

Tugas Besar 1 IF3170 Inteligensi Artifisial :
Pencarian Solusi Penjadwalan Kelas Mingguan
dengan Local Search



Disusun oleh Kelompok 46:

Sebastian Enrico Nathanael 13523134

Jonathan Kenan Budianto 13523139

Mahesa Fadhillah Andre 13523140

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

2025

DAFTAR ISI

DAFTAR ISI	2
BAB I	3
DESKRIPSI PERSOALAN	3
BAB II	5
PEMBAHASAN	5
2.1 Pemilihan Objective Function	5
2.1.1 Source Code Objective Function	7
2.1.2 Deskripsi Fungsi Objective Function	7
2.2 Penjelasan Implementasi Algoritma Local Search	8
2.2.1 Steepest Ascent Hill-climbing	9
2.2.1.1 Source Code Steepest Ascent Hill-climbing	11
2.2.1.2 Deskripsi Fungsi Steepest Ascent Hill-climbing	11
2.2.2 Stochastic Hill-climbing	13
2.2.2.1 Source Code Stochastic Hill-climbing	15
2.2.2.2 Deskripsi Fungsi Stochastic Hill-climbing	15
2.2.3 Hill-climbing with Sideways	17
2.2.3.1 Source Code Hill-climbing with Sideways	19
2.2.3.2 Deskripsi Fungsi Hill-climbing with Sideways	19
2.2.4 Random Restart Hill-climbing	21
2.2.4.1 Source Random Restart Hill-climbing	23
2.2.4.2 Deskripsi Fungsi Random Restart Hill-climbing	23
2.2.5 Simulated Annealing	25
2.2.5.1 Source Code Simulated Annealing	27
2.2.5.2 Deskripsi Kelas/Fungsi Simulated Annealing	27
2.2.6 Genetic Algorithm	29
2.2.6.1 Source Code Genetic Algorithm	29
2.2.6.2 Deskripsi Fungsi Genetic Algorithm	29
2.3 Hasil Eksperimen dan Analisis	29
2.3.1 Hasil Steepest Ascent Hill-Climbing	29
2.3.2 Hasil Stochastic Hill-climbing	29
2.3.3 Hasil Hill-climbing with Sideways	29
2.3.4 Hasil Random Restart Hill-climbing	29
2.3.5 Hasil Simulated Annealing	29
2.3.6 Hasil Genetic Algorithm	29
BAB III	30

KESIMPULAN & SARAN	30
3.1 Kesimpulan	30
3.2 Saran	30
PEMBAGIAN TUGAS	31
REFERENSI	32

BAB I

DESKRIPSI PERSOALAN

Permasalahan yang diangkat dalam tugas ini adalah penjadwalan kelas mingguan untuk sejumlah mata kuliah yang harus dipasangkan dengan waktu dan ruangan tertentu. Setiap kelas memiliki jumlah mahasiswa, jumlah SKS, dan membutuhkan sejumlah pertemuan yang disesuaikan dengan beban SKS-nya dalam satu minggu. Masalah muncul ketika beberapa keterbatasan dan kendala harus dipenuhi secara bersamaan, antara lain:

- **Keterbatasan ruangan**, setiap ruangan memiliki kapasitas tertentu dan tidak dapat digunakan oleh lebih dari satu kelas pada waktu yang sama.
- **Keterbatasan waktu**, jadwal perkuliahan berlangsung pada slot waktu tertentu dalam satu minggu sehingga penempatan kelas harus mempertimbangkan tumpang tindih antar waktu.
- **Keterlibatan mahasiswa**, seorang mahasiswa dapat mengambil beberapa mata kuliah yang berbeda, sehingga perlu dihindari benturan jadwal antar mata kuliah yang diambil oleh mahasiswa yang sama.
- **Distribusi prioritas**, beberapa mata kuliah memiliki tingkat prioritas berbeda bagi setiap mahasiswa, sehingga tabrakan jadwal pada mata kuliah dengan prioritas tinggi menjadi lebih kritis untuk dihindari.
- **Kapasitas ruangan**, apabila jumlah mahasiswa yang mengikuti kelas melebihi kapasitas ruangan, maka jadwal tersebut dianggap tidak valid atau menghasilkan penalti dalam evaluasi penjadwalan.

Dengan demikian, inti dari permasalahan ini adalah bagaimana menyusun kombinasi jadwal yang layak dan seimbang, sehingga setiap kelas memiliki waktu dan ruangan yang tidak saling bertabrakan, serta seluruh mahasiswa dapat mengikuti perkuliahan sesuai kapasitas dan prioritas masing-masing.

Selain itu, kompleksitas masalah penjadwalan ini meningkat secara signifikan seiring bertambahnya jumlah kelas, dosen, ruangan, dan slot waktu yang tersedia. Jumlah kemungkinan kombinasi jadwal yang valid bertumbuh secara eksponensial, sehingga metode pencarian solusi konvensional seperti brute force menjadi tidak efisien untuk digunakan. Oleh karena itu, dibutuhkan pendekatan heuristik yang mampu menelusuri ruang solusi besar secara efisien tanpa harus mengeksplorasi seluruh kemungkinan.

Pada konteks ini, algoritma local search menjadi pendekatan yang tepat karena mampu melakukan eksplorasi terhadap solusi dengan cara memperbaiki kandidat solusi secara bertahap melalui operasi kecil (neighboring moves). Algoritma seperti Steepest Ascent Hill-Climbing, Simulated Annealing, dan Genetic Algorithm dapat digunakan untuk mencari solusi penjadwalan yang mendekati optimal dengan waktu komputasi yang lebih efisien.

Permasalahan penjadwalan kelas ini kemudian dapat diformulasikan sebagai constraint optimization problem, di mana setiap pelanggaran terhadap batasan (seperti konflik waktu atau kapasitas ruangan) akan memberikan penalti tertentu, dan tujuan dari algoritma adalah meminimalkan total penalti tersebut. Dengan demikian, solusi terbaik adalah yang menghasilkan jadwal tanpa konflik atau dengan penalti seminimal mungkin, sambil tetap mempertahankan kelayakan dan keseimbangan distribusi kelas.

BAB II

PEMBAHASAN

2.1 Pemilihan Objective Function

Objective function atau fungsi objektif merupakan komponen inti dalam setiap permasalahan optimasi. Fungsi ini berperan sebagai ukuran kuantitatif yang menunjukkan seberapa baik atau “buruk” suatu solusi dibandingkan dengan solusi lain. Dalam konteks algoritma pencarian berbasis heuristik seperti Steepest Ascent Hill-Climbing, Simulated Annealing, maupun Genetic Algorithm, fungsi objektif digunakan untuk mengevaluasi kualitas solusi pada setiap langkah iterasi. Dengan kata lain, fungsi objektif bertugas mengubah kondisi kompleks suatu masalah dunia nyata menjadi nilai numerik tunggal yang bisa dibandingkan. Nilai ini dapat berupa:

- Nilai maksimum, jika tujuan pencarian adalah memaksimalkan efisiensi, utilitas, atau keuntungan.
- Nilai minimum, jika tujuan pencarian adalah meminimalkan kesalahan, konflik, atau penalti.

Dalam permasalahan penjadwalan kelas mingguan, fungsi objektif biasanya dirancang untuk meminimalkan jumlah konflik dan pelanggaran terhadap batasan-batasan jadwal, seperti tabrakan waktu antar kelas, kelebihan kapasitas ruangan, atau benturan jadwal mahasiswa. Semakin kecil nilai fungsi objektif, semakin baik kualitas jadwal yang dihasilkan.

Fungsi objektif yang kami gunakan dalam tugas ini adalah **jumlah waktu yang bertabrakan untuk setiap mahasiswa** (*total student time conflicts*). Inti dari fungsi ini adalah mengukur seberapa parah tingkat konflik jadwal yang dialami oleh mahasiswa akibat penempatan kelas yang tidak tepat.

Secara konseptual, setiap mahasiswa idealnya hanya dapat mengikuti satu mata kuliah pada satu slot waktu (misalnya, Selasa pukul 09.00-11.00). Jika seorang mahasiswa terdaftar pada dua atau lebih mata kuliah yang berada pada slot waktu yang sama, maka kondisi ini dianggap sebagai konflik. Setiap kali terjadi konflik, fungsi objektif akan menambahkan poin penalti yang merepresentasikan tingkat ketidakefisienan jadwal tersebut.

Contohnya, jika mahasiswa A memiliki dua mata kuliah yang dijadwalkan pada jam 09.00-11.00 hari Selasa, maka jadwal tersebut akan memberikan kontribusi **4 poin konflik** (konflik dihitung berdasarkan total jam mata kuliah yang bertabrakan) terhadap total nilai fungsi objektif. Apabila ada mahasiswa lain dengan konflik serupa, nilainya juga akan ditambahkan ke total, sehingga semakin banyak konflik yang terjadi, semakin besar nilai fungsi objektif yang dihasilkan.

Dengan demikian, fungsi objektif ini bersifat **minimization-based** semakin kecil nilai yang dihasilkan, semakin baik kualitas jadwal yang diusulkan. Nilai **0** menunjukkan bahwa tidak ada

mahasiswa yang mengalami tabrakan jadwal, yang berarti jadwal tersebut sepenuhnya valid dan optimal dari sisi distribusi waktu mahasiswa.

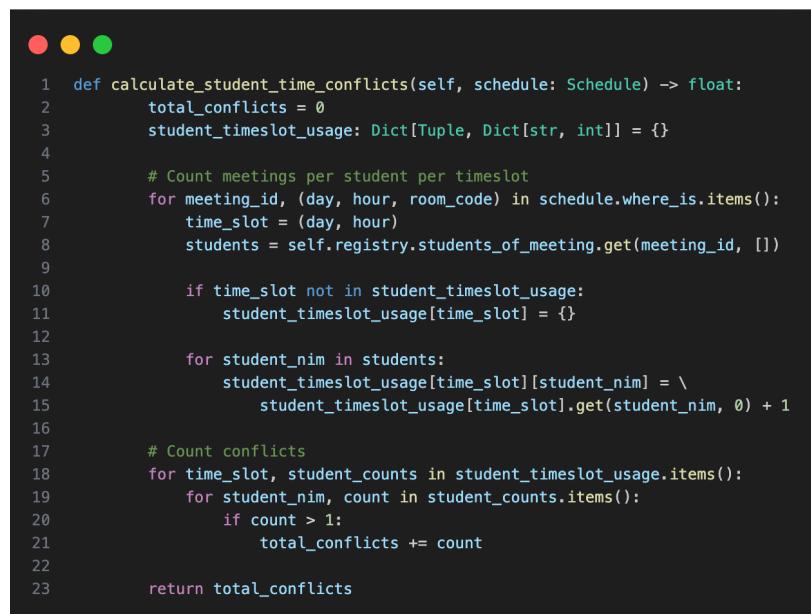
Dari sudut pandang algoritmik, fungsi ini memberikan ukuran yang jelas dan mudah dihitung untuk mengevaluasi suatu state (kandidat jadwal). Proses evaluasi dilakukan dengan menghitung berapa kali setiap mahasiswa muncul pada waktu yang sama di jadwal. Jika seorang mahasiswa terdaftar pada beberapa pertemuan dalam slot waktu identik, maka jumlah tersebut diakumulasikan sebagai konflik.

Pendekatan ini memiliki beberapa kelebihan:

- **Sederhana namun representatif**, mudah dihitung dan secara langsung mencerminkan kualitas jadwal dari perspektif mahasiswa.
- **Efisien untuk evaluasi berulang**, karena hanya bergantung pada data mahasiswa dan slot waktu, perhitungannya dapat dilakukan dengan kompleksitas waktu yang relatif rendah.
- **Relevan dengan tujuan praktis penjadwalan**, konflik antar kelas mahasiswa merupakan salah satu faktor utama yang menentukan keberhasilan sistem jadwal akademik.

Dengan memilih fungsi objektif ini, fokus algoritma local search menjadi jelas: **mengurangi jumlah konflik mahasiswa hingga seminimal mungkin** untuk mencapai jadwal yang layak dan efisien.

2.1.1 Source Code Objective Function



```
 1 def calculate_student_time_conflicts(self, schedule: Schedule) -> float:
 2     total_conflicts = 0
 3     student_timeslot_usage: Dict[Tuple, Dict[str, int]] = {}
 4
 5     # Count meetings per student per timeslot
 6     for meeting_id, (day, hour, room_code) in schedule.where_is.items():
 7         time_slot = (day, hour)
 8         students = self.registry.students_of_meeting.get(meeting_id, [])
 9
10         if time_slot not in student_timeslot_usage:
11             student_timeslot_usage[time_slot] = {}
12
13         for student_nim in students:
14             student_timeslot_usage[time_slot][student_nim] = \
15                 student_timeslot_usage[time_slot].get(student_nim, 0) + 1
16
17     # Count conflicts
18     for time_slot, student_counts in student_timeslot_usage.items():
19         for student_nim, count in student_counts.items():
20             if count > 1:
21                 total_conflicts += count
22
23
24     return total_conflicts
```

2.1.2 Deskripsi Fungsi Objective Function

Fungsi calculate_student_time_conflicts() bertugas untuk menghitung jumlah konflik waktu yang dialami oleh setiap mahasiswa dalam satu jadwal (schedule). Nilai yang dikembalikan oleh fungsi ini berupa angka (float), yang menunjukkan total tingkat konflik pada state jadwal yang sedang dievaluasi semakin kecil nilainya, semakin baik kualitas jadwal tersebut. Secara garis besar, fungsi ini bekerja melalui dua tahap utama:

1. Pencatatan penggunaan slot waktu oleh mahasiswa

Fungsi akan memeriksa seluruh pertemuan kelas (meeting) yang tercatat di jadwal. Untuk setiap pertemuan, diambil informasi hari dan jam pelaksanaan sebagai satu timeslot. Kemudian, fungsi menelusuri daftar mahasiswa yang terdaftar pada pertemuan tersebut melalui objek registry.

Setiap kali seorang mahasiswa ditemukan dalam suatu timeslot, nilai penggunaan waktunya pada slot tersebut akan dinaikkan satu poin. Hasil akhirnya adalah sebuah struktur data yang memetakan setiap timeslot terhadap daftar mahasiswa yang menggunakan beserta jumlah kelas yang diikuti pada slot tersebut.

2. Perhitungan total konflik antar kelas mahasiswa

Setelah data penggunaan waktu terkumpul, fungsi menghitung jumlah konflik dengan menelusuri setiap timeslot. Jika seorang mahasiswa tercatat memiliki lebih dari satu kelas pada timeslot yang sama (misalnya count > 1), maka hal tersebut dianggap sebagai konflik. Nilai count tersebut ditambahkan ke variabel total_conflicts, yang menyimpan akumulasi konflik dari seluruh mahasiswa dan slot waktu.

Secara konseptual, logika fungsi ini dapat diilustrasikan sebagai berikut:

- Mahasiswa dengan satu kelas pada suatu waktu -> tidak menambah konflik (0 poin)
- Mahasiswa dengan dua kelas pada waktu yang sama -> menambah 2 poin konflik
- Mahasiswa dengan tiga kelas pada waktu yang sama -> menambah 3 poin konflik, dan seterusnya

Dengan menggunakan fungsi ini, sistem dapat menilai seberapa “buruk” suatu jadwal dari sisi tumpang-tindih perkuliahan mahasiswa. Tujuan dari setiap algoritma optimasi yang diterapkan adalah menurunkan nilai total konflik seminimal mungkin, hingga diperoleh jadwal yang ideal dengan nilai fungsi objektif mendekati nol artinya tidak ada mahasiswa yang memiliki kelas pada waktu yang sama.

2.2 Penjelasan Implementasi Algoritma Local Search

Algoritma Local Search merupakan salah satu pendekatan dalam bidang Artificial Intelligence dan Optimization yang digunakan untuk menemukan solusi mendekati optimal (near-optimal solution) pada permasalahan dengan ruang solusi yang sangat besar. Berbeda dengan algoritma pencarian sistematis seperti Breadth-First Search atau Depth-First Search yang menelusuri seluruh ruang solusi, local search berfokus pada **perbaikan bertahap terhadap satu solusi kandidat yang sedang dievaluasi**.

Prinsip dasarnya adalah sebagai berikut:

1. Algoritma memulai proses dari sebuah **solusi awal** yang mungkin belum optimal.
2. Kemudian algoritma menghasilkan sekumpulan **solusi tetangga (neighbors)**, yaitu variasi kecil dari solusi saat ini yang diperoleh melalui perubahan terbatas (misalnya menukar jadwal dua kelas atau memindahkan satu pertemuan ke slot waktu lain).
3. Setiap solusi tetangga dievaluasi menggunakan **fungsi objektif** untuk mengukur kualitasnya.
4. Berdasarkan strategi yang digunakan, algoritma akan memilih salah satu solusi tetangga untuk dijadikan keadaan baru (current state) dan mengulangi proses tersebut sampai memenuhi kondisi terminasi misalnya ketika sudah tidak ditemukan solusi yang lebih baik, atau jumlah iterasi maksimum tercapai.

Tujuan utama dari local search bukanlah menemukan solusi optimal secara global, melainkan mencari solusi yang cukup baik dalam waktu komputasi yang wajar. Hal ini menjadikannya cocok untuk masalah-masalah NP-hard seperti penjadwalan kelas, di mana jumlah kemungkinan kombinasi solusi tumbuh secara eksponensial terhadap jumlah variabel (kelas, ruangan, waktu, dan mahasiswa).

Pendekatan ini memiliki kelebihan utama dalam efisiensi, karena:

- Tidak perlu menyimpan seluruh pohon pencarian.
- Dapat dengan cepat memperbaiki solusi melalui langkah-langkah lokal.
- Dapat digunakan dengan berbagai variasi teknik eksplorasi (misalnya probabilistik seperti Simulated Annealing atau evolusioner seperti Genetic Algorithm).

Namun demikian, local search juga memiliki keterbatasan yaitu kecenderungan **terjebak pada local optimum**, yaitu titik di mana tidak ada solusi tetangga yang lebih baik meskipun solusi tersebut belum optimal secara global. Untuk mengatasi hal ini, digunakan beberapa strategi tambahan seperti random restart, acceptance probability (dalam Simulated Annealing), atau populasi dan crossover (dalam Genetic Algorithm).

2.2.1 Steepest Ascent Hill-climbing

Steepest Ascent Hill-Climbing merupakan salah satu varian paling dasar dan deterministik dari algoritma Local Search. Tujuan utama algoritma ini adalah mencari solusi terbaik secara bertahap dengan selalu memilih tetangga yang memberikan perbaikan terbesar terhadap nilai fungsi objektif.

Prinsip kerja algoritma ini menyerupai analogi mendaki bukit (hill climbing) adalah setiap langkah diibaratkan sebagai upaya untuk naik ke posisi yang lebih tinggi (dalam konteks maksimisasi) atau turun ke posisi yang lebih rendah (dalam konteks minimisasi fungsi objektif). Dalam tugas ini, karena fungsi objektif bertujuan **meminimalkan jumlah konflik antar jadwal mahasiswa**, maka arah pencarian dilakukan menuju nilai fungsi objektif yang semakin kecil.

Langkah-langkah umum algoritma Steepest Ascent Hill-Climbing adalah sebagai berikut:

1. **Inisialisasi solusi awal**, algoritma dimulai dengan satu state awal yang merepresentasikan penjadwalan acak.
2. **Pembangkitan tetangga (neighbor generation)**, dari solusi saat ini, algoritma menghasilkan sekumpulan state tetangga dengan melakukan perubahan kecil (misalnya menukar slot waktu dua kelas atau memindahkan satu kelas ke waktu lain).
3. **Evaluasi fungsi objektif**, setiap neighbor dievaluasi menggunakan fungsi objektif untuk menghitung total konflik mahasiswa.
4. **Pemilihan solusi terbaik**, dari seluruh neighbor, algoritma memilih solusi dengan **peningkatan terbesar (atau penurunan konflik terbesar)** dibandingkan solusi saat ini.
5. **Pergantian state**, jika solusi terbaik tersebut lebih baik daripada solusi sebelumnya, algoritma berpindah ke state tersebut dan mengulang proses dari langkah kedua. Jika tidak ada perbaikan, proses berhenti karena telah mencapai local optimum.

Kelebihan dari algoritma ini terletak pada **sifat deterministik dan efisiensinya** dalam menemukan solusi yang lebih baik secara cepat. Karena selalu memilih perbaikan terbesar pada setiap iterasi, algoritma ini cenderung konvergen lebih cepat dibandingkan varian *stochastic*. Namun, kekurangannya adalah mudah terjebak pada **local optimum**, yaitu kondisi ketika semua solusi tetangga memiliki nilai yang lebih buruk, padahal masih ada solusi global yang lebih baik di tempat lain dalam ruang solusi.

2.2.1.1 Source Code Steepest Ascent Hill-climbing

```
1  class SteepestAscentHillClimbing:
2      def __init__(self, registry: Registry, max_iterations: Optional[int] = None):
3          self.registry = registry
4          self.max_iterations = max_iterations
5          self.objective = ScheduleObjective(registry)
6
7      def run(self) -> tuple[Schedule, Schedule, float, list, float, int]:
8          start_time = time.time()
9          initial_schedule = Schedule.random_initial_assignment(self.registry)
10         current = initial_schedule
11         current_score = self.objective.evaluate(current)
12
13         history = [current_score]
14         iteration = 0
15
16         while self.max_iterations is None or iteration < self.max_iterations:
17             iteration += 1
18
19             neighbors = generate_neighbors(current, self.registry)
20
21             if not neighbors:
22                 break
23
24             best_neighbor = None
25             best_score = current_score
26
27             for neighbor in neighbors:
28                 score = self.objective.evaluate(neighbor)
29                 if score < best_score:
30                     best_score = score
31                     best_neighbor = neighbor
32
33             if best_neighbor is None:
34                 break
35
36             current = best_neighbor
37             current_score = best_score
38             history.append(current_score)
39
40             if current_score == 0:
41                 break
42
43         end_time = time.time()
44         duration = end_time - start_time
45         return initial_schedule, current, current_score, history, duration, iteration
```

2.2.1.2 Deskripsi Fungsi Steepest Ascent Hill-climbing

Kelas SteepestAscentHillClimbing merupakan implementasi dari algoritma Steepest Ascent Hill-Climbing yang digunakan untuk mengoptimalkan jadwal kuliah dengan cara **meminimalkan jumlah konflik waktu antar mahasiswa**.

Fungsi utama dari kelas ini adalah `run()`, yang menjalankan proses pencarian solusi terbaik berdasarkan perbaikan nilai fungsi objektif. Struktur algoritma ini mengikuti prinsip dasar local search, yaitu memperbaiki solusi secara bertahap dengan mengevaluasi tetangga terdekat dan memilih solusi dengan nilai terbaik.

Secara garis besar, logika kerja fungsi run() dapat dijelaskan sebagai berikut:

1. Inisialisasi Jadwal Awal

Algoritma dimulai dengan membuat jadwal acak menggunakan `Schedule.random_initial_assignment()`. Jadwal ini menjadi state awal (initial schedule) yang akan diperbaiki secara iteratif. Nilai fungsi objektif dari jadwal awal dihitung melalui `self.objective.evaluate()`, menghasilkan ukuran awal tingkat konflik mahasiswa.

2. Iterasi dan Pembangkitan Solusi Tetangga

Pada setiap iterasi, algoritma membangkitkan sekumpulan neighbor states menggunakan fungsi `generate_neighbors()`. Setiap tetangga merupakan variasi kecil dari jadwal saat ini misalnya, satu kelas dipindahkan ke slot waktu atau ruangan lain. Jika tidak ada tetangga yang dapat dihasilkan, proses berhenti.

3. Evaluasi dan Pemilihan Solusi Terbaik

Seluruh neighbor dievaluasi dengan menghitung nilai fungsi objektifnya. Dari hasil evaluasi tersebut, algoritma memilih **neighbor dengan nilai konflik paling kecil (best_score)** sebagai `best_neighbor`. Jika tidak ada neighbor yang lebih baik dari jadwal saat ini, berarti algoritma telah mencapai local optimum, dan proses berhenti.

4. Perubahan State dan Pencatatan Perkembangan

Jika ditemukan neighbor yang lebih baik, algoritma mengganti current state dengan `best_neighbor`, memperbarui `current_score`, dan menyimpannya ke dalam daftar history untuk mencatat perkembangan skor di setiap iterasi.

5. Kondisi Terminasi

Proses berlanjut hingga salah satu dari dua kondisi tercapai:

- 1) Jumlah iterasi telah mencapai batas maksimum (`max_iterations`), atau
- 2) Nilai konflik sudah mencapai **0**, artinya jadwal tanpa tabrakan berhasil ditemukan.

Terminasi dan Pengembalian Hasil

Setelah pencarian selesai, fungsi `run()` mengembalikan beberapa informasi penting:

- Jadwal awal (`initial_schedule`),
- Jadwal akhir yang telah dioptimasi (`current`),
- Nilai fungsi objektif terakhir (`current_score`),
- Riwayat perubahan skor selama iterasi (`history`),
- Durasi eksekusi algoritma (`duration`), dan
- Jumlah iterasi yang dilakukan (`iteration`).

2.2.2 Stochastic Hill-climbing

Stochastic Hill-Climbing merupakan varian dari algoritma Hill-Climbing yang memperkenalkan unsur **acak (stochastic)** dalam proses pemilihan solusi tetangga. Jika pada Steepest Ascent Hill-Climbing algoritma selalu memilih tetangga terbaik secara deterministik, maka Stochastic Hill-Climbing memilih **satu tetangga secara acak dari sekumpulan tetangga yang memiliki nilai fungsi objektif lebih baik** daripada solusi saat ini.

Tujuan utama pendekatan ini adalah untuk **mengurangi kemungkinan terjebak pada local optimum**, dengan cara memberikan variasi arah pencarian. Dengan kata lain, Stochastic Hill-Climbing tidak selalu bergerak ke arah “tercuram” seperti Steepest Ascent, melainkan memungkinkan langkah-langkah yang lebih acak selama solusi tersebut tetap memperbaiki nilai fungsi objektif.

Langkah-langkah umum algoritma **Stochastic Hill-Climbing** adalah sebagai berikut:

1. **Inisialisasi solusi awal**, algoritma dimulai dengan membangkitkan satu state acak (jadwal awal) yang berisi penempatan kelas, ruangan, dan waktu secara acak. Solusi ini kemudian dievaluasi menggunakan fungsi objektif untuk memperoleh nilai awal (jumlah total konflik mahasiswa).
2. **Pembangkitan tetangga (neighbor generation)**, dari solusi saat ini, algoritma membangkitkan sekumpulan neighbor dengan melakukan perubahan kecil terhadap jadwal, misalnya menukar slot waktu dua kelas atau memindahkan satu pertemuan ke slot waktu lain.
3. **Seleksi acak solusi lebih baik**, dari seluruh neighbor yang memiliki nilai fungsi objektif lebih baik dibandingkan solusi saat ini, algoritma memilih **satu tetangga secara acak** untuk dijadikan state berikutnya. Pemilihan acak ini memungkinkan algoritma menjelajahi ruang solusi yang lebih luas dibandingkan hanya memilih perbaikan terbesar.
4. **Pembaruan state**, state baru yang terpilih menggantikan state sebelumnya, kemudian algoritma mengulangi proses pembangkitan dan evaluasi tetangga sampai tidak ditemukan solusi yang lebih baik atau batas iterasi maksimum tercapai.
5. **Terminasi**, algoritma berhenti ketika tidak ada lagi neighbor yang menghasilkan perbaikan nilai fungsi objektif, atau ketika jumlah iterasi maksimum tercapai.

Kelebihan dari algoritma ini terletak pada **fleksibilitas dan kemampuan eksplorasinya**. Dengan memasukkan unsur acak dalam pemilihan neighbor, Stochastic Hill-Climbing memiliki peluang lebih besar untuk keluar dari local optimum dibandingkan varian deterministiknya. Namun, karena pemilihan tetangga dilakukan secara acak, algoritma ini

juga memiliki **stabilitas hasil yang lebih rendah**, artinya hasil akhirnya dapat berbeda-beda pada setiap eksekusi tergantung pada distribusi peluang pemilihan tetangga.

2.2.2.1 Source Code Stochastic Hill-climbing



```
1 class StochasticHillClimbing:
2     def __init__(self, registry: Registry, max_iterations: Optional[int] = None):
3         self.registry = registry
4         self.max_iterations = max_iterations
5         self.objective = ScheduleObjective(registry)
6
7     def run(self) -> tuple[Schedule, Schedule, float, list, float, int]:
8         start_time = time.time()
9         initial_schedule = Schedule.random_initial_assignment(self.registry)
10        current = initial_schedule
11        current_score = self.objective.evaluate(current)
12        history = [current_score]
13        iteration = 0
14
15        while self.max_iterations is None or iteration < self.max_iterations:
16            iteration += 1
17
18            neighbors = generate_neighbors(current, self.registry)
19
20            if not neighbors:
21                break
22
23            # Pilih satu neighbor secara acak
24            next_schedule = random.choice(neighbors)
25            next_score = self.objective.evaluate(next_schedule)
26
27            # Hanya update jika neighbor lebih baik
28            if next_score < current_score:
29                current = next_schedule
30                current_score = next_score
31
32            history.append(current_score)
33
34            if current_score == 0:
35                break
36
37        end_time = time.time()
38        duration = end_time - start_time
39        return initial_schedule, current, current_score, history, duration, iteration
```

2.2.2.2 Deskripsi Fungsi Stochastic Hill-climbing

Kelas StochasticHillClimbing merepresentasikan implementasi dari algoritma Stochastic Hill-Climbing, yaitu varian local search yang menggunakan pemilihan tetangga secara acak untuk memperluas ruang pencarian solusi. Tujuan utamanya

adalah menurunkan nilai fungsi objektif, yang dalam kasus ini merupakan jumlah konflik jadwal antar mahasiswa.

Fungsi utama dalam kelas ini adalah run(), yang mengatur seluruh proses iterasi pencarian solusi. Secara konseptual, alur kerjanya dapat dijelaskan sebagai berikut:

1. **Inisialisasi Solusi Awal**

Algoritma dimulai dengan membangkitkan jadwal awal (initial_schedule) secara acak melalui pemanggilan Schedule.random_initial_assignment(). Jadwal ini dievaluasi menggunakan self.objective.evaluate() untuk menghitung nilai awal fungsi objektif (current_score), yang menggambarkan jumlah konflik mahasiswa pada penjadwalan awal. Nilai ini kemudian disimpan sebagai kondisi awal untuk perbandingan di iterasi berikutnya.

2. **Proses Iteratif Pencarian Solusi Tetangga**

Pada setiap iterasi, algoritma menghasilkan sekumpulan neighbors dengan fungsi generate_neighbors(). Setiap neighbor merupakan variasi kecil dari jadwal saat ini misalnya, pemindahan waktu satu kelas atau pertukaran antara dua pertemuan kuliah.

3. **Pemilihan Tetangga Secara Acak**

Tidak seperti Steepest Ascent Hill-Climbing yang memilih tetangga terbaik, algoritma ini memilih **satu tetangga secara acak** dari seluruh kandidat tetangga yang dihasilkan. Pemilihan acak dilakukan dengan random.choice(neighbors), yang memberikan elemen eksploratif terhadap arah pencarian.

4. **Evaluasi dan Pembaruan Solusi**

Setelah satu neighbor dipilih, jadwal tersebut dievaluasi kembali untuk memperoleh nilai fungsi objektif baru (next_score). Jika nilai next_score **lebih kecil daripada current_score**, artinya jadwal baru memiliki konflik yang lebih sedikit, maka neighbor tersebut diterima sebagai solusi baru. Proses ini dicatat dalam variabel history agar perkembangan nilai fungsi objektif sepanjang iterasi dapat dianalisis.

5. **Kondisi Terminasi**

Iterasi berlangsung hingga salah satu kondisi berikut tercapai:

- Tidak ada neighbor yang dapat dibangkitkan (tidak ada langkah perbaikan yang tersisa), atau
- Nilai fungsi objektif mencapai **0**, menandakan jadwal tanpa konflik, atau
- Jumlah iterasi mencapai batas maksimum yang telah ditentukan (max_iterations).

Terminasi dan Pengembalian Hasil

Setelah proses selesai, fungsi run() mengembalikan sejumlah informasi penting:

- Jadwal awal sebelum proses optimasi (initial_schedule)
- Jadwal terbaik yang ditemukan selama proses (current)
- Nilai akhir fungsi objektif (current_score)
- Daftar perubahan nilai fungsi objektif sepanjang iterasi (history)
- Lama waktu eksekusi algoritma (duration)
- Jumlah iterasi yang dijalankan (iteration)

2.2.3 Hill-climbing with Sideways

Hill-Climbing with Sideways Moves merupakan varian dari algoritma Hill-Climbing yang dirancang untuk mengatasi masalah plateau dan local optimum dalam proses pencarian solusi. Pada varian Steepest Ascent dan Stochastic Hill-Climbing, algoritma akan berhenti apabila tidak menemukan tetangga (neighbor) dengan nilai fungsi objektif yang lebih baik. Namun, dalam praktiknya, sering kali algoritma menemui kondisi di mana banyak tetangga memiliki nilai fungsi objektif yang sama dengan solusi saat ini. Kondisi ini disebut plateau, dan dapat menyebabkan algoritma berhenti terlalu cepat sebelum mencapai solusi optimal.

Untuk mengatasi hal tersebut, Hill-Climbing with Sideways Moves memperkenankan **langkah ke samping (sideways move)**, yaitu berpindah ke tetangga dengan **nilai fungsi objektif yang sama** dengan keadaan saat ini. Dengan memberikan batas maksimum jumlah langkah ke samping yang boleh dilakukan, algoritma dapat tetap menjelajahi ruang solusi meskipun tidak ada perbaikan langsung, sehingga memiliki peluang lebih besar untuk menemukan solusi yang lebih baik di kemudian hari.

Langkah-langkah umum dari algoritma Hill-Climbing with Sideways Moves adalah sebagai berikut:

1. **Inisialisasi solusi awal**, algoritma memulai pencarian dari solusi acak (initial schedule) yang kemudian dievaluasi untuk memperoleh nilai awal fungsi objektif, yaitu jumlah total konflik mahasiswa dalam jadwal tersebut.
2. **Pembangkitan tetangga (neighbor generation)**, algoritma menghasilkan beberapa neighbor dari solusi saat ini dengan melakukan perubahan kecil, misalnya memindahkan waktu kelas atau menukar ruangan antar pertemuan.
3. **Pemilihan tetangga terbaik**, dari sekumpulan neighbor, algoritma memilih neighbor dengan nilai fungsi objektif paling kecil (jumlah konflik paling sedikit).
4. **Pergerakan (move)**
 - Jika nilai neighbor lebih baik dari current state, algoritma berpindah ke neighbor tersebut.

- Jika nilai neighbor **sama**, maka algoritma boleh melakukan langkah ke samping (sideways move) hingga batas tertentu (misalnya `max_sideways_moves`).
 - Jika tidak ada neighbor yang lebih baik atau setara, proses berhenti.
5. **Terminasi**, algoritma berhenti ketika tidak ditemukan neighbor yang lebih baik atau setara, atau ketika jumlah langkah ke samping telah mencapai batas maksimum yang diperbolehkan, atau ketika batas iterasi maksimum tercapai.

Kelebihan utama dari algoritma ini adalah **kemampuannya keluar dari plateau**, yaitu area datar pada lanskap solusi di mana semua tetangga memiliki nilai sama. Dengan memperbolehkan sejumlah langkah ke samping, algoritma memiliki kesempatan untuk menemukan jalur menuju area solusi yang lebih baik secara global. Namun, kelemahannya adalah jika batas sideways move terlalu besar, algoritma bisa menghabiskan waktu berputar di area yang tidak menghasilkan perbaikan berarti. Oleh karena itu, pemilihan batas langkah ke samping menjadi faktor penting dalam menjaga keseimbangan antara eksplorasi dan efisiensi waktu.

2.2.3.1 Source Code Hill-climbing with Sideways

```
1  class HillClimbingSidewaysMove:
2      def __init__(self, registry: Registry, max_consecutive_sideways: int, max_total_sideways: int, max_iterations: Optional[int] = None):
3          self.registry = registry
4          self.max_consecutive_sideways = max_consecutive_sideways
5          self.max_total_sideways = max_total_sideways
6          self.max_iterations = max_iterations
7          self.objective = ScheduleObjective(registry)
8
9      def run(self) -> tuple[Schedule, Schedule, float, list, float, int]:
10         start_time = time.time()
11         initial_schedule = Schedule.random_initial_assignment(self.registry)
12         current = initial_schedule
13         current_score = self.objective.evaluate(current)
14
15         history = [current_score]
16         iteration = 0
17         consecutive_sideways = 0
18         total_sideways = 0
19
20         while self.max_iterations is None or iteration < self.max_iterations:
21             iteration += 1
22
23             neighbors = generate_neighbors(current, self.registry)
24
25             if not neighbors:
26                 break
27
28             best_neighbor = None
29             best_score = current_score
30
31             for neighbor in neighbors:
32                 score = self.objective.evaluate(neighbor)
33                 if score <= best_score:
34                     best_score = score
35                     best_neighbor = neighbor
36
37             if best_neighbor is None:
38                 break
39
40             if best_score == current_score:
41                 consecutive_sideways += 1
42                 total_sideways += 1
43
44                 if consecutive_sideways >= self.max_consecutive_sideways:
45                     break
46
47                 if total_sideways >= self.max_total_sideways:
48                     break
49             else:
50                 consecutive_sideways = 0
51
52             current = best_neighbor
53             current_score = best_score
54             history.append(current_score)
55
56             if current_score == 0:
57                 break
58
59         end_time = time.time()
60         duration = end_time - start_time
61
62         return initial_schedule, current, current_score, history, duration, iteration
```

2.2.3.2 Deskripsi Fungsi Hill-climbing with Sideways

Kelas HillClimbingSidewaysMove merupakan implementasi dari algoritma Hill-Climbing with Sideways Moves, yaitu varian local search yang memungkinkan pergerakan ke solusi tetangga dengan nilai fungsi objektif yang

sama untuk mengatasi kondisi plateau dan local optimum. Algoritma ini digunakan untuk meminimalkan jumlah konflik jadwal mahasiswa dengan tetap memberikan ruang eksplorasi pada lanskap solusi yang datar.

