

Project Scheduling with Mitigation Measures under Renewable Resource Constraints

A Time-Indexed Integer Linear Programming Approach

AM3001: Bachelor project

Juhyun Han



Project Scheduling with Mitigation Measures under Renewable Resource Constraints

A Time-Indexed
Integer Linear Programming
Approach

by

Juhyun Han

Student Name	Student Number
Han	5684900

Supervisor: T. Janssen
Second Examiner: M. Keijzer
Project Duration: April, 2026 - June, 2026
Faculty: Faculty of Electrical Engineering,
Mathematics and Computer Science, Delft

Cover: Construction worker in hard hat on building frame by Jeriden Villegas via Unsplash
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

Abstract

This thesis studies an extension of the Resource-Constrained Project Scheduling Problem (RCPSP) that incorporates multiple execution modes, time–cost trade-offs, and project tardiness penalties. In contrast to traditional time-indexed formulations that use activity start times as their primary decision reference, this research proposes a finish-time indexed integer linear programming formulation. The proposed model allows project schedules and activity execution modes to be optimized simultaneously while respecting precedence relations and renewable resource constraints.

From a theoretical perspective, the computational complexity of the problem is analysed. It is shown that the proposed formulation is NP-hard through a polynomial-time reduction from the classical RCPSP. Furthermore, even when renewable resource constraints are removed, the resulting problem remains NP-complete through a reduction from the Discrete Time–Cost Trade-Off Problem (DTCTP). These results indicate that the integration of discrete execution modes, precedence constraints, and tardiness penalties leads to a computationally challenging optimization problem.

To enable practical solution of the model, a computational framework was implemented in Python using Pyomo and the Gurobi Optimizer. Several preprocessing techniques were introduced, including sparse variable generation, earliest-time feasibility filtering, and resource activity preprocessing. In addition, a lexicographic optimization procedure was developed to identify the shortest project schedule among solutions with the same optimal objective value. Computational experiments were conducted on 72 benchmark instances derived from DTCTP datasets and extended with renewable resource information.

The results show that the preprocessing procedures substantially reduce the number of binary variables without affecting solution quality. Their impact on runtime depends on the formulation and instance characteristics, but they consistently reduce model size. A comparison with a start-time indexed formulation inspired by Bianco and Caramia [4] demonstrates that both formulations produce identical objective values, makespans, and tardiness values across all benchmark categories. The computational results further show that the proposed finish-time indexed formulation generally achieves shorter solution times and generates fewer constraints than the benchmark formulation, despite not always containing fewer binary variables. These findings indicate that finish-time indexing provides a competitive exact solution approach for project scheduling problems with discrete time–cost trade-offs and renewable resource constraints.

Keywords: Resource-Constrained Project Scheduling Problem (RCPSP), Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP), Discrete Time–Cost Trade-Off Problem (DTCTP), Finish-Time Indexing, Renewable Resource Constraints, Mitigation Measures, Preprocessing, Lexicographic Optimization.

Contents

Abstract	i
Nomenclature	v
1 Introduction	1
1.1 Research Motivation	1
1.2 Problem Description and Research Contributions	2
2 Preliminaries	4
2.1 Resource-Constrained Project Scheduling Problem	4
2.2 Critical Path Method	6
2.3 Relation between CPM and RCPSP	7
2.4 Summary	7
3 Mathematical Formulation with Mitigation Measures	8
3.1 Mathematical Modeling approach	9
3.2 Sets and Parameters	9
3.3 Decision Variables	9
3.4 Objective Function	10
3.5 Model Constraints	10
3.5.1 Mode and Time Selection	10
3.5.2 Precedence Constraints	10
3.5.3 Resource Constraints	10
3.5.4 Tardiness Constraints	11
3.5.5 Variable Domains	11
3.6 Discussion	11
3.7 Relation to CPM	11
3.8 Summary	12
4 Complexity Analysis	13
4.1 Complexity Classes	13
4.2 Complexity of the PSPMM	14
4.2.1 Motivation	14
4.2.2 Decision Version	14
4.2.3 Polynomial-Time Verification	14
4.2.4 NP-Hardness Result	14
4.2.5 NP-Completeness	16
4.3 Complexity Without Renewable Resource Constraints	16
4.3.1 Motivation	16
4.3.2 Decision Version	16
4.3.3 Polynomial-Time Verification	16
4.3.4 NP-Hardness Result	17
4.3.5 NP-Completeness	18
4.4 Discussion and Summary	18
5 Computational Implementation and Experimental Framework	19
5.1 Implementation Framework	19
5.1.1 Implementation of the PSPMM formulation	19
5.2 Lexicographic Optimization Procedure	20
5.3 Implementation Versions	21
5.4 Dataset Generation	21

5.4.1	Source Data	21
5.4.2	Generation Procedure	21
5.4.3	Instance Classification	22
5.5	Preprocessing Techniques	22
5.5.1	Sparse Variable Generation	23
5.5.2	Earliest-Time Preprocessing	23
5.5.3	Resource Activity Preprocessing	23
5.5.4	Summary of Preprocessing Techniques	24
5.6	Comparable Formulation	24
5.7	Experimental Design	25
5.8	Evaluation Metrics and Analysis Framework	25
5.9	Summary	26
6	Computational Results and Analysis	27
6.1	Generated Test Instances	27
6.2	Influence of Preprocessing Techniques	28
6.2.1	Impact on Solution Quality	28
6.2.2	Impact on Computational Efficiency	29
6.3	Influence of Lexicographic Optimization	30
6.3.1	Impact on Solution Quality	30
6.3.2	Impact on Computational Efficiency	31
6.4	Comparison of Formulations	32
6.4.1	Solution Quality Comparison	33
6.4.2	Computational Efficiency Comparison	33
6.5	Discussion and Interpretation of Results	34
6.6	Summary	35
7	Conclusion and Future Research	36
7.1	Limitations and Future Research	37
	Use of Artificial Intelligence Tools	39
	Reference	40
A	Benchmark Instance Characteristics	42
B	Detailed Computational Solution Quality Results	44
B.1	PSPMM Formulation	44
B.1.1	Version V1	44
B.1.2	Version V3	45
B.1.3	Version V4	47
B.2	Bianco–Caramia Formulation	48
B.2.1	Version V1	48
B.2.2	Version V3	50
B.2.3	Version V4	51
C	Detailed Computational Efficiency Results	54
C.1	PSPMM Formulation	54
C.1.1	Version V1	54
C.1.2	Version V3	55
C.1.3	Version V4	57
C.2	Bianco–Caramia Formulation	58
C.2.1	Version V1	58
C.2.2	Version V3	60
C.2.3	Version V4	61
D	Pyomo Model Implementations	64
D.1	PSPMM Formulation	64
D.2	Bianco–Caramia Formulation	68

E	Dataset Generation and Preprocessing	73
E.1	Dataset Augmentation	73
E.2	Data Loading	75

Nomenclature

Abbreviations

Abbreviation	Definition
CPM	Critical path method
PSPMM	Project scheduling problem with mitigation measures
NP	Nondeterministic polynomial time
DTCTP	Discrete time–cost trade-off problem
MRCPSP	Multi-mode resource constrained project scheduling problem

Symbols

Symbol	Definition	Unit
Sets and indices		
$\mathcal{J}$	Set of project activities	–
$\mathcal{K}$	Set of renewable resource types, where $\mathcal{K} = \{1, \dots, K\}$	–
$\mathcal{M}_j$	Set of execution modes for activity j	–
$\mathcal{P}_j$	Set of predecessor activities of activity j	–
$\mathcal{T}$	Set of discrete time periods, where $\mathcal{T} = \{0, \dots, UB\}$	–
$\mathcal{A}(t)$	Set of activities active during time period t	–
$\mathcal{I}_t$	Precomputed set of feasible activity–mode–finish-time tuples (j, m, τ) that are active during time period t	–
Decision variables		
$x_{j,t}^{(m)}$	Binary variable equal to 1 if activity j finishes at time τ in mode m , and 0 otherwise	–
L	Project tardiness	[time units]
Derived quantities		
F_j	Completion time of activity j	[time units]
Parameters		
$c_j^{(m)}$	Execution (mitigation) cost of activity j in mode m	[cost units]
D	Project deadline	[time units]
$d_j^{(m)}$	Duration of activity j in mode m	[time units]
$r_{j,k}^{(m)}$	Renewable resource requirement of activity j for resource k in mode m	[resource units]
R_k	Available capacity of renewable resource k	[resource units]
UB	Upper bound on the scheduling horizon	[time units]
α	Tardiness penalty parameter	[cost units/time unit]
Lexicographic optimization		
Z^*	Optimal objective value obtained in the first optimization phase	[cost units]

1

Introduction

Project scheduling is an important area of Operations Research that supports planning and decision-making in industries such as construction, manufacturing, infrastructure development, and engineering. In many projects, activities must satisfy precedence relationships while competing for limited resources. Determining an efficient schedule under these restrictions is commonly known as the Resource-Constrained Project Scheduling Problem (RCPSP). For example, consider a construction project consisting of foundation work, structural assembly, electrical installation, and final inspection. Structural assembly cannot begin until the foundation has been completed, while the final inspection can only start after all construction activities have finished. At the same time, several activities may require the same workers or equipment, whose availability is limited. The RCPSP seeks a schedule that respects both precedence relationships and resource capacities while achieving a project objective such as minimizing project completion time. The classical RCPSP determines activity start times such that all precedence and resource constraints are satisfied while minimizing the project makespan [6, 2, 21]. Due to its practical importance and computational difficulty, the problem has been studied extensively in the project scheduling literature [5, 17].

Although the classical RCPSP provides a useful foundation, many real-world projects require additional flexibility. Activities may be performed using different methods or technologies, resulting in alternative combinations of duration, cost, and resource consumption. Such situations are commonly modeled using the Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP) [6, 10]. In addition, project managers often implement acceleration or mitigation measures to reduce activity durations and avoid project delays. Examples of such measures include assigning additional workers, allowing overtime, using more efficient equipment, outsourcing certain tasks, or adopting alternative methods of execution. These decisions are closely related to the Discrete Time–Cost Trade-Off Problem (DTCTP), which examines the relationship between project duration and activity acceleration costs [9, 1, 14]. In practice, scheduling decisions, execution-mode selection, and mitigation measures are often interdependent. As a result, there is a need for optimization models that can simultaneously account for precedence relations, renewable resource constraints, alternative execution modes, mitigation decisions, and project tardiness costs within a single decision framework.

1.1. Research Motivation

Several streams of research have contributed to the development of project scheduling models. The RCPSP and MRCPSP literature provides extensive modeling frameworks and solution approaches for scheduling under resource constraints [6, 2]. Similarly, the DTCTP literature investigates the trade-off between project completion time and activity acceleration costs [9, 1, 14]. Many exact optimization approaches for these problems are based on time-indexed formulations. Early work by Pritsker [22] introduced a zero–one programming framework for project scheduling, while later studies proposed alternative formulations to improve computational performance [4, 23, 18]. Among these approaches, start-time indexed formulations remain one of the most widely used modeling frameworks.

Despite the extensive literature on project scheduling, relatively little attention has been given to alternative indexing structures for problems that simultaneously incorporate renewable resource constraints, multiple execution modes, mitigation decisions, and tardiness penalties. In particular, most existing formulations associate binary decision variables with activity start times. Comparatively little research has examined whether a finish-time indexed representation can provide a competitive alternative from both a modeling and computational perspective.

This thesis addresses this gap by developing a finish-time indexed integer linear programming formulation for a project scheduling problem that integrates scheduling decisions, execution-mode selection, mitigation measures, renewable resource constraints, and project tardiness costs within a unified optimization framework. The formulation is subsequently compared with the start-time indexed formulation of Bianco and Caramia [4] through a computational study.

1.2. Problem Description and Research Contributions

This thesis considers a project scheduling problem in which each activity can be executed in one of several available modes. Different execution modes result in different activity durations, mitigation costs, and resource requirements. Activities are linked through precedence relationships and compete for limited renewable resources. In addition, the project is associated with a completion deadline, and tardiness costs are incurred when the project finishes after this deadline.

The objective is to determine activity schedules and execution modes that minimize total project cost while satisfying all precedence and resource constraints.

From a computational perspective, time-indexed mixed-integer programming formulations can become very large because binary decision variables must be generated for multiple activities, execution modes, and time periods. Consequently, both formulation design and preprocessing techniques play an important role in determining computational tractability.

The study is guided by the following main research question:

How can a finish-time indexed integer linear programming formulation, combined with preprocessing techniques, effectively model and solve a resource-constrained project scheduling problem with multiple execution modes, mitigation measures, and project tardiness costs?

To answer this question, the following sub-questions are investigated:

1. What is the computational complexity of the project scheduling problem with mitigation measures under different problem settings?
2. How does the proposed finish-time indexed formulation compare with the start-time indexed formulation of Bianco and Caramia in terms of solution quality and computational performance?
3. To what extent do preprocessing techniques improve the computational tractability of time-indexed scheduling formulations?

This thesis contributes to the project scheduling literature in three main ways.

First, a finish-time indexed integer linear programming formulation is developed for a project scheduling problem that integrates activity scheduling, execution-mode selection, mitigation decisions, renewable resource constraints, and project tardiness costs within a single optimization framework.

Second, the thesis contributes to the theoretical understanding of the problem by establishing its computational complexity. The complete problem is shown to be NP-hard through a reduction from the classical RCPSP. Furthermore, the problem remains NP-complete even when renewable resource constraints are removed, demonstrating that its computational difficulty is not solely caused by resource limitations.

Third, several preprocessing techniques are developed and implemented to improve computational efficiency. These include horizon reduction, earliest-time feasibility filtering, sparse feasible variable generation, and resource preprocessing. Their effectiveness is evaluated through computational experiments on a set of benchmark instances.

Finally, the proposed formulation is compared with the start-time indexed formulation of Bianco and Caramia [4]. The comparison evaluates both solution quality and computational efficiency under a common experimental framework. This makes it possible to assess the impact of alternative indexing structures while controlling for differences in implementation, preprocessing, and solver settings.

The remainder of this thesis is organized as follows. Chapter 2 presents the theoretical background and reviews the relevant literature on project scheduling, the RCPSP, the MRCPSP, the DTCTP, and time-indexed optimization formulations. Chapter 3 introduces the proposed finish-time indexed mathematical formulation, including its sets, parameters, decision variables, objective function, and constraints. Chapter 4 analyzes the computational complexity of the problem and presents the NP-hardness and NP-completeness proofs. Chapter 5 describes the implementation methodology, including data preparation, preprocessing techniques, model generation, and lexicographic optimization procedures. Chapter 6 presents and analyzes the computational results, including an evaluation of the preprocessing techniques and a comparison between the proposed formulation and the formulation of Bianco and Caramia [4]. Finally, Chapter 7 summarizes the main findings of the study, answers the research questions, discusses the limitations of the research, and outlines directions for future work.

2

Preliminaries

Before developing the optimization model studied in this thesis, it is necessary to introduce the classical scheduling concepts on which the model is built. Project scheduling problems arise in many practical settings, where a collection of interdependent activities must be executed while limited resources are available. Activities are often linked through precedence relationships, which specify that certain activities must be completed before others may begin. These relationships represent technological or logical dependencies within a project network. In such environments, decisions regarding the timing of activities directly influence the duration of the project, resource utilization, and ultimately project cost.

One of the most widely studied models in the project scheduling literature is the Resource-Constrained Project Scheduling Problem (RCPSP). The RCPSP provides a mathematical framework for scheduling activities under precedence relations and limited renewable resources. Although the model is relatively compact, the interaction between activity dependencies and resource limitations creates a highly complex optimization problem. As the project size increases, finding optimal schedules quickly becomes computationally difficult. In addition to resource-constrained scheduling, precedence-based scheduling methods also play an important role in project planning. The Critical Path Method (CPM) determines the shortest possible project duration when resource limitations are ignored and identifies the activities that directly determine the project completion time. These activities form the critical path, meaning that any delay in one of them immediately increases the overall project duration. Although CPM represents a simplified scheduling framework, it provides important analytical insights into project structure and schedule sensitivity.

The concepts introduced in this chapter form the theoretical foundation for the remainder of this thesis. The RCPSP provides the basic scheduling framework used throughout the study, while the CPM offers useful analytical insights into project structure and project duration. Together, these concepts provide the basis for the extended optimization model developed in the next chapter.

2.1. Resource-Constrained Project Scheduling Problem

The Resource-Constrained Project Scheduling Problem (RCPSP) consists of scheduling a set of activities subject to precedence and resource constraints [17]. Each activity has a fixed duration and requires a certain amount of renewable resources during its execution. Renewable resources are resources whose availability renews in every time period, such as workers, machines, or equipment.

The objective of the RCPSP is typically to minimize the project makespan, that is, the completion time of the final activity. Two main types of constraints define the problem. First, precedence constraints specify the order in which activities must be executed. Second, resource constraints ensure that the total usage of the resources at any time does not exceed the available capacity.

