Program 1

Aim: Write code for min min scheduling algorithm

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

processes = ['P1', 'P2', 'P3', 'P4']

machines = ['M1', 'M2', 'M3']

execution\_times = np.array([[2, 4, 6], [5, 3, 4], [4, 2, 7], [6, 5, 3]])

completion\_times = {machine: 0 for machine in machines}

schedule = []

remaining\_processes = set(range(len(processes)))

while remaining\_processes:

    min\_time = float('inf')

    selected\_process = None

    selected\_machine = None

    for process\_idx in remaining\_processes:

        for machine\_idx, machine in enumerate(machines):

            exec\_time = execution\_times[process\_idx, machine\_idx]

            total\_time = completion\_times[machine] + exec\_time

            if total\_time < min\_time:

                min\_time = total\_time

                selected\_process = process\_idx

                selected\_machine = machine\_idx

    process\_name = processes[selected\_process]

    machine\_name = machines[selected\_machine]

    exec\_time = execution\_times[selected\_process, selected\_machine]

    start\_time = completion\_times[machine\_name]

    completion\_times[machine\_name] += exec\_time

    schedule.append((machine\_name, process\_name, start\_time, exec\_time))

    remaining\_processes.remove(selected\_process)

schedule\_df = pd.DataFrame(schedule, columns=['Machine', 'Process', 'Start', 'Duration'])

schedule\_df['End'] = schedule\_df['Start'] + schedule\_df['Duration']

print("Schedule:")

print(schedule\_df[['Process', 'Machine', 'Start', 'End']])

fig, ax = plt.subplots(figsize=(10, 6))

for machine, process, start, exec\_time in schedule:

    ax.barh(machine, exec\_time, left=start, height=0.5, color="white", edgecolor='black')

    ax.text(start + exec\_time / 2, machines.index(machine), process, ha='center', va='center', color='black')

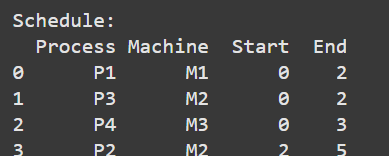
ax.set\_xlabel('Time')

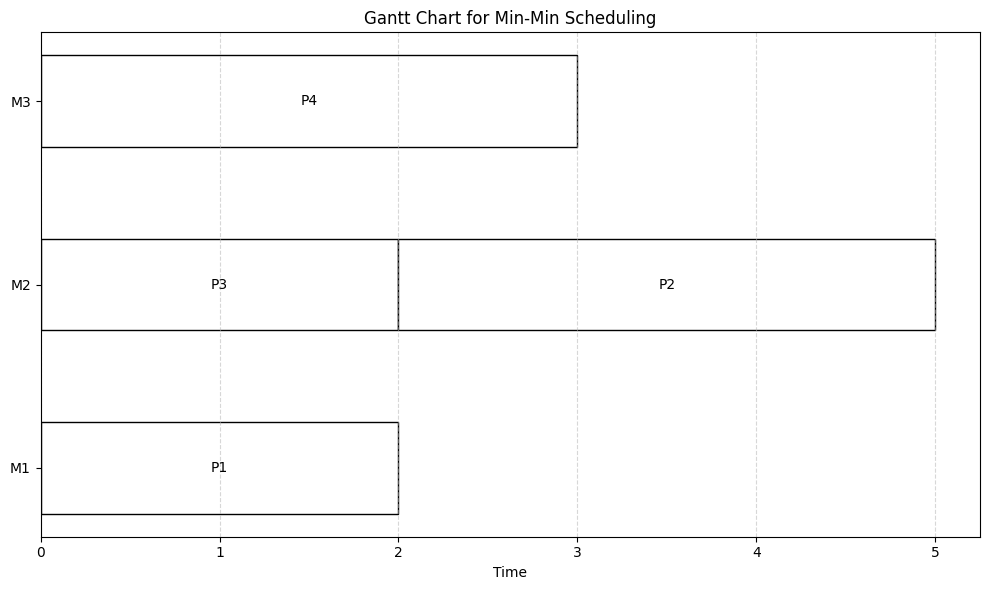
ax.set\_title('Gantt Chart for Min-Min Scheduling')

ax.grid(axis='x', linestyle='--', alpha=0.5)

plt.tight\_layout()

plt.show()