Fungsi utama dalam kelas ini adalah run(), yang menjalankan seluruh proses iteratif pencarian solusi terbaik. Berikut adalah deskripsi dari setiap tahap dalam fungsinya:

1. Inisialisasi Solusi Awal

Proses dimulai dengan pembangkitan jadwal awal secara acak menggunakan Schedule.random_initial_assignment(self.registry). Jadwal awal ini kemudian dievaluasi dengan self.objective.evaluate() untuk menghitung nilai awal fungsi objektif (current_score), yaitu jumlah konflik mahasiswa pada penjadwalan awal. Nilai ini disimpan sebagai kondisi awal, sementara daftar history digunakan untuk merekam perkembangan nilai fungsi objektif pada setiap iterasi.

2. Parameter Pengendali Pergerakan Samping (Sideways Moves)

Dua parameter penting digunakan untuk mengontrol jumlah langkah ke samping:

- 1) max_consecutive_sideways: batas maksimum jumlah langkah ke samping **berturut-turut** yang diperbolehkan.
- 2) max_total_sideways: batas maksimum **total langkah ke samping** yang dapat dilakukan selama seluruh proses pencarian.

Kedua batas ini memastikan algoritma tidak terus berpindah di area datar tanpa menghasilkan perbaikan.

3. Perulangan Pencarian Solusi (Iterasi)

Proses pencarian dilakukan selama jumlah iterasi belum mencapai max_iterations atau hingga tidak ada tetangga (neighbor) yang dapat dibangkitkan. Pada setiap iterasi:

- Sekumpulan neighbors dihasilkan menggunakan generate_neighbors(current, self.registry).
- Setiap neighbor dievaluasi dengan self.objective.evaluate(neighbor) untuk menentukan nilai fungsi objektifnya.
- Dari semua neighbor, dipilih satu best_neighbor yang memiliki nilai fungsi objektif paling kecil (terbaik).

4. Evaluasi dan Penanganan Langkah ke Samping (Sideways Handling)

Setelah best_neighbor ditemukan, algoritma membandingkan nilainya (best_score) dengan nilai saat ini (current_score):

- Jika `best_score` **lebih kecil**, artinya solusi lebih baik, maka algoritma berpindah ke `best_neighbor` dan mereset penghitung sideways move.
- Jika `best_score` **sama**, algoritma menganggapnya sebagai sideways move, menambah penghitung `consecutive_sideways` dan `total_sideways` sebesar 1.
- Jika jumlah langkah ke samping melebihi `max_consecutive_sideways` atau `max_total_sideways`, maka proses pencarian dihentikan.

5. Pembaruan dan Pencatatan Nilai

Setelah pergerakan dilakukan, nilai `current` dan `current_score` diperbarui, serta nilai baru ditambahkan ke daftar `history`. Jika pada suatu iterasi nilai fungsi objektif mencapai 0 (artinya tidak ada konflik jadwal sama sekali), algoritma berhenti lebih awal.

Terminasi dan Pengembalian Hasil

Setelah kondisi berhenti tercapai, fungsi `run()` mengembalikan enam komponen hasil:

- 1) Jadwal awal sebelum proses optimasi(`initial_schedule`),
- 2) Jadwal terbaik yang diperoleh(`current`),
- 3) Nilai akhir fungsi objektif(`current_score`),
- 4) Daftar perubahan nilai fungsi objektif selama iterasi(`history`),
- 5) Waktu total eksekusi algoritma(`duration`), dan
- 6) Jumlah iterasi yang dijalankan(`iteration`)

2.2.4 Random Restart Hill-climbing

Random-Restart Hill-Climbing merupakan perluasan dari algoritma Hill-Climbing yang bertujuan untuk mengurangi risiko terjebak pada local optimum dengan melakukan proses pencarian berulang kali dari berbagai titik awal yang berbeda. Pada dasarnya, algoritma ini menjalankan Hill-Climbing berkali-kali dengan solusi awal yang dibangkitkan secara acak, kemudian memilih hasil terbaik dari seluruh percobaan.

Setiap kali proses Hill-Climbing selesai (baik karena mencapai local optimum maupun plateau), algoritma akan **melakukan restart**, yaitu membangkitkan solusi awal baru dan menjalankan pencarian ulang. Strategi ini membantu memperluas ruang eksplorasi solusi, karena setiap percobaan dimulai dari posisi yang berbeda dalam lanskap fungsi objektif.

Langkah-langkah umum algoritma **Random-Restart Hill-Climbing** adalah sebagai berikut:

1. **Inisialisasi jumlah restart**, algoritma menentukan berapa kali pencarian akan dilakukan (misalnya max_restarts).
2. **Menjalankan Hill-Climbing biasa**, untuk setiap percobaan, algoritma menjalankan varian Hill-Climbing (misalnya Steepest Ascent) dengan solusi awal acak.
3. **Evaluasi hasil tiap percobaan**, setiap hasil dari proses Hill-Climbing dievaluasi berdasarkan nilai fungsi objektifnya (misalnya jumlah konflik mahasiswa).
4. **Pemilihan hasil terbaik**, setelah seluruh percobaan selesai, algoritma memilih solusi dengan nilai fungsi objektif terendah sebagai solusi akhir terbaik.

Dengan pendekatan ini, Random-Restart Hill-Climbing tidak bergantung pada satu titik awal pencarian, melainkan mencoba menemukan solusi optimal global dengan menjelajahi berbagai area dalam ruang solusi.

Kelebihan dari algoritma ini terletak pada kemampuannya untuk **menghindari jebakan local optimum** dan menghasilkan solusi yang lebih stabil karena pencarian dilakukan dari banyak titik awal berbeda. Pendekatan ini juga membuat hasil akhir menjadi lebih robust terhadap variasi distribusi solusi awal dan bentuk lanskap fungsi objektif yang kompleks. Sementara itu, kekurangan dari Random-Restart Hill-Climbing adalah **waktu eksekusinya yang lebih lama** karena algoritma menjalankan beberapa kali proses pencarian. Selain itu, meskipun peluang menemukan solusi global meningkat, algoritma ini tetap **tidak menjamin hasil terbaik secara mutlak**, karena efektivitasnya sangat bergantung pada jumlah restart dan variasi solusi awal yang dihasilkan.

2.2.4.1 Source Code Random Restart Hill-climbing

```
1 def run(self) -> tuple[Schedule, Schedule, float, list, int, float, list]:
2     start_time = time.time()
3     global_best_schedule = None
4     global_best_score = float('inf')
5     global_history = []
6     iterations_list = []
7
8     initial_schedule = None
9
10    for restart in range(self.max_restarts):
11        current = Schedule.random_initial_assignment(self.registry)
12        current_score = self.objective.evaluate(current)
13
14        local_history = [current_score] # History for this restart
15
16        if restart == 0:
17            initial_schedule = current
18
19        iteration = 0
20
21        while self.max_iterations_per_restart is None or iteration < self.max_iterations_per_restart:
22            iteration += 1
23
24            neighbors = generate_neighbors(current, self.registry)
25
26            if not neighbors:
27                break
28
29            best_neighbor = None
30            best_score = current_score
31
32            for neighbor in neighbors:
33                score = self.objective.evaluate(neighbor)
34                if score < best_score:
35                    best_score = score
36                    best_neighbor = neighbor
37
38            if best_neighbor is None:
39                break
40
41            current = best_neighbor
42            current_score = best_score
43            local_history.append(current_score)
44
45            if current_score == 0:
46                break
47
48            iterations_list.append(iteration)
49            global_history.append(local_history) # Append history of this restart
50
51            if current_score < global_best_score:
52                global_best_score = current_score
53                global_best_schedule = current
54
55            if global_best_score == 0:
56                break
57
58    end_time = time.time()
59    duration = end_time - start_time
60
61    return initial_schedule, global_best_schedule, global_best_score, global_history, restart, duration, iterations_list
```

2.2.4.2 Deskripsi Fungsi Random Restart Hill-climbing

Kelas RandomRestartHillClimbing merupakan implementasi dari algoritma Random-Restart Hill-Climbing yang digunakan untuk mengoptimalkan penjadwalan kuliah dengan cara **meminimalkan jumlah konflik waktu antar**

mahasiswa melalui proses local search yang dijalankan berulang kali dari berbagai titik awal yang berbeda.

Tujuan utama algoritma ini adalah untuk menghindari jebakan local optimum yang sering terjadi pada algoritma Hill-Climbing biasa, dengan cara melakukan restart beberapa kali menggunakan solusi awal acak, lalu memilih hasil terbaik dari seluruh percobaan tersebut.

Fungsi utama dari kelas ini adalah run(), yang menjalankan keseluruhan proses pencarian solusi terbaik dengan melakukan beberapa kali iterasi Hill-Climbing dari berbagai solusi awal.

Secara garis besar, logika kerja fungsi run() dapat dijelaskan sebagai berikut:

1. Inisialisasi Variabel dan Jadwal Awal

Proses dimulai dengan mendefinisikan variabel global seperti global_best_schedule, global_best_score, dan global_history yang berfungsi untuk menyimpan hasil terbaik selama seluruh proses restart. Pada restart pertama, jadwal awal dibangkitkan secara acak menggunakan Schedule.random_initial_assignment(self.registry), kemudian dievaluasi dengan self.objective.evaluate(current) untuk menghitung jumlah konflik awal.

2. Iterasi dan Pembangkatan Solusi Tetangga

Untuk setiap restart, algoritma melakukan proses Hill-Climbing konvensional. Pada setiap iterasi, dibangkitkan sekumpulan neighbor menggunakan generate_neighbors(). Setiap neighbor mewakili variasi kecil dari jadwal sebelumnya, seperti memindahkan satu kelas ke waktu atau ruangan lain. Jika tidak ada neighbor yang dapat dibangkitkan, proses restart tersebut berhenti.

3. Evaluasi dan Pemilihan Solusi Terbaik

Semua neighbor yang dihasilkan dievaluasi menggunakan fungsi objektif untuk menghitung jumlah konflik. Dari hasil evaluasi tersebut, algoritma memilih neighbor dengan nilai konflik paling kecil sebagai best_neighbor. Jika tidak ditemukan solusi yang lebih baik, Hill-Climbing pada restart tersebut dianggap selesai.

4. Perubahan State dan Pembaruan Hasil Global

Jika ditemukan neighbor yang lebih baik, algoritma memperbarui current dengan best_neighbor, mengganti nilai current_score, dan mencatatnya ke dalam riwayat local_history. Setelah setiap restart selesai, hasil terbaik dibandingkan dengan global_best_score; jika lebih baik, maka

`global_best_schedule` diperbarui. Riwayat perkembangan hasil tiap restart disimpan dalam `global_history` untuk keperluan analisis.

5. Kondisi Terminasi

Proses pencarian berlanjut hingga seluruh jumlah restart telah dijalankan atau solusi dengan konflik 0 telah ditemukan. Jika nilai fungsi objektif mencapai 0, artinya jadwal tanpa konflik berhasil dihasilkan, dan proses langsung dihentikan sebelum semua restart dijalankan.

Terminasi dan Pengembalian Hasil

Setelah seluruh proses pencarian selesai, fungsi `run()` akan mengembalikan beberapa informasi penting, yaitu:

- Jadwal awal dari restart pertama (`initial_schedule`),
- Jadwal terbaik yang ditemukan (`global_best_schedule`),
- Nilai fungsi objektif terbaik (`global_best_score`),
- Riwayat hasil setiap restart (`global_history`),
- Jumlah restart yang dijalankan (`restart`),
- Durasi waktu eksekusi algoritma (`duration`),
- Jumlah iterasi pada tiap restart (`iterations_list`).

2.2.5 Simulated Annealing

Simulated Annealing (SA) merupakan algoritma *local search* yang terinspirasi dari proses annealing dalam bidang fisika, yaitu proses pendinginan logam secara perlahan untuk memperoleh struktur yang stabil dengan energi paling rendah.

Dalam konteks optimasi, prinsip tersebut diterapkan untuk mencari solusi dengan **nilai fungsi objektif minimum**, di mana konsep “pendinginan” digunakan untuk **mengontrol seberapa besar peluang algoritma menerima solusi yang lebih buruk** selama proses pencarian.

Berbeda dari algoritma *Hill-Climbing* yang hanya menerima solusi baru jika lebih baik, *Simulated Annealing* sesekali **menerima solusi yang lebih buruk secara terkendali**. Tujuan dari mekanisme ini adalah untuk **menghindari jebakan local optimum** dan memperluas ruang eksplorasi solusi sebelum akhirnya mencapai konvergensi ke solusi terbaik global.

Prinsip Dasar Simulated Annealing

1. Inisialisasi Suhu (Temperature Initialization)

Algoritma dimulai dengan menentukan suhu awal (T). Suhu ini menunjukkan tingkat kebebasan algoritma untuk menerima solusi yang lebih buruk. Pada awal

proses, suhu tinggi membuat algoritma lebih fleksibel dalam mengeksplorasi berbagai solusi.

2. Pembangkitan dan Evaluasi Solusi Baru

Dari solusi saat ini (current state), algoritma menghasilkan solusi tetangga (neighbor) dengan melakukan perubahan kecil, seperti memindahkan atau menukar slot waktu tertentu. Nilai fungsi objektif dari solusi baru kemudian dihitung untuk dibandingkan dengan nilai sebelumnya.

3. Keputusan Penerimaan Solusi (Acceptance Probability)

- Jika solusi baru lebih baik, maka solusi tersebut diterima.
- Jika solusi baru lebih buruk, algoritma masih dapat menerimanya dengan peluang tertentu yang dihitung berdasarkan rumus:

$$P = \exp(-(\delta_E) / T)$$

di mana δ_E adalah selisih nilai fungsi objektif antara solusi baru dan solusi lama. Semakin besar δ_E (semakin buruk solusi baru) dan semakin kecil T , maka peluang penerimaan semakin kecil.

4. Pendinginan (Cooling Schedule)

Setelah setiap iterasi, suhu T akan diturunkan secara bertahap berdasarkan rumus umum:

$$T_{\text{new}} = \alpha * T_{\text{current}}$$

dengan α adalah konstanta antara 0 dan 1. Penurunan suhu ini membuat algoritma semakin selektif terhadap solusi buruk seiring berjalannya waktu. Pada tahap awal (T tinggi), algoritma fokus pada eksplorasi, sedangkan pada tahap akhir (T rendah), algoritma fokus pada eksplotasi atau penyempurnaan solusi.

5. Terminasi

Proses pencarian berhenti ketika suhu sudah mencapai nilai minimum (T_{min}) atau jumlah iterasi telah melewati batas maksimum.

2.2.5.1 Source Code Simulated Annealing

```
1  class SimulatedAnnealing:
2      def __init__(self, registry: Registry, max_iterations: Optional[int] = None, initial_temp: float = 100.0, cooling_rate: float = 0.99, random_func: Optional[callable] = None):
3          self.registry = registry
4          self.max_iterations = max_iterations
5          self.initial_temp = initial_temp
6          self.cooling_rate = cooling_rate
7          self.objective = ScheduleObjective(registry)
8          self.random_func = random_func if random_func is not None else random.random
9
10     def run(self) -> tuple[Schedule, Schedule, float, list, list, int, float]:
11         start_time = time.time()
12         initial_schedule = Schedule.random_initial_assignment(self.registry)
13         current = initial_schedule
14         current_score = self.objective.evaluate(current)
15         best = copy.deepcopy(current)
16         best_score = current_score
17         temp = self.initial_temp
18
19         history = [current_score]
20         acceptance_history = []
21         stuck_count = 0
22         iterations_without_improvement = 0
23         iteration = 0
24
25         while self.max_iterations is None or iteration < self.max_iterations:
26             iteration += 1
27
28             neighbor = generate_random_neighbor(current, self.registry)
29             neighbor_score = self.objective.evaluate(neighbor)
30             delta = neighbor_score - current_score
31
32             if delta < 0:
33                 acceptance_prob = 1.0
34             else:
35                 acceptance_prob = math.exp(-delta / (temp + 1e-8))
36
37             acceptance_history.append(acceptance_prob)
38
39             if delta < 0 or self.random_func() < acceptance_prob:
40                 current = neighbor
41                 current_score = neighbor_score
42
43                 if current_score < best_score:
44                     best = copy.deepcopy(current)
45                     best_score = current_score
46                     iterations_without_improvement = 0
47                 else:
48                     iterations_without_improvement += 1
49             else:
50                 iterations_without_improvement += 1
51
52             history.append(current_score)
53
54             if iterations_without_improvement >= 50:
55                 stuck_count += 1
56                 iterations_without_improvement = 0
57
58             temp *= self.cooling_rate
59
60             if best_score == 0:
61                 break
62
63         end_time = time.time()
64         duration = end_time - start_time
65
66         return initial_schedule, best, best_score, history, acceptance_history, stuck_count, duration
```

2.2.5.2 Deskripsi Kelas/Fungsi Simulated Annealing

Kelas SimulatedAnnealing mengimplementasikan algoritma *Simulated Annealing* untuk mengoptimalkan jadwal kuliah dengan **meminimalkan jumlah konflik waktu antar mahasiswa**. Berbeda dari varian hill-climbing, algoritma ini **kadang menerima solusi yang lebih buruk** secara terkontrol (berdasarkan suhu/temperature) agar tidak mudah terjebak pada local optimum.

Secara garis besar, logika kerja fungsi run() adalah sebagai berikut:

1. Inisialisasi parameter dan jadwal awal

Algoritma menyiapkan parameter initial_temp (suhu awal), cooling_rate (laju pendinginan), dan fungsi acak random_func (default

random.random). Dibangkitkan jadwal awal acak dengan Schedule.random_initial_assignment(), lalu dihitung nilai objektifnya sebagai current_score. Nilai terbaik sementara (best, best_score) diset dari keadaan awal. Riwayat skor (history) dan riwayat probabilitas penerimaan (acceptance_history) juga dimulai dari nilai awal.

2. Iterasi pencarian dan pembangkitan tetangga

Pada setiap iterasi, algoritma membuat **tetangga acak** dari solusi saat ini melalui generate_random_neighbor(current, registry). Skor tetangga dihitung dengan fungsi objektif, lalu selisih kualitas didefinisikan sebagai delta = neighbor_score - current_score (semakin kecil semakin baik karena tujuan adalah minimisasi).

3. Keputusan penerimaan solusi

- Jika delta < 0, tetangga **lebih baik** dan **selalu diterima**.
- Jika delta ≥ 0 , tetangga **lebih buruk** dan **dapat diterima** dengan peluang

$$\text{acceptance_prob} = \exp(-\delta / (\text{temp} + 1e-8)).$$

Nilai peluang ini dicatat ke acceptance_history, lalu dibandingkan dengan angka acak dari random_func(). Jika angka acak lebih kecil dari acceptance_prob, tetangga diterima sebagai solusi baru.

4. Pembaruan solusi terbaik dan pencatatan riwayat

Setelah keputusan penerimaan, current dan current_score diperbarui sesuai hasil. Jika skor saat ini lebih baik daripada best_score, maka best dan best_score diperbarui, serta penghitung iterations_without_improvement direset ke nol. Jika tidak membaik, penghitung tersebut dinaikkan satu. Nilai current_score pada tiap iterasi disimpan ke history.

5. Deteksi kebuntuan dan metrik diagnostik

Jika tidak ada perbaikan selama 50 iterasi berturut-turut, penghitung stuck_count dinaikkan satu dan iterations_without_improvement direset. Metrik ini membantu menganalisis seberapa sering algoritma tersangkut pada area sulit.

6. Pendinginan suhu

Di akhir setiap iterasi, suhu diturunkan secara geometrik: temp = temp * cooling_rate. Dengan suhu yang makin rendah, peluang menerima solusi buruk makin kecil sehingga pencarian beralih dari eksplorasi ke eksloitasi.

7. Kondisi terminasi

Iterasi berhenti ketika mencapai max_iterations (jika diset) atau ketika best_score == 0 yang menandakan **jadwal tanpa konflik** telah ditemukan.

Terminasi dan pengembalian hasil

Setelah selesai, fungsi run() mengembalikan:

- Jadwal awal yang dipakai(initial_schedule),
- Jadwal terbaik yang ditemukan(best),
- Nilai objektif terbaik(best_score),
- Jejak skor pada tiap iterasi(history),
- Jejak peluang penerimaan solusi buruk(acceptance_history),
- Jumlah kali terdeteksi kebuntuan(stuck_count), dan
- Waktu eksekusi total (duration)

2.2.6 Genetic Algorithm

Genetic Algorithm (GA) adalah metode pencarian berbasis populasi yang meniru proses evolusi biologis. Di konteks penjadwalan kelas mingguan, GA menjaga sekumpulan kandidat jadwal lalu melakukan seleksi orang tua, crossover, dan mutasi untuk menghasilkan jadwal baru yang semakin minim konflik menurut fungsi objektif.

Pada implementasi ini, setiap individu pada populasi direpresentasikan sebagai Schedule. Seluruh operator GA beroperasi langsung pada objek Schedule dengan memanfaatkan API seperti place, remove, move, swap, is_empty, dan get_position. Evaluasi kualitas dilakukan melalui ScheduleObjective.evaluate(schedule).

2.2.6.1 Source Code Genetic Algorithm



```
 1  from core.registry import Registry
 2  from core.schedule import Schedule
 3  from core.objective import ScheduleObjective
 4  from typing import Optional
 5  from typing import Tuple, List, Optional
 6  import random
 7  import time
 8
 9
10 class Genetic_Algorithm:
11     def __init__(self, registry: Registry, population_size: int, max_iteration: int):
12         self.registry = registry
13         self.population_size = population_size
14         self.max_iteration = max_iteration
15         self.population = []
16         self.parents = []
17         self.objective = ScheduleObjective(self.registry)
18
19     # Step 1: Initialize Population
20     def init_population(self):
21         """
22             Generate initial population of total = population_size random schedules
23         """
24         self.population = [] # Reset population
25         for i in range (self.population_size):
26             schedule = Schedule.random_initial_assignment(self.registry)
27             self.population.append(schedule)
28         # print("Successfully initialized population")
29         schedule.display(self.registry)
30         return self.population
31
```

```
 1 # Step 2: Choose Parents
 2 # Strategy -> use probability function. Evaluate each individual with the fitness function/ObjFunc
 3 # Choose a total of [total_population] parents.
 4
 5     def tournament_selection(self, tournament_size: Optional[int] = None):
 6         # Determine tournament size
 7         if not (tournament_size):
 8             if self.population_size < 5:
 9                 tournament_size = 1
10             elif self.population_size < 10:
11                 tournament_size = 2
12             elif self.population_size < 20:
13                 tournament_size = 3
14             else:
15                 tournament_size = 5
16
17         population_scored = {}
18         for schedule in self.population:
19             score = self.objective.evaluate(schedule)
20             population_scored[schedule] = score # Map candidates with score
21
22         parents = []
23         # Select [num of parents] population
24         for i in range(self.population_size):
25             candidates = random.sample(self.population, tournament_size)
26
27             best_candidate = None
28             best_score = float('inf')
29
30             for candidate in candidates:
31                 score = population_scored[candidate]
32                 if score < best_score:
33                     best_candidate = candidate
34                     best_score = score
35
36             parents.append(best_candidate)
37
38         # Clear old parents
39         self.parents = parents
40
41         return self.parents
42
```

```
 1 # Step 3: Crossover (Recombination)
 2 # Strategy -> Use one-point crossover. Explore other options
 3 # One-point crossover func, split by sorted index (day + timeslots).
 4 # crossover(total_iteration)
 5
 6     def _find_and_place(self, schedule: Schedule, meeting_id: int, preferred_room: str) -> bool:
 7         """Helper to find any free spot for a meeting and place it."""
 8         legal_rooms = self.registry.legal_classrooms_by_meeting.get(meeting_id, [preferred_room])
 9         if not legal_rooms:
10             return False
11
12         random.shuffle(schedule.days)
13         random.shuffle(schedule.hours)
14         random.shuffle(legal_rooms)
15
16         for day in schedule.days:
17             for hour in schedule.hours:
18                 for room in legal_rooms:
19                     if schedule.is_empty(day, hour, room):
20                         schedule.place(meeting_id, day, hour, room)
21                         return True
22
23         return False # No free spot found
24
25     def one_point_crossover(self, parent1: Schedule, parent2: Schedule):
26         # 1) Collect ALL meeting IDs from both parents
27         all_meeting_ids = set(parent1.where_is.keys()) | set(parent2.where_is.keys())
28
29         # Group by course code
30         meetings_by_course = {}
31         for meeting_id in all_meeting_ids:
32             meeting = self.registry.meetings[meeting_id]
33             course_code = meeting.course_code
34             if course_code not in meetings_by_course:
35                 meetings_by_course[course_code] = []
36             meetings_by_course[course_code].append(meeting_id)
37
38         # 2) Sort courses for a consistent crossover point
39         sorted_courses = sorted(meetings_by_course.keys())
40
41         if len(sorted_courses) <= 1:
42             import copy
43             return copy.deepcopy(parent1), copy.deepcopy(parent2)
44
45         # 3) Select crossover point
46         crossover_point = random.randint(1, len(sorted_courses) - 1)
47
48         # 4) Create children as deep copies of parents
49         import copy
50         child1 = copy.deepcopy(parent1)
51         child2 = copy.deepcopy(parent2)
```

```

51
52     # 5) Crossover by swapping courses after the crossover point
53     for i, course_code in enumerate(sorted_courses):
54         if i >= crossover_point:
55             meeting_ids_to_swap = meetings_by_course[course_code]
56
57             # Step A: Clear all meetings for this course from both children
58             # This creates a clean slate to prevent duplicates or lost meetings.
59             for mid in meeting_ids_to_swap:
60                 pos1 = child1.get_position(mid)
61                 if pos1:
62                     child1.remove(*pos1)
63                 pos2 = child2.get_position(mid)
64                 if pos2:
65                     child2.remove(*pos2)
66
67             # Step B: Re-populate from the opposite parent's genes
68             for mid in meeting_ids_to_swap:
69                 # Give child1 the genes from parent2
70                 p2_pos = parent2.get_position(mid)
71                 if p2_pos:
72                     # Try to place at the exact same position
73                     was_placed = child1.place(mid, *p2_pos)
74                     if not was_placed:
75                         # If that spot is taken, find any other free spot
76                         self._find_and_place(child1, mid, p2_pos[2])
77
78                 # Give child2 the genes from parent1
79                 p1_pos = parent1.get_position(mid)
80                 if p1_pos:
81                     # Try to place at the exact same position
82                     was_placed = child2.place(mid, *p1_pos)
83                     if not was_placed:
84                         # If that spot is taken, find any other free spot
85                         self._find_and_place(child2, mid, p1_pos[2])
86
87             return child1, child2
88
89     def crossover_population(self):
90         offspring = []
91
92         for i in range(0, len(self.parents) - 1, 2):
93             parent1 = self.parents[i]
94             parent2 = self.parents[i+1]
95
96             child1, child2 = self.one_point_crossover(parent1, parent2)
97
98             # Validate
99             try:
100                 self._validate_schedule_credits(child1)
101                 self._validate_schedule_credits(child2)
102                 offspring.extend([child1, child2])
103             except ValueError as e:
104                 print(f"Unexpected validation failure: {e}")
105                 # Fallback to parents
106                 import copy
107                 offspring.extend([copy.deepcopy(parent1), copy.deepcopy(parent2)])
108
109         if len(self.parents) % 2 == 1:
110             import copy
111             offspring.append(copy.deepcopy(self.parents[-1]))
112
113         # print(f"Generated {len(offspring)} offspring")
114         return offspring
115

```

```
28     def _validate_schedule_credits(self, schedule: Schedule):
29         meetings_per_course = {}
30         for meeting_id in schedule.where_is.keys():
31             meeting = self.registry.meetings[meeting_id]
32             course_code = meeting.course_code
33             meetings_per_course[course_code] = meetings_per_course.get(course_code, 0) + 1
34
35         # Check against expected credits
36         for course_code, count in meetings_per_course.items():
37             expected_credits = self.registry.courses[course_code].credits
38             if count != expected_credits:
39                 print(f"Schedule: Course {course_code} has {count} meetings, expected {expected_credits}")
40                 raise ValueError(f"Credit validation failed for {course_code}")
41
```

```
● ● ●

1 # Step 4: Mutation
2 # Change a random room in legal_classrooms_by_meeting
3 # Use a randomizer func to change which meetings get's moved to a different slot.
4     def mutate_schedule(self, schedule: Schedule, mutation_rate: float = 0.1):
5         # 1) Decide mutation action randomly
6         mutation_type = random.choice(['swap', 'move', 'time_shift'])
7
8         if random.random() > mutation_rate:
9             return schedule # No mutation
10
11        if mutation_type == 'swap':
12            # Swap two random meetings
13            meeting_ids = list(schedule.where_is.keys())
14            if len(meeting_ids) >= 2:
15                mid1, mid2 = random.sample(meeting_ids, 2)
16                pos1 = schedule.get_position(mid1)
17                pos2 = schedule.get_position(mid2)
18
19                if pos1 and pos2:
20                    schedule.swap(pos1, pos2)
21
22        elif mutation_type == 'move':
23            # Move one meeting to a free position
24            meeting_ids = list(schedule.where_is.keys())
25            if meeting_ids:
26                mid = random.choice(meeting_ids)
27                old_pos = schedule.get_position(mid)
28
29                # Get legal classrooms for this meeting
30                meeting = self.registry.meetings[mid]
31                legal_rooms = self.registry.legal_classrooms_by_meeting.get(mid, [])
32
33                if legal_rooms and old_pos:
34                    # Try random new position
35                    new_day = random.choice(schedule.days)
36                    new_hour = random.choice(schedule.hours)
37                    new_room = random.choice(legal_rooms)
38                    new_pos = (new_day, new_hour, new_room)
39
40                    # Only move if new position is free
41                    if schedule.is_empty(new_day, new_hour, new_room):
42                        schedule.move(old_pos, new_pos)
43
44        elif mutation_type == 'time_shift':
45            # Shift one meeting to adjacent time slot
46            meeting_ids = list(schedule.where_is.keys())
47            if meeting_ids:
48                mid = random.choice(meeting_ids)
49                old_pos = schedule.get_position(mid)
50
51                if old_pos:
52                    day, hour, room = old_pos
53                    # Try shift +1 or -1 hour
54                    new_hour = hour + random.choice([-1, 1])
55                    if new_hour in schedule.hours:
56                        new_pos = (day, new_hour, room)
57                        if schedule.is_empty(day, new_hour, room):
58                            schedule.move(old_pos, new_pos)
59
60    return schedule
```

```
60
61     def mutate_population(self, offspring: list[Schedule], mutation_rate: float = 0.1):
62         mutated = []
63         mutation_count = 0
64
65         for schedule in offspring:
66             original_fitness = self.objective.evaluate(schedule)
67             mutated_schedule = self.mutate_schedule(schedule, mutation_rate)
68             new_fitness = self.objective.evaluate(mutated_schedule)
69
70             # Accept mutation if it improves or with small probability if worse
71             if new_fitness <= original_fitness or random.random() < 0.1:
72                 mutated.append(mutated_schedule)
73                 if new_fitness < original_fitness:
74                     mutation_count += 1
75             else:
76                 mutated.append(schedule) # Keep original
77
78         # print(f"Mutation: {mutation_count}/{len(offspring)} improved")
79
80         return mutated
```



```
1 # Step 5: Evaluation
2 def get_best_schedule(self, population: list[Schedule]) -> tuple[Schedule, float]:
3     best_schedule = None
4     best_fitness = float('inf')
5
6     for schedule in population:
7         fitness = self.objective.evaluate(schedule)
8         if fitness < best_fitness:
9             best_fitness = fitness
10            best_schedule = schedule
11
12    return best_schedule, best_fitness
13
```

```
1 Step 6: Main GA Loop
2     def run(self, mutation_rate: float = 0.1) -> Tuple[Optional[Schedule], Optional[Schedule], float, List[float], int, float]:
3         """
4             Executes the genetic algorithm and returns detailed diagnostics.
5
6             Args:
7                 mutation_rate: The probability of a mutation for each schedule.
8
9             Returns:
10                A tuple containing:
11                    - initial_best_schedule: The best schedule from the first generation.
12                    - global_best_schedule: The best schedule found across all generations.
13                    - global_best_score: The fitness score of the best schedule.
14                    - score_history: A list of the best score at each generation.
15                    - generations_run: The total number of generations executed.
16                    - duration: The total execution time in seconds.
17            """
18         start_time = time.time()
19
20         # Step 1: Initialize Population
21         self.init_population()
22         if not self.population:
23             return None, None, float('inf'), [], 0, time.time() - start_time
24
25         initial_best_schedule, initial_best_fitness = self.get_best_schedule(self.population)
26
27         best_ever_schedule = initial_best_schedule
28         best_ever_fitness = initial_best_fitness
29         score_history = [initial_best_fitness]
30         generations_run = 0
31
32         # Step 2: Main Evolution Loop
33         for generation in range(self.max_iteration):
34             generations_run += 1
35
36             self.parents = self.tournament_selection()
37             offspring = self.crossover_population()
38             offspring = self.mutate_population(offspring, mutation_rate)
39             self.population = offspring
40
41             current_best_schedule, current_best_fitness = self.get_best_schedule(self.population)
42
43             if current_best_fitness < best_ever_fitness:
44                 best_ever_fitness = current_best_fitness
45                 best_ever_schedule = current_best_schedule
46
47             score_history.append(best_ever_fitness)
48
49             if best_ever_fitness == 0:
50                 break
51
52         # Step 3: Finalize and Return Results
53         end_time = time.time()
54         duration = end_time - start_time
55
56         return (
57             initial_best_schedule,
58             best_ever_schedule,
59             best_ever_fitness,
60             score_history,
61             generations_run,
62             duration
63         )
```

2.2.6.2 Deskripsi Fungsi Genetic Algorithm

Kelas Genetic_Algorithm mengimplementasikan GA untuk mengoptimasi jadwal mingguan. Setiap generasi terdiri dari tiga tahap utama: seleksi orang tua dengan turnamen, crossover satu titik berbasis kelompok mata kuliah, dan mutasi ringan yang tetap menghormati ruang legal penempatan. Evaluasi kualitas solusi menggunakan ScheduleObjective.

Output run() mencakup jadwal terbaik generasi awal, jadwal terbaik global, nilai objektif terbaik, riwayat skor terbaik lintas generasi, jumlah generasi berjalan, serta durasi eksekusi. Ini selaras dengan kebutuhan eksperimen GA yang meminta pencatatan nilai objektif per iterasi dan durasi.

1. Inisialisasi Populasi: init_population()
 - Membentuk populasi awal berukuran population_size menggunakan Schedule.random_initial_assignment(registry).
 - Setiap Schedule sudah konsisten dengan domain data pada Registry dan memenuhi format state jadwal.
 - Mengembalikan daftar individu yang siap dievaluasi.
2. Seleksi Orang Tua: tournament_selection(tournament_size)
 - Menghitung skor fitness semua anggota populasi: population_scored[schedule] = objective.evaluate(schedule).
 - Melakukan seleksi turnamen. Ukuran turnamen ditentukan adaptif terhadap population_size agar tetap kompetitif walau populasi kecil:
 - Populasi sangat kecil memilih tournament_size minimal
 - Populasi besar menggunakan turnamen sampai lima kandidat
 - Untuk setiap parent yang dibutuhkan, sampling kandidat acak lalu memilih yang berskor paling kecil.
 - Menghasilkan array parents sepanjang ukuran populasi sehingga tekanan seleksi cukup tinggi namun tetap beragam.
3. Crossover: one_point_crossover(parent1, parent2)
 - Unit genetik dikelompokkan per mata kuliah. Semua meeting_id dari kedua parent dihimpun, lalu dikelompokkan berdasarkan course_code. Ini memastikan integritas paket perkuliahan ketika ditukar.
 - Titik crossover satu titik dipilih pada urutan course_code yang sudah diurutkan stabil. Semua pertemuan yang indeks mata kuliahnya di sisi kanan titik ini dipertukarkan antar anak.
 - Strategi aman dari kehilangan kredit:
 - 1) Sebelum menaruh gen dari parent lawan, anak menghapus semua meeting milik course_code itu agar tidak terjadi duplikasi.

