RESOURCE-CONSTRAINED PROJECT SCHEDULING WITH AUTONOMOUS LEARNING EFFECTS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Industrial Engineering

by

Jordan Ticktin

December 2019

COMMITTEE MEMBERSHIP

TITLE:	Resource-Constrained Project Scheduling with Autonomous Learning Effects
AUTHOR:	Jordan Ticktin
DATE SUBMITTED:	December 2019
COMMITTEE CHAIR:	Alessandro Hill, Ph.D. Assistant Professor of Industrial Engineering
COMMITTEE MEMBER:	Tali Freed, Ph.D. Professor of Industrial Engineering
COMMITTEE MEMBER:	Mohamed Awwad, Ph.D. Assistant Professor of Industrial Engineering
COMMITTEE MEMBER:	Eric Olsen, Ph.D. Professor of Industrial Technology and Packaging

ABSTRACT

Resource-Constrained Project Scheduling with Autonomous Learning Effects

Jordan Ticktin

It's commonly assumed that experience leads to efficiency, yet this is largely unaccounted for in resource-constrained project scheduling. This thesis considers the idea that learning effects could allow selected activities to be completed within reduced time, if they're scheduled after activities where workers learn relevant skills. This paper computationally explores the effect of this autonomous, intra-project learning on optimal makespan and problem difficulty. A learning extension is proposed to the standard RCPSP scheduling problem. Multiple parameters are considered, including project size, learning frequency, and learning intensity. A test instance generator is developed to adapt the popular PSPLIB library of scheduling problems to this model. Four different Constraint Programming model formulations are developed to efficiently solve the model. Bounding techniques are proposed for tightening optimality gaps, including four lower bounding model relaxations, an upper bounding model relaxation, and a Destructive Lower Bounding method. Hundreds of thousands of scenarios are tested to empirically determine the most efficient solution approaches and the impact of learning on project schedules. Potential makespan reduction as high as 50% is discovered, with the learning effects resembling a learning curve with a point of diminishing returns. A combination of bounding techniques is proven to produce significantly tighter optimality gaps.

Keywords: Operations research, Constraint programming, IBM ILOG CP Optimizer, Resource-constrained project scheduling problem, RCPSP, Lower and upper bounding

ACKNOWLEDGMENTS

This thesis took a little over a year to complete, and lots of people helped me along the way. I'd especially like to thank the following:

Dr. Alessandro Hill, for being an outstanding advisor and always being generous with his time. I couldn't have done this without his knowledge, passion, advice, encouragement, and delightfully dry sense of humor.

My family, for their love, support, and encouragement.

The Graduate Education Office for awarding me the Graduate Presentation Award, and the Industrial and Manufacturing Engineering department's Student Fee Committee for providing supplemental travel funds. These awards enabled me to give a talk presenting this research in Seattle, WA at the 2019 INFORMS Annual Meeting.

Dr. Tali Freed, Dr. Mohamed Awwad, and Dr. Eric Olsen for participating on my thesis committee.

Dr. Qian Hu and Ying Liu from Nanjing University in China, for sparking the research idea and helping find bugs in my code.

Dr. Thomas Vossen from University of Colorado Boulder, for his advice on how to flesh out the research and prepare it for publication.

Michael Williams, for his help improving the presentation of this material.

Kate Parkinson and the rest of the Cal Poly CubeSat Lab ATLO team, for being understanding every time I had to temporarily step back from my responsibilities because I unearthed yet another problem to solve in some part of my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

INTRODUCTION

This chapter introduces the thesis and the research conducted within. It first describes the motivation for conducting the research, along with problem and research questions posed. Next, it details contributions made to the literature. It ends with an explanation of this paper's organizational structure.

1.1 Motivation and Research Questions

Large projects in all industries face the problem of how best to schedule many complex components. This is commonly formulated as the Resource-Constrained Project Scheduling Problem (RCPSP) (see Section 3.1): given resource and precedence constraints, what schedule can complete the project the fastest? Historically, these problems were solved by hand, but today we can use advanced software packages to solve them more quickly. Much literature exists about efficiently solving these types of scheduling problems and their many variations (see Chapter 2). Many of these extensions of the base RCPSP give the solver more alternatives; this can match the problem more closely to the real world, and the additional options often enable the solver to find shorter optimal schedules (albeit with more difficulty in solving).

One facet of the real world that is often not accounted for in scheduling problems is learning. Suppose a construction company is building an apartment building. Because many of the apartments will have designs that are similar, or even identical, a task like laying hardwood floor may be performed multiple time throughout the project (Figure 1.1). While building the first floor of apartments, the team will run into

Figure 1.1: Repeated Tasks Within a Project

problems while laying the hardwood floor: parts won't fit right, workers will discover issues with the design, etc. By the time the first floor is complete, and it's time to build the second floor, the workers will have worked out many of the kinks. They will be able to do it better, faster, and possibly even cheaper.

This concept of learning effects motivates the central problem of this thesis: How does learning affect the way projects are scheduled? The corollary research questions to this are as follows:

- How would learning be incorporated into scheduling problems?
- What would the effects of learning be?
- How hard would these problems be to solve?
- How could they be solved more efficiently?

1.2 Contributions

In seeking to answer the central problem and corollary research questions, this paper contributes the following to the literature:

- Definition of RCPSP+L, an RCPSP-based learning model (Section 3.2). Formulation of this model using Constraint Programming (Chapter 5).
- Development of an RCPSP+L instance generator (Section 7.2) and reduction techniques for the instance set (Chapter 4). Extensive computational analysis through hundreds of thousands of experiments, identifying significant makespan reduction potential (Section 8.2).
- Examination of the impact of various parameters on problem difficulty (Section 8.4).
- Exploration of more efficient ways to solve these problems, including multiple Constraint Programming formulations (Section 8.1) and several bounding techniques (Section 8.3), which solve instances to optimality more quickly and tighten optimality gaps.

1.3 Paper Structure

These contributions are dispersed throughout the paper. After this introduction, literature relevant to this research is reviewed in Chapter 2. Chapter 3 explains the standard RCPSP model and introduces the new learning extension. Reduction techniques are described in Chapter 4. Chapter 5 proposes multiple Constraint Programming formulations for solving the instances, and Chapter 6 specifies some model relaxations and a destructive lower bounding technique that can all be used to reduce optimality gaps. Chapter 7 describes the data source, custom test instance generator, and system/software configuration used to set up the experiments. These experiments are performed and results are analyzed in Chapter 8. Finally, Chapter 9 summarizes the research and offers some ways to continue it in the future.

Chapter 2

LITERATURE REVIEW

Extensive research on related topics exists that this paper can build upon. This chapter reviews literature relevant to this research, including literature related to scheduling, learning, Constraint Programming, and intersection points between the three. Note that more works from the literature are referenced in other parts of the paper, in addition to or instead of being included in this chapter, wherever the subject is most relevant.

2.1 Project Scheduling

Project scheduling is a problem with a long history in the literature for operations research and management science. In 1959, Kelley and Walker introduced the critical path method to help planners manually craft efficient schedules [26]. Today, we can solve difficult scheduling problems with advanced computer algorithms instead. For example, NASA uses custom software packages to do activity planning for the Mars rovers [6], and Roscosmos (the Russian space agency) uses mathematical models to schedule training for cosmonauts heading to the International Space Station [36].

Scheduling problems come in all shapes and sizes. They vary in constraints, options, complexity, and more. By far one of the most common scheduling problems in the literature is the Resource-Constrained Project Scheduling Problem (RCPSP). (As of November 2019, a Google Scholar search for “resource constrained project scheduling” returns 438,000 results.) This standard scheduling problem covers a wide variety of abstract and real world scheduling problems, all while being straightforward to work with. RCPSP has been applied to problems in civil engineering and construction management [27], waterway traffic management [21], mining [20], research labs [43], and much more. It has even been used for project management in the Colombian video game and animation industry [40].

One of RCPSP's main benefits is its adaptability. Many extensions to the base RCPSP have been explored in the literature, attempting to improve its applicability to "real world" problems and reduce the optimal makespan for a given instance. Hartmann and Briskorn conduct an extensive survey of RCPSP variants in [18], covering generalizations of the problem with respect to activity concepts, temporal constraints, and resource constraints, and describing alternative objectives and problems involving multiple projects. One well-known generalization of the RCPSP in the literature is the Multi-Mode RCPSP (often shortened to MM-RCPSP), which is shown to be computationally efficient [35].

2.2 Learning Curves

Human learning has been discussed scientifically as far back as 1885 [12]. In 1936, Wright documented the benefits of learning for production [53]. He described an "eighty percent curve", wherein doubling the number of planes produced only increased labor costs by 80%, which he attributed to a more skilled workforce and economies of scale. In the 1950s, semiconductor manufacturers were able to dramatically reduce costs by taking the effect into account [11], and in more recent years the price of photovoltaics (i.e. solar cells) have plunged for similar reasons [41]. The effect has also been applied outside of industrial settings: Hanakawa et al. demonstrated that not only does a software developer's productivity increase the longer the developer stays on a task, the efficiency gain can be practically modeled using a learning curve [17]; Wu and Sun utilized the learning curve effect when exploring how to minimize outsourcing costs during research and development [54]. The learning curve has been a popular research subject since its introduction, with Yelle documenting its popularity back in 1979 [55], Glock et al. reaffirming its continued popularity in 2019 [15], and multiple concurring surveys made in the years between [23, 24].

2.3 Learning Effects in Scheduling

Given how popular studying the learning effect is in project management, it should be no surprise that this popularity extends into the scheduling literature. One of the earliest cases of this is by Keachie and Fontana in 1966 [25], with an early scheduling formulation that includes learning considerations in 1999 [3].

Biskup conducts an extensive survey of learning in scheduling in [4]. He identifies two main learning methods in the scheduling literature: autonomous learning ("learn by doing") and induced learning ("learn by review"). Autonomous refers to learning that occurs automatically when workers gain experience from performing an activity themselves and "learn the hard way". Induced refers to learning that is intentionally mined from others' experiences, such as when listening to a lecture in a classroom or reviewing a video, and takes additional time.

Surprisingly, though both learning effects and the RCPSP are extensively researched in the literature, not much literature exists that integrates the two. One of the closest examples is a 2015 paper by Van Peteghem and Vanhoucke titled "Influence of learning in resource-constrained project scheduling" [50]. They incorporate learning effects into the discrete time/resource trade-off problem (DTRTP), a subproblem of the MM-RCPSP, where jobs are defined by work content instead of fixed duration/resource requirements. They only examine one resource in their experiments: workers.

Different modes are then defined with varying combinations of duration/resources to satisfy the work required. They find that learning has a significant impact on project schedule, so much so that baseline schedules which ignore this effect aren't accurate predictors of real project progress; 88% of the instances they examine which ignore learning deviate by $> 10\%$ from the actual observed makespan.

This thesis builds upon Van Peteghem and Vanhoucke's work through the following: Where they modeled the problem as an extension of the DTRTP, this thesis uses a model that generalizes the RCPSP so that the work is easily integrated with other RCPSP research and variants. Four resources are considered in the instances in this thesis, to better understand the effects resource constraints would have on a realistic scenario, instead of assuming workers are the only resource that constrains the instance. This thesis also utilizes Constraint Programming, including multiple model formulations for more efficient solving (see Chapter 5), instead of the Mathematical Programming formulation they use. Additionally, this thesis explores techniques for tightening instance optimality gaps (see Chapter 6).

2.4 Constraint Programming

There are many different solution approaches for scheduling problems, such as Mathematical Programming (MP) [45], Constraint Programming (CP) [46], and heuristics [1]. MP is frequently used for scheduling problems; indeed, the paper by Van Peteghem and Vanhoucke mentioned in the previous section uses MP. Possible uses for MP and heuristics with the research in this thesis are discussed in Section 9.2.2.

CP was selected for this thesis due to its exceptional track record with this kind of problem: scheduling has been one of the most successful uses for CP for over 20 years [34]. Furthermore, CP has specifically been shown to excel at solving RCPSP instances [38]. Beyond just project scheduling problems, CP is now used extensively (and highly successfully) for a broad array of problems [52]: sports tournament scheduling [8]; just-in-time cross-docking planning in supply chains [13]; inventory management for natural gas, where CP is shown to be 4–10 times faster than Mixed Integer Programming (a type of MP) [16]; the stable marriage [14], stable roommate [44], and hospitals/residents matching [39] problems; and more.

As mentioned in the previous section, one of the topics explored in this thesis is how to tighten optimality gaps for instances the solver can't easily solve to optimality. Bounding techniques are commonly applied to improve the upper and lower bounds automatically produced by the solver. One frequent approach to producing improved bounds is to use model relaxations [42], and another method is destructive lower bounding [7, 28]. Both are used in this thesis (see Chapter 6).

Chapter 3

MODEL DEFINITIONS

This chapter defines the standard RCPSP frequently used in the literature for scheduling problems, followed by an introduction to the RCPSP+L learning model extension central to this thesis. A common example instance is used throughout the chapter to demonstrate some concepts and compare the two models.

3.1 The Resource-Constrained Project Scheduling Problem (RCPSP)

The Resource-Constrained Project Scheduling Problem (RCPSP) is a standard scheduling problem in the literature. It is a strongly NP-hard optimization problem [5]. A Constraint Programming formulation of the RCPSP is described in Section 5.2. The RCPSP is defined as follows [30]:

JOBS A project of size n has a set of jobs (tasks) $J = j_0, j_1, \ldots, j_n, j_{n+1}$. Jobs j_0 and j_{n+1} are empty start/end tasks to mark the beginning/end of the project. As such, J has a total number of tasks $|J| = n+2$. For clarity, projects will usually be referred to by their size n in this paper, unless explicitly stated otherwise.

DURATIONS All jobs have duration $d \in \mathbb{Z}^+$, except for the empty start/end tasks (which have $d = 0$). The time periods start at 0 and increase by increments of 1.

RESOURCE CONSTRAINTS The project has a set of resource capacities $R = \{r_1, \ldots, r_k\}$, where $r \in \mathbb{Z}^+$ is a resource's per-period capacity, and k is the number of different types of resources in the instance. Resources are renewable:

the resource capacity is fully available every time period and is not permanently depleted over the course of the project. Each job has a fixed per-period resource consumption $u_r \in \mathbb{Z}_0^+ \; \forall\, r \in R$ during each time period in which it's scheduled, except for the empty start/end tasks (which have $u_r = 0 \; \forall\, r \in R$). To respect the resource capacities, the following must be true during each time period of the solution:

$$\sum_{i=1}^{n} u_{r,i} \leq r \; \forall\, r \in R \tag{3.1}$$

PRECEDENCE CONSTRAINTS The set of precedence relationships A can be visualized as an acyclic precedence network (example shown in Figure 3.1). In a direct precedence relationship (i, j), a task i must always be scheduled before a task j. In an indirect precedence relationship, if $(i, j) \in A$ and $(j, k) \in A$ are true, $(i, k) \in A$ must be true as well. The empty start/end jobs are such that $(j_0, j_x) \; \forall\, j_x \in \{j_1, \ldots, j_{n+1}\}$ and $(j_x, j_{n+1}) \; \forall\, j_x \in \{j_0, \ldots, j_n\}$. All jobs, except j_0, must have at least one predecessor.

OBJECTIVE The objective in RCPSP is to minimize the project makespan (total time from start to finish) $MS = t(j_{n+1})$.

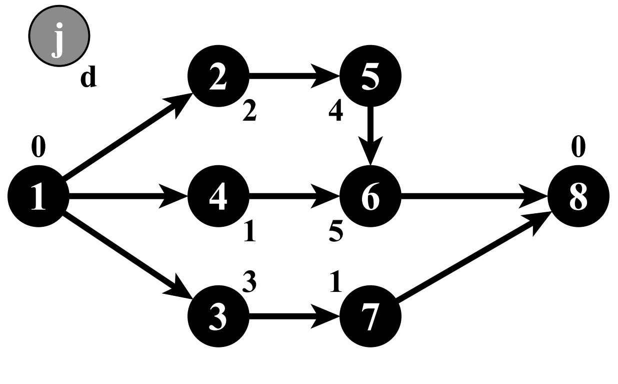

Figure 3.1: Example Precedence Network

A solution is said to be feasible if it's valid given all of the above constraints. Furthermore, a solution is optimal if it's proven that there are no feasible solutions with a shorter makespan. Solutions are frequently represented as a Gantt chart (as in Figure 3.3).

3.2 RCPSP with Autonomous Learning (RCPSP+L)

This thesis explores a generalization of RCPSP called RCPSP+L which incorporates the effects of autonomous intra-project learning. Autonomous refers to learning that takes place automatically as workers move from job to job (see Chapter 2), and intra-project means learning that occurs from job to job within the same project.

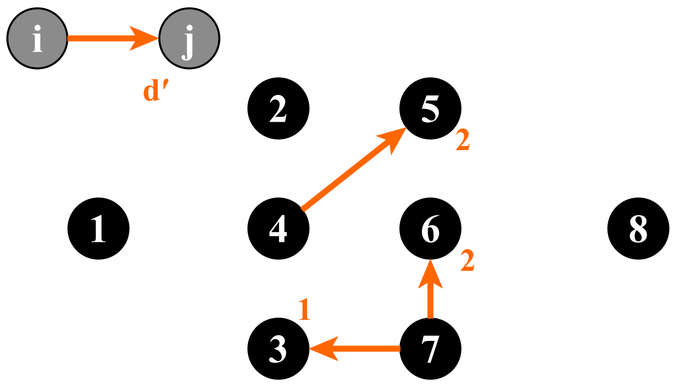

Figure 3.2: Example Learning Network

RCPSP+L reduces to RCPSP if no learning relations exist. (Thus, RCPSP+L is also NP-hard.) Everything from the previous section's description of RCPSP still holds true for RCPSP+L, with the following changes regarding learning relationships:

- The set of potential learning relationships L can be visualized as a learning network. An example is shown in Figure 3.2. (Note: The instance in this chapter's example would be reduced after the techniques in Chapter 4).
- In a learning relationship (i, j), i is referred to as the teacher task and j as the student. If $i \prec j$ in the schedule (i ends before or at the same time as

j starts), then the workers performing j can benefit from the experience they gained during i, and j can be completed more quickly.

- The normal non-learning duration d is now called a task's original duration. The shorter post-learning duration d' is referred to as a task's alternate duration, where $d' \in \mathbb{Z}^+$. It is always the case that $d > d'$.
- A student's per-period resource consumption u_r doesn't change post-learning, thus, the overall resource consumption for a task that learns is reduced.
- Each student task can only have one potential teacher designated in the instance, but a task can be a teacher for multiple students.

An example of the learning effect's benefits is presented in Figure 3.3, in which learning reduces the optimal makespan of the example project from 12 to 9.

