PROJECT SCHEDULING PROBLEMS WITH LINEAR PROGRAMMING

IRIS XU , TIM LEE , KEVIN JIA , AND ALEX LIN

Abstract. The Project Scheduling Problem (PSP) is a fundamental optimization problem in operations research, where the goal is to determine an optimal schedule for a set of interdependent tasks while minimizing the total project duration. We focus on formulating the precedence-constrained PSP as a linear programming (LP) model, ensuring that task dependencies are met while minimizing overall completion time. We then extend our discussion to the time-cost trade-off problem (TCTP). In addition, we analyze the sensitivity of the schedule using the dual problem, providing insights into how changes in task duration impact the optimal makespan. The computational results illustrate the effectiveness of the LP formulation in optimizing real-world project schedules.

Key words. linear programming, project scheduling, precedence constraints, time-cost trade-off

1. Introduction. Effective project scheduling is essential for optimizing time management, resource allocation, and overall project efficiency in fields such as construction, manufacturing, software development, and supply chain management. The Project Scheduling Problem (PSP) aims to determine the optimal start times of tasks in a project while ensuring that all precedence constraints are satisfied and the total project duration is well minimized.

This project develops a linear programming (LP) formulation for the precedence-constrained PSP, where:

- tasks have fixed durations and must be completed once started,
- task execution is subject to precedence constraints: A task cannot begin until all its prerequisite tasks are completed, and
- simultaneous execution is allowed as long as precedence constraints are met.

We present a specific example with a standard software engineering project workflow, detailing common tasks, their typical durations, and their order of precedence. We propose an optimized schedule using our LP formulation for general PSP's.

Another aspect of project scheduling is the trade-off between time spent and cost. In many cases, project managers may accelerate certain tasks by allocating additional resources, a concept known as the time-cost trade-off problem (TCTP). However, crashing tasks come at an additional cost, making it crucial to find an optimal balance between minimizing the makespan and controlling necessary costs. We then extend this LP formulation to generalize the TCTP, allowing tasks to be crashed at an additional cost per day, and introduce a budget constraint to limit the total budget on task acceleration. The LP model determines an optimal schedule that minimizes the total project duration while ensuring that acceleration costs remain within the allocated budget. In addition, we explore the dual problem, which provides insight into how each scheduling constraints affect the variability of the solution.

The remainder of this project is structured as:

- presenting the LP formulation for PSP and its mathematical representation in standard and matrix form.
- providing a computational example, solving the LP model using real-world project data.
- discussing potential insights through duality analysis on how significant each task duration is to the makespan
- extending the model to the time-cost trade-off problem (TCTP) and introduces a cost-constrained formulation.

This research reveals the effectiveness of linear programming in project scheduling and provides a practical optimization framework for minimizing project duration while balancing costs.

2. General Project Scheduling Problem. We use an LP formulation based on start time to minimize the completion time of a project consisting of n tasks. The goal is to optimize the project's *makespan* (total project duration) while adhering to precedence constraints.

Notation.

- S_i is the start time of task i (decision variable).
- d_i is the duration of task i (constant, $d_i > 0$).
- $C_{\max}$ is the project makespan (decision variable).

Objective Function. The makespan is equivalent to the completion time of the final task. We assume that once a task begins, it runs continuously until completion. Therefore, the completion time of any task i is given by $S_i + d_i$, where S_i is the start time and d_i is the duration of task i.

In general, we do not assume that the project has a single final task. Instead, we use a flexible model that can allow an optimal solution to schedule multiple final tasks in parallel when those tasks are not prerequisites for any other task. Thus, the makespan is determined by the latest completion time among all tasks:

$$C_{max} = \max(S_1 + d_1, S_2 + d_2, \ldots, S_n + d_n).$$

To linearize this, we introduce constraints that bound C_{max} by all task completion times:

$$C_{\max} \geq S_i + d_i, \quad \forall i \in \{1, \ldots, n\}.$$

This is equivalent to defining C_{max} as the start time of a final dummy task (with zero duration) that depends on all other tasks. Our objective function is simplified to minimizing C_{max}.