- 2) Saat menaruh, algoritma mencoba posisi yang sama seperti di parent sumber. Jika bentrok, `_find_and_place` mencari posisi legal lain yang kosong dengan mematuhi daftar ruang legal `legal_classrooms_by_meeting`.
- Hasilnya adalah dua child yang mewarisi blok-blok jadwal per mata kuliah dari parent berbeda, dengan resolusi konflik ruang dan waktu yang aman.
4. Validasi Kredit: `_validate_schedule_credits(schedule)`
- Menghitung jumlah meeting per `course_code` di `schedule`.
 - Memastikan jumlahnya sama dengan `credits` yang ditentukan pada `registry.courses[course_code].credits`.
 - Jika mismatch, melempar `ValueError`. Pada pipeline `crossover_population`, kegagalan validasi memicu fallback aman: anak dibuang dan diganti deep copy parent, sehingga tidak ada “kredit hilang” pasca-crossover.
5. Mutasi Individu: `mutate_schedule(schedule, mutation_rate)`
- Bila `random.random()` melebihi `mutation_rate`, mutasi dilewati demi efisiensi.
 - Tiga operator mutasi ringan yang menjaga legalitas:
 - `swap`: menukar dua meeting acak dengan `schedule.swap(pos1, pos2)`.
 - `move`: memindahkan satu meeting ke slot kosong lain, dengan ruang dipilih dari `legal_classrooms_by_meeting[mid]`.
 - `time_shift`: menggeser satu meeting ke jam tetangga di hari yang sama bila kosong.
 - Mutasi ini konsisten dengan langkah neighbor yang diperbolehkan: tukar dua pertemuan atau memindahkan pertemuan ke slot kosong. Operator waktu memastikan eksplorasi lokal yang halus.
6. Mutasi Populasi: `mutate_population(offspring, mutation_rate)`
- Mengevaluasi fitness sebelum dan sesudah mutasi.
 - Menerima mutasi jika membaik atau, bila memburuk, tetap bisa diterima dengan probabilitas kecil 0.1. Heuristik ini menambah variasi dan membantu lepas dari plateau tanpa mengorbankan konvergensi.
 - Mengembalikan populasi baru yang sudah melalui seleksi mutasi.
7. Evaluasi Terbaik: `get_best_schedule(population)`
- Melintasi populasi dan mencari jadwal dengan skor objektif terkecil.
 - Mengembalikan pasangan (`best_schedule, best_fitness`) untuk dipakai pada pelacakan performa per generasi.

8. Main Loop dan Terminasi: run(mutation_rate)
 - 1) Inisialisasi populasi lalu catat best awal sebagai baseline.
 - 2) Untuk setiap generasi sampai max_iteration:
 - Seleksi parent via turnamen
 - Crossover berpasangan menjadi offspring ditambah validasi kredit
 - Mutasi populasi anak dengan mutation_rate
 - Evaluasi best generasi dan update best global bila membaik
 - Catat score_history sebagai jejak konvergensi
 - Terminasi dini jika best_ever_fitness == 0 yang menandakan solusi bebas konflik menurut fungsi objektif.
 - 3) Kembalikan:
 - initial_best_schedule
 - best_ever_schedule
 - best_ever_fitness
 - score_history untuk plotting nilai objektif vs generasi
 - generations_run
 - duration eksekusi untuk pelaporan

2.3 Hasil Eksperimen

Semua percobaan menggunakan dataset yang sama (large_test.json).

2.3.1 Hasil Steepest Ascent Hill-Climbing

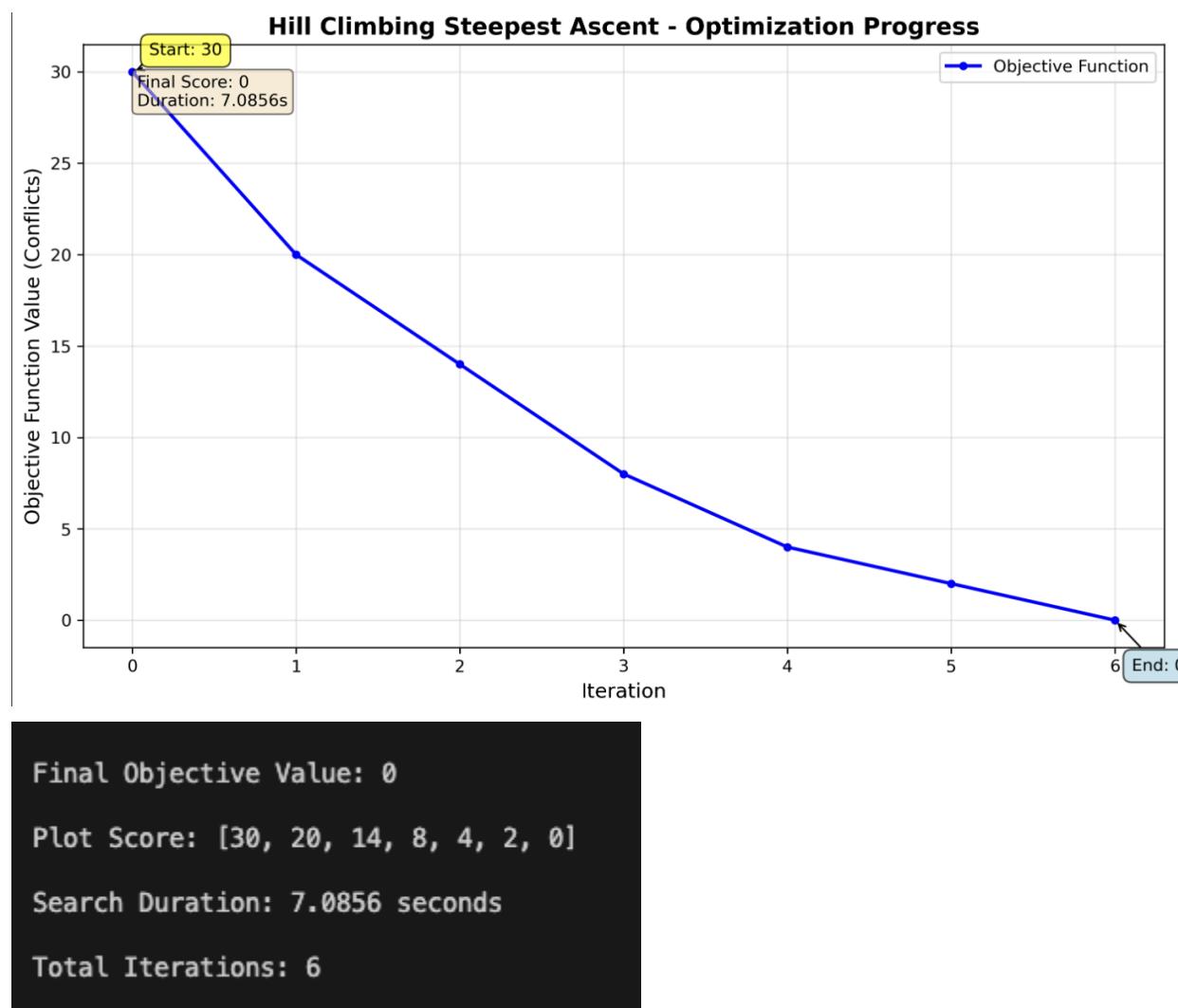
2.3.1.1 Percobaan 1

Hill Climbing Steepest Ascent - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7606:IF2140_K01	7605:IF2111_K01	-	-	-
8:00	7607:IF2190_K01	-	-	7602:IF2160_K01	-
9:00	7608:IF2150_K02	-	7601:IF2120_K01 7605:IF2160_K01	7606:IF2120_K01	7601:IF2120_K01
10:00	7602:IF2110_K01	7602:IF2210_K01 7604:IF2120_K02	7605:IF2150_K02 7607:IF2200_K01 7608:IF2170_K01	7606:IF2110_K01	-
11:00	-	-	-	-	7601:IF2190_K01
12:00	7607:IF2140_K01	-	7606:IF2170_K01	-	7602:IF2180_K01
13:00	-	7605:IF2110_K02	7601:IF2110_K02 7602:IF2180_K01 7604:IF2160_K01	-	7607:IF2310_K02 7608:IF2130_K01
14:00	-	7602:IF2110_K02	7603:IF2190_K01	7608:IF2120_K02	-
15:00	-	-	7606:IF2120_K02	7603:IF2210_K01 7607:IF2110_K01	-
16:00	7604:IF2150_K01 7607:IF2130_K01 7608:IF2111_K01	7602:IF2140_K01	-	7605:IF2180_K01	-
17:00	7602:IF2200_K01 7608:IF2110_K01	7603:IF2140_K01	-	7603:IF2230_K01 7604:IF2200_K01 7607:IF2180_K01	7603:IF2150_K01

Hill Climbing Steepest Ascent - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7606:IF2140_K01	7605:IF2111_K01	-	-	-
8:00	7603:IF2200_K01 7607:IF2190_K01	-	-	7602:IF2160_K01	-
9:00	7602:IF2111_K01 7604:IF2110_K01 7608:IF2150_K02	-	7601:IF2120_K01 7605:IF2160_K01	7606:IF2120_K01	7601:IF2120_K01
10:00	7602:IF2110_K01 7604:IF2110_K02	7602:IF2210_K01	7605:IF2150_K02 7608:IF2170_K01	7606:IF2110_K01	-
11:00	7602:IF2130_K01	-	-	-	7601:IF2190_K01
12:00	7607:IF2140_K01	-	7606:IF2170_K01	-	7602:IF2180_K01
13:00	7603:IF2120_K02	7605:IF2110_K02	7602:IF2180_K01 7604:IF2160_K01	-	7607:IF2110_K02 7608:IF2130_K01
14:00	-	7602:IF2110_K02	7603:IF2190_K01	7608:IF2120_K02	-
15:00	-	-	7606:IF2120_K02	7603:IF2210_K01	-
16:00	7604:IF2150_K01 7607:IF2130_K01	7602:IF2140_K01	-	7605:IF2180_K01	-
17:00	7602:IF2200_K01 7608:IF2110_K01	7603:IF2140_K01	-	7604:IF2200_K01 7607:IF2180_K01	7603:IF2150_K01



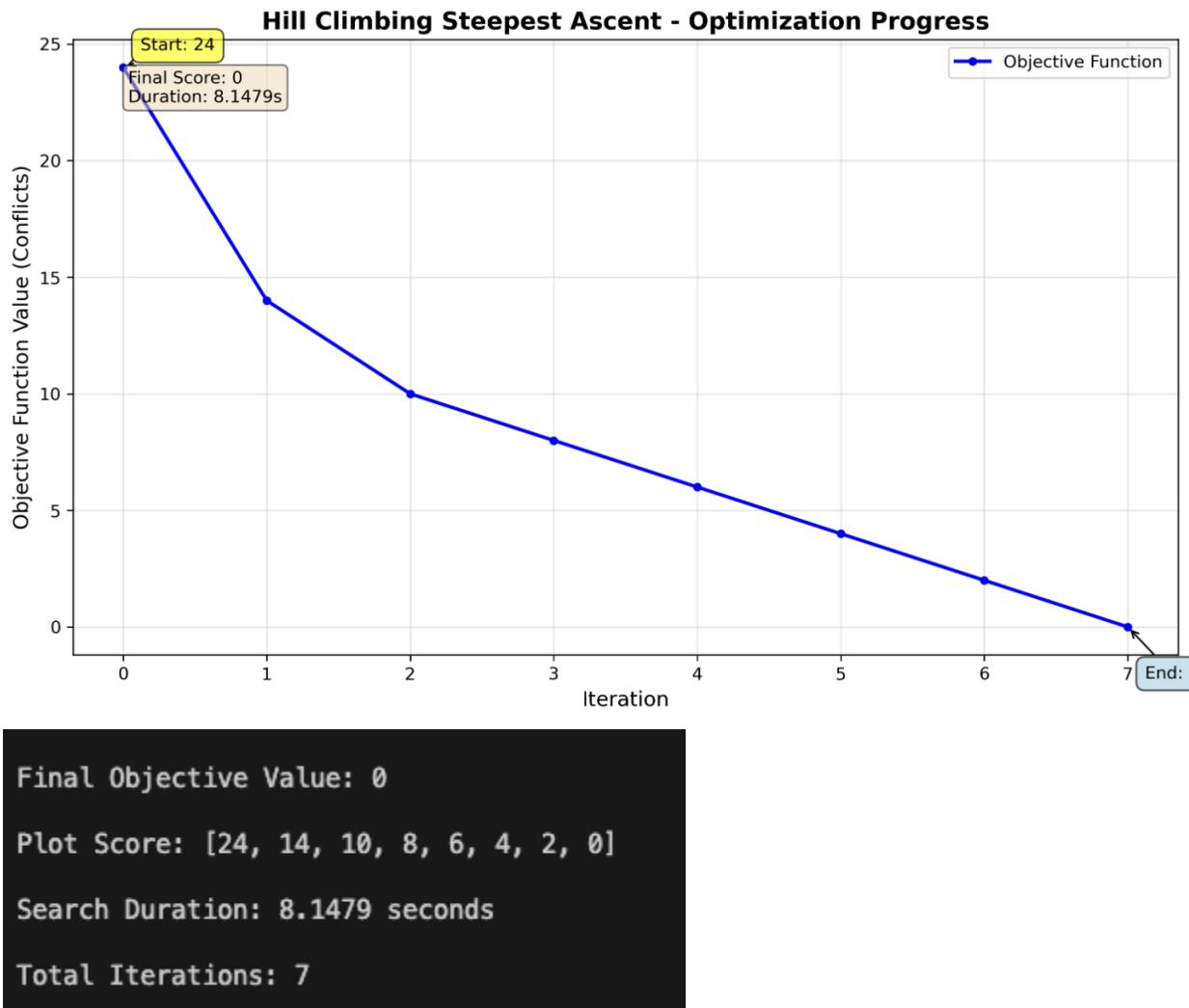
2.3.1.2 Percobaan 2

Hill Climbing Steepest Ascent - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7608:IF2110_K02	-	7601:IF2190_K01 7603:IF2200_K01	7602:IF2190_K01	-
8:00	-	-	7601:IF2140_K01 7605:IF2110_K01	7605:IF2210_K01	7602:IF2130_K01
9:00	-	7604:IF2120_K01 7607:IF2110_K02	-	-	-
10:00	7604:IF2200_K01	-	7604:IF2120_K02	7605:IF2110_K01	-
11:00	7607:IF2130_K01 7608:IF2170_K01	7605:IF2110_K01	7604:IF2111_K01	7603:IF2150_K01	-
12:00	7601:IF2110_K02 7602:IF2140_K01	-	7603:IF2170_K01 7605:IF2210_K01	-	7605:IF2200_K01 7607:IF2150_K02
13:00	7606:IF2140_K01	-	7606:IF2111_K01	-	-
14:00	7606:IF2130_K01	-	7606:IF2160_K01	7607:IF2180_K01	7604:IF2160_K01
15:00	-	-	7603:IF2180_K01 7604:IF2180_K01	7605:IF2120_K01 7608:IF2120_K02	7602:IF2110_K02
16:00	7608:IF2120_K01	7606:IF2140_K01	-	7602:IF2180_K01 7603:IF2150_K02 7604:IF2150_K01	7601:IF2190_K01
17:00	-	-	-	7602:IF2160_K01 7607:IF2110_K01	7601:IF2120_K02

Hill Climbing Steepest Ascent - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7606:IF2180_K01 7608:IF2110_K02	-	7601:IF2190_K01 7603:IF2200_K01	7602:IF2190_K01	-
8:00	7604:IF2110_K01 7605:IF2110_K02	-	7601:IF2140_K01	7605:IF2210_K01	7602:IF2130_K01
9:00	7601:IF2150_K02 7604:IF2110_K01	7604:IF2120_K01 7607:IF2110_K02	-	-	-
10:00	7604:IF2200_K01	-	7604:IF2120_K02	7605:IF2110_K01	-
11:00	7608:IF2170_K01	7605:IF2110_K01	7604:IF2111_K01	7603:IF2150_K01	-
12:00	7601:IF2150_K01 7602:IF2140_K01	-	7603:IF2170_K01 7605:IF2210_K01	-	7605:IF2200_K01
13:00	7606:IF2140_K01	-	7606:IF2111_K01	-	-
14:00	7606:IF2130_K01	-	7606:IF2160_K01	7607:IF2180_K01	7604:IF2160_K01
15:00	7602:IF2130_K01	-	7603:IF2180_K01	7605:IF2120_K01 7608:IF2120_K02	7602:IF2110_K02
16:00	7608:IF2120_K01	7606:IF2140_K01	-	7602:IF2180_K01 7603:IF2150_K02	7601:IF2190_K01
17:00	-	-	-	7602:IF2160_K01	7601:IF2120_K02



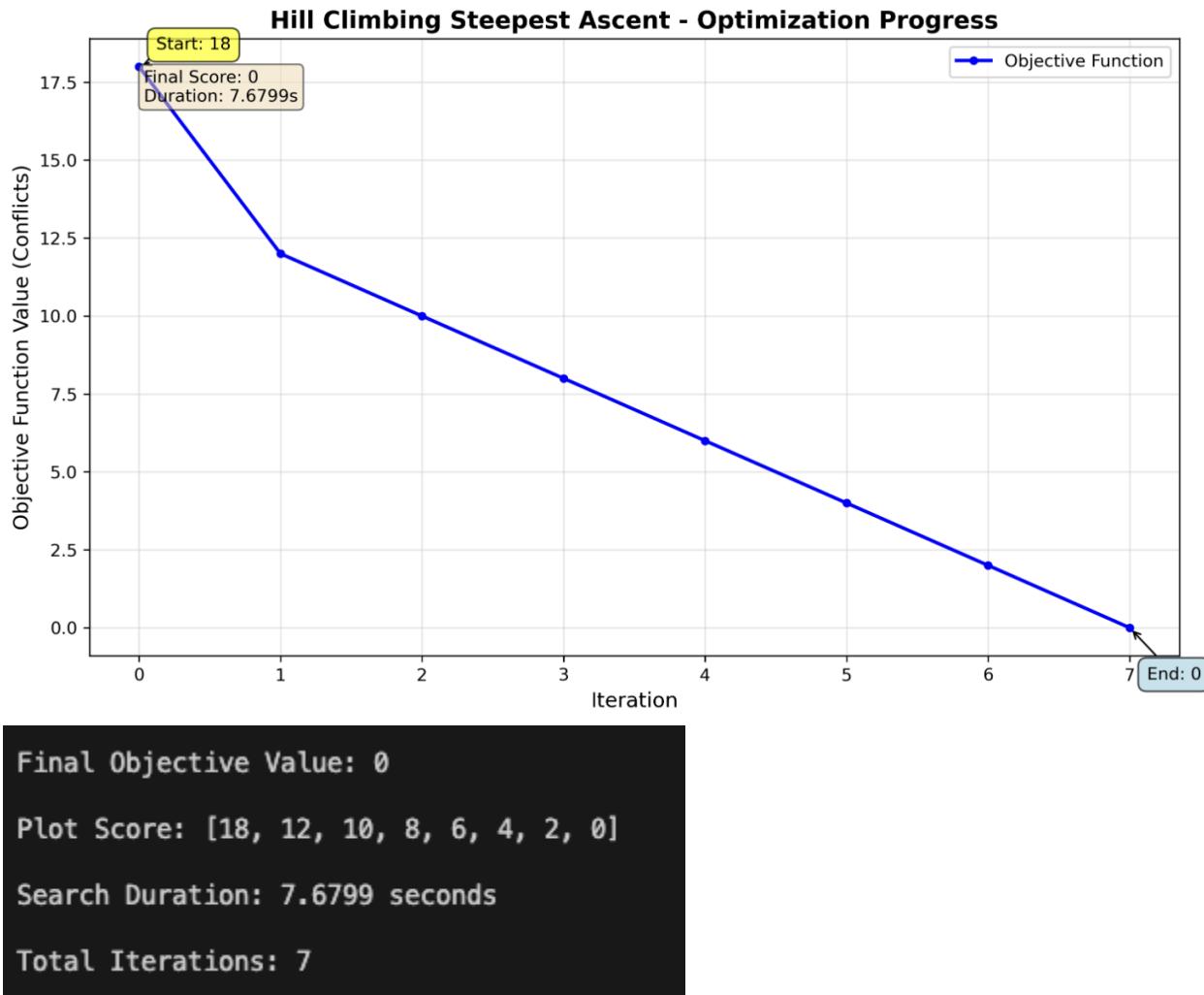
2.3.1.3 Percobaan 3

Hill Climbing Steepest Ascent - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7604:IF2120_K01 7606:IF2110_K02 7608:IF2140_K01	-	-	7605:IF2110_K01	7604:IF2140_K01 7608:IF2130_K01
8:00	7608:IF2130_K01	7604:IF2200_K01	-	7605:IF2110_K01	7605:IF2160_K01
9:00	-	-	7608:IF2170_K01	-	-
10:00	-	7603:IF2180_K01	-	7605:IF2150_K01	7601:IF2130_K01 7605:IF2190_K01
11:00	7605:IF2110_K01	7605:IF2200_K01	-	7603:IF2150_K02 7606:IF2140_K01	-
12:00	-	7604:IF2190_K01	-	7604:IF2180_K01	7602:IF2110_K01
13:00	-	-	-	7603:IF2120_K01	-
14:00	-	-	7601:IF2111_K01 7602:IF2210_K01	7601:IF2111_K01	7607:IF2150_K01
15:00	7607:IF2160_K01	-	7604:IF2110_K02	7606:IF2120_K02 7608:IF2190_K01	-
16:00	7603:IF2110_K02	7604:IF2210_K01 7607:IF2120_K02	7603:IF2160_K01 7605:IF2180_K01	7605:IF2180_K01	-
17:00	7603:IF2140_K01	7603:IF2120_K02 7604:IF2150_K02	7601:IF2120_K01	7608:IF2170_K01	7603:IF2110_K02 7608:IF2200_K01

Hill Climbing Steepest Ascent - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7604:IF2120_K01 7606:IF2110_K02	-	-	7605:IF2110_K01	7604:IF2140_K01
8:00	7604:IF2110_K02 7608:IF2130_K01	7604:IF2200_K01	-	7605:IF2110_K01	7605:IF2160_K01
9:00	7605:IF2140_K01	-	7608:IF2170_K01	-	-
10:00	7602:IF2111_K01 7603:IF2120_K02	7603:IF2180_K01	-	7605:IF2150_K01	7605:IF2190_K01
11:00	7603:IF2120_K02 7605:IF2110_K01	7605:IF2200_K01	-	7603:IF2150_K02 7606:IF2140_K01	-
12:00	7602:IF2130_K01	7604:IF2190_K01	-	7604:IF2180_K01	7602:IF2110_K01
13:00	7602:IF2130_K01	-	-	7603:IF2120_K01	-
14:00	-	-	7602:IF2210_K01	7601:IF2111_K01	7607:IF2150_K01
15:00	7607:IF2160_K01	-	7604:IF2110_K02	7606:IF2120_K02 7608:IF2190_K01	-
16:00	7603:IF2110_K02	7604:IF2210_K01	7603:IF2160_K01 7605:IF2180_K01	7605:IF2180_K01	-
17:00	7603:IF2140_K01	7604:IF2150_K02	7601:IF2120_K01	7608:IF2170_K01	7608:IF2200_K01



2.3.2 Hasil Stochastic Hill-climbing

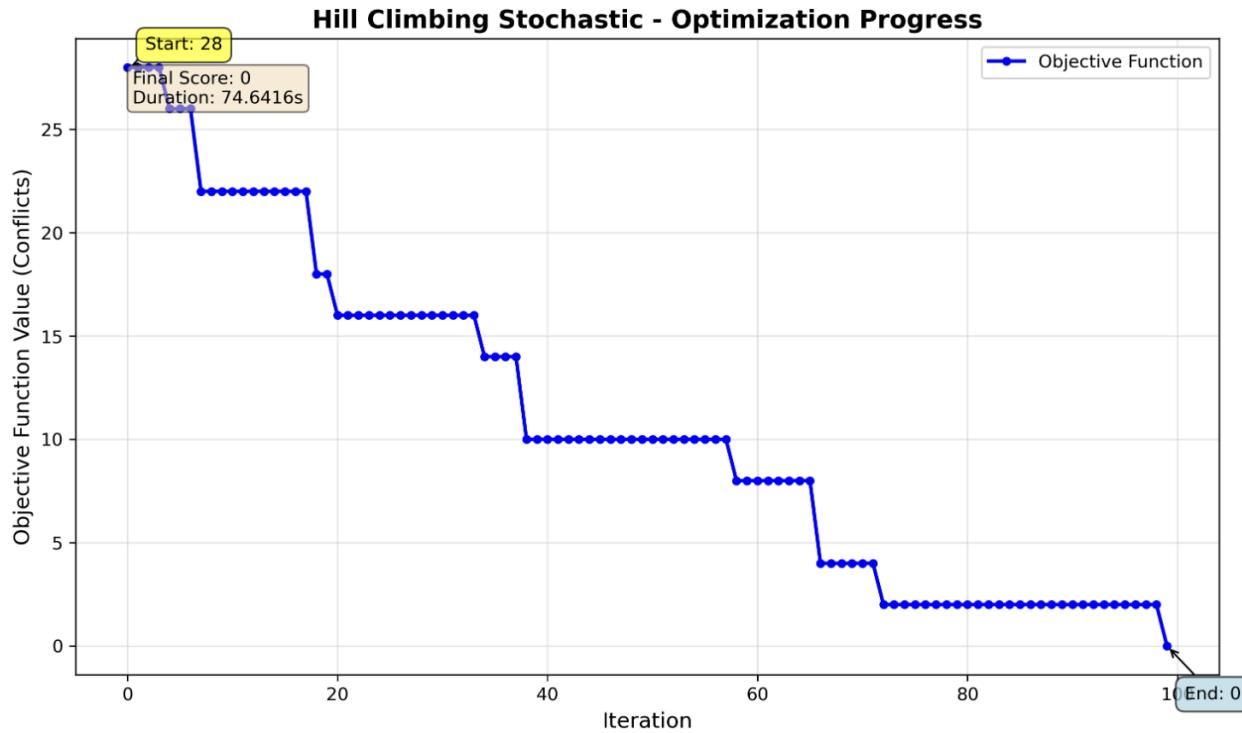
2.3.2.1 Percobaan 1

Hill Climbing Stochastic - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7605:IF2180_K01	7601:IF2190_K01	-	7607:IF2110_K01
8:00	7601:IF2110_K02	7607:IF2110_K02	7602:IF2110_K01	-	7601:IF2120_K01 7608:IF2140_K01
9:00	-	-	7607:IF2160_K01	-	-
10:00	7606:IF2110_K01 7608:IF2130_K01	7602:IF2140_K01 7608:IF2180_K01	7604:IF2170_K01 7605:IF2210_K01	7601:IF2130_K01	-
11:00	-	7604:IF2200_K01	-	7605:IF2160_K01	7601:IF2110_K02 7604:IF2120_K02 7608:IF2150_K01
12:00	7602:IF2140_K01	-	7606:IF2190_K01 7607:IF2120_K02	7601:IF2150_K01	-
13:00	7607:IF2180_K01	-	-	-	-
14:00	7601:IF2110_K02 7607:IF2130_K01	7604:IF2210_K01 7605:IF2160_K01	7601:IF2120_K02 7604:IF2150_K02	7604:IF2200_K01	7602:IF2110_K01
15:00	7607:IF2150_K02	7608:IF2200_K01	-	7603:IF2170_K01	-
16:00	-	-	7602:IF2190_K01 7608:IF2120_K01	7604:IF2140_K01 7607:IF2111_K01	-
17:00	7606:IF2110_K01	-	-	7602:IF2111_K01 7604:IF2180_K01	-

Hill Climbing Stochastic - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7605:IF2180_K01	7601:IF2190_K01	-	7607:IF2110_K01
8:00	7601:IF2110_K02	7607:IF2110_K02	7602:IF2110_K01	-	7608:IF2140_K01
9:00	7601:IF2150_K01	-	7607:IF2160_K01	-	-
10:00	7608:IF2130_K01	7602:IF2140_K01	7604:IF2170_K01 7605:IF2210_K01	7601:IF2130_K01	-
11:00	-	7604:IF2200_K01	7602:IF2210_K01	7605:IF2160_K01	7604:IF2120_K02
12:00	7602:IF2140_K01	-	7606:IF2190_K01 7607:IF2120_K02	7601:IF2150_K01	-
13:00	7607:IF2180_K01	-	-	-	-
14:00	7601:IF2110_K02 7607:IF2130_K01	7605:IF2160_K01	7604:IF2150_K02	7604:IF2200_K01	7602:IF2110_K01 7604:IF2111_K01
15:00	7607:IF2150_K02	7608:IF2200_K01	-	7603:IF2170_K01	7603:IF2120_K01
16:00	-	7605:IF2110_K02 7606:IF2180_K01	7602:IF2190_K01 7604:IF2111_K01	7604:IF2140_K01	7603:IF2120_K01
17:00	7606:IF2110_K01	-	-	7604:IF2180_K01	7605:IF2120_K01 7606:IF2120_K02



Final Objective Value: 0

Search Duration: 74.6416 seconds

Total Iterations: 99

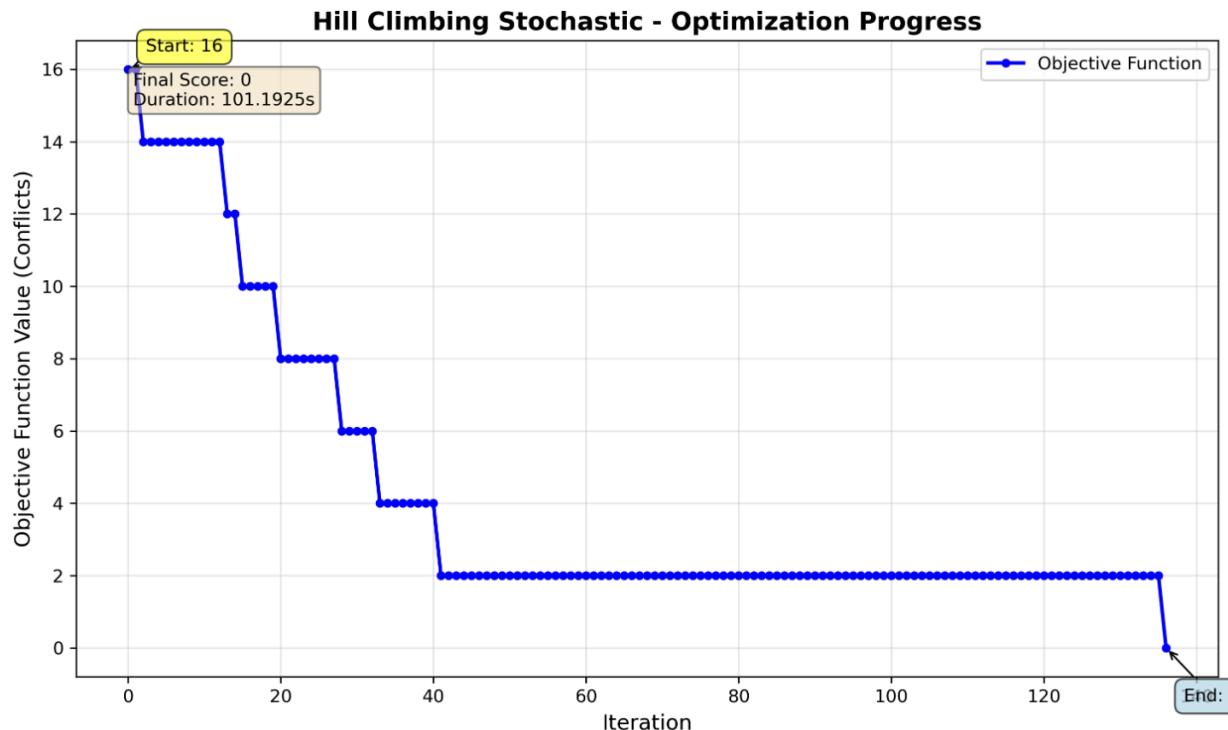
2.3.2.2 Percobaan 2

Hill Climbing Stochastic - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7601:IF2160_K01	7604:IF2200_K01	7605:IF2170_K01 7607:IF2120_K02	7606:IF2160_K01	7602:IF2110_K02
8:00	7605:IF2120_K01	7602:IF2140_K01	-	-	7604:IF2150_K01
9:00	-	7607:IF2110_K01	-	7605:IF2110_K02	-
10:00	-	7602:IF2210_K01 7603:IF2110_K02	-	-	-
11:00	-	7601:IF2140_K01 7604:IF2150_K01	-	7606:IF2110_K01	-
12:00	7605:IF2180_K01 7606:IF2111_K01	7602:IF2150_K02 7603:IF2190_K01	7601:IF2200_K01 7602:IF2120_K02 7608:IF2190_K01	-	7601:IF2190_K01 7604:IF2130_K01
13:00	-	7607:IF2110_K01 7608:IF2170_K01	-	-	7607:IF2110_K02
14:00	-	7604:IF2180_K01	7603:IF2180_K01	-	7603:IF2120_K02 7607:IF2210_K01
15:00	-	7607:IF2140_K01	7606:IF2111_K01	7604:IF2130_K01	-
16:00	-	-	7605:IF2130_K01	7604:IF2200_K01	7606:IF2120_K01
17:00	7604:IF2150_K02 7606:IF2120_K01	7606:IF2110_K01	7603:IF2180_K01 7608:IF2160_K01	7608:IF2140_K01	-

Hill Climbing Stochastic - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7601:IF2160_K01	7604:IF2200_K01	7605:IF2170_K01	7606:IF2160_K01	7602:IF2110_K02
8:00	7605:IF2120_K01 7606:IF2120_K02	7602:IF2140_K01	-	-	7604:IF2150_K01
9:00	-	7607:IF2110_K01	-	7605:IF2110_K02	7605:IF2150_K02
10:00	7604:IF2120_K02	7602:IF2210_K01	-	-	-
11:00	-	7601:IF2140_K01 7604:IF2150_K01	-	7603:IF2111_K01 7606:IF2110_K01	7603:IF2190_K01
12:00	7605:IF2180_K01	7603:IF2190_K01	7601:IF2200_K01 7608:IF2190_K01	-	7604:IF2130_K01
13:00	-	7607:IF2110_K01 7608:IF2170_K01	-	-	7607:IF2110_K02
14:00	-	7604:IF2180_K01	7603:IF2180_K01	-	7607:IF2210_K01
15:00	-	7607:IF2140_K01	7606:IF2111_K01	7604:IF2130_K01	7605:IF2110_K02
16:00	-	-	7605:IF2130_K01	7604:IF2200_K01	7605:IF2120_K02 7606:IF2120_K01
17:00	7604:IF2150_K02 7606:IF2120_K01	7606:IF2110_K01	7603:IF2180_K01 7608:IF2160_K01	7608:IF2140_K01	-



Final Objective Value: 0

Search Duration: 101.1925 seconds

Total Iterations: 136

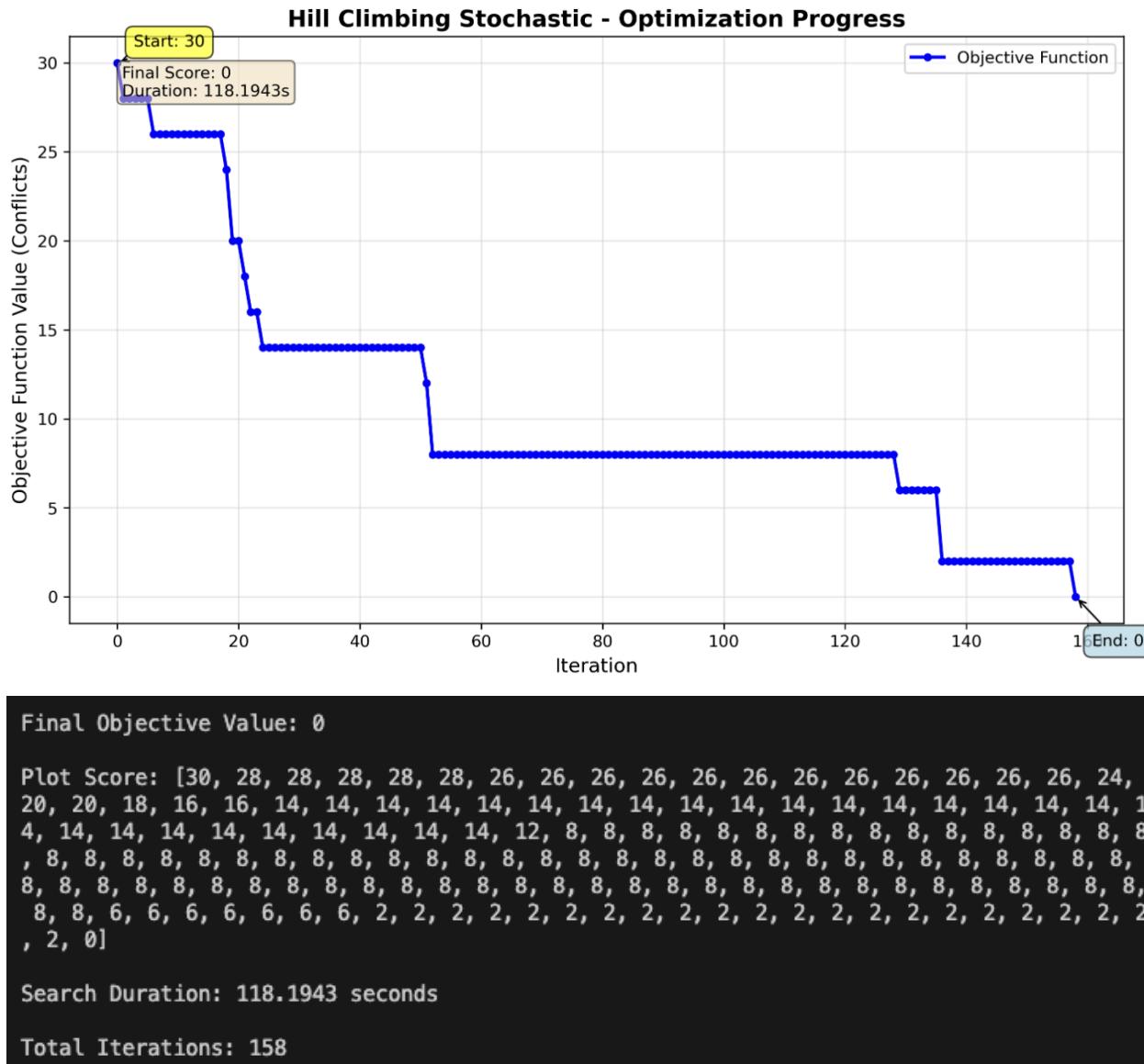
2.3.2.3 Percobaan 3

Hill Climbing Stochastic - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7602:IF2190_K01 7607:IF2110_K02	7606:IF2110_K01 7607:IF2180_K01	7603:IF2120_K02	-	-
8:00	-	7605:IF2130_K01	7603:IF2150_K01	7605:IF2111_K01	-
9:00	7601:IF2170_K01 7603:IF2110_K02 7606:IF2110_K01	7602:IF2120_K02	-	-	7608:IF2110_K01
10:00	-	-	7602:IF2120_K02 7603:IF2190_K01 7605:IF2200_K01	-	-
11:00	7604:IF2140_K01	7603:IF2130_K01	7602:IF2110_K02 7606:IF2140_K01	7602:IF2120_K01 7606:IF2170_K01 7607:IF2180_K01	-
12:00	7602:IF2200_K01 7603:IF2180_K01	-	-	7606:IF2180_K01	-
13:00	-	-	7608:IF2140_K01	7603:IF2190_K01 7604:IF2110_K01 7605:IF2160_K01	7604:IF2110_K02
14:00	-	7604:IF2210_K01 7606:IF2120_K01	7605:IF2150_K02	7605:IF2160_K01	-
15:00	7606:IF2160_K01 7608:IF2150_K01	-	7602:IF2140_K01	-	-
16:00	7603:IF2120_K01	7607:IF2210_K01	-	7602:IF2130_K01	7603:IF2150_K02
17:00	-	7602:IF2111_K01 7606:IF2200_K01	-	-	-