The RCPSP is known to be NP-hard, which means that optimally solving large instances is computationally difficult [5]. Despite this complexity, the RCPSP is widely used in practice because it provides a realistic representation of project scheduling problems with limited resources.

The formulation presented in this section follows the standard RCPSP framework described in the literature. In particular, the definitions of activities, precedence relations, and resource constraints are adopted from the classical models of Kolisch and Hartmann [17]. At this stage, no extensions are introduced to the basic RCPSP structure.

Project Structure and Activities

Consider a set of activities

$$\mathcal{J} = \{0, 1, \dots, n, n+1\},$$

where activity 0 represents the start of the project and activity $n+1$ represents the end. These two activities are dummy activities with zero duration and no resource requirements. The remaining activities $j = 1, \dots, n$ are actual project tasks.

Each activity $j \in \mathcal{J}$ has:

- a duration $p_j \geq 0$,
- resource requirements $r_{j,k} \geq 0$ for each type of resource $k = 1, \dots, K$.

The precedence relations between activities are represented by a directed acyclic graph. For each activity j , let $\mathcal{P}_j$ denote the set of immediate predecessors of activity j . These relations ensure that certain activities must be completed before others can begin. By convention, the dummy start activity 0 is treated as a predecessor of every activity without an actual predecessor, while the dummy end activity $n+1$ is defined as a successor of every activity without an actual successor. Consequently, every project schedule begins at activity 0 and terminates at activity $n+1$.

Schedule Representation

A schedule assigns a start time S_j to each activity j . The dummy start activity is fixed at time zero, so $S_0 = 0$. The corresponding finish time is defined as

$$F_j = S_j + p_j.$$

A feasible schedule must satisfy both precedence and resource constraints.

Constraints

Precedence Constraints If activity h must be completed before activity j , then

$$S_j \geq F_h = S_h + p_h, \quad \forall h \in \mathcal{P}_j.$$

Hence, an activity may only start after all predecessor activities have finished.

Resource Constraints At any time period t , the total use of each renewable resource may not exceed its available capacity. Let

$$\mathcal{A}(t) = \{j \in \mathcal{J} \mid S_j \leq t < F_j\}$$

denote the set of activities being processed at time t . Then, for each resource type k ,

$$\sum_{j \in \mathcal{A}(t)} r_{j,k} \leq R_k,$$

where R_k denotes the available capacity of resource k . These constraints ensure that renewable resources such as workers or machines are not overutilized.

Objective Function

The objective of the classical RCPSP is to minimize the project makespan:

$$C_{\max} = F_{n+1}.$$

This equality follows from the project network structure. Since the dummy end activity $n+1$ is defined as a successor of all terminal project activities and has zero duration, it can only be completed after all other activities have been completed. Therefore, its finish time coincides with the completion time of the entire project and represents the project makespan.

Discussion

The RCPSP is challenging because precedence constraints and resource constraints interact simultaneously. Even when activities could start earlier based solely on precedence relations, limited resource availability may force delays. Consequently, the project makespan may increase compared with the CPM schedule.

Computational Complexity

The RCPSP generalizes several known NP-hard scheduling problems [5]. As a result, no polynomial-time algorithm is expected to solve all instances optimally. For this reason, practical solution approaches often rely on heuristics, approximation methods, or decomposition techniques.

2.2. Critical Path Method

The Critical Path Method (CPM) is a classical technique used to analyze project schedules under precedence constraints [16]. Unlike the RCPSP, CPM does not consider resource limitations. Instead, it focuses exclusively on the technological order in which activities must be executed. CPM determines the shortest possible project duration when only precedence constraints are taken into account.

Earliest Start and Finish Times

CPM first computes the earliest feasible schedule using a forward pass. Let S_j^E denote the earliest start time and F_j^E the earliest finish time of activity j . Then,

$$S_j^E = \max_{h \in \mathcal{P}_j} \{F_h^E\}, \quad F_j^E = S_j^E + p_j,$$

with

$$S_0^E = 0.$$

The resulting schedule starts each activity as early as possible while satisfying all precedence relations.

Latest Start Times

CPM also determines how much an activity can be delayed without affecting the project completion time. This is computed using a backward pass. Let S_j^L denote the latest start time of activity j . Then,

$$S_j^L = \min_{k \in \text{Succ}(j)} \{S_k^L - p_j\},$$

where $\text{Succ}(j)$ denotes the set of immediate successors of activity j .

For the final activity,

$$S_{n+1}^L = F_{n+1}^E.$$

The difference between earliest and latest start times determines the scheduling flexibility of an activity. The forward and backward pass equations used in CPM originate from the classical formulation introduced by Kelley [16]. In this thesis, these results are used directly as an analytical tool and are not modified further.

Makespan and Critical Path

The CPM makespan is given by

$$F_{n+1}^{CPM} = F_{n+1}^E.$$

The **critical path** is the sequence of activities that determines this makespan. An activity j is called critical if

$$S_j^E = S_j^L.$$

Critical activities have zero slack and therefore cannot be delayed without increasing the total project duration.

Role of CPM in This Thesis

In this thesis, CPM serves as a reference model for evaluating schedules. Since CPM ignores resource constraints, it provides a lower bound on the project duration. The critical path is particularly important when mitigation measures are introduced. Reducing the duration of activities on the critical path may shorten the overall project duration, whereas reducing the duration of non-critical activities may have no effect on the project makespan.

2.3. Relation between CPM and RCPSP

The CPM provides a lower bound for the RCPSP. Since CPM ignores resource constraints, it determines the shortest possible project duration based solely on precedence relations:

$$F_{n+1}^{CPM} \leq F_{n+1}^{RCPSP}.$$

In practice, resource limitations often delay activities and therefore increase the project makespan relative to the CPM solution.

In the extended model introduced in the next chapter, CPM remains useful as a reference framework. For a fixed choice of activity execution modes, CPM defines the corresponding precedence-based project duration. This makes it possible to analyze how mitigation decisions influence both the critical path and the overall project duration. In this thesis, the CPM results are used directly as an analytical tool for interpreting project structure and schedule duration and are not modified further.

2.4. Summary

In this chapter, the RCPSP and the CPM has been introduced. The RCPSP models project scheduling under precedence and renewable resource constraints, while CPM provides a precedence-based lower bound on the project duration.

The classical RCPSP assumes fixed activity durations and fixed resource capacities. However, practical project environments often allow durations to be influenced through mitigation measures at additional cost. This observation motivates the extended optimization model developed in the next chapter, where scheduling decisions and mitigation decisions are integrated within a single mathematical framework.

3

Mathematical Formulation with Mitigation Measures

In Chapter 2, the classical Resource-Constrained Project Scheduling Problem (RCPSP) was introduced. In this model, activity durations and resource requirements are fixed. However, in real projects, managers can often influence how activities are executed. For example, an activity may be completed faster by assigning additional workers, using more equipment, or applying alternative execution methods. These actions can reduce the project duration, but they usually increase project costs. This creates a trade-off between time and cost.

The classical RCPSP does not include this trade-off directly, because activity durations are treated as fixed parameters. As a result, the model cannot represent mitigation decisions that change activity durations during the optimization process. To address this limitation, this chapter introduces an extended formulation in which activities can be executed in different **modes**. Each mode represents a different mitigation level with its own:

- activity duration,
- resource usage,
- and cost.

This allows scheduling decisions and mitigation decisions to be made together in one optimization model. The formulation is related to the multi-mode RCPSP (MRCPSP) [17, 10]. A time-indexed integer linear programming formulation is used based on the zero-one model of Pritsker et al. [22]. This formulation makes it possible to model resource usage explicitly over time.

The main goal of this chapter is to develop a mathematical model that combines:

- scheduling decisions,
- mitigation decisions through mode selection,
- renewable resource constraints,
- and project tardiness penalties

within one framework. The resulting optimization problem is referred to as the Project Scheduling Problem with Mitigation Measures (PSPMM). The PSPMM extends the classical RCPSP by incorporating execution-mode selection, mitigation costs, and project tardiness penalties. The remainder of this chapter presents a time-indexed integer programming formulation of the PSPMM. This integrated model forms the basis for the computational and complexity analysis in the following chapters.

3.1. Mathematical Modeling approach

The classical RCPSP assumes that each activity has a fixed duration and fixed resource usage. In practice, however, project managers can often influence how activities are performed. For example, activities may be completed faster by assigning additional workers, using more equipment, or applying alternative execution methods. These actions reduce activity durations, but they usually increase project costs. This creates a trade-off between project duration and project cost. A common way to model this trade-off is through multiple execution modes. In the literature, this idea is used in the multi-mode RCPSP (MRCPS) [17, 10] and in time–cost trade-off models [20].

In this chapter, each activity can be executed in different modes. Each mode represents a specific mitigation strategy with its own duration, resource usage, and cost. The main idea of the formulation is that scheduling decisions and mitigation decisions are made simultaneously. The model therefore determines:

- when activities are scheduled,
- which execution modes are selected,
- and how renewable resources are allocated over time.

To model these decisions explicitly over time, a time-indexed integer linear programming formulation is used. This formulation is based on the zero–one model of Pritsker et al. [22]. The following sections present the complete mathematical formulation of the PSPMM.

3.2. Sets and Parameters

Let:

- $\mathcal{T} = \{0, 1, \dots, H\}$ be the planning horizon,
- $\mathcal{J} = \{0, 1, \dots, n, n + 1\}$ be the set of activities,
- $\mathcal{P}_j \subseteq \mathcal{J}$ be the set of predecessors of activity j ,
- $\mathcal{K} = \{1, \dots, K\}$ be the set of renewable resources,
- $\mathcal{M}_j$ be the set of execution modes for activity j .

Activities 0 and $n + 1$ represent the start and end of the project and are dummy activities with zero duration and zero resource usage.

For each mode $m \in \mathcal{M}_j$, define:

- $d_j^{(m)}$: duration of activity j in mode m ,
- $r_{j;k}^{(m)}$: resource usage of resource k ,
- $c_j^{(m)}$: cost associated with mode m .

Additional parameters:

- R_k : capacity of renewable resource k ,
- D : project deadline,
- α : penalty cost per unit time of project tardiness.

The parameter α represents the penalty associated with one unit of project tardiness. It controls the trade-off between mitigation cost and project delay. Larger values of α prioritize earlier project completion, while smaller values favor lower mitigation costs.

3.3. Decision Variables

A time-indexed binary formulation based on Pritsker et al. [22] is adopted. The decision variable is defined as:

$$x_{j,t}^{(m)} = \begin{cases} 1 & \text{if activity } j \text{ is completed at time } t \text{ in mode } m, \\ 0 & \text{otherwise.} \end{cases}$$

This formulation integrates both scheduling and mitigation decisions into a single variable structure.

The completion time of activity j is given by:

$$F_j = \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} t \cdot x_{j,t}^{(m)}.$$

Because each activity is executed exactly once, this representation uniquely determines both timing and mode selection.

Project tardiness is represented by the variable:

$$L = \max\{0, F_{n+1} - D\}.$$

3.4. Objective Function

The objective is to minimize the total mitigation cost and project tardiness penalty. The objective function is defined as:

$$\min \left(\alpha L + \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} c_j^{(m)} x_{j,t}^{(m)} \right).$$

The first term penalizes project tardiness and the second term represents the total mitigation cost associated with the selected execution modes.

3.5. Model Constraints

3.5.1. Mode and Time Selection

$$\sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} x_{j,t}^{(m)} = 1, \quad \forall j \in \mathcal{J}.$$

Each activity must be executed exactly once, in exactly one mode and at exactly one completion time.

3.5.2. Precedence Constraints

$$F_j - \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} d_j^{(m)} x_{j,t}^{(m)} \geq F_h, \quad \forall j \in \mathcal{J}, \forall h \in \mathcal{P}_j.$$

These constraints ensure that no activity starts before all its predecessors have been completed. Since the formulation uses completion-time variables, the start time of activity j is given by its completion time minus its duration. Because activity durations depend on the selected execution modes, scheduling and mitigation decisions are directly linked.

3.5.3. Resource Constraints

$$\sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{\tau=t+1}^{t+p_j^{(m)}} r_{j,k}^{(m)} x_{j,\tau}^{(m)} \leq R_k, \quad \forall k \in \mathcal{K}, \forall t \in \mathcal{T}.$$

These constraints ensure that renewable resource capacities are not exceeded at any point in time. Since the decision variable indicates the completion time of an activity, an activity that finishes at time τ and has duration $p_j^{(m)}$ is active during the periods $\tau - p_j^{(m)}, \dots, \tau - 1$. Therefore, at time period t , all

activities with completion times $\tau \in \{t+1, \dots, t+p_j^{(m)}\}$ consume resources. The total resource usage of these activities may not exceed the available resource capacity R_k .

3.5.4. Tardiness Constraints

Let $L \geq 0$ denote the project tardiness.

$$L \geq F_{n+1} - D,$$

$$L \geq 0.$$

These constraints linearize project tardiness and ensure that tardiness is only counted when the project completion time exceeds the deadline [21].

3.5.5. Variable Domains

$$x_{j,t}^{(m)} \in \{0,1\}, \quad \forall j \in \mathcal{J}, m \in \mathcal{M}_j, t \in \mathcal{T}.$$

$$L \in \mathbb{R}_{\geq 0}.$$

These domain restrictions ensure that the scheduling decisions remain discrete while allowing project tardiness to take any non-negative continuous value. The binary variables guarantee that each activity is assigned to exactly one completion time and one execution mode, whereas the tardiness variable measures the extent to which the project completion time exceeds the specified deadline.

3.6. Discussion

The PSPMM extends the classical RCPSP by incorporating mitigation decisions through multiple execution modes. Instead of assigning only completion times to activities, the model must also determine which execution mode is selected for each activity. This creates a larger decision space, since scheduling and mitigation decisions are made simultaneously.

The model simultaneously determines the timing of activities, the execution mode selected for each activity, the allocation of renewable resources over time, and the resulting level of project tardiness. These decisions are strongly interconnected. Changing the execution mode of an activity affects not only its duration, but also its resource usage and mitigation cost. As a consequence, the feasibility and quality of the complete schedule may change. In addition, the project completion time influences the tardiness penalty in the objective function. The inclusion of project tardiness introduces additional flexibility compared to fixed-deadline formulations. Instead of requiring strict completion before a specified deadline, the model allows late completion when the cost of additional mitigation becomes too high. The objective function therefore balances two competing goals: reducing mitigation costs and reducing project delay. The parameter α controls this trade-off. Larger values of α encourage earlier project completion by increasing the penalty for tardiness, while smaller values place more emphasis on limiting mitigation costs.

Although the formulation provides a more realistic representation of project management decisions, it also increases computational complexity. The number of decision variables grows with the number of activities, execution modes, and time periods in the planning horizon. As a result, the model becomes difficult to solve exactly for larger instances. This increased complexity motivates the computational analysis presented in the following chapters. In particular, the next chapter studies the computational complexity of the PSPMM and analyzes its relation to known NP-hard scheduling problems.

3.7. Relation to CPM

For a fixed selection of execution modes, each activity has a fixed duration. In that case, the project network becomes deterministic and the Critical Path Method (CPM) can be applied. CPM determines the longest path in the precedence network while ignoring renewable resource constraints. The resulting project duration provides a lower bound for the PSPMM. However, mitigation decisions may

change activity durations and therefore alter the critical path itself. In addition, renewable resource constraints may delay activities beyond their CPM schedule. Therefore, CPM remains useful as a structural reference, but it does not capture the full complexity of the PSPMM.

3.8. Summary

In this chapter, the Project Scheduling Problem with Mitigation Measures (PSPMM) was introduced together with a time-indexed integer linear programming formulation. The PSPMM extends the classical RCPSP by allowing activities to be executed in different modes and by incorporating project delay costs directly into the objective function. This formulation makes it possible to model scheduling decisions, mitigation decisions, and the trade-off between mitigation cost and project tardiness within one optimization framework.

4

Complexity Analysis

In Chapter 3, the Project Scheduling Problem with Mitigation Measures (PSPMM) was introduced together with a time-indexed integer programming formulation. The PSPMM extends the classical Resource-Constrained Project Scheduling Problem (RCPSP) by allowing activities to be executed in different modes. Each mode represents a different combination of activity duration, renewable resource usage, and mitigation cost. In addition, the PSPMM incorporates project deadlines and tardiness penalties, thereby explicitly modeling the trade-off between mitigation costs and project delay. The complexity analysis therefore focuses on the decision version of the PSPMM.

Compared to the classical RCPSP, the PSPMM contains additional decision dimensions. The problem requires determining activity timing, execution-mode selection, renewable resource allocation, and the balance between mitigation costs and project tardiness. As the number of activities, execution modes, and time periods increases, the number of feasible schedules grows rapidly. Consequently, solving the PSPMM exactly may become computationally difficult for larger instances.