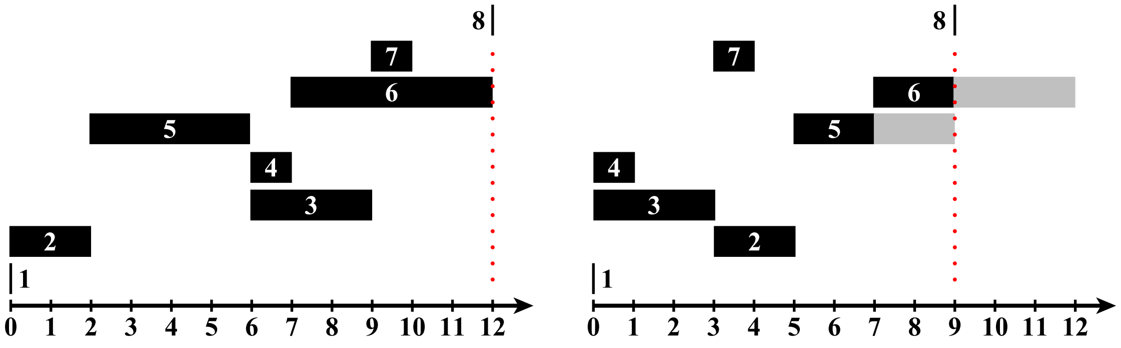

Figure 3.3: Comparison of Example RCPSP and RCPSP+L Solutions

Learning network generation for the test instances in this thesis relies on two parameters (see Section 7.2):

LEARNING FREQUENCY (ϕ) Target percentage of tasks with the potential to be students (i.e. present in the learning network).

LEARNING INTENSITY (λ) Percentage a student's original duration d is reduced by to calculate its alternate duration d'.

This concept for integrating learning effects is chosen because of its simplicity. While other learning models have the potential to be more accurate due to increased complexity, such as allowing multiple teachers or utilizing intricate similarity matrices, that complexity is also a major drawback. Identifying potential learning relationships alone is difficult and time-consuming enough when implementing this problem in the real world. While the amount of additional effort to work with this model is well worth the returns (see Chapter 8), that may not be as true for a model that's too much more complex.

Chapter 4

REDUCTION TECHNIQUES

Reduction techniques are well known to be important when solving hard optimization problems [48]. They aim to reduce the instance data for a problem while preserving optimal solutions, with the goal of increasing solver efficiency. This is commonly achieved by applying logic-based modifications.

Reduction techniques were applied before attempting to solve RCPSP+L instances. Any instance that is invalid (see explanation of RCPSP+L model in Section 3.2) or doesn't add any value to examine is removed. Out of convenience, these reduction steps were performed during instance data file generation. This chapter describes the applied reduction techniques, which affect directed cycles, precedence-induced learning effects, and minimum durations.

4.1 Directed Cycles

All instances with precedence networks that include directed cycles have been removed. If the precedence relationship $(i, j) \in A$ is true, then $(j, i) \in A$ would be impossible to satisfy. This is true for both direct and indirect precedence relationships. Removal of these invalid instances is achieved by using a data source with known-good RCPSP data sets (see Section 7.1).

4.2 Precedence-Induced Learning Effects

All instances with learning relationships between two tasks that also had a precedence relationship were removed. If a direct or indirect precedence relationship $(i, j) \in A$ is true, then the solver never has the opportunity to examine a potential schedule where $j \prec i$. Adding a learning relationship between the two tasks would always have a completely predictable outcome and thus be unhelpful in answering the research questions posed in this paper.

If both $(i, j) \in A$ and a learning relationship $(i, j) \in L$ are true, learning would always occur. This case would still provide some information about how much learning effects can impact the calculation of project makespans. However, this paper centers on how projects might be scheduled differently if learning effects are taken into account in the first place, not just how much they'd be shortened by during after-the-fact makespan calculations.

If both $(i, j) \in A$ and $(j, i) \in L$ are true, learning would never occur. This case is thus irrelevant to this paper's examination of learning effects.

Both cases also eliminate start/end tasks ever being teachers or students; they are a precedence or successor for all other tasks (see Section 7.1).

Removal of these invalid instances is achieved by only selecting valid students while constructing test instances (see Section 7.2.2).

4.3 Minimum Durations

All instances where students have a duration too small to improve were removed. If no improvement is possible, then no learning effect could ever be observed, so it has no effect on how the solver makes the schedule.

If $d \leq 1$ for a task, its duration can't be shortened. The data source used for this paper only uses whole numbers for durations (see Section 7.1). Thus, the minimum duration any real task can have is 1. This also eliminates the start/end tasks ever being students because $d_{j_0} = d_{j_{n+1}} = 0$ (see Section 7.1).

Additionally, if $d = d'$ for a task, then by definition the task's duration can't be shortened. Depending on an instance's learning intensity, this frequently affects short tasks (e.g. $d = 2$ at $\lambda = 10\%$), and even affects longer tasks at high learning intensities (e.g. $d = 10$ at $\lambda = 90\%$).

Removal of these invalid instances is achieved by only selecting valid students while constructing test instances (see Section 7.2.2).

Chapter 5

CONSTRAINT PROGRAMMING FORMULATIONS

To characterize the RCPSP+L model, this thesis utilizes Constraint Programming (CP), a solution approach proven to be highly successful for scheduling problems. Multiple CP formulations are proposed with the goal of improving solver efficiency. Each formulation in this chapter is assigned a label of the form $F_{(\#)}$, in addition to its descriptive name, for easier referencing.

This chapter begins with an explanation of what CP is. It then defines the CP formulation for the traditional RCPSP model (F_0). The final section proposes CP formulations for the RCPSP+L model: Logical Duration (F_1), Dynamic Duration (F_2), Multi-Mode (F_3), and Bi-Objective (F_4).

5.1 Introduction to Constraint Programming

CP is an exact solution approach for operations research problems. A scheduling problem formulated using CP largely consists of four main parts:

DATA Information from the instance to be solved (e.g. a list of jobs).

DECISION VARIABLE(S) The answer(s) the solver is being asked to produce (e.g. when each job is scheduled).

OBJECTIVE The solver's goal, which defines what an optimal solution would be (e.g. minimize the makespan).

CONSTRAINTS Restrictions that solutions must meet to be valid (e.g. precedence).

The decision variables used in the formulations in this chapter are time-interval variables that specify a start and an end time. The schedules in in these formulations are represented by an array of all the interval decision variables, one interval per task.

CP solvers work by defining the domain of all possible values for the decision variables and dividing it into branches at every decision point. It then uses propagators to narrow down the branches on the tree until it finds the optimal one [38]. The propagators search in different ways depending on the solver: it may focus on investigating branches with feasible solutions, it may seek to first eliminate infeasible branches, etc. By default, this version of the CP Optimizer solver used in this thesis (see Section 7.3) uses a combination of the Large Neighborhood Search and Failure-Directed Search algorithms when solving scheduling problems [51]. A feasible solution is proven to be optimal once all other branches have been eliminated.

5.2 Base RCPSP (F_0)

A known-efficient RCPSP formulation F_0 included with this version of CP Optimizer is used as the basis for all the RCPSP+L formulations in this paper [35]. In addition to the notation from Chapter 3, let y_j be an interval variable for each job $j \in J$:

$$\textit{Min} \max_{\mathtt{j\in J}}\big(\mathtt{end}(y_j)\big) \tag{5.1a}$$

$$\textit{Subject to } \mathtt{cumulative\_function}(u_{r,y_1}, \ldots, u_{r,y_n}) \leq r \qquad \forall\, r \in R, \tag{5.1b}$$

$$\mathtt{end\_before\_start}(y_i, y_j) \qquad \forall\,(i,j) \in A, \tag{5.1c}$$

$$y_j \;\mathtt{interval\ variable\ in}\; [0, z] \;\mathtt{of\ length}\; d_j \qquad \forall\, j \in J. \tag{5.1d}$$

The interval for a given job j is represented by y_j. The objective in eq. (5.1a) is to minimize the time period during which the final task ends. The per-period resource capacity constraints are handled in eq. (5.1b) by a cumulative function expression. Precedence constraints are implemented by eq. (5.1c). Equation (5.1d) fixes the length of each job's interval to that job's pre-defined duration.

5.3 RCPSP+L Formulations

Four RCPSP+L CP formulations are proposed in this section. They vary in objectives, decision variables, and constraints. They all reduce to F_0 when $L = \emptyset$.

5.3.1 Logical Duration (F_1)

The Logical Duration (F_1) RCPSP+L formulation incorporates learning effects through individual duration constraints for every possibility. Building upon F_0:

$$\begin{aligned}
&\mathit{Min} \max_{\mathtt{j \in J}} \big(\mathtt{end}(y_j)\big) && && (5.2a)\\
&\mathit{Subject\ to} \text{ eqs. (5.1b) and (5.1c),} && && (5.2b)\\
&\qquad \big(\mathtt{end}(y_i) \le \mathtt{start}(y_j)\big) \implies \big(\mathtt{length}(y_j) = d'_j\big) && \forall\,(i,j) \in L, && (5.2c)\\
&\qquad \big(\mathtt{end}(y_i) > \mathtt{start}(y_j)\big) \implies \big(\mathtt{length}(y_j) = d_j\big) && \forall\,(i,j) \in L, && (5.2d)\\
&\qquad \mathtt{length}(y_j) = d_j && \forall\,(i,j) \notin L, && (5.2e)\\
&\qquad y_j \ \mathtt{interval\ variable\ in}\ [0, z]\ \mathtt{of\ length}\ [d'_j, d_j] && \forall\, j \in J. && (5.2f)
\end{aligned}$$

Equations (5.2a) and (5.2b) show that F_1 uses the same objective, resource constraints, and precedence constraints as F_0. F_1 differs from F_0 in how the duration of a task is set; this incorporates any learning that occurs. Instead of being fixed, eq. (5.2f) allows the length of each job interval to be between its alternate and original durations, and the value is set depending on a task's circumstances:

- If $i \prec j$, eq. (5.2c) ensures j uses its alternate duration.
- If $i \succeq j$, eq. (5.2d) ensures j uses its original duration.
- If j has no possible teacher provided in the instance data file, eq. (5.2e) ensures j uses its original duration.

5.3.2 Dynamic Duration (F_2)

The Dynamic Duration (F_2) RCPSP+L formulation incorporates learning effects by calculating duration dynamically.

$$\textit{Min} \max_{\mathtt{j \in J}} \big(\mathtt{end}(y_j)\big) \tag{5.3a}$$

$$\textit{Subject to } \text{eqs. (5.1b), (5.1c), (5.2e), and (5.2f),} \tag{5.3b}$$

$$\mathtt{length}(y_j) = d_j - (d_j - d'_j) \times \big(\mathtt{end}(y_i) \leq \mathtt{start}(y_j)\big) \quad \forall\,(i,j) \in L. \tag{5.3c}$$

Equation (5.3c) calculates the duration of learning-capable jobs all in one, without needing separate constraints to handle $i \prec j$ vs. $i \succeq j$. The math works because this version of CP Optimizer evaluates the Boolean $\big(\mathtt{end}(y_i) \leq \mathtt{start}(y_j)\big)$ to 1 if true and 0 if not when performing the calculation.

5.3.3 Multi-Mode (F_3)

Multi-Mode RCPSP (often shortened to MM-RCPSP or MRCPSP) is a well-known generalization of the RCPSP in the literature that has been shown to be computationally efficient. Instead of fixed duration and resource consumption, MM-RCPSP allows tasks to be performed in one of multiple alternative modes that may have different requirements.

The Multi-Mode (F_3) RCPSP+L formulation incorporates the effect of learning by choosing between a learning and non-learning mode for each task. Let optional interval variables z_j and z'_j represent the two modes for each job $j \in J$:

$$\mathit{Min}\ \max_{\mathtt{j \in J}}\big(\mathtt{end}(y_j)\big) \tag{5.4a}$$

$$\textit{Subject to}\text{ eqs. (5.1b) and (5.1c),} \tag{5.4b}$$

$$\mathtt{alternative}(y_j, [z_j, z'_j]) \qquad \forall\,(i,j) \in L, \tag{5.4c}$$

$$\mathtt{presence\_of}(z'_j) = \big(\mathtt{end}(y_i) \le \mathtt{start}(y_j)\big) \qquad \forall\,(i,j) \in L, \tag{5.4d}$$

$$\begin{array}{l} z_j, z'_j\ \mathtt{optional\ interval\ variables} \\ \qquad\qquad \mathtt{in}\ [0, z]\ \mathtt{of\ length}\ [d'_j, d_j] \end{array} \qquad \forall\, j \in J. \tag{5.4e}$$

Note that unlike in eq. (5.2f), the duration intervals in eq. (5.4e) are optional — this is necessary because the solver can only use one of them and thus must leave the other blank. The alternative constraint in eq. (5.4c) requires the solver to use exactly one of the task mode options. Like in F_2's duration calculation, eq. (5.4d) takes advantage of the fact that this version of the CP Optimizer solver evaluates Booleans to 1 or 0 when determining which mode to use: the learning mode z'_j (which has duration d') is used if $i \prec j$. Otherwise, the learning mode is suppressed, which inherently requires the presence of z_j due to eq. (5.4c).

5.3.4 Bi-Objective (F_4)

The Bi-Objective (F_4) RCPSP+L formulation is identical to F_2, but with the addition of a secondary objective that encourages the use of learning [10]. Let l represent the number of tasks which actually learn from their teachers in the solution (i.e. where $\mathtt{length}(y_j) = d'_j$):

$$Min \max_{\mathtt{j \in J}} \big(\mathtt{end}(y_j)\big) - \frac{l}{|J|+1} \tag{5.5a}$$

$$\textit{Subject to } \text{eqs. (5.1b), (5.1c), (5.2e), (5.2f), and (5.3c).} \tag{5.5b}$$

The only difference between this formulation and F_2 is the additional factor $\frac{l}{|J|+1}$ in the objective in eq. (5.5a). This factor incentivizes the solver to prefer solutions that utilize more learning relationships, but its value is small enough $\frac{l}{|J|+1} < 1$ that the solver should only use it as a tie breaker between two solutions with the same makespan, acting as a secondary objective.

Note that the actual makespan thus no longer corresponds to the obtained objective function value. However, it's still easily obtained as $MS = \mathtt{max}\big(\mathtt{end}(y_j)\big)$.

Chapter 6

BOUNDING TECHNIQUES

At a certain instance difficulty level or size, it's inevitable that even the best CP formulation may not be able to solve the instance to optimality within a reasonable amount of time. In these cases, bounding techniques are utilized to intelligently tighten the ensuing optimality gaps.

This chapter opens with an introduction to what suboptimal solutions are and bounding techniques are used to address them. It then delves into the bounding techniques used in this paper for RCPSP+L instances, including bounds automatically generated by the CP formulations, lower and upper bounding model relaxations, and a destructive lower bounding process.

Propositions regarding the bounding techniques in this chapter use the following notation: For a given instance solved with a model x, let $opt(x)$ be the optimal makespan, with $lb(x)$ and $ub(x)$ respectively be the valid lower and upper bounds.

6.1 Introduction to Bounding

A feasible makespan is one for which a valid solution is known. A feasible makespan is said to be optimal if it's proven that no shorter makespan is possible for the given instance, and an instance with a known optimal solution is said to be solved. Makespans that are feasible but proven not to be optimal are called suboptimal. A lower bound lb is an integer such that makespan of an optimal solution is $\geq lb$.

Similarly, An upper bound ub is an integer such that makespan of an optimal solution is $\leq lb$.

Though optimality is the goal, it can years to solve some of the most difficult instances (such as ones with many jobs). In the PSPLIB library of RCPSP instances (discussed further in Section 7.1), some instances with just 60 jobs still have yet to be solved to optimality over 20 years later [29]. For each of these unsolved instances, its optimality gap is considered, which is the gap between its lb and ub. (This measurement is sometimes called the optimality guarantee because it lends insight into how good a given makespan is for the instance.) Because the objective in RCPSP is to minimize the makespan, ub in this case is the smallest valid makespan known. Optimality gaps in this paper are calculated as follows, with makespan MS:

$$\text{Optimality Gap \%} = \frac{MS - lb}{MS} \times 100 \tag{6.1}$$

A gap of $< 5\%$ is generally considered acceptable in practical applications.

6.2 Formulation Bounds

The CP Optimizer solver version used in this thesis (see Section 7.3) automatically outputs a lower and upper bound for every solution it produces. (For RCPSP+L, $ub = MS$, so in practice this means the solver outputs lb and MS.) By solving each instance using all of the formulations in Chapter 5, the different formulations' strengths can easily be taken advantage of by choosing the best lb's and ub's produced.

6.3 Model Relaxations

Model relaxations involve taking the main model to be solved and relax its constraints, with the goal of producing a problem that's easier to solve. These easier problems can then be used to more quickly produce lower or upper bounds for the main model. In this paper, the relaxations involved RCPSP+L formulations with the resource and/or precedence constraints completely removed and/or the durations fixed at the original or alternate values.

The model relaxations used to improve lower bounds in this paper are listed in Table 6.1. Like with the $F_{(\#)}$ notation for the various CP Formulations in Chapter 5, the lower bounding model relaxations are labelled with $LB_{(\#)}$ for easier referencing. The makespans for these models are valid lower bounds for RCPSP+L.

Table 6.1: Lower Bounding Model Relaxations

	Resources	**Precedences**	**Durations**
RCPSP$^-$ (LB_1)	✓	✓	Alternate (d')
RCSP+L (LB_2)	✓	—	Variable (d or d')
PSP+L (LB_3)	—	✓	Variable (d or d')
PSP$^-$ (LB_4)	—	✓	Alternate (d')

Fewer upper bounding relaxations were used than lower bounding ones because relaxing resource and precedence constraints inherently shortens the makespan. Thus, relaxing the only other constraint in RCPSP+L, duration, produced the sole upper bounding relaxation described in this section (listed in Table 6.2), labelled as UB_1 for easier referencing. The makespan for this model is a valid upper bound for RCPSP+L.

Table 6.2: Upper Bounding Model Relaxations

	Resources	**Precedences**	**Durations**
RCPSP$^+$ (UB_1)	✓	✓	Original (d)

These lower and upper bounding model relaxations are covered in the subsections that follow.

6.3.1 RCPSP^{-} (LB_1)

The RCPSP^{-} (LB_1) relaxation fixes all potential student tasks to their alternate durations d' (tasks with no learning ability are left at d). This represents the maximum learning possible for an instance. Potentially, this is even shorter than is possible for a optimal RCPSP+L solution of the instance, if it's impossible for that instance to schedule every potential student after their teacher.

The learning effect can only ever shorten the makespan, and LB_1 has the maximum possible amount of learning, so LB_1 will always produce makespans at least as good as RCPSP+L.

Proposition 1. For an RCPSP+L instance, it holds that $lb(LB_1) \leq opt(\text{RCPSP+L})$.

6.3.2 RCSP+L (LB_2)

The RCSP+L (LB_2) model relaxes the precedence constraints (i.e. $A = \emptyset$), allowing tasks to be scheduled in any order whatsoever. This effectively becomes a resource packing problem, but with the added benefit of potentially reduced durations and resource consumption due to the learning effect. LB_2 will always produce makespans at least as good as RCPSP+L.

Proposition 2. For an RCPSP+L instance, it holds that $lb(LB_2) \leq opt(\text{RCPSP+L})$.

6.3.3 PSP+L (LB_3)

Though RCPSP is a standard scheduling problem in the literature, RCPSP itself is an extension of the classic Project Scheduling Problem (PSP); PSP is RCPSP without resource considerations. In a similar vein, the PSP+L (LB_3) model relaxes RCPSP+L's resource constraints. Thus, LB_3 is also a learning extension of the classic PSP. Without the restriction of resource capacities, LB_3 will always produce makespans at least as good as RCPSP+L.

Proposition 3. For an RCPSP+L instance, it holds that $lb(LB_3) \leq opt(\text{RCPSP+L})$.

