dengan solusi optimal (*global optimum*), *Simulated Annealing* menunjukkan hasil terbaik karena memiliki mekanisme probabilitas yang memungkinkan algoritma untuk memilih solusi yang kurang optimal pada tahap awal, sehingga lebih fleksibel dalam mencari solusi optimal secara keseluruhan dan tidak mudah terjebak pada solusi lokal terbaik (*local optimum*). *Genetic Algorithm* juga menunjukkan potensi mendekati solusi optimal dengan proses seleksi dan kombinasi individu dalam populasi, tetapi kinerjanya sangat dipengaruhi oleh ukuran populasi dan jumlah iterasi. Di sisi lain, algoritma *Hill Climbing* (*Steepest Ascent, Stochastic*, dan *Sideways Move*) cenderung mencapai solusi lokal terbaik lebih cepat, tetapi terbatas dalam kemampuannya untuk mendekati solusi global karena hanya mencari solusi terbaik di sekitar solusi saat ini.

Stochastic Hill Climbing menghasilkan hasil yang lebih bervariasi karena pemilihan langkah dilakukan secara acak, meskipun hasil akhirnya tetap terbatas pada solusi lokal. *Sideways Move Hill Climbing* memiliki fleksibilitas lebih tinggi pada kondisi datar (plateau), sehingga memungkinkan algoritma untuk tetap melangsungkan pencarian solusi bahkan ketika terjebak pada nilai yang sama, namun algoritma ini membutuhkan waktu pencarian lebih lama. Berikutnya, untuk *Random Restart Hill Climbing* dapat memberikan hasil yang lebih baik dengan mencoba ulang dari posisi awal yang berbeda, tetapi tetap bergantung pada kondisi awal yang dipilih secara acak.

Dilihat dari waktu pencarian, *Simulated Annealing* paling efisien dan mendekati hasil optimal, karena menggabungkan strategi pencarian lokal dengan probabilitas perpindahan solusi, sehingga membuat algoritma ini mampu mengeksplorasi ruang solusi secara efektif, tanpa menghabiskan terlalu banyak waktu dalam prosesnya. Algoritma *Hill Climbing Stochastic* memiliki durasi pencarian tercepat karena metode ini tidak memerlukan pertimbangan terhadap semua *state* yang dapat terjadi, namun hanya melangsungkan percobaan secara acak dan langsung mengambil hasil yang terbaik, tetapi metode ini masih sering berhenti pada solusi lokal. *Genetic Algorithm* memerlukan waktu lebih lama, terutama pada populasi besar, tetapi mampu mendekati solusi optimal jika parameter yang digunakan tepat.

Konsistensi hasil juga bervariasi. *Steepest Ascent Hill Climbing* menghasilkan hasil yang stabil tetapi terbatas pada solusi lokal. *Simulated Annealing* menunjukkan hasil yang lebih konsisten mendekati optimal dengan pengaturan suhu dan laju pendinginan yang diatur dengan tepat. *Genetic Algorithm* dapat menghasilkan

variasi hasil yang besar tergantung pada pengaturan persilangan (*crossover*) dan mutasi, tetapi menjadi lebih stabil jika populasinya besar dan iterasinya cukup.

Jumlah iterasi dan ukuran populasi sangat mempengaruhi hasil *Genetic Algorithm*. Iterasi yang lebih tinggi memungkinkan eksplorasi lebih luas, tetapi setelah titik tertentu, peningkatan iterasi tidak lagi memberikan peningkatan signifikan. Populasi besar memperkaya variasi solusi dan membantu menghindari solusi lokal terbaik, tetapi memperlambat komputasi. Simulated Annealing juga bergantung kepada parameter suhu awal, suhu akhir, dan *cooling rate*. Ketiga parameter tersebut berdampak besar pada kualitas hasil eksperimen. Oleh karena itu, pengaturan suhu yang seimbang sangat penting untuk mencapai solusi yang optimal. Secara keseluruhan, *Simulated Annealing* dan *Genetic Algorithm* paling mendekati solusi optimal, dengan *Simulated Annealing* unggul dalam waktu pencarian dan *Genetic Algorithm* cocok untuk eksplorasi lebih luas jika waktu komputasi mencukupi.