The purpose of this chapter is to analyze the computational complexity of the PSPMM. In particular, the analysis investigates whether the PSPMM is NP-hard and whether its decision version belongs to the class NP. Complexity analysis provides a theoretical explanation for why many scheduling problems are difficult to solve exactly. If a problem is NP-hard, no known polynomial-time algorithm exists that solves all instances optimally [11]. This has direct practical implications, since exact methods typically become intractable for larger instances. To establish NP-hardness, optimization problems are reformulated as decision problems. A decision problem asks whether a feasible solution exists that satisfies a given bound. Such formulations are required for NP, NP-hard, and NP-complete classifications [11].

In this chapter, two variants of the PSPMM are studied. First, the complete PSPMM with renewable resource constraints is analyzed. Second, a reduced PSPMM without renewable resource constraints is considered. The first result shows that the complete PSPMM is NP-hard because it contains the classical RCPSP as a special case. The second result shows that the reduced PSPMM remains NP-complete through a reduction from the discrete time–cost trade-off problem (DTCTP). Unlike the RCPSP, the DTCTP does not include renewable resource constraints, but it still involves mode-dependent durations and costs [14]. Standard NP-hardness techniques based on polynomial-time reductions are used throughout this chapter [5, 6].

4.1. Complexity Classes

Computational complexity theory studies the intrinsic difficulty of computational problems. The class NP contains decision problems for which a proposed solution can be verified in polynomial time. A problem is NP-hard if every problem in NP can be reduced to it in polynomial time. NP-hard problems are therefore at least as difficult as the hardest problems in NP [11].

The classical RCPSP is known to be NP-hard [5], and the DTCTP is also NP-hard [9]. These results are used as foundations for the reductions presented in this chapter.

4.2. Complexity of the PSPMM

4.2.1. Motivation

The PSPMM extends the classical RCPSP by incorporating multiple execution modes, mitigation costs, project deadlines, tardiness penalties, and renewable resource constraints. The resulting problem must determine both scheduling and mitigation decisions simultaneously. To analyze its computational complexity, the decision version of the PSPMM is considered. The goal is to determine whether a feasible schedule exists that satisfies all constraints while remaining within a given objective bound. The NP-hardness proof is based on a polynomial-time reduction from the classical RCPSP. Since the RCPSP is NP-hard, the PSPMM is at least as difficult if it contains the RCPSP as a special case.

4.2.2. Decision Version

The decision version of the PSPMM is defined as follows.

Given an instance of the PSPMM together with a deadline D , a tardiness penalty α , and an objective threshold K , does there exist a feasible schedule satisfying:

- the mode selection constraints from Subsection 3.5.1,
- the precedence constraints from Subsection 3.5.2,
- the renewable resource constraints from Subsection 3.5.3,
- and the tardiness constraints from Subsection 3.5.4,

such that

$$\alpha L + \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} c_j^{(m)} x_{j,t}^{(m)} \leq K,$$

where

$$L = \max\{0, F_{n+1} - D\}?$$

4.2.3. Polynomial-Time Verification

Lemma 4.1. *The decision version of the PSPMM belongs to NP.*

Proof. A certificate consists of a selected execution mode for each activity, completion times for all activities, and the project tardiness value L . Knowing the selected execution mode and completion time for each activity uniquely determines all binary variables $x_{j,t}^{(m)}$. Using the constraints introduced in Section 3.5, verification requires checking:

- whether exactly one execution mode and completion time are selected for each activity;
- whether all precedence constraints are satisfied;
- whether all renewable resource constraints are satisfied;
- whether the tardiness constraints hold;
- and whether the objective value does not exceed the threshold K .

Each individual check can be performed in polynomial time. Furthermore, the total number of constraints that must be verified is polynomial in the size of the problem instance. Therefore, the complete verification procedure requires polynomial time. Hence, the decision version of the PSPMM belongs to NP. $\square$

4.2.4. NP-Hardness Result

Theorem 4.2. *The decision version of the PSPMM is NP-hard.*

Proof. The classical RCPSP is NP-hard [5]. A reduction from RCPSP to PSPMM is constructed.

Given a set of activities $j \in \mathcal{J}$, precedence relations $\mathcal{P}_j$, renewable resource constraints, activity durations p_j , renewable resource requirements $r_{j,k}$, renewable resource capacities R_k , and a project deadline D , does there exist a feasible schedule satisfying all precedence and renewable resource constraints such that

$$F_{n+1} \leq D?$$

Consider an arbitrary RCPSP decision instance consisting of:

- a set of activities $j \in \mathcal{J}$,
- precedence relations $\mathcal{P}_j \subseteq \mathcal{J}$,
- renewable resource types $k \in \mathcal{K}$,
- activity durations d_j ,
- renewable resource requirements $r_{j,k}$,
- renewable resource capacities R_k ,
- and a project deadline D .

The PSPMM instance becomes equivalent to the RCPSP instance. For each activity $j \in \mathcal{J}$, define exactly one execution mode: $\mathcal{M}_j = \{1\}$. Define $d_j^{(1)} = d_j$, $r_{j,k}^{(1)} = r_{j,k}$, and $c_j^{(1)} = 0$. Since every activity has only one execution mode, the durations remain fixed, the renewable resource requirements remain unchanged, and mitigation decisions are not possible. Therefore, the constructed PSPMM instance becomes equivalent to the classical RCPSP instance. Next, use the same project deadline D as in the RCPSP decision instance, choose $\alpha > 0$, and define the objective threshold as $K = 0$. Because each activity has only one execution mode and all mitigation costs satisfy $c_j^{(1)} = 0$, the mitigation cost component of the objective equals zero for every feasible solution. Therefore, the objective function reduces to $\alpha L + \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} c_j^{(m)} x_{j,t}^{(m)} = \alpha L$.

Now show that the transformation preserves the YES/NO status of the decision problem.

Forward Direction.

Suppose that the RCPSP instance is a YES instance. Then there exists a feasible schedule satisfying all precedence and renewable resource constraints such that $F_{n+1} \leq D$. Since the constructed PSPMM instance uses exactly the same activities, durations, precedence relations, and renewable resource requirements, the same schedule is also feasible for the PSPMM. Because the deadline is satisfied, $L = \max\{0, F_{n+1} - D\} = 0$. Therefore, the objective value becomes $\alpha L = 0 \leq K$. Hence, the constructed instance of the PSPMM is also a YES instance.

Backward Direction.

To prove the reverse implication, its contrapositive will be proved. Instead of directly proving 'constructed instance YES $\Rightarrow$ RCPSP instance YES', the logically equivalent statement 'RCPSP instance NO $\Rightarrow$ constructed instance NO' will be proved.

Suppose that the RCPSP instance is a NO instance. Then no feasible schedule exists satisfying all precedence and renewable resource constraints together with $F_{n+1} \leq D$. The constructed PSPMM instance contains exactly the same activities, durations, precedence relations, and renewable resource constraints. Therefore, any feasible solution of the constructed PSPMM instance with objective value at most $K = 0$ would imply $\alpha L \leq 0$. Since $\alpha > 0$, this requires $L = 0$, which implies $F_{n+1} \leq D$. Hence, any feasible solution of the constructed PSPMM instance would also be a feasible solution of the original RCPSP instance. Since the RCPSP instance is a NO instance, no such schedule exists. Therefore, the constructed PSPMM instance is also a NO instance.

The transformation only copies the original RCPSP data and selects the parameters α and K , which can clearly be done in polynomial time. Since the transformation is polynomial and the RCPSP decision problem is NP-hard, the decision version of the PSPMM is NP-hard.

□

4.2.5. NP-Completeness

Corollary 4.2.1. *The decision version of the PSPMM is NP-complete.*

Proof. Lemma 4.1 establishes that the decision version of the PSPMM belongs to NP. Theorem 4.2 shows that the PSPMM is NP-hard. Therefore, the decision version of the PSPMM is NP-complete. $\square$

4.3. Complexity Without Renewable Resource Constraints

4.3.1. Motivation

The previous section confirmed that the complete formulation is NP-hard. However, it is also important to determine whether the computational difficulty is mainly caused by the renewable resource constraints or whether the formulation remains difficult even after these constraints are removed.

To study this question, a PSPMM instance without renewable resource constraints (see Subsection 3.5.3) is considered. The PSPMM instance still contains precedence constraints, multiple execution modes, mode-dependent durations, mode-dependent costs, project deadlines, and tardiness penalties. Project tardiness is defined as $L = \max\{0, F_{n+1} - D\}$, where F_{n+1} denotes the project completion time and D the project deadline. The objective function minimizes both mitigation costs and project tardiness: $\min \left(\alpha L + \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_t c_j^{(m)} x_{j,t}^{(m)} \right)$. The parameter α determines the trade-off between the mitigation cost and the delay in the project.

Unlike the RCPSp, the DTCTP does not contain renewable resource constraints. Its computational difficulty originates from the combinatorial interaction between precedence constraints, discrete execution mode choices, project timing decisions, and mode-dependent costs [14]. Consequently, NP-hardness may persist even after renewable resource constraints are removed. Also, the DTCTP does not include tardiness penalties; therefore, in the reduction, tardiness is effectively neutralized by choosing α sufficiently large so that any delay becomes suboptimal.

4.3.2. Decision Version

Consider the decision version of the PSPMM instance in which renewable resource constraints are removed.

Given an instance of the PSPMM instance together with an objective threshold K , does there exist a feasible schedule satisfying:

- the mode selection constraints,
- the precedence constraints,
- and the tardiness constraints,

such that

$$\alpha L + \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} c_j^{(m)} x_{j,t}^{(m)} \leq K,$$

where

$$L = \max\{0, F_{n+1} - D\}?$$

4.3.3. Polynomial-Time Verification

Before proving NP-hardness, first show that feasible solutions of the PSPMM instance can be verified in polynomial time.

Lemma 4.3. *The decision version of the PSPMM instance lies in NP.*

Proof. A certificate consists of a selected execution mode for each activity, completion times for all activities and project tardiness L . Verification can be performed in polynomial time by checking the following linear constraints:

- whether exactly one execution mode is selected for each activity,
- whether all precedence constraints are satisfied,
- whether project tardiness satisfies $L \geq F_{n+1} - D$ and $L \geq 0$,
- and whether $\alpha L + \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_t c_j^{(m)} x_{j,t}^{(m)} \leq K$.

All checks require at most polynomial time in the number of activities and precedence relations. Therefore, the decision version of the PSPMM instance lies in NP. $\square$

4.3.4. NP-Hardness Result

The proof of NP-hardness is based on a polynomial-time reduction from the decision version of the DTCTP, which is known to be NP-hard [9]:

Given a DTCTP instance and a threshold K , does there exist a schedule that satisfies all precedence constraints with project cost $\leq K$ and project completion time $\leq D$?

The DTCTP contains the same principal components as the PSPMM instance, namely activities, precedence constraints, multiple execution modes, mode-dependent durations, and mode-dependent costs [14]. The main idea of the proof is that every DTCTP instance can be represented directly as an instance of the PSPMM instance, and that feasibility in both formulations corresponds under the construction.

Theorem 4.4. *The PSPMM instance without renewable resource constraints is NP-complete.*

Proof. Consider an arbitrary DTCTP decision instance consisting of:

- a set of activities $j \in J$,
- precedence relations $\mathcal{P}_j \subseteq J$,
- execution modes $m \in \mathcal{M}_j$,
- mode-dependent durations $d_j^{(m)}$,
- mode-dependent costs $c_j^{(m)}$,
- a project deadline D ,
- and a cost threshold K .

Construct a corresponding decision instance of the PSPMM instance by copying all activities, precedence relations, execution modes, durations, and costs. The project deadline is set to $D' = D$. Next, choose: $\alpha > \max \left\{ K, \sum_{j \in \mathcal{J}} \max_{m \in \mathcal{M}_j} c_j^{(m)} \right\}$. This choice ensures that one unit of project tardiness is more expensive than any possible reduction in mitigation cost. Consequently, the objective strongly penalizes delayed completion. Use the same cost threshold K for the PSPMM instance decision instance.

Suppose that the DTCTP instance is a YES instance. Then there exists a schedule satisfying all precedence constraints with project cost at most K and project completion time: $F_{n+1} \leq D$. Since the PSPMM instance uses the same activities, precedence relations, execution modes, durations, and costs, the same schedule is feasible for the PSPMM instance. Because the deadline is satisfied, project tardiness equals: $L = \max\{0, F_{n+1} - D\} = 0$. Therefore, the objective value reduces to: $\alpha \cdot 0 + \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_t c_j^{(m)} x_{j,t}^{(m)} = C$, where C denotes the total mitigation cost. Since $C \leq K$, the PSPMM instance instance is also a YES instance.

Suppose that the DTCTP instance is a NO instance. First, consider the case where no schedule satisfies the project deadline. Then every feasible solution satisfies the following: $F_{n+1} > D$, which implies: $L > 0$. Since time is discrete, deadline violations occur in integer time units and therefore: $L \geq 1$. The parameter α was chosen sufficiently large so that tardiness dominates mitigation cost. Consequently, every feasible solution exceeds the objective threshold K . Therefore, the PSPMM instance instance is also a NO instance. Now consider the case where schedules that satisfy the deadline exist, but all such schedules have cost greater than K . Solutions with $L = 0$ exceed the cost threshold K , while

solutions with $L > 0$ incur an additional tardiness penalty. Consequently, all feasible solutions violate the objective threshold. Therefore, the PSPMM instance is again a NO instance.

The transformation only copies input data and selects α based on the maximum cost value, which can be done in polynomial time. Since the DTCTP is NP-hard and the construction is a polynomial-time reduction preserving YES/NO equivalence, the reduced PSPMM is NP-hard. Combined with the lemma from Subsection 4.3.3, it follows that the decision version of the PSPMM instance is NP-complete. $\square$

4.3.5. NP-Completeness

Corollary 4.4.1. *The reduced PSPMM without renewable resource constraints is NP-complete.*

Proof. The reduced PSPMM is NP-hard by the preceding theorem and belongs to NP by Lemma 4.3. Therefore, the decision version of the reduced PSPMM is NP-complete. $\square$

4.4. Discussion and Summary

This chapter analyzed the computational complexity of the Project Scheduling Problem with Mitigation Measures (PSPMM). First, the complete PSPMM was studied. A polynomial-time reduction from the classical RCPSP established NP-hardness, while a polynomial-time verification procedure showed that the decision version belongs to NP. Consequently, the PSPMM is NP-complete.

The analysis was then repeated for a reduced version of the PSPMM in which renewable resource constraints were removed. Through a polynomial-time reduction from the Discrete Time–Cost Trade-Off Problem (DTCTP), this reduced problem was also shown to be NP-complete.

These results demonstrate that the computational difficulty of the PSPMM is not solely caused by renewable resource constraints. The source of complexity lies in the combination of precedence constraints, mode selection, and time-dependent cost structures rather than resource allocation alone. This explains the difficulty of solving large instances exactly and motivates the computational methods used in the next chapter.

5

Computational Implementation and Experimental Framework

Chapter 4 examined the theoretical complexity of the PSPMM formulation. Although complexity analysis provides insight into the expected difficulty of the problem, the practical performance of a formulation can only be evaluated through computational implementation and experimental testing.

This chapter describes the implementation framework used to evaluate the PSPMM formulation. The model was implemented in Python using the Pyomo optimization framework [13, 7] and solved using the Gurobi Optimizer [12]. In addition, a comparable start-time indexed formulation based on Bianco–Caramia [4] was implemented to provide a benchmark for computational evaluation. The chapter further describes the benchmark instances, the generation of renewable resource data, and the computational environment used in the experimental study. Particular attention is given to implementation issues arising from the large size of time-indexed formulations and to the preprocessing techniques introduced to improve computational tractability. Finally, the chapter presents the implementation versions considered in the computational study and defines the evaluation framework used for the numerical experiments reported in Chapter 6.

5.1. Implementation Framework

All optimization models were implemented in Python using the Pyomo optimization modeling framework [13, 7]. Pyomo was selected because it provides a flexible algebraic modeling environment for mixed-integer programming problems and supports integration with commercial optimization solvers. The optimization models were solved using the Gurobi Optimizer [12], which was selected because of its strong computational performance on large-scale mixed-integer programming problems and its compatibility with Pyomo. The computational experiments were executed on a system equipped with:

- AMD Ryzen 3 5400U processor,
- 8 logical CPU threads,
- Windows 11 operating system,
- Python 3.12,
- Pyomo,
- Gurobi 13.0.2.

5.1.1. Implementation of the PSPMM formulation

The PSPMM formulation introduced in Chapter 3 was implemented directly in Pyomo. The implementation follows the mathematical model closely by defining indexed binary decision variables, objective functions, and constraints using algebraic expressions.

The binary decision variable is defined as

$$x_{j,t}^{(m)} = \begin{cases} 1 & \text{if activity } j \text{ finishes at time } t \text{ in mode } m, \\ 0 & \text{otherwise.} \end{cases}$$

The objective function minimizes the combined mitigation cost and project tardiness:

$$\min \left(\alpha L + \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} c_j^{(m)} x_{j,t}^{(m)} \right).$$

The implementation includes all constraints presented in Chapter 3, namely:

- activity execution constraints,
- finish-time definition constraints,
- precedence constraints,
- renewable resource constraints,
- tardiness constraints.