Precedence Constraints. Let P be the set of task dependencies, where each ordered pair $(i, j) \in P$ indicates that task j can start only after task i completes. To simplify the formulation and prevent cyclic dependencies, we assume task indices follow a topological order, ensuring $i < j$ for all $(i, j) \in P$. These precedence constraints are expressed as

$$S_j \geq S_i + d_i, \quad \forall (i, j) \in P.$$

Non-Negativity Constraints. Start times must be non-negative values, so

$$S_i \geq 0, \quad \forall i \in \{1, \ldots, n\}.$$

2.1. LP in Standard Form. We convert the original minimization problem to an equivalent maximization problem in standard form. That is, we want to find the solution to

$$\begin{aligned} &\text{Maximize} && -C_{\max} \\ &\text{subject to} \\ &&& S_i - C_{\max} \leq -d_i, && \forall i \in \{1, \ldots, n\} && (2.1) \\ &&& S_i - S_j \leq -d_i, && \forall (i, j) \in P && (2.2) \\ &&& S_i \geq 0, && \forall i \in \{1, \ldots, n\}. \end{aligned}$$

2.2. Matrix Form. In matrix form, the LP can be written as

$$\begin{aligned} \text{Maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \boldsymbol{A}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq 0. \end{aligned}$$

where the decision variable vector is

$$\mathbf{x} = (S_1, S_2, \ldots, S_n, C_{\max})^T \in \mathbb{R}^{n+1},$$

and the objective coefficient vector is

$$\mathbf{c} = (0, \ldots, 0, -1)^T,$$

which has zeros everywhere except at the $(n+1)^{th}$ position so that $\mathbf{c}^T\mathbf{x} = -C_{\max}$.

Let $m = |P|$ be the number of precedence constraints. Then $\boldsymbol{A}$ is a $(n+m)\times(n+1)$ sparse matrix nonzero entries equal to 1 or -1. The first n rows correspond to the finish time constraints in (2.1) while the the next m rows correspond to the general precedence constraints (2.2). We use subscripts to refer to the upper and lower submatrices as $\boldsymbol{A}_{1:n}$ and $\boldsymbol{A}_{n+1:n+m}$. We have

$$\boldsymbol{A}_{1:n} = \begin{bmatrix} \boldsymbol{I}_n & -\mathbf{1}_n \end{bmatrix}.$$

Recall that $\boldsymbol{I}_n$ is the identity matrix and $\mathbf{1}_n$ is a column vector of ones. The righthand side vector $\mathbf{b}$ has the corresponding first n values

$$\mathbf{b}_{1:n} = (-d_1, -d_2, \ldots, -d_n)^T.$$

The task precedence constraints can be encoded by an incidence matrix. Let $\mathbf{Q}_P \in \{-1, 0, 1\}^{m\times n}$ be the incidence matrix on the set P. Suppose that the precedence constraints are indexed so that the kth constraint corresponds to the pair $(i_k, j_k) \in P$. Then we define $\mathbf{Q}_P$ by

$$(\mathbf{Q}_P)_{k,\ell} = \begin{cases} 1, & \text{if } \ell = i_k, \\ -1, & \text{if } \ell = j_k, \\ 0, & \text{otherwise.} \end{cases}$$

Thus, each row of $\mathbf{Q}_P$ has a 1 in the column corresponding to the prerequisite task and a -1 in the column corresponding to the dependent task. In our overall constraint matrix, these rows are appended as

$$\mathbf{A}_{n+1:n+m} = \begin{bmatrix} \mathbf{Q}_P & \mathbf{0}_m \end{bmatrix},$$

where $\mathbf{0}_m$ is an $m \times 1$ zero vector (since $C_{\max}$ does not appear in these constraints). The corresponding entries in $\mathbf{b}$ are

$$\mathbf{b}_{n+1:n+m} = (-d_{i_1}, -d_{i_2}, \ldots, -d_{i_m})^T.$$

2.2.1. Remark on Transitive Reduction and the Structure of the Precedence Constraint Matrix. The number of precedence constraints, m, is upper bounded by the number of edges in a complete graph on n vertices:

$$m \leq \frac{n(n-1)}{2} \propto n^2.$$

However, in a directed acyclic graph many of these edges represent transitive relationships and are therefore redundant in the scheduling problem. For example, in the set $\{(1,2),(2,3),(1,3)\}$ the constraint $(1,3)$ is redundant since it is implied by $(1,2)$ and $(2,3)$.

Although a full treatment of transitive reduction is beyond the scope of this paper, it is worth noting that applying transitive reduction to a task precedence graph (see, e.g., Figure 1) eliminates redundant constraints and yields a more economical representation of the task relationships [1]. In many practical scheduling problems, where task dependencies are nearly sequential, this process reduces the number of constraints to $m \propto n$. In particular, if the given set of task dependencies is minimal (i.e., it represents the transitive reduction of the complete dependency graph), then each column of the corresponding incidence matrix $\mathbf{Q}_P$ contains only the essential nonzero entries. Moreover, by an appropriate permutation of the rows, these nonzero entries can be arranged to lie near the main diagonal. Finally, since our problem formulation enforces $i < j$ for all $(i,j) \in P$, it follows that the resulting precedence constraint matrix is nearly upper triangular.