Program 2

Aim: Write code for min min scheduling algorithm with Arrival Time

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

num\_machines = 3

num\_processes = 4

execution\_times = np.array([

    [12, 15, 9],

    [8, 18, 13],

    [14, 11, 10],

    [6, 7, 12]

])

arrival\_times = [0, 3, 7, 2]

machine\_available\_times = [0] \* num\_machines

completion\_times = [0] \* num\_processes

schedule = []

unscheduled\_processes = list(range(num\_processes))

while unscheduled\_processes:

    min\_completion\_time = float('inf')

    selected\_process = -1

    selected\_machine = -1

    for process in unscheduled\_processes:

        process\_arrival = arrival\_times[process]

        for machine in range(num\_machines):

            start\_time = max(machine\_available\_times[machine], process\_arrival)

            completion\_time = start\_time + execution\_times[process][machine]

            if completion\_time < min\_completion\_time:

                min\_completion\_time = completion\_time

                selected\_process = process

                selected\_machine = machine

    process\_arrival = arrival\_times[selected\_process]

    start\_time = max(machine\_available\_times[selected\_machine], process\_arrival)

    end\_time = start\_time + execution\_times[selected\_process][selected\_machine]

    machine\_available\_times[selected\_machine] = end\_time

    completion\_times[selected\_process] = end\_time

    schedule.append({

        'Process': f'P{selected\_process}',

        'Machine': f'M{selected\_machine}',

        'Start': start\_time,

        'End': end\_time

    })

    unscheduled\_processes.remove(selected\_process)

schedule\_df = pd.DataFrame(schedule)

print("Schedule:")

print(schedule\_df)

fig, gnt = plt.subplots(figsize=(10, 6))

gnt.set\_xlabel('Time')

gnt.set\_ylabel('Machines')

gnt.set\_yticks([1, 2, 3])

gnt.set\_yticklabels(['M0', 'M1', 'M2'])

gnt.set\_title('Gantt Chart for Min-Min Scheduling Algorithm')

colors = ['white', 'white', 'white', 'white']

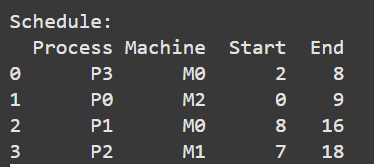
for i, task in enumerate(schedule):

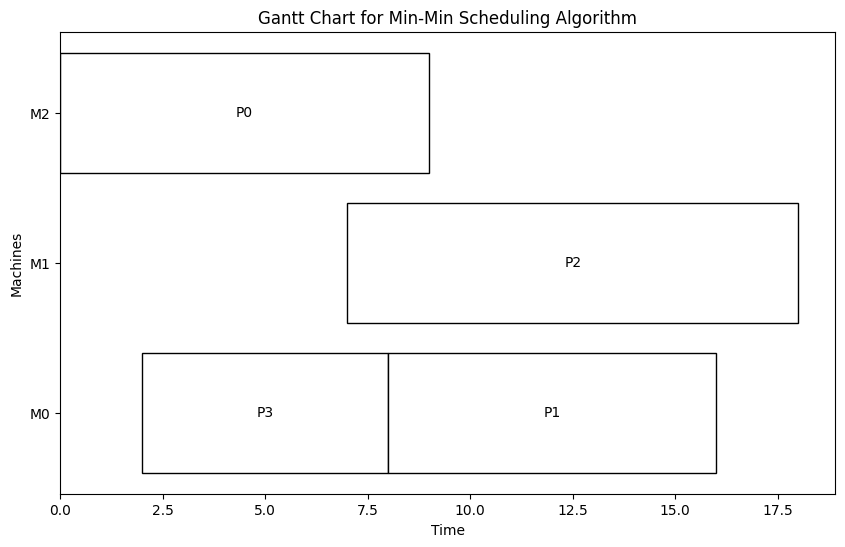
    machine\_index = int(task['Machine'][1]) + 1  # Convert machine name to index

    gnt.barh(machine\_index, left=task['Start'], width=task['End'] - task['Start'], color=colors[i % len(colors)], edgecolor='black')

    gnt.text((task['Start'] + task['End']) / 2, machine\_index, task['Process'], va='center', ha='center', color='black')

plt.show()





