Project 2: The Job Shop Scheduling Problem

The Job-Shop problem is found in a typical factory environment where different tasks or operations must be performed to complete a job. A number of jobs are processing at any one time on a number of machines. A schedule must be derived which aims to complete all jobs as quickly as possible on the given production plant.

We are given a set of jobs \( J \), each comprising of an ordered series of precedence-constrained operations. Each job-operation pair \( (j,i) \) is to be processed on a specified machine \( M(j,i) \) for \( T(j,i) \) time units. The matrices \( M \) and \( T \) are given in advance. We are also given due dates/times for each job. The goal of this optimization problem is to find a non-preemptive schedule such that no machine processes more than one operation at a time and such that the “tardiness” of the schedule is minimized. Once the operation sequences are established for each machine, the JSP reduces to a longest path problem in an directed acyclic graph.

Input Data:

- **NUMOFMC**: Number of Machines
- **\( J \)**: Number of Jobs to be performed on the given machines
- **\( I \)**: Maximum possible number of Operations to be performed for a Job
- **\( M \)**: Matrix of Machine Assignment for Operation-Job pair
- **\( T \)**: Matrix of processing times for Operation-Job pair
- **DDate**: Vector of due dates for each Job

A schedule \( s \) specifies the order in which the operations are done on the machines. For eg. Consider the simple job shop problem (discussed in class) with matrix \( M \) given as follows:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

The network representation has \( I \times J \) nodes (besides the source and sink nodes), and each node is identified by node number given from 1 : \( I \times J \). So a random permutation of numbers 1 to \( I \times J \) specifies a schedule for the problem.

The nodes (1,1), (1,2), … (3,3) are numbered as 1, 2 … 9. The schedule \( s = [5,1,4,7,2,8,3,6,9] \) represents the following ordering of operations:

- Machine 1 : (2,2) → (1,1) → (3,1) (since 5 precedes 1, which precedes 7 in \( s \))
- Machine 2 : (2,1) → (1,2) → (3,2) (since 4 precedes 2, which precedes 8 in \( s \))
- Machine 3 : (1,3) → (2,3) → (3,3) (since 3 precedes 6, which precedes 9 in \( s \))
**Objective Function:** The objective function is to minimize the following:

\[ \sum \max(0, C(j) - DDate(j)) \]

where \( C(j) \) is the completion time for job \( j \).

You are provided with a `cost.m` file (available on the course homepage) which takes in a schedule \( s \) and computes the above value. The schedule \( s \) should satisfy the **precedence constraints** due to the order in which the operations need to be done for each of the jobs and also the **sequencing constraints** to ensure at most one job is performed on a machine at a given time. A randomly generated schedule can turn out to be **infeasible** due to generation of cycles in the network, making it impossible to compute the longest paths to all the nodes. Infeasible schedules are given an arbitrarily high cost of \( 1e5 \). For the purposes of this project, data for a 6 job, 6 machine problem are provided (in files M1.txt, T1.txt, and Ddate1.txt). The schedule vector \( s \) is therefore a permutation of numbers from 1 to 36.

**When submitting your final report for this project, please indicate the best schedule you were able to find, its associated cost, and show the corresponding Gantt chart.**

**Matlab Files provided:**

`ws.m` - preparing the workspace, loading variables into memory

`cost.m` - computes the tardiness of the schedule
    assumes the variables have been loaded into the memory

`genM.m` - consists of the function `genM(s)` which generates the matrix for schedule \( s \)

`computeCtime.m` - computes the completion times for all the jobs called by `cost`

`doLabel.m` - topologically sorts the nodes in the graph & returns
    the array of the nodes in this order and also returns
    0/1 for the presence of a cycle
    It is called by `computeCtime`.

Note: you will only need to run `ws.m` and `cost.m`. The other MATLAB files are required for `cost.m` to work.