2.3. Example Software Engineering PSP. We provide a simple example of the general project scheduling problem (PSP) that commonly arises in the software engineering domain. Suppose that a project manager wants to deliver a project in the shortest possible time. The project consists of an initial task for defining requirements, followed by implementation and extensive testing. The project tasks and their associated durations and dependencies are shown in Table 1.

Task	**Description**	**Prerequisites**	**Duration (days)**
1	Define project specifications	-	10
2	Implement backend architecture	1	17
3	Implement frontend architecture	1	10
4	Implement backend APIs	2	21
5	Implement UI and integrate APIs	3, 4	28
6	Unit testing	4, 5	10
7	Integration testing	6	14
8	Security and performance testing	7	21
9	Write technical and user documentation	5	7
10	Deploy and monitor	8	5

TABLE 1
A software engineering project might involve these ten tasks.

For this small example of a project with $n = 10$ tasks, we can easily visualize the precedence constraints on a graph and find an optimal schedule by inspection. In an optimal schedule, tasks that do not depend on each other should be completed in parallel. We follow our LP problem formulation and express the task dependencies as

$$P = \{(1,2),(1,3),(2,4),(3,4),(4,5),(4,6),(5,6),(6,7),(7,8),(5,9),(8,10)\}$$

and the incident matrix which is a submatrix of our constraint matrix,

$$Q_P = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}.$$

2.3.1. Precedence Graph. The *precedence graph* is a weighted, acyclic, directed graph $G = (T, P)$, where the vertices represent tasks, and each directed edge (i, j) indicates that task i must be completed before task j begins. The edge is weighted by the duration d_i required for task i.

If a valid precedence graph cannot be constructed for the proposed project, that is, if the graph contains a cycle, then there is no valid schedule that satisfies the precedence constraints. A simple example of this is the "chicken and egg" problem, where two tasks depend on each other, creating a circular dependency that cannot be resolved. For inherently iterative projects, such as software engineering projects with repeated testing, cycles can be broken by introducing new tasks for each iteration. This ensures that the precedence graph is acyclic and that the PSP is feasible.

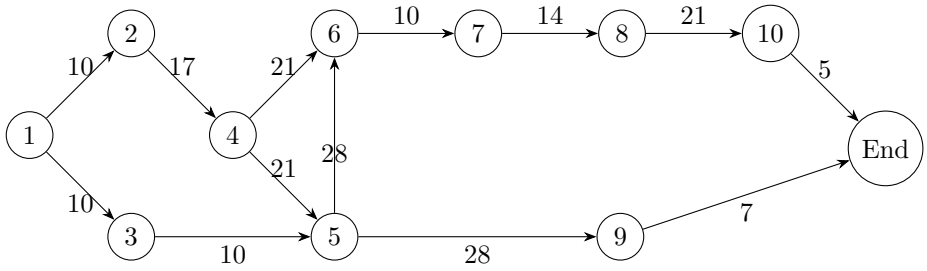

FIG. 1. *The precedence graph helps visualize an example of a software engineering PSP. There is only one possible starting task. There are two possible final tasks, so we introduce a dummy node to indicate project completion. We can see that some tasks can be scheduled simultaneously, while others must be completed in sequence.*

We set up and solve the example software engineering problem in subsection 2.3 using the `scipy.optimize.linprog()` function. Let $\mathbf{x}^*$ denote the optimal solution to our problem in matrix form as described in subsection 2.2. Our solver (see supplementary notebook) finds the optimal start times x_j^* and finish times $x_j^* + d_j$ of each task t_j, as well as the minimal makespan $-\mathbf{c}^T\mathbf{x}^*$. We summarize the optimal solution in Figure 2.

