Project Assignment: Task Scheduling with Eligibility Constraints and TA-dependent Durations

Mohamad Jehad
mohamadjehad@aucegypt.edu

October 2025

Contents

1 Problem Definition and Formal Setup (corrected)

This project studies a scheduling problem where tasks require specific skills and each task can only be performed by a subset of teaching assistants (TAs). Furthermore, a task's processing time may depend on which TA performs it.

1.1 System Parameters

- **TAs:** Let $T = \{t_1, t_2, \ldots, t_m\}$ denote the set of m teaching assistants (general $m \geq 1$).
- **Tasks:** Let $P = \{1, 2, \ldots, n\}$ be the set of tasks (jobs) to schedule.
- **Eligibility:** Each task $i \in P$ has an eligibility set $E_i \subseteq T$. Task i may only be assigned to a TA $t \in E_i$. (In the base instance of the assignment, $|E_i| = 2$ for all i, but we allow general E_i.)
- **Processing times:** For each task i and TA t, let $p_{i,t}$ denote the processing time of task i if assigned to TA t. If $t \notin E_i$ then assignment is forbidden (equivalently $p_{i,t} = +\infty$ or simply undefined).
- **Assignment mapping:** An assignment is a mapping $\phi : P \to T$ such that $\phi(i) \in E_i$ for every task i.
- **Load of a TA:** Given assignment ϕ, the load of TA t is:

$$L_t(\phi) \;=\; \sum_{i:\,\phi(i)=t} p_{i,t}.$$

- **Makespan:** The makespan is the maximum load across all TAs:

$$C_{\max}(\phi) \;=\; \max_{t \in T} L_t(\phi).$$

- **Objective:** Find an assignment ϕ minimizing the makespan. Denote

$$C^* \;=\; \min_{\phi:\,\phi(i)\in E_i} C_{\max}(\phi).$$

2 Brute-Force (Exact) Approach for the Corrected Model

2.1 Enumeration and complexity

When each task i has eligibility set E_i, the total number of feasible assignments is:

$$\prod_{i=1}^{n} |E_i|.$$

Computing the makespan for one assignment requires summing n terms (distributed across TAs), so brute-force runtime is

$$O\left(n \cdot \prod_{i=1}^{n} |E_i|\right).$$

In the worst case when every task is eligible on all m TAs, this becomes $O(n \cdot m^n)$, which is exponential in n.

2.2 Brute-force pseudocode

Algorithm 1 Brute force over eligible assignments

```
Input: tasks P, TAs T, eligibilities {E_i}, times {p_{i,t}}.
bestMakespan ← +∞
bestAssign ← null
function RecurseAssign(i, currentAssign)
    if i ¿ n then
        compute loads L_t from currentAssign
        makespan ← max_t L_t
        if makespan ¡ bestMakespan then
            bestMakespan ← makespan; bestAssign ← currentAssign
        end if
        return
    end if
    for each t ∈ E_i do
        currentAssign[i] ← t
        RecurseAssign(i+1, currentAssign)
    end for
end function
RecurseAssign(1, emptyAssign)
return bestMakespan, bestAssign
```

3 Worked Example (small, uses eligibility and TA-dependent times)

Consider $T = \{\text{A1}, \text{A2}, \text{A3}\}$ and two tasks A, B.

- Task A: $E_A = \{\text{A1}, \text{A3}\}$, with $p_{A,\text{A1}} = 10,\ p_{A,\text{A3}} = 7$.
- Task B: $E_B = \{\text{A2}, \text{A3}\}$, with $p_{B,\text{A2}} = 8,\ p_{B,\text{A3}} = 12$.

Feasible assignments: choose one TA from E_A for task A and one from E_B for task B, so total $2 \times 2 = 4$ assignments.

Table 1: Feasible assignments and resulting loads (TA-dependent times)

Assignment	Description	Loads $(L_{\text{A1}}, L_{\text{A2}}, L_{\text{A3}})$	Makespan
1	$A \to \text{A1},\ B \to \text{A2}$	$(10,\ 8,\ 0)$	10
2	$A \to \text{A1},\ B \to \text{A3}$	$(10,\ 0,\ 12)$	12
3	$A \to \text{A3},\ B \to \text{A2}$	$(0,\ 8,\ 7)$	8
4	$A \to \text{A3},\ B \to \text{A3}$	$(0,\ 0,\ 7+12=19)$	19

From the table, the optimal makespan is $C^* = 8$ achieved by assignment (3): $A \to$ A3, $B \to$ A2.

4 Eligibility-aware LPT with Constraint Prioritization (Uniform Durations)

We now assume that each task i has a uniform processing time p_i for all eligible TAs ($p_{i,t} = p_i$ for $t \in E_i$). The objective remains minimizing the makespan $C_{\max}$. This version represents the restricted assignment problem with uniform processing times.

4.1 Heuristic idea

We adapt the classical LPT (Longest Processing Time first) strategy to respect eligibility and prioritize the most constrained TAs (those eligible for fewer tasks). The heuristic combines:

1. Sorting tasks by descending p_i (LPT principle),
2. Always assigning to the least-loaded eligible TA,
3. Breaking ties by choosing the most constrained TA (minimum eligibility count).

