LAMPIRAN

Lampiran 1 Kode Program Python

```python
# Input dari kode ini adalah :
# n = Banyak job (integer)
# Oij = Banyak operasi dari setiap job i (list of integer)
# m = Banyak mesin (integer)
# Ωij = Alternatif mesin yang layak untuk mengerjakan operasi j pada job i (matriks jumlah(Oij) baris, setiap baris dengan n' mesin yang layak)
# Pijk = Waktu proses dari setiap alternatif mesin pada operasi j job i (matriks jumlah(Oij) baris, setiap baris dengan n' mesin yang layak)

# import library time, untuk pengaturan waktu
from time import time

# input data
# Banyak job
n = 47
# Banyak operasi dari setiap job i
Oij = [7,7,7,7,7,7,7,7,7,7,7,7,6,6,6,6,6,6,6,6,6,6,5,5,5,5,5,5,5,5,5,5,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,]
# Banyak mesin
m = 14
# Alternatif mesin yang layak untuk mengerjakan operasi j pada job i
Ωij = [[1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],[1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],
       [1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],[1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],
       [1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],[1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],
       [1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],[1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],
       [1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],[1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],
       [1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],[1,2,3,4,5],[6,7],[1,2,3],[8,9],[10,11,12,13],[1,2,3],[1,2,3,4,5],
       [1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],[1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],
       [1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],[1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],
       [1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],[1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],
       [1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],[1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],
       [1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],[1,2,4,5],[6,7],[1,2,3,4,5],[8,9],[10,11,12,13],[1,2,3,4,5],
       [10,11,12,13],[10,11,12,13],[6,7],[1,2,3,4,5],[1,2,3,4,5],[10,11,12,13],[10,11,12,13],[6,7],[1,2,3,4,5],[1,2,3,4,5],
       [10,11,12,13],[10,11,12,13],[6,7],[1,2,3,4,5],[1,2,3,4,5],[10,11,12,13],[10,11,12,13],[6,7],[1,2,3,4,5],[1,2,3,4,5],
       [10,11,12,13],[10,11,12,13],[6,7],[1,2,3,4,5],[1,2,3,4,5],[10,11,12,13],[10,11,12,13],[6,7],[1,2,3,4,5],[1,2,3,4,5],
```

```python
       [6,7],[2,3,4],[14],[6,7],[2,3,4],[14],[6,7],[2,3,4],[14],[6,7],[2,3,4],[14],[6,7],[2,3,4],[14],
       [6,7],[2,3,4],[14],[6,7],[2,3,4],[14],[6,7],[2,3,4],[14],[6,7],[2,3,4],[14],[6,7],[2,3,4],[14],
       [6,7],[2,3,4],[14],[6,7],[2,3,4],[14],[6,7],[2,3,4],[14],[6,7],[2,3,4],[14],[6,7],[2,3,4],[14]]

# Waktu proses dari setiap alternatif mesin pada operasi j job i
Pijk = [[14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [14,15,12,20,17,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,16
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [26,25,1000,30,31,1000,1000,1000,1000,1000,1000,1000,1000,1000],[1000,1000,1000,1000,1000,5,3,1000,1000,1000,1000,1000,1000,1000],[15,14,
        [1000,1000,1000,1000,1000,1000,1000,1000,1000,60,55,54,58,1000],[1000,1000,1000,1000,1000,1000,1000,1000,1000,40,36,32,38,1000],[1000,10
        [1000,1000,1000,1000,1000,1000,1000,1000,1000,60,55,54,58,1000],[1000,1000,1000,1000,1000,1000,1000,1000,1000,40,36,32,38,1000],[1000,10
        [1000,1000,1000,1000,1000,1000,1000,1000,1000,60,55,54,58,1000],[1000,1000,1000,1000,1000,1000,1000,1000,1000,40,36,32,38,1000],[1000,10
        [1000,1000,1000,1000,1000,1000,1000,1000,1000,60,55,54,58,1000],[1000,1000,1000,1000,1000,1000,1000,1000,1000,40,36,32,38,1000],[1000,10
```

```python
# import library random, untuk membantu pembangkitan bilangan acak
import random

# membuat list kosong untuk data, kemudian menambahkan nilai yang disimpan dalam variabel n, Oij, m, Qij, Pijk ke dalam list data
data=[]

data.append(n)
data.append(Oij)
data.append(m)
data.append(Qij)
data.append(Pijk)

# membuat fungsi untuk melakukan iterasi algoritma genetika
def iterasi_genetika(data, ukuran_populasi, crossover_rate, mutation_rate):

  # cek validitas data
  def cek_data_valid(data):
    sum_Oij=0
    for i in range(0,len(data[1])):
      sum_Oij+=data[1][i]
    if len(data[1])!=data[0]:
      print("Data invalid. Please check the length of Oij list")
      exit
    elif len(data[3])!=sum_Oij:
      print("Data invalid. Please check the length of Qij list")
      exit
    elif len(data[4])!=sum_Oij:
      print("Data invalid. Please check the length of Pijk list")
      exit

  cek_data_valid(data)
```