Hill Climbing Stochastic - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7607:IF2110_K02	7607:IF2180_K01	7603:IF2120_K02	-	7605:IF2120_K01
8:00	-	7605:IF2130_K01	7603:IF2150_K01	7605:IF2111_K01	-
9:00	7603:IF2110_K02 7606:IF2110_K01	7602:IF2120_K02 7604:IF2190_K01	-	7604:IF2170_K01	7608:IF2110_K01
10:00	-	-	7603:IF2190_K01 7605:IF2200_K01	-	7606:IF2110_K01
11:00	7604:IF2140_K01	7603:IF2130_K01	7602:IF2110_K02	7606:IF2170_K01 7607:IF2180_K01	-
12:00	7602:IF2200_K01 7603:IF2180_K01	-	-	7606:IF2180_K01	7605:IF2111_K01
13:00	-	7606:IF2120_K02	7608:IF2140_K01	7605:IF2160_K01	7604:IF2110_K02
14:00	7603:IF2190_K01	7606:IF2120_K01	7605:IF2150_K02	7605:IF2160_K01	-
15:00	7606:IF2160_K01	-	7602:IF2140_K01	-	7603:IF2210_K01
16:00	7603:IF2120_K01	7607:IF2210_K01	7605:IF2140_K01	7602:IF2130_K01	7603:IF2150_K02
17:00	-	7606:IF2200_K01	7606:IF2110_K01	7606:IF2150_K01	-



2.3.3 Hasil Hill-climbing with Sideways

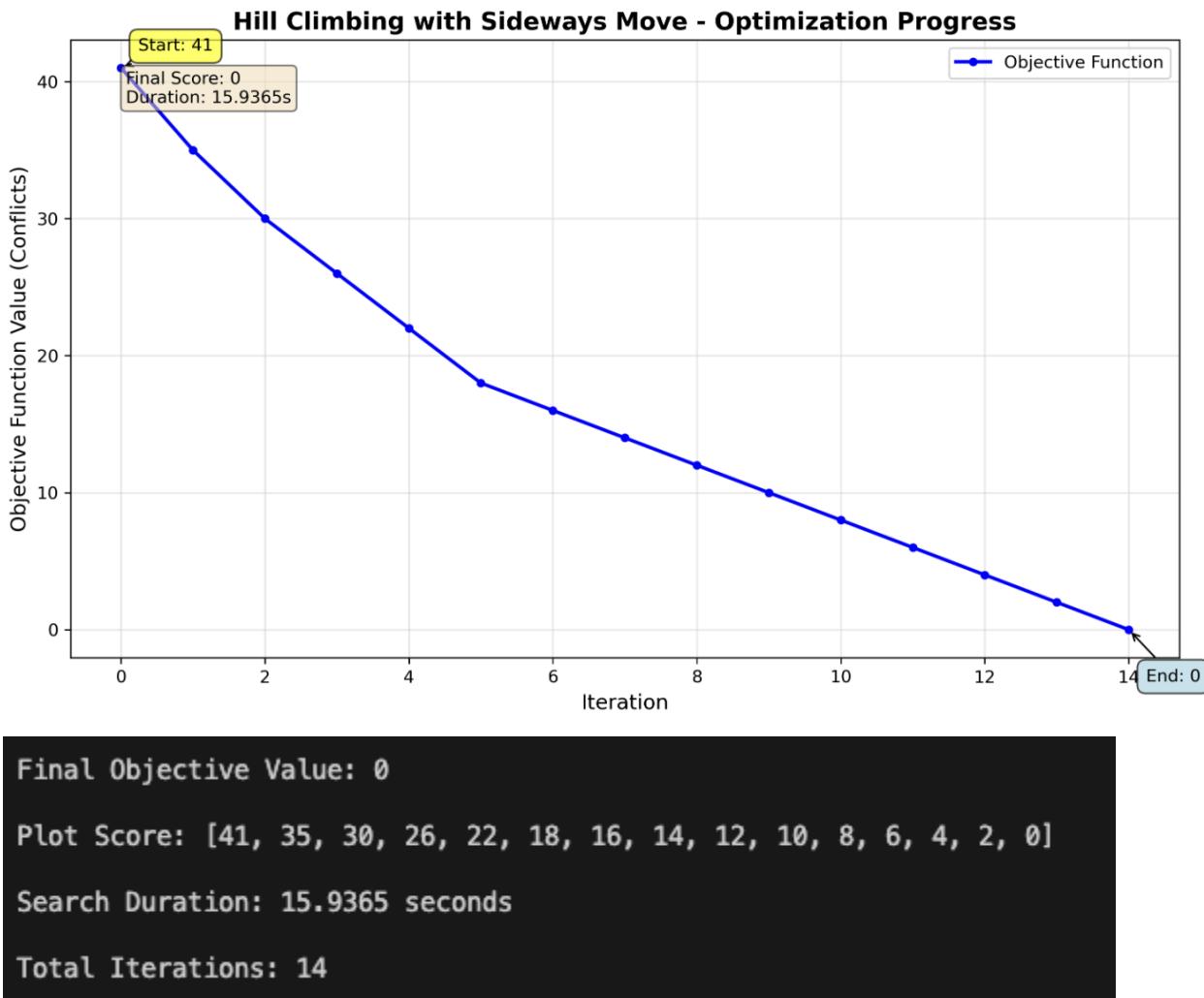
2.3.3.1 Percobaan 1 (max sideways move = 20)

Hill Climbing with Sideways Move - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	-	7603:IF2120_K01	7608:IF2140_K01
8:00	-	-	7604:IF2200_K01 7607:IF2120_K02 7608:IF2150_K02	-	7607:IF2140_K01
9:00	7604:IF2120_K02 7608:IF2110_K02	7601:IF2210_K01	-	-	7603:IF2160_K01 7605:IF2120_K01
10:00	-	7603:IF2110_K01	-	7606:IF2170_K01 7608:IF2190_K01	-
11:00	7602:IF2210_K01	7601:IF2111_K01 7603:IF2180_K01	-	7608:IF2120_K01	-
12:00	-	-	-	-	-
13:00	-	-	7603:IF2180_K01 7608:IF2110_K02	-	-
14:00	7604:IF2150_K02 7608:IF2130_K01	7604:IF2190_K01 7605:IF2170_K01 7606:IF2110_K01	7604:IF2110_K01	-	7608:IF2200_K01
15:00	-	7605:IF2180_K01 7607:IF2130_K01 7608:IF2140_K01	7606:IF2120_K02	-	7603:IF2190_K01 7607:IF2110_K01
16:00	7603:IF2160_K01 7606:IF2150_K01	7601:IF2110_K02 7607:IF2150_K01	7608:IF2130_K01	-	7602:IF2200_K01 7603:IF2111_K01
17:00	-	-	7601:IF2160_K01 7606:IF2140_K01	7604:IF2110_K02	-

Hill Climbing with Sideways Move - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	-	7603:IF2120_K01	7608:IF2140_K01
8:00	-	-	7607:IF2120_K02	-	7607:IF2140_K01
9:00	7608:IF2110_K02	7601:IF2210_K01	-	-	7603:IF2160_K01 7605:IF2120_K01
10:00	-	7603:IF2110_K01	-	7606:IF2170_K01 7608:IF2190_K01	-
11:00	7602:IF2210_K01	7601:IF2111_K01	-	7608:IF2120_K01	7605:IF2140_K01 7606:IF2150_K01 7608:IF2150_K02
12:00	-	-	-	-	7606:IF2180_K01 7608:IF2150_K02
13:00	-	-	7603:IF2180_K01 7608:IF2110_K02	-	7605:IF2160_K01 7606:IF2180_K01
14:00	7608:IF2130_K01	7605:IF2170_K01 7606:IF2110_K01	7604:IF2110_K01	-	7605:IF2160_K01 7606:IF2190_K01 7608:IF2200_K01
15:00	-	7607:IF2130_K01	7606:IF2120_K02	-	7606:IF2200_K01 7607:IF2110_K01
16:00	7606:IF2150_K01	7601:IF2110_K02	7608:IF2130_K01	-	7603:IF2111_K01 7605:IF2120_K02 7606:IF2190_K01
17:00	-	-	7606:IF2140_K01	7604:IF2110_K02	7605:IF2200_K01 7606:IF2180_K01



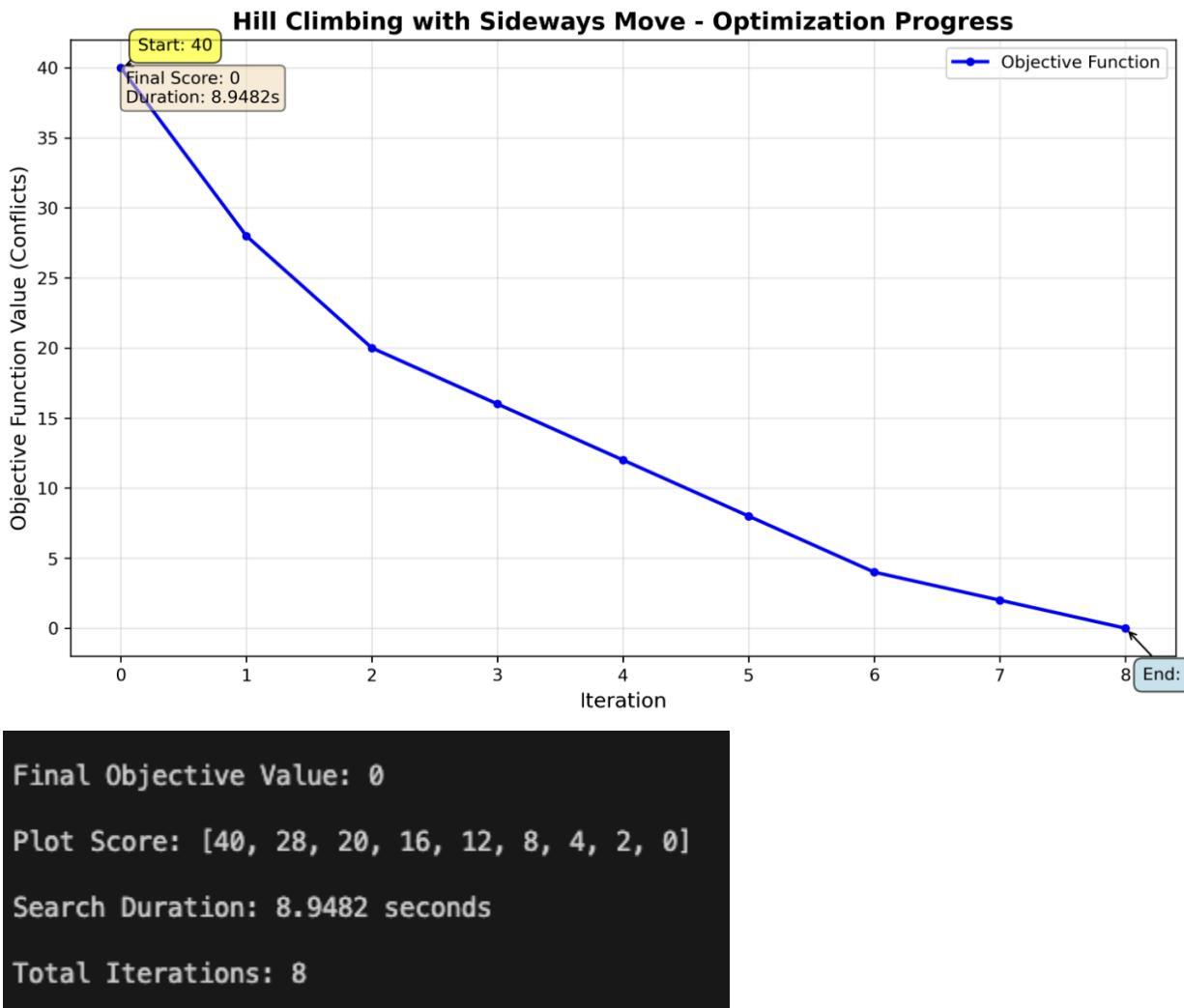
2.3.3.2 Percobaan 2 (max sideways move = 40)

Hill Climbing with Sideways Move - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7604:IF2120_K01 7607:IF2110_K01	7603:IF2160_K01	7608:IF2120_K02	-	7601:IF2140_K01
8:00	7604:IF2190_K01 7608:IF2120_K02	-	7605:IF2150_K01	7604:IF2200_K01 7608:IF2210_K01	7603:IF2120_K02 7605:IF2110_K02
9:00	-	7603:IF2130_K01	7602:IF2170_K01	-	-
10:00	-	7607:IF2120_K01	7601:IF2190_K01	-	7608:IF2180_K01
11:00	-	7606:IF2130_K01	-	-	-
12:00	-	7601:IF2170_K01 7603:IF2200_K01	-	7601:IF2140_K01 7603:IF2140_K01 7608:IF2180_K01	7604:IF2110_K02
13:00	7601:IF2200_K01 7607:IF2111_K01	7604:IF2130_K01	7608:IF2210_K01	-	-
14:00	7602:IF2160_K01	-	7607:IF2150_K02	7605:IF2150_K02	7603:IF2110_K01
15:00	7603:IF2140_K01	7606:IF2111_K01	-	7602:IF2190_K01	7605:IF2110_K02 7608:IF2160_K01
16:00	-	7607:IF2150_K01	-	7601:IF2180_K01	7605:IF2110_K02
17:00	7602:IF2110_K01	-	-	7604:IF2110_K01 7606:IF2180_K01	7602:IF2120_K01

Hill Climbing with Sideways Move - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7607:IF2110_K01	7603:IF2160_K01	7608:IF2120_K02	-	7601:IF2140_K01
8:00	7604:IF2190_K01 7608:IF2120_K02	-	7605:IF2150_K01	7604:IF2200_K01 7608:IF2210_K01	7605:IF2110_K02
9:00	-	7603:IF2130_K01	7602:IF2170_K01	-	-
10:00	-	7607:IF2120_K01	7601:IF2190_K01	-	7606:IF2120_K02 7608:IF2180_K01
11:00	-	7606:IF2130_K01	-	-	7606:IF2200_K01
12:00	-	7601:IF2170_K01	-	7603:IF2140_K01	7604:IF2110_K02 7606:IF2120_K01
13:00	7607:IF2111_K01	7604:IF2130_K01	7608:IF2210_K01	-	7606:IF2140_K01
14:00	7602:IF2160_K01	-	7607:IF2150_K02	7605:IF2150_K02	7603:IF2110_K01 7606:IF2200_K01
15:00	7603:IF2140_K01	7606:IF2111_K01	-	7602:IF2190_K01	7605:IF2110_K02 7606:IF2180_K01
16:00	-	7607:IF2150_K01	-	7601:IF2180_K01	7605:IF2110_K02 7606:IF2180_K01
17:00	7602:IF2110_K01	-	-	7604:IF2110_K01	7602:IF2120_K01 7606:IF2160_K01



2.3.3.3 Percobaan 3 (max sideways move = 60)

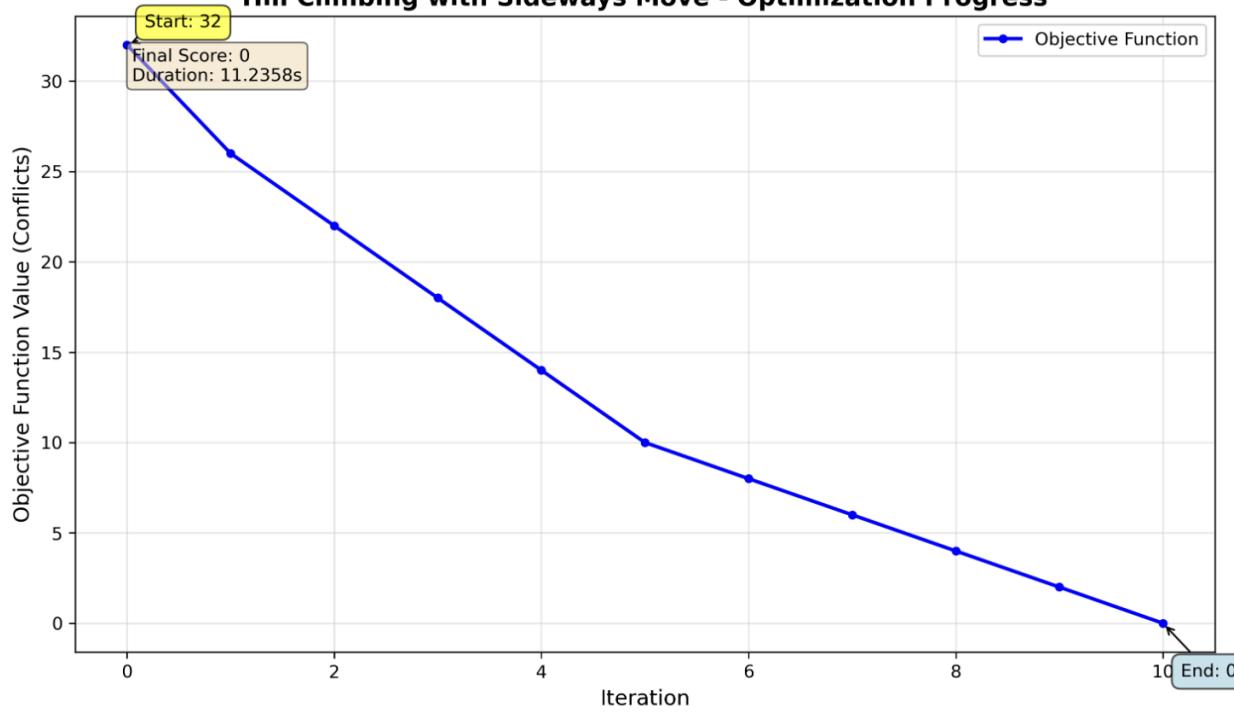
Hill Climbing with Sideways Move - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00		-	-	-	-
8:00	7601:IF2130_K01 7604:IF2210_K01 7607:IF2190_K01	-	7602:IF2110_K02 7606:IF2111_K01	7601:IF2200_K01 7606:IF2110_K02	-
9:00	-	-	7602:IF2111_K01	-	7607:IF2150_K01
10:00	7606:IF2120_K01	7601:IF2130_K01 7604:IF2120_K01	7601:IF2110_K01	-	7602:IF2140_K01 7605:IF2180_K01
11:00	-	7604:IF2150_K01	-	7604:IF2110_K01 7606:IF2110_K02	7604:IF2180_K01
12:00	7605:IF2190_K01	-	7607:IF2120_K02	7601:IF2170_K01	-
13:00	7608:IF2200_K01	7607:IF2160_K01	7604:IF2130_K01 7605:IF2140_K01	7602:IF2140_K01 7604:IF2180_K01	-
14:00	7604:IF2180_K01	-	-	-	-
15:00	7603:IF2210_K01	-	-	7603:IF2110_K02	7601:IF2200_K01 7602:IF2190_K01 7604:IF2120_K02
16:00	7604:IF2140_K01	7601:IF2120_K01 7606:IF2170_K01	7608:IF2150_K02	7608:IF2110_K01	7608:IF2150_K02
17:00	-	-	7603:IF2160_K01 7605:IF2110_K01 7607:IF2120_K02	-	7605:IF2160_K01

Hill Climbing with Sideways Move - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	-	-	-
8:00	7604:IF2210_K01 7607:IF2190_K01	-	7602:IF2110_K02	7606:IF2110_K02	-
9:00	-	-	7602:IF2111_K01	-	7606:IF2140_K01 7607:IF2150_K01
10:00	7606:IF2120_K01	7604:IF2120_K01	7601:IF2110_K01	-	7602:IF2140_K01
11:00	-	7604:IF2150_K01	-	7604:IF2110_K01 7606:IF2110_K02	7604:IF2180_K01
12:00	7605:IF2190_K01	-	7607:IF2120_K02	7601:IF2170_K01	7606:IF2130_K01
13:00	7608:IF2200_K01	7607:IF2160_K01	7604:IF2130_K01	7602:IF2140_K01	7606:IF2130_K01
14:00	7604:IF2180_K01	-	-	-	7605:IF2200_K01 7606:IF2160_K01
15:00	7603:IF2210_K01	-	-	7603:IF2110_K02	7602:IF2190_K01 7604:IF2120_K02 7606:IF2111_K01
16:00	7604:IF2140_K01	7601:IF2120_K01	7608:IF2150_K02	7608:IF2110_K01	7606:IF2180_K01 7607:IF2170_K01 7608:IF2150_K02
17:00	-	-	7605:IF2110_K01 7607:IF2120_K02	-	7604:IF2200_K01 7605:IF2160_K01 7606:IF2180_K01

Hill Climbing with Sideways Move - Optimization Progress



Final Objective Value: 0

Plot Score: [32, 26, 22, 18, 14, 10, 8, 6, 4, 2, 0]

Search Duration: 11.2358 seconds

Total Iterations: 10

2.3.4 Hasil Random Restart Hill-climbing

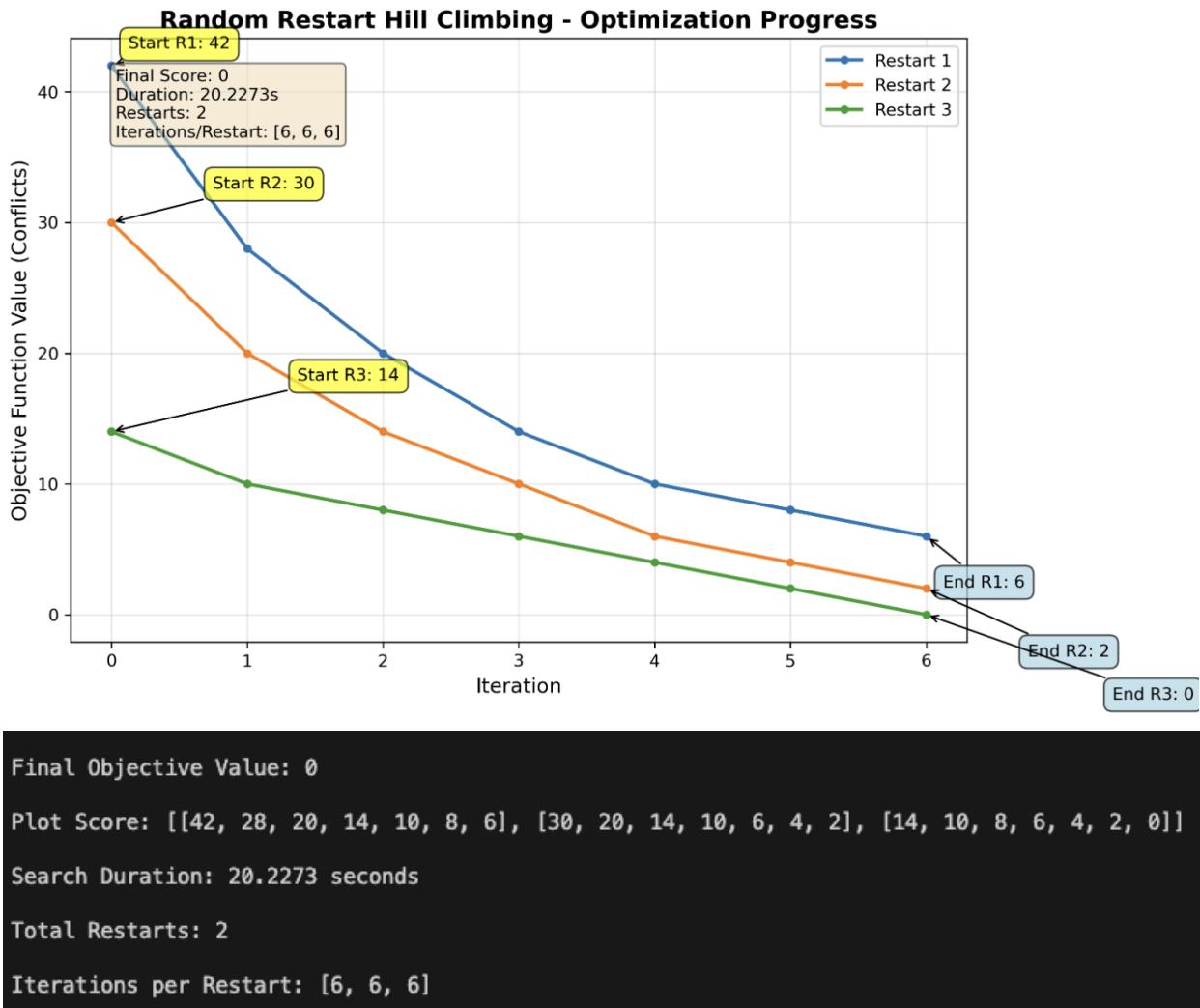
2.3.4.1 Percobaan 1 (max restart = 5, max iteration/restart=6)

Random Restart Hill Climbing - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7605:IF2130_K01	-	7605:IF2111_K01	-	7603:IF2140_K01
8:00	-	7601:IF2120_K01 7602:IF2110_K01	-	7602:IF2111_K01	-
9:00	7602:IF2140_K01 7605:IF2140_K01	-	7605:IF2110_K02 7608:IF2190_K01	7604:IF2210_K01	7602:IF2110_K01 7606:IF2120_K02 7607:IF2150_K01
10:00	7602:IF2180_K01	7606:IF2200_K01	7605:IF2170_K01	-	-
11:00	-	7604:IF2110_K02	-	7602:IF2110_K02 7607:IF2110_K02	7605:IF2120_K01
12:00	-	-	7606:IF2110_K01	7604:IF2180_K01	-
13:00	7605:IF2180_K01 7606:IF2150_K01	-	-	7602:IF2120_K02	7607:IF2190_K01
14:00	7603:IF2190_K01	7604:IF2110_K01	-	7603:IF2150_K02	7602:IF2160_K01
15:00	7607:IF2160_K01	7607:IF2200_K01	-	-	7607:IF2130_K01 7608:IF2140_K01
16:00	-	7604:IF2210_K01	7608:IF2180_K01	-	7602:IF2130_K01 7605:IF2150_K02 7607:IF2120_K01
17:00	7601:IF2200_K01 7603:IF2120_K02	7608:IF2160_K01	-	-	7603:IF2170_K01

Random Restart Hill Climbing - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7604:IF2110_K02 7607:IF2110_K01	7602:IF2200_K01	7602:IF2150_K02 7604:IF2120_K01	7601:IF2150_K01	-
8:00	7608:IF2210_K01	7606:IF2200_K01	7602:IF2140_K01	-	7608:IF2110_K02
9:00	7607:IF2140_K01	7606:IF2180_K01	-	7602:IF2190_K01 7604:IF2120_K02	7607:IF2110_K01
10:00	7604:IF2110_K02 7606:IF2180_K01	7602:IF2130_K01	-	-	7602:IF2140_K01
11:00	7602:IF2130_K01	-	-	7606:IF2130_K01	7601:IF2120_K02
12:00	7604:IF2110_K01 7605:IF2200_K01	-	7602:IF2160_K01	7608:IF2120_K02	-
13:00	7603:IF2170_K01 7608:IF2110_K01	7602:IF2190_K01	7604:IF2160_K01	7606:IF2210_K01	7604:IF2180_K01
14:00	7602:IF2111_K01 7603:IF2120_K01	-	-	-	-
15:00	7607:IF2180_K01	-	7606:IF2190_K01 7608:IF2170_K01	7603:IF2160_K01	-
16:00	7604:IF2140_K01	-	-	7605:IF2150_K01	7602:IF2150_K02
17:00	7608:IF2110_K01	-	7601:IF2110_K02	7601:IF2120_K01	-



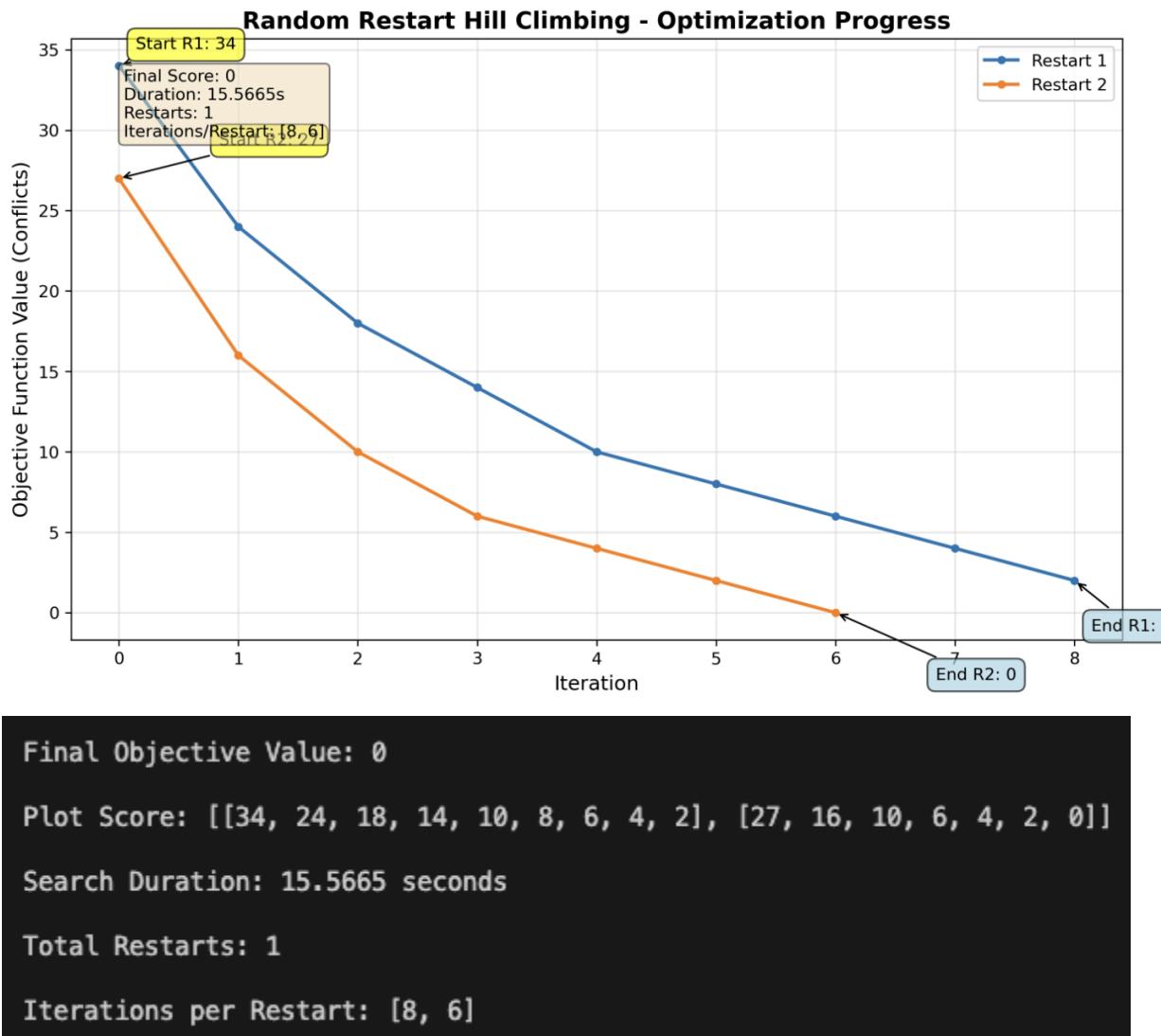
2.3.4.2 Percobaan 2 (max restart = 2, max iteration/restart=8)

Random Restart Hill Climbing - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7602:IF2140_K01 7608:IF2110_K02	-	-	7605:IF2120_K01	-
8:00	-	-	7606:IF2180_K01	7604:IF2111_K01	7601:IF2110_K02 7608:IF2111_K01
9:00	-	-	7601:IF2130_K01 7604:IF2160_K01	-	7601:IF2110_K01
10:00	-	-	-	7605:IF2110_K02	-
11:00	7601:IF2180_K01	7601:IF2180_K01	7604:IF2130_K01	7604:IF2170_K01	-
12:00	-	7602:IF2170_K01	-	7603:IF2110_K02	-
13:00	7608:IF2130_K01	7608:IF2120_K02	7603:IF2140_K01	-	-
14:00	7607:IF2120_K02	7602:IF2110_K01 7608:IF2190_K01	-	7606:IF2150_K02	7603:IF2190_K01
15:00	-	7607:IF2140_K01	7603:IF2150_K01 7605:IF2150_K02	7605:IF2210_K01 7606:IF2140_K01	-
16:00	7605:IF2120_K01 7608:IF2120_K01	7601:IF2120_K02 7602:IF2180_K01 7603:IF2190_K01	7606:IF2200_K01	-	7601:IF2200_K01 7603:IF2150_K01 7605:IF2210_K01
17:00	7601:IF2110_K01	-	7606:IF2160_K01 7607:IF2110_K01	7605:IF2200_K01 7608:IF2160_K01	-

Random Restart Hill Climbing - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7601:IF2200_K01	7605:IF2180_K01 7606:IF2110_K02	7601:IF2110_K01	-	7605:IF2140_K01
8:00	7602:IF2180_K01 7603:IF2120_K02	7607:IF2120_K02	7601:IF2140_K01	-	7606:IF2120_K01
9:00	7602:IF2130_K01 7604:IF2110_K02	-	-	7602:IF2150_K01	7608:IF2190_K01
10:00	7606:IF2210_K01	7607:IF2180_K01	-	7604:IF2140_K01 7608:IF2150_K02	-
11:00	7603:IF2170_K01 7604:IF2110_K01	-	7601:IF2130_K01	-	7602:IF2110_K02
12:00	7604:IF2160_K01	7607:IF2150_K01	-	-	-
13:00	7603:IF2110_K01 7604:IF2120_K02 7606:IF2111_K01	7604:IF2120_K01	7607:IF2210_K01	-	7601:IF2111_K01
14:00	7605:IF2140_K01	7603:IF2110_K02	-	7608:IF2200_K01	7604:IF2160_K01
15:00	7606:IF2190_K01	-	-	-	7602:IF2200_K01 7605:IF2160_K01
16:00	7602:IF2130_K01	7605:IF2150_K02	7603:IF2190_K01	-	7603:IF2110_K01
17:00	7608:IF2180_K01	7605:IF2120_K01	-	7602:IF2170_K01	-



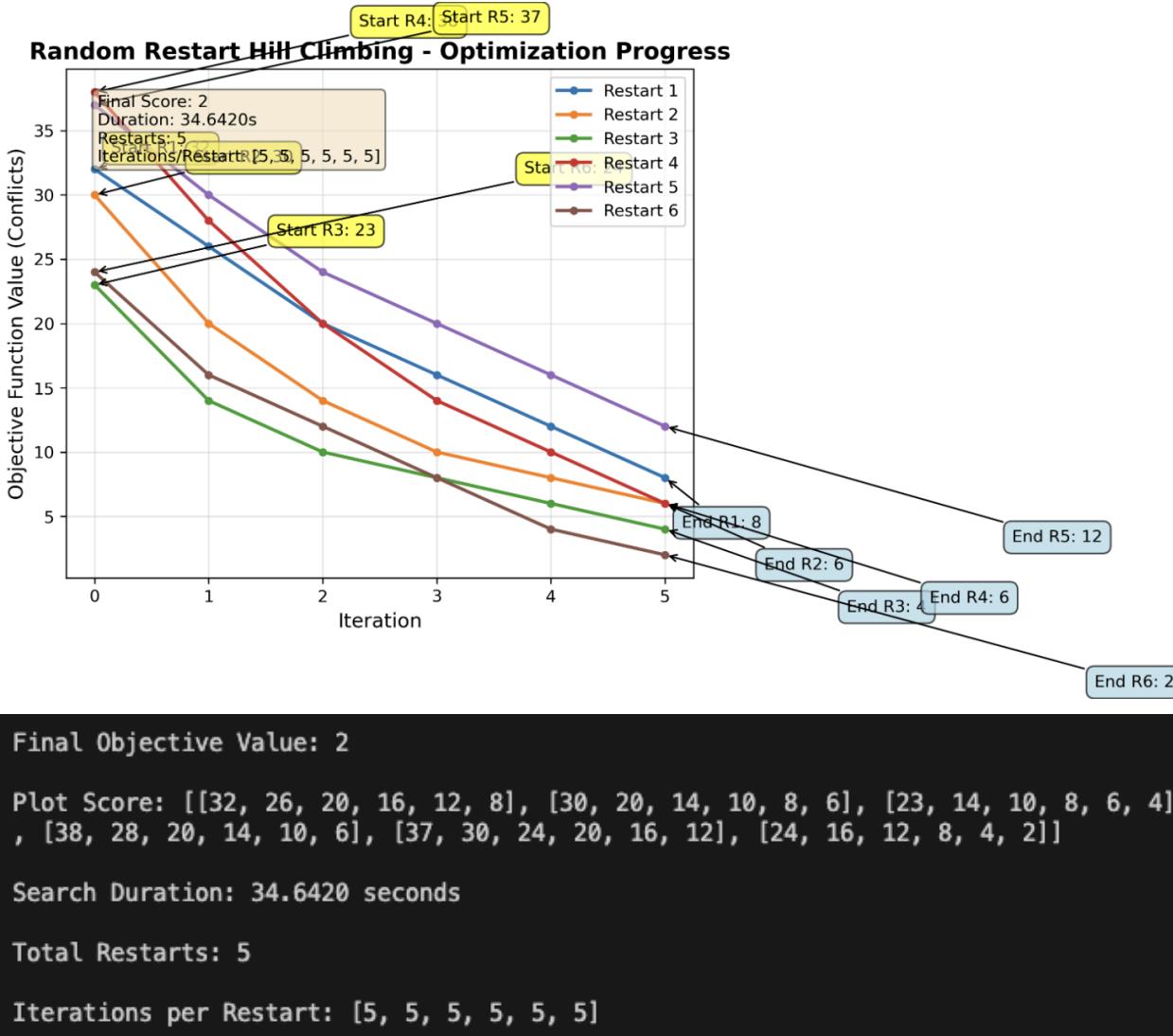
2.3.4.3 Percobaan 3 (max restart = 6, max iteration/restart=5)