4.2 Pseudocode

Algorithm 2 Eligibility-aware LPT with Constraint Prioritization

Input: tasks P with times $\{p_i\}$, TAs T, eligibilities $\{E_i\}$
Initialize loads $L_t \leftarrow 0$ for all $t \in T$
Compute TA constraint counts $C_t = |\{i \in P \mid t \in E_i\}|$
Sort tasks by p_i descending (LPT order)
for each task i in sorted list **do**
    $E_i =$ eligible TAs for task i
    Find $L_{\min} = \min_{t \in E_i} L_t$
    $T_{\text{best}} = \{t \in E_i \mid L_t = L_{\min}\}$
    Choose $t^* = \arg\min_{t \in T_{\text{best}}} C_t$
    Assign $i \to t^*$
    Update $L_{t^*} \leftarrow L_{t^*} + p_i$
end for
Return assignment and makespan $\max_t L_t$

4.3 Example: worked instance for the uniform-duration algorithm

We use a similar small instance but now we set uniform processing times (same p_i for every eligible TA).

Let $T = \{\text{A1}, \text{A2}, \text{A3}\}$ and two tasks A, B.

- Task A: $E_A = \{\text{A1}, \text{A3}\}$, with uniform $p_A = 10$.
- Task B: $E_B = \{\text{A2}, \text{A3}\}$, with uniform $p_B = 8$.

Compute TA constraint counts:

$$C_{\text{A1}} = 1, \quad C_{\text{A2}} = 1, \quad C_{\text{A3}} = 2.$$

Apply the algorithm (LPT order: $A(10)$, then $B(8)$):

1. Assign task A: eligible $\{\text{A1}, \text{A3}\}$. Current loads all zero, so $L_{\min} = 0$ and $T_{\text{best}} = \{\text{A1}, \text{A3}\}$. Tie-breaker chooses the most constrained TA (min C_t), which is A1 (since $C_{\text{A1}} = 1 < C_{\text{A3}} = 2$). Assign $A \to \text{A1}$. Loads become $(10, 0, 0)$.
2. Assign task B: eligible $\{\text{A2}, \text{A3}\}$. Current loads for eligible TAs are $(0, 0)$, so $L_{\min} = 0$ and $T_{\text{best}} = \{\text{A2}, \text{A3}\}$. Tie-breaker picks A2 ($C_{\text{A2}} = 1$). Assign $B \to \text{A2}$. Final loads $(10, 8, 0)$, makespan 10.

Table 2: Algorithm assignments and resulting loads (uniform durations)

Step	Assignment	Loads $(L_{\text{A1}}, L_{\text{A2}}, L_{\text{A3}})$	Makespan
After A	$A \to \text{A1}$	(10, 0, 0)	10
After B	$B \to \text{A2}$	(10, 8, 0)	10

In this instance the algorithm attains makespan 10. The optimal makespan C^* is also 10 (you can check all feasible assignments), so the heuristic is optimal here.

4.4 Approximation guarantee: 2-approximation proof

We now prove that the above algorithm is a **2-approximation**: let ALG denote the makespan produced by the algorithm and C^* the optimal makespan; then $\text{ALG} \leq 2C^*$.

Proof. Let $\text{ALG} = \max_t L_t$ be the maximum load in the schedule produced by the algorithm, and let t^* be a TA achieving this maximum load. Let j be the last job (task) assigned to t^* by the algorithm, and let p_j be its processing time. Denote by L'_{t^*} the load of t^* immediately before assigning j; thus

$$\text{ALG} = L'_{t^*} + p_j.$$

Observe first that $p_j \leq C^*$. Indeed, C^* is at least the size of the largest job (otherwise that job alone would exceed C^*), so every job's processing time is at most C^*.

Assume, for contradiction, that $\text{ALG} > 2C^*$. Then

$$L'_{t^*} = \text{ALG} - p_j > 2C^* - p_j \geq 2C^* - C^* = C^*,$$

so $L'_{t^*} > C^*$.

By the algorithm's rule (assign each job to the eligible TA with minimum current load, breaking ties by constraint count), at the moment we assigned job j every TA in E_j had load $\geq L'_{t^*} > C^*$. Therefore the total load (sum of current loads) on the machines in E_j strictly exceeds $|E_j| \cdot C^*$.

Now consider the optimal schedule of makespan C^*. In OPT, each TA has load at most C^*, therefore the total load on the same set of machines E_j in OPT is at most $|E_j| \cdot C^*$. But the set of jobs assigned to E_j in the algorithm at that time is a subset of all jobs assigned to E_j in total (across the entire run), and the sum of those loads already exceeds $|E_j| \cdot C^*$, which cannot be matched by OPT's at-most-$|E_j| \cdot C^*$ capacity on those machines. This contradiction implies our assumption ALG $> 2C^*$ is false.

Hence ALG $\leq 2C^*$, i.e., the algorithm is a 2-approximation.

4.5 Remarks

- The bound is (worst-case) tight up to constant factors for this simple greedy: there are instances where the algorithm can produce a schedule arbitrarily close to $2C^*$.
- In practice, the LPT ordering and the constraint-based tie-breaker often produce results much better than the worst-case bound—especially on instances where constrained TAs are protected early.
- Further improvements are possible: local moving/post-processing, or LP-based techniques (Lenstra–Shmoys–Tardos style) can yield better approximation factors if theoretical guarantees beyond 2 are required.

5 Evaluation Plan

1. Implement both the exact brute-force method and this heuristic.
2. Generate test instances with different n, m, and $|E_i|$.
3. Compare makespan results and compute
$$\text{approx\_ratio} = \frac{C_{\max}(\text{heuristic})}{C^*}.$$
4. Report runtime and performance comparison.

Appendix: Example Results Table

Table 3: Example comparison between brute-force and heuristic

Instance	n	m	brute C^*	heuristic C	ratio
inst-1	8	3	42	44	1.048
inst-2	10	4	56	59	1.053