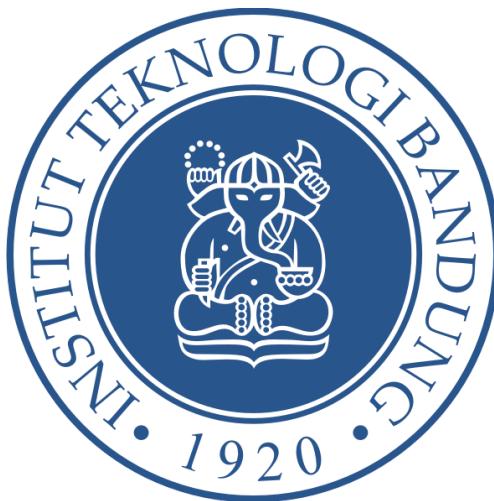


**TUGAS BESAR 1 IF3170 INTELEGensi BUATAN
SEMESTER II TAHUN 2025/2026**

**PENCARIAN SOLUSI PENJADWALAN KELAS MINGGUAN
DENGAN LOCAL SEARCH**



Kelompok 26 - kebodohan alami

Disusun oleh:

Muhammad Aufa Farabi	13523023
Ferdinand Gabe Tua Sinaga	13523051
Muhammad Iqbal Haidar	13523111

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025**

DAFTAR ISI

DESKRIPSI PERSOALAN.....	3
Daftar Variabel.....	3
Ketentuan Representasi.....	4
Objective Function.....	6
PEMBAHASAN.....	8
Pemilihan Objective Function.....	8
Implementasi.....	8
Steepest Ascent Hill Climbing.....	8
Sideways Move Hill Climbing.....	9
Random Restart Climbing.....	10
Stochastic Hill Climbing.....	12
Simulated Annealing.....	13
Genetic Algorithm.....	14
Eksperimen.....	17
Steepest Ascent Hill Climbing.....	17
Sideways Move Hill Climbing.....	24
Random Restart Hill Climbing.....	32
Stochastic Hill Climbing.....	39
Simulated Annealing.....	47
Genetic Algorithm.....	54
Analisis.....	56
KESIMPULAN DAN SARAN.....	61
PEMBAGIAN TUGAS.....	61
REFERENSI.....	62

DESKRIPSI persoalan

Tugas ini bertujuan untuk membuat jadwal kelas selama satu minggu untuk sekumpulan mata kuliah yang harus dipasangkan dengan waktu dan tempat (ruang/kelas).

Daftar Variabel

Entitas	Atribut	Contoh	Keterangan
Kelas Mata kuliah	Kode	IF3071_K01	Kode mata kuliah dan kode kelas digabung.
	Jumlah mahasiswa terdaftar	60	Jumlah mahasiswa 60 hanya untuk satu kelas saja (dalam kasus ini untuk K01 sesuai kode di atas).
	Jumlah SKS	3	
Waktu	Waktu mulai	[Senin, 11]	Jam dalam integer. 1 SKS = 1 jam. Struktur data gabungan harus list .
	Waktu akhir	[Selasa, 13]	
Ruangan	Kode ruangan	7609	Untuk ruangan yang tidak memiliki kode, gunakan nama ruangan sebagai kode (misalnya “multimedia”).
	Kuota ruangan	60	Ruangan tersebut dapat menampung 60 mahasiswa.
Mahasiswa	NIM	13523999	
	Daftar mata kuliah	[IF3071_K01, IF9999_K99]	Bentuk data bebas, bisa menggunakan dictionary, dua list, atau implementasi lain.
	Prioritas mata kuliah	[1,2]	Elemen harus unik (satu orang hanya memiliki 1 matkul untuk prioritas tertentu).

Ketentuan Representasi

- Struktur data yang digunakan untuk representasi bebas. Misal, menggunakan key-value seperti JSON.
- Representasi solusi (**state**) adalah pemetaan setiap Pertemuan Mata Kuliah ke Waktu dan Ruangan.
- Inisialisasi (**initial state**) random.
- Langkah neighbor (**move**) yang diperbolehkan tiap iterasi:
 1. menukar posisi dua pertemuan mata kuliah
 2. memindahkan pertemuan mata kuliah ke waktu dan ruangan yang kosong*Pertemuan Mata Kuliah merupakan kombinasi dari Kelas Mata Kuliah, Waktu, dan Ruangan sesuai dengan jumlah SKS yang dimiliki. Misalnya IF3071_K01 dengan 3 sks memiliki 2 Pertemuan Mata Kuliah, kelas di jam 7-9 hari Senin di ruang 7609 dan kelas di jam 9-10 hari Selasa di ruang multimedia.

Contoh Input

```
{
  "kelas_mata_kuliah": [
    {
      "kode": "IF3071_K01",
      "jumlah_mahasiswa": 60,
      "skls": 3
    },
    {
      "kode": "IF3130_K01",
      "jumlah_mahasiswa": 45,
      "skls": 2
    },
    {
      "kode": "IF3110_K02",
      "jumlah_mahasiswa": 70,
      "skls": 3
    },
    {
      "kode": "IF3140_K01",
      "jumlah_mahasiswa": 55,
      "skls": 2
    }
  ],
  "ruangan": [
    {
      "kode": "7609",
      "kuota": 60
    },
    {
      "kode": "7606",
      "kuota": 80
    }
  ]
}
```

```

    },
    {
      "kode": "multimedia",
      "kuota": 40
    }
  ],
  "mahasiswa": [
    {
      "nim": "13523601",
      "daftar_mk": ["IF3071_K01", "IF3130_K01"],
      "prioritas": [1, 2]
    },
    {
      "nim": "135236641",
      "daftar_mk": ["IF3110_K02", "IF3130_K01"],
      "prioritas": [1, 2]
    },
    {
      "nim": "13523669",
      "daftar_mk": ["IF3140_K01", "IF3071_K01"],
      "prioritas": [1, 2]
    },
    {
      "nim": "13523600",
      "daftar_mk": ["IF3110_K02"],
      "prioritas": [1]
    }
  ]
}

```

Contoh Output

Kode ruang: 7609

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7				IF3140	
8	IF3130		IF3110		IF3130
9	IF3130	IF3140	IF3110		
10		IF3140	IF3110		
11					
12					
13					
14					

15					
16					
17					

*kolom jam di output merupakan jam mulai

*contoh di atas merupakan output untuk satu ruangan.

Objective Function

Objective function yang digunakan **dibebaskan** (minimal 1), namun tetap berpedoman pada acuan berikut:

1. **Jumlah Waktu yang bertabrakan untuk tiap Mahasiswa**

Misalnya, mahasiswa A memiliki 2 mata kuliah di jam 9-11 hari Selasa akan berkontribusi 2 poin ke state tersebut.

2. **Jumlah Pertemuan Beberapa Mata Kuliah yang bertabrakan di Ruangan dan Waktu tertentu dikali dengan bobot prioritas Mata Kuliah tiap peserta kelas**

Misalnya, IF2230_K01 dan IF1221_K01 sama-sama dilaksanakan di ruangan 7606 pada jam 13 - 14 . Bobot prioritas matkul adalah sebagai berikut.

Prioritas	Bobot
1	1.75
2	1.5
3	1.25
dll	1

Peserta kelas IF2330_K01 adalah sebagai berikut.

```
{
  "nim": "13523601",
  "daftar_mk": ["IF2330_K01"],
  "prioritas": [1]
},
{
  "nim": "135236641",
  "daftar_mk": ["IF2330_K01"],
  "prioritas": [2]
}
```

Sedangkan peserta kelas IF1221_K01 adalah sebagai berikut.

```
{  
    "nim": "13523701",  
    "daftar_mk": ["IF1221_K01"],  
    "prioritas": [1]  
},  
{  
    "nim": "135237641",  
    "daftar_mk": ["IF1221_K01"],  
    "prioritas": [3]  
}
```

Maka state valuenya akan bertambah/berkurang (tergantung implementasi) sebanyak
1 jam * (1.75 bobot prioritas 1 * 2 jumlah mahasiswa prioritas 1 + 1.5 bobot prioritas 2 *
1 jumlah mahasiswa prioritas 2 + 1.25 bobot prioritas 3 * 1 jumlah mahasiswa prioritas 3
+ 1 bobot prioritas lainnya * 0 jumlah mahasiswa prioritas lainnya) = 6.25

3. Selisih Jumlah Mahasiswa di Pertemuan Mata Kuliah tertentu dengan Kuota

Ruangan tersedia (dalam kasus Jumlah Mahasiswa > Kuota Ruangan)

Misalnya terdapat 65 mahasiswa di ruangan 7609 (Kuota Ruangan: 60) di suatu
Pertemuan Mata Kuliah **2** SKS akan berkontribusi 10 poin $((65 - 60) * 2)$ ke state
tersebut.

PEMBAHASAN

Pemilihan Objective Function

Dalam tugas penjadwalan mata kuliah ini, diperlukan suatu fungsi tujuan yang digunakan untuk mengukur seberapa baik atau efektifnya suatu penjadwalan. Fungsi tujuan tersebut nantinya dapat dimanfaatkan dalam algoritma optimasi seperti *Hill Climbing*, *Simulated Annealing*, dan *Genetic Algorithm* untuk membantu menemukan solusi penjadwalan yang optimal.

Berdasarkan diskusi dengan kelompok, kami memutuskan untuk menggunakan 3 buah fungsi tujuan dimana total nilai dari 3 fungsi tersebut akan dijadikan pertimbangan dipilih atau tidaknya suatu penjadwalan dalam masing-masing algoritma. Berikut fungsi tujuannya:

1. Selisih jumlah mahasiswa terhadap kapasitas ruangan

Fungsi ini dipilih dengan harapan hasil penjadwalan untuk masing-masing matkul selalu terpenuhi kebutuhan minimum kuotanya

2. Jumlah matkul yang bertabrakan dalam satu waktu untuk 1 mahasiswa

Untuk meminimalkan konflik jadwal antar mata kuliah yang ada

3. Jumlah pertemuan mata kuliah yang bertabrakan di ruangan dan waktu tertentu yang dikalikan dengan bobot prioritas

Implementasi

Steepest Ascent Hill Climbing

Implementasi	Penjelasan
<pre>hillclimbing.py class SteepestAscentHC: def __init__(self, stateAwal : State): self.stateAwal = stateAwal def findBestSuccessor(self, successors : list): minSuccessorObjFunction = 99999999 selectedSuccessor = None for successor in successors: if successor.countObjective() < minSuccessorObjFunction: selectedSuccessor = successor minSuccessorObjFunction = successor.countObjective() return selectedSuccessor</pre>	Definisi class dari <i>Steepest Ascent Hill Climbing</i> (<i>SteepestAscentHC</i>) dengan atributnya, yakni state awal ditunjukkan melalui constructor <code>_init_</code> nya dan fungsi untuk mengambil successor terbaik dari semua successor yang di-generate

```

hillclimbing.py

def solve(self):
    current = self.stateAwal
    currentScore = self.stateAwal.countObjective()
    plotObjFunc = []
    iterationCount = 0

    start_time = time.perf_counter()
    while (True):
        iterationCount += 1

        successors1 = current.swapMethod()
        successors2 = current.moveAllSuccessorMethod()

        neighbor1 = self.findBestSuccessor(successors1)
        neighbor2 = self.findBestSuccessor(successors2)

        # Bandingkan nilai cost / objective
        if neighbor1.countObjective() < neighbor2.countObjective():
            neighbour = neighbor1
        else:
            neighbour = neighbor2

        neighbourScore = neighbour.countObjective()

        # keluarkan hasil ketika tidak ada neighbour yang lebih rendah dengan nilai obj func
        if (neighbourScore ≥ currentScore):
            plotObjFunc.append(currentScore)
            break
        else:
            current = neighbour
            currentScore = neighbourScore
            plotObjFunc.append(currentScore)

    end_time = time.perf_counter()
    elapsed_time = end_time - start_time
    return copy.deepcopy(current), currentScore, plotObjFunc, iterationCount, elapsed_time

```

Fungsi `solve()` dari class `SteepestAscentHC` menjalankan proses utama algoritma *Steepest Ascent Hill Climbing* dengan melakukan proses pembaruan solusi state, yakni dari state `current` berdasarkan perbandingan nilai fungsi objektif dengan nilai fungsi objektif `neighbour`nya hingga tidak ada `neighbour` yang memiliki nilai fungsi objektif lebih baik. Pemilihan `neighbour` sendiri dilakukan dengan membandingkan semua successor dari tiap jenis perpindahan (`swap` dan `move`) yang kemudian masing-masing dipilih successor terbaik kemudian `neighbour` dipilih berdasarkan satu dari dua successor terpilih dengan nilai fungsi objektif lebih baik.

Sideways Move Hill Climbing

Implementasi	Penjelasan
<pre> hillclimbing.py class SidewaysMoveHC: def __init__(self, stateAwal : State, maxSidewaysMove : int): self.stateAwal = stateAwal self.maxSidewaysMove = maxSidewaysMove def findBestSuccessor(self, successors : list): minSuccessorObjFunction = 99999999 selectedSuccessor = None for successor in successors: if successor.countObjective() < minSuccessorObjFunction: selectedSuccessor = successor minSuccessorObjFunction = successor.countObjective() return selectedSuccessor </pre>	<p>Definisi class dari <i>Sideways Move Hill Climbing</i> (<code>SidewaysMoveHC</code>) dengan atributnya, yakni <code>state awal</code> dan nilai maksimal sideways move nya (<code>maxSidewaysMove</code>) ditunjukkan melalui constructor <code>_init_</code> nya dan fungsi untuk mengambil successor terbaik dari semua successor yang di-generate</p>

```

●●● hillclimbing.py
def solve(self):
    current = self.stateAwal
    currentScore = self.stateAwal.countObjective()
    iterationCount = 0
    sidewaysMove = 0
    plotObjFunc = []

    start_time = time.perf_counter()
    while (True):
        iterationCount += 1

        successors1 = current.swapMethod()
        successors2 = current.moveAllSuccessorMethod()

        neighbor1 = self.findBestSuccessor(successors1)
        neighbor2 = self.findBestSuccessor(successors2)

        # Bandingkan nilai cost / objective
        if neighbor1.countObjective() < neighbor2.countObjective():
            neighbour = neighbor1
        else:
            neighbour = neighbor2

        neighbourScore = neighbour.countObjective()

        # keluarkan hasil ketika tidak ada neighbour yang lebih rendah sama dengan nilai obj func
        if (neighbourScore > currentScore):
            plotObjFunc.append(currentScore)
            break
        else:
            # jumlah sideways move bertambah jika nilai obj func sama
            if (neighbourScore == currentScore):
                sidewaysMove += 1

            current = neighbour
            currentScore = neighbourScore
            plotObjFunc.append(currentScore)
            if (sidewaysMove == self.maxSidewaysMove):
                break

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

    return copy.deepcopy(current), currentScore, plotObjFunc, iterationCount, elapsed_time