6.3.4 PSP$^-$ (LB_4)

The PSP$^-$ (LB_4) model relaxes the resource constraint and fixes all potential students to their alternate durations d'. That makes it similar to LB_1 without the resource constraint, and LB_3 with enforced minimum durations for potential students, which means LB_4 will always produce makespans at least as good as either one. As a result, LB_4 will also always produce makespans at least as good as RCPSP+L.

Proposition 4. For an RCPSP+L instance, it holds that $lb(LB_4) \leq opt(\text{RCPSP+L})$.

6.3.5 RCPSP$^+$ (UB_1)

The RCPSP$^+$ (UB_1) relaxation fixes all tasks to their original durations d. This effectively removes the learning effect from RCPSP+L (i.e. $L = \emptyset$), reducing the problem to RCPSP. Thus, UB_1 can be simply implemented as F_0 (Section 5.2).

Like in LB_1, the learning effect in RCPSP+L can only ever shorten the makespan, so RCPSP+L will always produce makespans at least as good as UB_1.

Proposition 5. For an RCPSP+L instance, it holds that $ub(UB_1) \geq opt(\text{RCPSP+L})$.

6.4 Destructive Lower Bounding (DLB)

Destructive Lower Bounding (DLB) is a method for producing strong lower bounds. DLB is not a model or formulation in and of itself; it's a technique applied using an existing model formulation. The bounds produced by DLB can potentially be even better than the ones provided automatically by CP for that formulation. In essence, DLB starts at a specified *lb* and attempts to prove its infeasibility. If the makespan is infeasible, the *lb* is "destroyed", the next *lb* is similarly tested, and the process repeats until a destruction attempt hits a specified time limit or the *lb* is proven feasible [47]. Any *lb* proven feasible by DLB is inherently the optimal one because every makespan less than it has just been proven infeasible. The DLB implementation used in this thesis is detailed in Algorithm 6.1.

Algorithm 6.1: Destructive Lower Bounding

```
  Input: $P$ (RCPSP+L instance)
         $t_{max}$ (time limit per iteration)
  Output: $lb$ (valid lower bound for $P$)
1 $lb \leftarrow 0$
2 while isInfeasible($P, MS \leq lb, t_{max}$) do
3     $lb \leftarrow lb + 1$
4 return $lb$
```

On line 1 of Algorithm 6.1, *lb* could instead be initialized with a known-good lower bound to reduce the solve time, such as one calculated by a model relaxation (e.g. PSP+L). In practice, the lowest *lb* values are destructed so quickly for RCPSP+L instances that the time savings from starting at a higher relaxation-defined *lb* are almost nonexistent. Note that the time limit t_{max} applies to each tested *lb* individually, not the time spent cumulatively.

Proposition 6. For an RCPSP+L instance, it holds that $lb(DLB) \leq opt(\text{RCPSP+L})$.

Chapter 7

EXPERIMENT SETUP

To understand the RCPSP+L model and evaluate solution approaches, this thesis examines many parameters, data sets, formulations, and bounding approaches. As a result, hundreds of thousands of tests were run. This chapter describes the connection of our instances to the literature, generation of the learning test instances, and the computer setup used to perform the experiments.

7.1 Data Source

All data was drawn from the popular PSPLIB library of resource-constrained project scheduling problems [32], generated using the ProGen data generator [33]. It covers many types of RCPSP-focused problems, and each problem type has data sets for a range of problem sizes. Published solutions are included from the literature: optimal solutions are published when available, heuristic and other suboptimal solutions are published for problems that have not yet been solved to optimality. Indeed, some of the PSPLIB RCPSP instances with just 60 jobs are still unsolved today.

The data for the experiments in this paper were all pulled from the RCPSP single mode data sets. Depending on the experiment, either the easiest or most difficult instances from this set are used. The easiest data set in this category, the instances with $n = 30$ jobs ("j30"), comprises 480 test instances which have all been solved to optimality. The most difficult has $n = 120$ job instances ("j120"), and many of that set's 600 test instances are still unsolved today.

The instance data files start by defining the number of jobs and the total resource capacities. Each problem includes an empty start node and an empty end node that is respectively an direct/indirect predecessor or successor for all of the other jobs in the problem. Thus, the j30 problem actually has $|J| = 32$ jobs. All the problems used in this paper have 4 different renewable resources (that is, they replenish every time period). However, each problem has a different provided capacity for each resource. Additionally, each of the jobs in a test instance has the following attributes:

- Task ID
- Duration
- Consumption of each resource type
- Successors (empty for final node)

7.2 Test Instance Generator

A Python script is used to generate RCPSP+L test instances. First, it inputs data for an RCPSP test instance (the "base case") from a PSPLIB `.RCP` data file. Next, it selects which jobs will be students, assigns them teachers, and calculates their alternate durations. Then, it outputs this RCPSP+L test instance to a folder as an OPL `.dat` data file (see Section 7.3). Afterward, it repeats this process for every base case, learning frequency, learning intensity, and random seed requested. Finally, it outputs a `.dat` file containing a list of filenames for all the test instances it just generated.

An example instance data file and an example filenames file are included in Appendix B. The source code for the test instance generator script is included in Appendix A.

This whole script is rerun every time a new experiment has different parameters. The script must be altered and instances re-generated when used for destructive lower bounding. In all, 124,800 test instances were generated for this thesis.

7.2.1 Test Instance Parameters

The script utilizes the following three parameters when creating each test instance:

LEARNING FREQUENCY (ϕ) *Target* percentage of tasks with the *potential* to be students. Minimum of 0%, maximum of 100%.

LEARNING INTENSITY (λ) Percentage a student's duration is reduced by to calculate the alternate duration. Minimum of 0%, maximum of 100%.

RANDOM SEED (a) Value to seed the random number generator with reduce the influence of outlier results and ensure consistency during testing.

Note that ϕ is a target value because the generator may not be able to provide that many if there are insufficient legal student/teacher pairs. Also note that ϕ denotes potential learning because it simply provides the option for the solver to utilize learning for those tasks; it doesn't guarantee that the solver will take advantage of the learning effect for all of the potential students in the solutions it produces.

Using these parameters, the script selects which tasks will have the potential to be students, and then assigns each student a teacher and an alternate duration d'.

7.2.2 Student Selection

Because the parameter for the target number of potential students is given as a percentage, the first step is to determine how many tasks that represents. The script converts it to an integer value, rounding up and ensuring a minimum value of 0. Specifically, given a data file with project size n (not including empty start/end nodes) and a learning frequency ϕ (in this equation as a whole number), the script calculates the target student count s_{tc}:

$$s_{tc} = \left\lceil n \times \frac{\phi}{100} \right\rceil, \quad s_{tc} \geq 0 \tag{7.1}$$

For example, a project with $n = 30$ and $\phi = 25$ (learning frequency of 25%) would have $s_{tc} = 8$.

After calculating s_{tc}, the script attempts to select that many students. The student selection process is shown in Algorithm 7.1.

Algorithm 7.1: Student Selection Process

```
  Input: $J$ (set of jobs)
         $s_{tc}$ (target student count)
         $a$ (random seed)
  Output: $S$ (set of selected students)
1 $random.seed \leftarrow a$
2 $S \leftarrow \{\}$
3 $Tested \leftarrow \{\}$
4 while $|S| < s_{tc}$ and $|Tested| < |J| - 2$ do
5     $candidate \leftarrow$ randomChoice($J \setminus Tested$)
6     if isValidStudent($candidate$) then
7         $S \leftarrow S \cup \{candidate\}$
8     $Tested \leftarrow Tested \cup \{candidate\}$
9 return $S$
```

Note in line 6 that the script will only select students that are valid (see Chapter 4 for definition of valid). Sometimes the number of tasks in the data file that qualify as valid students is lower than s_{tc}. This often happens when a very high ϕ is requested. In these cases, the script selects as many valid students as possible and then stops; the number that was selected in the end is called the actual student count s_{ac}.

Given actual student count s_{ac} and project size n (not including empty start/end nodes), the actual student percentage s_{ap} for the data file is:

$$s_{ap} = \frac{s_{ac}}{n} \times 100 \tag{7.2}$$

In the instance data file, any task that's not selected as a potential student is listed with a teacher of 0 and $d' = 0$.

7.2.3 Teacher Selection

After selecting students, the script assigns each student a teacher. The teacher selection process is shown in Algorithm 7.2.

Algorithm 7.2: Teacher Selection Process

```
Input: $J$ (set of jobs)
       $M$ (set of selected students)
       $a$ (random seed)
Output: $P$ (set of student-teacher pairs)
1 $random.seed \leftarrow a$
2 for $student$ in $M$ do
3     $teacher \leftarrow 0$
4     while $teacher$ is 0 do
5         $teacher \leftarrow$ randomChoice($J$)
6         if not isValidTeacher($teacher$) then
7             $teacher \leftarrow 0$
8     $P = P \cup \{student, teacher\}$
```

Note in line 6 that the script will only select teachers that are valid (see Chapter 4 for definition of valid). Note that multiple students may share the same teacher. Not mentioned in this algorithm is that all non-student jobs that are assigned a teacher of 0 to indicate that they can't learn; the formulations in Chapter 5 take advantage of this in their duration constraints. Hence, the solver will always be able to assign a teacher for any given student unless that student is a predecessor or successor for all other tasks in the instance (which is only ever true for the empty start/end tasks in a valid RCPSP or RCPSP+L instance).

7.2.4 Alternate Duration Calculation

The alternate duration for a student is calculated as follows:

$$d' = d - \left\lceil d \times \frac{\lambda}{100} \right\rceil, \quad d' \geq 1 \tag{7.3}$$

Additionally, the script ensures $d' \geq 1$ for all students, and manually sets , $d' = 0$ for all non-students. The formulations in Chapter 5 take advantage of the latter fact in their duration constraints.

7.3 System and Software Configuration

All testing is performed on a Windows 10 PC with a 3 GHz/4 core Intel Core i5-2320 CPU and 8 GB of RAM.

The tests are solved using CP Optimizer version 12.9.0, which is part of the IBM ILOG CPLEX Optimization Studio software package. While running the tests, the only programs running on the computer are the `oplrun` command line tool (which runs the tests) and TeamViewer (to control the computer remotely). `oplrun` is used instead of the GUI interface to reduce memory consumption. CP Optimizer's default settings (specified in [22]) are used, except for the following changes:

- cp.param.TimeLimit — time limit for each test — varies by test regimen

- cp.param.Workers — the number of computer cores used while solving — set to 4 (this computer's maximum)
- cp.param.TimeMode — the time reporting method — set to "ElapsedTime" (to be more human readable)
- cp.param.LogVerbosity — how frequently to write to the engine log — set to "Quiet" (to consume less computer resources)

The solver models are written in IBM's OPL language. Each model defines what data to import from a problem's data file, the decision variables, the objective, the constraints, and solution validation post-processing steps. Every formulation and bounding technique described in Chapter 5 is represented by a different model file. The source code for one of the OPL models is included in Chapter C as an example.

Batch runner systems are constructed using IBM's ILOG Script to automate running the experiments en masse. Different batch runners are used depending on how many models are being executed, whether destructive lower bounding is being utilized, etc. To execute a test regimen, a batch runner is provided with which model(s) to use and the set of problem data files to solve. For every model and problem file, the runner spawns a new instance of the model solver and attempts to solve it. The results for each test is added as a new row in a common `.csv` data file, including a wide variety of statistics like lower bound, upper bound, solve time, whether the solution is optimal, etc. After every test, a file containing the best solution the solver could find within the time limit is stored in a central repository

An example solution file is included in Appendix E. The source code for one of the batch runners is included in Appendix D.

Since CP Optimizer is usually used to run one problem at a time, and not over a hundred thousand, this system required a surprising amount of work to create. It pushes the boundaries of what CP Optimizer and OPL are designed to do. While running large test regimens, a memory leak was discovered due to the continuous spawning of new OPL models. By working with one of the developers of CP Optimizer at IBM, we discovered that the leak was due to a bug in OPL itself [49]. Though he was fortunately able to provide a workaround for part of the problem, the memory leak still continues with a reduced intensity. There was not enough time to continue investigating the problem before the experiments needed to begin, and the batch runner code couldn't be changed after starting the first experiments — any code change could affect the solver's results, rendering fair comparisons between all of the tests impossible. As a result, the batch runner's test regimens still must be restarted periodically, whenever the computer's memory becomes too clogged, to ensure that the amount of free memory available on the machine doesn't affect what solutions the solver finds. Depending how frequently the batch runner spawns new models for a given experiment, this must be done somewhere between every few hours and every few days for the duration of the test regimen.

Chapter 8

COMPUTATIONAL ANALYSIS

In all, 284,849 experiments are performed to characterize the RCPSP+L model and various solution approaches, requiring the testing system to run continuously for five months. This chapter analyzes the results of that experimentation, including a comparison of the CP formulations, analysis of the RCPSP+L model, study of the efficacy of the bounding techniques, and review of the impact the parameters have on instance difficulty.

8.1 Formulation Comparison

To compare the efficacy of the different CP formulations in Chapter 5, an RCPSP+L instance set is generated with the parameters specified in Table 8.1.

Table 8.1: Formulation Comparison Experiment Parameters

Parameter	**Values**
Number of PSPLIB Base Cases	600 ($n = 120$)
Learning Frequencies (ϕ)	$\{25, 50, 75, 100\}$
Learning Intensities (λ)	$\{10, 50, 90\}$
Random Seeds (a)	$\{0, 1, 2\}$
Formulations	$\{F_1, F_2, F_3, F_4\}$
Time Limit (t_{max})	60 seconds

With 600 PSPLIB base cases, 4 ϕ's, 3 λ's, and 3 a's, this requires the generation of 21,600 instances. At 21,600 instances $\times$ 4 formulations = 86,400 total solver runs, and a time limit of 60 seconds per run, this means the experiments in this section have a potential maximum total run time of 60 days.

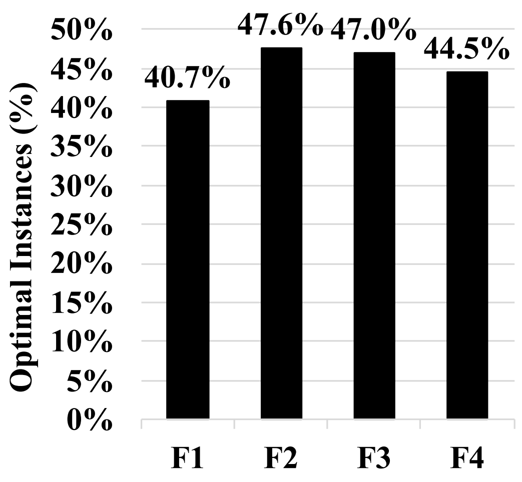

Figure 8.1: Percentage of Instances Solved to Optimality, by Formulation

Figure 8.1 shows what percentage of the instances were solved to optimality. The formulations in order from best to worst performing are F_2, F_3, F_4, F_1. However, the difference between F_2 and F_3 on this graph is negligible, and F_4 is not far behind. That being said, it does appear that F_1 performs worse than the other formulations. Interestingly, all of the formulations were unable to solve most of the instances; this makes sense, considering how difficult 120-job instances are known to be.

Most Effective: Too close to call. Either F_2 or F_3. Least Effective: F_1.

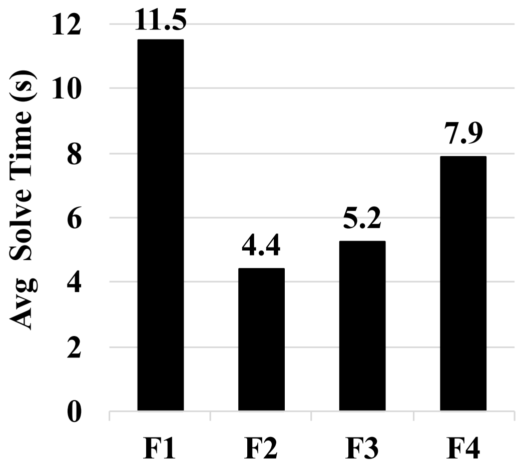

Figure 8.2: Average Solve Time for Optimal Instances, by Formulation

To gain more clarity, the formulations are next compared in Figure 8.2 by how long those optimal instances took to solve. This graph gives even stronger evidence that F_1 is dominated by the other formulations. F_4 also performs worse than F_1 and F_2 by a smaller margin, but it's still a significant difference with F_4 at twice the solve time for F_2. It makes sense that the formulations that solve more slowly in this chart also solved fewer to optimality in the previous one; after all, all formulations were given the same time limit. There is now evidence that F_2 is the best performing, solving instances roughly 20% faster on average than F_3

Most Effective: F_2. Least Effective: F_1 and F_4.

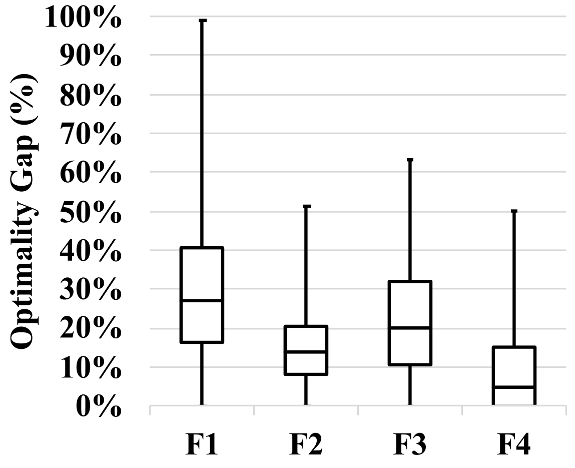

Figure 8.3: Optimality Gaps for Unsolved Instances, by Formulation

Since the previous analyses only considered the formulations' successes for optimal instances, Figure 8.3 shows the optimality gaps for suboptimal instances to give the whole picture. F_1 is again dominated by the other formulations and is clearly the poorest performing formulation. F_2 outperforms F_3 in the worst cases and ties in the best cases. F_4 does surprisingly well, easily outperforming all other formulations on average (with an average that's roughly 5%, a common benchmark for a good optimality gap) and tying F_2 at its worst.

Most Effective: F_4. Least Effective: F_1.

In the end, the only finding that can be made definitively is that F_1 is consistently dominated by the other formulations, and F_3 usually (albeit sometimes only slightly) is outperformed. It appears that F_2 is the best at solving to optimality, and F_4 is the best at tightening (but not closing) gaps. This makes some intuitive sense: F_4 is simply F_2 with a modified objective. The bi-objective modification in F_4 incentivizes the solver to take advantage of learning opportunities, which likely makes it easy to quickly find short solutions (learning inherently shortens durations), but makes it harder to arrive at the optimal one (learning options require more decisions and make it harder to solve).

Overall Most Effective: F_2 for solving to optimality, F_4 for tightening gaps.

8.2 RCPSP+L Model Analysis

After determining the most efficient method for solving RCPSP+L instances, that method can be used to characterize the RCPSP+L problem itself. An RCPSP+L instance set is generated for this task using the parameters specified in Table 8.2.

Table 8.2: RCPSP+L Model Analysis Experiment Parameters

Parameter	**Values**
Number of PSPLIB Base Cases	480 ($n =$ j30)
Learning Frequencies (ϕ)	$\{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$
Learning Intensities (λ)	$\{10, 30, 50, 70, 90\}$
Random Seeds (a)	$\{0, 1, 2\}$
Formulations	$\{F_2\}$
Time Limit (t_{max})	∞

All 79,200 instances are run until solved to optimality. Bounding techniques aren't necessary in this case because they're only used when optimal solutions are *not* reached. F_2 is used because it's shown in Section 8.1 to be the fastest formulation for solving RCPSP+L instances to optimality — while this has no effect on the solutions produced, it significantly reduces the time required to produce the results.