Program 3

Aim: Write code for min max scheduling algorithm

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

processes = ['P1', 'P2', 'P3', 'P4']

machines = ['M1', 'M2', 'M3']

execution\_times = np.array([[2, 4, 6], [5, 3, 4], [4, 2, 7], [6, 5, 3]])

completion\_times = {machine: 0 for machine in machines}

schedule = []

remaining\_processes = set(range(len(processes)))

while remaining\_processes:

    min\_exec\_times = {process\_idx: min(execution\_times[process\_idx]) for process\_idx in remaining\_processes}

    max\_of\_mins\_process = max(min\_exec\_times, key=min\_exec\_times.get)

    selected\_process = max\_of\_mins\_process

    min\_time = float('inf')

    selected\_machine = None

    for machine\_idx, machine in enumerate(machines):

        exec\_time = execution\_times[selected\_process, machine\_idx]

        total\_time = completion\_times[machine] + exec\_time

        if total\_time < min\_time:

            min\_time = total\_time

            selected\_machine = machine\_idx

    process\_name = processes[selected\_process]

    machine\_name = machines[selected\_machine]

    exec\_time = execution\_times[selected\_process, selected\_machine]

    start\_time = completion\_times[machine\_name]

    completion\_times[machine\_name] += exec\_time

    schedule.append((machine\_name, process\_name, start\_time, exec\_time))

    remaining\_processes.remove(selected\_process)

schedule\_df = pd.DataFrame(schedule, columns=['Machine', 'Process', 'Start', 'Duration'])

schedule\_df['End'] = schedule\_df['Start'] + schedule\_df['Duration']

print("Schedule:")

print(schedule\_df[['Process', 'Machine', 'Start', 'End']])

fig, ax = plt.subplots(figsize=(10, 6))

for machine, process, start, exec\_time in schedule:

    ax.barh(machine, exec\_time, left=start, height=0.5, color="white", edgecolor='black')

    ax.text(start + exec\_time / 2, machines.index(machine), process, ha='center', va='center', color='black')

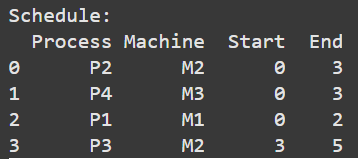
ax.set\_xlabel('Time')

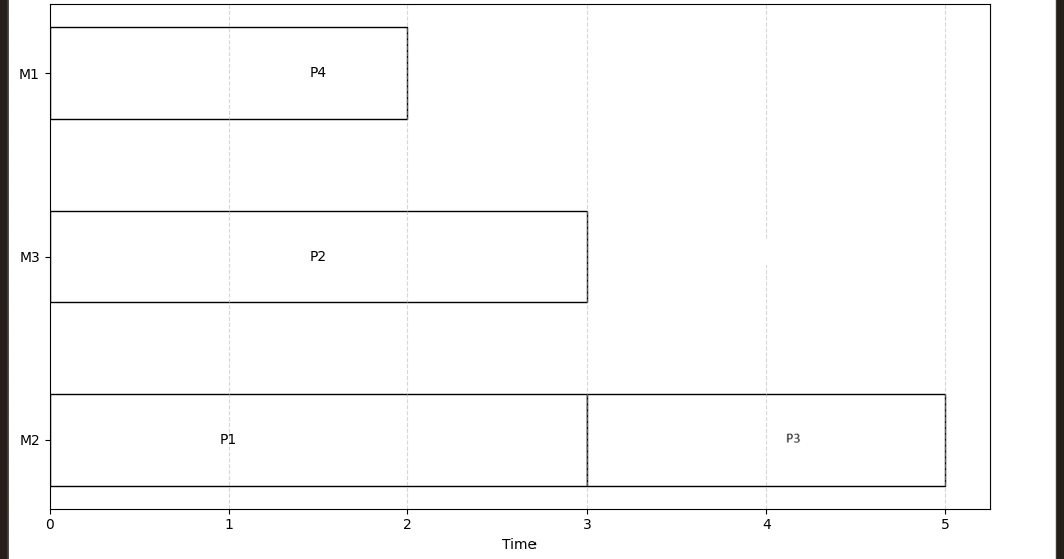
ax.set\_title('Gantt Chart for Modified Min-Min Scheduling')

ax.grid(axis='x', linestyle='--', alpha=0.5)

plt.tight\_layout()

plt.show()