```

Fungsi solve() dari class SidewaysMoveHC menjalankan proses utama algoritma *Sideways Move Hill Climbing* dengan proses yang mirip seperti pada fungsi solve() pada StochasticHC, bedanya pada fungsi solve() di class ini solusi state sekarang, yakni current, akan berpindah juga ke state neighbour apabila neighbour memiliki nilai objektif yang sama dengan yang dimiliki state current yang dalam hal ini disebut sebagai sideways move. Selain itu, proses pembaruan solusi state ini hanya dilakukan selama jumlah sideways move tidak melebihi batas maksimal sideways move-nya

Random Restart Climbing

Implementasi	Penjelasan
<pre> ●●● hillclimbing.py class RandomRestartHC: def __init__(self, listRuangan : list, listMatkul : list, listMahasiswa : list, maxRestart: int): self.listRuangan = listRuangan self.listMatkul = listMatkul self.listMahasiswa = listMahasiswa self.maxRestart = maxRestart def findBestSuccessor(self, successors : list): minSuccessorObjFunction = 99999999 selectedSuccessor = None for successor in successors: if successor.countObjective() < minSuccessorObjFunction: selectedSuccessor = successor minSuccessorObjFunction = successor.countObjective() return selectedSuccessor </pre>	<p>Definisi class dari Random Restart Climbing (RandomRestartHC) dengan atributnya, yakni listRuangan, listMatkul, listMahasiswa yang akan digunakan untuk menginisialisasi state awal tiap restart dan atribut maksimum restart nya (maxRestart) yang ditunjukkan melalui constructor __init__ nya. Di sini juga terdapat fungsi yang</p>

	memilih successor terbaik dari semua successor yang di- <i>generate</i> dari suatu aksi perpindahan state
<pre> ••• def solve(self): restartCount = 0 initialStates = [] plotObjFunc, iterationCountList = [], [] # Obj func keseluruhan tiap restart start_time = time.perf_counter() bestState = None bestScore = 9999999 while (restartCount < self.maxRestart and bestScore > 0): # initial random state current = State(self.listRuangan, self.listMahasiswa) current.makeComplete(self.listMatkul) currentScore = current.countObjective() initialStates.append(current) if (restartCount == 0): bestState = current subPlotObjFunc = [] iterationCount = 0 while (True): iterationCount += 1 successors1 = current.swapMethod() successors2 = current.moveAllSuccessorMethod() neighbor1 = self.findBestSuccessor(successors1) neighbor2 = self.findBestSuccessor(successors2) # Bandingkan nilai cost / objective if neighbor1.countObjective() < neighbor2.countObjective(): neighbour = neighbor1 else: neighbour = neighbor2 neighbourScore = neighbour.countObjective() # Buat hasil ketika tidak ada neighbour yang lebih rendah nilai obj func if (neighbourScore >= currentScore): bestScore = currentScore subPlotObjFunc.append(currentScore) break else: current = neighbour currentScore = neighbourScore subPlotObjFunc.append(currentScore) # Mengambil state terbaik saat ini dengan membandingkan nilai # obj func dari hasil state restart if (bestScore > currentScore): bestState = current # Masukkan obj func per iterasi dalam satu restart ke plot obj iterationCountList.append(iterationCount) plotObjFunc.append(subPlotObjFunc) restartCount += 1 end_time = time.perf_counter() elapsed_time = end_time - start_time return initialStates, copy.deepcopy(bestState), bestScore, plotObjFunc, iterationCountList, restartCount, elapsed_time </pre>	Fungsi <code>solve()</code> dari class <code>RandomRestartHC</code> berfungsi untuk menjalankan proses utama algoritma <i>Random Restart Climbing</i> . Pada dasarnya, fungsi <code>solve</code> di sini didasarkan pada proses <i>Steepest Ascent Hill Climbing</i> yang dikembangkan algoritma nya untuk dilakukan berulang kali (restart) hingga mencapai maksimum jumlah restart. Setiap restart akan melakukan pemrosesan <code>solve()</code> seperti pada class <code>SteepestAscentHC</code> yang solusi state dari tiap restart akan dibandingkan berdasarkan nilai objektifnya untuk dipilih solusi state yang terbaik

Stochastic Hill Climbing

Implementasi	Penjelasan
<pre>hillclimbing.py class StochasticHC: def __init__(self, stateAwal : State, jumlahIterasi: int): self.stateAwal = stateAwal self.jumlahIterasi = jumlahIterasi</pre>	Definisi class dari Stochastic Hill Climbing (stochasticHC) dengan atributnya, yakni state awal dan jumlah iterasi nya yang ditunjukkan melalui constructor <code>_init_</code> nya
<pre>hillclimbing.py def solve(self): current = self.stateAwal currentScore = self.stateAwal.countObjective() plotObjFunc = [] start_time = time.perf_counter() for i in range(self.jumlahIterasi): listTuple = current.serialize() neighbor1 = current.swapSatuMatkul(listTuple) neighbor2 = current.moveOneSuccessorMethod() # Bandingkan nilai cost / objective if neighbor1.countObjective() < neighbor2.countObjective(): neighbour = neighbor1 else: neighbour = neighbor2 neighbourScore = neighbour.countObjective() # Pindah ke neighbour jika nilai obj func nya lebih rendah if neighbourScore < currentScore: current = neighbour currentScore = neighbourScore plotObjFunc.append(currentScore) else: plotObjFunc.append(currentScore) end_time = time.perf_counter() elapsed_time = end_time - start_time return copy.deepcopy(current), currentScore, plotObjFunc, elapsed_time</pre>	Fungsi <code>solve()</code> dari class <code>StochasticHC</code> berfungsi untuk menjalankan proses utama algoritma <i>Stochastic Hill Climbing</i> .melakukan proses pembaruan solusi state berdasarkan pemilihan <i>neighbour</i> terbaik sebanyak jumlah Iterasi. Pemilihan <i>neighbour</i> dilakukan dengan membandingkan dua state perpindahan (swap dan move) yang masing-masing dipilih secara random kemudian memilih state yang menjadi <i>neighbour</i> berdasarkan nilai fungsi objektif terbaiknya. Setelah itu, nilai fungsi objektif dari state current dibandingkan dengan state neighbour terpilih. Jika nilai fungsi objektif neighbour lebih baik, maka state current akan berpindah ke state neighbour sebagai solusi terbaru. Hal ini dilakukan berulang kali hingga mencapai iterasi ke jumlahIterasi - 1.

Simulated Annealing

Implementasi	Penjelasan
<pre>class SimulatedAnnealing: def __init__(self, stateAwal : State, jumlahIterasi: int, temperature: int): self.stateAwal = stateAwal self.jumlahIterasi = jumlahIterasi self.temperature = temperature self.coolingRate = 0.985</pre>	<p>Ini adalah kelas dari algoritma SA dimana ia akan memiliki atribut cooling rate, temperatur dan batas maksimum iterasinya.</p>
<pre>def solve(self): plotObjFunc = [] plotExp = [] current = self.stateAwal currentScore = self.stateAwal.countObjective() plotObjFunc.append([currentScore]) start_time = time.perf_counter() for i in range(self.jumlahIterasi): if currentScore == 0: break listTuple = current.serialize() neighbor1 = current.swapSatuMatkul(listTuple) neighbor2 = current.moveOneSuccessorMethod() # Bandingkan nilai cost / objective if neighbor1.countObjective() < neighbor2.countObjective(): neighbour = neighbor1 else: neighbour = neighbor2 neighbourScore = neighbour.countObjective() deltaE = neighbourScore - currentScore #better if deltaE < 0: current = neighbour currentScore = neighbourScore prob = 1.0 else: prob = math.exp(-deltaE / max(self.temperature, 1e-8)) if random.random() < prob: current = neighbour currentScore = neighbourScore plotExp.append((i,prob)) self.temperature = self.temperature * self.coolingRate plotObjFunc.append(currentScore) end_time = time.perf_counter() elapsed_time = end_time - start_time localOptimum = False N = min(10, len(plotObjFunc)) if currentScore != 0: #kalau 10 trakhir atau n buah obj terakhir nilainya sama local optimum if len(set(plotObjFunc[-N:])) == 1: localOptimum = True return copy.deepcopy(current), currentScore, plotObjFunc, plotExp, elapsed_time, localOptimum</pre>	<p>Fungsi solve() berfungsi untuk menjalankan proses utama algoritma <i>Simulated Annealing</i>. Pada algoritma ini, solusi awal akan diperbaiki secara bertahap dengan cara memilih satu <i>neighbour</i> (tetangga) dari solusi saat ini. Namun, sebelum memilih <i>neighbour</i> tersebut, algoritma terlebih dahulu membangkitkan dua buah state jadwal secara acak, masing-masing menggunakan metode swapSatuMatkul() dan moveOneSuccessorMethod(), yang merepresentasikan dua jenis perpindahan berbeda dalam ruang solusi. Setelah itu, nilai fungsi objektif keduanya dibandingkan dan yang lebih besar akan dijadikan <i>neighbour</i> jika <i>neighbour.value</i> > <i>current.value</i> jika tidak maka akan dihitung probabilitas eksponennya untuk menentukan apakah akan dipilih atau tidak.</p>

Genetic Algorithm

Implementasi	Penjelasan
<pre> 1 # Mengembalikan sebuah individual (objek State) secara random dari populasi 2 # population: List of State 3 @staticmethod 4 def random_selection(population): 5 fitness = [] 6 7 # Rumus: fitness = 1 / (obj + 1) 8 for individual in population: 9 obj = individual.countObjective() 10 fitness.append(1 / (obj + 1)) 11 12 total_value = 0 13 for value in fitness: 14 total_value += value 15 16 probability = [] 17 18 sum_percentage = 0 19 for value in fitness: 20 percentage = round((value / total_value) * 100) 21 sum_percentage += percentage 22 probability.append(sum_percentage) 23 24 # batas atasnya probability[-1] karena rounding issue, terkadang cuman sampe 99 bukan 100 25 random_num = random.randint(1, probability[-1]) 26 27 i = 0 28 is_found_chosen_individual = False 29 while not is_found_chosen_individual: 30 if random_num <= probability[i]: 31 is_found_chosen_individual = True 32 chosen_individual = population[i] 33 i += 1 34 35 return copy.deepcopy(chosen_individual)</pre>	Fungsi random_selection() Memilih individu secara random untuk menjadi parent. Probabilitas keterpilihan individu proporsional dengan nilai fitness function individu tersebut
<pre> 1 # Mengembalikan State (1 child) hasil crossover yang memiliki nilai fitness terbaik 2 # parent_1, parent_2: State 3 @staticmethod 4 def reproduce(parent_1, parent_2): 5 serialized_parent_1 = parent_1.serialize() 6 serialized_parent_2 = parent_2.serialize() 7 8 crossover_point = random.randint(0, len(serialized_parent_1) - 1) 9 10 child_1 = serialized_parent_1[:crossover_point] + serialized_parent_2[crossover_point:] 11 child_2 = serialized_parent_2[:crossover_point] + serialized_parent_1[crossover_point:] 12 13 deserialized_child_1 = copy.deepcopy(parent_1) 14 deserialized_child_2 = copy.deepcopy(parent_2) 15 16 deserialized_child_1.deserialize(child_1) 17 deserialized_child_2.deserialize(child_2) 18 19 objective_value_1 = deserialized_child_1.countObjective() 20 fitness_value_1 = 1 / (objective_value_1 + 1) 21 22 objective_value_2 = deserialized_child_2.countObjective() 23 fitness_value_2 = 1 / (objective_value_2 + 1) 24 25 if fitness_value_1 > fitness_value_2: 26 return deserialized_child_1 27 else: 28 return deserialized_child_2</pre>	Fungsi reproduce() Melakukan crossover antara dua individu (parent) lalu memilih child dengan nilai fitness function terbaik. Crossover point ditentukan secara random.

Fungsi mutate()

Melakukan mutasi pada individu. Sebuah slot yang dipilih secara random milik individu tersebut akan diberikan waktu dan ruangan baru yang juga dipilih secara random

```
1 # Mutasi State/individual dengan pick random sebuah tuple {hari, slot, Matkul} lalu randomize hari, slot, dan ruangan
2 # Mengembalikan individual hasil random mutation
3 # listRuangan = (Opsional) List of Ruangan
4 @staticmethod
5 def mutate(individual, listRuangan=None):
6     serialized_individual = individual.serialize()
7
8     mutation_point = random.randint(0, len(serialized_individual) - 1)
9
10    hari = ["senin", "selasa", "rabu", "kamis", "jumat"]
11    randomized_hari = hari[random.randint(0, 4)]
12    randomized_slot = random.randint(0, 10)
13
14    randomized_ruangan = None
15    if listRuangan is not None:
16        randomized_ruangan = listRuangan[random.randint(0, len(listRuangan) - 1)]
17
18    randomized_tuple = (randomized_hari, randomized_slot, serialized_individual[mutation_point][2])
19    serialized_individual.pop(mutation_point)
20
21    serialized_individual.insert(mutation_point, randomized_tuple)
22
23    if randomized_ruangan is not None:
24        serialized_individual[mutation_point][2].setRuangan(randomized_ruangan)
25
26    deserialized_individual = copy.deepcopy(individual)
27    deserialized_individual.deserialize(serialized_individual)
28
29 return deserialized_individual
```

Fungsi GA()

Fungsi utama yang melakukan genetic algorithm

```
● ● ●
1 # Function genetic algorithm
2 # Akan berhenti ketika ada individu yang memiliki nilai fitness function maximum atau tercapai iterasi maksimal
3 # k = banyaknya iterasi maksimal yang dapat dilakukan
4 # n = banyaknya individu dalam populasi
5 def GA(self, k, n):
6     start_time = time.perf_counter()
7
8     start_population = []
9
10    for i in range(k):
11        individual = State(self.listRuangan, self.listMahasiswa)
12        individual.makeComplete(self.listMatkul)
13        start_population.append(individual)
14
15    population = copy.deepcopy(start_population)
16    plot = []
17
18    max = -1
19    sum = 0
20    avg = 0
21
22    for individual in population:
23        # Rumus: fitness = 1 / (obj + 1) -> Nilai fitness function = [0, 1], 1 terbaik
24        objective_value = individual.countObjective()
25        fitness_value = 1 / (objective_value + 1)
26
27        if (fitness_value > max):
28            max = fitness_value
29
30        sum = sum + fitness_value
31
32    avg = sum / k
33
34    data = [max, avg]
35    plot.append(data)
36
37    is_individual_fit = False
38    if max == 1:
39        is_individual_fit = True
40
41    itr = 0
42    while itr < n and not is_individual_fit:
43        new_population = []
44        for i in range(k):
45            parent_1 = genetic_algorithm.random_selection(population)
46            parent_2 = genetic_algorithm.random_selection(population)
47            child = genetic_algorithm.reproduce(parent_1, parent_2)
48            child = genetic_algorithm.mutate(child, self.listRuangan)
49            new_population.append(child)
50
51    population = new_population
52
53    max = -1
54    sum = 0
55    avg = 0
56
57    for individual in population:
58        # Rumus: fitness = 1 / (obj + 1) -> Nilai fitness function = [0, 1], 1 terbaik
59        objective_value = individual.countObjective()
60        fitness_value = 1 / (objective_value + 1)
61
62        if (fitness_value > max):
63            max = fitness_value
64
65        sum = sum + fitness_value
66
67    avg = sum / k
68
69    data = [max, avg]
70    plot.append(data)
71
72    if max == 1:
73        is_individual_fit = True
74
75    itr = itr + 1
76
77    max = -1
78    fittest_individual = population[0]
79
80    for individual in population:
81        # Rumus: fitness = 1 / (obj + 1) -> Nilai fitness function = [0, 1], 1 terbaik
82        objective_value = individual.countObjective()
83        fitness_value = 1 / (objective_value + 1)
84
85        if (fitness_value > max):
86            max = fitness_value
87            fittest_individual = individual
88
89    end_time = time.perf_counter()
90    elapsed_time = end_time - start_time
91
92    data = (start_population, population, plot, k, n, elapsed_time, fittest_individual)
93
94    return data
```