Finish times are represented through Pyomo expressions, while the precedence and resource constraints are generated dynamically using constraint lists. The implementation therefore preserves the exact mathematical structure of the PSPMM formulation while enabling efficient model generation and solution through the Gurobi solver. Because the PSPMM is formulated as a time-indexed mixed-integer program, a finite planning horizon must be specified before the binary variables can be generated. In this study, the planning horizon is defined as

$$UB = CP_{\max},$$

where $CP_{\max}$ denotes the duration of the critical path when all activities are executed in their most time consuming and thus least expensive execution modes. This value provides a valid upper bound on the project completion time for the benchmark instances considered in this study and is used to define the set of discrete time periods $T = 0, 1, \dots, UB$. Consequently, binary variables are generated only for completion times within this planning horizon.

5.2. Lexicographic Optimization Procedure

A lexicographic optimization procedure solves a multi-objective optimization problem by optimizing objectives sequentially according to a predefined priority order [19]. In this study, total project cost is treated as the primary objective, while project makespan is optimized as a secondary objective without allowing any deterioration in the optimal cost value.

The PSPMM formulation uses a lexicographic optimization strategy consisting of two sequential optimization phases. This approach allows the model to first optimize the primary objective and subsequently improve schedule compactness without worsening the original objective value. In the first phase, the combined objective consisting of mitigation cost and tardiness is minimized:

$$\min \left(\alpha L + \sum c_j^{(m)} x_{j,t}^{(m)} \right)$$

After obtaining the best objective value, a second optimization problem is solved. The original objective value is fixed using an additional constraint, while the project makespan is minimized:

$$\min F_n$$

subject to:

$$\alpha L + \sum c_j^{(m)} x_{j,t}^{(m)} \leq Z^*$$

where Z^* denotes the optimal objective value obtained during the first optimization phase.

5.3. Implementation Versions

Four implementation versions were developed during the computational study. The versions differ in the application of preprocessing techniques and lexicographic optimization.

Version	Preprocessing	Lexicographic optimization
V1	No	No
V2	No	Yes
V3	Yes	No
V4	Yes	Yes

Table 5.1: Implementation versions considered in the computational study

Version 1 represents the baseline implementation and corresponds most closely to the original mathematical formulation. Version 2 was introduced to isolate the effect of lexicographic optimization. Version 3 extends the baseline implementation with the preprocessing procedures described in Section 5.5, while Version 4 combines preprocessing and lexicographic optimization.

The preprocessing framework used in Versions 3 and 4 consists of sparse variable generation, earliest-time preprocessing, and resource activity preprocessing. During the development phase, an additional horizon reduction procedure was investigated. However, computational experiments showed that restricting the planning horizon could exclude schedules that yield superior objective values. Consequently, horizon reduction was removed from the final implementation and is not included in the reported computational results.

The computational analysis focuses primarily on Versions 1, 3 and 4. Version 2 was mainly used during implementation to verify the isolated effect of lexicographic optimization and is therefore omitted from the main comparison. The selected versions allow the individual influence of preprocessing and the combined influence of preprocessing and lexicographic optimization to be evaluated.

5.4. Dataset Generation

5.4.1. Source Data

The computational experiments are based on benchmark instances derived from the discrete time–cost trade-off problem (DTCTP) test beds proposed by Akkan, Drexler, and Kimms [1]. These benchmark datasets provide project networks with precedence relations and multiple execution modes for each activity.

5.4.2. Generation Procedure

Since the original DTCTP benchmark instances do not contain renewable resource information, additional renewable resource constraints were generated synthetically to extend the problem into a resource-constrained project scheduling environment. Two renewable resources (R_1 and R_2) were generated for each benchmark instance. Resource usage values were assigned proportionally to activity durations, such that activities with shorter durations generally require larger amounts of resources. This reflects the acceleration effects commonly observed in discrete time–cost trade-off problems, where reducing activity duration is associated with increased resource consumption. The generated resource data were used consistently across all computational experiments and for both formulations considered in this study.

Since the original DTCTP instances do not contain renewable resource capacities, capacities were generated based on the resource requirements assigned to the activity modes. For each renewable resource k , the capacity was calculated as

$$R_k = \left\lceil \gamma \cdot \max(r_{j;k}^{(m)}) \right\rceil,$$

where $r_{j;k}^{(m)}$ denotes the requirement of resource k for activity j executed in mode m , and γ is a scaling parameter that controls the tightness of the resource capacities. Larger values of γ lead to higher resource capacities and therefore less competition for resources, while smaller values result in tighter capacities and stronger competition between activities. In this study, the value $\gamma = 8$ was used for all generated instances. The resulting capacities are therefore eight times the largest resource requirement observed for the corresponding resource within an instance. The value $\gamma = 8$ was selected based on preliminary testing. It provides sufficient resource capacity to maintain feasibility while still limiting the simultaneous execution of activities and creating resource competition within the project schedule.

Preliminary experiments on the original benchmark instances resulted in large optimization models and long solution times. To obtain a computationally manageable test set while preserving the essential structural characteristics of the project networks, reduced instances containing 20 activities were generated. The reduction procedure preserved:

- dummy start and end activities,
- precedence feasibility,
- network connectivity,
- resource feasibility.

As a result, the reduced instances maintain the fundamental scheduling characteristics of the original datasets while allowing extensive computational experimentation.

5.4.3. Instance Classification

The final dataset consists of 72 benchmark instances, labeled (I1, I2, ..., I72). To facilitate structured analysis, the instances were grouped according to their time–cost trade-off characteristics and resource-capacity configurations:

- M1–CCV,
- M1–CVX,
- M1–HYB,
- M2–CCV,
- M2–CVX,
- M2–HYB.

The labels CCV, CVX, and HYB denote concave, convex, and hybrid time–cost trade-off structures, respectively, while M1 denotes single-mode instances and M2 denotes multi-mode instances with two execution modes per activity. In a concave (CCV) time–cost trade-off structure, initial reductions in activity duration can be achieved at relatively low additional cost, whereas further reductions become increasingly expensive. Conversely, in a convex (CVX) structure, the first reductions are relatively expensive while later reductions require smaller incremental cost increases. Hybrid (HYB) instances contain a mixture of both patterns across activities. This classification enables the computational results to be analyzed both at the individual-instance level and at the aggregated instance-set level.

5.5. Preprocessing Techniques

Both the PSPMM formulation and the Bianco–Caramia formulation belong to the class of time-indexed mixed-integer programming models. A common characteristic of such formulations is the potentially large number of binary variables generated by the combination of activities, execution modes, and time periods.

Preliminary experiments revealed that both formulations generate a large number of binary variables and resource constraints. Since the size of a time-indexed mixed-integer programming model depends directly on the planning horizon and the number of feasible activity–time combinations, model size can increase rapidly even for moderately sized instances. To improve computational tractability, several preprocessing techniques were developed and incorporated into the implementation. These techniques reduce model size and model-generation effort without altering the feasible solution space of

the scheduling problem. Only preprocessing procedures that preserve optimal solutions are retained in the final implementation framework.

5.5.1. Sparse Variable Generation

Without preprocessing, binary variables are generated for all feasible combinations of activities, modes, and time periods within the planning horizon. This approach creates many variables that are theoretically possible but highly unlikely to appear in an optimal schedule. For example, if an activity requires five time units to complete, it cannot finish at times 1, 2, 3, or 4. To reduce model size, Versions 3 and 4 generate only variables that satisfy basic feasibility conditions. For the PSPMM formulation, binary variables are created only for finish times satisfying

$$t \geq d_j^{(m)},$$

where $d_j^{(m)}$ denotes the duration of activity j executed in mode m . In addition, dummy activities are assigned a single feasible time period. Since these activities have zero duration and serve only to represent the start and completion of the project network, generating multiple time assignments for them provides no additional information. By excluding activity–time combinations that can never occur in a feasible schedule, sparse variable generation substantially reduces the number of binary variables and the memory required for model construction.

5.5.2. Earliest-Time Preprocessing

A further reduction is achieved by exploiting precedence information. Before generating binary variables, the earliest possible finish time of each activity is computed using the shortest execution mode for all activities. Let EF_j denote the earliest finish time of activity j . It is computed recursively as

$$EF_j = \begin{cases} \min_{m \in \mathcal{M}_j} d_j^{(m)}, & \text{if } \mathcal{P}_j = \emptyset \\ \max_{h \in P_j} EF_h + \min_{m \in \mathcal{M}_j} d_j^{(m)}, & \text{otherwise} \end{cases}$$

where P_j denotes the set of predecessors of activity j . The value EF_j represents the earliest possible completion time of activity j assuming that all predecessor activities start as early as possible and are executed in their shortest available modes. Binary variables are then generated only for finish times satisfying

$$t \geq EF_j.$$

Any finish time occurring before EF_j is impossible because at least one predecessor activity would not yet have been completed. Such time assignments can therefore be excluded from the model without removing any feasible schedules. This preprocessing step reduces the number of binary variables while preserving all feasible schedules.

5.5.3. Resource Activity Preprocessing

The resource constraints require identification of all activities that are active during each time period. Constructing these constraints directly within the optimization model would repeatedly evaluate the same activity–time relationships, leading to unnecessary computational overhead. To avoid this issue, Version 3 and 4 precompute the set of active activity–mode–time combinations for every time period. For the finish-time indexed formulation, an activity j executed in mode m and finishing at time τ is active during all time periods from $\tau - d_j^{(m)}$ up to, but not including, τ . Rather than repeatedly determining these active periods when generating the resource constraints, the relationships are calculated once in advance. Let

$$\mathcal{I}_t = \left\{ (j, m, \tau) \mid \tau - d_j^{(m)} \leq t < \tau \right\},$$

denote the set of feasible activity–mode–finish-time combinations that are active during time period t . The resource constraints can then be generated using these precomputed sets:

$$\sum_{(j,m,\tau) \in \mathcal{I}_t} r_{j;k}^{(m)} x_{j,\tau}^{(m)} \leq R_k, \quad \forall k \in \mathcal{K}, \forall t \in \mathcal{T},$$

where $r_{j;k}^{(m)}$ denotes the resource consumption of activity j in mode m for resource k , and R_k denotes the corresponding resource capacity. This preprocessing step does not modify the mathematical formulation or the feasible solution space. Instead, it improves implementation efficiency by storing activity–resource relationships in advance. Consequently, the resource constraints can be generated more efficiently, reducing model construction time and avoiding repeated evaluation of the same activity–time relationships.

5.5.4. Summary of Preprocessing Techniques

The preprocessing framework implemented in Versions 3 and 4 aims to improve computational efficiency without altering the feasible solution space of the optimization problem. Sparse variable generation removes activity–time assignments that can never occur in a feasible schedule, earliest-time preprocessing exploits precedence relations to eliminate impossible completion times, and resource activity preprocessing accelerates model construction by precomputing activity–resource relationships. Unlike heuristic model reductions, these preprocessing procedures preserve all feasible schedules and therefore maintain optimality. Their primary purpose is to reduce model size, memory consumption, and model-generation effort while leaving the underlying optimization problem unchanged.

The impact of these preprocessing techniques is evaluated experimentally in Chapter 6 through a comparison of Versions 1, 3, and 4 for both the PSPMM formulation and the Bianco–Caramia formulation.

5.6. Comparable Formulation

To evaluate the effectiveness of the PSPMM formulation, a comparison is made with a start-time indexed formulation inspired by the work of Bianco–Caramia [4]. The formulation of Bianco–Caramia was selected because it represents a well-established time-indexed mixed-integer programming approach for project scheduling problems. Similar to the PSPMM formulation, it uses binary variables indexed by activities, modes and time periods. However, the two formulations differ in their temporal representation. The PSPMM formulation uses finish-time indexed variables:

$$x_{j,t}^{(m)} = \begin{cases} 1 & \text{if activity } j \text{ finishes at time } t \text{ in mode } m, \\ 0 & \text{otherwise,} \end{cases}$$

whereas the Bianco–Caramia formulation uses start-time indexed variables:

$$x_{j,t}^{(m)} = \begin{cases} 1 & \text{if activity } j \text{ starts at time } t \text{ in mode } m, \\ 0 & \text{otherwise.} \end{cases}$$

Although both formulations represent the same scheduling problem, they construct precedence and resource constraints differently due to their alternative time-indexing structures. The implementation of the Bianco–Caramia formulation follows the same problem definition, objective function and dataset structure used for the PSPMM formulation. Consequently, any observed differences in computational performance can be attributed primarily to formulation structure rather than differences in problem data.

To ensure a fair comparison, both formulations were implemented in Pyomo, solved using Gurobi, and executed on identical hardware under the same solver settings. To analyze the influence of implementation choices, four implementation versions were developed during the computational study. The same implementation strategy was applied to both formulations. In particular, all preprocessing procedures described in Section 5.5 were implemented for both formulations. Consequently, any differences observed in the computational results can be attributed to differences in formulation structure rather than unequal implementation choices.

The computational experiments reported in Chapter 6 compare these implementation versions to evaluate the individual contributions of preprocessing and lexicographic optimization, as well as their combined effect on computational performance.

5.7. Experimental Design

The computational experiments were designed to evaluate the performance of the PSPMM formulation and the comparable Bianco–Caramia start-time indexed formulation. All benchmark instances were solved using both formulations under the same computational conditions. The same hardware environment, solver settings, objective function, and benchmark datasets were used for both models. This ensures that any observed differences are caused by the formulation structure rather than by differences in implementation.

The main comparison is based on Version 4, which represents the final implementation used in this research. Version 4 includes both the preprocessing techniques described in Section 5.5 and the lexicographic optimization procedure presented in Section 5.2. To better understand the influence of the implementation choices, additional experiments were conducted using Versions 1 and 3. These versions were used to evaluate the individual effects of preprocessing and lexicographic optimization before the final implementation was selected. The experiments were carried out on all generated benchmark instances described earlier in this chapter. Results are reported both for individual instances and for aggregated instance sets in order to identify overall performance trends across different problem categories.

5.8. Evaluation Metrics and Analysis Framework

The computational results are evaluated using two groups of performance measures: solution quality metrics and computational efficiency metrics.

The first group of measures evaluates the quality of the schedules produced by the optimization models. For each benchmark instance, the following indicators are recorded:

- objective value;
- project makespan;
- project tardiness;
- solver termination status.

The objective value represents the trade-off between mitigation cost and project tardiness. The makespan measures the total project duration, while tardiness indicates how much the project completion time exceeds the specified deadline. The solver termination status is recorded to verify whether an optimal solution was obtained within the available solution time.

The second group of measures evaluates the computational performance of the formulations. For each benchmark instance, the following indicators are collected:

- solver runtime;
- number of binary variables;
- number of constraints.

Runtime measures the computational effort required to solve the model. The numbers of binary variables and constraints provide information about model size and complexity. These measures are particularly important because the main objective of this research is to investigate how formulation structure and preprocessing techniques influence computational tractability.

The results are analysed at two levels. First, individual-instance results are examined and reported in the appendices. These results provide detailed information about the behaviour of the formulations on specific benchmark instances. Second, aggregated results are calculated for each instance set. Averaging the results across instance categories makes it easier to identify general trends and compare the overall performance of the different formulations and implementation versions. The detailed analyses based on these metrics are presented in Chapter 6.

5.9. Summary

This chapter presented the computational implementation and experimental framework used to evaluate the PSPMM formulation. The optimization models were implemented in Python using the Pyomo optimization framework and solved using the Gurobi Optimizer. In addition, several preprocessing techniques were introduced to improve computational tractability while preserving the feasible solution space of the scheduling problem. During preliminary development, a horizon reduction procedure was investigated but was not retained because it could adversely affect solution quality. The benchmark instances were derived from DTCTP datasets and extended with renewable resource constraints and generated resource-capacity structures. Furthermore, a benchmark formulation inspired by Bianco–Caramia was implemented to provide a computational comparison with the PSPMM formulation. To ensure methodological fairness, identical preprocessing and reduction procedures were applied to both formulations before the computational comparison was conducted. The next chapter presents the computational results and comparative performance analysis of the implemented formulations.

6

Computational Results and Analysis

This chapter presents the computational results obtained from the experimental evaluation of the PSPMM finish-time indexed formulation and the comparable start-time indexed formulation inspired by Bianco and Caramia [4]. The experimental framework described in Chapter 5 was applied to a set of 72 benchmark instances derived from DTCTP datasets and extended with renewable resource constraints. The objective of the computational study is to evaluate both solution quality and computational efficiency while assessing the impact of preprocessing techniques and lexicographic optimization.

The analysis is organized into three stages. First, Versions 1 and 3 are compared to evaluate the effect of the preprocessing techniques introduced in Section 5.5. This comparison focuses on the resulting changes in model size and computational performance. Second, Versions 3 and 4 are compared to assess the impact of the lexicographic optimization procedure. Since both versions employ the same preprocessing framework and generate the same model structure, any differences in performance can be attributed to the additional optimization phase introduced in Version 4. Finally, the PSPMM formulation and the Bianco–Caramia formulation are compared using the final implementations (Version 4). Both formulations are evaluated under the same experimental framework, including the same preprocessing procedures and lexicographic optimization strategy. This comparison aims to determine whether the different time-indexing approaches influence computational performance while maintaining comparable solution quality. Detailed computational results for all benchmark instances are provided in Appendix B and Appendix C.

