Assignment 1

There are m jobs to be processed on n machines. The time for job i to be processed on machine j is τ_{ij}, and the cost is c_{ij}. The available time of machine j is T_j. The goal is to assign jobs to machines such that:

1. **The total production cost is minimized.**
2. **The processing time on every machine does not exceed its available time.**
3. **Each job must be assigned to exactly one machine.**

Mathematical Formulation

This problem can be formulated as an integer linear program.

Indices

- i: Index for jobs, $i \in \{1, 2, \ldots, m\}$.
- j: Index for machines, $j \in \{1, 2, \ldots, n\}$.

Parameters

- c_{ij}: The cost for processing job i on machine j.
- τ_{ij}: The time required for job i to be processed on machine j.
- T_j: The total available time on machine j.

Decision Variables

Let x_{ij} be a binary variable defined as:

$$x_{ij} = \begin{cases} 1, & \text{if job } i \text{ is assigned to machine } j \\ 0, & \text{otherwise} \end{cases}$$

Objective Function

The objective is to minimize the total production cost, which is the sum of the costs of all assignments made.

$$\min Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

Constraints

The assignment is subject to the following constraints:

1. **Job Assignment Constraint:** Each job must be assigned to exactly one machine. This means that for each job i, the sum of its assignments across all machines must be 1.

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, m\} \tag{2}$$

2. **Machine Capacity Constraint:** The total time of all jobs assigned to a machine must not exceed its available time.

$$\sum_{i=1}^{m} \tau_{ij} x_{ij} \le T_j, \quad \forall j \in \{1, 2, \ldots, n\} \tag{3}$$

The problem can also be expressed in a compact standard form.

Let $\mathbf{x}$ be the vector of all decision variables x_{ij}, flattened into a single column vector of size $mn \times 1$. For instance, by stacking the columns of the decision variable matrix:

$$\mathbf{x} = [x_{11}, x_{21}, \dots, x_{m1}, x_{12}, \dots, x_{m2}, \dots, x_{1n}, \dots, x_{mn}]^T, x_{ij} \in \{0, 1\}$$

Let $\mathbf{c}$ be the corresponding vector of costs, arranged in the same order:

$$\mathbf{c} = [c_{11}, c_{21}, \dots, c_{m1}, c_{12}, \dots, c_{m2}, \dots, c_{1n}, \dots, c_{mn}]^T$$

The optimization problem can then be written as:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & A_{eq}\mathbf{x} = \mathbf{b}_{eq} \\ & A_{ineq}\mathbf{x} \leq \mathbf{b}_{ineq} \\ & \mathbf{x} \in \{0, 1\}^{mn} \end{aligned}$$

1. A_{eq} is an $m \times mn$ matrix that represents the job assignment constraints. Each row corresponds to a single job and contains 1s for the variables associated with that job being assigned to any machine.

2. $\mathbf{b}_{eq}$ is an $m \times 1$ vector of ones.

3. A_{ineq} is an $n \times mn$ matrix representing the machine capacity constraints. Each row corresponds to a machine and contains the processing times τ_{ij} for that machine.

4. $\mathbf{b}_{ineq}$ is an $n \times 1$ vector of machine time capacities $[T_1, T_2, \dots, T_n]^T$.

*Solution

For a specific question, we can solve the problem. In python,we use the following codes to generate the necessary data for the problem:

```
num_jobs = 16
num_machines = 4
costs = np.random.randint(10, 100, size=(num_jobs, num_machines))
times = np.random.randint(5, 25, size=(num_jobs, num_machines))
machine_capacities = np.random.randint(60, 90, size=num_machines)
```

We can use ortool to solve optimise problems:

```
model = cp_model.CpModel()
x = {}
for i in range(num_jobs):
    for j in range(num_machines):
        x[i, j] = model.NewBoolVar(f'x_{i}_{j}')

for i in range(num_jobs):
    model.AddExactlyOne(x[i, j] for j in range(num_machines))

for j in range(num_machines):
    total_time_on_machine = sum(times[i, j] * x[i, j] for i in range(num_jobs))
    model.Add(total_time_on_machine <= machine_capacities[j])

total_cost = sum(costs[i, j] * x[i, j]

for i in range(num_jobs) for j in range(num_machines))
model.Minimize(total_cost)

solver = cp_model.CpSolver()
status = solver.Solve(model)
```

The solution is in the figure:

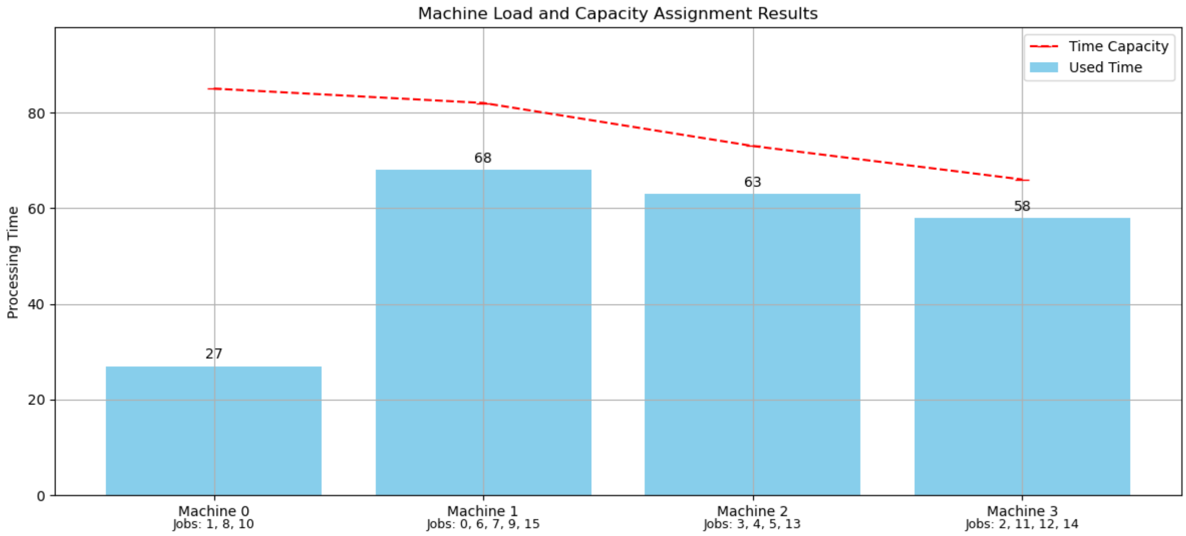

Figure 1: Solution