Eksperimen

Steepest Ascent Hill Climbing

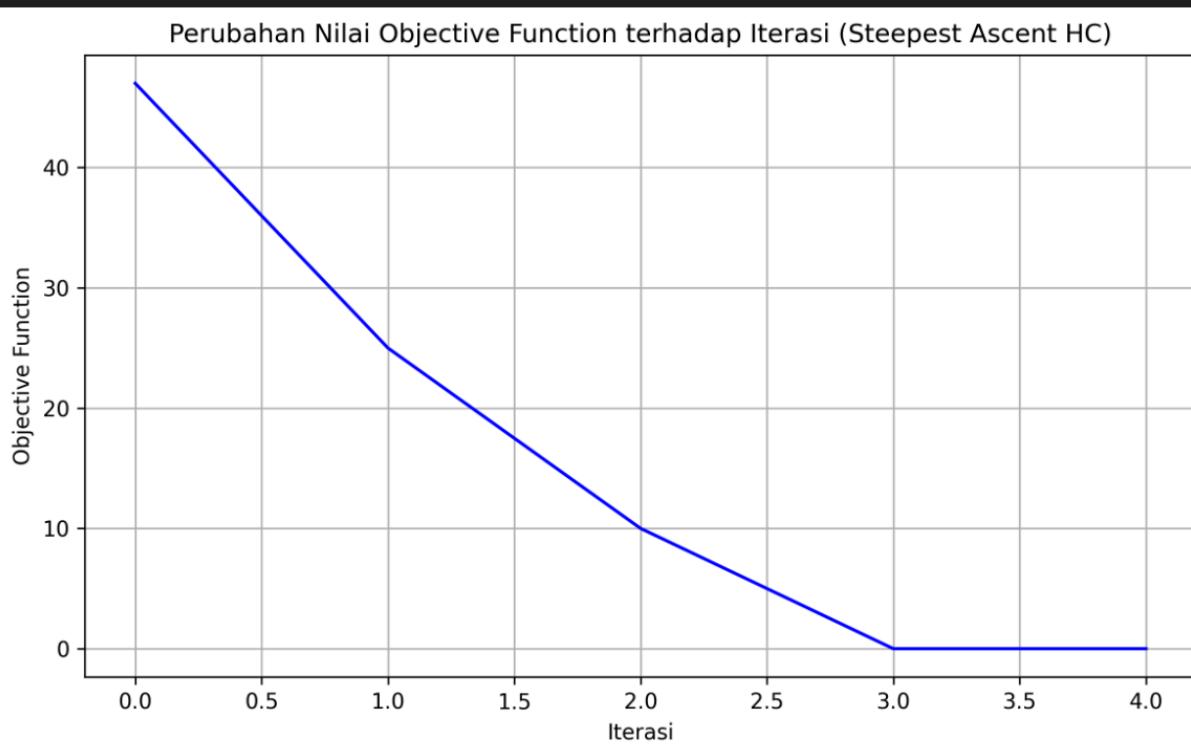
File: input.json

State Awal					
+	+	+	+	+	+
	Jam	Senin	Selasa	Rabu	Kamis
+	-----+-----+-----+-----+-----+-----+				
	7				
+	-----+-----+-----+-----+-----+-----+				
	8				IF3071_K01 @ multimedia IF3140_K01 @ 7609
+	-----+-----+-----+-----+-----+-----+				
	9				
+	-----+-----+-----+-----+-----+-----+				
	10		IF3110_K02 @ 7606		
+	-----+-----+-----+-----+-----+-----+				
	11				
+	-----+-----+-----+-----+-----+-----+				
	12		IF3071_K01 @ 7609		IF3071_K01 @ 7606
+	-----+-----+-----+-----+-----+-----+				
	13				
+	-----+-----+-----+-----+-----+-----+				
	14				IF3110_K02 @ multimedia
+	-----+-----+-----+-----+-----+-----+				
	15			IF3130_K01 @ 7609	IF3110_K02 @ 7609
+	-----+-----+-----+-----+-----+-----+				
	16				
+	-----+-----+-----+-----+-----+-----+				
	17			IF3130_K01 @ 7606	IF3140_K01 @ multimedia
+	-----+-----+-----+-----+-----+-----+				
State Akhir			Objective Function Akhir = 0		

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	IF3071_K01 @ 7606				
8	IF3140_K01 @ 7606			IF3140_K01 @ 7609	
9	IF3110_K02 @ 7606				
10		IF3110_K02 @ 7606			
11					
12		IF3071_K01 @ 7609			IF3071_K01 @ 7606
13	IF3110_K02 @ 7606				
14					
15				IF3130_K01 @ 7609	
16					
17			IF3130_K01 @ 7606		

Durasi = 0.3295888000866398 s

Banyak iterasi hingga mencapai proses berhenti = 5 (0 ... 4)



File: input1.json

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7				IF3170 @ 7609 IF3130 @ 7602 IF3140 @ 9018	
8				IF3130 @ 7609 IF3110 @ 7609 IF2224 @ 9018 IF3130 @ 9018	WI2022 @ 9018
9					
10		IF3110 @ 9018			KU5004 @ 9018
11					IF3170 @ 9018
12				IF3140 @ 7609	
13	IF3140 @ 7602			IF3110 @ 7602 WI2022 @ 7609	
14					
15	IF2224 @ 7609				
16		IF3170 @ 7602			
17		KU5004 @ 7609	IF3170 @ 7602		IF2224 @ 7602 IF2224 @ 7602

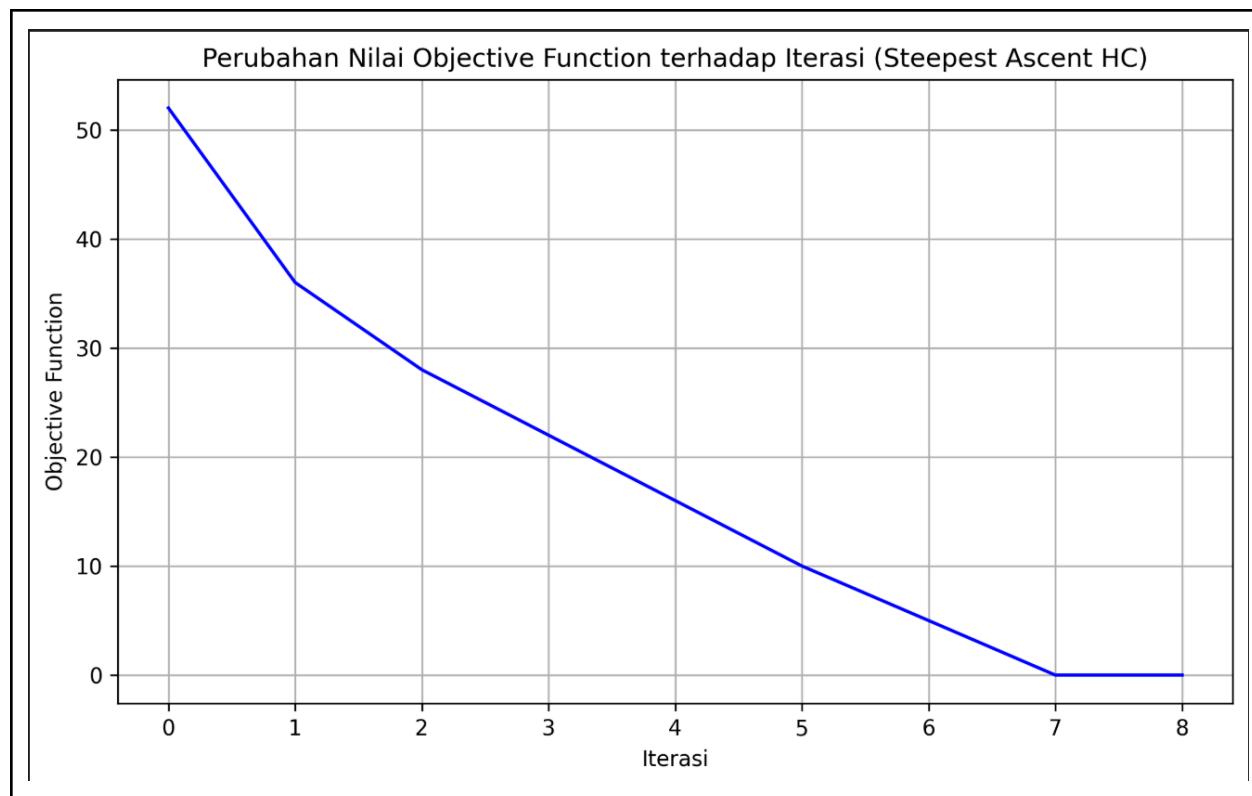
State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	KU5004 @ 9018			IF3170 @ 7609	IF3140 @ 9018
8	IF2224 @ 9018		IF2224 @ 9018	IF3130 @ 9018	WI2022 @ 9018
9	IF3110 @ 7602				
10	IF3130 @ 7609	IF3110 @ 9018		KU5004 @ 9018	
11	IF3110 @ 7602			IF3170 @ 9018	
12	IF3130 @ 9018		IF3140 @ 7609		
13	IF3140 @ 7602		WI2022 @ 7609		
14	IF3170 @ 9018				
15	IF2224 @ 7609				
16	IF3170 @ 9018				
17					IF2224 @ 7602

Durasi = 1.0256002000533044 s

Banyak iterasi hingga mencapai proses berhenti = 9



File: input2.json

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7					IF2224 @ 7602
8		IF3130 @ 7602 WI2022 @ 9018			
9		KU3001 @ 7609			IF2224 @ 9018 IF3140 @ 7609 KU3002 @ 7602
10					
11	KU5004 @ 9018 IF2224 @ 7602		KU3001 @ 7602		
12	IF3110 @ 9018 IF3170 @ 9018	IF2224 @ 7609	WI2022 @ 7609		IF3110 @ 7602
13		KU5004 @ 7609		IF3140 @ 7602	
14	IF3110 @ 7609		IF3170 @ 7602 IF3130 @ 7609		
15	IF3140 @ 9018				IF3170 @ 7609
16				IF3170 @ 7602	
17	IF3130 @ 9018				KU3002 @ 9018

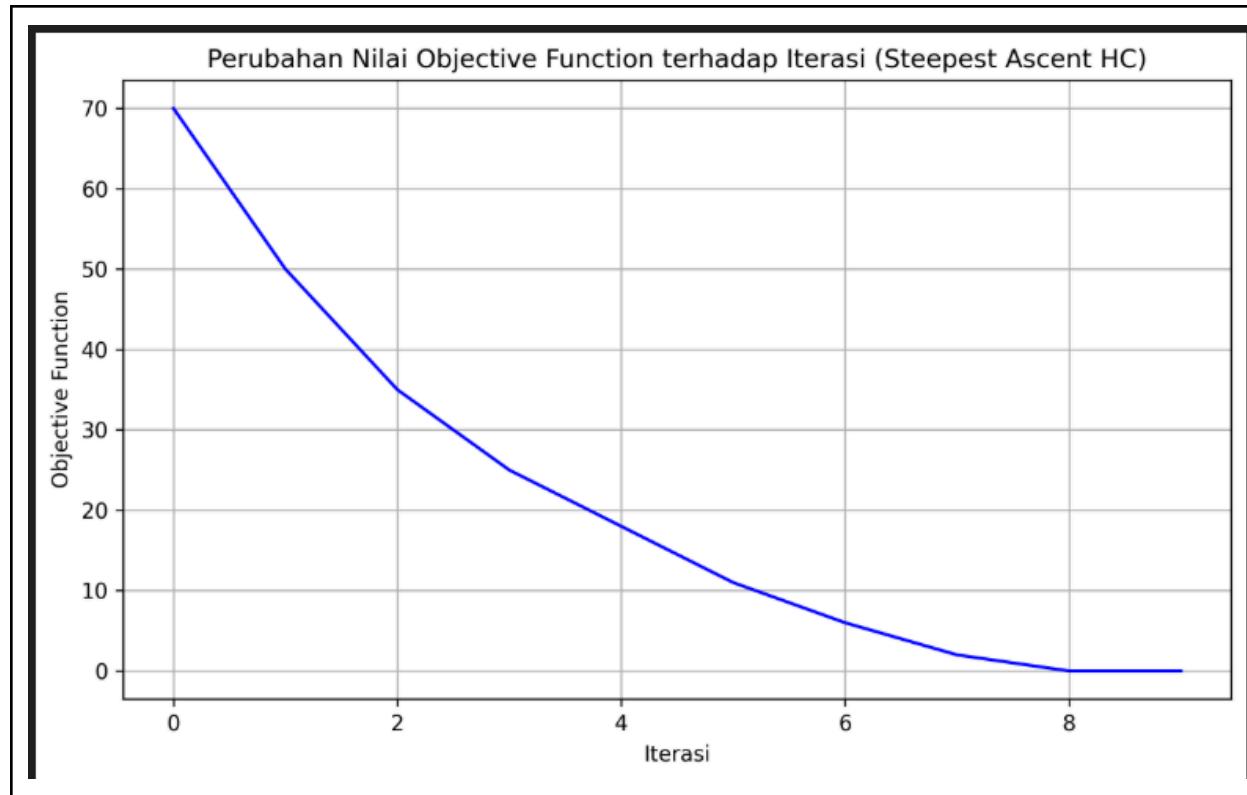
State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7		KU5004 @ 9018	IF3140 @ 7602		
8			KU3001 @ 9018	IF2224 @ 7602	
				IF3130 @ 7609	
9		IF3140 @ 9018	IF3130 @ 7602		WI2022 @ 9018
10	KU3001 @ 9018	IF3110 @ 7602		IF3110 @ 7602	
11		IF2224 @ 7602			
12		IF3130 @ 9018	IF3170 @ 7609	IF3170 @ 7609	
13		IF3170 @ 9018	IF2224 @ 7602		
			IF3170 @ 7602		
14			KU3002 @ 7609	KU3002 @ 9018	
15	IF3140 @ 7602		IF2224 @ 7602		
16	WI2022 @ 7609		IF3110 @ 9018	KU5004 @ 9018	
17					