Kesimpulan dan Saran

Berdasarkan eksperimen yang dilakukan, *Simulated Annealing* terbukti sebagai algoritma yang paling efektif untuk mendekati solusi optimal dalam permasalahan *Diagonal Magic Cube*. Di sisi lain, algoritma *Genetic Algorithm* juga menunjukkan potensi yang kuat untuk mendekati solusi optimal melalui proses seleksi, *crossover*, dan mutasi.

Untuk algoritma *Steepest Ascent Hill-Climbing, Stochastic Hill-Climbing*, dan *Sideways Move*, meskipun stabil dalam pencarian solusi, ketiganya rentan terhadap jebakan local optima dan cenderung berhenti ketika tidak ada perbaikan langsung pada *objective function*. Terlebih, *Random Restart Hill-Climbing* menunjukkan performa yang bervariasi tergantung pada jumlah restart yang dilakukan, namun tetap memiliki keterbatasan dalam mencapai *global optimum* tanpa eksplorasi lebih lanjut.

Secara keseluruhan, hasil eksperimen ini menunjukkan bahwa *Simulated Annealing* paling cocok untuk masalah dengan kompleksitas tinggi seperti *Diagonal Magic Cube*, sedangkan *Genetic Algorithm* dapat menjadi pilihan alternatif yang kuat dengan parameter yang tepat.

Sebagai pengembangan lebih lanjut, disarankan untuk menambahkan visualisasi interaktif yang dapat menunjukkan setiap tahapan pencarian solusi secara berurutan, misalnya dalam bentuk video. Hal ini akan membantu pengguna memahami proses pencarian solusi secara lebih mendalam. Selain itu, penyesuaian parameter seperti jumlah iterasi maksimum, tingkat mutasi pada *Genetic Algorithm*, dan laju penurunan temperatur pada *Simulated Annealing* dapat dioptimalkan untuk mendapatkan hasil yang lebih baik. Menghindari pendekatan *deadliner* juga sangat disarankan, agar eksperimen dapat dilakukan dengan waktu pengujian yang cukup dan terstruktur.

Pembagian Tugas

Berikut merupakan pembagian tugas pada pekerjaan Tugas Besar I Intelegensi Artifisial Kelompok 7.

Tabel 6.1 Pembagian Tugas

NIM	Nama Lengkap	Tugas
18222113	Angelica Aliwinata	Algoritma & Laporan ● Random Restart Hill Climbing
18222116	Jason Jahja	Algoritma & Laporan ● Steepest Ascent Hill Climbing ● Hill Climbing with Sideways Move ● Stochastic Hill Climbing
18222123	Melissa Trenggono	Algoritma & Laporan ● Simulated Annealing
18222128	Anindita Widya Santoso	Algoritma & Laporan ● Genetic Algorithm

Lampiran

Link *Repository* Github: https://github.com/aninditaws/IF3070_TugasBesar1

Referensi

Algorithm Afternoon. (n.d.). *Stochastic Hill Climbing with Random Restarts*. https://algorithmafternoon.com/stochastic/stochastic_hill_climbing_with_random_restarts/

Features of the magic cube. Magisch Vierkant. (n.d.). https://www.magischvierkant.com/three-dimensional-eng/magic-features/

GeeksforGeeks. (2023, March 10). Crossover in Genetic Algorithm. GeeksforGeeks. https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/

Khodra, M. L. (n.d.). *Simulated Annealing*. EDUNEX ITB, KK IF – Teknik Informatika – STEI ITB, Modul 3: Beyond Classical Search.

Khodra, M. L. (n.d.). *Hill-climbing Search*. EDUNEX ITB, KK IF – Teknik Informatika – STEI ITB, Modul 3: Beyond Classical Search.

KK IF – Teknik Informatika – STEI ITB. (n.d.). *Genetic Algorithm*. EDUNEX ITB, Modul 3: Beyond Classical Search.

Scholz, J. (2019). Genetic Algorithms and the Traveling Salesman Problem a Historical Review. https://doi.org/ https://doi.org/10.48550/arXiv.1901.05737

Trump, W. (2003, June 19). The Successful Search for the Smallest Perfect Magic Cube. Perfect Magic Cubes. https://www.trump.de/magic-squares/magic-cubes/cubes-1.html