Program 4

Aim: Write code for min max scheduling algorithm with arrival time

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

processes = ['P1', 'P2', 'P3', 'P4']

machines = ['M1', 'M2', 'M3']

execution\_times = np.array([[2, 4, 6], [5, 3, 4], [4, 2, 7], [6, 5, 3]])

arrival\_times = [0, 1, 2, 3]

completion\_times = {machine: 0 for machine in machines}

schedule = []

remaining\_processes = set(range(len(processes)))

current\_time = 0

while remaining\_processes:

    available\_processes = {p for p in remaining\_processes if arrival\_times[p] <= current\_time}

    if not available\_processes:

        current\_time = min(arrival\_times[p] for p in remaining\_processes)

        continue

    min\_exec\_times = {process\_idx: min(execution\_times[process\_idx]) for process\_idx in available\_processes}

    max\_of\_mins\_process = max(min\_exec\_times, key=min\_exec\_times.get)

    selected\_process = max\_of\_mins\_process

    min\_time = float('inf')

    selected\_machine = None

    for machine\_idx, machine in enumerate(machines):

        exec\_time = execution\_times[selected\_process, machine\_idx]

        total\_time = max(completion\_times[machine], current\_time) + exec\_time

        if total\_time < min\_time:

            min\_time = total\_time

            selected\_machine = machine\_idx

    process\_name = processes[selected\_process]

    machine\_name = machines[selected\_machine]

    exec\_time = execution\_times[selected\_process, selected\_machine]

    start\_time = max(completion\_times[machine\_name], current\_time)

    completion\_times[machine\_name] = start\_time + exec\_time

    schedule.append((machine\_name, process\_name, start\_time, exec\_time))

    remaining\_processes.remove(selected\_process)

    current\_time = start\_time + exec\_time

schedule\_df = pd.DataFrame(schedule, columns=['Machine', 'Process', 'Start', 'Duration'])

schedule\_df['End'] = schedule\_df['Start'] + schedule\_df['Duration']

print("Schedule:")

print(schedule\_df[['Process', 'Machine', 'Start', 'End']])

fig, ax = plt.subplots(figsize=(10, 6))

for machine, process, start, exec\_time in schedule:

    ax.barh(machine, exec\_time, left=start, height=0.5, color="white", edgecolor='black')

    ax.text(start + exec\_time / 2, machines.index(machine), process, ha='center', va='center', color='black')

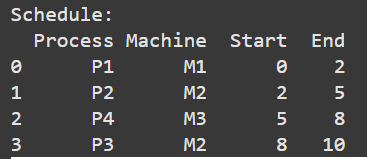
ax.set\_xlabel('Time')

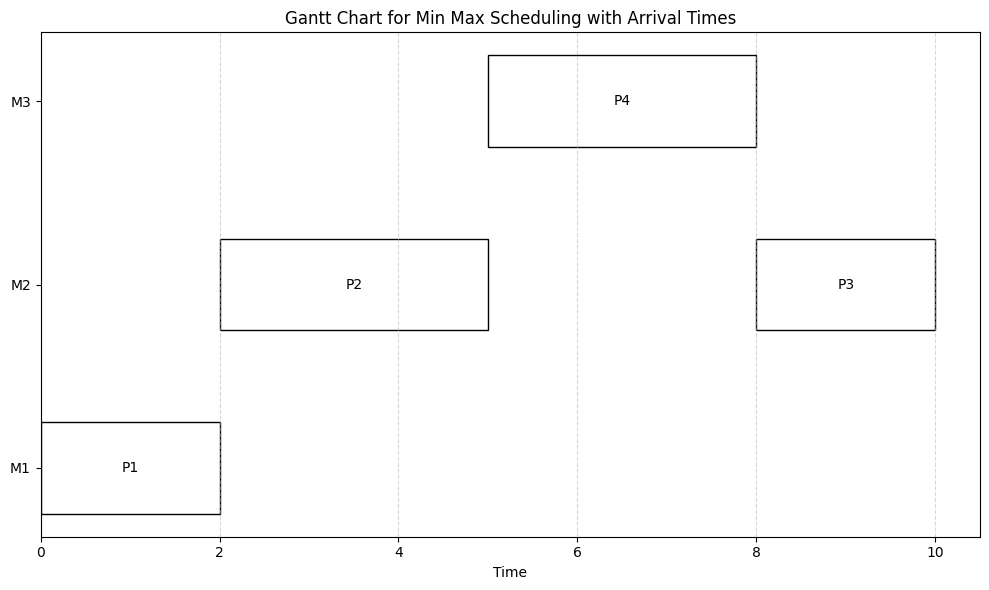
ax.set\_title('Gantt Chart for Min Max Scheduling with Arrival Times')

ax.grid(axis='x', linestyle='--', alpha=0.5)

plt.tight\_layout()

plt.show()