89.76% of the $n = 30$ instances solve in under a second, often virtually instantaneously. The maximum solve time of any instance in the set is 604 seconds. The entire experiment regimen is solved in ~18.5 hours. When compared to the $n = 120$ instances used in the other sections, this demonstrates the strong correlation between instance difficulty and number of jobs in the project.

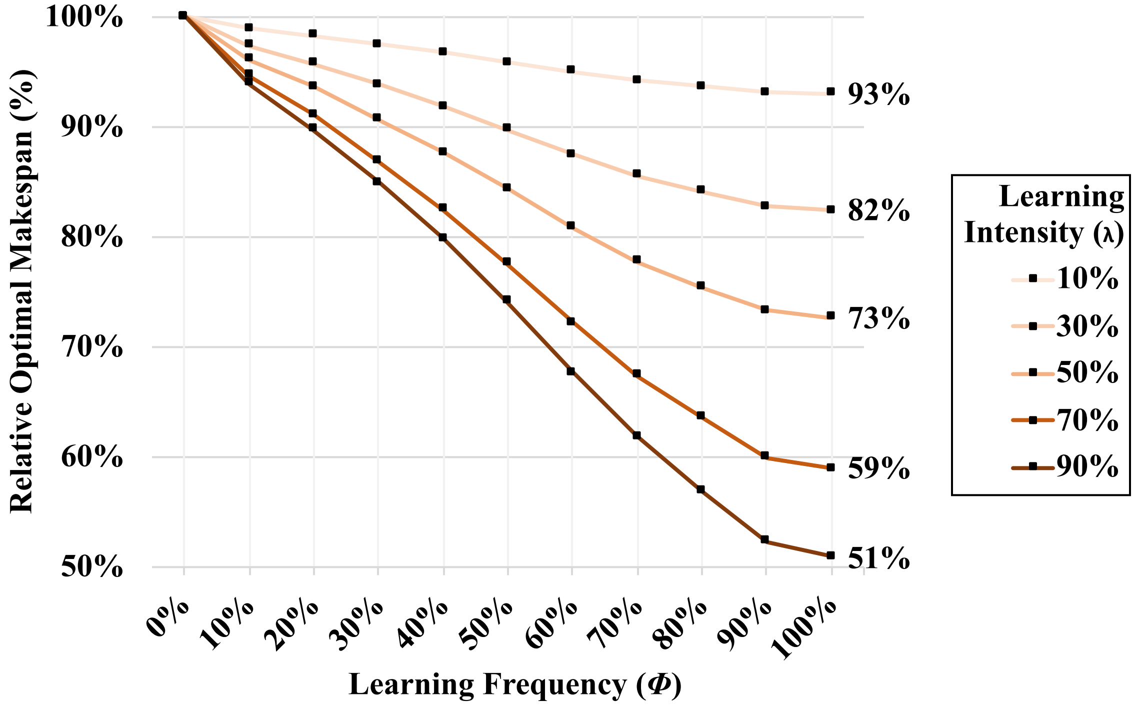

Figure 8.4: RCPSP+L Makespan Reduction Potential

Figure 8.4 shows some of the central results of this thesis. (It should be noted that while all points on the graph are produced by averaging to mitigate the influence of outliers, data points at $\phi \geq 80\%$ are composed of a smaller sample size when the target student count can't be reached, as discussed in Section 7.2.2.) The learning curves related to efficiency increases can be observed as learning frequency (ϕ) and intensity (λ) vary — upholding a common theme in the literature (Section 2.2). At the greatest ϕ and λ, the project makespan is theoretically shown to be more than halved. Increases in ϕ or λ alone enables significant makespan reductions, but the combination of both being strong has outstanding benefits.

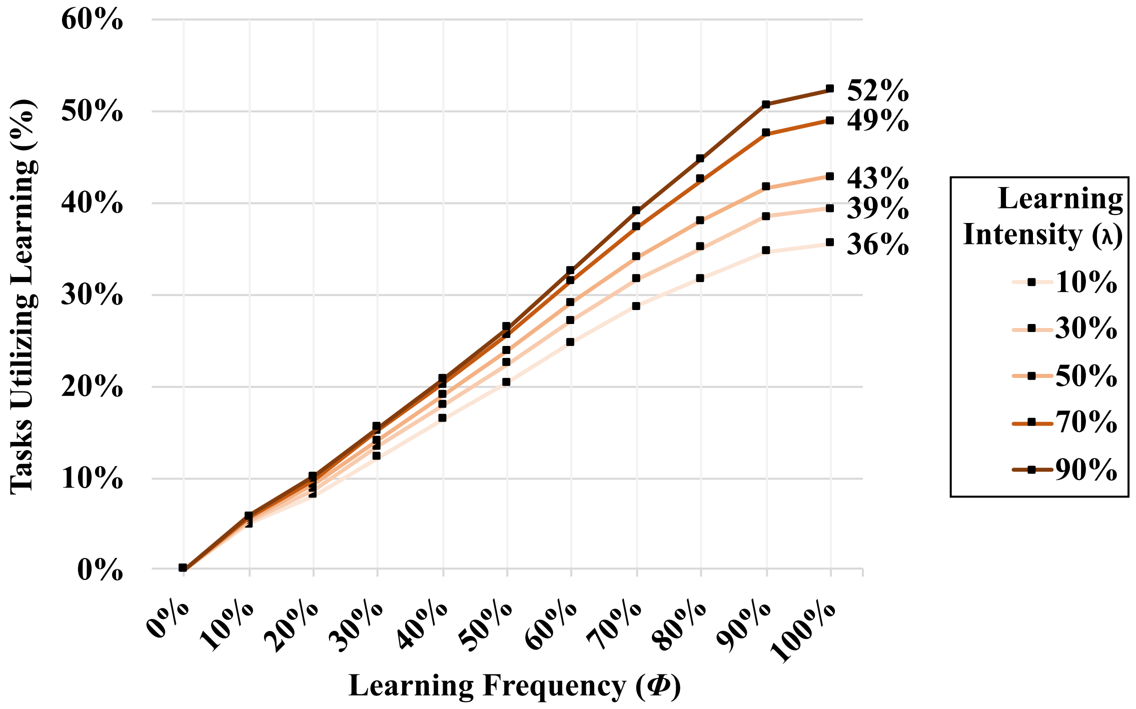

Figure 8.5: RCPSP+L Learning Utilization

Note that the makespan reduction plateaus around 50% even though the values of both ϕ and λ are significantly greater. This is because it's frequently impossible to utilize of all the potential learning relationships in L due how tasks must be ordered in the schedule. In fact, a schedule with the optimal makespan may skip a significant number of potential learning relationships. This phenomenon is displayed in Figure 8.5, which shows the average percentage of tasks in the project that actually utilize a learning duration reduction in the optimal solution at varying ϕ and λ. Even in the most extreme case of learning ($\phi = 100\%,\ \lambda = 90\%$), this value plateaus at 52.2%, much like the maximum average makespan reduction in Figure 8.4. However, while reduced λ is associated with reduced makespan reduction ($\phi\ =\ 100\%,\ \lambda\ =\ 10\% \implies 7\%$ makespan reduction), learning utilization remains relatively high whenever ϕ

does (e.g. $\phi = 100\%$, $\lambda = 10\% \implies 35.5\%$ learning utilization). Additionally, the maximum learning utilization by an individual instance solution in this experiment regimen in 83.3%, and the lowest individual utilization at $\phi = 100\%$, $\lambda = 90\%$ is 23.3%.

8.3 Bounding Efficacy

Since many of the instances still aren't solved to optimality in Section 8.1, regardless of formulation, bounding techniques from Chapter 6 can be applied to tighten the remaining optimality gaps. To compare the effectiveness of these bounding techniques, an RCPSP+L instance set is used with the parameters specified in Table 8.3. This set of experiments can utilize the instances generated and results produced in Section 8.1. These instances are then additionally solved with the bounding techniques.

Table 8.3: Bounding Efficacy Experiment Parameters

Parameter	**Values**
Number of PSPLIB Base Cases	600 ($n = 120$)
Learning Frequencies (ϕ)	$\{25, 50, 75, 100\}$
Learning Intensities (λ)	$\{10, 50, 90\}$
Random Seeds (a)	$\{0, 1, 2\}$
Formulations/Bounding Techniques	$\{F_1, F_2, F_3, F_4,$ $LB_1, LB_2, LB_3, LB_4,$ DLB (F_2, unsolved instances only), $UB_1\}$
Time Limit (t_{max})	60 seconds

Note that in this section, Destructive Lower Bounding (DLB) is performed using F_2 since that is shown in Section 8.1 to be the best formulation for solving RCPSP+L instances to optimality. (DLB is a bounding technique applied using an existing model formulation, rather than a separate model or formulation in and of itself.) Also note that DLB is only run on the instances that F_2 isn't able to solve to optimality within t_{max}; since DLB is solely a lower bounding technique and not a formulation, a technique for tightening the gap is irrelevant when the optimality gap is already 0%.

In addition to the 86,400 solver runs reused from the formulation comparisons (Section 8.1), 108,000 more are conducted for the bounding model relaxations, and DLB is applied to the 11,249 unsolved cases. This means that the analyses in this section draw from 205,649 total solver runs. A time limit of 60 seconds is still used to enable direct comparison between the bounding techniques and the CP formulations.

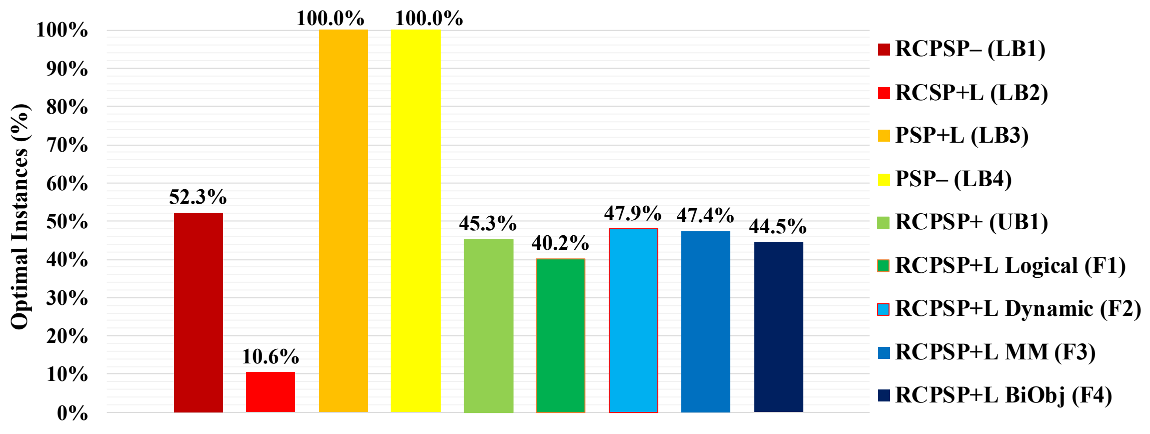

Figure 8.6: Percentage of Instances Solved to Optimality, by Bounding Technique

Figure 8.6 compares what percentage of the time all of the bounding model relaxations can solve instances to optimality, which is helpful for characterizing the relaxations. Formulations are included as a reference point. This comparison is unlike the one in Figure 8.1 because the relaxations are different models entirely, not just different formulations for solving the same problem. LB_3 and LB_4, for example, are able to solve instances to optimality 100% of the time. It's doubtful that this is because their formulations are so unbelievably superior to all the others — most likely, those problems are just significantly easier to solve. It could be inferred from this effect that resource constraints make the problems much more difficult. Strangely, precedence constraints seemingly make the problems easier to solve (see LB_2). It's possible that the combination of resource and learning constraints (or even the learning constraint alone) overwhelms the solver with too many options, and the precedence constraint limits the options just enough to give the solver guidance.

How successful the bounding techniques are at finding optimal solutions is interesting, but ultimately that's only one way to achieve their main job: producing better bounds. Yes, optimal solutions are inherently the best bounds, but the propositions in Chapter 6 define that any lower (or upper) bound produced by these lower (or upper) bounding methods is inherently useful as an RCPSP+L bound — of which, optimal ones are just a subset.

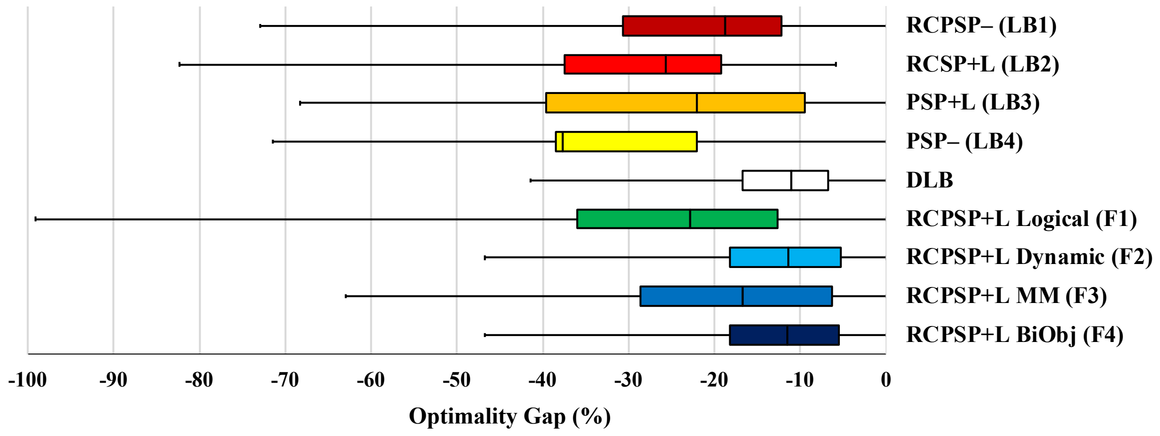

Figure 8.7: Lower Bounding Optimality Gaps, by Bounding Technique

Figure 8.7 shows how powerful the techniques are at producing any lower bound. The figure includes lower bounds produced by the lower bounding model relaxations, DLB, and the CP RCPSP+L formulations. Of note is that the model relaxations produce very poor lower bounds. The hardest model to solve (LB_2) is amusingly also one of the least useful for lower bounding. F_1 predictably performs poorly. F_2 and F_4 perform almost identically. (CP solvers have improved dramatically over the years, so it's possible the solver automatically takes many of the benefits from both F_2 and F_4 into account, thus making manually implemented differences between the two formulations less impactful. It's likely that this wouldn't have been the case if these tests were performed with solvers from a decade ago.)

Though a couple models have lower Q1 values on the chart, DLB actually dominates in performance — DLB beats or ties the best lower bound for 98.5% of the instances. F_2, F_3, and F_4 are able to beat or tie the best lower bound only 27.4%, 23.4%, and 24.8% of the time, respectively. F_1 was only able to beat or match the best lower bound a paltry 5.5% of the time. Nonetheless, F_1 did produce a better bound than DLB in exactly two of the tens of thousands of instances tested; though even in those cases it was still tied with F_4, and DLB was only one off from the best value.

Most Effective LB: DLB. Least Effective: The model relaxations and F_1.

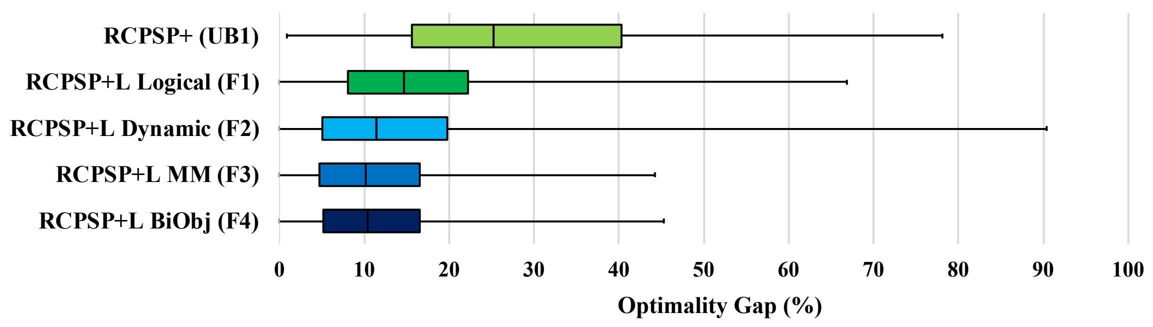

Figure 8.8: Upper Bounding Optimality Gaps, by Bounding Technique

The upper bounding methods are compared in Figure 8.8. As expected, F_1 performs poorly, again only producing the best bounds 4.9% of the time — however, it does outperform UB_1, which is the traditional RCPSP. UB_1 produces poor bounds for RCPSP+L, much like all the other model relaxations. The best upper bounds are produced (i.e. beat or tied) by F_2 58.5%, F_3 58.3%, and F_4 51.0% of the time. This means that F_2 and F_3 are tied for producing the best bounds the most often. However, F_3 and F_4 are tied in terms of spread on the box plot. This means that while F_2 has a better best case scenario than F_4, F_4 has the less bad worst case scenario, and F_3 is tied for the best by both metrics.

Most Effective UB: F_3 Least Effective: UB_1 and F_1.

Overall Most Effective: DLB for lower bounds, F_3 for upper bounds.

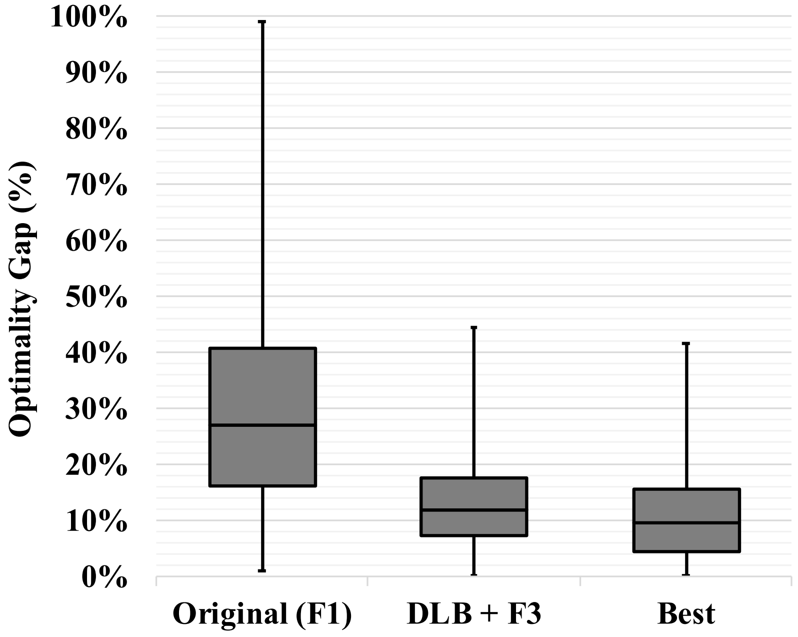

Figure 8.9: Improvement from Original to Best Bounding Technique

Figure 8.9 shows how much of an improvement was achieved by using more advanced formulations and bounding techniques. The box plot on the left of the graph is F_1, the first formulation made during the research for this thesis. While it's not surprising that later, more advanced formulations and bounding techniques have been able to outperform its bounds, the amount the most effective bounding techniques improve upon the original formulation is dramatic. The plot in the middle of the graph represents the superior bounding technique proven in this section: combining DLB's lower bounds with F_3's upper bounds. For reference, included on the right is what would happen if all the techniques were applied to every instance and the best possible bounds were taken (e.g. for the two instances where F_1 produced the best lower bounds, use F_1 instead of DLB). It would be highly impractical in reality to always solve every instance 10 times over in search of a better bound (and it'd probably be more successful in that case to just run the solver 10 times longer), so it's convenient that the simpler DLB + F_3 combination is almost as successful.