Durasi = 6.1116043999791145 s

Banyak iterasi hingga mencapai proses berhenti: 10



Sideways Move Hill Climbing

File: input.json

Maximum Sideways Move = 20

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7					
8				IF3071_K01 @ multimedia IF3140_K01 @ 7609	
9					
10		IF3110_K02 @ 7606			
11					
12		IF3071_K01 @ 7609			IF3071_K01 @ 7606
13					
14					IF3110_K02 @ multimedia
15				IF3130_K01 @ 7609	IF3110_K02 @ 7609
16					
17			IF3130_K01 @ 7606	IF3140_K01 @ multimedia	

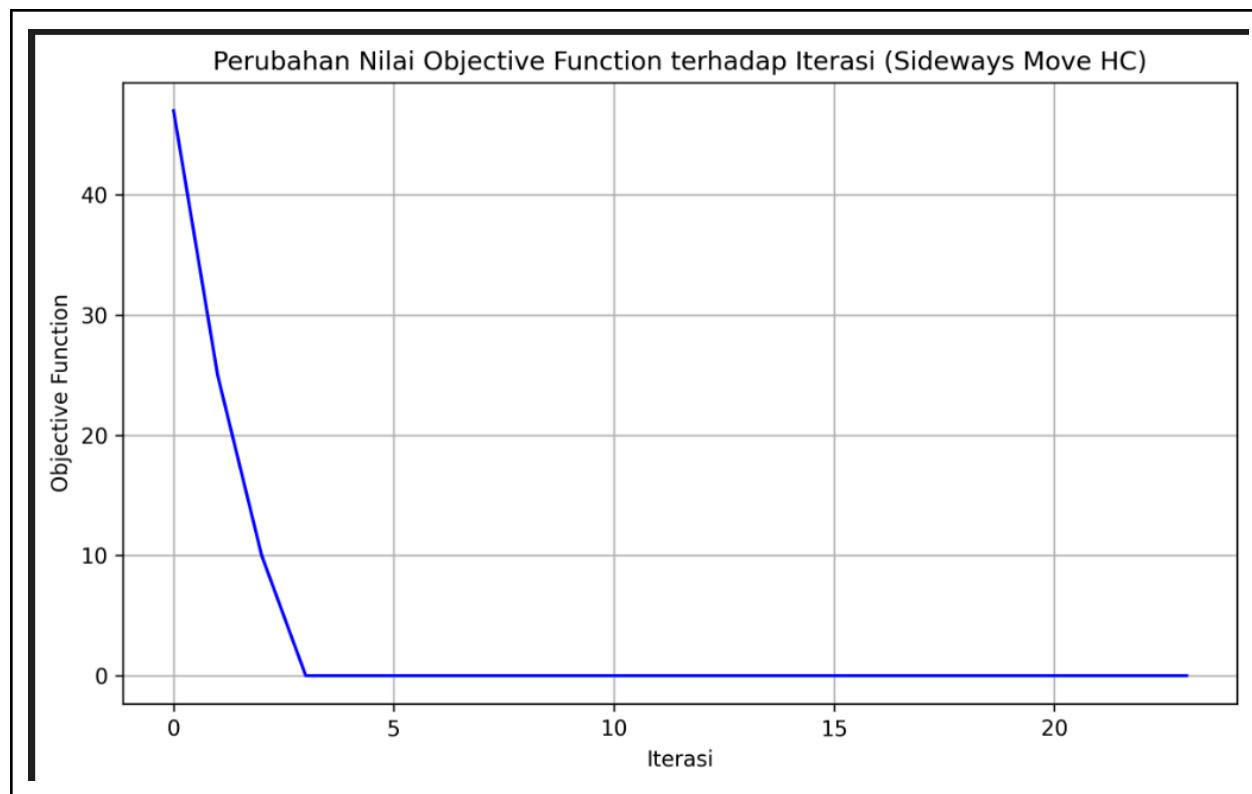
State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	IF3071_K01 @ 7609				
8		IF3110_K02 @ 7606		IF3140_K01 @ 7609	
9					
10		IF3110_K02 @ 7606			
11					
12		IF3071_K01 @ 7609			IF3071_K01 @ 7606
13					
14	IF3140_K01 @ 7606				
15	IF3110_K02 @ 7606			IF3130_K01 @ 7609	
16					
17			IF3130_K01 @ 7606		

Durasi = 1.5201694999122992 s

Banyak iterasi hingga mencapai proses berhenti = 24



File: input1.json

Maximum Sideways Move = 8

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7		IF2224 @ 7609		IF3170 @ 9018	
8					
9	IF2224 @ 7602				
10	IF3110 @ 7602 KU5004 @ 7609 IF2224 @ 7602 IF3140 @ 7602			IF3130 @ 9018	WI2022 @ 9018
11			IF3110 @ 7609		
12					
13		IF3170 @ 7602 KU5004 @ 9018	WI2022 @ 7609		
14			IF3110 @ 9018 IF3170 @ 7602		
15	IF2224 @ 9018				
16	IF3170 @ 7609 IF3140 @ 9018	IF3130 @ 7609	IF3130 @ 7602		
17	IF3140 @ 7609				

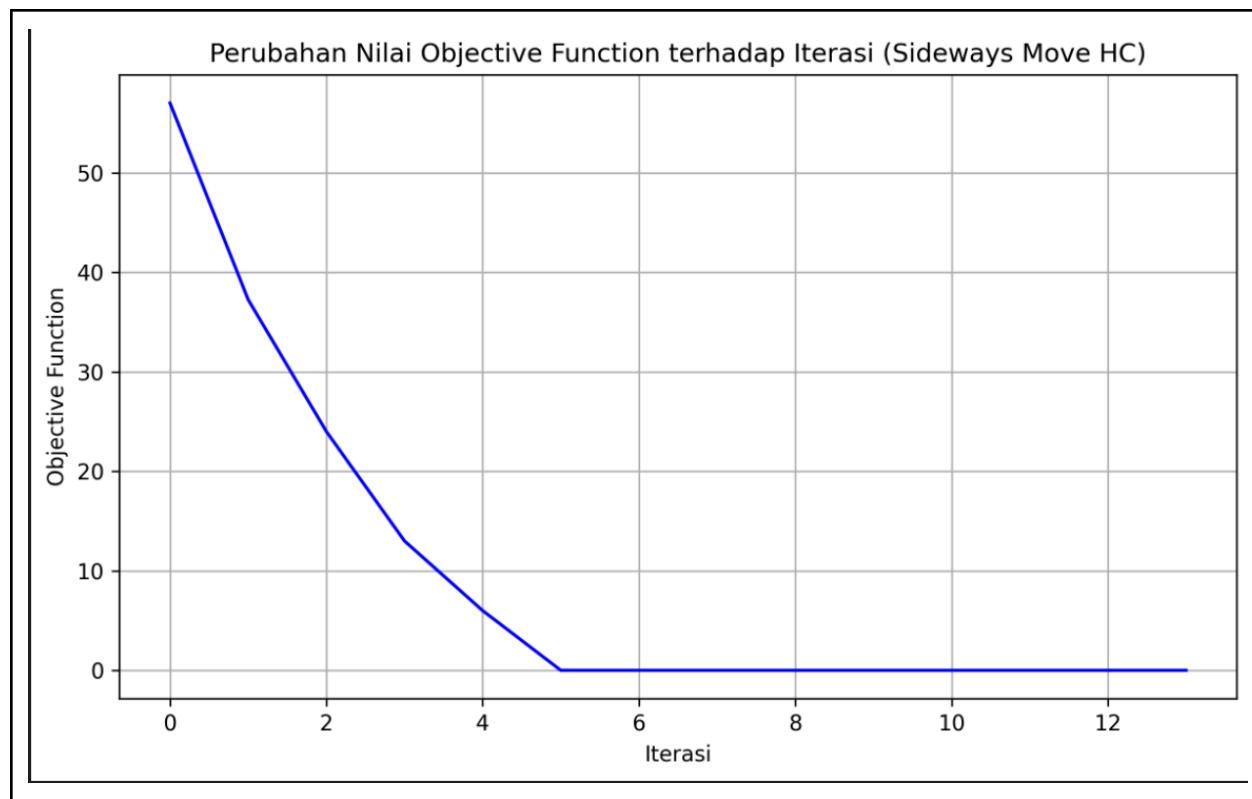
State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	IF3110 @ 7609	IF2224 @ 7609		IF3170 @ 9018	
8		IF3140 @ 7609			
9	IF2224 @ 7602				
10	IF2224 @ 7602			IF3130 @ 9018	WI2022 @ 9018
11	IF3170 @ 7609		IF3110 @ 7609		
12	KU5004 @ 9018				
13	IF3170 @ 9018	KU5004 @ 9018	WI2022 @ 7609		
14	IF3170 @ 7609		IF3110 @ 9018		
15	IF2224 @ 9018				
16	IF3140 @ 9018	IF3130 @ 7609	IF3130 @ 7602		
17	IF3140 @ 7609				

Durasi = 3.3637747999746352 s

Banyak iterasi hingga mencapai proses berhenti = 14



File: input2.json

Maximum Sideways Move = 6

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7			IF3130 @ 7602 WI2022 @ 9018	IF2224 @ 7609	IF3110 @ 7609
8			IF2224 @ 7602		
9		KU5004 @ 7609	IF3130 @ 9018 KU3002 @ 7602		WI2022 @ 7609
10			IF3170 @ 7602		
11		IF3110 @ 7602	IF3140 @ 7609	IF3110 @ 9018	
12		KU3001 @ 7609			
13		IF3170 @ 7609	KU3001 @ 7602 KU3002 @ 9018		IF2224 @ 9018
14					
15	IF3170 @ 9018		IF3170 @ 7602 IF3130 @ 7609		KU5004 @ 9018 IF3140 @ 9018
16					
17				IF3140 @ 7602	IF2224 @ 7602

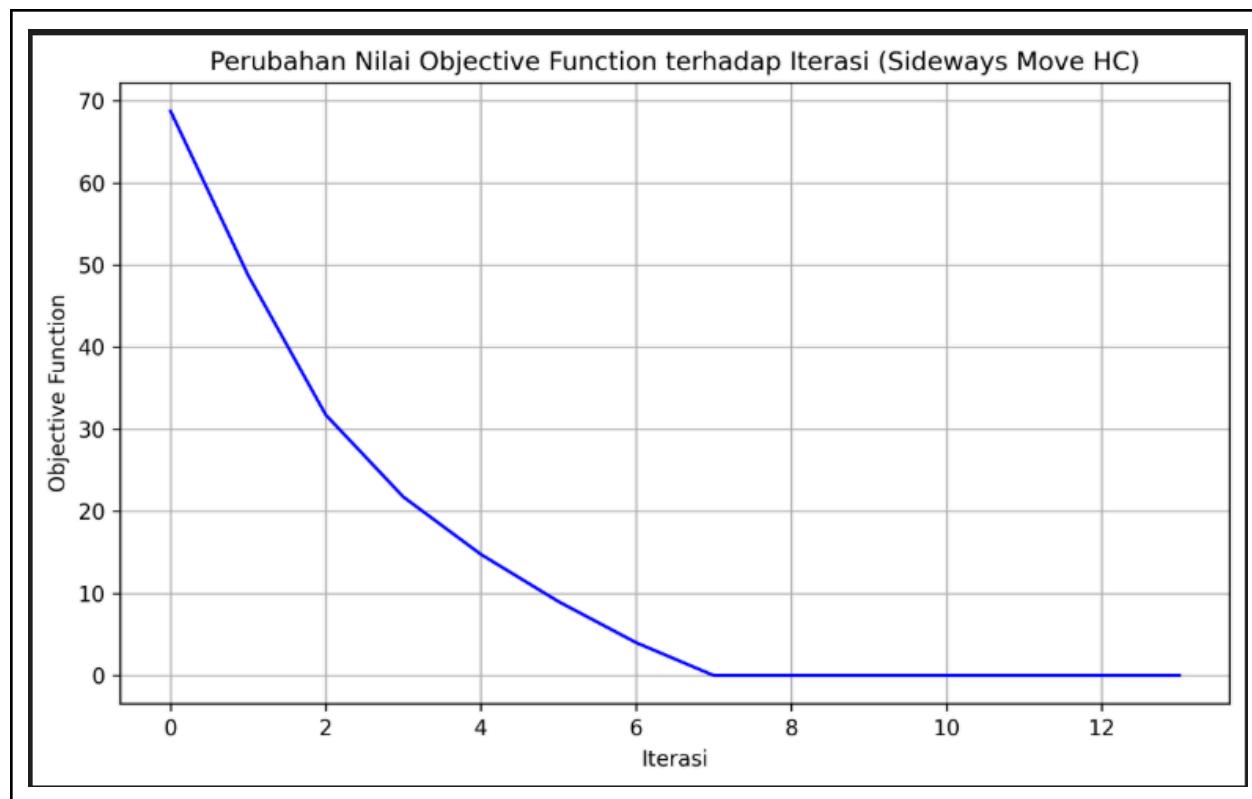
State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	KU3001 @ 9018		WI2022 @ 9018	IF2224 @ 7609	IF3110 @ 7609
8		KU5004 @ 9018	IF2224 @ 7602		
9			IF3130 @ 9018		WI2022 @ 7609
10		KU5004 @ 9018			
11	KU3001 @ 9018	IF3110 @ 7602	IF3140 @ 7609	IF3110 @ 9018	
12	IF3170 @ 7609				
13	IF3170 @ 7609	IF3170 @ 7609	KU3002 @ 9018		IF2224 @ 9018
14	IF3130 @ 7609				
15	IF3170 @ 9018		IF3130 @ 7609		IF3140 @ 9018
16	KU3002 @ 9018				
17				IF3140 @ 7602	IF2224 @ 7602

Durasi = 6.306304600089788 s

Banyak iterasi hingga mencapai proses berhenti = 14



Random Restart Hill Climbing

File: input.json

Maximum Restart = 5

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7					IF3110_K02 @ multimedia
8			IF3071_K01 @ 7606 IF3071_K01 @ multimedia		
9					IF3071_K01 @ 7609
10					
11		IF3110_K02 @ 7606		IF3140_K01 @ 7609	
12					
13					
14	IF3140_K01 @ multimedia				
15				IF3130_K01 @ 7606	
16		IF3130_K01 @ 7609			
17	IF3110_K02 @ 7609				