Random Restart Hill Climbing - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7605:IF2110_K01 7607:IF2180_K01	-	-	-	7601:IF2111_K01
8:00	-	7604:IF2180_K01 7605:IF2120_K02	-	7602:IF2190_K01	-
9:00	-	7606:IF2110_K02	7603:IF2150_K01	-	-
10:00	7602:IF2111_K01	7602:IF2150_K02	7601:IF2190_K01	-	-
11:00	-	7605:IF2120_K01	-	-	7606:IF2110_K02
12:00	7607:IF2200_K01	7602:IF2170_K01 7603:IF2110_K02	7604:IF2180_K01 7605:IF2210_K01 7608:IF2200_K01	-	-
13:00	7602:IF2120_K02	7606:IF2120_K01	7602:IF2130_K01 7603:IF2140_K01 7606:IF2210_K01	-	7606:IF2160_K01
14:00	-	-	7601:IF2110_K01	7602:IF2190_K01	7606:IF2160_K01
15:00	7607:IF2120_K02	7601:IF2150_K01 7602:IF2130_K01	7602:IF2110_K01 7606:IF2140_K01	7602:IF2140_K01	7604:IF2170_K01
16:00	-	7605:IF2110_K01	7606:IF2180_K01	-	-
17:00	7607:IF2200_K01	7602:IF2110_K02 7603:IF2140_K01 7607:IF2120_K01	7603:IF2160_K01 7604:IF2130_K01	-	-

Random Restart Hill Climbing - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7602:IF2120_K02 7603:IF2120_K01 7604:IF2111_K01	7604:IF2170_K01 7608:IF2150_K02	7602:IF2150_K01	7607:IF2160_K01	-
8:00	7604:IF2200_K01	7605:IF2170_K01	7605:IF2200_K01	-	-
9:00	7605:IF2160_K01	7604:IF2111_K01	7604:IF2180_K01	7605:IF2110_K01	7606:IF2110_K01 7607:IF2110_K02
10:00	7603:IF2120_K02 7604:IF2120_K01	7606:IF2110_K02	-	-	7605:IF2110_K02
11:00	7607:IF2110_K01	-	7608:IF2180_K01	-	-
12:00	7605:IF2140_K01	-	7601:IF2120_K01	7603:IF2160_K01	7604:IF2110_K01
13:00	7602:IF2130_K01	-	7604:IF2190_K01	-	7603:IF2180_K01
14:00	7605:IF2150_K02 7607:IF2210_K01	7601:IF2210_K01 7602:IF2140_K01	7604:IF2120_K02 7605:IF2190_K01	-	-
15:00	7605:IF2140_K01	7608:IF2200_K01	7601:IF2190_K01	7601:IF2150_K01	-
16:00	7602:IF2130_K01	7608:IF2110_K02	7604:IF2140_K01	7604:IF2180_K01	-
17:00	-	-	-	-	7602:IF2130_K01



2.3.5 Hasil Simulated Annealing

Parameters: Initial Temperature=100; Cooling Rate=0.95; Max Iterations=None;
Probability Threshold Value=Random

2.3.5.1 Percobaan 1

Simulated Annealing - Initial Schedule

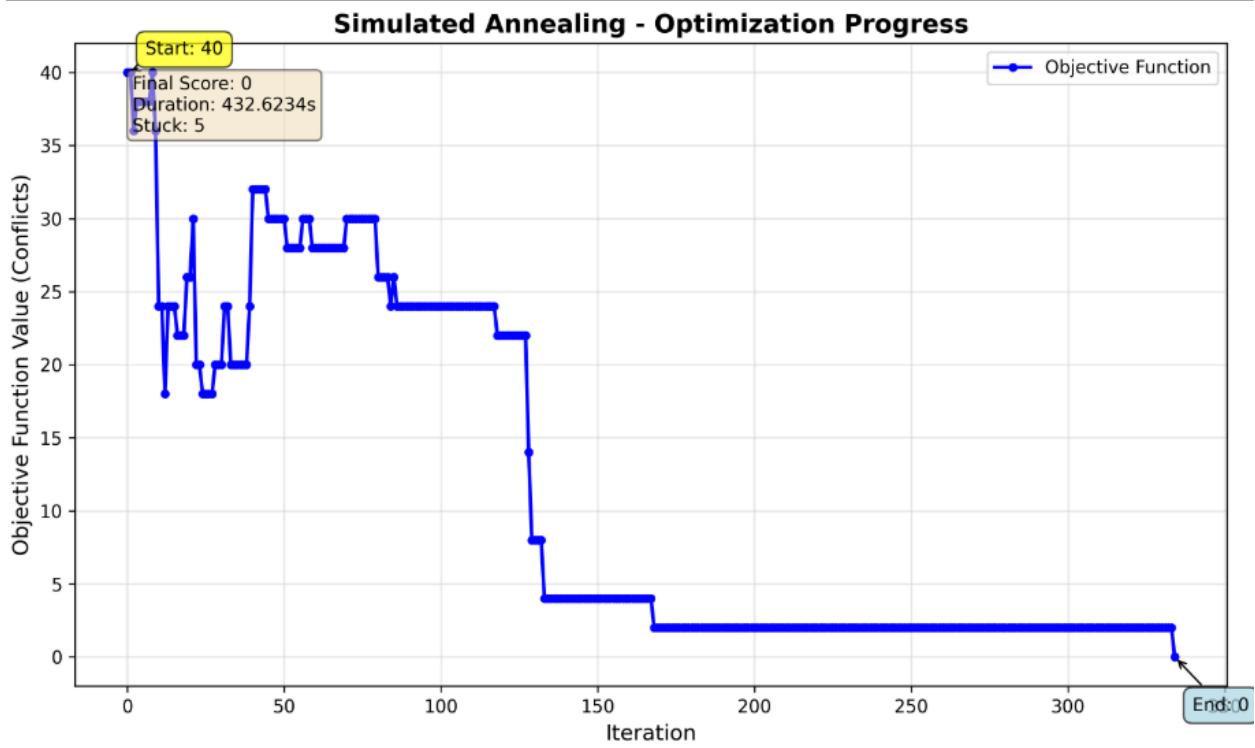
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7601:IF2210_K01	-	-	7603:IF2130_K01 7605:IF2200_K01	-
8:00	7601:IF2180_K01	7602:IF2190_K01	-	7602:IF2110_K02 7603:IF2160_K01 7605:IF2170_K01	-
9:00	-	7602:IF2160_K01	7602:IF2200_K01	-	7605:IF2110_K01
10:00	-	7601:IF2180_K01	-	7608:IF2150_K01	7603:IF2111_K01
11:00	7604:IF2110_K01 7606:IF2190_K01 7607:IF2120_K02	-	-	7607:IF2120_K01	-
12:00	-	7607:IF2120_K01	7601:IF2110_K02 7603:IF2170_K01	7601:IF2160_K01	7603:IF2150_K01 7606:IF2190_K01
13:00	-	-	-	7602:IF2140_K01	7603:IF2130_K01 7608:IF2130_K01
14:00	7605:IF2140_K01	7604:IF2120_K02	7601:IF2110_K01 7605:IF2180_K01	7603:IF2200_K01 7607:IF2110_K01	7606:IF2110_K02
15:00	7602:IF2150_K02 7607:IF2140_K01	-	-	-	7605:IF2120_K02 7608:IF2120_K01
16:00	-	7601:IF2110_K02 7607:IF2111_K01	-	7604:IF2140_K01	7607:IF2150_K02
17:00	-	-	-	-	7602:IF2210_K01 7603:IF2180_K01

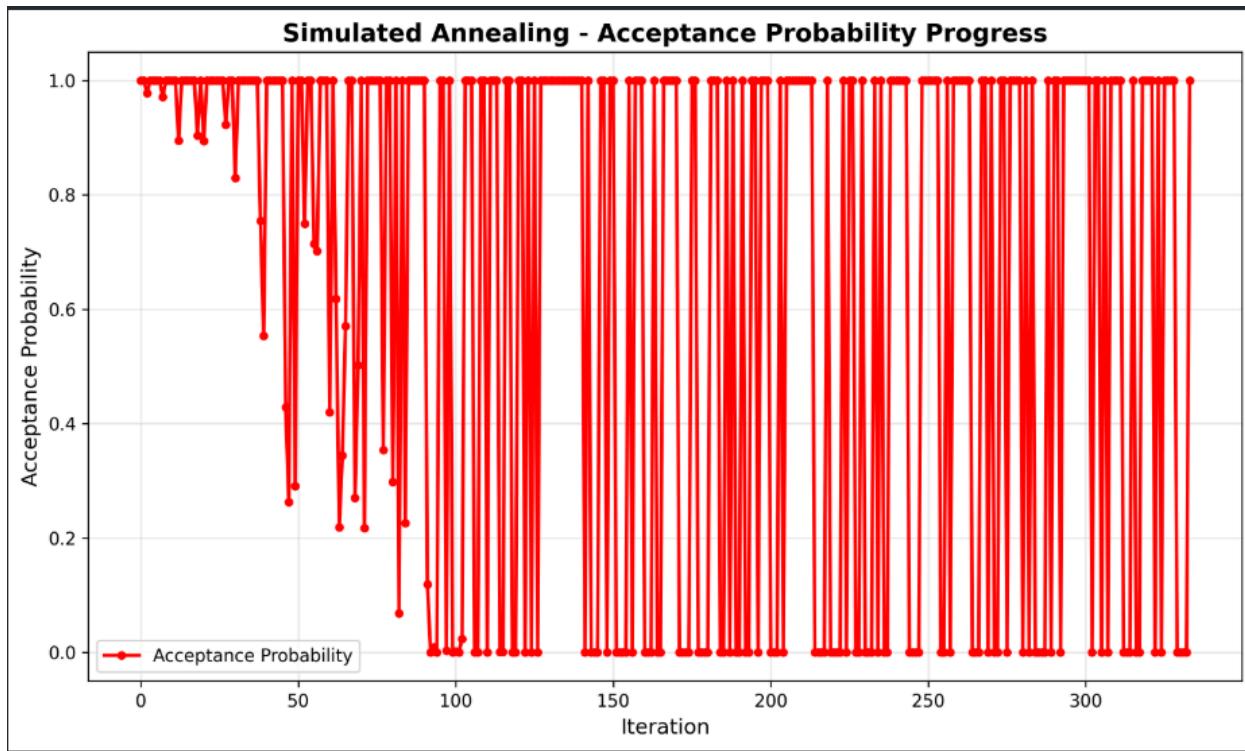
Figure 1

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	7604:IF2150_K01 7605:IF2140_K01	7605:IF2110_K02	7604:IF2130_K01
8:00	7604:IF2200_K01	7603:IF2200_K01	7606:IF2111_K01	7603:IF2160_K01 7606:IF2190_K01	7603:IF2190_K01
9:00	-	7603:IF2210_K01	-	7605:IF2110_K01 7606:IF2110_K02	-
10:00	-	7601:IF2180_K01	-	-	-
11:00	7606:IF2150_K02	7601:IF2150_K01 7606:IF2140_K01	7603:IF2130_K01	7606:IF2160_K01	7603:IF2130_K01
12:00	7604:IF2110_K01	-	7606:IF2110_K01	7601:IF2160_K01	-
13:00	7604:IF2190_K01	7604:IF2110_K02	-	7605:IF2120_K01 7606:IF2110_K02	-
14:00	7605:IF2210_K01	7606:IF2170_K01	7605:IF2140_K01	-	-
15:00	7607:IF2140_K01	7605:IF2200_K01	7606:IF2180_K01	-	7604:IF2170_K01
16:00	7604:IF2110_K01	-	7603:IF2150_K02 7605:IF2120_K01	-	7606:IF2180_K01
17:00	-	7604:IF2120_K02	7601:IF2150_K02 7605:IF2120_K01	7606:IF2120_K01	7603:IF2180_K01 7606:IF2120_K02

Search Duration: 432.6234 seconds
Stuck Frequency: 5

Final Objective Value: 0





2.3.5.2 Percobaan 2

Simulated Annealing - Initial Schedule

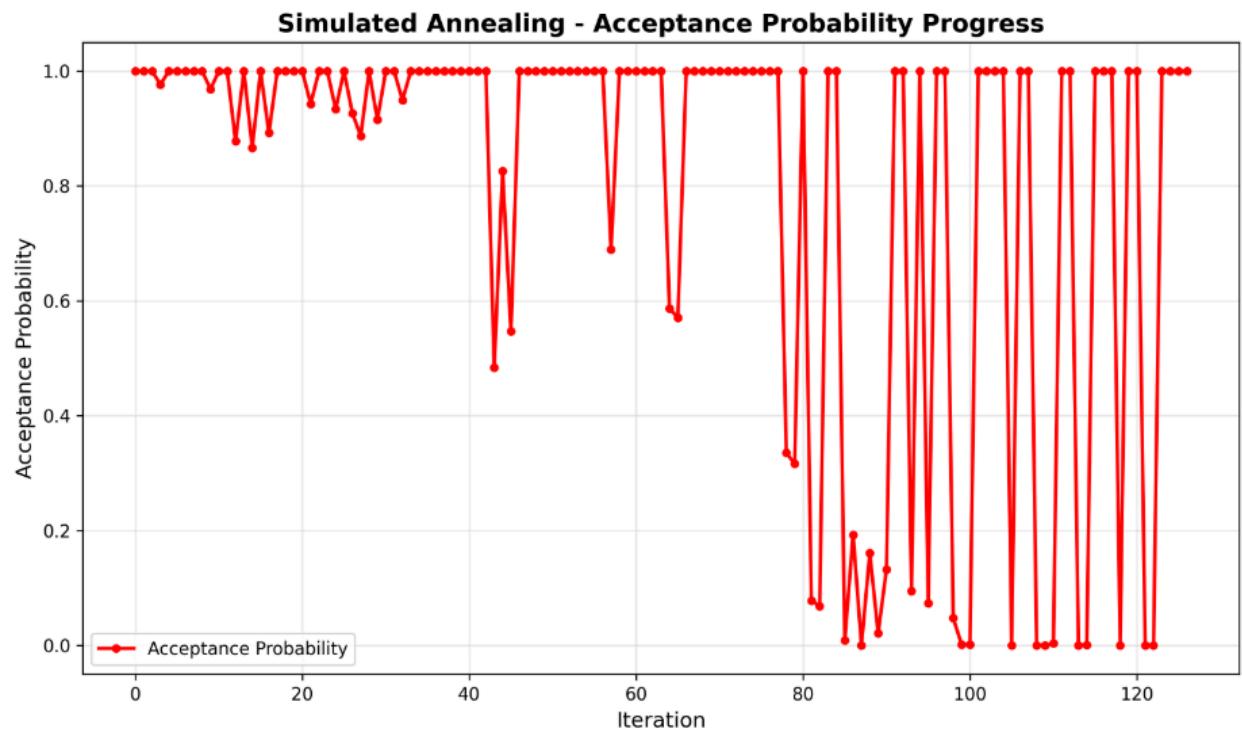
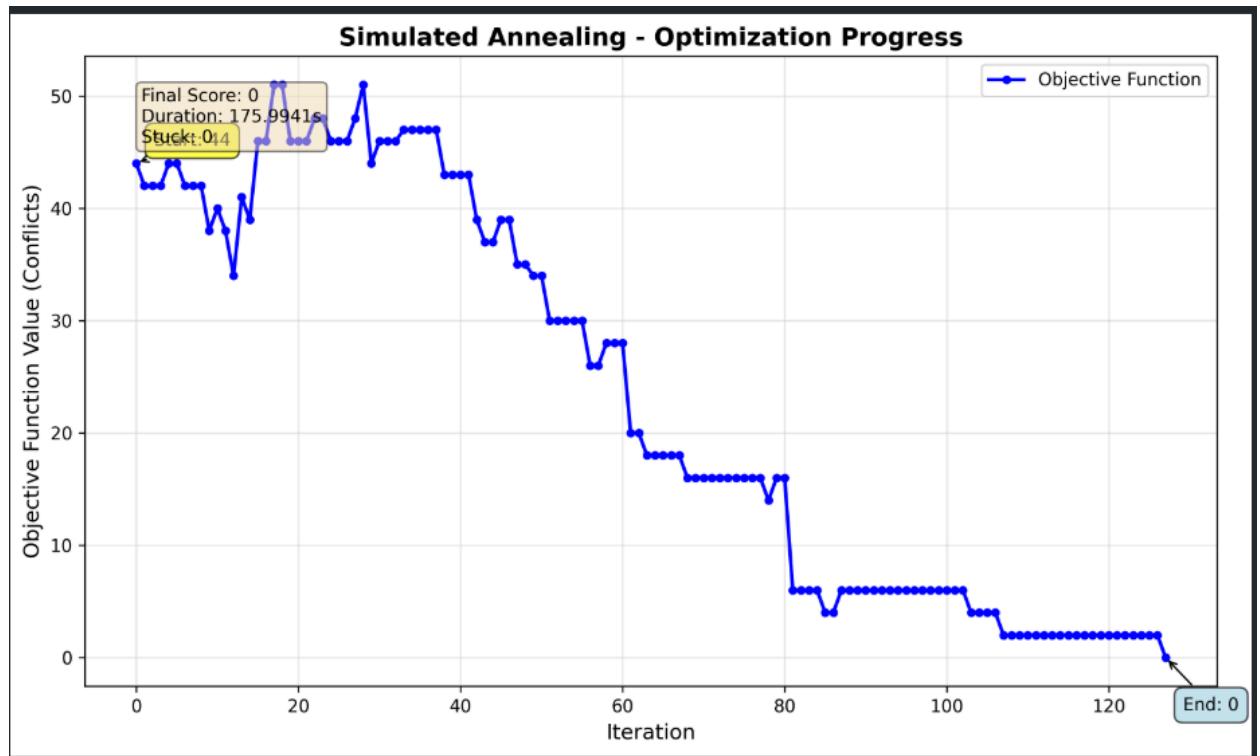
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	-	7603:IF2210_K01	7605:IF2210_K01
8:00	7608:IF2110_K01	7601:IF2170_K01 7607:IF2150_K01	7604:IF2111_K01	-	7604:IF2140_K01 7608:IF2120_K02
9:00	7605:IF2140_K01	7607:IF2110_K02 7608:IF2150_K02	-	7605:IF2200_K01	7603:IF2160_K01
10:00	7602:IF2110_K02	-	-	7602:IF2120_K01	7603:IF2110_K02 7605:IF2200_K01
11:00	-	-	-	7604:IF2110_K01	-
12:00	-	7601:IF2110_K01 7604:IF2130_K01 7607:IF2110_K02	-	-	-
13:00	-	7606:IF2170_K01	7604:IF2120_K02 7606:IF2140_K01	7602:IF2130_K01 7605:IF2190_K01 7608:IF2150_K02	7604:IF2180_K01
14:00	-	-	7605:IF2160_K01 7608:IF2160_K01	7601:IF2120_K01	7607:IF2120_K02
15:00	-	-	7601:IF2180_K01 7606:IF2130_K01	7601:IF2190_K01 7602:IF2150_K01 7608:IF2120_K01	-
16:00	-	-	-	7603:IF2110_K01 7606:IF2200_K01 7608:IF2190_K01	-
17:00	7601:IF2180_K01	-	7605:IF2180_K01	7604:IF2140_K01	7606:IF2111_K01

Simulated Annealing - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7606:IF2120_K02	7606:IF2120_K01	7607:IF2170_K01	-
8:00	-	7603:IF2120_K01	-	7605:IF2190_K01 7606:IF2160_K01	7606:IF2160_K01
9:00	7605:IF2140_K01	7605:IF2120_K02	7608:IF2150_K02	-	7604:IF2130_K01
10:00	-	-	7604:IF2110_K01	7605:IF2200_K01	7603:IF2110_K02
11:00	7604:IF2190_K01	7602:IF2150_K01	-	7604:IF2110_K01	7605:IF2111_K01
12:00	7601:IF2170_K01	-	7604:IF2130_K01	7603:IF2190_K01 7605:IF2210_K01	7604:IF2110_K01 7605:IF2110_K02
13:00	-	7605:IF2110_K02 7606:IF2180_K01	-	7606:IF2210_K01	7604:IF2180_K01
14:00	-	-	7605:IF2160_K01	7605:IF2110_K01	7607:IF2120_K02
15:00	7606:IF2180_K01	7604:IF2200_K01	7606:IF2110_K02	7605:IF2140_K01	7603:IF2120_K01
16:00	7605:IF2140_K01	7606:IF2200_K01	7603:IF2130_K01	-	7604:IF2150_K02 7605:IF2111_K01
17:00	7601:IF2180_K01	-	7604:IF2150_K01	7604:IF2140_K01	-

Search Duration: 175.9941 seconds
Stuck Frequency: 0

Final Objective Value: 0



2.3.5.3 Percobaan 3

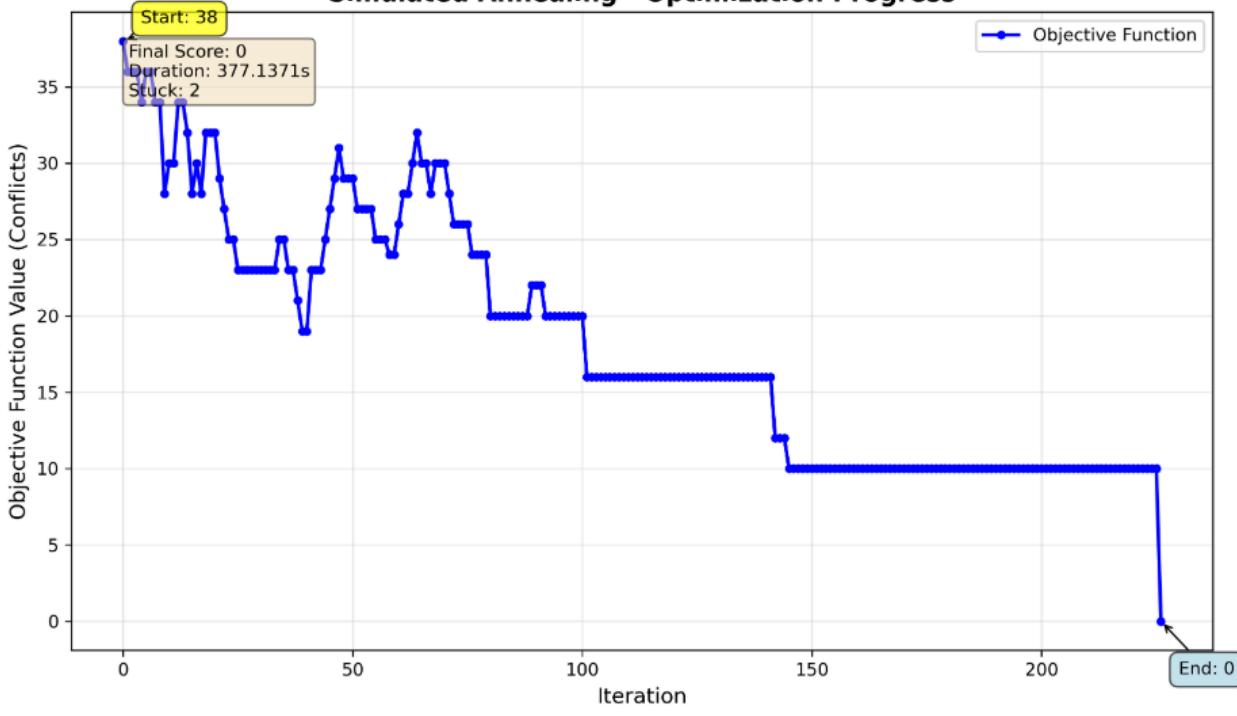
Simulated Annealing - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	-	7604:IF2180_K01	7606:IF2120_K02 7607:IF2180_K01 7608:IF2180_K01
8:00	7602:IF2111_K01	7602:IF2110_K02 7603:IF2190_K01	7604:IF2190_K01	-	-
9:00	7606:IF2120_K01	7606:IF2160_K01	-	7603:IF2140_K01 7605:IF2130_K01	-
10:00	-	-	-	7606:IF2110_K02	7608:IF2120_K01
11:00	7607:IF2120_K01 7608:IF2160_K01	-	-	-	-
12:00	7602:IF2200_K01	7603:IF2170_K01	-	7604:IF2120_K02	-
13:00	7607:IF2140_K01	7605:IF2210_K01	7601:IF2140_K01	7606:IF2110_K01	7601:IF2110_K01
14:00	7601:IF2200_K01 7605:IF2111_K01	7603:IF2200_K01 7605:IF2111_K01	-	7601:IF2110_K01	-
15:00	-	-	-	-	7603:IF2120_K02 7607:IF2210_K01 7608:IF2130_K01
16:00	7608:IF2150_K02	7607:IF2180_K01	7603:IF2150_K02 7607:IF2110_K02 7608:IF2110_K01	-	7603:IF2170_K01 7604:IF2160_K01 7607:IF2110_K02
17:00	7603:IF2150_K01 7605:IF2150_K01 7607:IF2130_K01	7607:IF2190_K01	-	7606:IF2140_K01	-

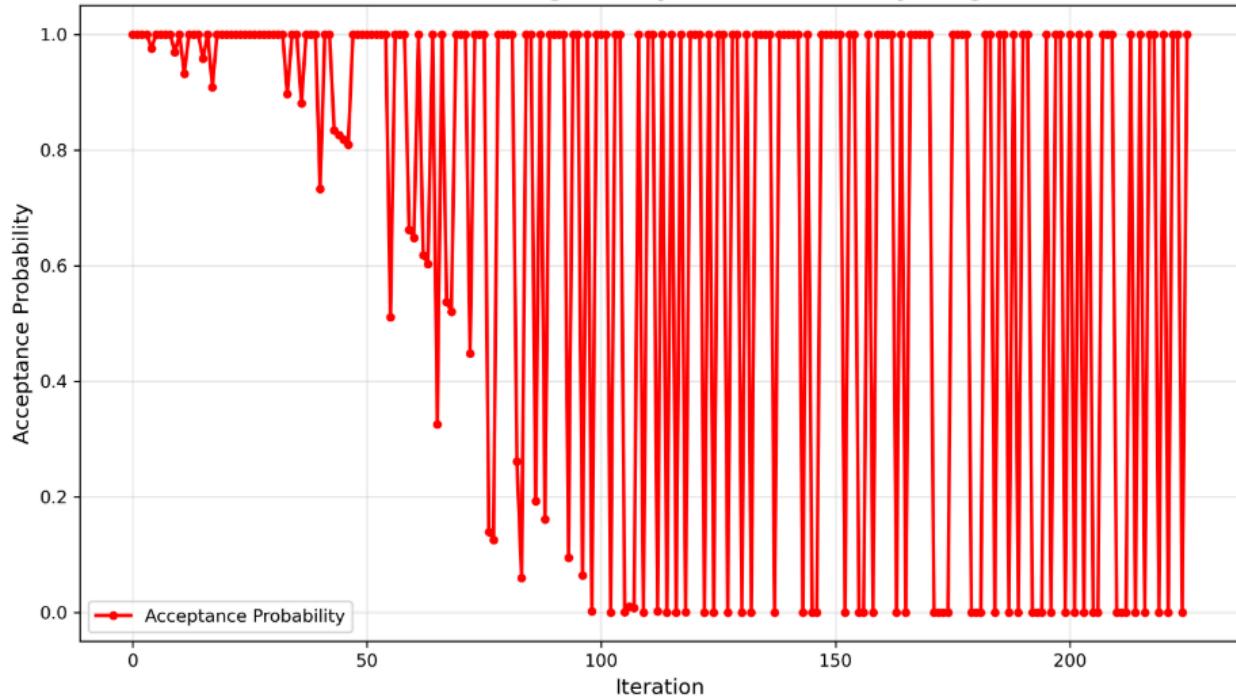
Simulated Annealing - Final Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7606:IF2140_K01	7604:IF2130_K01	-	7604:IF2120_K01	7608:IF2180_K01
8:00	7604:IF2200_K01	7604:IF2190_K01	7605:IF2120_K01 7606:IF2110_K02	7604:IF2210_K01	7605:IF2120_K01
9:00	7605:IF2200_K01	-	7603:IF2200_K01	7605:IF2160_K01	7603:IF2150_K02
10:00	-	-	-	-	7605:IF2110_K02 7606:IF2180_K01
11:00	7608:IF2160_K01	-	7606:IF2180_K01	7606:IF2110_K02	7603:IF2150_K01 7604:IF2130_K01
12:00	-	-	-	7605:IF2110_K01 7608:IF2170_K01	7606:IF2210_K01
13:00	7603:IF2150_K02	7603:IF2130_K01	7606:IF2180_K01	7606:IF2110_K01	-
14:00	-	7606:IF2160_K01	-	7606:IF2140_K01	7603:IF2120_K02 7604:IF2111_K01
15:00	7605:IF2140_K01	-	7605:IF2110_K01	7604:IF2111_K01	7606:IF2110_K02
16:00	-	7603:IF2120_K02	7604:IF2190_K01	7605:IF2120_K02	-
17:00	7601:IF2170_K01	-	7606:IF2110_K01	7604:IF2150_K01 7605:IF2190_K01	7606:IF2140_K01

Simulated Annealing - Optimization Progress



Simulated Annealing - Acceptance Probability Progress



Final Objective Value: 0

Search Duration: 377.1371 seconds
Stuck Frequency: 2

2.3.6 Hasil Genetic Algorithm

2.3.6.1 Variabel Kontrol: Populasi (50), mutation rate (1)

- a. Maksimum Iterasi: 50
 - i. Percobaan 1

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7603:IF2210_K01 7606:IF2110_K01 7607:IF2170_K01	-	7602:IF2210_K01	7606:IF2140_K01 7607:IF2110_K02
8:00	-	-	-	-	-
9:00	7601:IF2150_K01	7606:IF2190_K01	-	-	-
10:00	7603:IF2140_K01 7607:IF2150_K01	7602:IF2200_K01	-	7607:IF2111_K01	7604:IF2190_K01
11:00	-	-	7602:IF2120_K01 7604:IF2130_K01 7606:IF2170_K01	7608:IF2160_K01	7604:IF2140_K01
12:00	7601:IF2150_K02 7606:IF2130_K01	7606:IF2110_K02 7608:IF2120_K02	-	7604:IF2130_K01	7602:IF2120_K02 7606:IF2180_K01
13:00	7608:IF2200_K01	-	7605:IF2150_K02	7601:IF2130_K01 7602:IF2111_K01 7608:IF2140_K01	7603:IF2110_K01
14:00	7605:IF2110_K01	-	7606:IF2110_K02	7603:IF2120_K02	7608:IF2180_K01
15:00	-	7601:IF2160_K01	7606:IF2190_K01 7607:IF2180_K01	-	7605:IF2110_K01
16:00	-	-	7605:IF2200_K01	-	-
17:00	7602:IF2110_K02 7603:IF2120_K01	-	7604:IF2180_K01	7607:IF2160_K01	-

Genetic Algorithm - Final Best

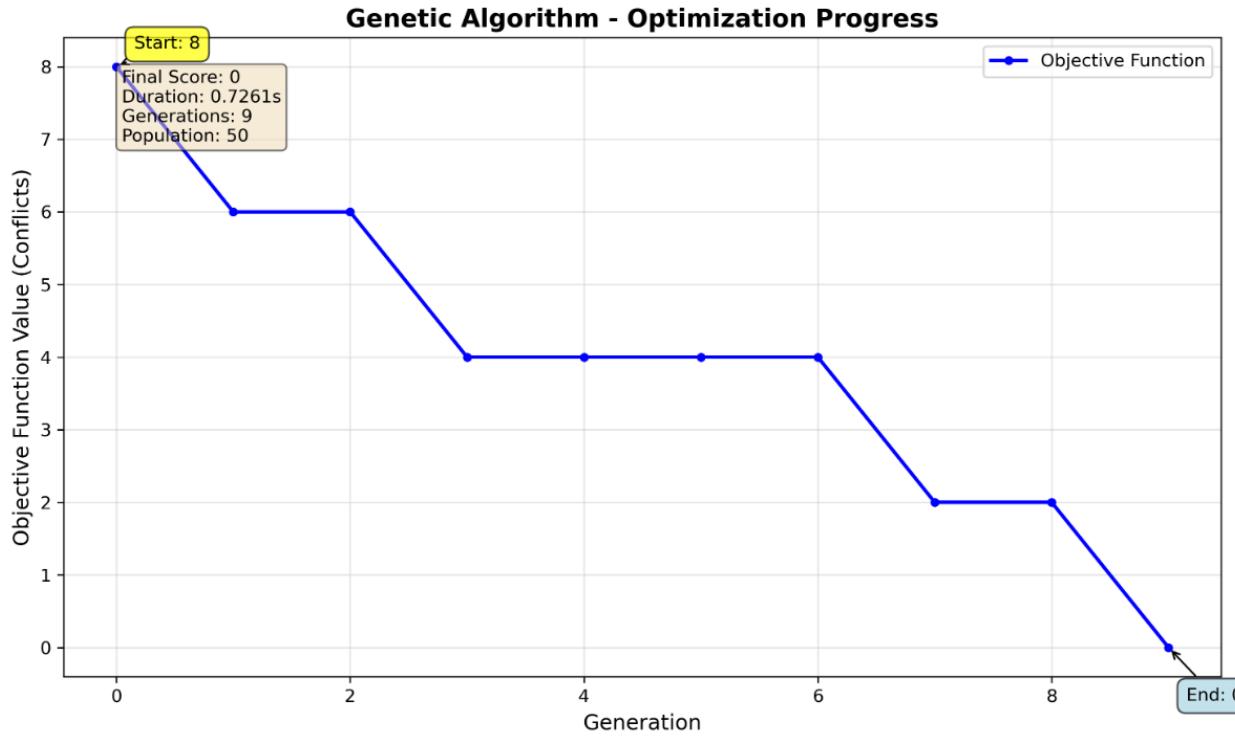
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7601:IF2120_K02	7606:IF2130_K01	-	7606:IF2200_K01	7608:IF2120_K01
8:00	-	7607:IF2111_K01	7602:IF2120_K01	-	7606:IF2150_K02
9:00	7601:IF2150_K01	7604:IF2140_K01	7605:IF2170_K01	7603:IF2120_K02	7608:IF2110_K02
10:00	7607:IF2150_K01	7605:IF2140_K01	7603:IF2130_K01	7601:IF2120_K01	7605:IF2180_K01
11:00	7602:IF2150_K02 7605:IF2110_K01	7602:IF2110_K01	7601:IF2180_K01	7606:IF2160_K01	7603:IF2120_K02
12:00	7602:IF2111_K01	7602:IF2110_K02	-	7602:IF2140_K01	-
13:00	7606:IF2180_K01	7603:IF2110_K01	7602:IF2180_K01	7605:IF2210_K01 7606:IF2200_K01	7607:IF2110_K02
14:00	-	7601:IF2110_K02	7602:IF2130_K01	7605:IF2140_K01	7608:IF2190_K01
15:00	-	7603:IF2190_K01	7605:IF2160_K01	7602:IF2190_K01	-
16:00	-	-	7605:IF2210_K01	-	7605:IF2160_K01 7606:IF2170_K01
17:00	-	7605:IF2110_K01	-	7608:IF2200_K01	-

Final Objective Value: 0

Generations Run: 9

Search Duration: 0.7261 seconds

Convergence History: [8, 6, 6, 4, 4, 4, 4, 2, 2, 0]...