8.4 Difficulty Impact of Parameters

One of the topics mentioned throughout this chapter is the difficulty of solving $n = 120$ RCPSP+L instances. What has yet to be mentioned is that the difficulty is not uniform across all large project instances. The results from the experiments in Section 8.1 are used again in this section to analyze how the various parameters impact instance difficulty.

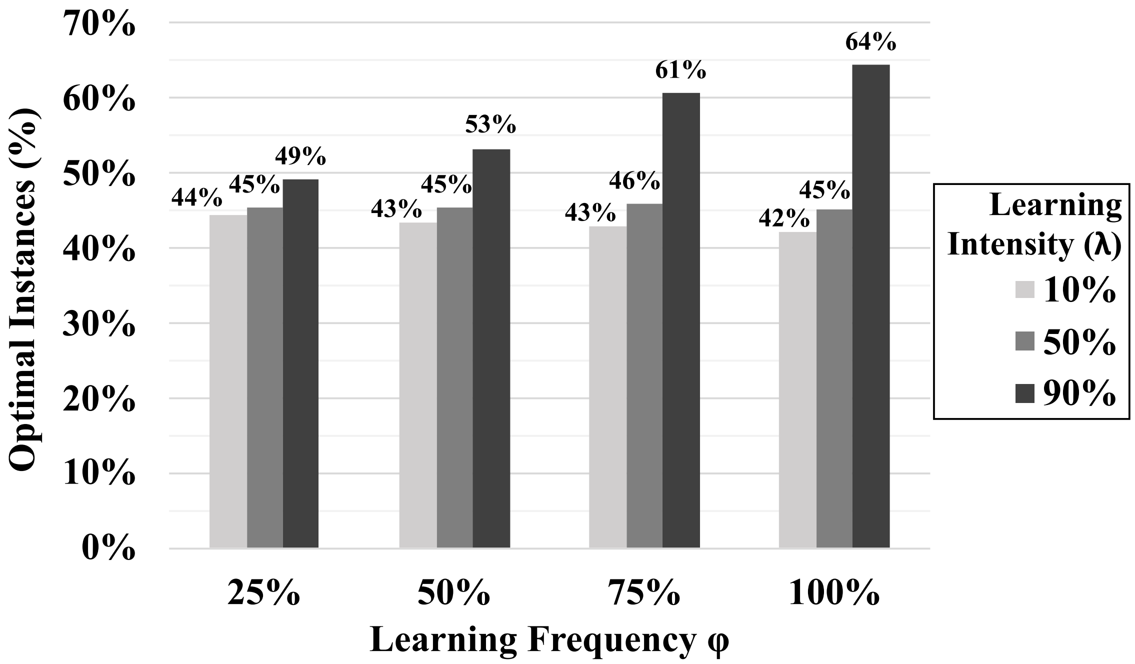

Figure 8.10: Percentage of Instances Solved to Optimality, by Parameter

Figure 8.10 shows that instances are of fairly consistent difficulty and low and medium values of λ. However, at high λ's the problem difficulty appears to have an inverse relationship with ϕ. This is likely because the solver is able to take advantage of learning benefits more easily, potentially even by accident, as more learning relationships exist and as the learning effect leads to a greater duration reduction.

However, the majority of those problems on average remain unsolved. In Figure 8.11, the optimality gaps tell a slightly different story than the percentage of optimal solutions did in Figure 8.10.

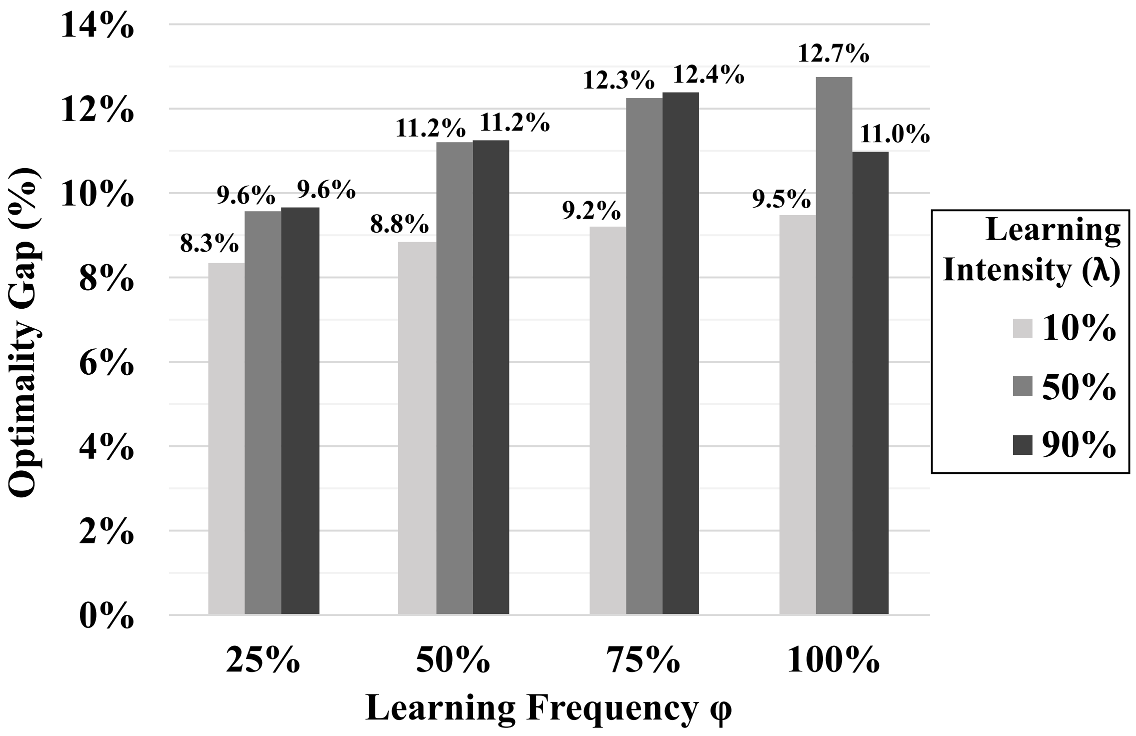

Figure 8.11: Optimality Gaps for Unsolved Instances, by Parameter

Low ϕ is associated with good optimality gaps, but increased ϕ or λ is correlated with larger gaps. The fact that $\lambda = 50\%$ and $\lambda = 90\%$ are often the hardest for suboptimal instances demonstrates that number of decisions correlates with instance difficulty. Low λ means the solver doesn't have too many options to choose between, but medium and high λ's have just enough decisions to make it hard. At high ϕ and high λ the difficulty drops relative to medium λ, possibly because the solver has more learning options that it can take advantage of, so learning may no longer be as much of a deciding factor when generating schedules. However, it's unclear why these effects are not apparent in Figure 8.10.

Chapter 9

CONCLUSION

This chapter summarizes the research conducted in this thesis and the contributions made to the literature. It ends with some ideas for next steps in continuing this research.

9.1 Summary

This thesis set out to explore how learning affects the way projects are scheduled. The research conducted while addressing this problem contributes much to the literature, namely: introducing RCPSP+L extension, developing efficient solution approaches, and characterizing the effects of learning on the RCPSP.

9.1.1 RCPSP Learning Extension

While learning has been explored in project scheduling before, a literature review shows that it's never been integrated into the standard RCPSP scheduling problem (Section 2.3). This thesis proposes an RCPSP-based learning model called "RCPSP+L" that takes into account the effects of autonomous, intra-project learning, accounting for both learning frequency and learning intensity (Section 3.2). It introduces definitions for this model and explains how it relates to the base RCPSP. Constraint Programming formulations of the model are developed to verify it and enable others to utilize it (Chapter 5). An instance generator is developed to adapt the popular PSPLIB RCPSP data set library for this model, enabling easier research of the problem. The generator includes algorithms for valid student selection, teacher selection, and alternate post-learning durations (Section 7.2). Reduction techniques are devised to reduce the instance set (Chapter 4)

9.1.2 Learning Effects Characterization

Many experiments are conducted to characterize the problem (Chapter 8). The effects of learning on the RCPSP are shown to be extensive. Analysis of the model shows significant makespan reduction potential as high as 50% in extreme cases. Makespan reduction potentials are found to follow the learning curve concept. It's discovered that potential learning opportunities are not inherently worthwhile for every job when scheduling — there's a point of diminishing returns, and it's impossible to ever have every task utilize learning effects (Section 8.2).

9.1.3 Efficient Solution Approaches

Much like the RCPSP it generalizes, RCPSP+L is found to be a very difficult problem to solve. A survey is conducted on the effects of the various model parameters on the difficulty of the problem, which finds that while learning benefits can make an instance easier to solve, it can also make it harder by overwhelming the solver with too many options and decisions to make (Section 8.4). Four Constraint Programming formulations of the problem are developed in an attempt to solve the problem efficiently, including ones that utilize logical duration determination, dynamic duration calculation, multi-mode alternatives, and bi-objective incentives (Chapter 5). Many bounding techniques are created to tighten optimality gaps more intelligently than the automatic bounds provided by the solver. This includes four lower bounding model relaxations, one upper bounding model relaxation, and a destructive lower bounding method (Chapter 6). Hundreds of thousands of tests are conducted to determine the most efficient solution approaches (Chapter 8), including the best-performing model formulation (Section 8.1) and bounding techniques (Section 8.3). The best formulation and bounding technique are shown to be a significant improvement over the initial formulations. This testing regimen was extensive enough that it required the development of a custom batch runner for the CP Optimizer solver, and it stress tested the software enough that it unearthed bugs in the solver itself that the development team is using to improve the software for future users (Section 7.3).

9.2 Future Work

There are many exciting ways this research could be applied and further advanced, including further extending the RCPSP+L model, examining additional solution approaches, and using data from industry.

9.2.1 Advanced Learning Model Extensions

The learning extension explored in this thesis is simple: if a student task is scheduled after its teacher, an alternate duration is used; otherwise, there's no change. While powerful already, more advanced learning models could enable further opportunities for the solver, potentially increasing learning benefits and bringing the problem closer to the real world.

Currently, there's only one alternate duration d' for each student, with learning intensity λ set for each instance as a whole. A similarity-based learning extension could enable varying learning intensity depending on how similar two tasks are. Each learning relationship would have a unique λ, greatly rewarding the solver when extremely similar tasks experience learning and minimally rewarding when more dissimilar tasks take advantage of learning. This could encourage the solver to prioritize more effective learning opportunities when selecting optimal schedules. This extension would only require changes to the test instance generator, with no changes to the data file format or solver models.

Currently, each student can only have one teacher. A joint learning extension could imply that a job can only experience learning if multiple teachers occur in the past (i.e. job j can only learn if i_1, ..., i_k have finished). This extension would require minimal changes to the test instance generator and data file format. The solver models could be kept as-is if dummy jobs are used: for each student i, insert an artificial job i' with precedences (i_1, i'), ..., (i_k, i') for each of i's teachers.

An even more advanced learning extension could be developed that includes both of the above extensions simultaneously. For example, a student could experience differing levels of learning benefit from each of its teachers. Potentially, these learning effects could even compound if the student was scheduled after both teachers.

9.2.2 Additional Solution Approaches

Much of this thesis was spent examining ways to solve the RCPSP+L problem more efficiently. Chapter 5 describes various Constraint Programming (CP) formulations, and Chapter 6 describes techniques for further reducing the bounds produced by CP. Undoubtedly, even more formulations and bounding techniques could be developed, but there are also other categories of solution approaches that could be tried.

Mathematical Programming (MP), such as Integer Linear Programming and Mixed Integer Programming, is used extensively in the literature for RCPSP problems [34]. Dr. Qian Hu and Ying Liu from Nanjing University in China, mentioned in the Acknowledgements page, are researching MP approaches to a similar learning extension of RCPSP. Indeed, we were able to collaborate some for several months. Unfortunately, their learning extension was different enough that a direct comparison between our solvers was outside the scope of this paper. It would be interesting to perform this direct comparison in the future.

CP and MP seek to produce exact, optimal solutions, which can take a very long time. Indeed, much of Chapters 5, 6, and 8 centered on this issue. But in some cases, such as industrial settings, time to solve can be a higher priority that solution optimality — this is the strength of heuristic methods [1]. Heuristics can produce approximate solutions very quickly. They are widely used in RCPSP literature [19,31], and include methods like schedule generation schemes, X-pass, and metaheuristics (e.g. simulated annealing, tabu search, genetic algorithms [37]). Depending on the context, they could be a valuable complement to the CP approaches laid out in this paper.

Hybrid approaches have been shown to sometimes be more effective than a lone method [2,9]. A combination of multiple methods may be worth pursuing.

9.2.3 Industry Data

Though PSPLIB is well-renowned library of RCPSP data sets, research using theoretical data is fundamentally different from research using real world data. Applying the research in this thesis to a real project would give concrete evidence of the time and cost savings possible. Beyond interesting theoretical findings, it could provide a material benefit to a real company.

REFERENCES

[1] D. Bai, M. Tang, Z.-H. Zhang, and E. D. Santibanez-Gonzalez. Flow shop learning effect scheduling problem with release dates. *Omega*, 78:21–38, 2018.

[2] T. Berthold, S. Heinz, M. E. Lübbecke, R. H. Möhring, and J. Schulz. A constraint integer programming approach for resource-constrained project scheduling. In A. Lodi, M. Milano, and P. Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 313–317, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[3] D. Biskup. Single-machine scheduling with learning considerations. *European Journal of Operational Research*, 115(1):173 – 178, 1999.

[4] D. Biskup. A state-of-the-art review on scheduling with learning effects. *European Journal of Operational Research*, 188(2):315–329, 2008.

[5] J. Błażewicz, J. K. Lenstra, and A. R. Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete applied mathematics*, 5(1):11–24, 1983.

[6] J. L. Bresina, A. K. Jónsson, P. H. Morris, and K. Rajan. Activity planning for the mars exploration rovers. In *ICAPS*, pages 40–49, 2005.

[7] P. Brucker and S. Knust. Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research*, 149(2):302–313, 2003.

[8] M. Carlsson, M. Johansson, and J. Larson. Scheduling double round-robin tournaments with divisional play using constraint programming. *European Journal of Operational Research*, 259(3):1180 – 1190, 2017.

[9] A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. Mrspock—steps in developing an end-to-end space application. *Computational Intelligence*, 27(1):83–102, 2011.

[10] Y.-H. Chung and L.-I. Tong. Bi-criteria minimization for the permutation flowshop scheduling problem with machine-based learning effects. *Computers & Industrial Engineering*, 63(1):302–312, 2012.

[11] J. A. Cunningham. Management: Using the learning curve as a management tool: The learning curve can help in preparing cost reduction programs, pricing forecasts, and product development goals. *IEEE spectrum*, 17(6):45–48, 1980.

[12] H. Ebbinghaus. *Über das gedächtnis: untersuchungen zur experimentellen psychologie*. Duncker & Humblot, 1885. Early scientific discussion of human learning.

[13] M. H. Fazel Zarandi, H. Khorshidian, and M. Akbarpour Shirazi. A constraint programming model for the scheduling of jit cross-docking systems with preemption. *Journal of Intelligent Manufacturing*, 27(2):297–313, Apr 2016.

[14] I. P. Gent, R. W. Irving, D. F. Manlove, P. Prosser, and B. M. Smith. A constraint programming approach to the stable marriage problem. In T. Walsh, editor, *Principles and Practice of Constraint Programming — CP 2001*, pages 225–239, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[15] C. H. Glock, E. H. Grosse, M. Y. Jaber, and T. L. Smunt. Applications of learning curves in production and operations management: A systematic literature review. *Computers & Industrial Engineering*, 131:422–441, 2019.

[16] V. Goel, M. Slusky, W.-J. van Hoeve, K. Furman, and Y. Shao. Constraint programming for lng ship scheduling and inventory management. *European Journal of Operational Research*, 241(3):662 – 673, 2015.

[17] N. Hanakawa, S. Morisaki, and K.-i. Matsumoto. A learning curve based simulation model for software development. In *Proceedings of the 20th international conference on Software engineering*, pages 350–359. IEEE, 1998.

[18] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[19] S. Hartmann and R. Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127(2):394–407, 2000.

[20] A. Hill, A. Brickey, A. Newman, and M. Goycoolea. Hybrid optimization strategies for resource constrained project scheduling problems in underground mining. *under review*, 2019.

[21] A. Hill, E. Lalla-Ruiz, S. Voß, and M. Goycoolea. A multi-mode resource-constrained project scheduling reformulation for the waterway ship scheduling problem. *Journal of Scheduling*, 22(2):173–182, 2019.

[22] IBM Corporation. *CP Optimizer User's Manual*. IBM, 12.9.0 edition, 2019.

[23] M. Y. Jaber. Learning and forgetting models and their applications. *Handbook of industrial and systems engineering*, 30(1):30–127, 2006.

[24] M. Y. Jaber. *Learning curves: Theory, models, and applications.* CRC Press, 2011.

[25] E. Keachie and R. J. Fontana. Effects of learning on optimal lot size. *Management Science*, 13(2):B–102, 1966.

[26] J. E. Kelley and M. R. Walker. Critical-path planning and scheduling. In *Proceedings of the Eastern Joint Computer Conference, IRE-AIEE-ACM 1959*, 1959.

[27] J.-L. Kim. Genetic algorithm stopping criteria for optimization of construction resource scheduling problems. *Construction Management and Economics*, 31(1):3–19, 2013.

[28] R. Klein and A. Scholl. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. *European Journal of Operational Research*, 112(2):322–346, 1999.

[29] R. Kolisch. Psplib j60lb.sm. `https://www.om-db.wi.tum.de/psplib/files/j60lb.sm`, Feb 2019. Accessed: 2019-11-20.

[30] R. Kolisch and S. Hartmann. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In J. Węglarz, editor, *Project Scheduling: Recent Models, Algorithms and Applications*, pages 147–178. Springer US, Boston, MA, 1999.

[31] R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, 2006.

[32] R. Kolisch and A. Sprecher. PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, 1997.

[33] R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management science*, 41(10):1693–1703, 1995.

[34] P. Laborie. An update on the comparison of mip, cp and hybrid approaches for mixed resource allocation and scheduling. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 403–411. Springer, 2018.

[35] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. Ibm ilog cp optimizer for scheduling. *Constraints*, 23(2):210–250, Apr 2018.

[36] A. A. Lazarev, S. V. Bronnikov, A. Gerasimov, E. G. Musatova, A. Petrov, K. Ponomarev, M. Kharlamov, N. Khusnullin, and D. Yadrentsev. Mathematical modeling of the astronaut training scheduling. *Upravlenie Bol'shimi Sistemami*, 63:129–154, 2016.

[37] J.-K. Lee and Y.-D. Kim. Search heuristics for resource constrained project scheduling. *Journal of the Operational Research Society*, 47(5):678–689, 1996.

[38] O. Liess and P. Michelon. A constraint programming approach for the resource-constrained project scheduling problem. *Annals of Operations Research*, 157(1):25–36, Jan 2008.