2.3.2. Dual Problem. Given an LP problem in the matrix form described in subsection 2.2, consider the corresponding dual problem

```
Optimal Schedule:

Task  Start  Finish
  t1    0.0    10.0
  t2   10.0    27.0
  t3   10.0    20.0
  t4   27.0    48.0
  t5   48.0    76.0
  t6   76.0    86.0
  t7   86.0   100.0
  t8  100.0   121.0
  t9   76.0    83.0
 t10  121.0   126.0

Project makespan (C_max): 126.00
```

FIG. 2. *The output of our solver shows the optimal project schedule with the minimum overall completion time.*

$$\begin{aligned} \text{Minimize} \quad & \mathbf{b}^T\mathbf{y} \\ \text{subject to} \quad & \boldsymbol{A}^T\mathbf{y} \geq \mathbf{c}, \\ & \mathbf{y} \geq 0. \end{aligned}$$

where $\mathbf{y}$ is the decision variable vector of the dual problem. In the context of our general project scheduling problem, this would be represented as

$$\mathbf{y} = (y_1, y_2, \ldots, y_{n+m-1}, y_{n+m})^T \in \mathbb{R}^{n+m},$$

where we recall n is the number of constraints that bound C_{max} with task completion time, and m is the number of task precendence constraints. Therefore there is one dual variable for each of the $n+m$ constraints in the primal problem (excluding non-negativity constraints). The objective of the dual would then be to minimize the function $\mathbf{b}^T\mathbf{y}$ which takes the form

$$-\sum_{k=1}^{n} d_k y_k - \sum_{l=1}^{m} d_{i_m} y_{n+l}$$

It follows that the constraint matrix $\mathbf{A}^T$ is an $(n+1) \times (n+m)$ matrix where

$$\mathbf{A}^T_{1:n} = \begin{bmatrix} \mathbf{I}_n & \mathbf{Q}_P^T \end{bmatrix}, \quad \mathbf{A}^T_{n+1} = \begin{bmatrix} -\mathbf{1}_n^T & \mathbf{0}_m^T \end{bmatrix}$$

Therefore there is a dual constraint for each of the $n+1$ primal variables, the first n correponding to each primal dual variable $x_1, x_2, ...x_n$ and the $n+1^{th}$ corresponding to c_{max} The first n dual constraints follow the form:

$$y_k + \sum_{l=k}^{n+m} y_i I(k,i) \geq 0, \forall k \in \{1,2,\ldots,n\}$$

$$\text{where } I(x,y) = \begin{cases} 1, & \text{if } (x_y, j_y) \in P \\ -1, & \text{if } (j_y, x_y) \in P \\ 0, & \text{otherwise} \end{cases}$$

and j is any task other than x

And the $n+1^{th}$ dual constraint is:

$$\sum_{k=1}^{n} -y_k \geq -1 \implies \sum_{k=1}^{n} y_k \leq 1$$

which, together with the non-negativity constraints ensures that the 1^{st} to the n^{th} dual variables must be in the range [0,1]. Let $\mathbf{y}^*$ denote the optimal solution to the dual problem. Assuming the primal scheduling problem is a non-trivial problem where $C_{max} > 0$ at optimality, then by complementary slackness we know

$$\sum_{k=1}^{n} y_k^* = 1$$

And for all $k \in \{1,2,....,n\}$, $l \in \{1,2,....,m\}$

$$y_k^* \neq 0 \implies C_{max} = S_k + d_k$$
$$\text{and } y_{n+l}^* \neq 0 \implies S_{j_l} = S_{i_l} + d_{i_l}$$

That is,

- the first n dual variables sum to 1 at optimality and
- any k^{th} dual decision variable among the first n which is non-zero at the optimum would act as an indicator that the k^{th} task is a final task in the optimal schedule.
- any l^{th} dual decision variable among the remaining m that are non-zero at the optimum would act as an indicator that for the tasks involved in the l^{th} precedence constraint, $(i_l, j_l) \in P$, the subsequent task begins immediately after the prerequisite task in the optimal schedule

Let us consider the "shadow price" interpretation of the dual variable. Optimal value of dual variable y_k^* represents the rate of increase in the primal optimal objective value with respect to increase in the right hand side of the corresponding k^{th} primal constraint, in standard form. In the standard form of the primal, the RHS of the constraints, $\mathbf{b}$, consists of negated task durations. Similarly, since the primal problem is framed in the form of a maximization problem the objective value at optimality, $\mathbf{c}^T\mathbf{x}^*$, would be the negated value of the optimal timespan. Therefore, each optimal dual variable value can be interpreted as the decrease in optimal project duration upon decreasing the duration of the preceding task in the associated primal constraint by one unit.