```python
  # membuat fungsi pembangkitan individu, S untuk waktu mulai proses dan X untuk mesin tempat operasi ini akan dilakukan.
  # contoh hasil individu : [0,5,0,6,0,7,3,4] -> gen sebanyak (jumlah operasi x 2)
  def inisialisasi_individual(data):
    individual=[]
    start_times=[0]*data[2]
    jobs=data[0]
    reference=random.choice([0, 1, 2])
    if reference == 1:
      a=0
      for i in range(0,jobs):
          for j in range(0,data[1][i]):
              position_X=random.randint(0,len(data[3][a])-1)
              X=data[3][a][position_X]
              S=start_times[X-1]
              individual.append(S)
              individual.append(X)
              start_times[X-1]=start_times[X-1]+data[4][a][X-1]
              a+=1
    elif reference == 2:
      a=len(data[3])-1
      for i in range(0,jobs):
          for j in range(0,data[1][i]):
              position_X=random.randint(0,len(data[3][a])-1)
              X=data[3][a][position_X]
              S=start_times[X-1]
              individual.append(S)
              individual.append(X)
              start_times[X-1]=start_times[X-1]+data[4][a][X-1]
              a-=1
    else:
```

```
# mengenerate Populasi
population = []
for _ in range(ukuran_populasi):
  individual = inisialisasi_individual(data)
  if len(individual) == 2 * sum(Oij):  # Validasi panjang individual
      population.append(individual)
  else:
      print(f"Invalid individual length: {len(individual)}")

# melakukan Crossover dengan metode Partially Mapped Crossover (PMX)
def pmx_crossover(parent1, parent2):

  # tentukan daftar indeks ganjil
  odd_indices1 = [i for i in range(len(parent1)) if i % 2 != 0]
  odd_indices2 = [i for i in range(len(parent2)) if i % 2 != 0]

  # pilih dua indeks secara acak yang berbeda dari daftar ganjil
  cx_points1 = []
  cx_points2 = []

  while len(cx_points1) < 2:
      random_index = random.choice(odd_indices1)
      if random_index not in cx_points1:
          cx_points1.append(random_index)

  while len(cx_points2) < 2:
      random_index = random.choice(odd_indices2)
      if random_index not in cx_points2:
          cx_points2.append(random_index)
```

```
# inisialisasi offspring dengan salinan parent
offspring1 = parent1.copy()
offspring2 = parent2.copy()

# tukarkan gen pada dua indeks ganjil yang dipilih
for i in range(2):
    offspring1[cx_points1[i]], offspring2[cx_points2[i]] = parent2[cx_points2[i]], parent1[cx_points1[i]]

# buat mapping berdasarkan dua indeks ganjil yang dipilih
mapping1 = {parent1[cx_points1[i]]: parent2[cx_points2[i]] for i in range(2)}
mapping2 = {parent2[cx_points2[i]]: parent1[cx_points1[i]] for i in range(2)}

def map_gene(gene, mapping):
  visited = set()  # Set untuk melacak gen yang sudah dilihat
  while gene in mapping and gene not in visited:
    visited.add(gene)  # Tambahkan gen saat ini ke dalam set
    gene = mapping[gene]  # Perbarui gen dengan nilai yang dipetakan
  return gene

# perbaiki gen di luar indeks ganjil sesuai dengan mapping
for i in range(len(parent1)):
    if i not in cx_points1:
        offspring1[i] = map_gene(offspring1[i], mapping1)
for i in range(len(parent2)):
    if i not in cx_points2:
        offspring2[i] = map_gene(offspring2[i], mapping2)

return offspring1, offspring2
```

```
# menghitung jumlah child crossover
num_children_crossover = int (ukuran_populasi * crossover_rate)

# loop untuk melakukan crossover sebanyak num_children_crossover kali
parent1 = random.choice(population)
parent2 = random.choice(population)
# Memulai dengan crossover pertama
offspring1, offspring2 = pmx_crossover(parent1, parent2)
# Melakukan crossover berulang kali
for _ in range(num_children_crossover - 1):  # Sudah sekali crossover dilakukan di atas
  offspring1, offspring2 = pmx_crossover(offspring1, offspring2)

# melakukan mutasi dengan metode Reciprocal Exchange Mutation
def reciprocal_exchange_mutation(offspring1, offspring2):

  # cek offspring hasil crossover memiliki panjang yang sama
  if len(offspring1) != len(offspring2):
      raise ValueError("Offspring must have the same length")

  # salin offspring sebagai offspring hasil mutasi
  mutated_offspring1 = offspring1.copy()
  mutated_offspring2 = offspring2.copy()