[39] D. F. Manlove, G. O'Malley, P. Prosser, and C. Unsworth. A constraint programming approach to the hospitals / residents problem. In P. Van Hentenryck and L. Wolsey, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 155–170, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[40] G. Mejía, M. Sánchez, K. Niño, and P. Figueroa. A petri net based algorithm for the resource constrained project scheduling problem (rcpsp): A real life application in the animation and videogame industry. In *Proceedings of the 22th ICPR International Conference of Production Research*, 2013.

[41] G. F. Nemet. Beyond the learning curve: factors influencing cost reductions in photovoltaics. *Energy Policy*, 34(17):3218 – 3232, 2006.

[42] E. Néron, C. Artigues, P. Baptiste, J. Carlier, J. Damay, S. Demassey, and P. Laborie. *Lower Bounds for Resource Constrained Project Scheduling Problem*, pages 167–204. Springer US, Boston, MA, 2006.

[43] O. Polo Mejia, M.-C. Anselmet, C. Artigues, and P. Lopez. A new RCPSP variant for scheduling research activities in a nuclear laboratory. In *47th International Conference on Computers & Industrial Engineering (CIE47)*, page 8p., Lisbonne, Portugal, Oct. 2017.

[44] P. Prosser. Stable roommates and constraint programming. In H. Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 15–28, Cham, 2014. Springer International Publishing.

[45] T. Rihm, N. Trautmann, and A. Zimmermann. Mip formulations for an application of project scheduling in human resource management. *Flexible Services and Manufacturing Journal*, 30(4):609–639, Dec 2018.

[46] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of Constraint Programming.* Elsevier, 2006.

[47] C. Schwindt, J. Zimmermann, et al. *Handbook on Project Management and Scheduling Vol. 2.* Springer, 2015.

[48] S. A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.

[49] J. Ticktin and D. Junglas. How to fix errors when using opl.end() in main flow control loop. `https://stackoverflow.com/a/56341522/6402733`, May 2019.

[50] V. Van Peteghem and M. Vanhoucke. Influence of learning in resource-constrained project scheduling. *Computers & Industrial Engineering*, 87:569–579, 2015.

[51] P. Vilím, P. Laborie, and P. Shaw. Failure-directed search for constraint-based scheduling. In *CPAIOR 2015: Integration of AI and OR Techniques in Constraint Programming*, pages 437–453. Springer International Publishing, 2015.

[52] M. Wallace. Practical applications of constraint programming. *Constraints*, 1(1):139–168, Sep 1996.

[53] T. P. Wright. Factors affecting the cost of airplanes. *Journal of the Aeronautical Sciences*, 3(4):122–128, 1936.

[54] M.-C. Wu and S.-H. Sun. A project scheduling and staff assignment model considering learning effect. *The International Journal of Advanced Manufacturing Technology*, 28(11):1190–1195, May 2006.

[55] L. E. Yelle. The learning curve: Historical review and comprehensive survey. *Decision Sciences*, 10(2):302–328, 1979.

APPENDICES

APPENDIX A

INSTANCE GENERATOR CODE

This appendix contains the Python 3.7.3 code for `RCPSP+L_Instance_Generator.py`, which is used to generate all the RCPSP+L instance data files for this thesis. It includes options at the top where the user specifies the set of instances to be produced.

```
#!/usr/bin/env python 3.7.3
# coding: utf-8

# ***WARNING!*** This script will overwrite files in the export
↪  folder (i.e. filenames data file, test cases).

import os # For getting list of files in directory
import re # For sorting list of filenames
import random # For random selection
import math # For rounding


# # Parameters

# Percentage parameters
# - For 50% write 50, not 0.5
# - Calculations multiplied by these parameters are rounded up
# - Remember: `range()`'s `stop` parameter is exclusive
target_student_percent_options = [25, 50, 75, 100] #range(0, 101, 10)
↪  # Target percent of tasks to be students
duration_reduction_percent_options = [10, 50, 90] # Percent a
↪  student's duration is reduced by

# Random seeds to regenerate all test cases with
random_seed_options = range(3) # Remember: `range()`'s `stop`
↪  parameter is exclusive

# Test set
test_set = "j120" # Which PSPLIB set we're using (i.e. "j30", "j120")

# Import path
import_data_file_dir = "/[path redacted]/" + test_set + "/" + test_set
↪  + "rcp"

# Choose base cases
    ## Default: All of them
selected_base_cases = os.listdir(import_data_file_dir)
    ## Alternative: Use a prepopulated list
# selected_instance_bases = ["j12051", "j12031", "j12011", "j12016",
    "j12012", "j12047", "j1207", "j12038", "j12059", "j12039",
↪   "j12014", "j12060", "j12029", "j12035", "j12049", "j12041",
↪   "j1202", "j1203", "j12044", "j12025"]
↪
```

```python
# selected_base_cases = []
# for base in selected_instance_bases:
    # for instance_seed in range(1, 11): # Use all instance seeds for the base case
        # selected_base_cases.append(base + "_" + str(instance_seed) + ".RCP")

# Export path
export_dir_name = test_set + "-stu" + str(len(target_student_percent_options)).zfill(2) + "x-dur" + str(len(duration_reduction_percent_options)).zfill(2) + "x-ran" + str(len(random_seed_options)).zfill(2) + "x"
export_test_case_dir = "/[path redacted]/" + export_dir_name


# # Class Definitions

class Task(object):
    """Contain the attributes of a given task."""
    def __init__(self, task_ID=None, duration=None, resource_demands=None, successors=None, learning=[None, None]):
        """Create a Task object using the provided values."""
        self.task_ID = task_ID # int
        self.duration = duration # int
        self.resource_demands = resource_demands # list
        self.successors = successors # set
        self.teacher_ID = learning[0] # int
        self.alternate_duration = learning[1] # int

    def __repr__(self):
        """Present object's values as formatted in data file."""
        print_string = "< %r, %r, %r, {%r}, [%r, %r] >" % (self.task_ID, self.duration, self.resource_demands, self.successors, self.teacher_ID, self.alternate_duration)

        # Ensure set of successors prints as such, instead of as list
        # Can't just be passed a set initially because we want to maintain sort order
        print_string = print_string.replace("{[", "{")
        print_string = print_string.replace("]}", "}")

        return print_string

class Project(object):
    """Contain the attributes and tasks of a project."""
    def __init__(self, original_filename=None, nbTasks=None, nbRsrcs=None, capacities=None, task_list=None):
        """Create a Project object using the provided values."""
        self.original_filename = original_filename # string
        self.nbTasks = nbTasks # int
        self.nbRsrcs = nbRsrcs # int
        self.capacities = capacities # list of ints
        self.task_list = task_list # list of Task objects

    def __repr__(self):
        """Present object's values as formatted in test case file. Strictly for readability purposes, not for import."""
        return build_data_file(project)


# # Function Definitions

# ### OS Functions

def natural_sort_key(s, _nsre=re.compile('([0-9]+)')):
    """Key for sorting algorithms to sort names the way a human reads it, instead of in ASCII order.
       Example: ['J301_1.RCP', 'J3010_1.RCP', 'J3010_10.RCP', ...]
       Source: https://stackoverflow.com/a/16090640"""
    return [int(text) if text.isdigit() else text.lower() for text in _nsre.split(s)]


# ### Student Functions

def precedence_exists(predecessor, successor, task_list):
    """Check for a direct or indirect precedence relationship and return True/False whether one exists."""
    if (successor.task_ID in predecessor.successors):
        return True
    else:
        number_of_branches = len(predecessor.successors)
        branch_checks = [False] * number_of_branches
        for branch, successor_task_ID in zip(range(number_of_branches), predecessor.successors):
            branch_checks[branch] = precedence_exists(task_list[successor_task_ID-1], successor, task_list)
        return any(branch_checks)

def calculate_alternate_duration(student_task, duration_reduction_percent):
    """Calculate the alternate duration of a task."""
    # Set alternate duration (round up)
        # Use `ceil`; `floor` results in no learning occuring for single digit numbers at low duration_reduction_percent values
        # Otherwise, this can mark far too many tasks as invalid students to achieve good results
    alternate_duration = student_task.duration - math.ceil(student_task.duration * (duration_reduction_percent/100))

    # Ensure tasks take at least one period (subtracting `ceil` from duration can result in things being rounded to zero)
    alternate_duration = alternate_duration if alternate_duration >= 1 else 1

    return alternate_duration

def student_is_valid(student_task, duration_reduction_percent):
    """Return whether a task is valid as a student."""
    # Check for original duration of 1 (because it's impossible to learn and become faster)
    # Exclude empty tasks with duration of 0
    if (student_task.duration <= 1):
        return False

    # Check for alternate duration equal to its original duration (because it showed no learning occurred)
    if (student_task.duration == calculate_alternate_duration(student_task, duration_reduction_percent)):
        return False

    # If it didn't fail any validity tests, return that it's valid
    return True


# ### Data File Functions

def build_data_file(project, random_seed=-1, target_student_percent=-1, target_student_count=-1,
                    actual_student_count=-1, duration_reduction_percent=-1, instance="", test_case_filename=""):
    """Build and return string of test case data file.
Defaults to -1 for unset parameters."""
    # Calculate actual percentage of tasks that are students
    actual_student_percent = (actual_student_count/(len(project.task_list)-2)) * 100 # -2 to ignore empty start and end tasks

    # Beginning of data file
    intro_string = """// Converted by Jordan Ticktin
// %s from PSPLIB converted to learning-modified .dat OPL format

// Test Case Parameters:
// - Percentage of tasks that are students = Target: %d%% (%d students), Actual: %.2f%% (%d students)
// - Percentage of duration reduction = %d%%
// - Random generator seed = %d
TargetStudentPercent = %d;
TargetStudentCount = %d;
ActualStudentPercent = %.2f;
ActualStudentCount = %d;
DurationReductionPercent = %d;
Seed = %d;
Instance = "%s";
TestCaseFilename = "%s";

NbTasks = %d;
NbRsrcs = %d;

Capacity = %s;

Tasks = {
// < Task ID, Duration, [Resource Demands], {Successor(s) (optional)}, [Teacher, Alternate Duration] >
  """ % (project.original_filename,
         target_student_percent, target_student_count, actual_student_percent, actual_student_count, duration_reduction_percent, random_seed,
         target_student_percent, target_student_count, actual_student_percent, actual_student_count, duration_reduction_percent, random_seed, instance, test_case_filename,
         project.nbTasks, project.nbRsrcs, project.capacities)

    # List of tasks
    TASK_SEPARATOR = """,
  """
    tasks_string = TASK_SEPARATOR.join([str(task) for task in project.task_list])

    # End of data file
    outro_string = """
};
```

```python
"""

    # Combine parts and return
    return "".join([intro_string, tasks_string, outro_string])


# # Generate Test Cases

# ### Build Test Cases and Write to Folder

# List of test cases to write to filenames data file
test_case_filenames_list = []

# For every base case selected by the user
for original_filename in sorted(selected_base_cases,
↪   key=natural_sort_key): # Filenames are missing padded zeroes, use
↪   a natural sort
    # Store data in Project object
    project = Project()
    project.original_filename = original_filename

    # Read in data from import file
    with open(import_data_file_dir + "/" + original_filename, "r") as
     ↪  import_file:
        import_data_file_lines = import_file.readlines()

    # Import number of tasks and resources
    first_line_elements = import_data_file_lines[0].split()
    project.nbTasks = int(first_line_elements[0])
    project.nbRsrcs = int(first_line_elements[1])

    # Import resource capacities
    capacities = import_data_file_lines[1].split() # Separate numbers
     ↪  on line into individual list elements
    capacities = list(map(int, capacities)) # Convert all elements
     ↪  from string to int
    project.capacities = capacities

    # Import tasks
    project.task_list = []
    task_lines = import_data_file_lines[2:]
    for line, task_number in zip(task_lines, range(1,
     ↪  len(task_lines)+1)):
        components = line.split() # Separate numbers on line into
         ↪  individual list elements
        components = list(map(int, components)) # Convert all
         ↪  elements from string to int
        # task_ID, duration, resource_demands, successors,
         ↪  [teacher_ID, alternate_duration]
        project.task_list.append(Task(task_number, components[0],
         ↪  components[1:5], components[6:], [0, 0]))

    # For every random seed
    for seed in random_seed_options:
        # For every target % of students
        for target_student_percent in target_student_percent_options:
            # Prevent impossible values (ensures file names match
             ↪  values used in calculations)
            target_student_percent = target_student_percent if
             ↪  target_student_percent <= 100 else 100 # Prevent
             ↪  impossibly many students
            target_student_percent = target_student_percent if
             ↪  target_student_percent >= 0 else 0 # Prevent negative
             ↪  numbers

            # For every target % of duration reduction
            for duration_reduction_percent in duration_reduction_percent_options:
                # Prevent impossible values (ensures file names match values used in calculations)
             duration_reduction_percent = duration_reduction_percent if duration_reduction_percent <= 100 else 100 # Prevent impossibly many students
             duration_reduction_percent = duration_reduction_percent if duration_reduction_percent >= 0 else 0 # Prevent negative numbers

             # Calculate number of students
             assert(len(project.task_list) >= 4) # Must have at least 2 tasks (excluding empty start/end tasks) to have students
             target_student_count = math.ceil((len(project.task_list)-2) * (target_student_percent/100)) # Set target number of students (round up) (-2 to ignore empty start/end tasks)
             target_student_count = target_student_count if target_student_count >= 0 else 0 # Prevent negative numbers after subtracting

             # Select students
             random.seed(a=seed) # Seed random generator to ensure consistent results
             accepted_student_IDs = [] # Tasks accepted as students
             tested_student_IDs = [] # Tasks already tested (both accepted and rejected)
             while ((len(accepted_student_IDs) < target_student_count) # Stop searching once desired number of students are found
                    and (len(tested_student_IDs) < len(project.task_list)-2)): # Stop searching once every task has been tested
                 # Build list of student IDs which have not been tested yet
                 untested_student_IDs = [student_ID for student_ID in range(2, len(project.task_list)) if student_ID not in tested_student_IDs]

                 # Select a new candidate from the list of untested student IDs
                 candidate_ID = random.choice(untested_student_IDs)
                 candidate_task = project.task_list[candidate_ID-1]

                 # If candidate is valid, add it to list of students
                 if (student_is_valid(candidate_task, duration_reduction_percent)):
                     accepted_student_IDs.append(candidate_ID)

                 # Don't waste time investigating this task again
                 tested_student_IDs.append(candidate_ID)

             # Select teachers and set teacher/alternate duration values
             random.seed(a=seed) # Seed random generator again to ensure it starts by picking the same teachers as other student % test cases
             for student_ID in accepted_student_IDs:
                 # Identify student task
                 student_task = project.task_list[student_ID-1]

                 # Select teacher
                 teacher_ID = None
                 while (teacher_ID is None): # Keep searching for a teacher until one is found
                     # Select random teacher (exclude empty start/end tasks)
                     teacher_ID = random.choice(range(2, len(project.task_list)))

                     # If teacher is an illegal choice, reset teacher and pick again
                     if ((student_ID == teacher_ID) # Verify teacher and student are different tasks
                         or precedence_exists(project.task_list[teacher_ID-1], student_task, project.task_list) # Verify teacher isn't a precedence for the student
                         or precedence_exists(student_task, project.task_list[teacher_ID-1], project.task_list)): # Verify student isn't a precedence for the teacher
                         teacher_ID = None

                 # Calculate alternate duration
                 alternate_duration = calculate_alternate_duration(student_task, duration_reduction_percent)

                 # Set values in `project` for this student
                 student_task.teacher_ID = teacher_ID # Set teacher
                 student_task.alternate_duration = alternate_duration # Set alternate duration

             # Create filename
             base_filename = original_filename.lower().split(".")[0] # Extract the data file name without the old extension
             test_case_filename = "TestCase-%s-stu%03d-dur%03d-ran%02d.dat" % (base_filename, target_student_percent, duration_reduction_percent, seed)

             # Build converted data file
             export_data_file_string = build_data_file(project, seed, target_student_percent, target_student_count, len(accepted_student_IDs), duration_reduction_percent, base_filename, test_case_filename)

             # Write data file to the appropriate folder
```

```python
                test_case_file_path = export_test_case_dir + "/" + test_set + "-Test_Cases/" + test_case_filename
                with open(test_case_file_path, "w") as export_file: # WARNING: This will overwrite any existing file with this name
                   export_file.write(export_data_file_string)

                # Add this test case's filename to list of filenames in this folder
                test_case_filenames_list.append(test_case_filename)

                # Reset teachers/alternate durations in project
                for task in project.task_list:
                    task.teacher_ID = 0
                    task.alternate_duration = 0


# ### Write List of Test Case Filenames to Folder

# Write list of filenames in this folder to a data file that can be imported by OPL
filenames_data_file = export_test_case_dir + "/Filenames-" + export_dir_name + ".dat"
test_case_filenames_list_string = '["' + '", "'.join(test_case_filenames_list) + '"]'
with open(filenames_data_file, "w") as export_file: # WARNING: This will overwrite any existing file with this name
    # Write test case generation parameters
    export_file.write("// Test Case Generation Parameters")
    export_file.write("\n// - Test Set: " + test_set)
    export_file.write("\n// - Number of Base Cases: " + str(len(selected_base_cases)))
    export_file.write("\n// - Learning Frequencies: " + str(len(target_student_percent_options)) + " - " + str(list(target_student_percent_options)))
    export_file.write("\n// - Learning Intensities: " + str(len(duration_reduction_percent_options)) + " - " + str(list(duration_reduction_percent_options)))
    export_file.write("\n// - Random Seeds: " + str(len(random_seed_options)) + " - " + str(list(random_seed_options)))
    export_file.write("\n\n")

    # Write count and list of test case filenames
    export_file.write("NumberOfFiles = " + str(len(test_case_filenames_list)) + ";")
    export_file.write("\nTestCaseFilenames = " + test_case_filenames_list_string + ";")
```