State Akhir

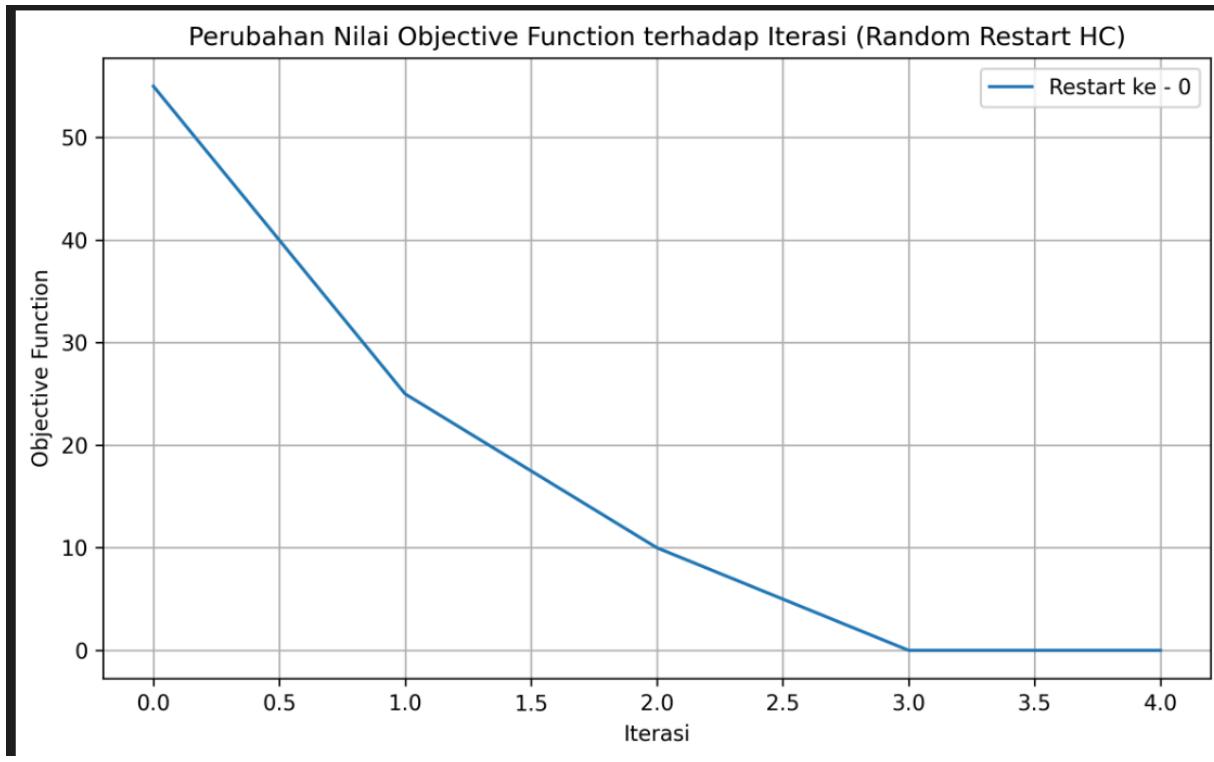
Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7					IF3110_K02 @ multimedia
8				IF3071_K01 @ 7606 IF3071_K01 @ multimedia	
9					IF3071_K01 @ 7609
10					
11		IF3110_K02 @ 7606		IF3140_K01 @ 7609	
12					
13					
14	IF3140_K01 @ multimedia				
15				IF3130_K01 @ 7606	
16		IF3130_K01 @ 7609			
17	IF3110_K02 @ 7609				

Durasi = 0.3352955000009388 s

Banyak Restart = 1

Banyak iterasi untuk restart index ke-0 = 5



File: input1.json

Maximum Restart = 5

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	IF3170 @ 7609		IF3170 @ 7602		
	IF3140 @ 9018				
8	WI2022 @ 9018		IF3170 @ 9018		IF2224 @ 7602
9	KU5004 @ 7609	IF2224 @ 7609	IF3130 @ 9018	KU5004 @ 9018	
10				IF2224 @ 9018	
				IF3140 @ 7609	
11					
12	IF3130 @ 7602		IF3110 @ 7609	IF3170 @ 7602	
				IF3140 @ 7602	
13					
14	WI2022 @ 7609				
15	IF3110 @ 7602				
	IF3130 @ 7609				
16			IF2224 @ 7602		
17					IF3110 @ 9018

State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	WI2022 @ 7609	IF3170 @ 9018			
8	KU5004 @ 9018		IF2224 @ 7609		
9	IF3130 @ 7609	IF3110 @ 7602			
10	IF3170 @ 7609				
11	IF2224 @ 7602				IF3140 @ 7609
12	IF3130 @ 7602		IF3170 @ 9018		
13	IF3110 @ 7602				
14	IF3140 @ 7602				IF3140 @ 7602
15	IF3130 @ 9018				
16	IF2224 @ 7602		IF2224 @ 7602	WI2022 @ 9018	
17	KU5004 @ 9018	IF3110 @ 9018	IF3170 @ 7609		

Durasi = 2.220384299987927 s

Banyak Restart = 1

Banyak iterasi untuk restart index ke-0 = 7

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	IF3170 @ 7609 IF3140 @ 9018		IF3170 @ 7602		
8	WI2022 @ 9018		IF3170 @ 9018		IF2224 @ 7602
9	KU5004 @ 7609	IF2224 @ 7609	IF3130 @ 9018	KU5004 @ 9018	
10				IF2224 @ 9018 IF3140 @ 7609	
11					
12	IF3130 @ 7602		IF3110 @ 7609 IF3170 @ 7602 IF3140 @ 7602		
13					
14	WI2022 @ 7609				
15	IF3110 @ 7602 IF3130 @ 7609				
16			IF2224 @ 7602		
17					IF3110 @ 9018

File: input2.json

Maximum Restart = 4

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	IF3170 @ 9018	IF3110 @ 9018			
8	IF3170 @ 7609 KU3001 @ 7602				
9					WI2022 @ 7609
10					
11	KU3002 @ 7602	IF3140 @ 7609	IF2224 @ 9018 KU3002 @ 9018	IF2224 @ 7609	KU5004 @ 9018
12					
13	IF3140 @ 9018				
14					
15		IF3110 @ 7602 IF3130 @ 9018 IF2224 @ 7602		IF2224 @ 7602 KU3001 @ 7609	
16	IF3130 @ 7609		IF3170 @ 7602	KU5004 @ 7609 IF3110 @ 7609 IF3130 @ 7602 IF3140 @ 7602	
17	WI2022 @ 9018			IF3170 @ 7602	

State Akhir

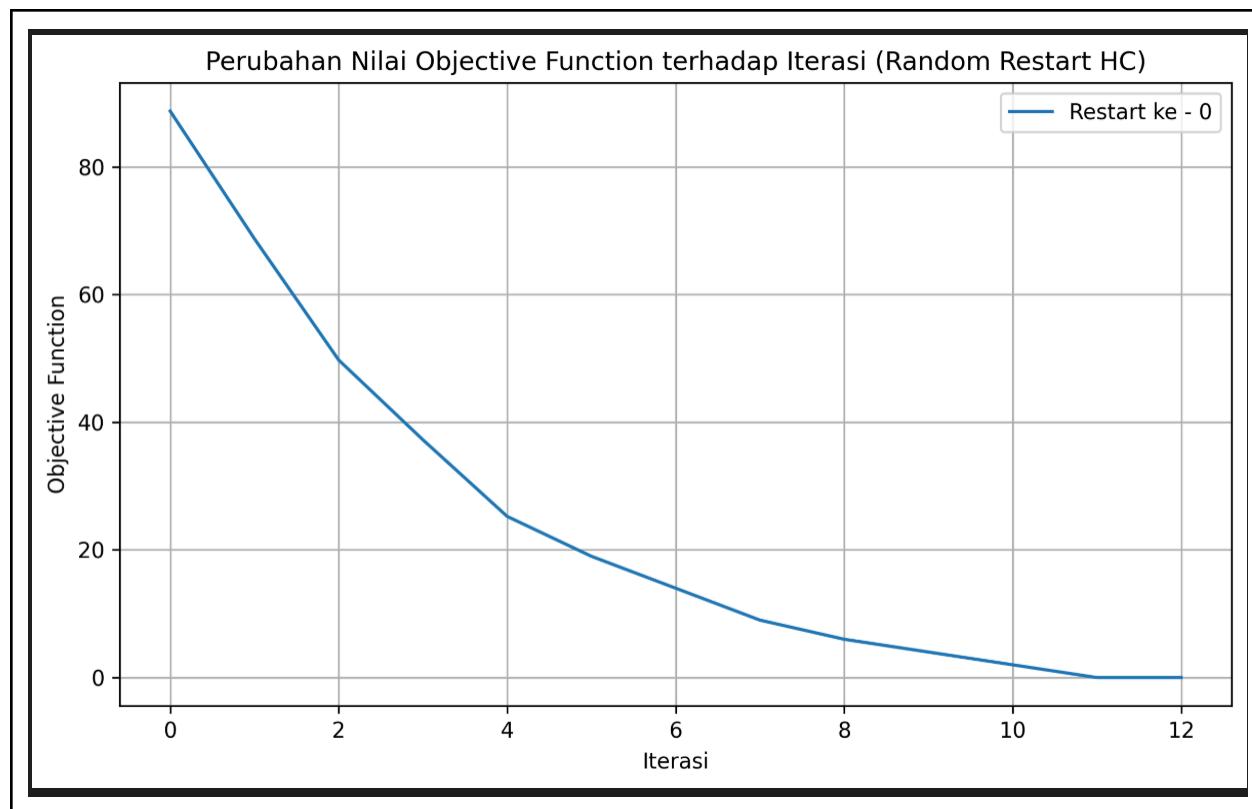
Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	IF3170 @ 9018	IF3110 @ 9018			
8	IF3170 @ 7609				
	KU3001 @ 7602				
9					WI2022 @ 7609
10					
11	KU3002 @ 7602	IF3140 @ 7609	IF2224 @ 9018	IF2224 @ 7609	KU5004 @ 9018
			KU3002 @ 9018		
12					
13	IF3140 @ 9018				
14					
15		IF3110 @ 7602		IF2224 @ 7602	
		IF3130 @ 9018		KU3001 @ 7609	
		IF2224 @ 7602			
16	IF3130 @ 7609		IF3170 @ 7602	KU5004 @ 7609	IF3110 @ 7609
					IF3130 @ 7602
					IF3140 @ 7602
17	WI2022 @ 9018			IF3170 @ 7602	

Durasi = 6.397725299932063 s

Banyak Restart = 1

Banyak iterasi untuk restart index ke-0 = 13



Stochastic Hill Climbing

File: input.json

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7					
8				IF3071_K01 @ multimedia IF3140_K01 @ 7609	
9					
10		IF3110_K02 @ 7606			
11					
12		IF3071_K01 @ 7609			IF3071_K01 @ 7606
13					
14					IF3110_K02 @ multimedia
15				IF3130_K01 @ 7609	IF3110_K02 @ 7609
16					
17			IF3130_K01 @ 7606	IF3140_K01 @ multimedia	

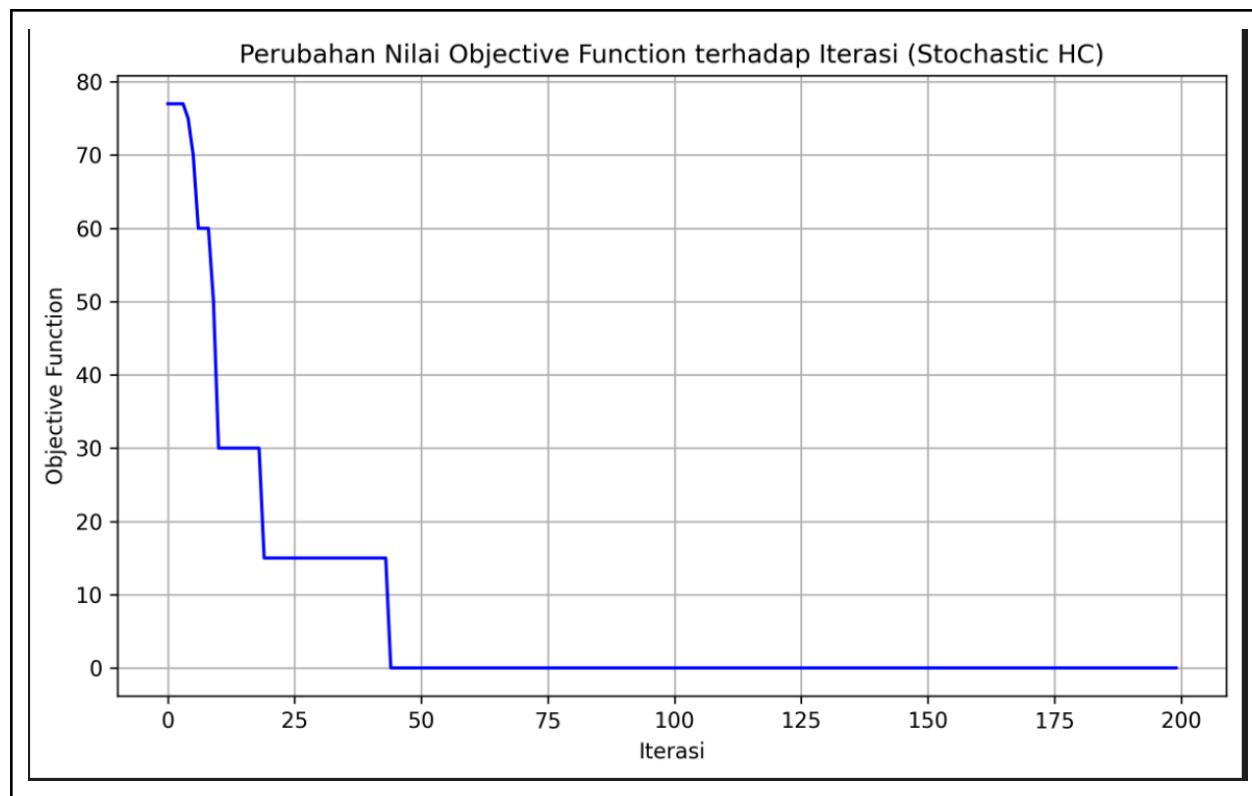
State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7				IF3140_K01 @ 7606	
8					
9					
10	IF3110_K02 @ 7606	IF3110_K02 @ 7606			
11					
12		IF3071_K01 @ 7609		IF3071_K01 @ 7609	IF3071_K01 @ 7606
13					
14					IF3110_K02 @ 7606
15				IF3130_K01 @ 7609	
16					
17	IF3140_K01 @ 7606		IF3130_K01 @ 7606		

Durasi = 0.07558679999783635 s

Banyak iterasi hingga mencapai proses berhenti = 200



File: input1.json

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7		WI2022 @ 9018	IF3170 @ 7602		
8	IF3130 @ 9018	KU5004 @ 9018	IF3110 @ 9018		
9	IF3170 @ 9018 IF3130 @ 7609			IF3140 @ 7602	IF3110 @ 7609
10	IF3170 @ 7602 KU5004 @ 7609		IF2224 @ 7602		
11	IF2224 @ 7609			IF3110 @ 7602	
12					IF3140 @ 7609
13					
14	IF3130 @ 7602	IF2224 @ 9018			
15				IF2224 @ 7602	
16	IF3140 @ 9018				
17	IF3170 @ 7609				WI2022 @ 7609

