import numpy as np

from ortools.sat.python import cp\_model

def optimize\_job\_scheduling(jobs, resources, time\_limits):

"""

Optimize job scheduling and resource allocation.

Args:

jobs (list of tuples): Each tuple contains (job\_id, duration, resource\_requirements).

resource\_requirements is a list of resource usage for the job.

resources (list of int): List of available quantities for each resource.

time\_limits (int): Maximum allowable time for all jobs to be completed.

Returns:

dict: A schedule with job start times and resource allocation.

"""

# Create the model

model = cp\_model.CpModel()

num\_jobs = len(jobs)

num\_resources = len(resources)

# Variables

start\_times = [model.NewIntVar(0, time\_limits, f'start\_{j}') for j in range(num\_jobs)]

end\_times = [model.NewIntVar(0, time\_limits, f'end\_{j}') for j in range(num\_jobs)]

intervals = []

demands = []

for j, (job\_id, duration, resource\_req) in enumerate(jobs):

end\_time = model.NewIntVar(0, time\_limits, f'end\_{job\_id}')

intervals.append(model.NewIntervalVar(start\_times[j], duration, end\_time, f'interval\_{job\_id}'))

demands.append(resource\_req)

# Cumulative resource constraints

for r in range(num\_resources):

resource\_profile = [demands[j][r] for j in range(num\_jobs)]

model.AddCumulative(intervals, resource\_profile, resources[r])

# Precedence constraints (optional: jobs requiring sequential execution)

for j1, j2 in [(0, 1)]: # Add your specific dependencies here

model.Add(start\_times[j1] + jobs[j1][1] <= start\_times[j2])

# Objective: minimize makespan (maximum end time)

makespan = model.NewIntVar(0, time\_limits, 'makespan')

model.AddMaxEquality(makespan, [end\_times[j] for j in range(num\_jobs)])

model.Minimize(makespan)

# Solve the model

solver = cp\_model.CpSolver()

status = solver.Solve(model)

if status in (cp\_model.OPTIMAL, cp\_model.FEASIBLE):

schedule = {

jobs[j][0]: {

'start\_time': solver.Value(start\_times[j]),

'end\_time': solver.Value(end\_times[j])

} for j in range(num\_jobs)

}

return {

'schedule': schedule,

'makespan': solver.Value(makespan)

}

else:

return None

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

jobs = [

("Job1", 3, [2, 1]),

("Job2", 2, [1, 3]),

("Job3", 4, [3, 2])

]

resources = [5, 5] # Resource capacities

time\_limits = 10 # Max allowable time

result = optimize\_job\_scheduling(jobs, resources, time\_limits)

if result:

print("Optimized Schedule:")

for job, schedule in result['schedule'].items():

print(f"{job}: Start at {schedule['start\_time']}, End at {schedule['end\_time']}")

print(f"Makespan: {result['makespan']}")

else:

print("No feasible schedule found.")