APPENDIX B

INSTANCE DATA

B.1 Example Instance Data File

Included below are the contents of `TestCase-j3029_7-stu060-dur030-ran01.dat`, one of the many instance data files used in this thesis. This example is a version of PSPLIB's `J3029_7.RCP` RCPSP instance that's adapted to incorporate the learning extension. This RCPSP+L instance is generated using the parameter values $\phi = 60$, $\lambda = 30$, and $a = 1$. It's written in the OPL `.dat` data format.

```
// Converted by Jordan Ticktin
// J3029_7.RCP from PSPLIB converted to learning-modified .dat OPL
↪  format

// Test Case Parameters:
// - Percentage of tasks that are students = Target: 60% (18
↪  students), Actual: 60.00% (18 students)
// - Percentage of duration reduction = 30%
// - Random generator seed = 1
TargetStudentPercent = 60;
TargetStudentCount = 18;
ActualStudentPercent = 60.00;
ActualStudentCount = 18;
DurationReductionPercent = 30;
Seed = 1;
Instance = "j3029_7";
TestCaseFilename = "TestCase-j3029_7-stu060-dur030-ran01.dat";

NbTasks = 32;
NbRsrcs = 4;

Capacity = [15, 15, 18, 17];

Tasks = {
// < Task ID, Duration, [Resource Demands], {Successor(s)
↪  (optional)}, [Teacher, Alternate Duration] >
  < 1, 0, [0, 0, 0, 0], {2, 3, 4}, [0, 0] >,
  < 2, 7, [2, 1, 5, 5], {6, 11, 18}, [16, 4] >,
  < 3, 4, [5, 1, 4, 10], {5}, [0, 0] >,
  < 4, 4, [7, 6, 1, 4], {8, 9, 16}, [5, 2] >,
  < 5, 2, [4, 6, 5, 5], {7, 13, 19}, [0, 0] >,
  < 6, 3, [2, 3, 2, 5], {12}, [20, 2] >,
  < 7, 10, [8, 4, 2, 7], {15, 20}, [2, 7] >,
  < 8, 3, [9, 9, 3, 1], {19, 20, 23}, [26, 2] >,
  < 9, 3, [3, 7, 5, 5], {10, 15}, [0, 0] >,
  < 10, 3, [7, 2, 7, 2], {11, 14}, [0, 0] >,
  < 11, 10, [3, 3, 10, 9], {12, 21}, [26, 7] >,
  < 12, 3, [3, 4, 10, 5], {13, 24, 30}, [17, 2] >,
  < 13, 8, [7, 4, 3, 8], {22}, [0, 0] >,
  < 14, 5, [10, 6, 3, 8], {17, 26}, [0, 0] >,
  < 15, 7, [5, 5, 4, 2], {25, 28}, [2, 4] >,
  < 16, 5, [8, 2, 4, 7], {20}, [2, 3] >,
  < 17, 4, [1, 3, 9, 6], {27, 28, 29}, [0, 0] >,
  < 18, 1, [5, 7, 8, 10], {22, 23}, [0, 0] >,
  < 19, 5, [4, 4, 6, 10], {21, 25}, [0, 0] >,
  < 20, 9, [8, 8, 9, 10], {21, 24, 27}, [14, 6] >,
  < 21, 6, [3, 9, 7, 1], {31}, [29, 4] >,
  < 22, 5, [1, 1, 7, 6], {31}, [30, 3] >,
  < 23, 10, [2, 2, 8, 9], {24, 25, 29}, [0, 0] >,
  < 24, 2, [7, 3, 5, 5], {28}, [15, 1] >,
  < 25, 9, [7, 10, 7, 8], {27}, [0, 0] >,
  < 26, 4, [5, 7, 8, 3], {29, 30}, [0, 0] >,
  < 27, 2, [10, 9, 5, 3], {30}, [12, 1] >,
  < 28, 10, [9, 4, 1, 1], {31}, [26, 7] >,
  < 29, 8, [2, 10, 10, 1], {32}, [5, 5] >,
  < 30, 2, [2, 9, 4, 3], {32}, [24, 1] >,
  < 31, 4, [2, 1, 2, 2], {32}, [27, 2] >,
  < 32, 0, [0, 0, 0, 0], {}, [0, 0] >
};
```

B.2 Example List of Filenames

The contents of `Filenames.dat` is included below. Every time the instance generator script is run, it outputs a file like the one below that lists all of the instances produced in this set. The batch script operates from this list when running a test regimen. The file is written in the OPL `.dat` data format.

```
NumberOfFiles = 3;

TestCaseFilenames = ["TestCase-j12012_5-stu100-dur010-ran02.dat",
↪   "TestCase-j1204_9-stu100-dur090-ran01.dat",
↪   "TestCase-j1201_3-stu100-dur090-ran00.dat"];
```

APPENDIX C

EXAMPLE OPL MODEL CODE

Below is the code for `F2_RCPSP+L_Dynamic.mod`, one of the many models used in this thesis. It provides the Constraint Programming formulation of the F_2 model from Chapter 5, including the instance data to use, decision variables, objective, constraints, and post-processing steps used to validate the produced solution. The model itself is written in OPL, and the post-processing code is written in ILOG Script (similar to JavaScript).

```
using CP;

// ----------------------------------------
// DATA
// ----------------------------------------

int TargetStudentPercent = ...;
int TargetStudentCount = ...;
float ActualStudentPercent = ...;
int ActualStudentCount = ...;
int DurationReductionPercent = ...;
int Seed = ...;
string Instance = ...;
string TestCaseFilename = ...;

int NbTasks = ...;
int NbRsrcs = ...;
range RsrcIds = 0..NbRsrcs-1;
int Capacity[r in RsrcIds] = ...;

tuple Task {
  key int id;
  int     pt;
  int     dmds[RsrcIds];
  {int}   succs;
  int     learning[0..1];
}

{Task} Tasks = ...;

// Map tasks to an ID
  // Initialization code provided by Daniel Junglas (IBM)
Task id2Task[id in 1..NbTasks] = first({ t | t in Tasks : t.id == id
↪  });

// Makespan UB constraint for DLB
range HorizonIds = 0..0;
int Horizon[h in HorizonIds] = ...;

```

```
// ----------------------------------------
// DECISION VARIABLES
// ----------------------------------------

// Tasks' schedules
dvar interval TaskItvs[t in Tasks];

// Resource consumption
cumulFunction rsrcUsage[r in RsrcIds] = sum(t in Tasks: t.dmds[r]>0)
 ↪  pulse(TaskItvs[t], t.dmds[r]);

// ----------------------------------------
// OBJECTIVE
// ----------------------------------------

minimize max(t in Tasks) endOf(TaskItvs[t]);

// ----------------------------------------
// CONSTRAINTS
// ----------------------------------------

subject to {
  // Precedences
  forall (Pred in Tasks, idSucc in Pred.succs)
    endBeforeStart(TaskItvs[Pred], TaskItvs[id2Task[idSucc]]);
  // Resource capacities
  forall (r in RsrcIds)
    rsrcUsage[r] <= Capacity[r];
  // Learning (new single-mode formulation)
  forall (t in Tasks) {
    // Learning allowed
    if (t.learning[0] != 0) {
     lengthOf(TaskItvs[t]) == t.pt - (t.pt-t.learning[1]) *
         (endOf(TaskItvs[id2Task[t.learning[0]]]) <=
     ↪  startOf(TaskItvs[t]));
    }
    // No learning allowed
    else {
      // Teacher doesn't exist -> use original duration
      lengthOf(TaskItvs[t]) == t.pt;
    }
  }
}

// ----------------------------------------
// POST-PROCESSING
// ----------------------------------------

execute {
  // ---------- Validate solution ----------
  // Check precedences, learning, and durations
  for (var t in Tasks) {
    // Check precedences
      // Verify all the task's successors start after the task ends
    for (var idSucc in t.succs) {
      if (TaskItvs[t].end > TaskItvs[id2Task[idSucc]].start) {
        writeln("FAIL Predecessor | Predecessor ID: " + t.id + ",
         ↪  Successor ID: " + idSucc);
        writeln(); // Flush buffer
        fail(); // Stop execution (will also stop the batch runner)
      }
    }

    // Check learning ability
      // Tasks without teacher don't have an alternate duration, and vice-versa
    if ((t.learning[0] == 0 && t.learning[1] != 0)
        || (t.learning[0] != 0 && t.learning[1] == 0)) {
        writeln("FAIL Learning Incorrectly Allowed | Teacher ID: " + t.learning[0]
                + ", Alternate Duration: " + t.learning[1]);
        writeln(); // Flush buffer
        fail(); // Stop execution (will also stop the batch runner)
    }

    // Check learning duration
      // If no teacher
    if (t.learning[0] == 0) {
      // Verify that non-learning tasks use the original duration
      if (TaskItvs[t].size != t.pt) {
        writeln("FAIL Duration | Task ID: " + t.id + ", Duration Used: " + TaskItvs[t].size
                + ", Original Duration: " + t.pt + ", Alternate Duration: " + t.learning[1]
                + ", Teacher ID: " + t.learning[0]);
        writeln(); // Flush buffer
        fail(); // Stop execution (will also stop the batch runner)
      }
    }
      // If teacher available
    else if (t.learning[0] > 0) {
      // Verify that non-learning tasks use the original duration
      if (TaskItvs[id2Task[t.learning[0]]].end > TaskItvs[t].start && TaskItvs[t].size != t.pt) {
        writeln("FAIL Duration | Task ID: " + t.id + ", Duration Used: " + TaskItvs[t].size
                + ", Original Duration: " + t.pt + ", Alternate Duration: " + t.learning[1]
                + ", Teacher ID: " + t.learning[0]);
        writeln(); // Flush buffer
        fail(); // Stop execution (will also stop the batch runner)
      }
      // Verify that learning tasks use the alternate duration
      else if (TaskItvs[id2Task[t.learning[0]]].end <= TaskItvs[t].start && TaskItvs[t].size != t.learning[1]) {
        writeln("FAIL Duration | Task ID: " + t.id + ", Duration Used: " + TaskItvs[t].size
                + ", Original Duration: " + t.pt + ", Alternate Duration: " + t.learning[1]
                + ", Teacher ID: " + t.learning[0]);
        writeln(); // Flush buffer
        fail(); // Stop execution (will also stop the batch runner)
      }
    }
      // If teacher is a negative number (invalid)
    else {
      writeln("FAIL Teacher | Task ID: " + t.id + ", Teacher ID: " + t.learning[0]);
      writeln(); // Flush buffer
      fail(); // Stop execution (will also stop the batch runner)
    }
  }
```

```

  // Check resources
    // For every time period
  for (var p = 1; p <= cp.getObjValue(); p++) {
    // For every resource
    for (var r in RsrcIds) {
      // Track resources consumed this period
      var rescUtilized = 0;

      // Add in each task's resource consumption
      for (var t in Tasks) {
        // If doesn't happen during this period, skip it
        if (TaskItvs[t].start >= p || TaskItvs[t].end < p) {
          continue;
        }
        // If it is during this time period, add the resources it
         ↪  uses to the total
        rescUtilized += t.dmds[r];
      }

      // Verify resource consumption didn't exceed the capacity
      if (rescUtilized > Capacity[r]) {
        writeln("FAIL Resources | Period: " + p + ", Resource: " + r
                + ", Resource Excess: " + (rescUtilized -
                 ↪  Capacity[r]));
        writeln(); // Flush buffer
        fail(); // Stop execution (will also stop the batch runner))
      }
    }
  }
}
```

APPENDIX D

BATCH SCRIPT CODE

Code is included below for `BatchScript_Bounding.mod`. This batch script runs all of the provided instances using the models specified in a list at the top and the DLB method; it's intended for all of the bounding techniques from Chapter 6. A separate (but similar) batch script is used when running just one model at a time to make things easier to manage. The code is written in ILOG Script (related to JavaScript) and stored in a `.mod` file.

```
using CP;

// Import test case filename data
int NumberOfFiles = ...;
range FileIDs = 0..NumberOfFiles-1;
string TestCaseFilenames[FileIDs] = ...;

// Run all test instances using this model
main {

  // ------------------------------
  // PARAMETERS
  // ------------------------------

  // Parameters
  var workers = 4; // Number of computer cores used
  var timeMode = "ElapsedTime"; // How time is measured. Allowed values: ["ElapsedTime", "CPUTime"]
  var logVerbosity = "Quiet"; // Control engine log output to reduce CPU/memory usage ["Quiet", "Terse", "Normal", "Verbose"]
//  var logPeriod = 100000; // Manually control engine log output frequency (only applies if logVerbosity isn't "Quiet")

  // Test types (excluding DLB)
  var testModelNameOptions = new Array("LB1_RCPSP-", "LB2_RCSP+L", "LB3_PSP+L", "LB4_PSP-", "UB1_RCPSP+"); // Model filenames/solution folder names
  var timeLimit            = 60; // In seconds
//  var lbIndexToUse         =         0; // Which test type's LB to use as horizon in DLB (not currently used)
  var TimeLimitDLB         = 60; // In seconds

  // Test case directory (include trailing slash)
  var testCaseDirectory = "C:\\[path redacted]\\"; // Thesis PC
//  var testCaseDirectory = "/[path redacted]/"; // Jordan's computer