ii. Percobaan 2

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	7601:IF2190_K01 7602:IF2160_K01 7603:IF2180_K01 7605:IF2130_K01	7603:IF2140_K01	7605:IF2210_K01
8:00	7606:IF2160_K01	-	-	7604:IF2111_K01	-
9:00	-	-	7601:IF2130_K01 7606:IF2150_K01	7602:IF2160_K01	-
10:00	7606:IF2200_K01	-	7606:IF2190_K01	7603:IF2140_K01	7601:IF2120_K02 7607:IF2120_K01
11:00	-	-	7601:IF2120_K01 7603:IF2170_K01 7604:IF2130_K01 7606:IF2150_K02	7602:IF2111_K01 7603:IF2150_K02	7601:IF2170_K01 7603:IF2170_K02 7604:IF2180_K01 7605:IF2190_K01
12:00	7603:IF2200_K01 7604:IF2120_K02	7602:IF2170_K01	-	-	7603:IF2110_K01
13:00	-	7604:IF2150_K01	-	-	-
14:00	7607:IF2180_K01	-	7601:IF2140_K01	7602:IF2110_K02	7607:IF2110_K01
15:00	-	7604:IF2110_K02	7604:IF2140_K01 7605:IF2210_K01 7608:IF2110_K02	-	-
16:00	7604:IF2200_K01	7602:IF2120_K01	-	-	-
17:00	-	7604:IF2180_K01	-	7602:IF2110_K01 7608:IF2120_K02	-

Genetic Algorithm - Final Best

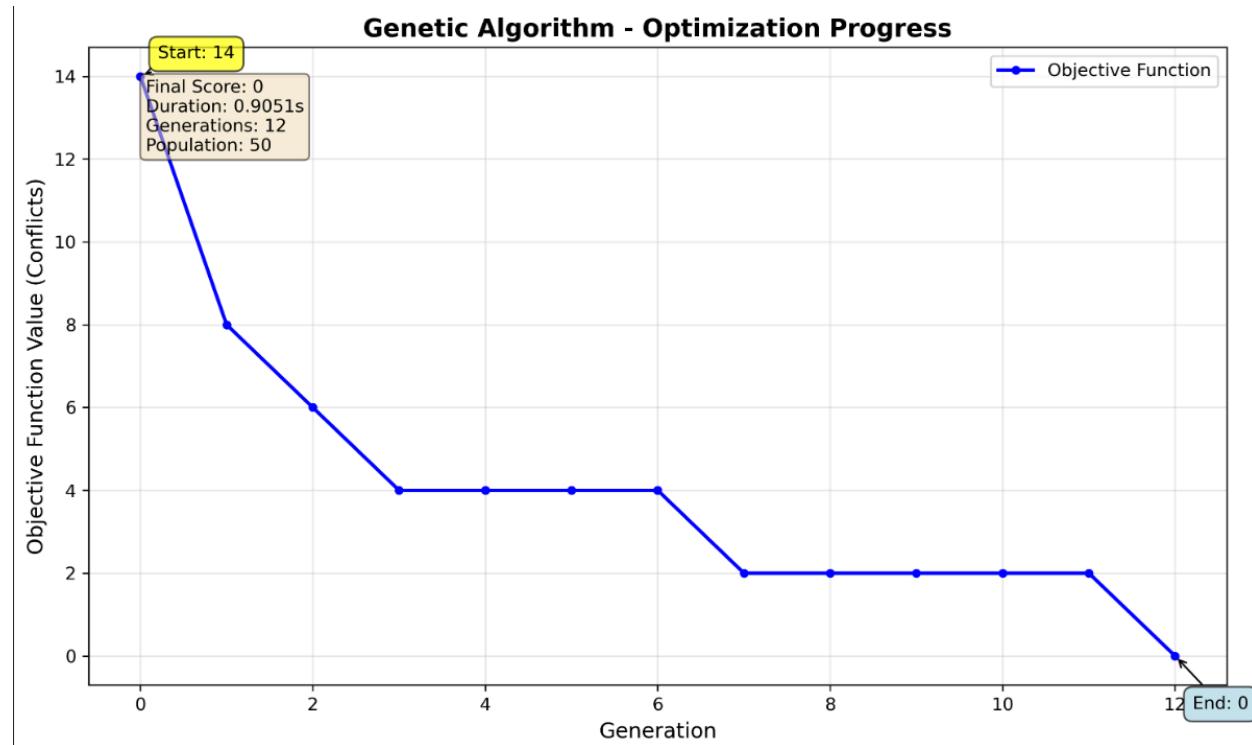
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	7602:IF2110_K01	-	7606:IF2200_K01
8:00	7605:IF2150_K02	-	7605:IF2111_K01	7606:IF2200_K01	7606:IF2130_K01
9:00	7604:IF2110_K01	7603:IF2110_K02	-	7603:IF2150_K01 7606:IF2130_K01	-
10:00	7602:IF2111_K01	7603:IF2180_K01	7606:IF2140_K01	7607:IF2190_K01	7602:IF2130_K01
11:00	-	7607:IF2160_K01	7606:IF2210_K01	7605:IF2140_K01	7604:IF2170_K01
12:00	7607:IF2110_K01	7607:IF2160_K01 7608:IF2120_K01	7606:IF2140_K01	7605:IF2200_K01	7604:IF2120_K02
13:00	-	7606:IF2140_K01	7602:IF2120_K02 7607:IF2120_K01	-	7608:IF2180_K01
14:00	7606:IF2170_K01	7606:IF2160_K01	7608:IF2110_K02	-	7603:IF2180_K01
15:00	7602:IF2110_K02	-	7603:IF2190_K01	7606:IF2120_K02	-
16:00	7602:IF2110_K02	7607:IF2120_K01	7605:IF2210_K01	7607:IF2180_K01	-
17:00	7601:IF2150_K02	7602:IF2150_K01	7605:IF2110_K01	7601:IF2190_K01	-

Final Objective Value: 0

Generations Run: 12

Search Duration: 0.9051 seconds

Convergence History: [14, 8, 6, 4, 4, 4, 4, 2, 2, 2]...



iii. Percobaan 3

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7604:IF2110_K01 7606:IF2200_K01	7607:IF2120_K01	7608:IF2110_K02	7603:IF2160_K01 7605:IF2200_K01
8:00	7603:IF2160_K01	-	7608:IF2130_K01	-	7604:IF2120_K01
9:00	7601:IF2170_K01 7607:IF2110_K01	7605:IF2140_K01 7606:IF2200_K01	7608:IF2140_K01	7604:IF2180_K01	-
10:00	-	7607:IF2150_K01	7603:IF2111_K01	-	7605:IF2110_K01
11:00	-	-	-	-	-
12:00	7608:IF2150_K02	7607:IF2110_K02 7608:IF2120_K02	7601:IF2180_K01 7602:IF2180_K01 7607:IF2110_K02	7603:IF2110_K01	7607:IF2210_K01
13:00	7601:IF2120_K02	7607:IF2210_K01	-	-	7608:IF2130_K01
14:00	-	7601:IF2190_K01 7606:IF2160_K01	-	7605:IF2190_K01	-
15:00	7606:IF2150_K02 7608:IF2170_K01	-	7607:IF2150_K01 7608:IF2140_K01	7608:IF2110_K02	-
16:00	-	7604:IF2120_K01 7607:IF2111_K01	7601:IF2120_K02	-	-
17:00	7608:IF2140_K01	7606:IF2130_K01	7607:IF2190_K01	7607:IF2180_K01	-

Genetic Algorithm - Final Best

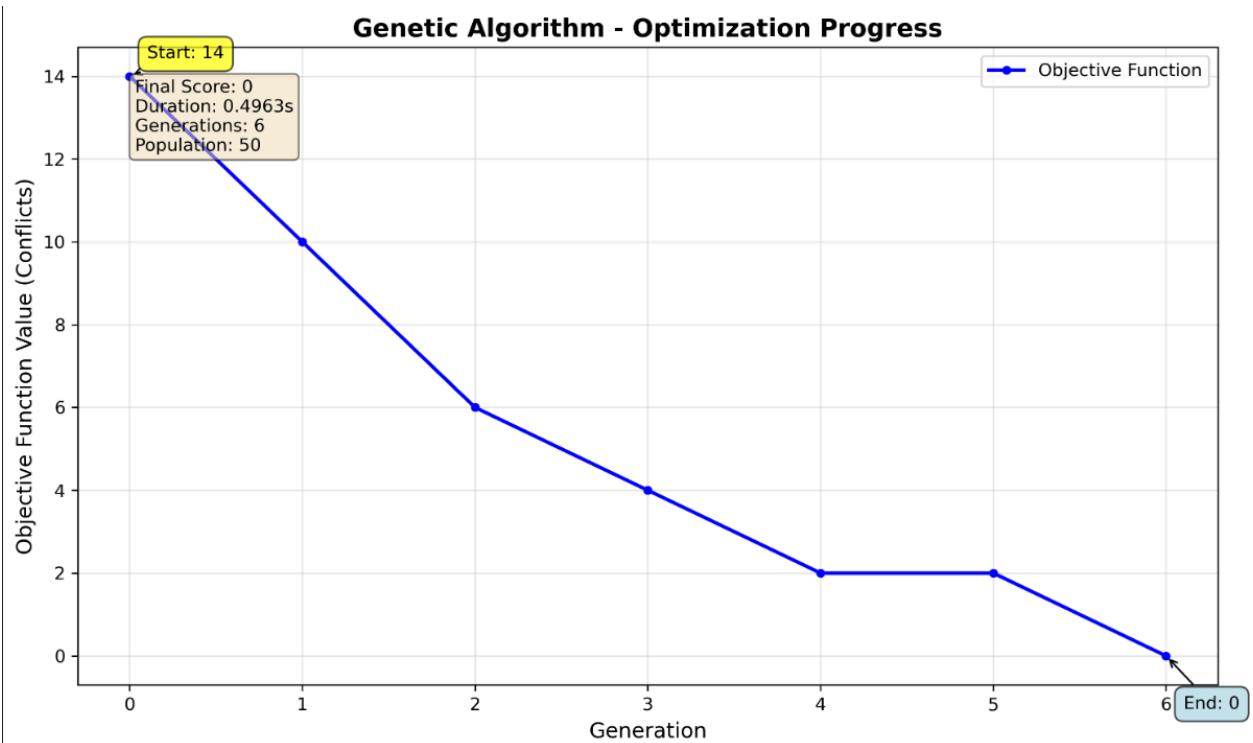
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7603:IF2200_K01	7604:IF2110_K01 7606:IF2200_K01	7607:IF2120_K01	-	7603:IF2160_K01 7605:IF2200_K01 7606:IF2180_K01
8:00	7603:IF2140_K01	-	7603:IF2130_K01	7608:IF2110_K02	7604:IF2120_K01
9:00	7607:IF2110_K01	7605:IF2140_K01	7608:IF2140_K01	-	-
10:00	-	7607:IF2150_K01	7603:IF2111_K01	-	7605:IF2110_K01
11:00	7607:IF2170_K01	-	-	7601:IF2180_K01	7604:IF2130_K01
12:00	7608:IF2150_K02	7608:IF2120_K02	7607:IF2110_K02	7603:IF2110_K01	7607:IF2210_K01
13:00	7601:IF2120_K02	7607:IF2210_K01	-	-	7604:IF2180_K01
14:00	7605:IF2180_K01	7601:IF2190_K01 7606:IF2160_K01 7608:IF2170_K01	-	7605:IF2190_K01	-
15:00	7606:IF2150_K02	7605:IF2140_K01	7607:IF2150_K01	7608:IF2110_K02	-
16:00	-	7604:IF2120_K01 7607:IF2111_K01	7601:IF2120_K02 7607:IF2190_K01	-	-
17:00	7608:IF2160_K01	7606:IF2130_K01	7604:IF2110_K02	-	-

Final Objective Value: 0

Generations Run: 6

Search Duration: 0.4963 seconds

Convergence History: [14, 10, 6, 4, 2, 2, 0]...

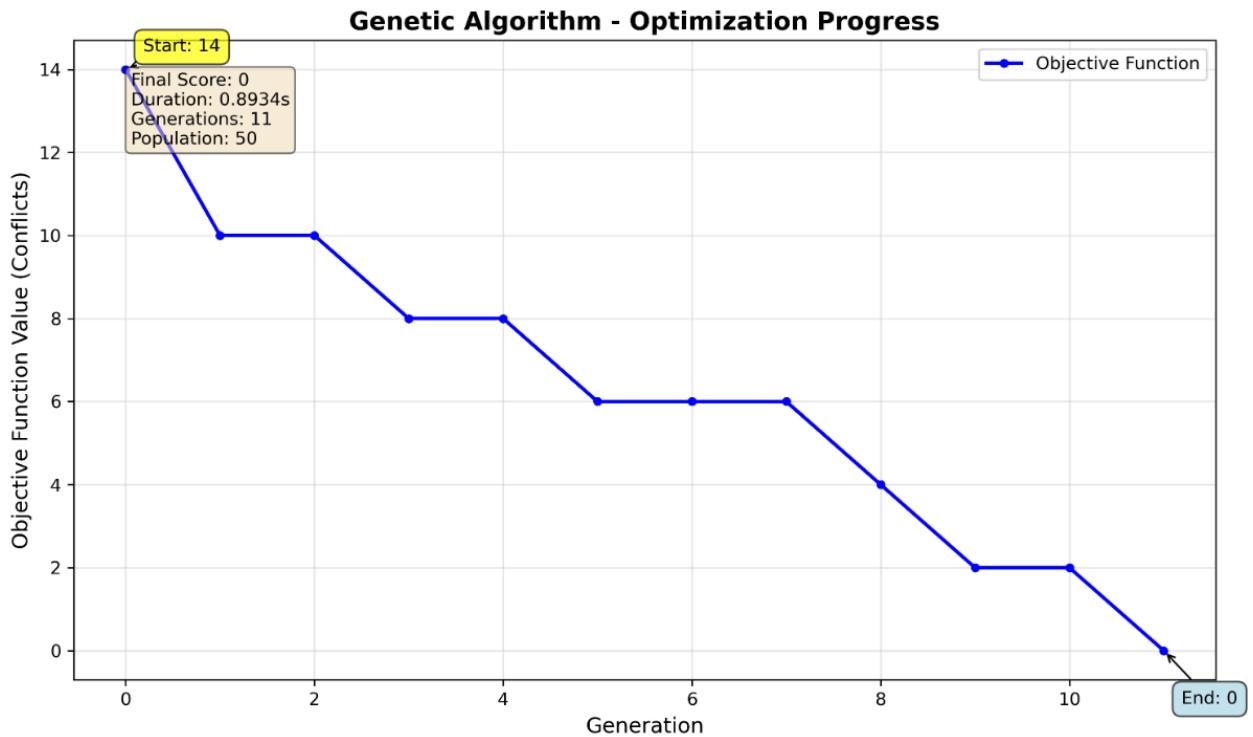


Final Objective Value: 0

Generations Run: 11

Search Duration: 0.8934 seconds

Convergence History: [14, 10, 10, 8, 8, 6, 6, 6, 4, 2]...



b. Maksimum Iterasi: 100

i. Percobaan 1

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7603:IF2180_K01	7603:IF2110_K01	-	-
8:00	7603:IF2120_K01	7608:IF2160_K01	-	7604:IF2140_K01	7604:IF2140_K01 7605:IF2150_K01
9:00	7602:IF2180_K01 7607:IF2190_K01	7605:IF2180_K01 7608:IF2110_K02	-	-	7604:IF2210_K01 7605:IF2180_K01 7607:IF2140_K01
10:00	7604:IF2130_K01	-	7601:IF2111_K01 7603:IF2150_K01	7605:IF2120_K02	7602:IF2110_K02
11:00	-	-	7603:IF2110_K01 7605:IF2110_K02 7607:IF2130_K01	-	-
12:00	-	-	-	-	7603:IF2190_K01
13:00	-	-	-	-	7604:IF2160_K01 7607:IF2190_K01 7608:IF2200_K01
14:00	7603:IF2160_K01	-	7602:IF2170_K01	7606:IF2110_K01	7602:IF2210_K01
15:00	-	7608:IF2130_K01	7603:IF2150_K02	-	7605:IF2140_K01
16:00	-	7602:IF2200_K01 7603:IF2120_K02	7606:IF2180_K01	7602:IF2111_K01 7606:IF2170_K01	7601:IF2150_K02 7607:IF2110_K01 7608:IF2120_K01
17:00	-	7607:IF2120_K02	-	7603:IF2110_K02 7605:IF2200_K01	-

Genetic Algorithm - Final Best

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7601:IF2170_K01 7604:IF2110_K01	-	7606:IF2200_K01	7605:IF2210_K01
8:00	-	-	-	7606:IF2160_K01	7605:IF2120_K01
9:00	7603:IF2110_K01	-	7604:IF2210_K01	7606:IF2140_K01	7604:IF2120_K02 7608:IF2180_K01
10:00	7605:IF2111_K01	7602:IF2180_K01 7603:IF2110_K02	7602:IF2140_K01	-	-
11:00	-	7605:IF2160_K01 7606:IF2200_K01	7603:IF2140_K01	7602:IF2130_K01 7608:IF2110_K02	-
12:00	7601:IF2190_K01	7601:IF2120_K02 7607:IF2150_K01	-	-	7603:IF2110_K01
13:00	7602:IF2200_K01	-	7603:IF2110_K01	7604:IF2130_K01	7604:IF2190_K01
14:00	7606:IF2180_K01	7606:IF2150_K02	-	7608:IF2120_K01	7603:IF2111_K01 7606:IF2120_K01
15:00	7601:IF2150_K02	-	-	7607:IF2120_K02	7607:IF2150_K01
16:00	7607:IF2140_K01	7602:IF2180_K01 7607:IF2110_K02	7605:IF2160_K01	7601:IF2170_K01	-
17:00	-	7608:IF2190_K01	7604:IF2110_K02 7608:IF2130_K01	-	-

ii. Percobaan 2

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7604:IF2140_K01	-	7607:IF2140_K01	-	-
8:00	-	-	7602:IF2150_K01	-	7607:IF2110_K01
9:00	-	-	7608:IF2120_K01	7603:IF2160_K01	7606:IF2120_K02
10:00	7607:IF2200_K01	7603:IF2180_K01	-	7606:IF2180_K01	-
11:00	7602:IF2140_K01 7608:IF2150_K01	-	-	7602:IF2200_K01 7603:IF2180_K01 7604:IF2150_K02	7604:IF2110_K02 7605:IF2110_K01
12:00	-	7604:IF2120_K02	-	7605:IF2110_K01	7602:IF2190_K01
13:00	7602:IF2170_K01 7607:IF2160_K01 7608:IF2130_K01	-	7606:IF2120_K02	7601:IF2130_K01	7603:IF2110_K02
14:00	-	7607:IF2110_K02	-	-	7603:IF2111_K01
15:00	-	-	7602:IF2140_K01 7603:IF2190_K01 7604:IF2160_K01 7605:IF2180_K01	7604:IF2130_K01 7608:IF2120_K01	-
16:00	-	7603:IF2210_K01	-	7608:IF2170_K01	7603:IF2210_K01
17:00	-	7605:IF2200_K01	7601:IF2110_K02	-	7601:IF2111_K01 7602:IF2120_K01 7604:IF2110_K01 7605:IF2190_K01

Genetic Algorithm - Final Best

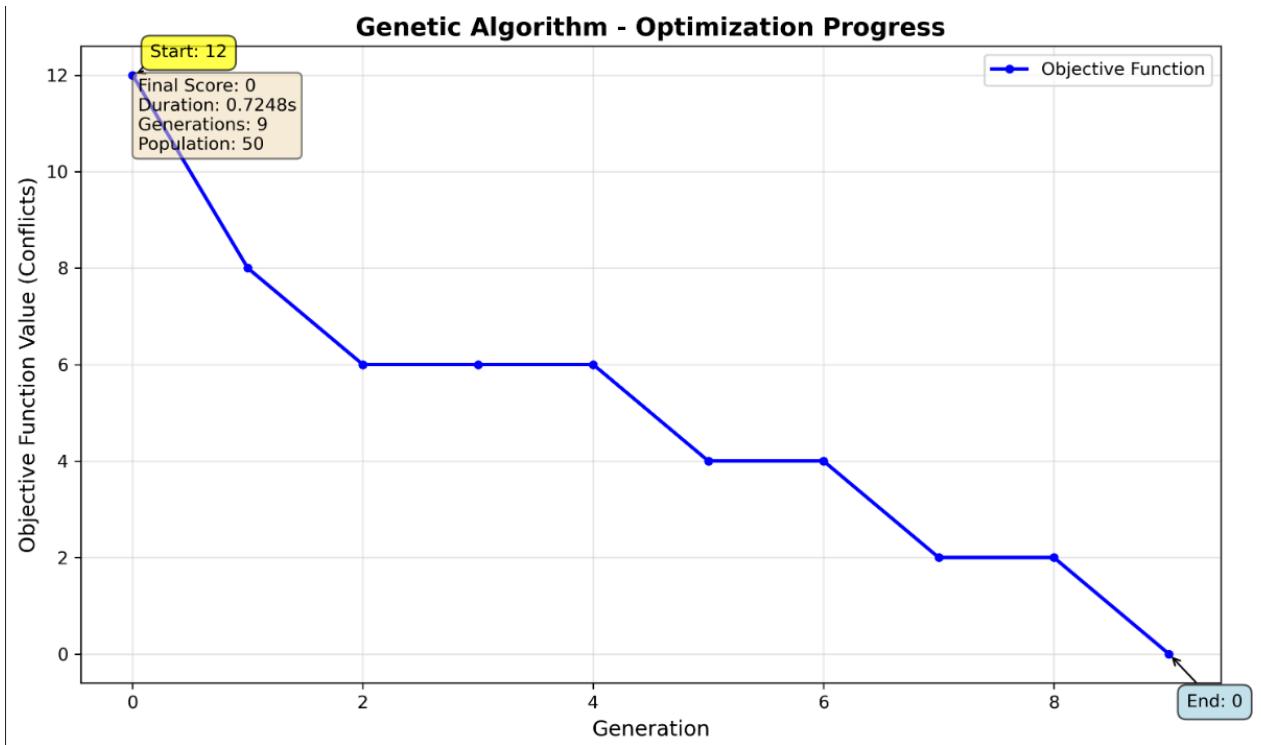
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7602:IF2140_K01	-	7605:IF2160_K01	7602:IF2150_K01 7605:IF2140_K01	7603:IF2120_K01 7604:IF2110_K02
8:00	-	-	-	-	-
9:00	7604:IF2120_K02	-	7603:IF2120_K02	7602:IF2200_K01 7603:IF2180_K01	7604:IF2170_K01
10:00	-	7601:IF2120_K01 7606:IF2120_K02	-	7602:IF2140_K01	7605:IF2110_K02
11:00	7601:IF2190_K01 7605:IF2160_K01	-	7606:IF2210_K01	-	-
12:00	7606:IF2180_K01 7608:IF2170_K01	7608:IF2110_K01	7607:IF2210_K01	-	-
13:00	7602:IF2111_K01	7608:IF2110_K02	7601:IF2110_K02	7605:IF2110_K01	7603:IF2180_K01 7605:IF2150_K02
14:00	7602:IF2110_K01 7603:IF2200_K01	7601:IF2130_K01	7603:IF2190_K01	7603:IF2190_K01	7602:IF2120_K01
15:00	-	7605:IF2140_K01	-	-	7605:IF2150_K01
16:00	-	7602:IF2200_K01 7603:IF2180_K01	-	7603:IF2130_K01	7605:IF2110_K01 7607:IF2111_K01
17:00	-	7607:IF2150_K02	-	7605:IF2130_K01	7606:IF2160_K01

Final Objective Value: 0

Generations Run: 9

Search Duration: 0.7248 seconds

Convergence History: [12, 8, 6, 6, 6, 4, 4, 2, 2, 0]...



iii. Percobaan 3

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7602:IF2150_K01	-	7603:IF2120_K01 7607:IF2170_K01	7606:IF2150_K02	-
8:00	7607:IF2111_K01	-	-	7603:IF2120_K02 7605:IF2110_K01	7604:IF2140_K01 7607:IF2110_K02
9:00	7601:IF2210_K01 7608:IF2110_K02	-	7607:IF2120_K02	-	7604:IF2140_K01
10:00	7605:IF2180_K01 7608:IF2140_K01	-	-	-	7602:IF2110_K01 7605:IF2130_K01
11:00	7607:IF2200_K01	7601:IF2200_K01	-	-	7603:IF2190_K01
12:00	7607:IF2130_K01	7601:IF2130_K01 7608:IF2110_K01	7606:IF2180_K01	-	-
13:00	7606:IF2170_K01	-	7602:IF2120_K02	7604:IF2210_K01 7606:IF2190_K01	-
14:00	-	-	7601:IF2180_K01	7608:IF2110_K02	-
15:00	-	-	-	7601:IF2110_K01 7605:IF2160_K01 7606:IF2190_K01	7608:IF2110_K02
16:00	-	7602:IF2150_K01	7607:IF2160_K01	-	7608:IF2160_K01
17:00	7602:IF2140_K01 7606:IF2120_K01	7602:IF2150_K02	7604:IF2180_K01	7605:IF2111_K01	7603:IF2120_K01 7607:IF2200_K01

Genetic Algorithm - Final Best

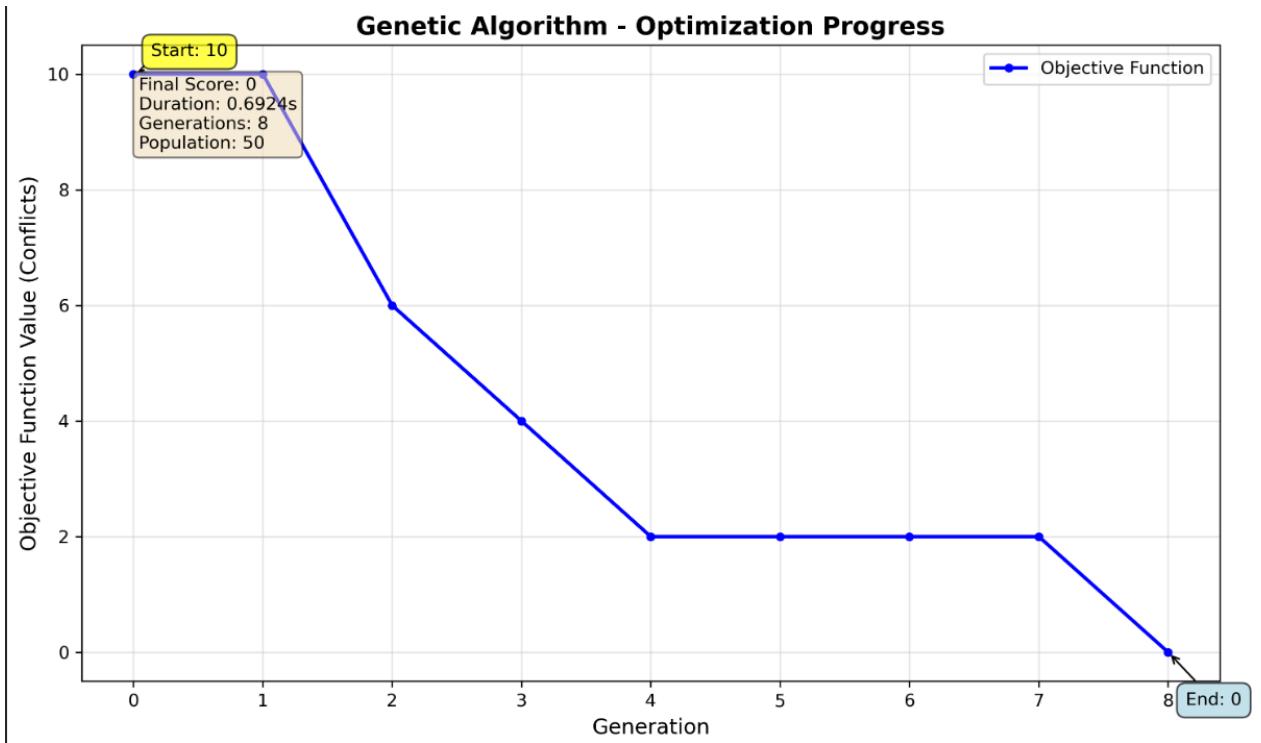
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7604:IF2160_K01	-	7601:IF2140_K01	-	-
8:00	7605:IF2180_K01 7608:IF2120_K02	-	7607:IF2190_K01	7604:IF2120_K02	7604:IF2110_K02
9:00	7607:IF2110_K01	-	7601:IF2200_K01 7606:IF2180_K01	7602:IF2130_K01 7604:IF2110_K02	7601:IF2150_K02 7606:IF2120_K01
10:00	7606:IF2170_K01	7602:IF2150_K02	-	7603:IF2180_K01	-
11:00	7602:IF2110_K01 7605:IF2200_K01	-	7607:IF2110_K02	7601:IF2140_K01	7607:IF2120_K01
12:00	7603:IF2190_K01 7604:IF2160_K01	7605:IF2110_K02	7604:IF2130_K01	-	-
13:00	-	7604:IF2140_K01	-	7604:IF2150_K01	-
14:00	7607:IF2190_K01	7603:IF2120_K01	7606:IF2170_K01	-	7606:IF2160_K01
15:00	7603:IF2120_K02 7607:IF2111_K01	-	7602:IF2140_K01	7601:IF2111_K01	7607:IF2110_K01
16:00	7603:IF2110_K01	7606:IF2210_K01	-	7608:IF2180_K01	-
17:00	-	-	7601:IF2150_K01 7602:IF2130_K01	7601:IF2200_K01	7601:IF2110_K01

Final Objective Value: 0

Generations Run: 8

Search Duration: 0.6924 seconds

Convergence History: [10, 10, 6, 4, 2, 2, 2, 2, 0]...



c. Maksimum Iterasi: 200

i. Percobaan 1

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7604:IF2190_K01 7605:IF2200_K01 7606:IF2160_K01	7604:IF2110_K01 7606:IF2110_K02	7607:IF2160_K01	-	7604:IF2180_K01
8:00	-	-	7608:IF2190_K01	-	7603:IF2160_K01 7606:IF2180_K01
9:00	7606:IF2200_K01	7605:IF2110_K02	7605:IF2110_K01	7602:IF2130_K01	7604:IF2130_K01
10:00	7608:IF2180_K01	7605:IF2120_K02	-	7606:IF2110_K02	-
11:00	-	7604:IF2150_K02 7605:IF2120_K01	7608:IF2170_K01	-	-
12:00	-	-	-	7602:IF2210_K01 7606:IF2120_K02 7608:IF2111_K01	7607:IF2150_K01
13:00	-	-	7605:IF2170_K01 7608:IF2180_K01	7605:IF2190_K01	7608:IF2140_K01
14:00	7601:IF2200_K01	-	7604:IF2210_K01 7605:IF2110_K02	7608:IF2150_K01	-
15:00	-	-	-	-	-
16:00	7607:IF2140_K01	7602:IF2140_K01	7603:IF2120_K01	7601:IF2110_K01 7602:IF2111_K01	7601:IF2120_K02 7602:IF2120_K01
17:00	7601:IF2110_K01	-	7601:IF2150_K02 7605:IF2140_K01	7604:IF2130_K01	-

Genetic Algorithm - Final Best

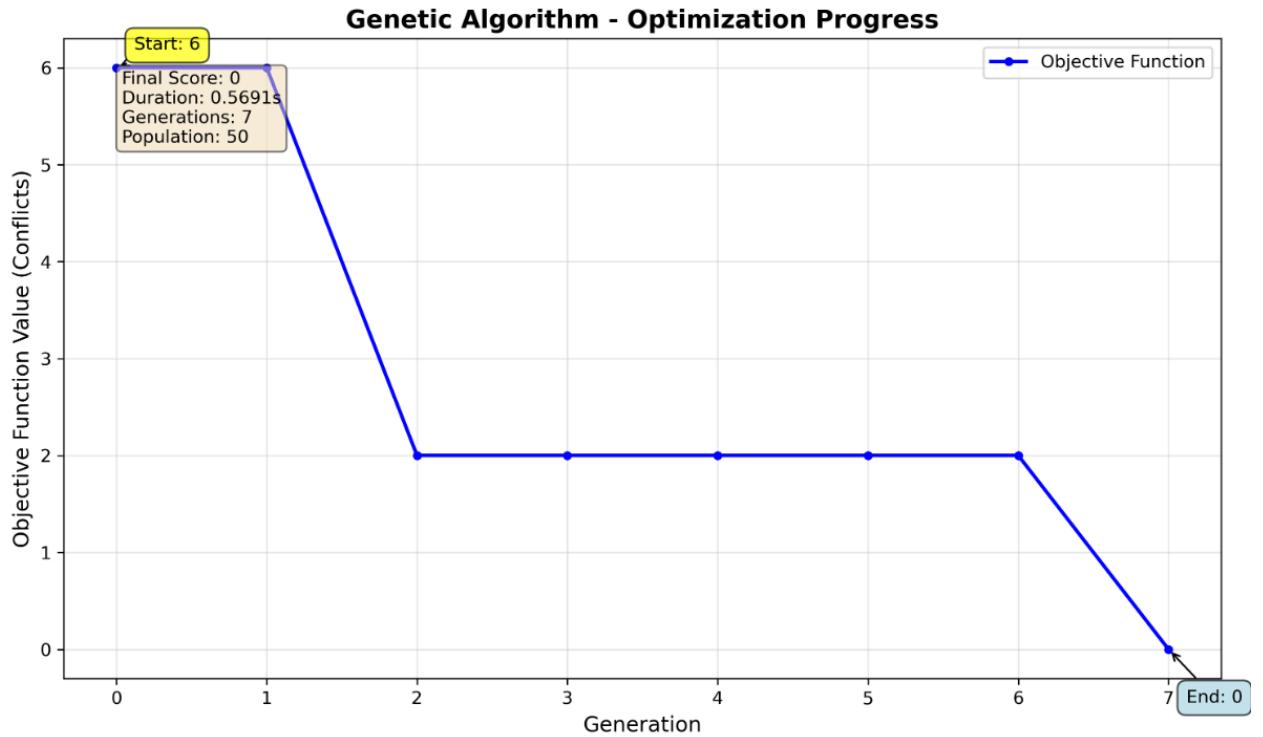
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7606:IF2160_K01	7604:IF2120_K01 7606:IF2110_K02	7606:IF2200_K01 7607:IF2160_K01	-	-
8:00	7606:IF2160_K01	7605:IF2210_K01	-	-	7606:IF2200_K01 7607:IF2190_K01
9:00	7604:IF2110_K01	7605:IF2110_K02	7605:IF2110_K01	7602:IF2130_K01	7604:IF2130_K01
10:00	7608:IF2190_K01	7605:IF2120_K02	-	7606:IF2110_K02	7603:IF2180_K01
11:00	7602:IF2180_K01	7604:IF2150_K02 7605:IF2120_K01	7604:IF2170_K01	-	7603:IF2180_K01
12:00	-	-	-	7606:IF2120_K02 7608:IF2111_K01	7607:IF2150_K01
13:00	-	-	7605:IF2180_K01	-	7608:IF2140_K01
14:00	-	7604:IF2200_K01	7605:IF2110_K02	7608:IF2150_K01	7608:IF2210_K01
15:00	7601:IF2170_K01	-	-	-	-
16:00	7607:IF2140_K01	7602:IF2140_K01	7603:IF2120_K01	7601:IF2110_K01 7602:IF2111_K01	7601:IF2120_K02 7602:IF2110_K01
17:00	-	-	7601:IF2150_K02 7605:IF2140_K01	7604:IF2130_K01	7604:IF2190_K01

Final Objective Value: 0

Generations Run: 7

Search Duration: 0.5691 seconds

Convergence History: [6, 6, 2, 2, 2, 2, 2, 0]...



ii. Percobaan 2

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7603:IF2140_K01	-	7603:IF2180_K01	7606:IF2170_K01	7604:IF2170_K01
8:00	-	7605:IF2150_K02 7608:IF2160_K01	-	7602:IF2110_K01	7603:IF2110_K01 7602:IF2180_K01 7605:IF2140_K01 7608:IF2120_K01
9:00	7604:IF2200_K01	7602:IF2120_K02 7603:IF2110_K01 7604:IF2160_K01	-	-	7603:IF2180_K01 7604:IF2110_K01 7605:IF2120_K02
10:00	-	7607:IF2130_K01	7602:IF2120_K01 7606:IF2120_K01	-	7602:IF2190_K01
11:00	-	7603:IF2180_K01 7607:IF2110_K02	7607:IF2111_K01 7608:IF2110_K02	7608:IF2110_K02	7604:IF2200_K01
12:00	-	7601:IF2190_K01 7608:IF2210_K01	7606:IF2190_K01	-	-
13:00	-	-	7607:IF2140_K01	7602:IF2160_K01	7602:IF2130_K01
14:00	-	7605:IF2120_K02	-	7602:IF2150_K02 7605:IF2200_K01 7606:IF2110_K01	7605:IF2111_K01
15:00	-	7605:IF2130_K01	7606:IF2150_K01	-	-
16:00	7606:IF2140_K01	-	-	-	-
17:00	-	7604:IF2110_K02	-	-	7608:IF2150_K01

Genetic Algorithm - Final Best

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7606:IF2160_K01	7602:IF2110_K02	7604:IF2200_K01 7605:IF2160_K01	7605:IF2170_K01 7606:IF2180_K01
8:00	7606:IF2111_K01	7601:IF2110_K01 7608:IF2110_K02	-	7601:IF2150_K01 7607:IF2130_K01	7607:IF2120_K02
9:00	7603:IF2130_K01	7601:IF2150_K02	7602:IF2210_K01 7606:IF2200_K01	-	-
10:00	-	7606:IF2140_K01	-	7604:IF2180_K01 7607:IF2200_K01	-
11:00	-	7607:IF2180_K01	7603:IF2160_K01 7607:IF2190_K01	7602:IF2140_K01	7606:IF2150_K01
12:00	7602:IF2110_K01	7601:IF2111_K01	7607:IF2120_K02	7608:IF2110_K01	7602:IF2140_K01
13:00	7607:IF2210_K01	-	7605:IF2190_K01	-	-
14:00	7601:IF2110_K02	-	7601:IF2120_K02	-	7607:IF2120_K01
15:00	7603:IF2170_K01 7606:IF2180_K01	-	7608:IF2190_K01	7608:IF2150_K02	-
16:00	7603:IF2110_K01	-	7603:IF2130_K01	-	7602:IF2120_K01
17:00	7606:IF2120_K01 7607:IF2110_K02	-	-	-	7606:IF2140_K01