Program 5

Aim: Write code for round robin scheduling algorithm

import matplotlib.pyplot as plt

import pandas as pd

processes = ['P1', 'P2', 'P3', 'P4', 'P5', 'P6']

machines = ['M1', 'M2', 'M3']

execution\_times = [3, 2, 4, 1, 5, 3]

arrival\_times = [0, 1, 2, 3, 4, 5]

completion\_times = {machine: 0 for machine in machines}

schedule = []

machine\_idx = 0

remaining\_processes = set(range(len(processes)))

while remaining\_processes:

    available\_processes = {p for p in remaining\_processes if arrival\_times[p] <= completion\_times[machines[machine\_idx]]}

    if not available\_processes:

        current\_machine\_time = min(arrival\_times[p] for p in remaining\_processes)

        available\_processes = {p for p in remaining\_processes if arrival\_times[p] <= current\_machine\_time}

    selected\_process = min(available\_processes, key=lambda p: arrival\_times[p])

    machine\_name = machines[machine\_idx]

    process\_name = processes[selected\_process]

    exec\_time = execution\_times[selected\_process]

    start\_time = max(completion\_times[machine\_name], arrival\_times[selected\_process])

    completion\_times[machine\_name] = start\_time + exec\_time

    schedule.append((machine\_name, process\_name, start\_time, exec\_time))

    machine\_idx = (machine\_idx + 1) % len(machines)

    remaining\_processes.remove(selected\_process)

schedule\_df = pd.DataFrame(schedule, columns=['Machine', 'Process', 'Start', 'Duration'])

schedule\_df['End'] = schedule\_df['Start'] + schedule\_df['Duration']

print("Schedule:")

print(schedule\_df[['Process', 'Machine', 'Start', 'End']])

fig, ax = plt.subplots(figsize=(10, 6))

for machine, process, start, exec\_time in schedule:

    ax.barh(machine, exec\_time, left=start, height=0.5, color="white", edgecolor='black')

    ax.text(start + exec\_time / 2, machines.index(machine), process, ha='center', va='center', color='black')

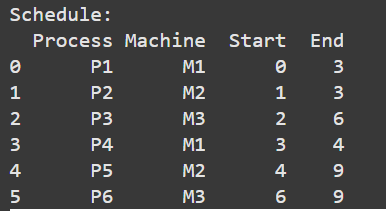
ax.set\_xlabel('Time')

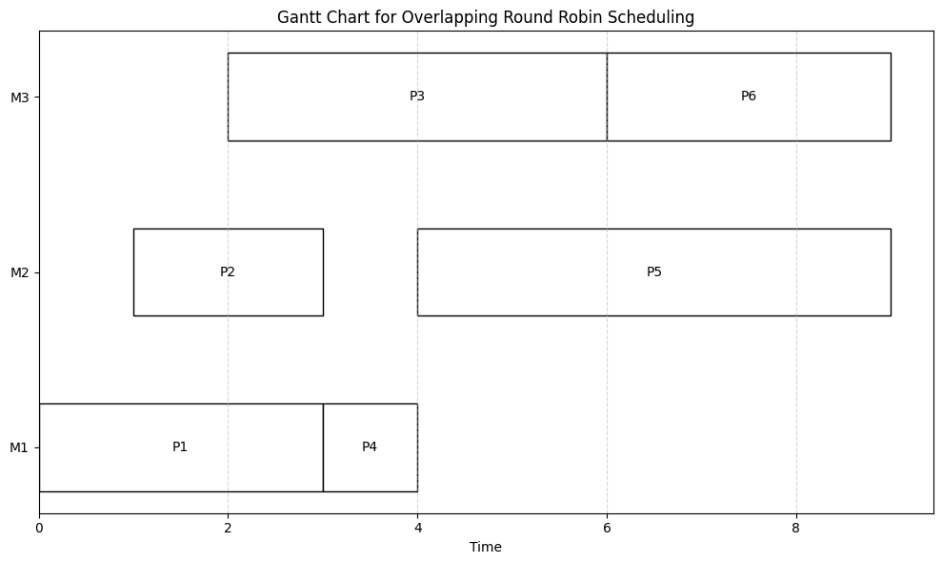
ax.set\_title('Gantt Chart for Overlapping Round Robin Scheduling')

ax.grid(axis='x', linestyle='--', alpha=0.5)

plt.tight\_layout()

plt.show()