```

```
  // Results directory (include trailing slash)
  var resultsDirectory = "C:\\[path redacted]\\" // Thesis PC
//  var resultsDirectory = "/Users/[path redacted]/"; // Jordan's computer

  // Platform directory separator
  var dirSep = "\\"; // PC
//  var dirSep = "/"; // Mac

  // ------------------------------
  // SET UP OUTPUT FILE
  // ------------------------------

  // Function to pad strings (e.g. pad a number with 0s)
  function pad(original, width, pad) {
    // Adapted from https://stackoverflow.com/a/10073788/6402733
    pad = pad || "0"; // "0" is default pad
    original = original + "";
    return original.length >= width ? original : new Array(width - original.length + 1).join(pad) + original;
  }

  // Set output file for results
  var systemDateTime = new Date();
  var formattedDateTime = "d" + pad(systemDateTime.getMonth()+1, 2) + "-" + pad(systemDateTime.getDate(), 2)
                          + "_t" + pad(systemDateTime.getHours(), 2) + "-" + pad(systemDateTime.getMinutes(), 2);
  var resultsFilename = "Results(" + formattedDateTime + ")(" + thisOplModel.NumberOfFiles + "-Tests).csv";
  var resultsPath = resultsDirectory + resultsFilename;
  writeln("Results written to: " + resultsPath);

  // Start output file with headers
  var resultsFile = new IloOplOutputFile(resultsPath);
  resultsFile.writeln( // Write headers for CSV file
   "Test Case"
   + ",Test Type"
   + ",Instance"
   + ",# of Tasks"
   + ",# of Resources"
   + ",Resource Capacities"
   + ",Test Case Generator - Target % of Students Allowed"
   + ",Test Case Generator - Target # of Students Allowed"
   + ",Test Case Generator - Actual % of Students Allowed"
   + ",Test Case Generator - Actual # of Students Allowed"
   + ",% of Task Duration Reduction"
   + ",Random Seed"
//   + ",CP Optimizer Version"
   + ",Makespan"
   + ",Gap - Absolute Value"
   + ",Gap - % of Current Makespan"
   + ",Solver - % of Tasks Utilizing Learning"
   + ",Solver - # of Tasks Utilizing Learning"
   + ",Lower Bound"
   + ",Number of Solutions"
   + ",Status"
   + ",Presolve Time (sec)"
   + ",Extraction Time (sec)"
   + ",Cumulative Solve Time (sec)"
   + ",Final Run's Solve Time (sec)"
   + ",Time Limit (sec)");

  // Determine CP Optimizer version number
    // Not exposed in the IloCP scripting class, so pull it from the IloCP Java class
    // Commented out for now because calling Java doesn't work from the oplrun CLI
//  var cpo = IloOplCallJava("ilog.cp.IloCP", "<init>", "()Lilog/cp/IloCP");
//  var cpVersion = cpo.getVersion();
//  cpo.end();

  // ------------------------------
  // SOLVE EACH TEST CASE
  // ------------------------------

  // Solve each test case in the list
  for (var testNumber = 0; testNumber < thisOplModel.NumberOfFiles; testNumber++) {

    // ------------------------------
    // SET UP TESTING FOR INSTANCE
    // ------------------------------

    // Create new CP setup
    var cp = new IloCP();

    // Set CP parameters
    cp.param.Workers = workers; // Number of computer cores used
    cp.param.TimeMode = timeMode; // How to report time
    cp.param.LogVerbosity = logVerbosity; // How much to write to the engine log
//    cp.param.LogPeriod = logPeriod; // Manual control of engine log output frequency

    // Set next test case as data source
    var testCaseFilename = thisOplModel.TestCaseFilenames[testNumber];
    var testCasePath = testCaseDirectory + testCaseFilename;
    var data = new IloOplDataSource(testCasePath);

    // Report test case name to script log for progress visibility
    writeln(testNumber+1 + ") Solving " + testCaseFilename + "...");

    // ------------------------------
    // SOLVE EACH TEST TYPE
    // ------------------------------

    // Create storage for optimal solution bounds
    var boundResults = new Array(testModelNameOptions.length); // LB or UB result of each test
    for (var i=0; i < boundResults.length; i++) { // Initialize all values to -1
      boundResults[i] = -1;
    }

    // Solve each test type
    for (var testModelNameIndex=0; testModelNameIndex < testModelNameOptions.length; testModelNameIndex++) {
      // Get name of test type being run
      var testModelName = testModelNameOptions[testModelNameIndex];
      writeln(testModelName + "...");

      // Set time limit
      cp.param.TimeLimit = timeLimit;

      // Generate model
      var source = new IloOplModelSource(testModelName + ".mod");
      var def = new IloOplModelDefinition(source);
      var opl = new IloOplModel(def,cp);
      opl.addDataSource(data);
      opl.generate();

      // Set file to write solution to
      var solutionsDirectory = resultsDirectory + "Solutions" + dirSep
       ↪  + "Solutions-" + testModelName + dirSep;
      var solutionPath = solutionsDirectory + "Solution-" +
       ↪  testModelName + "-" + testCaseFilename;
      var solutionFile = new IloOplOutputFile(solutionPath);

      // Write initial parameters to results file
      resultsFile.write(opl.TestCaseFilename); // Test case filename
      resultsFile.write("," + testModelName); // Test type
      resultsFile.write("," + opl.Instance); // Test instance derived
       ↪  from
      resultsFile.write("," + (opl.NbTasks-2)); // Number of tasks
       ↪  (excluding empty start/end tasks)
      resultsFile.write("," + opl.NbRsrcs); // Number of resources
      resultsFile.write("," + opl.Capacity); // Number of resources
      resultsFile.write("," + opl.TargetStudentPercent); // Target
       ↪  percent of students
      resultsFile.write("," + opl.TargetStudentCount); // Target
       ↪  number of students
      resultsFile.write("," + opl.ActualStudentPercent); // Actual
       ↪  percent of students
      resultsFile.write("," + opl.ActualStudentCount); // Actual
       ↪  number of students
      resultsFile.write("," + opl.DurationReductionPercent); //
       ↪  Percent of duration reduction
      resultsFile.write("," + opl.Seed); // Random seed
//      resultsFile.write("," + cpVersion); // CP Optimizer solver
 ↪  version

      // Solve model
      if (cp.solve()) {
        // Run solution validation code in model file's
         ↪  post-processing execute statement
          // If a solution is invalid, the entire batch runner will
           ↪  fail with an error at that test and line
        opl.postProcess();

        // Calculate solve-dependent learning utilization
        var utilizedLearningCount = 0;
        for (var t in opl.Tasks) {
          if (t.learning[0] > 0 && opl.TaskItvs[t].size ==
           ↪  t.learning[1]) {
            utilizedLearningCount++;
          }
        }
          // Calculate % of all tasks that learned (-2 to ignore
           ↪  empty start and end tasks)
        var utilizedLearningPercent =
         ↪  (utilizedLearningCount/(opl.Tasks.size-2))*100;
```

```
          // Round to max of 2 decimal places (borrowed from https://stackoverflow.com/a/18358056/6402733)
        utilizedLearningPercent = +(Math.round(utilizedLearningPercent + "e+2")  + "e-2");

        // Calculate gap
        var gapAbsolute = (cp.getObjValue()-cp.getObjBound());
        var gapPercent = (gapAbsolute/cp.getObjValue())*100;
          // Round to max of 2 decimal places (borrowed from https://stackoverflow.com/a/18358056/6402733)
        gapPercent = +(Math.round(gapPercent + "e+2")  + "e-2");

        // Report some solve-dependent results to script log for visibility
        writeln(" - Makespan = " + cp.getObjValue() + " (LB - " + cp.getObjBound() + ", Gap - " + gapPercent + "%)"); // Makespan
        writeln(" - % of Tasks Utilizing Learning = " + utilizedLearningPercent + "%"); // % of utilized students

        // Write solve-dependent results to results file
        resultsFile.write("," + cp.getObjValue()); // Makespan
        resultsFile.write("," + gapAbsolute); // Gap #
        resultsFile.write("," + gapPercent); // Gap %
        resultsFile.write("," + utilizedLearningPercent); // % of utilized students
        resultsFile.write("," + utilizedLearningCount); // Number of utilized students

        // Write results to solution file
        solutionFile.writeln("// Results");
        solutionFile.writeln("Makespan = " + cp.getObjValue()); // Makespan
        solutionFile.writeln("Lower Bound = " + cp.getObjBound()); // Lower bound
        solutionFile.writeln("Gap % = " + gapPercent + "%"); // Gap %
        solutionFile.writeln("Number of Solutions = " + cp.info.NumberOfSolutions); // Number of solutions found
        solutionFile.writeln("Status = " + cp.status); // Solver status
        solutionFile.writeln("Utilized Learning Percent = " + utilizedLearningPercent + "%"); // % of utilized students
        solutionFile.writeln("Utilized Learning Count = " + utilizedLearningCount); // Number of utilized students
        solutionFile.writeln("Solve Time (sec) = " + cp.info.SolveTime); // Solve Time
        solutionFile.writeln("Time Limit (sec) = " + cp.param.TimeLimit); // Time limit
//        solutionFile.writeln("CP Optimizer Solver Version = " + cpVersion); // CP Optimizer solver version
        solutionFile.writeln();
        // Write solution to solution file
        solutionFile.writeln("// Solution");
        solutionFile.writeln("Task Name | Start Period | Duration"); // Solution headers
        for (var t in opl.Tasks) {
          solutionFile.writeln("Task " + t.id + ";" + opl.TaskItvs[t].start + ";" + opl.TaskItvs[t].size);
        }
```

```
    solutionFile.writeln();
    // Write precedences to solution file for import into yEd
    solutionFile.writeln("// Precedences");
    solutionFile.writeln("Edge ID | Node 1 | Node 2"); // Print headers
    var edgeID = 0;
    for (var t in opl.Tasks) {
     for (var s in t.succs) {
      edgeID++;
      solutionFile.writeln(edgeID + ";" + t.id + ";" + s);
     }
    }
    solutionFile.writeln();
   }
   else {
    // Report some results to script log for visibility
    writeln(" - Makespan = UNDEFINED"); // Makespan

    // Leave empty columns in results file, one comma per column
     // (getObjValue() and others fail if called when no solution has been found)
    resultsFile.write(",,,,,");

    // Report failure to find solution in solution file
    solutionFile.writeln("No solution found.");
    solutionFile.writeln();
    // Write results to solution file
    solutionFile.writeln("// Results");
    solutionFile.writeln("Lower Bound = " + cp.getObjBound()); // Lower bound
    solutionFile.writeln("Number of Solutions = " + cp.info.NumberOfSolutions); // Number of solutions found
    solutionFile.writeln("Status = " + cp.status); // Solver status
    solutionFile.writeln("Solve Time (sec) = " + cp.info.SolveTime); // Solve Time
    solutionFile.writeln("Time Limit (sec) = " + cp.param.TimeLimit); // Time limit
//   solutionFile.writeln("CP Optimizer Solver Version = " + cpVersion); // CP Optimizer solver version
    solutionFile.writeln();
   }

   // Report some results to script log for visibility
   writeln(" - Status = " + cp.status); // Solver status
   writeln(" - Solve Time = " + cp.info.SolveTime + " sec"); // Solve time

   // Write values to results file
   resultsFile.write("," + cp.getObjBound()); // Lower bound
   resultsFile.write("," + cp.info.NumberOfSolutions); // Number of solutions found
   resultsFile.write("," + cp.status); // Solver status
   resultsFile.write("," + cp.info.PresolveTime); // Time spent on presolve (preprocessing phase that aims to reduce both number of variables and constraints using logical rules)
   resultsFile.write("," + cp.info.ExtractionTime); // Time spent on extraction
   resultsFile.write("," + cp.info.SolveTime); // Time spent solving all DLB bounds (moot for this model because it's not DLB)
   resultsFile.write("," + cp.info.SolveTime); // Time spent testing the final (only) bound
   resultsFile.write("," + cp.param.TimeLimit); // Time limit
   resultsFile.writeln(); // Flush buffer to file & start new line for next entry

   // Finish solution file
   solutionFile.writeln(opl.printExternalData()); // Write variable values
   solutionFile.close();

   // Save bound, if optimal
   if (cp.status == 2) { // Only if optimal
     boundResults[testModelNameIndex] = cp.getObjValue(); // Save at corresponding index
   }

   // End processes to prevent memory leaks
   opl.end();
   def.end();
   source.end();
 }

 // Output bound results to log
   // Print first value outside of loop to print comma separators correctly
 write("Bound Results: " + boundResults[0]);
 for (var i=1; i < boundResults.length; i++) { // Skip first value because it's already printed
   write(", " + boundResults[i]);
 }
 writeln();

 // -------------------------------
 // SOLVE RCPSP+L-DLB
 // -------------------------------

writeln("RCPSP+L-DLB...");

// Set DLB time limit
cp.param.TimeLimit = TimeLimitDLB; // Number is in seconds

// Set file to write solution to
var solutionsDirectoryDLB = resultsDirectory + "Solutions" + dirSep + "Solutions-RCPSP+L-DLB" + dirSep;
var solutionPathDLB = solutionsDirectoryDLB + "Solution-RCPSP+L-DLB-" + testCaseFilename;
var solutionFileDLB = new IloOplOutputFile(solutionPathDLB);

// Find destructive LB
var lb = 0; // Start from 0 to see how well DLB really performs
var cumulativeSolveTimeDLB = 0; // Cumulative amount of time taken on all DLB bounds tested for this test case
var firstRun = true; // So we can write initial parameters to results file only on first run of loop
var keepSearching = true;
while (keepSearching) {
  // Generate model
```

```
       // Must be inside this loop so it's re-generated for every LB
         ↪  we destruct
     var sourceDLB = new IloOplModelSource("RCPSP+L-DLB.mod");
     var defDLB = new IloOplModelDefinition(sourceDLB);
     var oplDLB = new IloOplModel(defDLB,cp);
     oplDLB.addDataSource(data);
     oplDLB.generate();

     // Write initial parameters to results file (if on first run of
      ↪  loop)
       // Must be inside loop because it uses the generated model
     if (firstRun) {
       resultsFile.write(oplDLB.TestCaseFilename); // Test case
         ↪  filename
       resultsFile.write(",RCPSP+L-DLB"); // Test type
       resultsFile.write("," + oplDLB.Instance); // Test instance
         ↪  derived from
       resultsFile.write("," + (oplDLB.NbTasks-2)); // Number of
         ↪  tasks (excluding empty start/end tasks)
       resultsFile.write("," + oplDLB.NbRsrcs); // Number of
         ↪  resources
       resultsFile.write("," + oplDLB.Capacity); // Number of
         ↪  resources
       resultsFile.write("," + oplDLB.TargetStudentPercent); //
         ↪  Target percent of students
       resultsFile.write("," + oplDLB.TargetStudentCount); // Target
         ↪  number of students
       resultsFile.write("," + oplDLB.ActualStudentPercent); //
         ↪  Actual percent of students
       resultsFile.write("," + oplDLB.ActualStudentCount); // Actual
         ↪  number of students
       resultsFile.write("," + oplDLB.DurationReductionPercent); //
         ↪  Percent of duration reduction
       resultsFile.write("," + oplDLB.Seed); // Random seed
//         resultsFile.write("," + cpVersion); // CP Optimizer solver
 ↪  version

       firstRun = false; // Prevent it from writing the initial
         ↪  parameters multiple times
     }

     // Use new yet-to-be-destructed LB on each successive run
     oplDLB.Horizon[0] = lb;

     // Attempt to solve
     cp.solve();

     // Test whether this potential LB is valid
     if (cp.status == 2 || cp.status == 1) { // "Optimal" or
      ↪  "Feasible" -- A solution is found
       // Since we're testing LBs, this solution inherently must be
         ↪  optimal

       // Add this run's SolveTime to the cumulative
       cumulativeSolveTimeDLB += cp.info.SolveTime;

       // Run solution validation code in model file's
         ↪  post-processing execute statement
         // If a solution is invalid, the entire batch runner will
           ↪  fail with an error at that test and line
       oplDLB.postProcess();

```

```
        // Calculate solve-dependent learning utilization
        var utilizedLearningCount = 0;
        for (var t in oplDLB.Tasks) {
          if (t.learning[0] > 0 && oplDLB.TaskItvs[t].size ==
           ↪  t.learning[1]) {
            utilizedLearningCount++;
          }
        }
          // Calculate % of all tasks that learned (-2 to ignore empty
           ↪  start and end tasks)
        var utilizedLearningPercent =
         ↪  (utilizedLearningCount/(oplDLB.Tasks.size-2))*100;
          // Round to max of 2 decimal places (borrowed from
           ↪  https://stackoverflow.com/a/18358056/6402733)
        utilizedLearningPercent = +(Math.round(utilizedLearningPercent
         ↪  + "e+2")  + "e-2");

        // Calculate gap
        var gapAbsolute = (cp.getObjValue()-oplDLB.Horizon[0]);
        var gapPercent = (gapAbsolute/cp.getObjValue())*100;
          // Round to max of 2 decimal places (borrowed from
           ↪  https://stackoverflow.com/a/18358056/6402733)
        gapPercent = +(Math.round(gapPercent + "e+2")  + "e-2");

        // Status for this final bound
        if (cp.status == 1) {
          writeln(" - Feasible LB = " + oplDLB.Horizon[0]); // Horizon
        }
        else {
          writeln(" - Optimal LB = " + oplDLB.Horizon[0]); // Horizon
        }

        writeln(" - Makespan = " + cp.getObjValue() + " (LB - " +
         ↪  oplDLB.Horizon[0] + ", Gap - " + gapPercent + "%)"); //
         ↪  Makespan
        writeln(" - % of Tasks Utilizing Learning = " +
         ↪  utilizedLearningPercent + "%"); // % of utilized students

        // Write solve-dependent results to results file
        resultsFile.write("," + cp.getObjValue()); // Makespan
        resultsFile.write("," + gapAbsolute); // Gap #
        resultsFile.write("," + gapPercent); // Gap %
        resultsFile.write("," + utilizedLearningPercent); // % of
         ↪  utilized students
        resultsFile.write("," + utilizedLearningCount); // Number of
         ↪  utilized students

        // Write results to solution file
        solutionFileDLB.writeln("// Results");
        solutionFileDLB.writeln("Makespan = " + cp.getObjValue()); //
         ↪  Makespan
        solutionFileDLB.writeln("Lower Bound = " + oplDLB.Horizon[0]);
         ↪  // Lower bound
        solutionFileDLB.writeln("Gap % = " + gapPercent + "%"); // Gap
         ↪  %
        solutionFileDLB.writeln("Number of Solutions = " +
         ↪  cp.info.NumberOfSolutions); // Number of solutions found
        solutionFileDLB.writeln("Status = " + cp.status); // Solver
         ↪  status
        solutionFileDLB.writeln("Utilized Learning Percent = " + utilizedLearningPercent + "%"); // % of utilized students
        solutionFileDLB.writeln("Utilized Learning Count = " + utilizedLearningCount); // Number of utilized students
        solutionFileDLB.writeln("Cumulative Solve Time (sec) = " + cumulativeSolveTimeDLB); // Time spent solving all DLB bounds
        solutionFileDLB.writeln("Final Run's Solve Time (sec) = " + cp.info.SolveTime); // Time spent testing the final bound
        solutionFileDLB.writeln("Time Limit (sec) = " + cp.param.TimeLimit); // Time limit
//         solutionFileDLB.writeln("CP Optimizer Solver Version = " + cpVersion); // CP Optimizer solver version
        solutionFileDLB.writeln();
        // Write solution to solution file
        solutionFileDLB.writeln("// Solution");
        solutionFileDLB.writeln("Task Name | Start Period | Duration"); // Solution headers
        for (var t in oplDLB.Tasks) {
          solutionFileDLB.writeln("Task " + t.id + ";" + oplDLB.TaskItvs[t].start + ";" + oplDLB.TaskItvs[t].size);
        }
        solutionFileDLB.writeln();
        // Write precedences to solution file for import into yEd
        solutionFileDLB.writeln("// Precedences");
        solutionFileDLB.writeln("Edge ID | Node 1 | Node 2"); // Print headers
        var edgeID = 0;
        for (var t in oplDLB.Tasks) {
          for (var s in t.succs) {
            edgeID++;
            solutionFileDLB.writeln(edgeID + ";" + t.id + ";" + s);
          }
        }
        solutionFileDLB.writeln();
        solutionFileDLB.writeln(oplDLB.printExternalData()); // Write variable values

        // Already found optimal solution, don't waste time looking for worse ones
        keepSearching = false;
      }
      else if (cp.status == 3) { // "Infeasible" -- Proven impossible
        // This one is impossible, let's test the next one
        writeln(testNumber+1 + ") - Infeasible LB = " + oplDLB.Horizon[0]); // Makespan
        lb += 1; // Set up loop to validate the next lowest makespan on next iteration
        cumulativeSolveTimeDLB += cp.info.SolveTime; // Add this run's SolveTime to the cumulative
      }
      else {  // "Unkown" or other (will be executed if solver is stopped by time limit)
        // Add this run's SolveTime to the cumulative
        cumulativeSolveTimeDLB += cp.info.SolveTime;

        writeln(" - Makespan = UNDEFINED"); // Makespan
        writeln(" -");
```

```
       writeln(" - Stopped at LB = " + oplDLB.Horizon[0]); // Makespan

       // Leave empty columns in results file, one comma per column
         // getObjValue() and others fail if called when no solution has been found
       resultsFile.write(",,,,,");

       // Report failure to find solution in solution file
       solutionFileDLB.writeln("No solution found.");
       solutionFileDLB.writeln();
       // Write results to solution file
       solutionFileDLB.writeln("// Results");
       solutionFileDLB.writeln("Lower Bound = " + oplDLB.Horizon[0]); // Lower bound
       solutionFileDLB.writeln("Number of Solutions = " + cp.info.NumberOfSolutions); // Number of solutions found
       solutionFileDLB.writeln("Status = " + cp.status); // Solver status
       solutionFileDLB.writeln("Cumulative Solve Time (sec) = " + cumulativeSolveTimeDLB); // Time spent solving all DLB bounds
       solutionFileDLB.writeln("Final Run's Solve Time (sec) = " + cp.info.SolveTime); // Time spent testing the final bound
       solutionFileDLB.writeln("Time Limit (sec) = " + cp.param.TimeLimit); // Time limit
//        solutionFileDLB.writeln("CP Optimizer Solver Version = " + cpVersion); // CP Optimizer solver version
       solutionFileDLB.writeln();
       solutionFileDLB.writeln(oplDLB.printExternalData()); // Write variable values

       // Hand off to solver to continue solving normally
       keepSearching = false;
     }

     // End processes to prevent memory leaks
     oplDLB.end();
     defDLB.end();
     sourceDLB.end();
   }

   // Report some results to script log for visibility
   writeln(" - Status = " + cp.status); // Solver status
   writeln(" - Cumulative Solve Time = " + cumulativeSolveTimeDLB + " sec"); // Time spent solving all DLB bounds
   writeln(" - Final Run's Solve Time = " + cp.info.SolveTime + " sec"); // Time spent testing the final bound

   // Write values to results file
   resultsFile.write("," + lb); // Lower bound
   resultsFile.write("," + cp.info.NumberOfSolutions); // Number of solutions found
   resultsFile.write("," + cp.status); // Solver status
   resultsFile.write("," + cp.info.PresolveTime); // Time spent on presolve (preprocessing phase that aims to reduce both number of variables and constraints using logical rules)
   resultsFile.write("," + cp.info.ExtractionTime); // Time spent on extraction
   resultsFile.write("," + cumulativeSolveTimeDLB); // Time spent solving all DLB bounds
```

```
    resultsFile.write("," + cp.info.SolveTime); // Time spent testing
     ↪  the final bound
    resultsFile.write("," + cp.param.TimeLimit); // Time limit
    resultsFile.writeln(); // Flush buffer to file & start new line
     ↪  for next entry

    // Finish solution file
    solutionFileDLB.close();

     // ------------------------------
     // CLOSE OUT LOOP INSTANCE
     // ------------------------------

     // End memory usage
     data.end();
     cp.end();

     // Separate tests visually
     writeln();
   }

   // Close out results file
   resultsFile.close(); // Close file

   // Confirm to user that tests are complete
   writeln("All done!");
 }
```

APPENDIX E

EXAMPLE SOLUTION FILE

This appendix includes `Solution-TestCase-j3029_7-stu060-dur030-ran01.dat`, which is the optimal solution found by the model in Appendix C for the instance in Appendix B.1.

```
// Results
Makespan = 61
Lower Bound = 61
Gap % = 0%
Number of Solutions = 7
Status = 2
Utilized Learning Percent = 43.33%
Utilized Learning Count = 13
Solve Time (sec) = 2.844
Time Limit (sec) = 5

// Solution
Task Name | Start Period | End Period | Duration | Learned?
Task 1;0;0;0;No
Task 2;4;11;7;No
Task 3;0;4;4;No
Task 4;0;4;4;No
Task 5;4;6;2;No
Task 6;11;14;3;No
Task 7;26;33;7;Yes
Task 8;7;10;3;No
Task 9;4;7;3;No
Task 10;10;13;3;No
Task 11;26;33;7;Yes
Task 12;41;43;2;Yes
Task 13;43;51;8;No
Task 14;17;22;5;No
Task 15;33;37;4;Yes
Task 16;14;17;3;Yes
Task 17;37;41;4;No
Task 18;15;16;1;No
Task 19;10;15;5;No
Task 20;33;39;6;Yes
Task 21;54;58;4;Yes
Task 22;51;56;5;No
Task 23;16;26;10;No
Task 24;48;49;1;Yes
Task 25;39;48;9;No
Task 26;22;26;4;No
Task 27;58;59;1;Yes
Task 28;51;58;7;Yes
Task 29;49;54;5;Yes
Task 30;59;60;1;Yes
Task 31;59;61;2;Yes
Task 32;61;61;0;No

// Precedences
```

```
Edge ID | Node 1 | Node 2
1;1;2
2;1;3
3;1;4
4;2;6
5;2;11
6;2;18
7;3;5
8;4;8
9;4;9
10;4;16
11;5;7
12;5;13
13;5;19
14;6;12
15;7;15
16;7;20
17;8;19
18;8;20
19;8;23
20;9;10
21;9;15
22;10;11
23;10;14
24;11;12
25;11;21
26;12;13
27;12;24
28;12;30
29;13;22
30;14;17
31;14;26
32;15;25
33;15;28
34;16;20
35;17;27
36;17;28
37;17;29
38;18;22
39;18;23
40;19;21
41;19;25
42;20;21
43;20;24
44;20;27
45;21;31
46;22;31
47;23;24
48;23;25
49;23;29
50;24;28
51;25;27
52;26;29
53;26;30
54;27;30
55;28;31
56;29;32
57;30;32
58;31;32

TargetStudentPercent = 60;
TargetStudentCount = 18;
ActualStudentPercent = 60;
ActualStudentCount = 18;
DurationReductionPercent = 30;
Seed = 1;
Instance = "j3029_7";
```

```
TestCaseFilename = "TestCase-j3029_7-stu060-dur030-ran01.dat";
NbTasks = 32;
NbRsrcs = 4;
Capacity = [15 15 18 17];
```