6.1. Generated Test Instances

The computational experiments were conducted using the benchmark instances generated according to the procedure described in Section 5.4. The final dataset consists of 72 resource-constrained project scheduling instances containing 20 activities each. The instances are grouped according to their time–cost trade-off characteristics and the number of execution modes available for each activity. Detailed characteristics of the generated benchmark instances are provided in Appendix A. Table 6.1 summarizes the average structural characteristics of the six instance groups.

Instance Set	Instances	Activities	Avg. Precedence	Avg. Modes	Avg. Cap R1	Avg. Cap R2
M1–CCV	12	20	42.00	53.50	69.33	53.33
M1–CVX	12	20	45.75	54.25	52.00	51.33
M1–HYB	12	20	43.17	53.58	53.33	53.33
M2–CCV	12	20	44.67	56.00	16.00	8.00
M2–CVX	12	20	42.75	56.00	16.00	8.00
M2–HYB	12	20	45.00	56.00	16.67	8.67

Table 6.1: Average characteristics of the generated instance groups. The columns Avg. Cap R1 and Avg. Cap R2 represent the average capacities of renewable resources R_1 and R_2 .

The following sections first examine the influence of preprocessing and lexicographic optimization before comparing the final implementations of the PSPMM formulation and the Bianco–Caramia formulation.

6.2. Influence of Preprocessing Techniques

The first objective of the computational study is to evaluate the effectiveness of the preprocessing techniques introduced in Chapter 5. To isolate their impact, Version 1 and Version 3 are compared for both formulations. Version 1 represents the baseline implementation and does not include any preprocessing techniques. The full scheduling horizon is used, all feasible time-index combinations are generated explicitly, and no additional reduction mechanisms are applied during model construction. Version 3 incorporates the preprocessing techniques discussed in Section 5.5. These include earliest-time feasibility filtering, sparse feasible variable generation and resource preprocessing. These preprocessing techniques reduce model size and model-generation effort without modifying the feasible solution space of the scheduling problem. Consequently, any differences between Versions 1 and 3 are expected to appear primarily in computational efficiency rather than solution quality.

6.2.1. Impact on Solution Quality

Tables 6.2 and 6.3 present the average objective values, makespans, and tardiness values obtained by Versions 1 and 3 for the PSPMM formulation and the Bianco–Caramia formulation, respectively.

Instance Set	Version	Avg. Objective	Avg. Makespan	Avg. Tardiness
M1–CCV	V1	101.91	260.17	37.25
M1–CCV	V3	101.91	260.17	37.25
M1–CVX	V1	45.71	253.92	37.58
M1–CVX	V3	45.71	253.92	37.58
M1–HYB	V1	74.81	251.58	34.00
M1–HYB	V3	74.81	251.58	34.00
M2–CCV	V1	63.63	281.33	19.83
M2–CCV	V3	63.63	281.33	19.83
M2–CVX	V1	24.26	291.50	17.58
M2–CVX	V3	24.26	291.50	17.58
M2–HYB	V1	53.29	313.83	21.42
M2–HYB	V3	53.29	313.83	21.42

Table 6.2: Average objective value, makespan, and tardiness obtained by PSPMM Versions 1 and 3 for each instance set.

Instance Set	Version	Avg. Objective	Avg. Makespan	Avg. Tardiness
M1–CCV	V1	101.91	260.17	37.25
M1–CCV	V3	101.91	260.17	37.25
M1–CVX	V1	45.71	253.92	37.58
M1–CVX	V3	45.71	253.92	37.58
M1–HYB	V1	74.81	251.58	34.00
M1–HYB	V3	74.81	251.58	34.00
M2–CCV	V1	63.63	281.33	19.83
M2–CCV	V3	63.63	281.33	19.83
M2–CVX	V1	24.26	291.50	17.58
M2–CVX	V3	24.26	291.50	17.58
M2–HYB	V1	53.29	313.83	21.42

M2-HYB	V3	53.29	313.83	21.42
--------	----	-------	--------	-------

Table 6.3: Average objective value, makespan, and tardiness obtained by the Bianco–Caramia formulation for Versions 1 and 3.

The results demonstrate that preprocessing has no influence on solution quality. For every instance category, Version 3 produces exactly the same average objective values, makespans, and tardiness values as Version 1. This outcome is consistent across both formulations. The identical results confirm that the preprocessing techniques preserve the feasible solution space and do not remove any schedules that could be part of an optimal solution. Sparse variable generation eliminates activity–time assignments that are infeasible by construction, earliest-time preprocessing removes completion times that violate precedence relationships, and resource activity preprocessing only affects the way resource constraints are generated. None of these procedures alters the underlying optimization problem.

Therefore, these findings validate the preprocessing techniques as an exact implementation improvement. The computational benefits obtained through preprocessing are achieved without any difference in objective value, project duration, or tardiness performance.

6.2.2. Impact on Computational Efficiency

Tables 6.4 and 6.5 present the average runtime, binary variables and constraints obtained by Versions 1 and 3 for the PSPMM formulation and the Bianco–Caramia formulation, respectively.

Instance Set	Version	Avg. Runtime (s)	Avg. Binary Variables	Avg. Constraints
M1-CCV	V1	2.42	14497.17	643.00
M1-CCV	V3	4.82	10410.50	643.00
M1-CVX	V1	2.05	13554.92	596.00
M1-CVX	V3	3.39	9522.08	596.00
M1-HYB	V1	1.83	13068.83	593.58
M1-HYB	V3	2.92	9751.50	593.58
M2-CCV	V1	3.27	14484.67	624.92
M2-CCV	V3	3.14	8476.33	624.92
M2-CVX	V1	3.47	15181.75	647.33
M2-CVX	V3	3.24	9331.33	647.33
M2-HYB	V1	4.19	15778.75	679.33
M2-HYB	V3	3.17	9440.83	679.33

Table 6.4: Average runtime, number of binary variables, and number of constraints for PSPMM Versions 1 and 3.

Instance Set	Version	Avg. Runtime (s)	Avg. Binary Variables	Avg. Constraints
M1-CCV	V1	9.25	14497.17	663.00
M1-CCV	V3	9.91	10103.00	663.00
M1-CVX	V1	7.40	13554.92	616.00
M1-CVX	V3	6.38	9307.00	616.00
M1-HYB	V1	6.46	13068.83	613.58
M1-HYB	V3	5.23	9413.08	613.58
M2-CCV	V1	10.34	14484.67	644.92
M2-CCV	V3	3.91	8438.50	644.92
M2-CVX	V1	10.45	15181.75	667.33
M2-CVX	V3	3.95	9228.83	667.33

M2-HYB	V1	11.19	15778.75	699.33
M2-HYB	V3	4.41	9256.33	699.33

Table 6.5: Average runtime, number of binary variables, and number of constraints for the Bianco–Caramia formulation in Versions 1 and 3.

The most noticeable effect of preprocessing is the reduction in the number of binary variables. Across all instance categories, Version 3 generates substantially fewer variables than Version 1 for both formulations. This reduction is achieved through sparse variable generation and earliest-time preprocessing, which eliminate activity–time combinations that cannot occur in a feasible schedule. The number of constraints, however, remains unchanged. The preprocessing procedures do not alter the mathematical formulation itself. Instead, they reduce the number of decision variables within the existing model structure. As a result, both versions contain the same precedence, resource, and tardiness constraints.

The impact on runtime is less straightforward. For the Bianco–Caramia formulation, preprocessing generally improves computational performance, particularly for the multi-mode instance sets. The reduction in binary variables appears to compensate for the additional effort required during preprocessing, leading to shorter solution times overall. For the PSPMM formulation, the runtime improvements are less consistent. Although the number of binary variables is reduced considerably, the M1 instance sets show slightly longer runtimes after preprocessing. In contrast, all M2 instance sets solve faster with preprocessing. This pattern suggests that the benefits of preprocessing become more pronounced as the scheduling problem becomes more complex. For relatively small M1 instances, the additional calculations required during preprocessing may outweigh the computational savings obtained during optimization. However, for more complex M2 instances, the reduction in model size appears to compensate for this overhead, resulting in shorter solution times.

Overall, the preprocessing framework successfully reduces model size while preserving solution quality. The results indicate that preprocessing is particularly valuable for more challenging instances, where the reduction in binary variables can lead to noticeable improvements in computational performance.

6.3. Influence of Lexicographic Optimization

Preliminary experiments were conducted using Version 2 to isolate the effect of the lexicographic optimization procedure. Since the observed effects were limited and the version served primarily as an implementation validation step, detailed results are omitted. The main analysis therefore focuses on Versions 1, 3, and 4.

The second objective of the computational study is to evaluate the influence of the lexicographic optimization procedure introduced in Section 5.3. To isolate this effect, Version 3 and Version 4 are compared. Both versions employ the same preprocessing procedures and generate the same mathematical model structure. The only difference is that Version 4 performs a second optimization phase after obtaining the optimal value of the primary objective. During this second phase, the objective value obtained in the first optimization is fixed, and the project makespan is minimized among all solutions that achieve the same objective value.

6.3.1. Impact on Solution Quality

Tables 6.6 and 6.7 compare the solution quality metrics obtained by Versions 3 and 4.

Instance Set	Version	Avg. Objective	Avg. Makespan	Avg. Tardiness
M1-CCV	V3	101.91	260.17	37.25
M1-CCV	V4	101.91	260.17	37.25
M1-CVX	V3	45.71	253.92	37.58
M1-CVX	V4	45.71	253.92	37.58
M1-HYB	V3	74.81	251.58	34.00
M1-HYB	V4	74.81	251.58	34.00
M2-CCV	V3	63.63	281.33	19.83

M2-CCV	V4	63.63	281.33	19.83
M2-CVX	V3	24.26	291.50	17.58
M2-CVX	V4	24.26	291.50	17.58
M2-HYB	V3	53.29	313.83	21.42
M2-HYB	V4	53.29	313.83	21.42

Table 6.6: Average objective value, makespan, and tardiness for PSPMM Versions 3 and 4, illustrating the effect of lexicographic optimization.

Instance Set	Version	Avg. Objective	Avg. Makespan	Avg. Tardiness
M1-CCV	V3	101.91	260.17	37.25
M1-CCV	V4	101.91	260.17	37.25
M1-CVX	V3	45.71	253.92	37.58
M1-CVX	V4	45.71	253.92	37.58
M1-HYB	V3	74.81	251.58	34.00
M1-HYB	V4	74.81	251.58	34.00
M2-CCV	V3	63.63	281.33	19.83
M2-CCV	V4	63.63	281.33	19.83
M2-CVX	V3	24.26	291.50	17.58
M2-CVX	V4	24.26	291.50	17.58
M2-HYB	V3	53.29	313.83	21.42
M2-HYB	V4	53.29	313.83	21.42

Table 6.7: Average objective value, makespan, and tardiness for Bianco-Caramia formulation Versions 3 and 4, illustrating the effect of lexicographic optimization.

The results indicate that lexicographic optimization has no observable influence on solution quality for the benchmark instances considered in this study. For both the PSPMM formulation and the Bianco-Caramia formulation, Versions 3 and 4 produce identical objective values, makespans, and tardiness values across all instance categories. This outcome suggests that the solutions obtained during the first optimization phase are already lexicographically optimal with respect to project makespan. In other words, once the primary objective has been optimized, no alternative schedules with the same objective value and a shorter makespan are available. As a result, the second optimization phase does not identify any further improvements.

Overall, the results show that lexicographic optimization provides no measurable benefit for the generated benchmark instances. Objective values, makespans, and tardiness values remained unchanged after the second optimization phase. This suggests that the optimal solutions obtained in the first phase were already unique, or that no alternative schedules existed with the same objective value and lower makespan. Consequently, the additional optimization phase increased computational effort without improving the final solution quality.

6.3.2. Impact on Computational Efficiency

Tables 6.8 and 6.9 compare the computational efficiency results obtained for Versions 3 and 4.

Instance Set	Version	Avg. Runtime (s)	Avg. Binary Variables	Avg. Constraints
M1-CCV	V3	4.82	10410.50	643.00
M1-CCV	V4	9.58	10410.50	644.00
M1-CVX	V3	3.39	9522.08	596.00
M1-CVX	V4	6.87	9522.08	597.00

M1–HYB	V3	2.92	9751.50	593.58
M1–HYB	V4	6.11	9751.50	594.58
M2–CCV	V3	3.14	8476.33	624.92
M2–CCV	V4	6.26	8476.33	625.92
M2–CVX	V3	3.24	9331.33	647.33
M2–CVX	V4	6.58	9331.33	648.33
M2–HYB	V3	3.17	9440.83	679.33
M2–HYB	V4	6.97	9440.83	680.33

Table 6.8: Average runtime, number of binary variables, and number of constraints for PSPMM formulation in Versions 3 and 4.

Instance Set	Version	Avg. Runtime (s)	Avg. Binary Variables	Avg. Constraints
M1–CCV	V3	9.91	10103.00	663.00
M1–CCV	V4	15.31	10103.00	664.00
M1–CVX	V3	6.38	9307.00	616.00
M1–CVX	V4	12.18	9307.00	617.00
M1–HYB	V3	5.23	9413.08	613.58
M1–HYB	V4	10.47	9413.08	614.58
M2–CCV	V3	3.91	8438.50	644.92
M2–CCV	V4	7.07	8438.50	645.92
M2–CVX	V3	3.95	9228.83	667.33
M2–CVX	V4	8.16	9228.83	668.33
M2–HYB	V3	4.41	9256.33	699.33
M2–HYB	V4	9.06	9256.33	700.33

Table 6.9: Average runtime, number of binary variables, and number of constraints for Bianco–Caramia formulation in Versions 3 and 4.

Since both versions use identical preprocessing techniques, the numbers of binary variables remain exactly unchanged. The number of constraints increases uniformly by exactly 1.00 across all instance groups in Version 4 due to the addition of the boundary constraint required to freeze the primary objective function value before initiating the second search phase. The main difference appears in the solver runtime. Version 4 requires a second optimization pass after solving the original model to minimize the project makespan. As a result, there is a clear increase in computational overhead for both models. For the Bianco–Caramia formulation, this multi-stage execution adds an extra burden, with average runtimes increasing by about 4 to 5 seconds across the benchmark sets. The PSPMM formulation handles this process similarly, showing a runtime increase of roughly 3 to 4 seconds. The results therefore show that lexicographic optimization systematically increases computational effort while offering only limited improvements in solution quality for these benchmark instances.

6.4. Comparison of Formulations

After evaluating the effects of preprocessing and lexicographic optimization individually, the final step is to compare the PSPMM formulation with the Bianco–Caramia start-time indexed formulation. To ensure a fair comparison, Version 4 is used for both formulations. Consequently, both implementations are evaluated using the same preprocessing framework and the same lexicographic optimization procedure. Although the preprocessing procedures follow the same principles, their implementation differs slightly because of the different time-indexing structures of the formulations. Any remaining differences therefore primarily reflect differences in the formulations themselves rather than differences in the overall solution approach.

6.4.1. Solution Quality Comparison

Table 6.10 presents the average objective values, makespans, and tardiness values obtained by both formulations using Version 4.

Instance Set	Formulation	Avg. Objective	Avg. Makespan	Avg. Tardiness
M1-CCV	PSPMM	101.91	260.17	37.25
M1-CCV	Bianco-Caramia	101.91	260.17	37.25
M1-CVX	PSPMM	45.71	253.92	37.58
M1-CVX	Bianco-Caramia	45.71	253.92	37.58
M1-HYB	PSPMM	74.81	251.58	34.00
M1-HYB	Bianco-Caramia	74.81	251.58	34.00
M2-CCV	PSPMM	63.63	281.33	19.83
M2-CCV	Bianco-Caramia	63.63	281.33	19.83
M2-CVX	PSPMM	24.26	291.50	17.58
M2-CVX	Bianco-Caramia	24.26	291.50	17.58
M2-HYB	PSPMM	53.29	313.83	21.42
M2-HYB	Bianco-Caramia	53.29	313.83	21.42

Table 6.10: Average objective value, makespan, and tardiness obtained by the PSPMM and Bianco-Caramia formulations using Version 4.

The results show that both formulations produce identical solution quality across all benchmark categories. For every instance set, the PSPMM formulation and the Bianco-Caramia formulation obtain the same average objective values, makespans, and tardiness values. These results confirm that both formulations represent the same underlying scheduling problem and identify the same optimal schedules for the generated benchmark instances. Despite the different time-indexing structures, no differences in solution quality are observed.

Overall, the results indicate that the choice between finish-time indexing and start-time indexing has no influence on solution quality for the benchmark instances considered in this study. Both formulations consistently produce identical objective values, project durations, and tardiness levels. Consequently, any differences observed in later comparisons can be attributed to computational performance rather than solution quality.