Program 6

Aim: Write code for weighted round robin scheduling algorithm

import matplotlib.pyplot as plt

import pandas as pd

processes = ['P1', 'P2', 'P3', 'P4', 'P5', 'P6']

machines = ['M1', 'M2', 'M3']

execution\_times = [3, 2, 4, 1, 5, 3]

arrival\_times = [0, 1, 2, 3, 4, 5]

machine\_weights = [2, 1, 2]

completion\_times = {machine: 0 for machine in machines}

schedule = []

current\_time = 0

remaining\_processes = set(range(len(processes)))

machine\_idx = 0

machine\_usage\_count = 0

while remaining\_processes:

    available\_processes = {p for p in remaining\_processes if arrival\_times[p] <= current\_time}

    if not available\_processes:

        current\_time = min(arrival\_times[p] for p in remaining\_processes)

        continue

    selected\_process = min(available\_processes, key=lambda p: arrival\_times[p])

    machine\_name = machines[machine\_idx]

    process\_name = processes[selected\_process]

    exec\_time = execution\_times[selected\_process]

    start\_time = max(completion\_times[machine\_name], arrival\_times[selected\_process])

    completion\_times[machine\_name] = start\_time + exec\_time

    schedule.append((machine\_name, process\_name, start\_time, exec\_time))

    remaining\_processes.remove(selected\_process)

    current\_time = start\_time + exec\_time

    machine\_usage\_count += 1

    if machine\_usage\_count >= machine\_weights[machine\_idx]:

        machine\_idx = (machine\_idx + 1) % len(machines)

        machine\_usage\_count = 0

schedule\_df = pd.DataFrame(schedule, columns=['Machine', 'Process', 'Start', 'Duration'])

schedule\_df['End'] = schedule\_df['Start'] + schedule\_df['Duration']

print("Schedule:")

print(schedule\_df[['Process', 'Machine', 'Start', 'End']])

fig, ax = plt.subplots(figsize=(10, 6))

for machine, process, start, exec\_time in schedule:

    ax.barh(machine, exec\_time, left=start, height=0.5, color="white", edgecolor='black')

    ax.text(start + exec\_time / 2, machines.index(machine), process, ha='center', va='center', color='black')

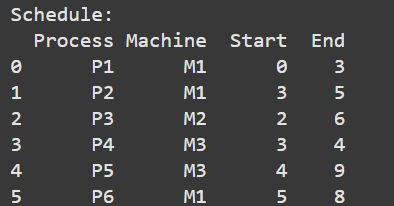
ax.set\_xlabel('Time')

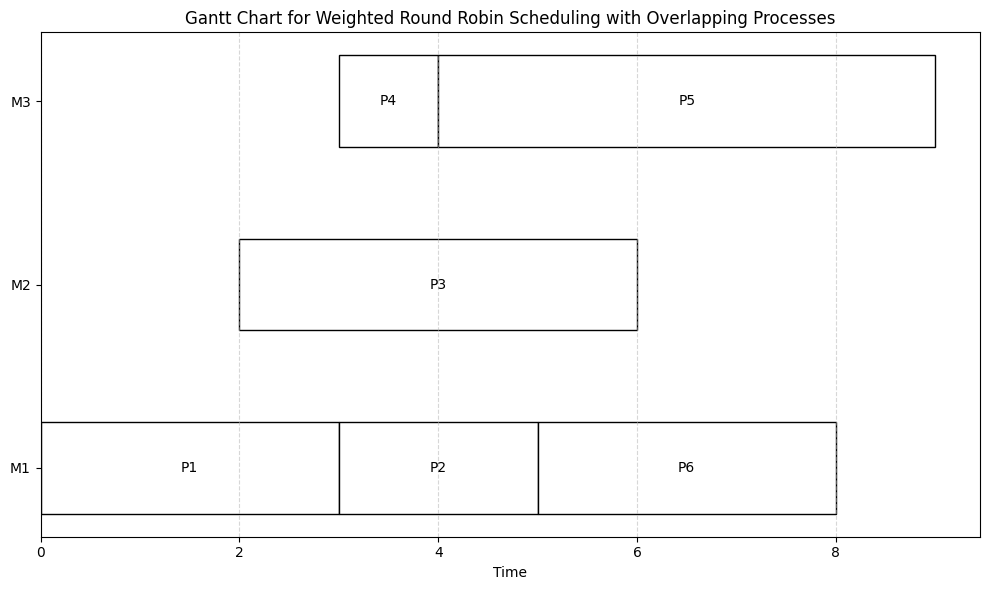
ax.set\_title('Gantt Chart for Weighted Round Robin Scheduling with Overlapping Processes')