We reason that at optimality all dual variables $y_i^* \in \{0, 1\}$. We omit a formal proof of this statement, but by intuition: either a corresponding constraint is binding and decreasing the associated task duration directly decreases the overall makespan, or the corresponding constraint is not binding and decreasing the associated task duration will not decrease the overall makespan. Since the first n dual variables sum to 1, there must be a single $y_k^* = 1, k \in \{1, 2, ...n\}$, which corresponds to the makespan constraint associated with the final task. It follows that any $y_{n+l} = 1, l \in \{1, 2, ...m\}$ would correspond to the precedence constraint associated with an immediately preceding prerequisite task. We signify this sequence of continuous prerequisite tasks as the critical path. This would be represented as the longest path from the starting node to the end node in the precedence graph. Thus, the dual variables at optimality indicate critical tasks, which for the purpose of project streamlining and task cutting, is valuable. We will later discuss the importance of the critical path involved in the TCTP problem.

3. Discussion on PSP Variations. We summarize some variations in project scheduling problems [4] below.

Temporal Constraints. Temporal constraints extend the basic precedence relationships and include:

- **Task or milestone deadlines:** Fixed dates by which certain tasks or milestones must be completed.
- **Time windows:** Restrictions that limit the allowable start and finish times for tasks.
- **Lag or rest times:** Minimum time intervals required between the completion of one task and the start of another.

For problems defined over a continuous time domain, these constraints can often be modeled using linear programming. However, time window constraints typically lead to event-based formulations [7] that require binary variables.

Resource Constraints. In many projects, tasks compete for limited resources. These resource restrictions can take various forms, such as constraining the number of tasks that can be executed simultaneously due to the availability of personnel, equipment, or budget. Resources can be renewable or non-renewable and can take on continuous or discrete values, or a mix of both. Models incorporating such constraints are typically more complex and are classified as resource-constrained project scheduling problems (RCPSP). These models often use mixed-integer programming for optimization [2]. We mention this concept only briefly because it is beyond the scope of this paper.

Objectives. Scheduling objectives vary depending on the context, but generally focus on minimizing some measure of cost. Common objectives include:

- **Project makespan:** Reducing the total time required to complete all tasks, often combined with other goals.
- **Tardiness:** Reducing the difference in task start times and their release times. This models the preference for beginning tasks as early as possible.
- **Lateness:** Reducing the penalty associated with completing tasks after their deadline. This models the preference for completing tasks as early as possible, assuming that there is no penalty for completing tasks too early.

In many real-world scenarios, these objectives are combined into a weighted multi-objective formulation, where each weight sets the relative importance of the corresponding objective. (3.2) is an example of this.

3.1. Time-Cost Trade-Off Problem. Consider a project in which certain tasks can be accelerated at an additional cost. For example, in the software engineering project example in subsection 2.3, suppose that the project manager hires a freelancer to help with tasks. In particular, assume that implementing the UI and integrating APIs can be shortened from 28 days to a minimum duration of 14 days. This process of reducing task duration from its normal time to its minimum (crashed) time is known as *crashing* or *project crashing* and can be approached using different techniques [5, 6, 3]. However, since the freelancer is paid $500 per day and the project has a fixed budget, the challenge lies in determining the optimal schedule. The goal is to minimize the project makespan while staying within budget. This inverse relationship between time and cost is known as the time-cost trade-off problem (TCTP), as reducing task durations typically incurs higher costs.

We can formulate an LP model for the continuous TCTP by modifying that for the general precedence-constrained PSP presented in section 2. In addition to finding optimal task start times, we introduce decision variables to represent the time reduction for each task. We define constants for the normal and crashed durations of each task, as well as the cost per unit of time to accelerate them (crashing cost) and a total budget.

Definition and Notation.

- S_i is the start time of task i (decision variable).
- d_i^n is the normal duration of task i (constant, $d_i^n > 0$).
- d_i^c is the crashed duration of task i (constant, $d_i^c > 0$).
- c_i is the crashing cost per unit time of task i (constant, $c_i \geq 0$),
- x_i is the time reduction (decision variable),
- $C_{\max}$ is the project makespan (decision variable),
- B is the total budget for the project (constant, $B \geq 0$).

The actual duration of task i is given by

$$d_i = d_i^n - x_i. \tag{3.1}$$

A flexible objective is to minimize a weighted sum of the project makespan and the total crashing cost:

$$\alpha C_{\max} + \sum_{i=1}^{n} c_i \, x_i, \tag{3.2}$$

where α is a positive weighting constant that balances the trade-off. A higher value α places more importance on optimizing the makespan.