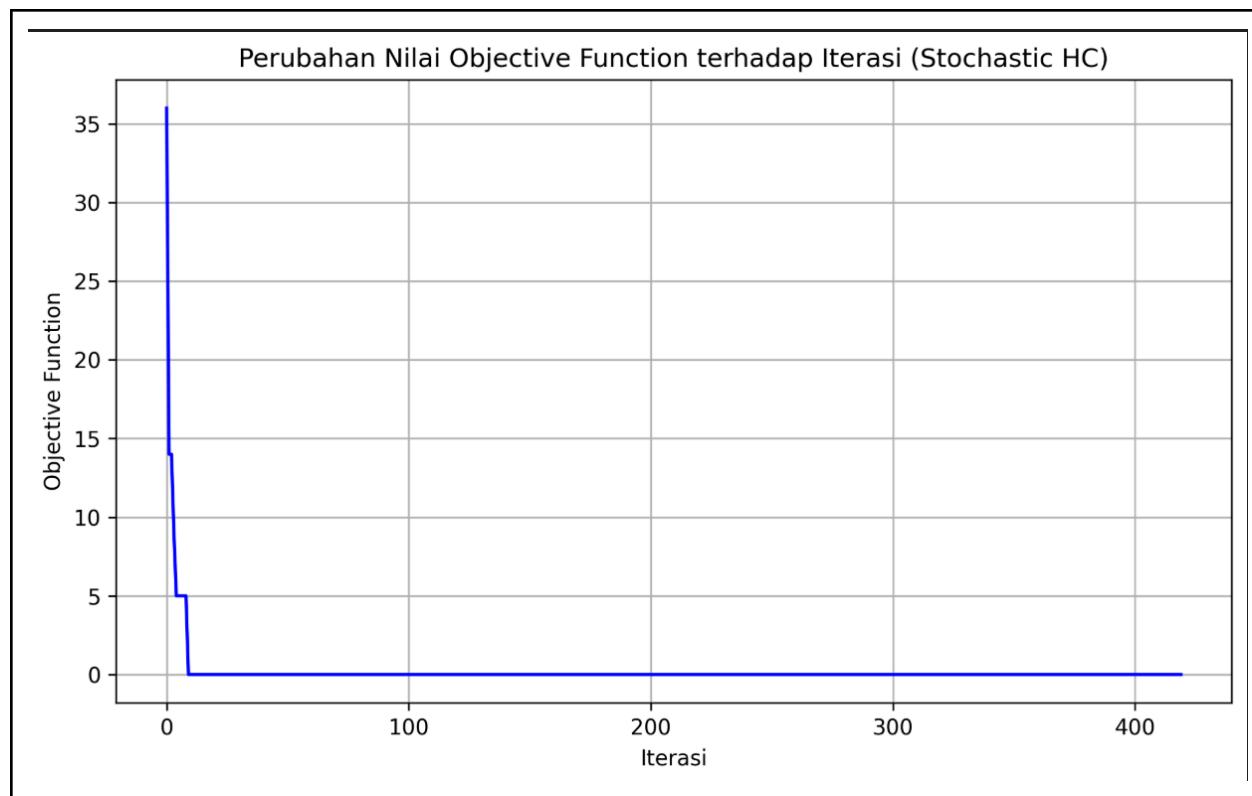
State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7		WI2022 @ 9018			
8	IF3130 @ 9018	KU5004 @ 9018	IF3110 @ 9018		
9	IF3130 @ 7609			IF3140 @ 7602	IF3110 @ 7609
10	IF3170 @ 7609		IF2224 @ 7602		
11	IF2224 @ 7609	IF3170 @ 9018		IF3110 @ 7602	
12					IF3140 @ 7602
13		KU5004 @ 9018			
14	IF3130 @ 7602	IF2224 @ 9018			
15		IF3170 @ 9018		IF2224 @ 7602	
16	IF3140 @ 9018				
17	IF3170 @ 7609				WI2022 @ 7609

Durasi = 0.30147790000773966 s

Banyak iterasi hingga mencapai proses berhenti = 420



File: input2.json

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7					IF2224 @ 7602
8		IF3130 @ 7602 WI2022 @ 9018			
9		KU3001 @ 7609			IF2224 @ 9018 IF3140 @ 7609 KU3002 @ 7602
10			.		
11	KU5004 @ 9018 IF2224 @ 7602		KU3001 @ 7602		
12	IF3110 @ 9018 IF3170 @ 9018	IF2224 @ 7609	WI2022 @ 7609		IF3110 @ 7602
13		KU5004 @ 7609		IF3140 @ 7602	
14	IF3110 @ 7609		IF3170 @ 7602 IF3130 @ 7609		
15	IF3140 @ 9018				IF3170 @ 7609
16				IF3170 @ 7602	
17	IF3130 @ 9018				KU3002 @ 9018

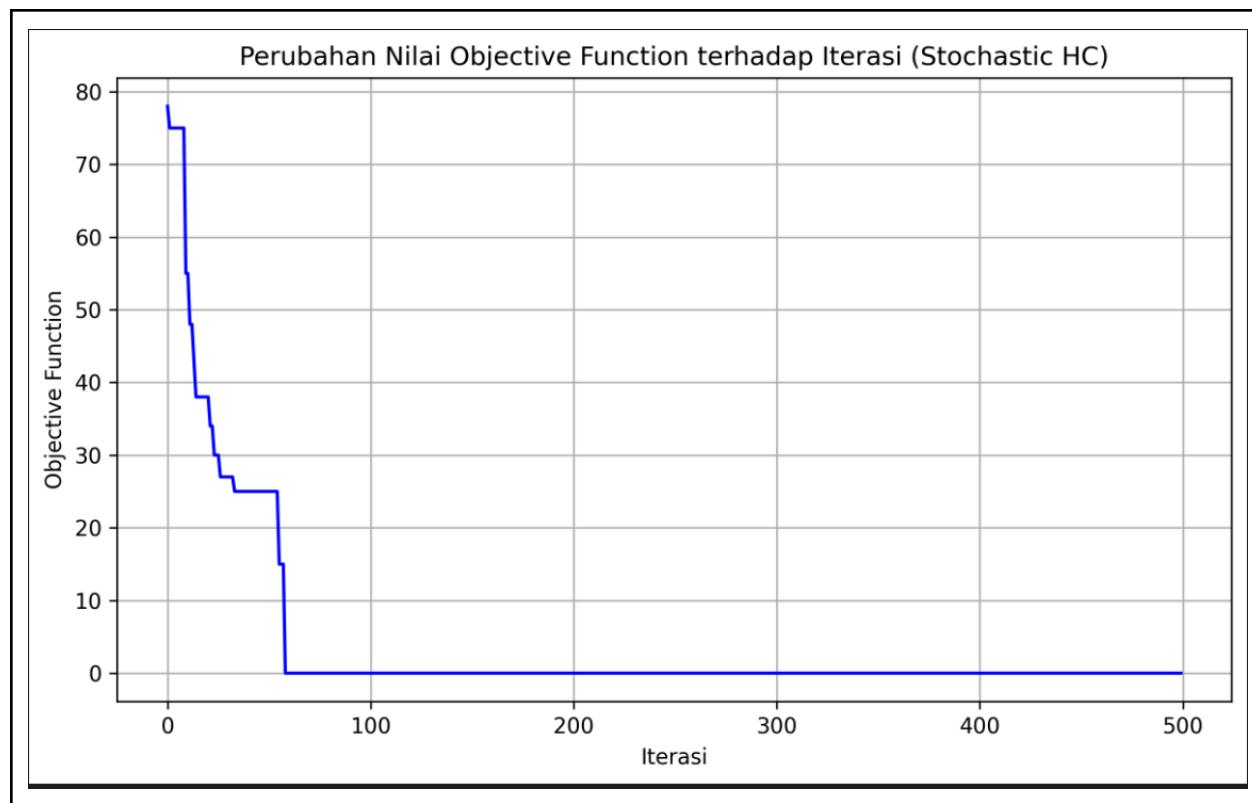
State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7					IF2224 @ 7602
8	KU5004 @ 9018	IF3130 @ 7602			
9		KU3001 @ 9018		IF2224 @ 7609	IF3140 @ 7602
10			KU3002 @ 9018		
11	IF2224 @ 7602 KU5004 @ 9018			WI2022 @ 7609	
12	IF3170 @ 7609	IF2224 @ 7609	WI2022 @ 7602		IF3110 @ 7602
13	IF3170 @ 7609			IF3140 @ 7602	
14	IF3110 @ 7609		IF3130 @ 7609		
15	IF3140 @ 9018				IF3170 @ 7609
16		KU3001 @ 9018		IF3170 @ 7609	
17	IF3130 @ 9018		IF3110 @ 7609		KU3002 @ 9018

Durasi = 0.638396100141108 s

Banyak iterasi hingga mencapai proses berhenti = 500



Simulated Annealing

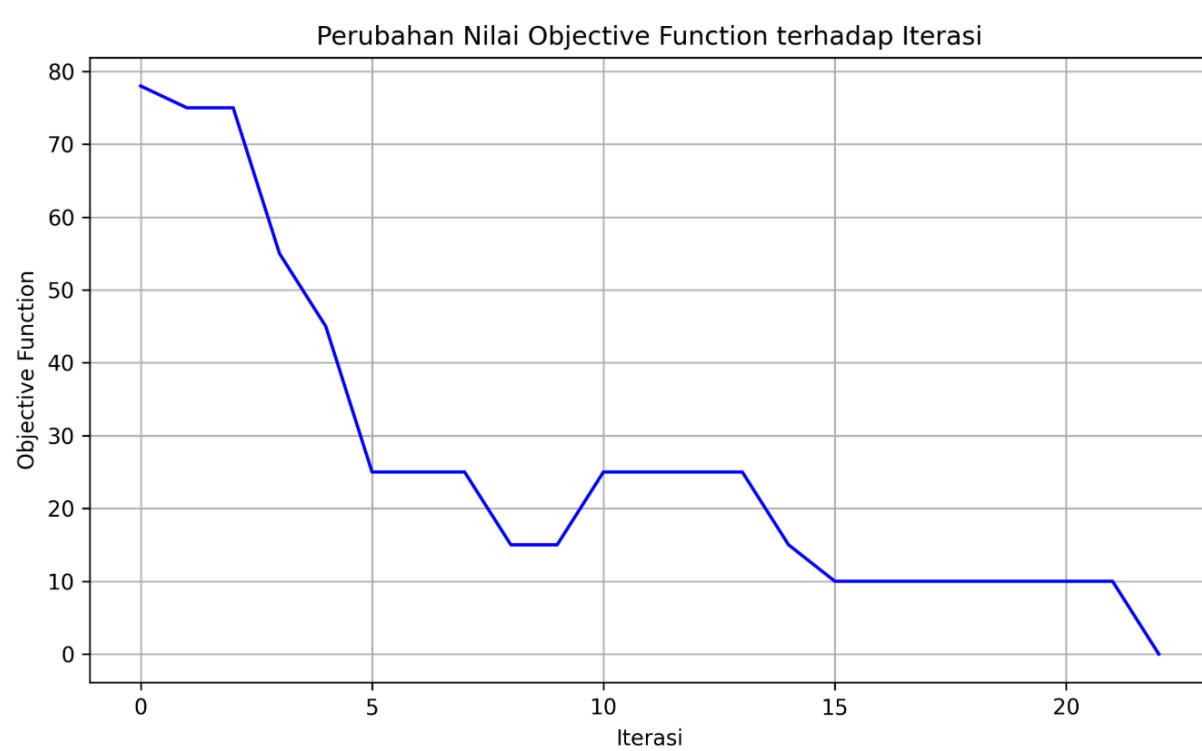
File: input.json

State Awal

Jam	Semin	selasa	Rabu	Kamis	Jumat
7					
8		IF3110_K02 @ 7609	IF3071_K01 @ 7609		
			IF3140_K01 @ 7609		
9		IF3130_K01 @ 7609	IF3071_K01 @ multimedia		
10	IF3071_K01 @ 7606				
11					
12			IF3110_K02 @ 7606		
13		IF3130_K01 @ 7606			
14					
15	IF3110_K02 @ multimedia				
16			IF3140_K01 @ multimedia		
17					

State Akhir		Objective Function Akhir = 0				
Jam	Senin	Selasa	Rabu	Kamis	Jumat	
7		IF3071_K01 @ 7606				IF3130_K01 @ 7606
8			IF3071_K01 @ 7606			
9						
10						
11						
12		IF3110_K02 @ 7606	IF3110_K02 @ 7606			
13		IF3130_K01 @ 7609				
14						
15						
16	IF3071_K01 @ 7609			IF3110_K02 @ 7606		
17		IF3140_K01 @ 7609			IF3140_K01 @ 7609	