6.4.2. Computational Efficiency Comparison

Table 6.11 presents the computational efficiency results obtained by both formulations under Version 4.

Instance Set	Formulation	Avg. Runtime (s)	Avg. Binary Vars	Avg. Constraints
M1-CCV	PSPMM	9.58	10410.50	644.00
M1-CCV	Bianco-Caramia	15.31	10103.00	664.00
M1-CVX	PSPMM	6.87	9522.08	597.00
M1-CVX	Bianco-Caramia	12.18	9307.00	617.00
M1-HYB	PSPMM	6.11	9751.50	594.58
M1-HYB	Bianco-Caramia	7.07	8438.50	645.92
M2-CCV	PSPMM	6.26	8476.33	625.92
M2-CCV	Bianco-Caramia	6.11	8421.17	642.17
M2-CVX	PSPMM	6.58	9331.33	648.33
M2-CVX	Bianco-Caramia	8.16	9228.83	668.33
M2-HYB	PSPMM	6.97	9440.83	680.33
M2-HYB	Bianco-Caramia	9.06	9256.33	700.33

Table 6.11: Average runtime, number of binary variables, and number of constraints for the PSPMM and Bianco–Caramia formulations using Version 4.

Although the two formulations produce similar solution quality results, differences can be observed in computational performance. The PSPMM formulation generates fewer constraints than the Bianco–Caramia formulation in all six instance categories. Differences in the number of binary variables are smaller and vary between instance sets. In some categories the PSPMM formulation contains slightly more binary variables, while in others the Bianco–Caramia formulation generates the smaller model.

The runtime results show a modest advantage for the PSPMM formulation. It achieves shorter average solution times in five of the six instance categories, namely M1–CCV, M1–CVX, M1–HYB, M2–CVX, and M2–HYB. The only exception is the M2–CCV category, where the Bianco–Caramia formulation solves the instances slightly faster. An interesting observation is that runtime does not appear to be determined solely by model size. For example, the PSPMM formulation often solves faster even when it contains more binary variables than the Bianco–Caramia formulation. This suggests that factors such as formulation structure, linear relaxation quality, and solver search behaviour also influence computational performance [18, 23].

Overall, both formulations demonstrate good computational performance on the benchmark instances. However, the PSPMM formulation exhibits a consistent runtime advantage across most instance categories, indicating that the finish-time indexing approach provides a modest improvement in computational efficiency while maintaining comparable solution quality.

6.5. Discussion and Interpretation of Results

The computational experiments provide several insights into the behaviour of time-indexed formulations for the RCPSP with discrete time–cost trade-offs. Based on the results presented in this chapter, four main observations can be made.

First, the results show that the PSPMM formulation and the Bianco–Caramia start-time indexed formulation produce identical solution results for the benchmark instances considered in this study. Across all benchmark categories, both formulations obtained the same average objective values, makespans, and tardiness values. This confirms that the two formulations represent the same scheduling problem and identify equivalent optimal solutions. The results therefore suggest that the choice between start-time indexing and finish-time indexing has no observable effect on solution quality for the generated benchmark instances.

Second, the experiments demonstrate the importance of preprocessing techniques. The comparison between Version 1 and Version 3 shows that preprocessing substantially reduces the number of binary variables while preserving solution quality. Since objective values, makespans, and tardiness values remain unchanged, the preprocessing framework successfully removes unnecessary decision variables without affecting the feasible solution space. The results therefore show that implementation choices can significantly influence model size and computational performance.

Third, the comparison of the final implementation (Version 4) shows that differences between the formulations remain even when both are evaluated using the same preprocessing framework and lexicographic optimization strategy. The PSPMM formulation consistently generated fewer constraints across all benchmark categories and achieved shorter runtimes in five of the six instance sets. However, the formulation did not always contain fewer binary variables. In several instance categories, the PSPMM model contained more binary variables while still solving faster. This observation suggests that computational performance depends not only on model size but also on the mathematical structure of the formulation and the way the solver processes the resulting branch-and-bound search tree [18, 23].

Finally, the results indicate that lexicographic optimization provides no benefit for the benchmark instances considered in this study. Although the additional optimization phase consistently increased runtime because a second optimization problem had to be solved, it did not change the resulting objective values, makespans, or tardiness values. This suggests that the optimal solutions identified in the first optimization phase were already unique or that no alternative schedules with the same objective value and a shorter makespan existed. Consequently, the additional computational effort required by

lexicographic optimization did not lead to measurable improvements in solution quality for the tested instances.

Overall, the results show that both formulations are capable of solving the considered RCPSP instances effectively. Although the differences are generally modest, the PSPMM formulation achieved shorter average runtimes in most benchmark categories, indicating that finish-time indexing can provide computational benefits compared with the start-time indexed alternative.

6.6. Summary

This chapter presented the computational results obtained from the evaluation of the proposed PSPMM finish-time indexed formulation and the Bianco–Caramia start-time indexed formulation. The analysis considered three implementation versions: a baseline version without preprocessing (Version 1), a version including preprocessing techniques (Version 3), and a final version combining preprocessing with lexicographic optimization (Version 4).

The comparison between Versions 1 and 3 showed that the preprocessing framework substantially reduced the number of binary variables generated by both formulations. At the same time, objective values, makespans, and tardiness values remained unchanged across all benchmark categories, demonstrating that the preprocessing procedures preserved solution quality. The impact on runtime was less pronounced. While preprocessing improved computational performance for the Bianco–Caramia formulation, the benefits for the PSPMM formulation were more limited.

The comparison between Versions 3 and 4 showed that lexicographic optimization had no observable influence on solution quality. Objective values, makespans, and tardiness values remained unchanged across all benchmark categories for both formulations. Consequently, the additional optimization phase increased computational effort without producing measurable improvements in the final schedules.

The final comparison between the PSPMM formulation and the Bianco–Caramia formulation demonstrated that both approaches produce identical solution quality. Average objective values, makespans, and tardiness values were the same across all benchmark categories. In terms of computational efficiency, the PSPMM formulation generated fewer constraints and achieved shorter runtimes in five of the six benchmark categories. These results indicate that the finish-time indexing approach can provide a modest computational advantage while maintaining the same level of solution quality.

Overall, the computational study shows that both formulations are effective for solving the considered RCPSP instances. The proposed PSPMM formulation performs competitively with an established start-time indexed formulation and demonstrates that finish-time indexing is a viable alternative for modeling project scheduling problems with discrete time–cost trade-offs.

Detailed instance-level results and the corresponding Python implementations are provided in Appendices B, C, and D.

Conclusion and Future Research

This thesis presented a time-indexed integer linear programming approach for the Resource-Constrained Project Scheduling Problem (RCPSP), extended to incorporate mitigation measures in the form of multiple execution modes and project tardiness penalties. By replacing the traditional start-time indexing approach with a finish-time indexing structure, an alternative mathematical formulation was developed that simultaneously optimizes activity scheduling and mitigation decisions while respecting renewable resource constraints.

The theoretical analysis demonstrated the computational complexity of the proposed framework. The full formulation, including renewable resource constraints and tardiness penalties, was shown to be NP-hard through a polynomial-time reduction from the classical RCPSP. In addition, the simplified formulation without renewable resource constraints was shown to be NP-complete through a polynomial-time reduction from the discrete time–cost trade-off problem (DTCTP). These results indicate that the computational difficulty of the problem originates not only from resource limitations but also from the interaction between discrete mode selection, precedence relationships, and tardiness penalties.

To translate the theoretical model into a practical optimization tool, a computational framework was developed in Python using the Pyomo modeling environment and the Gurobi Optimizer. Because time-indexed mixed-integer programming formulations can become very large, several preprocessing techniques were incorporated into the implementation. These techniques included sparse variable generation, earliest-time feasibility filtering, and resource activity precomputation. The computational experiments showed that these procedures substantially reduced the number of binary variables while preserving solution quality.

An important finding of this research is the influence of preprocessing on model size and computational performance. The comparison between the baseline implementation and the preprocessed implementation demonstrated that a considerable reduction in binary variables can be achieved without affecting objective values, project makespans, or tardiness levels. Although the impact on runtime varied between formulations, the results show that preprocessing is an effective way to simplify time-indexed optimization models while maintaining the same solution quality.

A central contribution of this thesis was the computational comparison between the PSPMM formulation and a start-time indexed benchmark formulation inspired by Bianco and Caramia [4]. The computational results showed that both formulations produce identical solution quality for the benchmark instances considered in this study. Across all benchmark categories, the formulations obtained the same average objective values, makespans, and tardiness values. These findings confirm that both formulations represent the same underlying scheduling problem and identify equivalent optimal solutions despite using different time-indexing structures.

The comparison further showed that differences emerge primarily in terms of computational efficiency. Under the final implementation (Version 4), which combines preprocessing techniques with lexicographic optimization, the PSPMM formulation consistently generated fewer constraints and achieved

shorter runtimes in five of the six benchmark categories. At the same time, the results demonstrated that model size alone does not fully explain computational performance. In several instance categories, the PSPMM formulation solved instances more quickly despite containing more binary variables than the benchmark formulation. This finding suggests that formulation structure plays an important role in determining solver performance.

The experiments also showed that lexicographic optimization provides little benefit for the benchmark instances considered in this study. Although the additional optimization phase increased computational effort, it did not improve objective values, project makespans, or tardiness levels. This indicates that the solutions obtained after optimizing the primary objective were already unique or that no alternative schedules with the same objective value and a shorter makespan existed.

Overall, the findings indicate that both formulation structure and implementation strategy influence the computational performance of time-indexed RCPSP models. While both formulations solved the benchmark instances effectively and produced identical solution quality, the PSPMM formulation demonstrated a modest computational advantage across most benchmark categories.

In conclusion, this research demonstrates that project scheduling with mitigation measures and renewable resource constraints remains a highly complex optimization problem. Nevertheless, exact mixed-integer programming approaches remain capable of solving realistically structured benchmark instances effectively when supported by appropriate implementation techniques. The PSPMM formulation contributes to the RCPSP literature by providing an alternative time-indexing perspective and by offering insights into the relationship between formulation design, preprocessing, and computational performance.

7.1. Limitations and Future Research

Several limitations of this study should be acknowledged.

First, the computational experiments were limited to benchmark instances containing 20 activities. Although these instances preserve the main structural characteristics of the original project networks, larger instances may exhibit different computational behavior and could require additional acceleration techniques or decomposition approaches [8].

Second, renewable resource data were generated synthetically because the original DTCTP benchmark instances do not contain resource information. Although the generation procedure was designed to create realistic resource conflicts, alternative generation methods may lead to different computational characteristics.

Third, the proposed formulation was compared with only one formulation from the literature. Additional comparisons with event-based, flow-based, or resource-flow formulations could provide a broader understanding of the advantages and limitations of finish-time indexing [3, 18].

Fourth, the preprocessing techniques were evaluated collectively rather than individually. Consequently, the exact contribution of sparse variable generation, earliest-time preprocessing, and resource activity preprocessing to the observed reductions in model size and runtime cannot be determined from the current experiments.

Fifth, all experiments were conducted using a single solver configuration and a single hardware environment. Different solver settings, solver versions, or computational platforms may influence the observed performance.

A further methodological limitation concerns the preprocessing procedures used in the study. The same preprocessing principles were applied to both formulations to ensure a consistent basis for comparison. However, preprocessing techniques may interact differently with start-time and finish-time indexed formulations. As a result, some of the observed computational differences may be influenced by the chosen preprocessing design rather than solely by the indexing structure of the formulations.

These limitations suggest several directions for future research:

- **Scalability testing:** Evaluating the proposed formulation on larger benchmark instances and real-world project networks to assess its practical applicability and computational limits.

- **Additional formulation comparisons:** Comparing finish-time indexing with a wider range of RCPSP formulations, including event-based and continuous-time approaches.
- **Advanced preprocessing methods:** Developing stronger preprocessing procedures to further reduce the number of feasible activity–time combinations before model construction. In particular, tighter lower and upper bounds on activity execution times may help reduce model size and improve computational performance [2].
- **Preprocessing analysis and formulation-specific preprocessing:** Investigating the individual effects of sparse variable generation, earliest-time preprocessing, and resource activity preprocessing separately. Such an analysis would identify which techniques contribute most to reductions in model size and runtime. Future research could also develop preprocessing procedures tailored specifically to start-time and finish-time indexed formulations to provide a more balanced comparison of their computational performance.
- **Adaptive horizon reduction:** Future research could investigate horizon reduction as an additional preprocessing technique for time-indexed RCPSP formulations. The objective would be to determine how much the scheduling horizon can be reduced without affecting solution quality, while improving computational efficiency.
- **Model extensions:** Extending the formulation to include additional project scheduling features such as non-renewable resources, doubly constrained resources, uncertain activity durations, or uncertain resource capacities [15].
- **Hybrid solution methods:** Combining the proposed exact optimization model with heuristic approaches to improve computational performance for larger project instances. Such approaches may provide high-quality solutions within shorter computation times when exact optimization becomes difficult.

Use of Artificial Intelligence Tools

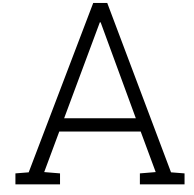
During the preparation of this thesis, generative artificial intelligence tools, including ChatGPT by OpenAI, were used for language refinement, LaTeX formatting assistance, programming support, and structural feedback on academic writing. All mathematical derivations, optimization models, computational implementations, and experimental results were independently verified and validated by the author.

The author takes full responsibility for the accuracy, originality, and scientific integrity of the contents of this thesis.

Reference

- [1] Can Akkan, Andreas Drexl, and Alf Kimms. “Network decomposition-based benchmark results for the discrete time–cost tradeoff problem”. In: *European Journal of Operational Research* 165.2 (Sept. 2005), pp. 339–358. DOI: 10.1016/j.ejor.2004.04.006.
- [2] Christian Artigues, Sophie Demasse, and Emmanuel Néron. *Resource-Constrained Project Scheduling: Models, algorithms, extensions and applications*. Wiley, 2013.
- [3] Christian Artigues, Philippe Michelon, and Stéphane Reusser. “Insertion techniques for static and dynamic resource-constrained project scheduling”. In: *European Journal of Operational Research* 149.2 (Sept. 2003), pp. 249–267. DOI: 10.1016/s0377-2217(02)00758-0.
- [4] Lucio Bianco and Massimiliano Caramia. “A new formulation for the project scheduling problem under limited resources”. In: *Flexible Services and Manufacturing Journal* 25.1–2 (Nov. 2011), pp. 6–24. DOI: 10.1007/s10696-011-9127-y.
- [5] J. Blazewicz, J.K. Lenstra, and A.H.G.Rinnooy Kan. “Scheduling subject to resource constraints: Classification and complexity”. In: *Discrete Applied Mathematics* 5.1 (Jan. 1983), pp. 11–24. DOI: 10.1016/0166-218x(83)90012-4.
- [6] Peter Brucker et al. “Resource-Constrained Project Scheduling: Notation, classification, models, and methods”. In: *European Journal of Operational Research* 112.1 (Jan. 1999), pp. 3–41. DOI: 10.1016/s0377-2217(98)00204-5.
- [7] Michael L. Bynum et al. *Pyomo – Optimization Modeling in Python*. Springer International Publishing Springer, 2021.
- [8] Eamonn T. Coughlan, Marco E. Lübbecke, and Jens Schulz. “A branch-price-and-cut algorithm for multi-mode resource leveling”. In: *European Journal of Operational Research* 245.1 (Aug. 2015), pp. 70–80. DOI: 10.1016/j.ejor.2015.02.043.
- [9] Prabuddha De et al. “The discrete time-cost tradeoff problem revisited”. In: *European Journal of Operational Research* 81.2 (Apr. 1995), pp. 225–238. DOI: 10.1016/0377-2217(94)00187-h.
- [10] Bert De Reyck and Willy Herroelen. “The multi-mode resource-constrained project scheduling problem with generalized precedence relations”. In: *European Journal of Operational Research* 119.2 (Dec. 1999), pp. 538–556. DOI: 10.1016/s0377-2217(99)00151-4.
- [11] Michael R. Garey and David S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, 2009.
- [12] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2026. URL: <https://www.gurobi.com>.
- [13] William E. Hart, Jean-Paul Watson, and David L. Woodruff. “Pyomo: Modeling and solving mathematical programs in Python”. In: *Mathematical Programming Computation* 3.3 (Aug. 2011), pp. 219–260. DOI: 10.1007/s12532-011-0026-8.
- [14] Öncü Hazır, Erdal Erel, and Yavuz Güralay. “Robust optimization models for the discrete time/-cost trade-off problem”. In: *International Journal of Production Economics* 130.1 (Mar. 2011), pp. 87–95. DOI: 10.1016/j.ijpe.2010.11.018.
- [15] Willy Herroelen and Roel Leus. “Project scheduling under uncertainty: Survey and research potentials”. In: *European Journal of Operational Research* 165.2 (Sept. 2005), pp. 289–306. DOI: 10.1016/j.ejor.2004.04.002.
- [16] James E. Kelley. “Critical-path planning and scheduling: Mathematical Basis”. In: *Operations Research* 9.3 (June 1961), pp. 296–320. DOI: 10.1287/opre.9.3.296.