Final Objective Value: 0

Generations Run: 11

Search Duration: 0.9347 seconds

Convergence History: [14, 8, 8, 4, 4, 4, 4, 4, 4, 2]...

iii. Percobaan 3

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7605:IF2200_K01	-	7606:IF2110_K02 7607:IF2140_K01	7602:IF2180_K01	-
8:00	-	-	7606:IF2170_K01	-	-
9:00	7606:IF2120_K02 7608:IF2160_K01	7604:IF2111_K01	-	7607:IF2190_K01	-
10:00	-	7601:IF2150_K02	-	-	7602:IF2150_K01
11:00	-	7606:IF2160_K01	7607:IF2120_K01	-	7601:IF2180_K01 7602:IF2180_K01
12:00	7601:IF2110_K01 7602:IF2190_K01	7608:IF2160_K01	-	7603:IF2190_K01 7606:IF2210_K01	-
13:00	7601:IF2120_K02 7608:IF2130_K01	7607:IF2130_K01	-	7601:IF2120_K01	7607:IF2140_K01
14:00	7605:IF2110_K01	7602:IF2110_K02	7605:IF2140_K01 7607:IF2200_K01	7602:IF2110_K02 7604:IF2180_K01	7604:IF2150_K02
15:00	-	7607:IF2120_K02 7608:IF2210_K01	7607:IF2200_K01	7608:IF2130_K01	-
16:00	-	7605:IF2170_K01	-	7607:IF2110_K02	-
17:00	-	7601:IF2120_K01 7603:IF2110_K01 7606:IF2111_K01	7604:IF2150_K01 7606:IF2140_K01	7606:IF2110_K01	-

Genetic Algorithm - Final Best

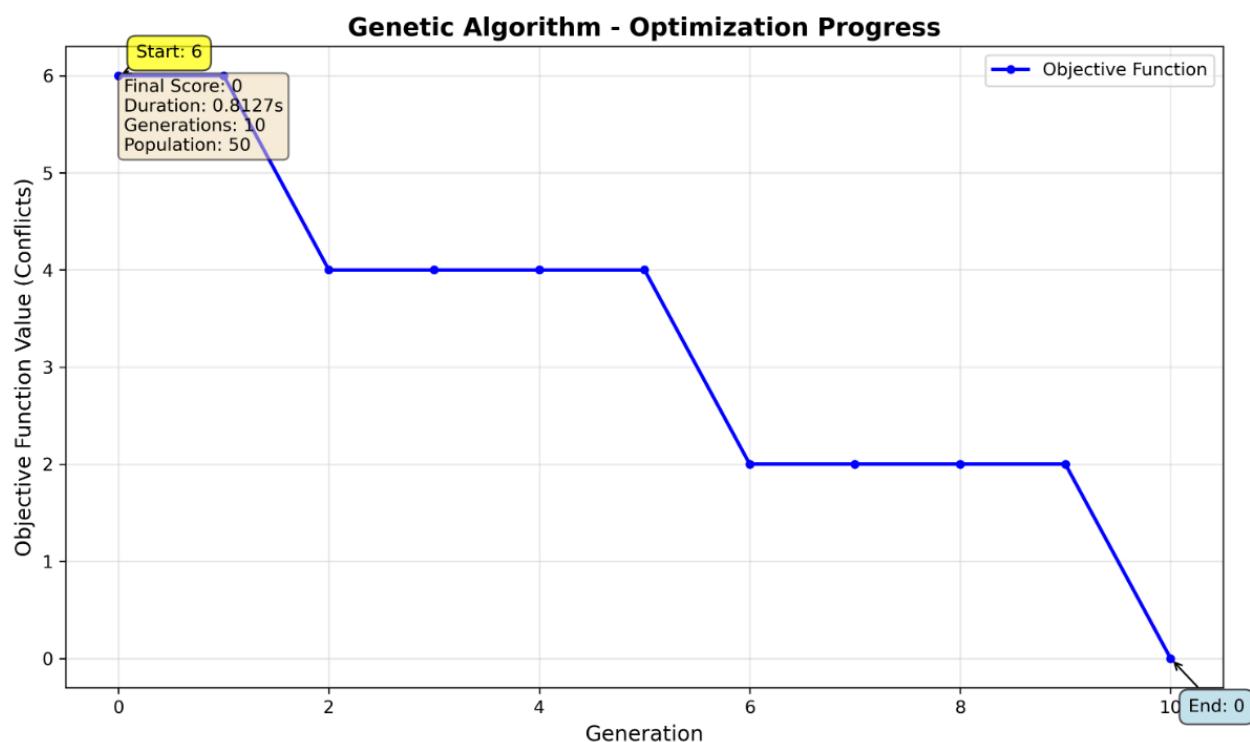
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7605:IF2150_K02	-	-	7603:IF2110_K02	-
8:00	7601:IF2110_K02	7606:IF2160_K01	7606:IF2120_K02	-	7604:IF2120_K02
9:00	7607:IF2150_K01	7607:IF2120_K01	-	7603:IF2180_K01	7601:IF2190_K01
10:00	7602:IF2140_K01	7603:IF2200_K01	7602:IF2180_K01	7608:IF2130_K01	7605:IF2170_K01
11:00	7608:IF2111_K01	7606:IF2110_K01	-	7604:IF2110_K02 7607:IF2130_K01	7608:IF2140_K01
12:00	7606:IF2190_K01	7604:IF2200_K01	7602:IF2140_K01	-	-
13:00	7601:IF2110_K01 7605:IF2111_K01	-	-	7607:IF2210_K01	-
14:00	7602:IF2200_K01	-	7607:IF2110_K01	7605:IF2120_K01	-
15:00	7605:IF2180_K01	7605:IF2210_K01 7606:IF2150_K02	-	-	7605:IF2110_K02 7606:IF2180_K01
16:00	-	7605:IF2140_K01	7602:IF2170_K01 7608:IF2160_K01	7602:IF2120_K02 7605:IF2120_K01	-
17:00	7605:IF2160_K01 7606:IF2190_K01	7602:IF2130_K01	7607:IF2110_K01	7608:IF2150_K01	-

Final Objective Value: 0

Generations Run: 10

Search Duration: 0.8127 seconds

Convergence History: [6, 6, 4, 4, 4, 4, 2, 2, 2, 2]...



2.3.6.2 Variabel Kontrol: Maksimum Iterasi (100), mutation rate (1)

- Populasi: 20
 - Percobaan 1

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	7601:IF2190_K01 7607:IF2120_K02	-	7604:IF2170_K01
8:00	7603:IF2110_K02 7606:IF2200_K01	7605:IF2110_K02	7607:IF2160_K01	-	-
9:00	-	-	-	7602:IF2190_K01	7601:IF2140_K01 7604:IF2120_K01 7605:IF2180_K01
10:00	-	7605:IF2130_K01 7608:IF2110_K01	-	-	7604:IF2150_K02
11:00	7603:IF2111_K01	7602:IF2130_K01 7607:IF2210_K01	7602:IF2190_K01 7603:IF2150_K01 7604:IF2110_K02	-	7602:IF2160_K01 7606:IF2180_K01 7608:IF2120_K01
12:00	-	7608:IF2140_K01	7608:IF2110_K01	7603:IF2200_K01	7606:IF2110_K02 7607:IF2110_K01
13:00	-	-	7605:IF2210_K01	7607:IF2180_K01	-
14:00	7604:IF2150_K02	7602:IF2111_K01	-	-	7606:IF2140_K01
15:00	-	7608:IF2200_K01	7607:IF2120_K02	7601:IF2150_K01 7607:IF2120_K01	-
16:00	-	7605:IF2120_K02 7607:IF2140_K01	7601:IF2110_K01	7607:IF2160_K01	-
17:00	7603:IF2130_K01	7606:IF2170_K01	-	-	7607:IF2180_K01

Genetic Algorithm - Final Best

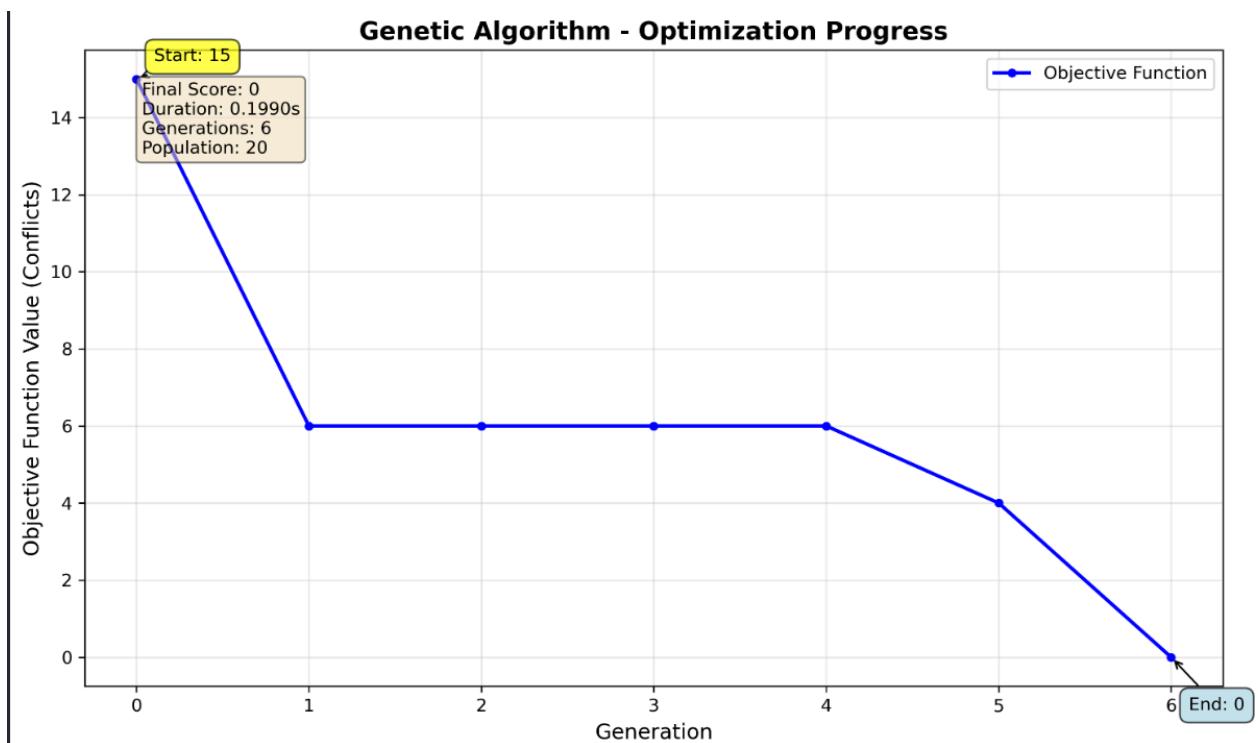
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7606:IF2120_K02	7603:IF2170_K01	-	7605:IF2120_K01	-
8:00	-	-	-	-	7607:IF2190_K01
9:00	-	7602:IF2130_K01	7605:IF2110_K01	-	7607:IF2210_K01
10:00	7608:IF2140_K01	7603:IF2110_K02	7605:IF2150_K01	7607:IF2150_K02	7606:IF2120_K01
11:00	7602:IF2130_K01	-	-	-	-
12:00	7601:IF2200_K01 7604:IF2110_K01	-	7603:IF2160_K01	7601:IF2140_K01	7607:IF2150_K01
13:00	7606:IF2190_K01	-	7606:IF2190_K01	-	7602:IF2150_K02 7606:IF2180_K01
14:00	-	7604:IF2110_K01	7608:IF2180_K01	7606:IF2140_K01	-
15:00	7606:IF2111_K01	7607:IF2160_K01	7607:IF2130_K01 7608:IF2110_K02	7602:IF2110_K02	7603:IF2210_K01 7604:IF2200_K01
16:00	7606:IF2180_K01	7602:IF2120_K02 7608:IF2111_K01	7601:IF2110_K01 7605:IF2170_K01	7606:IF2140_K01	7602:IF2120_K01 7608:IF2160_K01
17:00	7605:IF2180_K01 7607:IF2120_K02	7603:IF2200_K01	7602:IF2110_K02	-	-

Final Objective Value: 0

Generations Run: 6

Search Duration: 0.1990 seconds

Convergence History: [15, 6, 6, 6, 6, 4, 0]...



ii. Percobaan 2

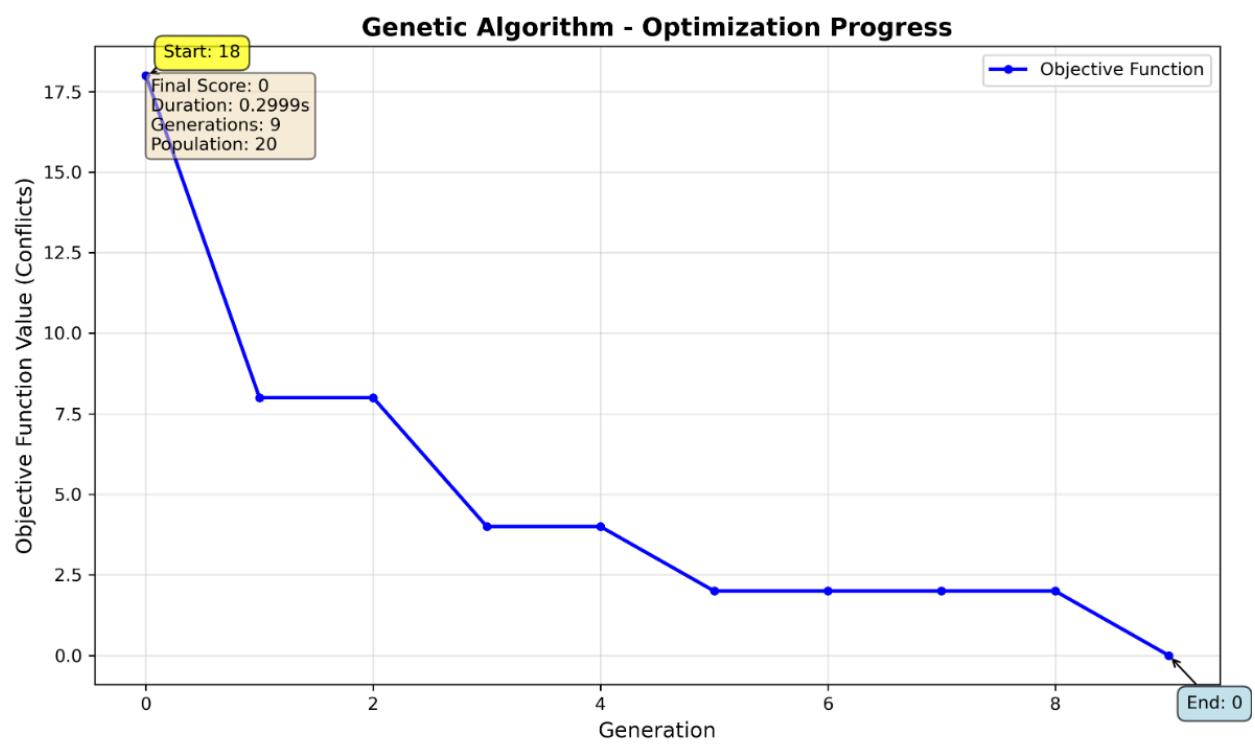
Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7608:IF2150_K01	-	7603:IF2130_K01	-
8:00	7602:IF2120_K02	7602:IF2110_K02 7605:IF2170_K01	7606:IF2110_K02	-	7608:IF2110_K02
9:00	-	7605:IF2180_K01	-	7601:IF2110_K01 7603:IF2200_K01 7605:IF2200_K01	7605:IF2160_K01
10:00	-	7602:IF2140_K01	7601:IF2160_K01 7604:IF2180_K01 7605:IF2120_K01	7602:IF2130_K01 7604:IF2150_K02	7601:IF2120_K02 7603:IF2150_K01
11:00	7605:IF2180_K01	7607:IF2111_K01 7608:IF2190_K01	7608:IF2170_K01	-	-
12:00	7607:IF2110_K01	-	-	7608:IF2120_K02	7605:IF2210_K01
13:00	-	7603:IF2150_K02	7605:IF2110_K01	-	-
14:00	7601:IF2140_K01	-	-	-	-
15:00	7606:IF2110_K02	7602:IF2160_K01 7604:IF2120_K01	7605:IF2180_K01	-	7601:IF2140_K01
16:00	-	7607:IF2110_K01	7608:IF2210_K01	7601:IF2130_K01	7608:IF2140_K01
17:00	-	7603:IF2200_K01 7607:IF2190_K01	7605:IF2111_K01	7603:IF2120_K01	7602:IF2190_K01

Genetic Algorithm - Final Best

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	-	7603:IF2130_K01	-
8:00	7602:IF2120_K02 7605:IF2110_K01	7602:IF2110_K02	7606:IF2110_K02	-	7608:IF2110_K02
9:00	-	7605:IF2180_K01	-	7601:IF2110_K01 7603:IF2200_K01	7605:IF2160_K01
10:00	7607:IF2170_K01	7602:IF2140_K01	7601:IF2160_K01 7605:IF2120_K01	-	7601:IF2120_K02 7603:IF2150_K01
11:00	7605:IF2180_K01	7607:IF2111_K01 7608:IF2190_K01	7604:IF2180_K01 7608:IF2170_K01	-	-
12:00	7607:IF2110_K01	-	7605:IF2120_K01	7608:IF2120_K02	7605:IF2210_K01
13:00	7606:IF2111_K01	-	-	-	-
14:00	7601:IF2140_K01	-	-	7601:IF2150_K02 7608:IF2150_K01	7604:IF2150_K02
15:00	7606:IF2110_K02	7602:IF2160_K01 7604:IF2120_K01	7605:IF2180_K01	7603:IF2130_K01	7601:IF2140_K01
16:00	-	7607:IF2110_K01	-	7601:IF2130_K01	7608:IF2140_K01
17:00	7604:IF2200_K01	7603:IF2200_K01 7607:IF2190_K01	-	7603:IF2210_K01	7602:IF2190_K01

```
Final Objective Value: 0  
Generations Run: 9  
Search Duration: 0.2999 seconds  
Convergence History: [18, 8, 8, 4, 4, 2, 2, 2, 2, 0]...
```



iii. Percobaan 3

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7607:IF2110_K01	7602:IF2120_K01 7605:IF2130_K01	-	-	7603:IF2110_K02
8:00	7602:IF2130_K01 7603:IF2180_K01	7603:IF2140_K01	7608:IF2170_K01	7606:IF2110_K02	-
9:00	7602:IF2160_K01 7604:IF2120_K02	-	-	7608:IF2150_K02	7604:IF2190_K01 7605:IF2210_K01
10:00	-	7604:IF2170_K01 7608:IF2160_K01	-	-	-
11:00	-	-	7606:IF2111_K01	7604:IF2110_K01	-
12:00	7604:IF2110_K01	7607:IF2120_K01 7608:IF2190_K01	7604:IF2150_K01	7606:IF2111_K01	-
13:00	7602:IF2140_K01	-	-	-	-
14:00	7602:IF2120_K02 7603:IF2180_K01	7601:IF2180_K01 7607:IF2180_K01	-	-	-
15:00	7603:IF2110_K01	-	-	7604:IF2140_K01	7604:IF2110_K02 7607:IF2200_K01
16:00	-	7602:IF2190_K01 7605:IF2140_K01 7608:IF2120_K01	7601:IF2200_K01 7602:IF2150_K01 7606:IF2160_K01	7607:IF2130_K01	7604:IF2110_K02 7607:IF2210_K01
17:00	-	7606:IF2150_K02	-	-	7604:IF2120_K02 7607:IF2200_K01

Genetic Algorithm - Final Best

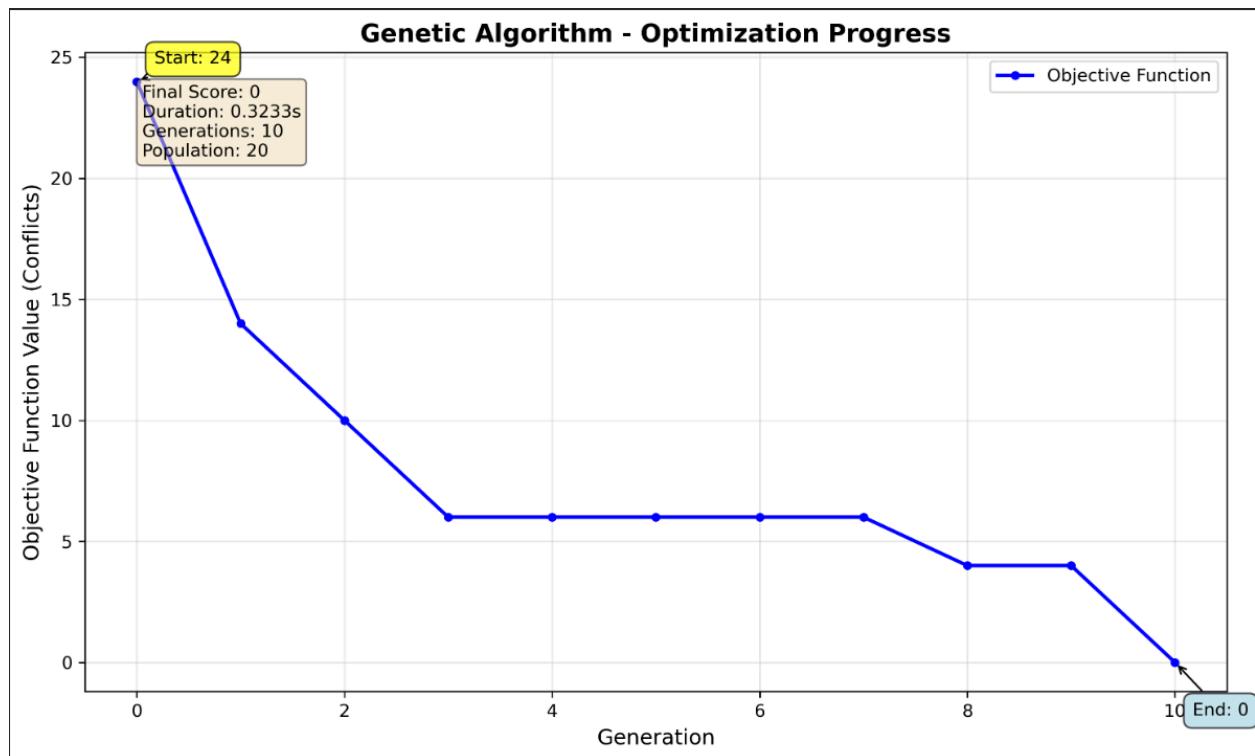
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	-	-	7605:IF2170_K01
8:00	-	7608:IF2140_K01	-	7601:IF2150_K02	7603:IF2200_K01
9:00	7605:IF2111_K01	7606:IF2150_K01	7604:IF2210_K01	7603:IF2110_K01	7602:IF2180_K01
10:00	-	7602:IF2110_K02	7602:IF2140_K01	7606:IF2190_K01	7601:IF2120_K01
11:00	7602:IF2150_K02	7603:IF2180_K01	-	7605:IF2110_K01	7606:IF2110_K01
12:00	7605:IF2170_K01	-	7606:IF2120_K02 7608:IF2150_K01	7601:IF2160_K01 7606:IF2200_K01	7607:IF2110_K01
13:00	-	7608:IF2190_K01	7604:IF2110_K02	-	7603:IF2130_K01
14:00	7603:IF2120_K02	7605:IF2120_K01	7603:IF2111_K01 7607:IF2120_K02	7607:IF2210_K01	7601:IF2200_K01
15:00	7603:IF2190_K01	7607:IF2180_K01 7608:IF2110_K02	7607:IF2130_K01	7605:IF2120_K01	7605:IF2110_K02
16:00	7608:IF2160_K01	7602:IF2140_K01	-	-	-
17:00	-	7603:IF2130_K01	7606:IF2160_K01	7607:IF2140_K01	7606:IF2180_K01

Final Objective Value: 0

Generations Run: 10

Search Duration: 0.3233 seconds

Convergence History: [24, 14, 10, 6, 6, 6, 6, 6, 4, 4]...



- b. Populasi: 50
i. Percobaan 1

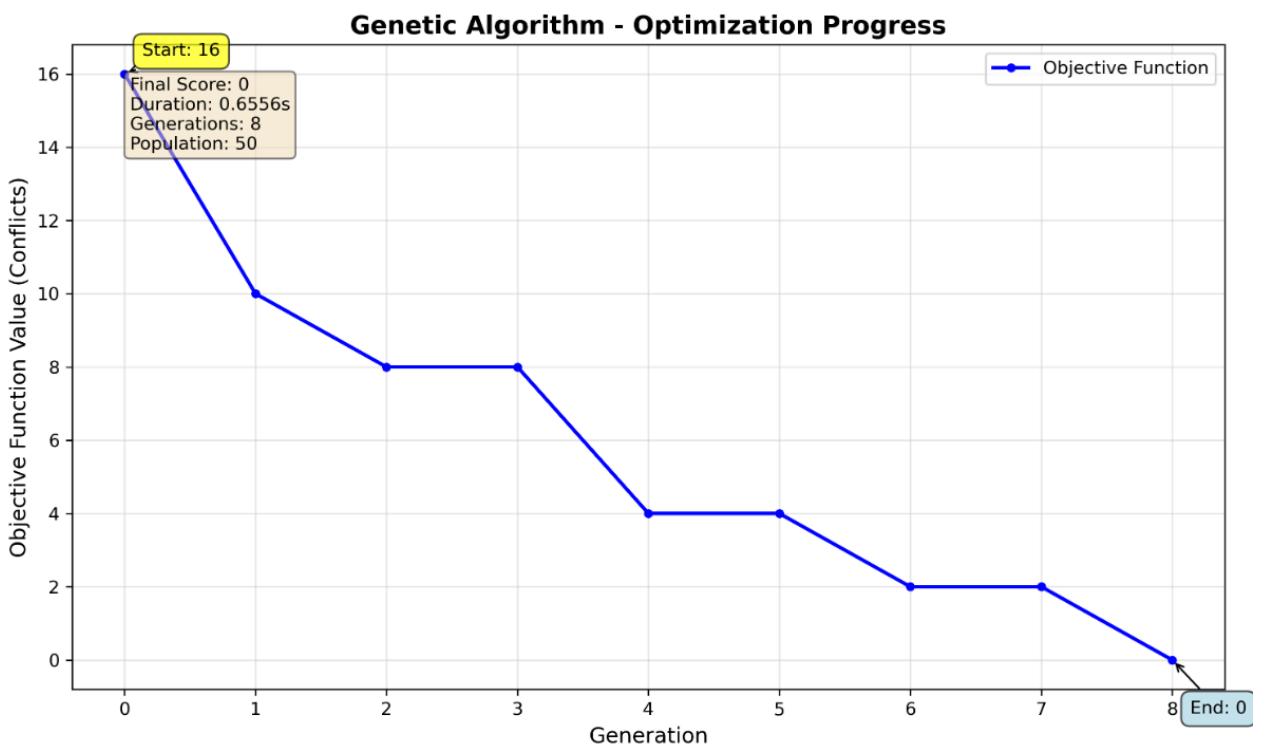
Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7602:IF2111_K01 7604:IF2120_K01	7608:IF2120_K01	-	7607:IF2110_K02
8:00	-	7606:IF2130_K01	7607:IF2200_K01	-	7607:IF2170_K01
9:00	7604:IF2160_K01	7605:IF2140_K01	-	-	7602:IF2140_K01 7608:IF2130_K01
10:00	7606:IF2110_K01	7607:IF2200_K01	7601:IF2170_K01	-	7605:IF2210_K01
11:00	7603:IF2110_K02 7604:IF2110_K01 7608:IF2160_K01	7604:IF2140_K01	-	7603:IF2200_K01	-
12:00	7601:IF2150_K02 7605:IF2110_K02	-	7603:IF2210_K01	7608:IF2150_K02	7605:IF2150_K01 7608:IF2160_K01
13:00	-	-	-	-	7606:IF2180_K01
14:00	7605:IF2140_K01	-	-	7602:IF2110_K02	7601:IF2110_K01
15:00	7608:IF2190_K01	7604:IF2180_K01	7601:IF2130_K01	-	7606:IF2120_K02
16:00	-	7602:IF2120_K02 7605:IF2190_K01 7608:IF2120_K01	7608:IF2110_K01	7601:IF2190_K01 7607:IF2111_K01 7608:IF2180_K01	7605:IF2120_K02
17:00	7604:IF2180_K01	-	-	7602:IF2150_K01	-

Genetic Algorithm - Final Best

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7606:IF2140_K01	-	-	-	7605:IF2110_K02
8:00	-	7605:IF2180_K01	7602:IF2160_K01 7607:IF2120_K01	7602:IF2130_K01	7604:IF2110_K02 7607:IF2110_K01
9:00	7608:IF2120_K01	7603:IF2180_K01	7604:IF2120_K02 7606:IF2110_K01	-	7605:IF2150_K01
10:00	7604:IF2190_K01	7601:IF2120_K02 7606:IF2190_K01	7605:IF2140_K01	7602:IF2170_K01	-
11:00	7605:IF2140_K01	-	7605:IF2110_K02	7604:IF2190_K01	7607:IF2180_K01
12:00	7601:IF2200_K01 7606:IF2110_K01	7605:IF2130_K01	7606:IF2150_K01	7606:IF2160_K01	-
13:00	-	-	7604:IF2150_K02 7608:IF2210_K01	7603:IF2111_K01	7603:IF2120_K01
14:00	-	-	7606:IF2110_K01	7605:IF2170_K01	-
15:00	7606:IF2180_K01	7606:IF2160_K01 7608:IF2200_K01	7608:IF2150_K02	-	7603:IF2200_K01
16:00	-	-	7601:IF2140_K01	7605:IF2120_K02	7602:IF2210_K01
17:00	7607:IF2111_K01	-	7605:IF2110_K02	7607:IF2130_K01	-

```
Final Objective Value: 0  
Generations Run: 8  
Search Duration: 0.6556 seconds  
Convergence History: [16, 10, 8, 8, 4, 4, 2, 2, 0]...
```



ii. Percobaan 2

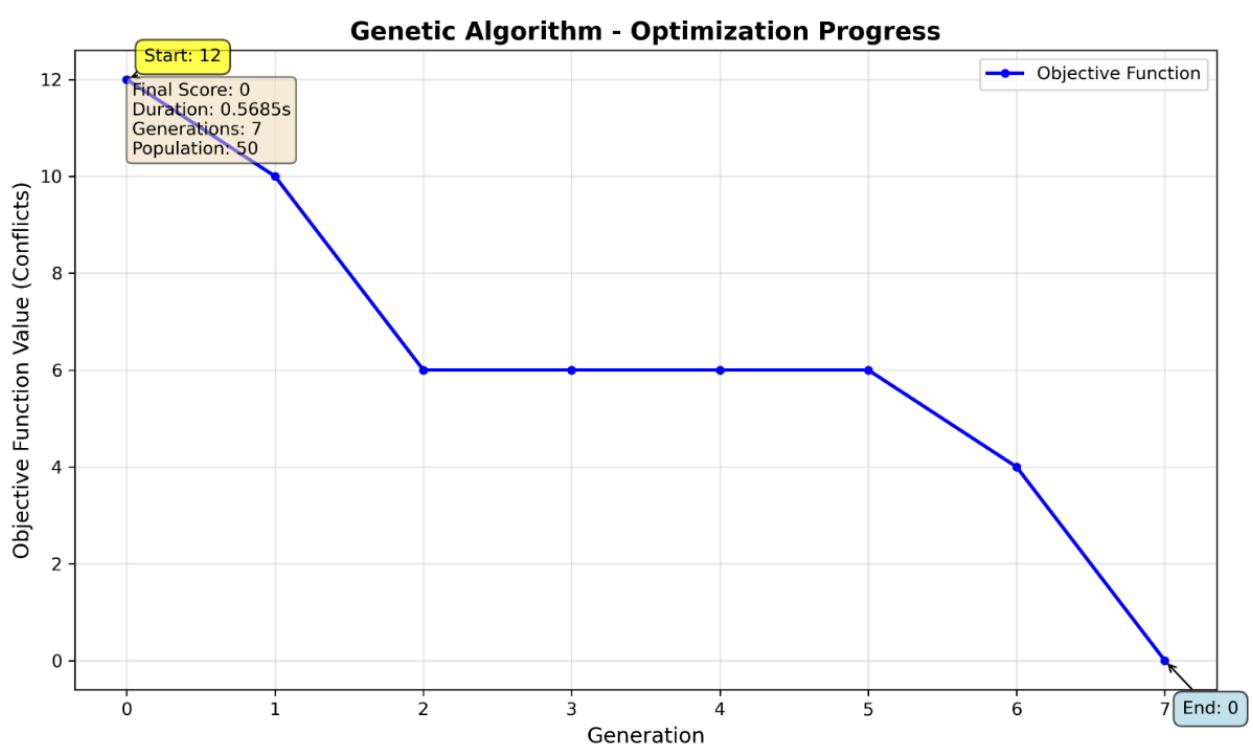
Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7606:IF2210_K01	7601:IF2160_K01	7601:IF2120_K01	-	7601:IF2110_K01 7602:IF2110_K02 7606:IF2130_K01
8:00	7601:IF2110_K01	7607:IF2160_K01	-	7607:IF2180_K01	7601:IF2120_K01
9:00	7605:IF2110_K02	7602:IF2120_K02	7607:IF2190_K01	7605:IF2170_K01	7607:IF2110_K02
10:00	-	7601:IF2200_K01 7603:IF2210_K01	7602:IF2120_K02 7605:IF2110_K01	7608:IF2110_K01	7601:IF2140_K01 7607:IF2180_K01
11:00	-	-	7604:IF2170_K01 7608:IF2140_K01	7604:IF2160_K01	7602:IF2190_K01 7606:IF2111_K01
12:00	-	-	7607:IF2120_K02	7602:IF2140_K01 7607:IF2111_K01 7608:IF2180_K01	7604:IF2140_K01 7605:IF2150_K02
13:00	7604:IF2200_K01 7606:IF2150_K01	7603:IF2200_K01 7605:IF2120_K01	-	-	-
14:00	7603:IF2150_K02	-	7602:IF2130_K01	-	-
15:00	-	-	7605:IF2110_K02	-	7607:IF2190_K01 7608:IF2180_K01
16:00	-	-	-	7605:IF2150_K01	-
17:00	-	7608:IF2130_K01	-	-	-

Genetic Algorithm - Final Best

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7601:IF2160_K01	-	7604:IF2110_K01 7605:IF2120_K02	7603:IF2130_K01
8:00	7605:IF2150_K02	7601:IF2120_K02	-	7606:IF2180_K01	-
9:00	7602:IF2110_K02 7604:IF2120_K01	7608:IF2160_K01	-	7603:IF2120_K01	7602:IF2130_K01 7608:IF2150_K01
10:00	7604:IF2140_K01	7607:IF2130_K01	7602:IF2190_K01	7602:IF2110_K02 7605:IF2120_K01	-
11:00	-	-	7606:IF2110_K01	7603:IF2180_K01 7605:IF2170_K01	7603:IF2210_K01
12:00	7607:IF2210_K01	-	7606:IF2110_K02	7608:IF2170_K01	7606:IF2140_K01
13:00	7603:IF2110_K01	-	-	7608:IF2110_K02	7602:IF2180_K01 7603:IF2200_K01
14:00	-	7608:IF2160_K01	7606:IF2190_K01	7603:IF2200_K01	7602:IF2111_K01 7604:IF2120_K02
15:00	7602:IF2111_K01	-	-	7604:IF2110_K01	7601:IF2180_K01
16:00	-	7605:IF2150_K02	-	-	7604:IF2190_K01
17:00	7604:IF2140_K01 7606:IF2150_K01	7603:IF2200_K01	-	-	7601:IF2140_K01

```
Final Objective Value: 0  
Generations Run: 7  
Search Duration: 0.5685 seconds  
Convergence History: [12, 10, 6, 6, 6, 6, 4, 0]...
```



iii. Percobaan 3

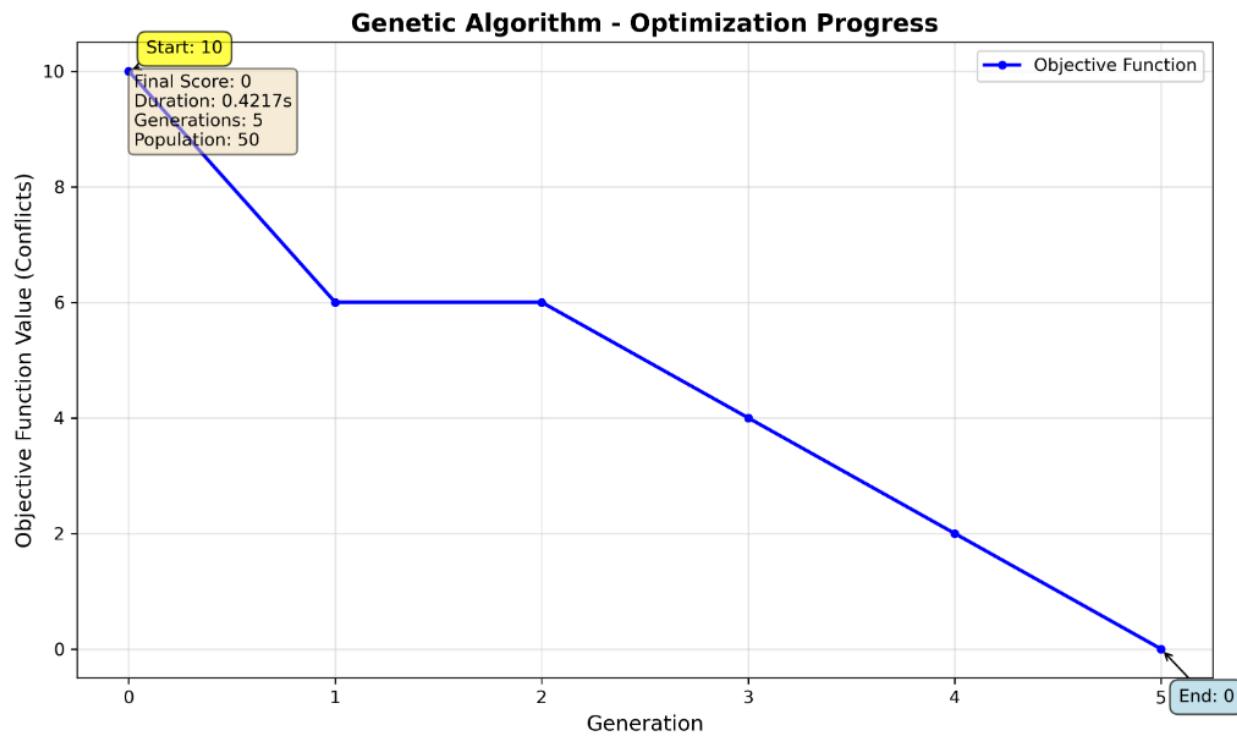
Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	-	7601:IF2110_K02 7608:IF2111_K01	7607:IF2111_K01 7608:IF2120_K02	7601:IF2190_K01
8:00	7608:IF2160_K01	7601:IF2170_K01	7606:IF2150_K01	7607:IF2140_K01	-
9:00	7601:IF2110_K01	-	7603:IF2180_K01	7606:IF2170_K01 7607:IF2210_K01 7608:IF2160_K01	-
10:00	7607:IF2140_K01	-	-	-	-
11:00	-	7604:IF2120_K01	-	7603:IF2180_K01 7608:IF2140_K01	-
12:00	7603:IF2200_K01	7608:IF2110_K01	-	7607:IF2160_K01	7603:IF2110_K02 7604:IF2180_K01
13:00	7601:IF2110_K02 7607:IF2110_K02	-	7607:IF2130_K01	7608:IF2120_K01	7603:IF2130_K01
14:00	7604:IF2200_K01	7604:IF2190_K01 7606:IF2150_K02	-	-	-
15:00	-	7601:IF2110_K01 7603:IF2130_K01	7602:IF2140_K01	7608:IF2180_K01	7602:IF2120_K01
16:00	7604:IF2120_K02	-	7603:IF2120_K02	7608:IF2110_K01	-
17:00	-	7601:IF2150_K01 7605:IF2190_K01	7603:IF2150_K02 7605:IF2200_K01 7607:IF2110_K01	-	-