Durasi = 0.011528499999258202 s





Frekuensi Terjebak Local Optimum = 1

File: input1.json

State Awal

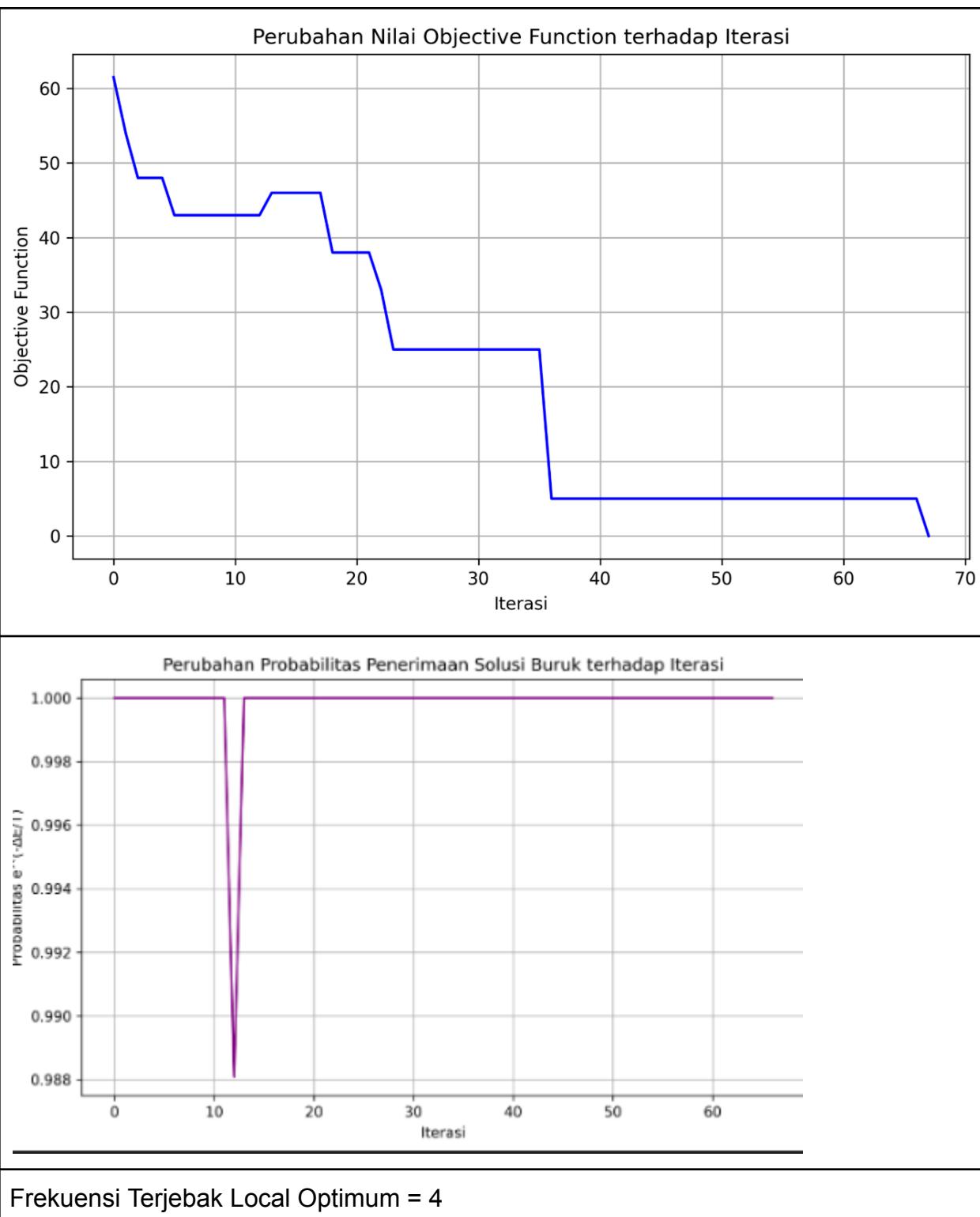
+-----+-----+-----+-----+-----+					
Jam	Senin	Selasa	Rabu	Kamis	Jumat
7					
8	IF3170 @ 7602 KU5004 @ 7609				
9		IF3170 @ 9018		IF2224 @ 7602	
10	IF3140 @ 7609 WI2022 @ 9018				KU5004 @ 9018
11	IF3130 @ 9018		IF3170 @ 7602		
12			IF3130 @ 7602		
13					IF3170 @ 7609 IF2224 @ 9018
14			IF2224 @ 7609		IF3110 @ 7602
15	WI2022 @ 7609			IF3110 @ 7609	
16			IF3140 @ 7602		IF3130 @ 7609
17		IF2224 @ 7602	IF3110 @ 9018 IF3140 @ 9018		

State Akhir

Objective Function Akhir = 0

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7					
8	IF2224 @ 7602		IF3170 @ 7609		
9			IF2224 @ 7609	IF3130 @ 7602	
10	WI2022 @ 7602				KU5004 @ 9018
11		IF3170 @ 7609	IF3170 @ 7609	IF2224 @ 9018	
12			IF3140 @ 7609	WI2022 @ 7602	
13		IF3140 @ 7602	IF3130 @ 7602	IF3130 @ 9018	
14		IF3110 @ 7602	IF3140 @ 7602		IF3110 @ 7609
15			KU5004 @ 9018		
16				IF2224 @ 9018	
17			IF3170 @ 7609		IF3110 @ 7609

Durasi = 0.07853070000055595



File: input2.json

State Awal

Jam	Senin	Selasa	Rabu	Kamis	Jumat
7	KU3002 @ 9018				
8					KU5004 @ 7609
9			IF3110 @ 7602		IF3110 @ 7609
10	IF2224 @ 9018 WI2022 @ 7609	KU3001 @ 7602	IF3140 @ 9018		
11		IF3170 @ 7602		IF3130 @ 7609	
12	IF3170 @ 7602				IF3140 @ 7602
13				TF2224 @ 7609	
14	WI2022 @ 9018		KU3001 @ 7609	IF3110 @ 9018	
15		IF2224 @ 7602		KU5004 @ 9018 IF3130 @ 7602 IF2224 @ 7602 KU3002 @ 7602	
16		IF3130 @ 9018	IF3170 @ 9018		IF3170 @ 7609
17		IF3140 @ 7609			

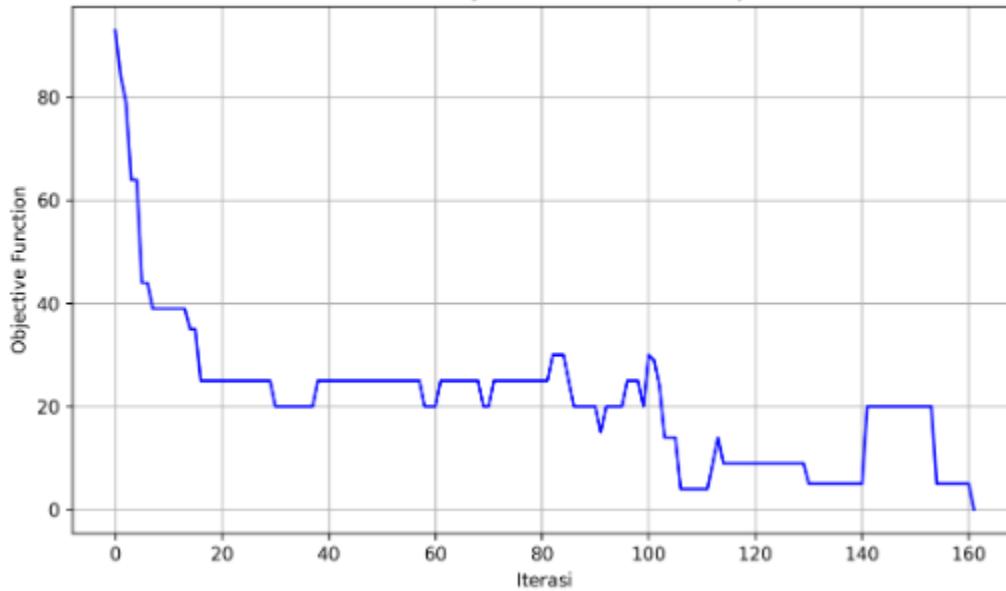
State Akhir

Objective Function Akhir = 22

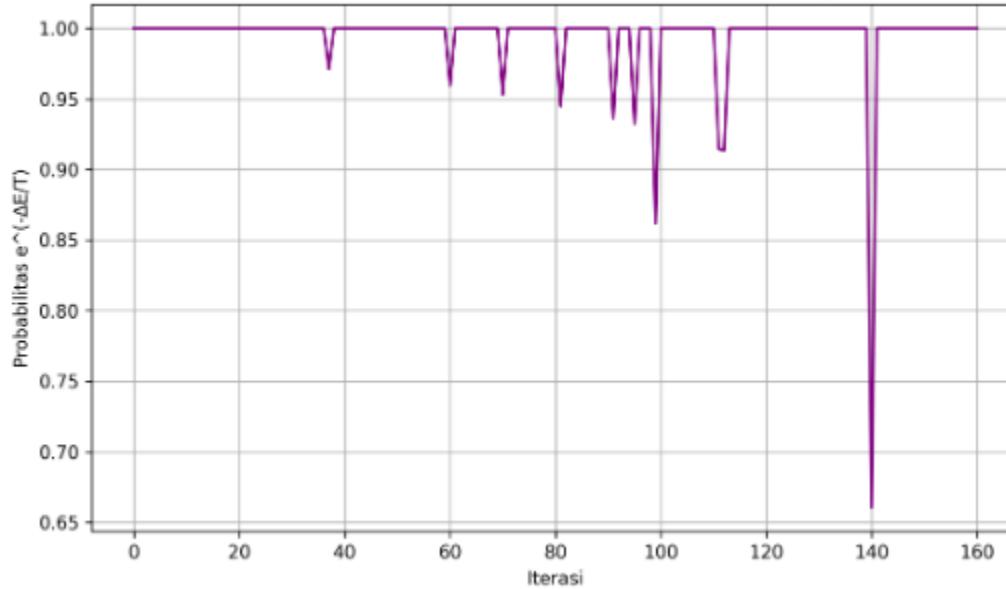
Jam	Senin	Selasa	Rabu	Kamis	Jumat
7				IF3130 @ 9018	
8	IF3110 @ 9018				IF2224 @ 7602
9		KU5004 @ 9018	IF2224 @ 7609	WI2022 @ 7609	IF3170 @ 9018
10		KU3001 @ 9018		IF3110 @ 9018	IF2224 @ 7602
11	IF3170 @ 9018	IF3130 @ 7609			IF2224 @ 7602
12					KU3001 @ 9018
13		IF3170 @ 7609		IF3140 @ 7602	
14	IF3130 @ 7602		KU3002 @ 9018		
15		IF3110 @ 7609		KU5004 @ 9018	IF3140 @ 9018
16			IF3140 @ 9018	IF3170 @ 7609	
17	WI2022 @ 9018				KU3002 @ 9018

Durasi = 0.19753180000407156 s

Perubahan Nilai Objective Function terhadap Iterasi



Perubahan Probabilitas Penerimaan Solusi Buruk terhadap Iterasi



Frekuensi Terjebak Local Optimum = 12

Genetic Algorithm

Catatan:

Plot (.jpg), Visualisasi Fittest Individual, Visualisasi Individual Pada Populasi Awal, Visualisasi Individual Pada Populasi Akhir, Input disediakan [disini](#). Nama file sesuai pada tabel.

File: input.json						
Run	Fitness Function Akhir (MAX)	Jumlah Populasi (k)	Batas Maksimum Iterasi (n)	Banyak Iterasi Dilakukan	Durasi (s)	Nama File
Populasi sebagai variabel kontrol						
1.	1	50	5	3	0.26	ga_input_k50_n5_run1
2.	1	50	5	4	0.36	ga_input_k50_n5_run2
3.	1	50	5	3	0.28	ga_input_k50_n5_run3
1.	1	50	75	7	0.65	ga_input_k50_n75_run1
2.	1	50	75	5	0.47	ga_input_k50_n75_run2
3.	1	50	75	5	0.45	ga_input_k50_n75_run3
1.	1	50	500	5	0.47	ga_input_k50_n500_run1
2.	1	50	500	4	0.36	ga_input_k50_n500_run2
3.	1	50	500	3	0.26	ga_input_k50_n500_run3
Iterasi sebagai variabel kontrol						
1.	1	4	200	30	0.11	ga_input_k4_n200_run1
2.	1	4	200	12	0.04	ga_input_k4_n200_run2
3.	1	4	200	11	0.04	ga_input_k4_n200_run3
1.	1	40	200	4	0.26	ga_input_k40_n200_run1
2.	1	40	200	4	0.26	ga_input_k40_n200_run2
3.	1	40	200	4	0.28	ga_input_k40_n200_run3
1.	1	80	200	5	0.99	ga_input_k80_n200_run1
2.	1	80	200	4	0.79	ga_input_k80_n200_run2
3.	1	80	200	3	0.58	ga_input_k80_n200_run3

File: input1.json						
Run	Fitness Function	Jumlah Populasi	Batas Maksimum	Banyak Iterasi	Durasi (s)	Nama File

	Akhir (MAX)	(k)	Iterasi (n)	Dilakukan		
Populasi sebagai variabel kontrol						
1.	0.33	50	5	5	0.76	ga_input1_k50_n5_run1
2.	0.33	50	5	5	0.76	ga_input1_k50_n5_run2
3.	0.16	50	5	5	0.76	ga_input1_k50_n5_run3
1.	1	50	75	7	1.05	ga_input1_k50_n75_run1
2.	1	50	75	6	0.94	ga_input1_k50_n75_run2
3.	1	50	75	7	1.04	ga_input1_k50_n75_run3
1.	1	50	500	6	0.91	ga_input1_k50_n500_run1
2.	1	50	500	11	1.62	ga_input1_k50_n500_run2
3.	1	50	500	6	0.91	ga_input1_k50_n500_run3
Iterasi sebagai variabel kontrol						
1.	1	4	200	27	0.18	ga_input1_k4_n200_run1
2.	1	4	200	9	0.06	ga_input1_k4_n200_run2
3.	1	4	200	17	0.11	ga_input1_k4_n200_run3
1.	1	40	200	8	0.88	ga_input1_k40_n200_run1
2.	1	40	200	5	0.55	ga_input1_k40_n200_run2
3.	1	40	200	9	0.97	ga_input1_k40_n200_run3
1.	1	80	200	4	1.23	ga_input1_k80_n200_run1
2.	1	80	200	7	2.15	ga_input1_k80_n200_run2
3.	1	80	200	4	1.29	ga_input1_k80_n200_run3

File: input2.json						
Run	Fitness Function Akhir (MAX)	Jumlah Populasi (k)	Batas Maksimum Iterasi (n)	Banyak Iterasi Dilakukan	Durasi (s)	Nama File
1	1000	100	100	100	100	input2.json

1.	0.03	50	5	5	0.88	ga_input2_k50_n5_run1
2.	0.04	50	5	5	0.87	ga_input2_k50_n5_run2
3.	0.03	50	5	5	0.88	ga_input2_k50_n5_run3
1.	1	50	75	15	2.59	ga_input2_k50_n75_run1
2.	1	50	75	20	3.50	ga_input2_k50_n75_run2
3.	1	50	75	13	2.27	ga_input2_k50_n75_run3
1.	1	50	500	12	2.08	ga_input2_k50_n500_run1
2.	1	50	500	15	2.64	ga_input2_k50_n500_run2
3.	1	50	500	10	1.78	ga_input2_k50_n500_run3

Iterasi sebagai variabel kontrol

1.	1	4	200	55	0.46	ga_input2_k4_n200_run1
2.	1	4	200	54	0.44	ga_input2_k4_n200_run2
3.	1	4	200	89	0.72	ga_input2_k4_n200_run3
1.	1	40	200	11	1.44	ga_input2_k40_n200_run1
2.	1	40	200	17	2.10	ga_input2_k40_n200_run2
3.	1	40	200	13	1.64	ga_input2_k40_n200_run3
1.	1	80	200	10	3.48	ga_input2_k80_n200_run1
2.	1	80	200	14	4.88	ga_input2_k80_n200_run2
3.	1	80	200	15	5.51	ga_input2_k80_n200_run3

Analisis

- Seberapa dekat tiap-tiap algoritma bisa mendekati global optima dan mengapa hasilnya demikian?
 - HC-Steepest Ascent: Pencarian semua successor untuk menentukan *neighbour* terbaik dari suatu *current state* pada pemrosesan algoritma ini dapat mengarahkan menuju solusi global optimum, tetapi di sisi lain algoritma *HC-Steepest Ascent* sangat mungkin terjebak dalam lokal optimum dalam kasus yang lebih kompleks tanpa jalan keluar.
 - HC-Sideways Move: Variansi dari algoritma *HC-Steepest Ascent* ini dapat mengarahkan menuju hasil global optima yang lebih baik dibandingkan

algoritma *HC-Steepest Ascent* biasa karena solusi state saat ini dalam tahapan pemrosesan algoritma dapat berpindah ke neighbour yang memiliki nilai objektif yang sama, sehingga dapat meningkatkan probabilitas menemukan suatu state successor yang menjadi global optimum-nya.