- [17] Rainer Kolisch and Sönke Hartmann. “Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis”. In: *International Series in Operations Research & Management Science* (1999), pp. 147–178. DOI: 10.1007/978-1-4615-5533-9_7.
- [18] Oumar Koné et al. “Event-based MILP models for resource-constrained project scheduling problems”. In: *Computers amp; Operations Research* 38.1 (Jan. 2011), pp. 3–13. DOI: 10.1016/j.cor.2009.12.011.
- [19] Hanane Krim et al. “Tabu search for a parallel-machine scheduling problem with periodic maintenance, job rejection and weighted sum of Completion Times”. In: *Journal of Scheduling* 25.1 (Oct. 2021), pp. 89–105. DOI: 10.1007/s10951-021-00711-9.
- [20] Hadi Mokhtari, Reza Baradaran Kazemzadeh, and Ali Salmasnia. “Time-cost tradeoff analysis in project management: An ant system approach”. In: *IEEE Transactions on Engineering Management* 58.1 (Feb. 2011), pp. 36–43. DOI: 10.1109/tem.2010.2058859.
- [21] Michael L. Pinedo. *Scheduling: Theory, algorithms, and systems*. Springer, 2022.
- [22] A. Alan B. Pritsker, Lawrence J. Watters, and Philip M. Wolfe. “Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach”. In: *Management Science* 16.1 (1969), pp. 93–108. URL: <http://www.jstor.org/stable/2628369>.
- [23] Alexander Tesch. “Improved compact models for the resource-constrained project scheduling problem”. In: *Operations Research Proceedings* (July 2017), pp. 25–30. DOI: 10.1007/978-3-319-55702-1_4.



Benchmark Instance Characteristics

Table A.1 reports the structural properties of each generated instance. The columns have the following meaning:

- **Instance:** Identifier used throughout the computational experiments.
- **Activities:** Total number of activities in the project network, including dummy start and end activities.
- **Resources:** Number of renewable resource types.
- **Precedence:** Number of precedence relations between activities.
- **Modes:** Total number of execution modes across all activities.
- **Avg. Duration:** Average activity duration over all activity-mode combinations.
- **Cap R1:** Capacity of renewable resource R_1 .
- **Cap R2:** Capacity of renewable resource R_2 .

Instance	Activities	Resources	Precedence	Modes	Avg. Duration	Cap R1	Cap R2
I1	20	2	47	51	21.00	32	32
I2	20	2	41	52	20.06	88	88
I3	20	2	42	55	19.62	64	56
I4	20	2	39	52	19.56	40	40
I5	20	2	45	54	20.19	40	24
I6	20	2	41	53	19.43	240	120
I7	20	2	41	53	19.23	120	120
I8	20	2	43	53	19.51	72	72
I9	20	2	37	55	20.18	24	16
I10	20	2	52	56	20.11	64	32
I11	20	2	39	52	21.50	24	24
I12	20	2	37	56	20.30	24	16
I13	20	2	49	54	20.43	88	88
I14	20	2	47	55	21.64	16	16
I15	20	2	41	55	20.49	96	96
I16	20	2	55	51	18.86	32	32
I17	20	2	47	55	21.25	72	72
I18	20	2	36	53	22.02	48	48
I19	20	2	58	55	19.25	48	48
I20	20	2	57	54	20.98	24	24
I21	20	2	36	56	22.25	48	40
I22	20	2	47	55	20.73	112	112

I23	20	2	39	56	20.52	16	16
I24	20	2	37	52	20.79	24	24
I25	20	2	44	54	20.26	24	24
I26	20	2	39	51	18.86	24	24
I27	20	2	43	54	18.67	112	112
I28	20	2	50	51	19.92	48	48
I29	20	2	46	52	21.27	16	16
I30	20	2	41	52	19.98	56	56
I31	20	2	40	55	21.71	40	40
I32	20	2	51	55	18.93	64	64
I33	20	2	39	56	21.38	112	112
I34	20	2	43	54	21.02	32	32
I35	20	2	42	54	19.80	56	56
I36	20	2	40	55	19.76	56	56
I37	20	2	45	56	25.95	16	8
I38	20	2	40	56	25.18	16	8
I39	20	2	43	56	25.16	16	8
I40	20	2	43	56	25.98	16	8
I41	20	2	38	56	26.16	16	8
I42	20	2	36	56	25.59	16	8
I43	20	2	52	56	25.48	16	8
I44	20	2	51	56	25.34	16	8
I45	20	2	46	56	26.02	16	8
I46	20	2	44	56	25.38	16	8
I47	20	2	57	56	25.77	16	8
I48	20	2	41	56	26.14	16	8
I49	20	2	49	56	25.39	16	8
I50	20	2	42	56	25.27	16	8
I51	20	2	45	56	26.04	16	8
I52	20	2	48	56	25.64	16	8
I53	20	2	47	56	26.34	16	8
I54	20	2	36	56	25.77	16	8
I55	20	2	38	56	25.61	16	8
I56	20	2	40	56	25.82	16	8
I57	20	2	40	56	25.39	16	8
I58	20	2	40	56	25.62	16	8
I59	20	2	47	56	25.91	16	8
I60	20	2	41	56	25.96	16	8
I61	20	2	73	56	25.55	16	8
I62	20	2	42	56	26.16	16	8
I63	20	2	38	56	26.21	16	8
I64	20	2	51	56	26.09	16	8
I65	20	2	40	56	26.64	16	8
I66	20	2	41	56	25.79	16	8
I67	20	2	44	56	26.07	16	8
I68	20	2	36	56	25.34	16	8
I69	20	2	43	56	25.55	16	8
I70	20	2	39	56	26.32	16	8
I71	20	2	46	56	25.73	16	8
I72	20	2	47	56	25.38	24	16

Table A.1: Detailed characteristics of generated test instances

B

Detailed Computational Solution Quality Results

B.1. PSPMM Formulation

B.1.1. Version V1

This table presents the detailed solution quality results for Version V1 of the PSPMM formulation, which represents the baseline implementation without preprocessing techniques or lexicographic optimization.

Instance	Objective	Makespan	Tardiness	Status
I1	58.44	213.00	1.00	optimal
I2	82.77	244.00	22.00	optimal
I3	129.63	291.00	50.00	optimal
I4	78.28	202.00	15.00	optimal
I5	110.13	306.00	57.00	optimal
I6	141.52	286.00	53.00	optimal
I7	119.08	287.00	60.00	optimal
I8	100.18	318.00	54.00	optimal
I9	70.44	279.00	29.00	optimal
I10	123.79	269.00	15.00	optimal
I11	85.24	242.00	11.00	optimal
I12	145.85	270.00	50.00	optimal
I13	50.76	207.00	18.00	optimal
I14	35.57	267.00	26.00	optimal
I15	42.40	271.00	45.00	optimal
I16	34.06	138.00	4.00	optimal
I17	54.19	290.00	48.00	optimal
I18	52.85	268.00	51.00	optimal
I19	47.74	253.00	52.00	optimal
I20	38.41	267.00	33.00	optimal
I21	34.99	267.00	23.00	optimal
I22	61.60	304.00	64.00	optimal
I23	40.59	275.00	46.00	optimal
I24	45.35	279.00	41.00	optimal
I25	32.81	217.00	3.00	optimal
I26	73.60	290.00	43.00	optimal
I27	84.31	233.00	38.00	optimal
I28	59.73	227.00	27.00	optimal

I29	74.75	272.00	46.00	optimal
I30	101.74	244.00	45.00	optimal
I31	41.54	223.00	16.00	optimal
I32	93.35	246.00	24.00	optimal
I33	85.23	281.00	49.00	optimal
I34	50.58	250.00	24.00	optimal
I35	128.61	283.00	65.00	optimal
I36	71.42	256.00	31.00	optimal
I37	58.38	349.00	23.00	optimal
I38	79.42	290.00	26.00	optimal
I39	73.01	260.00	20.00	optimal
I40	57.68	296.00	18.00	optimal
I41	68.02	350.00	27.00	optimal
I42	56.96	293.00	19.00	optimal
I43	51.50	196.00	12.00	optimal
I44	61.44	261.00	18.00	optimal
I45	51.46	317.00	19.00	optimal
I46	103.68	263.00	27.00	optimal
I47	53.17	293.00	17.00	optimal
I48	47.11	232.00	12.00	optimal
I49	25.82	265.00	19.00	optimal
I50	25.30	291.00	20.00	optimal
I51	24.70	294.00	20.00	optimal
I52	23.95	230.00	14.00	optimal
I53	23.54	292.00	18.00	optimal
I54	24.04	262.00	14.00	optimal
I55	23.89	302.00	14.00	optimal
I56	24.68	346.00	21.00	optimal
I57	24.37	339.00	22.00	optimal
I58	24.07	287.00	18.00	optimal
I59	23.65	289.00	17.00	optimal
I60	23.11	312.00	16.00	optimal
I61	75.18	267.00	28.00	optimal
I62	44.69	289.00	15.00	optimal
I63	42.88	293.00	18.00	optimal
I64	52.67	293.00	20.00	optimal
I65	43.97	349.00	19.00	optimal
I66	60.04	319.00	22.00	optimal
I67	62.57	324.00	29.00	optimal
I68	63.36	313.00	25.00	optimal
I69	56.02	356.00	30.00	optimal
I70	41.94	324.00	18.00	optimal
I71	46.41	289.00	17.00	optimal
I72	49.78	259.00	16.00	optimal

Table B.1: Solution quality results of the PSPMM formulation V1**B.1.2. Version V3**

This table presents the detailed solution quality results for Version V3 of the PSPMM formulation, which includes preprocessing techniques but does not apply lexicographic optimization.

Instance	Objective	Makespan	Tardiness	Status
I1	58.44	213.00	1.00	optimal
I2	82.77	244.00	22.00	optimal
I3	129.63	291.00	50.00	optimal

I4	78.28	202.00	15.00	optimal
I5	110.13	306.00	57.00	optimal
I6	141.52	286.00	53.00	optimal
I7	119.08	287.00	60.00	optimal
I8	100.18	318.00	54.00	optimal
I9	70.44	279.00	29.00	optimal
I10	123.79	269.00	15.00	optimal
I11	85.24	242.00	11.00	optimal
I12	145.85	270.00	50.00	optimal
I13	50.76	207.00	18.00	optimal
I14	35.57	267.00	26.00	optimal
I15	42.40	271.00	45.00	optimal
I16	34.06	138.00	4.00	optimal
I17	54.19	290.00	48.00	optimal
I18	52.85	268.00	51.00	optimal
I19	47.74	253.00	52.00	optimal
I20	38.41	267.00	33.00	optimal
I21	34.99	267.00	23.00	optimal
I22	61.60	304.00	64.00	optimal
I23	40.59	275.00	46.00	optimal
I24	45.35	279.00	41.00	optimal
I25	32.81	217.00	3.00	optimal
I26	73.60	290.00	43.00	optimal
I27	84.31	233.00	38.00	optimal
I28	59.73	227.00	27.00	optimal
I29	74.75	272.00	46.00	optimal
I30	101.74	244.00	45.00	optimal
I31	41.54	223.00	16.00	optimal
I32	93.35	246.00	24.00	optimal
I33	85.23	281.00	49.00	optimal
I34	50.58	250.00	24.00	optimal
I35	128.61	283.00	65.00	optimal
I36	71.42	256.00	31.00	optimal
I37	58.38	349.00	23.00	optimal
I38	79.42	290.00	26.00	optimal
I39	73.01	260.00	20.00	optimal
I40	57.68	296.00	18.00	optimal
I41	68.02	350.00	27.00	optimal
I42	56.96	293.00	19.00	optimal
I43	51.50	196.00	12.00	optimal
I44	61.44	261.00	18.00	optimal
I45	51.46	317.00	19.00	optimal
I46	103.68	263.00	27.00	optimal
I47	53.17	293.00	17.00	optimal
I48	47.11	232.00	12.00	optimal
I49	25.82	265.00	19.00	optimal
I50	25.30	291.00	20.00	optimal
I51	24.70	294.00	20.00	optimal
I52	23.95	230.00	14.00	optimal
I53	23.54	292.00	18.00	optimal
I54	24.04	262.00	14.00	optimal
I55	23.89	302.00	14.00	optimal
I56	24.68	346.00	21.00	optimal
I57	24.37	339.00	22.00	optimal
I58	24.07	287.00	18.00	optimal
I59	23.65	289.00	17.00	optimal

I60	23.11	312.00	16.00	optimal
I61	75.18	267.00	28.00	optimal
I62	44.69	289.00	15.00	optimal
I63	42.88	293.00	18.00	optimal
I64	52.67	293.00	20.00	optimal
I65	43.97	349.00	19.00	optimal
I66	60.04	319.00	22.00	optimal
I67	62.57	324.00	29.00	optimal
I68	63.36	313.00	25.00	optimal
I69	56.02	356.00	30.00	optimal
I70	41.94	324.00	18.00	optimal
I71	46.41	289.00	17.00	optimal
I72	49.78	259.00	16.00	optimal

Table B.2: Solution quality results of the PSPMM formulation V3**B.1.3. Version V4**

This table presents the detailed solution quality results for Version V4 of the PSPMM formulation, which combines preprocessing techniques with lexicographic optimization and represents the fully enhanced implementation.

Instance	Objective	Makespan	Tardiness	Status
I1	58.44	213.00	1.00	optimal
I2	82.77	244.00	22.00	optimal
I3	137.08	288.00	47.00	optimal
I4	78.28	202.00	15.00	optimal
I5	112.08	287.00	38.00	optimal
I6	142.74	278.00	45.00	optimal
I7	125.86	270.00	43.00	optimal
I8	101.00	308.00	44.00	optimal
I9	70.44	279.00	29.00	optimal
I10	123.79	269.00	15.00	optimal
I11	85.24	242.00	11.00	optimal
I12	149.75	264.00	44.00	optimal
I13	50.76	207.00	18.00	optimal
I14	35.57	267.00	26.00	optimal
I15	42.40	271.00	45.00	optimal
I16	34.06	138.00	4.00	optimal
I17	54.19	290.00	48.00	optimal
I18	53.61	250.00	33.00	optimal
I19	48.27	239.00	38.00	optimal
I20	38.41	267.00	33.00	optimal
I21	34.99	267.00	23.00	optimal
I22	63.90	286.00	46.00	optimal
I23	40.69	273.00	44.00	optimal
I24	45.35	279.00	41.00	optimal
I25	32.81	217.00	3.00	optimal
I26	73.60	290.00	43.00	optimal
I27	84.31	233.00	38.00	optimal
I28	59.73	227.00	27.00	optimal
I29	75.37	270.00	44.00	optimal
I30	106.93	230.00	31.00	optimal
I31	41.54	223.00	16.00	optimal
I32	93.35	246.00	24.00	optimal
I33	91.57	276.00	44.00	optimal

I34	50.58	250.00	24.00	optimal
I35	130.95	260.00	42.00	optimal
I36	71.42	256.00	31.00	optimal
I37	58.38	349.00	23.00	optimal
I38	79.42	290.00	26.00	optimal
I39	73.01	260.00	20.00	optimal
I40	57.68	296.00	18.00	optimal
I41	68.02	350.00	27.00	optimal
I42	56.96	293.00	19.00	optimal
I43	51.50	196.00	12.00	optimal
I44	61.44	261.00	18.00	optimal
I45	51.46	317.00	19.00	optimal
I46	103.68	263.00	27.00	optimal
I47	53.17	293.00	17.00	optimal
I48	47.11	232.00	12.00	optimal
I49	25.82	265.00	19.00	optimal
I50	25.30	291.00	20.00	optimal
I51	24.70	294.00	20.00	optimal
I52	23.95	230.00	14.00	optimal
I53	23.54	292.00	18.00	optimal
I54	24.04	262.00	14.00	optimal
I55	23.89	302.00	14.00	optimal
I56	24.68	346.00	21.00	optimal
I57	24.37	339.00	22.00	optimal
I58	24.07	287.00	18.00	optimal
I59	23.65	289.00	17.00	optimal
I60	23.11	312.00	16.00	optimal
I61	75.18	267.00	28.00	optimal
I62	44.69	289.00	15.00	optimal
I63	42.88	293.00	18.00	optimal
I64	52.67	293.00	20.00	optimal
I65	43.97	349.00	19.00	optimal
I66	60.04	319.00	22.00	optimal
I67	62.57	324.00	29.00	optimal
I68	63.36	313.00	25.00	optimal
I69	56.02	356.00	30.00	optimal
I70	41.94	324.00	18.00	optimal
I71	46.41	289.00	17.00	optimal
I72	49.78	259.00	16.00	optimal

Table B.3: Solution quality results of the PSPMM formulation V4

B.2. Bianco–Caramia Formulation

B.2.1. Version V1

This table presents the detailed solution quality results for Version V1 of the Bianco–Caramia formulation, which represents the baseline implementation without preprocessing techniques or lexicographic optimization.