Genetic Algorithm - Final Best

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7603:IF2180_K01	-	7602:IF2110_K02 7603:IF2110_K01	7607:IF2140_K01	-
8:00	7603:IF2110_K02	-	7602:IF2130_K01	7603:IF2210_K01	7607:IF2140_K01
9:00	-	7607:IF2160_K01	7607:IF2170_K01	7606:IF2110_K02	-
10:00	-	-	7607:IF2111_K01	-	7603:IF2120_K01
11:00	-	7608:IF2140_K01	7602:IF2120_K02	7607:IF2120_K01	7601:IF2200_K01
12:00	-	7601:IF2190_K01 7602:IF2120_K02	7604:IF2180_K01	-	7604:IF2140_K01
13:00	7608:IF2110_K01	-	7603:IF2190_K01 7606:IF2200_K01	7608:IF2111_K01	7604:IF2150_K02 7607:IF2110_K01
14:00	7603:IF2160_K01	-	7606:IF2110_K02	7603:IF2120_K01	-
15:00	7605:IF2180_K01	7607:IF2200_K01	7607:IF2110_K01	7604:IF2120_K02	-
16:00	7606:IF2190_K01 7607:IF2150_K01	7605:IF2160_K01 7606:IF2170_K01	-	-	-
17:00	7601:IF2130_K01 7607:IF2150_K01	7607:IF2210_K01	7604:IF2150_K02 7605:IF2180_K01	-	7602:IF2130_K01

```
Final Objective Value: 0  
Generations Run: 5  
Search Duration: 0.4217 seconds  
Convergence History: [10, 6, 6, 4, 2, 0]...
```



- c. Populasi 100
 - i. Percobaan 1

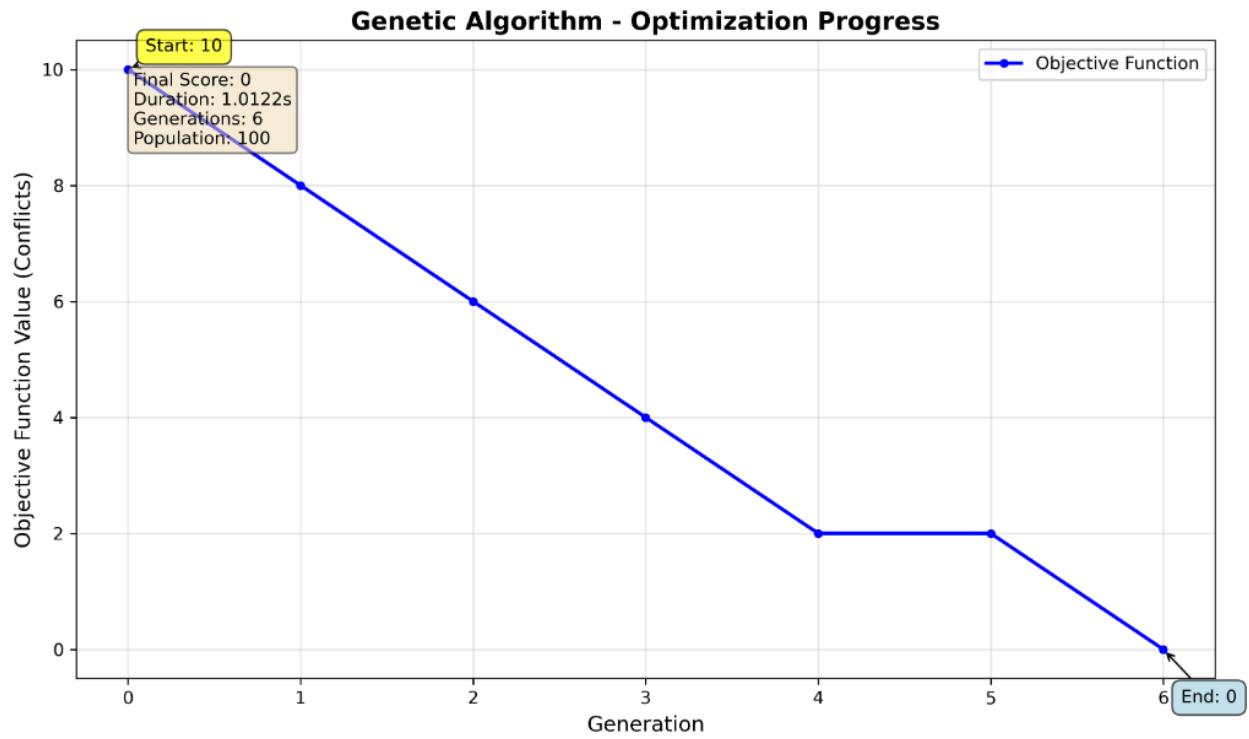
Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7604:IF2160_K01	7608:IF2210_K01	-	7601:IF2180_K01 7604:IF2200_K01	7603:IF2111_K01 7605:IF2111_K01
8:00	7602:IF2110_K02	7603:IF2110_K01	-	7602:IF2120_K01	-
9:00	-	7602:IF2130_K01 7606:IF2120_K02	7608:IF2150_K02	-	7608:IF2200_K01
10:00	-	-	7602:IF2120_K02	7601:IF2180_K01 7604:IF2130_K01	-
11:00	-	-	-	7606:IF2110_K01	7602:IF2190_K01
12:00	-	7606:IF2110_K01 7607:IF2140_K01	7603:IF2160_K01 7606:IF2110_K02	-	-
13:00	7604:IF2190_K01 7608:IF2190_K01	7604:IF2150_K01 7608:IF2210_K01	7607:IF2160_K01	7601:IF2111_K01	7605:IF2150_K01
14:00	-	7606:IF2170_K01	7603:IF2110_K01	7605:IF2140_K01 7607:IF2180_K01	-
15:00	-	-	-	-	7604:IF2110_K02
16:00	7606:IF2170_K01	7604:IF2140_K01	7606:IF2120_K01	7608:IF2110_K02	-
17:00	7602:IF2180_K01	7601:IF2140_K01	7602:IF2130_K01 7607:IF2200_K01	-	7601:IF2150_K02 7603:IF2120_K02

Genetic Algorithm - Final Best

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7605:IF2150_K01	-	7605:IF2140_K01	7606:IF2140_K01
8:00	-	7606:IF2190_K01	7606:IF2130_K01	7603:IF2120_K02	7607:IF2110_K01
9:00	-	-	7606:IF2120_K02	7605:IF2110_K01	-
10:00	7608:IF2120_K01	-	7608:IF2180_K01	7607:IF2120_K01	7604:IF2120_K02
11:00	7608:IF2180_K01	-	7603:IF2160_K01	7608:IF2200_K01	7602:IF2180_K01 7605:IF2170_K01
12:00	7602:IF2160_K01	7601:IF2120_K01	-	-	7604:IF2111_K01
13:00	7608:IF2110_K02	7602:IF2130_K01 7605:IF2110_K02	7601:IF2210_K01	7602:IF2160_K01	7605:IF2180_K01 7606:IF2200_K01
14:00	7605:IF2190_K01	7601:IF2170_K01	7607:IF2150_K02	7607:IF2110_K02	7602:IF2190_K01
15:00	7608:IF2110_K01	7602:IF2150_K01 7605:IF2150_K02	-	7603:IF2110_K01	7602:IF2111_K01
16:00	-	-	7603:IF2140_K01	7602:IF2210_K01	7608:IF2140_K01
17:00	-	7604:IF2130_K01	-	7604:IF2200_K01	7607:IF2110_K02

```
Final Objective Value: 0  
  
Generations Run: 6  
  
Search Duration: 1.0122 seconds  
  
Convergence History: [10, 8, 6, 4, 2, 2, 0]...
```



ii. Percobaan 2

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7608:IF2160_K01	-	7605:IF2130_K01	7603:IF2170_K01 7604:IF2120_K01 7607:IF2180_K01	7607:IF2120_K02
8:00	7603:IF2110_K01 7607:IF2180_K01	7601:IF2140_K01 7608:IF2200_K02	7601:IF2190_K01 7606:IF2200_K01	-	-
9:00	7601:IF2210_K01 7606:IF2120_K01	7603:IF2110_K02	7601:IF2110_K01	-	-
10:00	7604:IF2120_K02	7603:IF2190_K01	-	7603:IF2111_K01	-
11:00	-	7601:IF2110_K01	-	7608:IF2110_K02	-
12:00	-	-	7601:IF2130_K01	7601:IF2120_K02	-
13:00	7606:IF2150_K02 7607:IF2170_K01	-	7605:IF2160_K01	7605:IF2140_K01	-
14:00	-	-	7602:IF2180_K01 7607:IF2150_K01	7603:IF2110_K01	7605:IF2140_K01
15:00	7604:IF2111_K01 7605:IF2140_K01	7606:IF2150_K01	-	7605:IF2200_K01	-
16:00	-	7601:IF2110_K02 7604:IF2160_K01	7606:IF2180_K01	-	-
17:00	-	7601:IF2190_K01 7603:IF2130_K01	-	7601:IF2190_K02 7602:IF2210_K01 7604:IF2120_K01 7606:IF2200_K01	-

Genetic Algorithm - Final Best

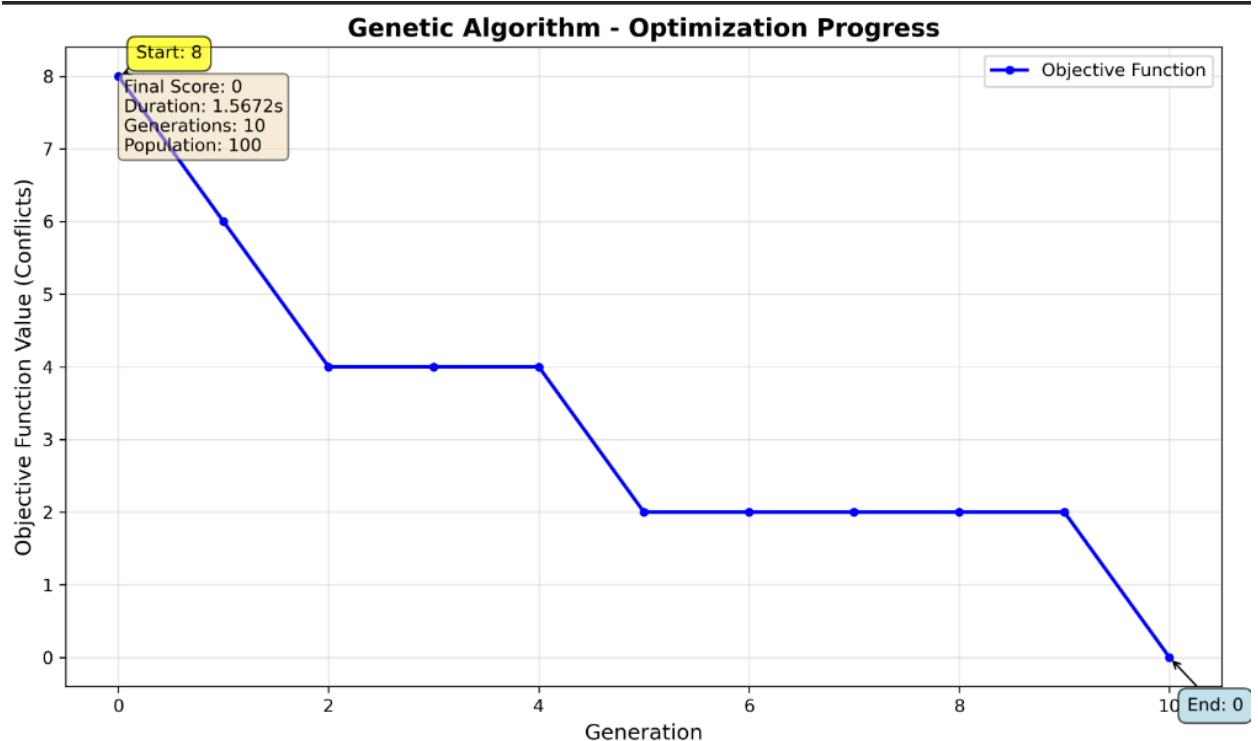
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7607:IF2120_K02	-	-	7605:IF2110_K02
8:00	-	7605:IF2170_K01	7605:IF2150_K01 7607:IF2110_K01	7606:IF2110_K01	7608:IF2150_K02
9:00	7606:IF2111_K01	7608:IF2160_K01	-	7605:IF2120_K02 7608:IF2190_K01	-
10:00	7605:IF2111_K01	7606:IF2110_K02	7605:IF2110_K01	-	7608:IF2110_K02
11:00	7601:IF2200_K01	7603:IF2190_K01	7606:IF2180_K01	7601:IF2120_K02	7603:IF2110_K01
12:00	7605:IF2120_K01	7606:IF2170_K01	7603:IF2200_K01	7605:IF2210_K01	-
13:00	7605:IF2160_K01	7604:IF2210_K01	7607:IF2150_K02	-	7606:IF2180_K01
14:00	7601:IF2130_K01	7607:IF2140_K01	7606:IF2180_K01	-	7607:IF2130_K01
15:00	7601:IF2180_K01	7605:IF2140_K01	7605:IF2150_K01	7602:IF2140_K01	7601:IF2190_K01 7608:IF2160_K01
16:00	7608:IF2110_K02	-	7605:IF2130_K01	-	-
17:00	7604:IF2200_K01	7604:IF2120_K01	7608:IF2110_K01	-	7602:IF2120_K01

Final Objective Value: 0

Generations Run: 10

Search Duration: 1.5672 seconds

Convergence History: [8, 6, 4, 4, 4, 2, 2, 2, 2, 2]...



iii. Percobaan 3

Genetic Algorithm - Initial Schedule

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	-	7602:IF2110_K01	-	-	7607:IF2110_K01
8:00	-	7605:IF2110_K01	-	-	-
9:00	7602:IF2140_K01	7602:IF2110_K02	7602:IF2160_K01	-	7607:IF2110_K02
10:00	7602:IF2110_K02	7603:IF2180_K01 7607:IF2180_K02 7608:IF2170_K01	7608:IF2110_K01	7607:IF2130_K01	-
11:00	7601:IF2120_K01	-	7602:IF2210_K01	7602:IF2140_K01	7604:IF2200_K01 7606:IF2190_K01
12:00	7607:IF2120_K02	7601:IF2140_K01	7601:IF2200_K01	7607:IF2190_K01	7603:IF2160_K01
13:00	-	7603:IF2150_K02	7601:IF2180_K01 7602:IF2180_K01 7604:IF2190_K01 7606:IF2111_K01	7608:IF2180_K01	7604:IF2111_K01
14:00	7606:IF2110_K02	7601:IF2150_K01 7606:IF2150_K01	-	7601:IF2210_K01 7604:IF2150_K02	-
15:00	-	-	-	-	7606:IF2180_K01 7607:IF2200_K01
16:00	7603:IF2130_K01 7608:IF2140_K01	7602:IF2110_K01	7604:IF2130_K01	-	7601:IF2160_K01
17:00	7605:IF2120_K02	-	-	-	7608:IF2170_K01

Genetic Algorithm - Final Best

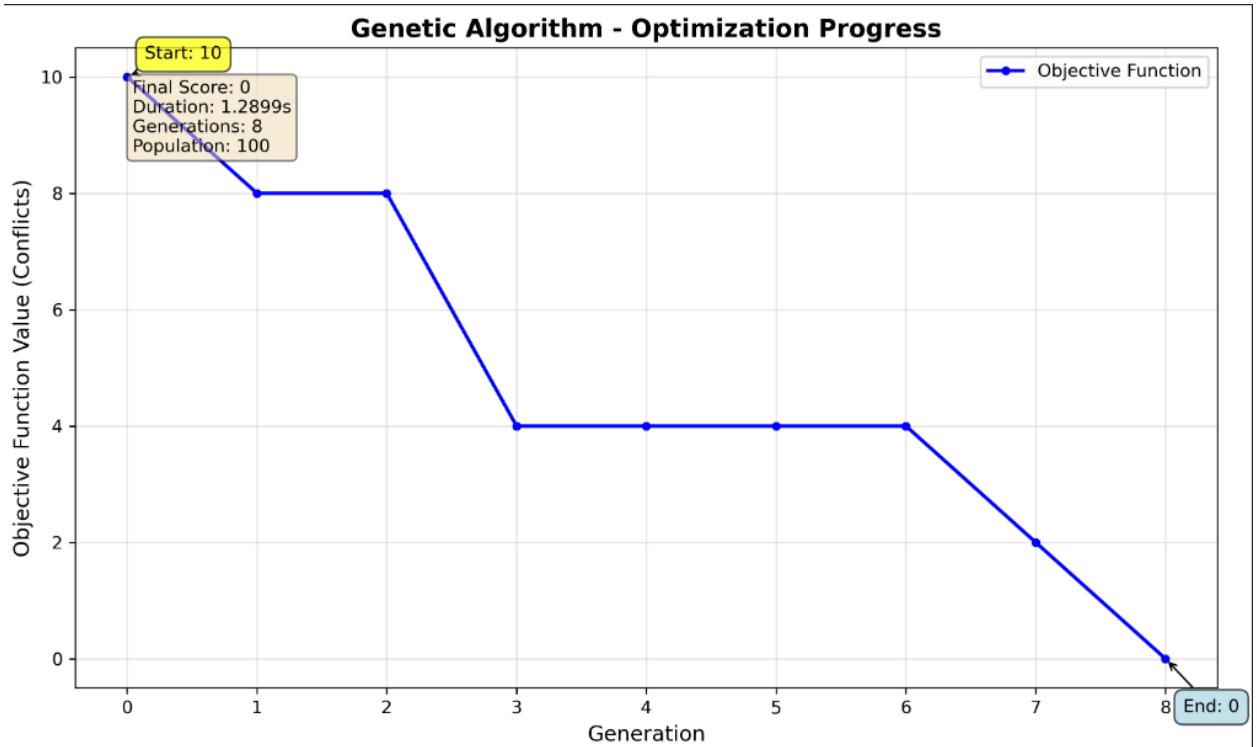
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
7:00	7607:IF2120_K02	-	-	-	7604:IF2150_K02 7606:IF2150_K01
8:00	7608:IF2140_K01	7602:IF2120_K02	-	-	7605:IF2170_K01 7608:IF2190_K01
9:00	7605:IF2200_K01	7606:IF2160_K01	-	-	-
10:00	7604:IF2180_K01	-	7608:IF2180_K01	7608:IF2110_K01	7608:IF2170_K01
11:00	-	7607:IF2110_K01	7606:IF2110_K01	7602:IF2110_K02	-
12:00	7607:IF2110_K02	7604:IF2110_K01	-	7608:IF2111_K01	7602:IF2160_K01
13:00	7603:IF2160_K01 7604:IF2200_K01	7605:IF2110_K02	7602:IF2190_K01	7601:IF2120_K01 7604:IF2150_K02	7606:IF2210_K01
14:00	7605:IF2120_K02 7606:IF2180_K01	7601:IF2130_K01	-	-	7603:IF2110_K02
15:00	7606:IF2130_K01	7604:IF2140_K01	7604:IF2130_K01	7608:IF2200_K01	7607:IF2120_K01
16:00	7604:IF2111_K01	7603:IF2110_K01	7606:IF2180_K01	-	7602:IF2190_K01
17:00	7603:IF2210_K01	-	7607:IF2140_K01	7608:IF2140_K01	7606:IF2150_K01

Final Objective Value: 0

Generations Run: 8

Search Duration: 1.2899 seconds

Convergence History: [10, 8, 8, 4, 4, 4, 4, 2, 0]...



2.4 Analisis Hasil dan Pembahasan

Algoritma	Percobaan	Parameter Khusus	Skor Awal (Konflik)	Skor Akhir (Konflik)	Total Iterasi / Generasi	Durasi (detik)
2.3.1 Steepest Ascent Hill-Climbing	1	-	30	0	6	7.0856
	2	-	24	0	7	8.1479
	3	-	18	0	7	7.6799
	1	-	28	0	99	74.6416

2.3.2 Stochastic Hill-climbing	2	-	16	0	136	101.1925
	3	-	30	0	158	118.1943
2.3.3 Hill-climbing with Sideways	1	Max Sideways = 20	41	0	14	15.9365
	2	Max Sideways = 40	40	0	8	8.9482
	3	Max Sideways = 60	32	0	10	11.2358
2.3.4 Random Restart Hill-climbing	1	Max Restart = 5, Max Iter/Restart = 6	42 (Skor R1)	0 (di R3)	[6, 6, 6]	20.2273
	2	Max Restart = 2, Max Iter/Restart = 8	34 (Skor R1)	0 (di R2)	[8, 6]	15.5665
	3	Max Restart = 6, Max Iter/Restart = 5	32 (Skor R1)	2 (Gagal)	[5, 5, 5, 5, 5]	34.642
2.3.5 Simulated Annealing	1	Initial Temp = 100, Cooling Rate = 0.95	40	0	~325	432.6234
	2	(Sama)	~48 (Puncak)	0	~128	175.9941
	3	(Sama)	38	0	~220	377.1371
	1a-i	Pop=50, MaxIter=50, MutRate=1	8	0	9	0.7261
	1a-ii	Pop=50, MaxIter=50, MutRate=1	14	0	12	0.9051
	1a-iii	Pop=50, MaxIter=50, MutRate=1	14	0	6	0.4963

2.3.6 Genetic Algorithm

	1b-ii	Pop=50, MaxIter=100, MutRate=1	12	0	9	0.7248
	1b-iii	Pop=50, MaxIter=100, MutRate=1	10	0	8	0.6924
	1c-i	Pop=50, MaxIter=200, MutRate=1	6	0	7	0.5691
	1c-ii	Pop=50, MaxIter=200, MutRate=1	14	0	11	0.9347
	1c-iii	Pop=50, MaxIter=200, MutRate=1	6	0	10	0.8127
	2a-i	Pop=20, MaxIter=100, MutRate=1	15	0	6	0.199
	2a-ii	Pop=20, MaxIter=100, MutRate=1	18	0	9	0.2999
	2a-iii	Pop=20, MaxIter=100, MutRate=1	24	0	10	0.3233
	2b-i	Pop=50, MaxIter=100, MutRate=1	16	0	8	0.6556
	2b-ii	Pop=50, MaxIter=100, MutRate=1	12	0	7	0.5685
	2b-iii	Pop=50, MaxIter=100, MutRate=1	10	0	5	0.4217
	2c-i	Pop=100, MaxIter=100, MutRate=1	10	0	6	1.0122

	2c-ii	Pop=100, MaxIter=100, MutRate=1	8	0	10	1.5672
	2c-iii	Pop=100, MaxIter=100, MutRate=1	10	0	8	1.2899

2.4.1 Steepest Ascent Hill-Climbing

Algoritma ini menunjukkan kinerja yang **sangat baik dan konsisten** pada semua percobaan.

- **Kecepatan:** Sangat cepat, menyelesaikan optimasi hanya dalam 6-7 iterasi dengan durasi rata-rata 7-8 detik.
- **Keberhasilan:** Selalu berhasil mencapai skor optimal (0 konflik).
- **Analisis:** Algoritma ini bersifat "serakah" (greedy), artinya ia selalu memilih langkah terbaik (penurunan konflik terbesar) yang bisa ditemukannya. Fakta bahwa ia berhasil dengan sangat cepat dan tidak pernah gagal menyiratkan bahwa *landscape* solusi untuk data uji ini relatif sederhana. Ia tidak memiliki banyak "jebakan" *local optimum* yang dalam, sehingga strategi serakah ini sudah cukup untuk menemukan solusi global terbaik.

2.4.2 Stochastic Hill-Climbing

Algoritma ini **berhasil, namun sangat tidak efisien** dibandingkan dengan Steepest Ascent.

- **Kecepatan:** Jauh lebih lambat, membutuhkan 74 hingga 118 detik.
- **Iterasi:** Membutuhkan jumlah iterasi yang sangat besar (99 hingga 158) untuk mencapai solusi.
- **Analisis:** Perilaku ini sesuai dengan deskripsinya. Alih-alih memilih langkah *terbaik* yang tersedia, ia memilih *salah satu* langkah yang lebih baik secara acak. Ini berarti ia sering kali tidak mengambil langkah yang paling efisien, menyebabkannya "berkeliling" di area solusi lebih lama sebelum akhirnya mencapai titik optimal.

2.4.3 Hill-Climbing with Sideways Move

Algoritma ini menunjukkan kinerja yang **sangat baik, mirip dengan Steepest Ascent**.

- **Kecepatan:** Sangat cepat, dengan durasi 8 hingga 16 detik.
- **Iterasi:** Membutuhkan sedikit iterasi (8-14) untuk berhasil.
- **Analisis:** Algoritma ini dirancang untuk mengatasi *plateau* (area datar di mana tidak ada langkah yang lebih baik, hanya setara). Kemampuannya untuk mengambil "langkah ke samping" (pindah ke solusi dengan skor yang sama) terbukti efektif. Kecepatan dan keberhasilannya yang konsisten menunjukkan bahwa ia dapat dengan mudah menavigasi area *plateau* mana pun yang mungkin telah menjebak algoritma Steepest Ascent murni.

2.4.4 Random Restart Hill-Climbing

Algoritma ini memberikan **hasil yang tidak konsisten** dan merupakan satu-satunya varian Hill-Climbing yang gagal.

- **Keberhasilan:** Berhasil di dua percobaan pertama (mencapai 0 di *restart* ke-2 atau ke-3), tetapi **gagal** di percobaan ketiga, berakhir dengan skor 2 konflik.
- **Analisis:** Strategi algoritma ini adalah menjalankan Steepest Ascent berulang kali dari titik awal yang acak. Keberhasilannya sepenuhnya bergantung pada keberuntungan; ia harus "mendarat" di titik awal yang memiliki jalur menuju solusi optimal global. Kegagalan di percobaan 3 menunjukkan bahwa setelah 6 kali *restart*, tidak satu pun dari titik awal tersebut yang cukup baik untuk menemukan solusi 0

2.4.5 Stimulated Annealing

Algoritma ini **paling lambat, tetapi sangat kuat (robust)**.

- **Kecepatan:** Jauh paling lambat dari semua algoritma, memakan waktu 175 hingga 432 detik (lebih dari 7 menit di percobaan 1).
- **Keberhasilan:** Selalu berhasil mencapai 0 konflik di semua percobaan.
- **Analisis:** Kelambatannya adalah bagian dari desainnya. Simulated Annealing secara sengaja menerima solusi yang lebih buruk (skor konflik lebih tinggi) berdasarkan "suhu". Ini memungkinkannya untuk "melompat" keluar dari *local optimum* yang dalam yang akan menjebak semua varian Hill-Climbing. Grafik progresnya menunjukkan perilaku "kacau" ini (skor naik-turun) sebelum akhirnya "mendingin" dan fokus pada solusi optimal.

2.4.6 Genetic Algorithm

Algoritma ini adalah **pemenang yang jelas dalam hal kecepatan dan konsistensi**.

- **Kecepatan:** Sangat cepat. Hampir semua percobaan selesai dalam **kurang dari 1 detik**.
- **Iterasi (Generasi):** Hanya membutuhkan 5 hingga 12 generasi untuk menemukan solusi 0.
- **Keberhasilan:** Berhasil 100% di semua variasi parameter.
- **Analisis:** Performanya yang superior berasal dari pendekatannya yang berbasis populasi. Alih-alih memperbaiki satu jadwal, ia mengelola puluhan jadwal sekaligus. Melalui proses *crossover* (menggabungkan jadwal baik) dan *mutasi* (perubahan acak), ia dapat menjelajahi ruang solusi secara paralel dan masif. Ini memungkinkannya untuk menemukan solusi optimal dengan sangat cepat. Menariknya, populasi yang lebih kecil (20) justru bekerja lebih cepat daripada populasi besar (100) untuk masalah ini, mungkin karena *overhead* yang lebih rendah

BAB III

KESIMPULAN & SARAN

3.1 Kesimpulan

Berdasarkan analisis komprehensif dari keenam algoritma *local search* yang diimplementasikan untuk menyelesaikan masalah penjadwalan kelas, dapat ditarik kesimpulan bahwa **pendekatan berbasis populasi (Genetic Algorithm) secara signifikan mengungguli semua metode pencarian berbasis single-state**. Meskipun hampir semua algoritma pada akhirnya berhasil menemukan solusi optimal (0 konflik), perbedaan dalam efisiensi, kecepatan, dan konsistensi sangatlah jelas.

Performa paling menonjol ditunjukkan oleh **Genetic Algorithm (2.3.6)**. Algoritma ini secara konsisten menjadi yang tercepat, dengan waktu eksekusi yang hampir selalu di bawah satu detik. Kecepatan yang superior ini dapat diatribusikan langsung ke metodologinya. Berbeda dengan algoritma lain yang memperbaiki satu jadwal (state) pada satu waktu, GA mengelola seluruh *populasi* kandidat jadwal . Dengan menerapkan operator *crossover* (menggabungkan jadwal-jadwal baik) dan *mutasi* (variasi acak) secara paralel, ia mampu menjelajahi ruang solusi secara eksponensial lebih efisien, memungkinkannya untuk "menemukan" solusi optimal hanya dalam 5 hingga 12 generasi.

Di spektrum yang berlawanan, algoritma **Simulated Annealing (2.3.5)** adalah yang **paling lambat secara absolut**, dengan waktu eksekusi mencapai 175 hingga 432 detik (lebih dari 7 menit). Ironisnya, kelambatannya ini justru merupakan bagian dari desainnya yang *robust*. SA dirancang untuk "menerima" solusi yang lebih buruk secara probabilistik untuk melarikan diri dari *local optimum* yang dalam. Namun, untuk data uji ini, algoritma *greedy* sederhana seperti Steepest Ascent terbukti sudah cukup. Ini menunjukkan bahwa kemampuan SA yang kuat untuk melarikan diri dari jebakan tidak diperlukan, sehingga fitur tersebut justru menjadi pemborosan waktu komputasi yang signifikan.

Varian-varian Hill-Climbing menunjukkan hasil yang beragam. **Steepest Ascent (2.3.1)** dan **Hill-Climbing with Sideways Move (2.3.3)** tampil sebagai pilihan yang "cukup baik". Keduanya relatif cepat (7-16 detik) dan konsisten menemukan solusi 0. Keberhasilan mereka

menyiratkan bahwa *landscape* solusi dari masalah ini tidak terlalu rumit; strategi "serakah" untuk selalu mengambil langkah terbaik sudah cukup untuk mencapai solusi global.

Sebaliknya, **Stochastic Hill-Climbing (2.3.2)**, yang memilih langkah perbaikan secara acak alih-alih memilih yang terbaik, terbukti **sangat tidak efisien** (74-118 detik). Terakhir, **Random Restart Hill-Climbing (2.3.4)** adalah satu-satunya algoritma yang **tidak konsisten**, gagal mencapai solusi 0 pada salah satu percobaannya. Hal ini menyoroti kelemahan utamanya: ia bergantung murni pada keberuntungan untuk "mendarat" di titik awal yang memiliki jalur ke solusi optimal global.

3.2 Saran

1. Penyempurnaan *Objective Function* dan Penanganan *Constraints*

- Fungsi objektif yang digunakan saat ini (2.1) telah berhasil meminimalkan konflik waktu mahasiswa. Namun, deskripsi masalah awal (Bab I) juga menyebutkan berbagai batasan lain yang krusial, seperti kapasitas ruangan, larangan penggunaan ruangan ganda pada waktu yang sama, dan prioritas mata kuliah.
- Disarankan untuk mengintegrasikan batasan-batasan ini secara eksplisit ke dalam *objective function*. Misalnya, dengan memberikan nilai penalti yang berbeda untuk setiap jenis pelanggaran (*weighted penalty*). Ini akan memastikan bahwa solusi optimal yang ditemukan tidak hanya bebas dari konflik jadwal mahasiswa, tetapi juga valid secara logistik (misalnya, tidak ada 100 mahasiswa di ruangan berkapasitas 30) dan menghormati prioritas akademik.

2. Optimasi *Hyperparameter* dan Operator pada *Genetic Algorithm* (GA)

- Hasil eksperimen (2.3.6) dengan jelas menunjukkan bahwa *Genetic Algorithm* (GA) adalah algoritma yang paling unggul secara performa, baik dari segi kecepatan maupun konsistensi. Untuk pengembangan lebih lanjut, disarankan untuk melakukan eksplorasi yang lebih mendalam terhadap GA itu sendiri.
- Ini dapat mencakup *hyperparameter tuning* yang lebih sistematis (misalnya, menguji dampak *population size* dan *mutation rate* secara lebih granular) serta bereksperimen dengan operator genetik yang berbeda, seperti metode seleksi (selain *tournament selection*) atau strategi *crossover* (selain *one-point crossover*).

3. Pengujian pada Set Data yang Lebih Besar dan Kompleks

- Keberhasilan algoritma *greedy* yang relatif sederhana seperti *Steepest Ascent Hill-Climbing* (2.3.1) dalam menemukan solusi optimal 0 konflik dengan sangat cepat mengindikasikan bahwa *landscape* solusi dari data uji yang digunakan saat ini mungkin tidak terlalu kompleks.
- Disarankan untuk menguji keenam algoritma ini pada set data yang lebih besar (lebih banyak mata kuliah, mahasiswa, dan ruangan) dan memiliki batasan yang lebih ketat. Pada skenario yang lebih sulit, keunggulan sejati dari algoritma yang lebih *robust* (seperti *Simulated Annealing* yang mampu lolos dari *local optimum*

dan GA yang melakukan pencarian paralel) diperkirakan akan terlihat jauh lebih jelas dibandingkan dengan algoritma *Hill-Climbing* standar.

PEMBAGIAN TUGAS

Anggota	Tugas
Sebastian Enrico Nathanael - 13523134	<ul style="list-style-type: none">- Simulated Annealing- Testing- Laporan
Jonathan Kenan Budianto - 13523139	<ul style="list-style-type: none">- Steepest Ascent Hill-Climbing- Hill-Climbing with Sideways Move- Random Restart Hill-Climbing- Stochastic Hill-Climbing- Objective Function- Visualization Output- Laporan- Testing
Mahesa Fadhillah Andre - 13523140	<ul style="list-style-type: none">- Basic Entities- Input Parser- State representation and logic (schedule.py, registry.py)- Workspace structure- Genetic Algorithm- Laporan- Testing

REFERENSI

Masayu Leylia Khodra, *Beyond Classical Search: Local Search*, EDUNEX ITB. Modul 3: Inteligensi Buatan, 2019

Masayu Leylia Khodra, *Beyond Classical Search: Hill-Climbing Search*, EDUNEX ITB. Modul 3: Inteligensi Buatan, 2019

Masayu Leylia Khodra, *Beyond Classical Search: Simulated Annealing*, EDUNEX ITB. Modul 3: Inteligensi Buatan, 2019

Masayu Leylia Khodra, *Beyond Classical Search: Genetic Algorithm*, EDUNEX ITB. Modul 3: Inteligensi Buatan, 2019