- *HC-Random Restart*: Algoritma ini sangat mampu untuk mencapai solusi global optimum karena dilakukan *restart* terus menerus untuk proses algoritma *HC-Steepest Ascent* yang dilakukan hingga mencapai *goal*, yakni solusi global optimum, apabila tidak dibatasi jumlah maksimum *restart*-nya. Berdasarkan hasil eksperimen yang dilakukan, algoritma *HC-Random Restart* berhasil mencapai solusi global optimum untuk semua percobaan karena pada pengujian tersebut solusi global optimum ditemukan pada suatu restart dari proses *Hill Climbing* sebelum mencapai batas restart yang ditentukannya.
 - *HC-Stochastic*: Pendekatan yang dilakukan oleh algoritma *HC-Stochastic* mampu mengarahkan pada solusi global optimum dengan probabilitas hasil tersebut akan semakin tinggi seiring dengan peningkatan batas maksimum jumlah iterasi yang dilakukan.
 - *Simulated Annealing*: Untuk algoritma Simulated Annealing, sebenarnya ia cukup mampu untuk mendekati atau mencapai solusi global. Hal ini bisa terjadi karena pada suhu yang masih tinggi algoritma ini berani untuk keluar dari local optimumnya dan menjelajahi ruang solusi lain. Namun jika kita membuat suhu dan batas iterasi menjadi besar ini akan menjadi kurang efektif, memang dengan membesarnya kedua variabel tersebut memungkinkan algoritma mendapatkan global optima akan tetapi pasti ada trade off waktunya
 - *Genetic Algorithm*: Semakin besar ukuran populasi dan batas maksimal iterasi maka semakin besar kemungkinannya untuk mencapai global optima. Hal ini terjadi karena pada setiap iterasi, individu dengan nilai *fitness function* lebih tinggi akan memiliki persentase lebih tinggi untuk terpilih menjadi *parent*. Akibatnya *child* akan mewariskan *gen/fitness function* tinggi milik kedua *parentnya*. Apabila ini dilakukan terus-menerus maka pada akhirnya akan ada *child* yang merupakan global optima.
- Bagaimana perbandingan hasil pencarian tiap-tiap algoritma dengan algoritma local search yang lain?
 - *HC-Steepest Ascent*: Proses pencarian neighbour dengan pemilihan *successor* terbaik dari semua *successor* yang di-generate membuat algoritma ini cukup menghasilkan solusi yang berkualitas, walaupun rentan terjebak di jebakan optimum lokal karena tidak memiliki mekanisme untuk keluar dari kondisi tersebut, tidak seperti pada Simulated Annealing. Namun, proses pencarian neighbour yang dilakukan pada algoritma ini memakan waktu yang lebih banyak dibandingkan algoritma yang hanya memerlukan satu *successor*, seperti

algoritma *HC-Stochastic*, karena perlu dilakukan eksplorasi terhadap semua *successor* dari state saat ini.

- *HC-Sideways Move*: Hasil pencarian yang dilakukan oleh *HC-Sideways Move* memberikan hasil yang mungkin sedikit lebih baik dibandingkan *HC-Steeppest Ascent* karena algoritma ini lebih mungkin menghindari jebakan optimum lokal dengan perpindahan solusi state sekarang ke neighbour jika nilai fungsi objektifnya sama (*sideways move*), tetapi di sisi lain algoritma ini dapat terjebak dalam area “plateau” yang akan menambah lama waktu pencarian.
 - *HC-Random Restart*: Algoritma ini merupakan algoritma yang dapat memberikan hasil yang sangat baik dalam mencari solusi global optimum dibandingkan algoritma Hill Climbing lainnya karena proses pencarian yang dilakukan berulang kali dengan restart yang dimulai dari titik *initial state* yang berbeda memungkinkan untuk menemukan solusi *global optimum* yang lebih mudah dan konsisten, tetapi memberikan trade-off terhadap waktu pencarian.
 - *HC-Stochastic*: Pemilihan neighbour yang acak menyebabkan kualitas hasil solusi yang bervariasi oleh algoritma *HC-Stochastic* jika dibandingkan dengan algoritma lain. Selain itu, semakin besar iterasi yang diperbolehkan dalam *HC-Stochastic*, maka akan semakin mungkin ditemukan *global optimum*, tetapi juga menambah waktu komputasi pencarian.
 - *Simulated Annealing*: Dibandingkan dengan algoritma Hill Climb seperti *HC-Steeppest Ascent* dan *HC-Stochastic* algoritma ini jauh lebih unggul dari segi kualitas hasil karena algoritma ini dapat keluar dari jebakan optimum lokal. Dibandingkan dengan *HC-Random Restart*, peluang *Simulated Annealing* untuk mencapai *global optimum* memang sedikit lebih rendah karena *Random Restart* melakukan pencarian dari banyak titik awal yang berbeda. Namun, SA memiliki keunggulan dari sisi efisiensi waktu karena tidak perlu mengulang proses pencarian dari awal berulang kali. Dengan parameter suhu dan *cooling rate* yang tepat, SA dapat mencapai hasil yang mendekati *global optimum* dalam waktu yang lebih singkat. Jika dibandingkan dengan GA dengan populasi besar, algoritma ini mungkin kurang konsisten juga untuk menghasilkan solusi global optimum
 - *Genetic Algorithm*: Nilai *fitness function* memiliki kecenderungan menaik seiring dengan bertambahnya jumlah iterasi yang dilakukan. Berdasarkan analisis dari hasil test, *genetic algorithm* dengan ukuran populasi yang besar hampir selalu bisa mencapai global optimum dengan jumlah iterasi yang relatif sedikit (<15) jika dibandingkan dengan algoritma lainnya.
- Bagaimana perbandingan durasi proses pencarian tiap algoritma relatif terhadap algoritma lainnya?
 - *HC-Steeppest Ascent*: Algoritma *HC-Steeppest Ascent* men-generate semua *successors* untuk memilih *neighbour* sehingga durasi dari proses algoritma ini

lebih lama terutama dalam tiap iterasi dari algoritma yang hanya mengambil satu successor, seperti algoritma HC-Stochastic dan Simulated Annealing, walaupun jumlah iterasi yang dilakukan dapat lebih sedikit dibandingkan algoritma HC-Stochastic

- HC-Sideways Move: Durasi pencarian dari algoritma ini pada umumnya lebih lambat dari algoritma HC-Steepest Ascent dan algoritma hill climbing lainnya, kecuali algoritma HC-Random Restart untuk banyak kasus. Hal ini karena sebagai varian dari HC-Steepest Ascent, algoritma HC-Sideways Move memiliki tambahan keunikan, yakni solusi state saat ini (current) dapat berpindah ke *neighbour state* yang memiliki nilai objektif yang sama sehingga proses pencarian sehingga proses algoritma ini dapat lebih lama terjebak di area “plateau” sebelum menemukan solusi state baru atau selesai.
- HC-Random Restart: Algoritma Random Restart-HC memiliki durasi proses pencarian yang relatif lebih lama dibandingkan algoritma-algoritma lainnya, utamanya jika dibandingkan dengan algoritma hill climbing lainnya, karena algoritma ini pada dasarnya melakukan proses pencarian seperti pada algoritma HC-Steepest Ascent, tetapi berulang kali sebanyak jumlah restart untuk mencapai solusi global optimum. Jumlah restart pada algoritma HC-Random Restart dapat memengaruhi durasi pencarian secara signifikan, terutama apabila solusi global optimum ditemukan pada restart yang mendekati restart terakhir.
- HC-Stochastic: durasi proses pencarian algoritma ini lebih cepat dibandingkan algoritma Hill Climbing lainnya dan Genetic Algorithm, bahkan mungkin juga lebih cepat dibandingkan algoritma Simulated Annealing. Hal ini karena algoritma ini hanya membangkitkan satu successor saja secara random untuk dipilih menjadi neighbour dan tidak melakukan restart atau memelihara populasi solusi sehingga waktu komputasinya berjalan cepat. Hal yang membuat algoritma Stochastic-HC sedikit lebih cepat daripada Simulated Annealing meskipun durasi keduanya sangat dipengaruhi oleh jumlah maksimum iterasi adalah proses pada tiap iterasi dalam algoritma ini yang lebih simpel, yakni solusi state tiap iterasi hanya berdasarkan perbandingan nilai fungsi objektif state *current* dengan *neighbour* nya, tanpa perlu adanya perhitungan tambahan seperti *probability move* yang ada pada Simulated Annealing.
- Simulated Annealing: Simulated annealing mungkin lebih lambat dari HC-Stochastic tetapi ia lebih cepat dibandingkan Genetic Algorithm, HC-Steepest Ascent, dan HC-Random Restart. . Algoritma ini lebih cepat dibandingkan dengan HC-steepest ascent dan HC-random restart karena kedua algoritma tersebut perlu membangkitkan seluruh kemungkinan yang ada terlebih dahulu sebelum memilih neighbournya. Sementara itu, SA hanya membangkitkan satu tetangga acak pada setiap iterasi, sehingga waktu komputasinya per langkah lebih rendah. Selain itu algoritma ini juga tidak perlu

melakukan restart atau memelihara populasi solusi seperti pada *Random Restart* maupun *Genetic Algorithm*. Dengan demikian, kompleksitas komputasinya jauh lebih ringan.

- *Genetic Algorithm*: Durasi proses pencarian cenderung berbanding lurus dengan bertambahnya ukuran populasi dan batas maksimum iterasi. Dibandingkan dengan algoritma lainnya, *genetic algorithm* memiliki durasi pencarian yang lebih lama. Hal itu disebabkan karena kompleksitas *genetic algorithm* lebih tinggi ketimbang algoritma lainnya.
- Seberapa konsisten hasil akhir yang didapatkan dari tiap-tiap eksperimen yang dilakukan?
 - *HC-Steepest Ascent*: Hasil dari beberapa eksperimen yang dilakukan untuk algoritma *HC-Steepest Ascent* telah menunjukkan bahwa dalam kasus tertentu algoritma *HC-Steepest Ascent* dapat memberikan hasil yang konsisten mendekati global optimum. Namun, kekonsistennan ini juga dipengaruhi oleh cara perhitungan nilai objektif dan kompleksitas dari kasus sehingga sangat mungkin bahwa hasil pengujian yang dilakukan terhadap algoritma *HC-Steepest Ascent* belum tentu konsisten untuk kasus lainnya.
 - *HC-Sideways Move*: Algoritma *HC-Sideways Move* berhasil memberikan hasil akhir yang konsisten untuk semua eksperimen yang dilakukan. Kekonsistennan dari algoritma ini dapat dijelaskan karena eksperimen yang dilakukan sebelumnya pada algoritma *HC-Steepest Ascent* juga memberikan hasil konsisten untuk kasus ini, sehingga *HC-Sideways Move* sebagai varian yang lebih optimum juga akan memberikan hasil yang konsisten juga.
 - *HC-Random Restart*: Hasil akhir yang didapatkan dari eksperimen untuk algoritma *HC-Random Restart* menunjukkan konsistensi, bahkan solusi global optimum sudah didapatkan pada restart pertama (restart indeks ke-0). Hasil yang konsisten pada algoritma *HC-Steepest Ascent* juga berkontribusi pada konsistensi *HC-Random Restart* karena dasar dari proses tiap restart pada algoritma ini adalah logika algoritma *HC-Steepest Ascent*,
 - *HC-Stochastic*: Algoritma ini memberikan hasil yang global optimum secara konsisten dari eksperimen-eksperimen yang dilakukan. Konsistensi ini sangat ditentukan dari parameter batas maksimum jumlah iterasi yang dilakukan pada algoritma *HC-Stochastic* sehingga penentuan batas maksimum jumlah iterasi yang baik akan memberikan hasil yang lebih konsisten.
 - *Simulated Annealing*: Untuk algoritma ini, didapat hasilnya cukup konsisten untuk mendekati solusi optimum. Konsistensi tersebut dicapai ketika parameter algoritma diatur dengan tepat, khususnya dengan batas iterasi pencarian yang cukup besar, suhu awal (*initial temperature*) sebesar 100, dan laju pendinginan (*cooling rate*) yang relatif lambat. Pengaturan tersebut memungkinkan algoritma untuk melakukan eksplorasi ruang solusi secara lebih luas pada awal proses,

- serta perlahan berfokus pada eksplorasi menuju solusi optimal seiring menurunnya suhu.
- *Genetic Algorithm*: Dengan ukuran populasi yang cenderung besar (≥ 20), *genetic algorithm* secara konsisten dapat menemukan *individu/state* yang merupakan global maksimum
 - Bagaimana pengaruh banyak iterasi dan jumlah populasi terhadap hasil akhir pencarian pada Genetic Algorithm?
 - Semakin besar ukuran populasi dan batas maksimum iterasi maka durasi pencarian akan semakin lama. Namun dengan kondisi tersebut, hampir dapat dipastikan *genetic algorithm* selalu menemukan *individu/state* yang merupakan global maksimum

KESIMPULAN DAN SARAN

Berdasarkan hasil eksperimen, seluruh algoritma local search yang telah dibuat berhasil menyelesaikan masalah penjadwalan kelas dengan hasil yang beragam. Algoritma Random Restart Hill Climbing dan Genetic konsisten mencapai solusi global optimum. Simulated Annealing juga menunjukkan kemampuan yang baik untuk menghindari jebakan optimum lokal, memberikan keseimbangan antara kualitas solusi dan efisiensi waktu. Algoritma Hill Climbing lainnya efektif, namun menunjukkan adanya trade-off antara kecepatan komputasi dan jaminan mencapai global optimum.

Sebagai saran pengembangan lebih lanjut, menambahkan jadwal dosen sebagai komponen fungsi objektif. Selain itu, dapat dieksplorasi algoritma lainnya untuk mendapatkan data perbandingan efisiensi pencarian. Pengujian lebih lanjut pada dataset yang lebih besar dan beragam juga diperlukan untuk mengukur skalabilitas algoritma dalam memecahkan permasalahan nyata.

PEMBAGIAN TUGAS

NIM	Tugas
13523023	Stochastic Hill Climbing, Steepest Ascent Hill Climbing, Sideways Move Hill Climbing, Random Restart Hill Climbing, State, Laporan
13523051	Simulated Annealing, State, Laporan
13523111	Genetic Algorithm, State, Laporan

REFERENSI

Russell, Stuart Jonathan, and Peter Norvig. *Artificial Intelligence: A Modern Approach*.
4th ed., Pearson, 2021.