ax.grid(axis='x', linestyle='--', alpha=0.5)

plt.tight\_layout()

plt.show()





Program 7

Aim: Write code for Ant Colony Optimization

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

from collections import defaultdict

NUM\_ANTS = 10

NUM\_ITERATIONS = 100

ALPHA = 1.0

BETA = 5.0

EVAPORATION\_RATE = 0.5

Q = 10

def create\_graph():

    G = nx.complete\_graph(5)

    pos = nx.spring\_layout(G)

    weights = {edge: np.random.randint(1, 10) for edge in G.edges()}

    nx.set\_edge\_attributes(G, weights, 'weight')

    return G, pos

def initialize\_pheromones(graph):

    return {edge: 1 for edge in graph.edges()}

def select\_next\_city(current\_city, unvisited, pheromones, distances):

    probabilities = []

    total = 0

    for city in unvisited:

        pheromone = pheromones[(min(current\_city, city), max(current\_city, city))]

        distance = distances[(min(current\_city, city), max(current\_city, city))]

        prob = (pheromone \* ALPHA) \* ((1 / distance) \* BETA)

        probabilities.append((city, prob))

        total += prob

    probabilities = [(city, prob / total) for city, prob in probabilities]

    next\_city = np.random.choice([city for city, \_ in probabilities],

                                 p=[prob for \_, prob in probabilities])

    return next\_city

def update\_pheromones(pheromones, ants\_paths, distances):

    for edge in pheromones:

        pheromones[edge] \*= (1 - EVAPORATION\_RATE)

    for path, length in ants\_paths:

        for i in range(len(path) - 1):

            edge = (min(path[i], path[i + 1]), max(path[i], path[i + 1]))

            pheromones[edge] += Q / length

def ant\_colony\_optimization(graph, num\_ants, num\_iterations):

    pheromones = initialize\_pheromones(graph)

    distances = nx.get\_edge\_attributes(graph, 'weight')

    best\_path = None

    best\_length = float('inf')

    for \_ in range(num\_iterations):

        ants\_paths = []

        for \_ in range(num\_ants):

            path = [0]

            unvisited = set(graph.nodes) - {0}

            while unvisited:

                current\_city = path[-1]

                next\_city = select\_next\_city(current\_city, unvisited, pheromones, distances)

                path.append(next\_city)

                unvisited.remove(next\_city)

            path.append(path[0])

            length = sum(distances[(min(path[i], path[i + 1]), max(path[i], path[i + 1]))]

                         for i in range(len(path) - 1))

            ants\_paths.append((path, length))

            if length < best\_length:

                best\_length = length

                best\_path = path

        update\_pheromones(pheromones, ants\_paths, distances)

    return best\_path, best\_length

def visualize(graph, pos, path):

    plt.figure(figsize=(8, 6))

    nx.draw(graph, pos, with\_labels=True, node\_color='lightblue', node\_size=700, font\_size=12)

    edge\_labels = nx.get\_edge\_attributes(graph, 'weight')

    nx.draw\_networkx\_edge\_labels(graph, pos, edge\_labels=edge\_labels, font\_color='red')

    path\_edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]

    nx.draw\_networkx\_edges(graph, pos, edgelist=path\_edges, width=3, edge\_color='blue')

    plt.title(f"Best Path with Length: {best\_length}")

    plt.show()

graph, pos = create\_graph()

best\_path, best\_length = ant\_colony\_optimization(graph, NUM\_ANTS, NUM\_ITERATIONS)

print(f"Best path: {best\_path} with length {best\_length}")

visualize(graph, pos, best\_path)

Best path: [0, 3, 1, 2, 4, 0] with length 12