Similarly to subsection 2.1, we formulate the linear program in standard form:

$$\text{Maximize} \quad -\alpha C_{\max} - \left(\sum_{i=1}^{n} c_i \, x_i \right)$$

subject to

$$S_i - C_{\max} - x_i \leq -d_i^n, \qquad \forall i \in \{1, \ldots, n\} \tag{3.3}$$

$$S_i - S_j - x_i \leq -d_i^n, \qquad \forall (i,j) \in P \tag{3.4}$$

$$x_i \leq d_i^n - d_i^c, \qquad \forall i \in \{1, \ldots, n\}, \tag{3.5}$$

$$\sum_{i=1}^{n} c_i \, x_i \leq B, \tag{3.6}$$

$$S_i \geq 0, \qquad \forall i \in \{1, \ldots, n\},$$

$$x_i \geq 0, \qquad \forall i \in \{1, \ldots, n\}.$$

Constraints (3.3) and (3.4) are identical to the constraints (2.1) and (2.2), respectively. They enforce the precedence relationships defined in P and with the final dummy task. The main differences are that we substitute d_i with its equivalent expression from (3.1) in terms of the time reduction decision variables and add the new constraints in (3.5) so that the actual task durations remain within the range defined by their normal and crashed durations. Constraint (3.6) enforces a hard limit on the budget, but depending on the value of α, this constraint may not be binding.

Task	**Normal duration (days)**	**Crash duration (days)**	**Cost per day (CAD)**
1	10	9	150
2	17	11	490
3	10	6	300
4	21	15	530
5	28	14	500
6	10	7	260
7	14	10	320
8	21	16	750
9	7	4	100
10	5	3	250

TABLE 2

Tasks in the software engineering project example in Table 1 can now be accelerated at an additional cost. A higher project budget will allow for more flexibility in reducing task durations, thereby helping to minimize the total project completion time.

4. Conclusions. We have detailed a widely studied area in operations research, project scheduling problem, initially based on three standard conditions:

- a task cannot be commenced until its prerequisites are completed
- tasks take a fixed duration to complete, and cannot be interrupted or progress in an interleaved manner
- multiple tasks can progress simultaneously as long as prerequisite constraints are complete

with the goal being to minimize the total completion time of tasks. We generate a list of tasks, durations, and precedences involved a software engineering project. Then

we formulate the project as an LP problem in our solver and determine optimal start times of each task.

We demonstrate that the solutions to the dual problem provides valuable information on constraints and tasks in the primal scheduling problem, such as which task durations are lengthening the overall completion time and to what degree. We also consider substantiating the scheduling problem to a time-cost trade-off problem, where we add nuances:

- the duration of a task can be shortened for a cost
- budget for task shortening is limited

for which the goal is to minimize a weighted sum of the makespan and amount spent. We generalize the TCTP as an LP problem and generate another software engineer project task list now containing crashed durations and crashing costs. It is expected that only crashing tasks in the critical path, the longest path in the precedence graph, can achieve a reduction in the makespan. These tasks correspond to the binding constraints. We leave the solving of this modified problem for future discussion.

REFERENCES

[1] A. V. Aho, M. R. Garey, and J. D. Ullman, *The transitive reduction of a directed graph*, SIAM Journal on Computing, 1 (1972), pp. 131–137.

[2] C. Artigues, S. Demassey, E. Neron, and W. O. Library, *Resource-constrained project scheduling: models, algorithms, extensions and applications*, John Wiley Sons, Hoboken, NJ, 1st;1; ed., 2008;2010;2013;.

[3] O. Elmabrouk and F. Aljiebali, *Crashing project activities using linear programming technique*, in Proceedings of the International Conference on Industrial Engineering and Operations Management. Turkey, 2012.

[4] S. Hartmann and D. Briskorn, *An updated survey of variants and extensions of the resource-constrained project scheduling problem*, European Journal of Operational Research, 297 (2022), pp. 1–14, https://doi.org/https://doi.org/10.1016/j.ejor.2021.05.004, https://www.sciencedirect.com/science/article/pii/S0377221721003982.

[5] N. S. Islam, *Complex project crashing algorithm*, IOSR Journal of Business and Management (IOSR-JBM), 11 (2013), pp. 10–17.

[6] A. Katti and M. Darade, *Project crashing to solve time-cost trade-off*, International Journal of Civil Engineering, 3 (2016), pp. 10–27.

[7] O. Koné, C. Artigues, P. Lopez, and M. Mongeau, *Event-based milp models for resource-constrained project scheduling problems*, Computers & Operations Research, 38 (2011), pp. 3–13.