  # tukar gen yang dipilih secara acak
  mutated_offspring1[mutate_index1], mutated_offspring2[mutate_index2] = mutated_offspring2[mutate_index2], mutated_offspring1[mutate_index1]

  return mutated_offspring1, mutated_offspring2

# menghitung jumlah child crossover
num_children_mutation = int (ukuran_populasi * mutation_rate)
```

```
# Memulai dengan crossover pertama
mutated_offspring1, mutated_offspring2 = reciprocal_exchange_mutation(offspring1, offspring2)
# loop untuk melakukan mutasi sebanyak num_children_mutation kali
for i in range(num_children_mutation - 1):
  mutated_offspring1, mutated_offspring2 = reciprocal_exchange_mutation(mutated_offspring1, mutated_offspring2)

"""
Fungsi fitness dibagi menjadi dua bagian:
1. Cmax (waktu penyelesaian akhir) (makespan) dihitung dari individu
2. Batasan kendala dari masalah optimisasi digunakan untuk menghitung berapa banyak pelanggaran (fouls) yang dilakukan oleh individu tersebut
Kemudian nilai fitness dihitung dengan fitness value = cmax + fouls
"""
# membuat fungsi untuk memeriksa apakah suatu operasi dapat dikerjakan pada mesin tertentu berdasarkan data yang tersedia.
def cek_operasi_dapat_diproses_mesin_tertentu (operation,machine,data):
  Oij=data[3]
  count=0
  for i in range(0,len(Oij[operation])):
      if machine==Oij[operation][i]:
          count+=1
  if count == 0:
      return False
  else:
      return True

# membuat fungsi untuk menemukan operasi-operasi yang akan dilakukan pada mesin tertentu berdasarkan solusi yang diwakili oleh individual
def menemukan_operasi_pada_mesin_tertentu(machine,individual):
  result=[]
  i=0
  while i<len(individual):
```

```
      if individual[i+1]==machine:
          result.append(int(i/2))
      i+=2
  return result

# menambahkan offspring dan mutated offspring ke dalam populasi baru
new_population = population + [offspring1, offspring2, mutated_offspring1, mutated_offspring2]

# menghitung nilai fitness untuk setiap individu dalam populasi
fitness_values = [fitness(individual, data) for individual in new_population]
print(fitness_values)
print("\n")

# Membuat daftar pasangan individu dan nilai fitness mereka
print("List Individu dengan Nilai Fitnessnya:")
print("\n")
list_population_with_fitness = list(zip(new_population, fitness_values))
for individual, fitness in list_population_with_fitness:
    print(f"Individual: {individual}")
    print(f"Fitness: {fitness}")
    print()  # tambahkan spasi enter

# melakukan seleksi dengan metode seleksi binary tournament
def binary_tournament_selection(new_population, fitness_values):
  selected_individual = None
  lowest_fitness = float('inf')

  for _ in range(len(population)):
      idx1 = random.choice(range(len(new_population)))
```

```
    idx1 = random.choice(range(len(new_population)))
    idx2 = random.choice(range(len(new_population)))
    individual1, individual2 = new_population[idx1], new_population[idx2]

    # pastikan individu yang dipilih berbeda
    while individual1 == individual2:
        idx2 = random.choice(range(len(new_population)))
        individual2 = new_population[idx2]

    # menghitung nilai fitness dari 2 individu yang terpilih
    fitness1 = fitness_values[idx1]
    fitness2 = fitness_values[idx2]

    # mencari nilai fitness terendah
    if fitness1 < fitness2:
        if fitness1 < lowest_fitness:
            lowest_fitness = fitness1
            selected_individual = individual1
    else:
        if fitness2 < lowest_fitness:
            lowest_fitness = fitness2
            selected_individual = individual2

return selected_individual

selected_individual = binary_tournament_selection(new_population, fitness_values)
if fitness(selected_individual, data) <= 892:
    print("selected_individual:", selected_individual)
    print("nilai fitness individu terbaik:", fitness(selected_individual, data))
```

```
selected_individual = binary_tournament_selection(new_population, fitness_values)
if fitness(selected_individual, data) <= 892:
    print("selected_individual:", selected_individual)
    print("nilai fitness individu terbaik:", fitness(selected_iSndividual, data))


num_generations = 50
ukuran_populasi = 80
crossover_rate = 0.1
mutation_rate = 0.1
for generation in range(num_generations):
    print(f"\nGenerasi {generation + 1}:")
    iterasi_genetika(data, ukuran_populasi, crossover_rate, mutation_rate)
```