Instance	Objective	Makespan	Tardiness	Status
I1	58.44	213.00	1.00	optimal
I2	82.77	244.00	22.00	optimal
I3	129.63	291.00	50.00	optimal
I4	78.28	202.00	15.00	optimal
I5	110.13	306.00	57.00	optimal

I6	141.52	286.00	53.00	optimal
I7	119.08	287.00	60.00	optimal
I8	100.18	318.00	54.00	optimal
I9	70.44	279.00	29.00	optimal
I10	123.79	269.00	15.00	optimal
I11	85.24	242.00	11.00	optimal
I12	145.85	270.00	50.00	optimal
I13	50.76	207.00	18.00	optimal
I14	35.57	267.00	26.00	optimal
I15	42.40	271.00	45.00	optimal
I16	34.06	138.00	4.00	optimal
I17	54.19	290.00	48.00	optimal
I18	52.85	268.00	51.00	optimal
I19	47.74	253.00	52.00	optimal
I20	38.41	267.00	33.00	optimal
I21	34.99	267.00	23.00	optimal
I22	61.60	304.00	64.00	optimal
I23	40.59	275.00	46.00	optimal
I24	45.35	279.00	41.00	optimal
I25	32.81	217.00	3.00	optimal
I26	73.60	290.00	43.00	optimal
I27	84.31	233.00	38.00	optimal
I28	59.73	227.00	27.00	optimal
I29	74.75	272.00	46.00	optimal
I30	101.74	244.00	45.00	optimal
I31	41.54	223.00	16.00	optimal
I32	93.35	246.00	24.00	optimal
I33	85.23	281.00	49.00	optimal
I34	50.58	250.00	24.00	optimal
I35	128.61	283.00	65.00	optimal
I36	71.42	256.00	31.00	optimal
I37	58.38	349.00	23.00	optimal
I38	79.42	290.00	26.00	optimal
I39	73.01	260.00	20.00	optimal
I40	57.68	296.00	18.00	optimal
I41	68.02	350.00	27.00	optimal
I42	56.96	293.00	19.00	optimal
I43	51.50	196.00	12.00	optimal
I44	61.44	261.00	18.00	optimal
I45	51.46	317.00	19.00	optimal
I46	103.68	263.00	27.00	optimal
I47	53.17	293.00	17.00	optimal
I48	47.11	232.00	12.00	optimal
I49	25.82	265.00	19.00	optimal
I50	25.30	291.00	20.00	optimal
I51	24.70	294.00	20.00	optimal
I52	23.95	230.00	14.00	optimal
I53	23.54	292.00	18.00	optimal
I54	24.04	262.00	14.00	optimal
I55	23.89	302.00	14.00	optimal
I56	24.68	346.00	21.00	optimal
I57	24.37	339.00	22.00	optimal
I58	24.07	287.00	18.00	optimal
I59	23.65	289.00	17.00	optimal
I60	23.11	312.00	16.00	optimal
I61	75.18	267.00	28.00	optimal

I62	44.69	289.00	15.00	optimal
I63	42.88	293.00	18.00	optimal
I64	52.67	293.00	20.00	optimal
I65	43.97	349.00	19.00	optimal
I66	60.04	319.00	22.00	optimal
I67	62.57	324.00	29.00	optimal
I68	63.36	313.00	25.00	optimal
I69	56.02	356.00	30.00	optimal
I70	41.94	324.00	18.00	optimal
I71	46.41	289.00	17.00	optimal
I72	49.78	259.00	16.00	optimal

Table B.4: Solution quality results of the Bianco and Caramia formulation V1**B.2.2. Version V3**

This table presents the detailed solution quality results for Version V3 of the Bianco–Caramia formulation, which includes preprocessing techniques but does not apply lexicographic optimization.

Instance	Objective	Makespan	Tardiness	Status
I1	58.44	213.00	1.00	optimal
I2	82.77	244.00	22.00	optimal
I3	129.63	291.00	50.00	optimal
I4	78.28	202.00	15.00	optimal
I5	110.13	306.00	57.00	optimal
I6	141.52	286.00	53.00	optimal
I7	119.08	287.00	60.00	optimal
I8	100.18	318.00	54.00	optimal
I9	70.44	279.00	29.00	optimal
I10	123.79	269.00	15.00	optimal
I11	85.24	242.00	11.00	optimal
I12	145.85	270.00	50.00	optimal
I13	50.76	207.00	18.00	optimal
I14	35.57	267.00	26.00	optimal
I15	42.40	271.00	45.00	optimal
I16	34.06	138.00	4.00	optimal
I17	54.19	290.00	48.00	optimal
I18	52.85	268.00	51.00	optimal
I19	47.74	253.00	52.00	optimal
I20	38.41	267.00	33.00	optimal
I21	34.99	267.00	23.00	optimal
I22	61.60	304.00	64.00	optimal
I23	40.59	275.00	46.00	optimal
I24	45.35	279.00	41.00	optimal
I25	32.81	217.00	3.00	optimal
I26	73.60	290.00	43.00	optimal
I27	84.31	233.00	38.00	optimal
I28	59.73	227.00	27.00	optimal
I29	74.75	272.00	46.00	optimal
I30	101.74	244.00	45.00	optimal
I31	41.54	223.00	16.00	optimal
I32	93.35	246.00	24.00	optimal
I33	85.23	281.00	49.00	optimal
I34	50.58	250.00	24.00	optimal
I35	128.61	283.00	65.00	optimal
I36	71.42	256.00	31.00	optimal

I37	58.38	349.00	23.00	optimal
I38	79.42	290.00	26.00	optimal
I39	73.01	260.00	20.00	optimal
I40	57.68	296.00	18.00	optimal
I41	68.02	350.00	27.00	optimal
I42	56.96	293.00	19.00	optimal
I43	51.50	196.00	12.00	optimal
I44	61.44	261.00	18.00	optimal
I45	51.46	317.00	19.00	optimal
I46	103.68	263.00	27.00	optimal
I47	53.17	293.00	17.00	optimal
I48	47.11	232.00	12.00	optimal
I49	25.82	265.00	19.00	optimal
I50	25.30	291.00	20.00	optimal
I51	24.70	294.00	20.00	optimal
I52	23.95	230.00	14.00	optimal
I53	23.54	292.00	18.00	optimal
I54	24.04	262.00	14.00	optimal
I55	23.89	302.00	14.00	optimal
I56	24.68	346.00	21.00	optimal
I57	24.37	339.00	22.00	optimal
I58	24.07	287.00	18.00	optimal
I59	23.65	289.00	17.00	optimal
I60	23.11	312.00	16.00	optimal
I61	75.18	267.00	28.00	optimal
I62	44.69	289.00	15.00	optimal
I63	42.88	293.00	18.00	optimal
I64	52.67	293.00	20.00	optimal
I65	43.97	349.00	19.00	optimal
I66	60.04	319.00	22.00	optimal
I67	62.57	324.00	29.00	optimal
I68	63.36	313.00	25.00	optimal
I69	56.02	356.00	30.00	optimal
I70	41.94	324.00	18.00	optimal
I71	46.41	289.00	17.00	optimal
I72	49.78	259.00	16.00	optimal

Table B.5: Solution quality results of the Bianco and Caramia formulation V3**B.2.3. Version V4**

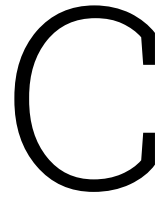
This table presents the detailed solution quality results for Version V4 of the Bianco–Caramia formulation, which combines preprocessing techniques with lexicographic optimization and represents the fully enhanced implementation.

Instance	Objective	Makespan	Tardiness	Status
I1	58.44	213.00	1.00	optimal
I2	82.77	244.00	22.00	optimal
I3	129.63	291.00	50.00	optimal
I4	78.28	202.00	15.00	optimal
I5	110.13	306.00	57.00	optimal
I6	141.52	286.00	53.00	optimal
I7	119.08	287.00	60.00	optimal
I8	100.18	318.00	54.00	optimal
I9	70.44	279.00	29.00	optimal
I10	123.79	269.00	15.00	optimal

I11	85.24	242.00	11.00	optimal
I12	145.85	270.00	50.00	optimal
I13	50.76	207.00	18.00	optimal
I14	35.57	267.00	26.00	optimal
I15	42.40	271.00	45.00	optimal
I16	34.06	138.00	4.00	optimal
I17	54.19	290.00	48.00	optimal
I18	52.85	268.00	51.00	optimal
I19	47.74	253.00	52.00	optimal
I20	38.41	267.00	33.00	optimal
I21	34.99	267.00	23.00	optimal
I22	61.60	304.00	64.00	optimal
I23	40.59	275.00	46.00	optimal
I24	45.35	279.00	41.00	optimal
I25	32.81	217.00	3.00	optimal
I26	73.60	290.00	43.00	optimal
I27	84.31	233.00	38.00	optimal
I28	59.73	227.00	27.00	optimal
I29	74.75	272.00	46.00	optimal
I30	101.74	244.00	45.00	optimal
I31	41.54	223.00	16.00	optimal
I32	93.35	246.00	24.00	optimal
I33	85.23	281.00	49.00	optimal
I34	50.58	250.00	24.00	optimal
I35	128.61	283.00	65.00	optimal
I36	71.42	256.00	31.00	optimal
I37	58.38	349.00	23.00	optimal
I38	79.42	290.00	26.00	optimal
I39	73.01	260.00	20.00	optimal
I40	57.68	296.00	18.00	optimal
I41	68.02	350.00	27.00	optimal
I42	56.96	293.00	19.00	optimal
I43	51.50	196.00	12.00	optimal
I44	61.44	261.00	18.00	optimal
I45	51.46	317.00	19.00	optimal
I46	103.68	263.00	27.00	optimal
I47	53.17	293.00	17.00	optimal
I48	47.11	232.00	12.00	optimal
I49	25.82	265.00	19.00	optimal
I50	25.30	291.00	20.00	optimal
I51	24.70	294.00	20.00	optimal
I52	23.95	230.00	14.00	optimal
I53	23.54	292.00	18.00	optimal
I54	24.04	262.00	14.00	optimal
I55	23.89	302.00	14.00	optimal
I56	24.68	346.00	21.00	optimal
I57	24.37	339.00	22.00	optimal
I58	24.07	287.00	18.00	optimal
I59	23.65	289.00	17.00	optimal
I60	23.11	312.00	16.00	optimal
I61	75.18	267.00	28.00	optimal
I62	44.69	289.00	15.00	optimal
I63	42.88	293.00	18.00	optimal
I64	52.67	293.00	20.00	optimal
I65	43.97	349.00	19.00	optimal
I66	60.04	319.00	22.00	optimal

I67	62.57	324.00	29.00	optimal
I68	63.36	313.00	25.00	optimal
I69	56.02	356.00	30.00	optimal
I70	41.94	324.00	18.00	optimal
I71	46.41	289.00	17.00	optimal
I72	49.78	259.00	16.00	optimal

Table B.6: Solution quality results of the Bianco and Caramia formulation V4



Detailed Computational Efficiency Results

C.1. PSPMM Formulation

C.1.1. Version V1

This table presents the detailed computational efficiency results for Version V1 of the PSPMM, which represents the baseline implementation without preprocessing techniques or lexicographic optimization.

Instance	Runtime	Binary variables	Constraints
I1	2.01	12546.00	600
I2	1.68	12737.00	590
I3	2.48	15916.00	679
I4	1.38	10787.00	512
I5	2.64	15542.00	680
I6	2.09	14711.00	654
I7	2.29	14457.00	644
I8	2.44	16456.00	722
I9	3.22	15170.00	648
I10	3.36	17354.00	731
I11	2.33	13442.00	618
I12	3.04	14039.00	598
I13	1.70	11803.00	546
I14	2.40	14210.00	626
I15	1.87	13833.00	604
I16	0.91	7300.00	398
I17	2.53	15386.00	668
I18	2.04	13355.00	603
I19	1.72	13021.00	589
I20	2.38	13987.00	636
I21	2.00	14658.00	623
I22	2.52	15800.00	682
I23	2.53	14307.00	610
I24	2.54	13999.00	636
I25	1.92	12028.00	549
I26	2.09	14491.00	664
I27	1.58	11628.00	530
I28	1.38	11020.00	541

I29	2.21	13090.00	611
I30	1.76	11805.00	554
I31	1.42	11511.00	521
I32	2.01	14524.00	636
I33	1.88	14595.00	622
I34	1.77	13229.00	594
I35	2.14	14537.00	639
I36	1.82	14368.00	621
I37	3.76	18147.00	764
I38	3.80	14886.00	641
I39	3.04	13207.00	584
I40	3.67	15177.00	656
I41	3.44	18191.00	759
I42	3.51	15031.00	643
I43	1.87	9605.00	465
I44	3.18	13253.00	594
I45	4.12	16351.00	701
I46	3.83	13363.00	591
I47	3.88	15021.00	664
I48	3.12	11584.00	526
I49	3.23	13474.00	600
I50	3.45	15049.00	649
I51	3.80	15230.00	660
I52	2.67	11668.00	535
I53	3.80	15157.00	660
I54	3.00	13397.00	585
I55	3.38	16094.00	683
I56	3.54	18154.00	759
I57	3.81	17898.00	749
I58	3.33	14805.00	639
I59	3.79	14901.00	650
I60	3.89	16354.00	696
I61	7.34	13577.00	628
I62	3.52	14775.00	641
I63	3.67	14996.00	645
I64	3.80	15003.00	658
I65	4.70	18164.00	761
I66	3.93	16476.00	700
I67	4.56	16740.00	713
I68	3.12	16165.00	683
I69	4.59	18617.00	778
I70	4.55	16726.00	708
I71	3.64	14799.00	645
I72	2.81	13307.00	592

Table C.1: Computational efficiency results of the PSPMM formulation V1**C.1.2. Version V3**

This table presents the detailed computational efficiency results for Version V3 of the PSPMM formulation, which includes preprocessing techniques but does not apply lexicographic optimization.

Instance	Runtime	Binary variables	Constraints
I1	4.73	9421.00	600
I2	3.90	8657.00	590
I3	6.31	12059.00	679

I4	3.08	7442.00	512
I5	5.39	10685.00	680
I6	4.06	10732.00	654
I7	5.04	11234.00	644
I8	5.61	11406.00	722
I9	5.17	10112.00	648
I10	7.43	13193.00	731
I11	3.40	9641.00	618
I12	3.76	9371.00	598
I13	3.15	9284.00	546
I14	3.18	9280.00	626
I15	3.17	9782.00	604
I16	1.66	5403.00	398
I17	3.89	10518.00	668
I18	3.51	10188.00	603
I19	2.79	9711.00	589
I20	2.75	8884.00	636
I21	6.30	9997.00	623
I22	3.86	11814.00	682
I23	3.29	9901.00	610
I24	2.98	9500.00	636
I25	2.43	7939.00	549
I26	3.44	10240.00	664
I27	2.22	8047.00	530
I28	2.41	7686.00	541
I29	3.10	8867.00	611
I30	2.72	7888.00	554
I31	2.16	7616.00	521
I32	3.72	11368.00	636
I33	3.09	11010.00	622
I34	2.89	8735.00	594
I35	3.95	11796.00	639
I36	2.88	10426.00	621
I37	4.24	10616.00	764
I38	3.84	9663.00	641
I39	2.48	7787.00	584
I40	2.89	8156.00	656
I41	4.79	10615.00	759
I42	3.28	8679.00	643
I43	1.91	5590.00	465
I44	2.91	7996.00	594
I45	4.06	10163.00	701
I46	3.64	9302.00	591
I47	3.29	8112.00	664
I48	2.29	5846.00	526
I49	2.61	7920.00	600
I50	3.22	9313.00	649
I51	3.44	9568.00	660
I52	2.86	6773.00	535
I53	3.58	9510.00	660
I54	2.82	8199.00	585
I55	3.65	10961.00	683
I56	3.33	9624.00	759
I57	3.90	11360.00	749
I58	3.10	9477.00	639
I59	3.18	9286.00	650

I60	3.18	9985.00	696
I61	3.03	8670.00	628
I62	2.38	7744.00	641
I63	3.19	9835.00	645
I64	3.12	9143.00	658
I65	3.38	9353.00	761
I66	3.54	10540.00	700
I67	3.68	9995.00	713
I68	3.12	9463.00	683
I69	4.03	11820.00	778
I70	3.31	10080.00	708
I71	2.67	7733.00	645
I72	2.55	7914.00	592

Table C.2: Computational efficiency results of the PSPMM formulation V3**C.1.3. Version V4**

This table presents the detailed computational efficiency results for Version V4 of the PSPMM formulation, which combines preprocessing techniques with lexicographic optimization and represents the fully enhanced implementation.

Instance	Runtime	Binary variables	Constraints
I1	10.93	9421.00	601
I2	7.96	8657.00	591
I3	13.05	12059.00	680
I4	6.05	7442.00	513
I5	10.82	10685.00	681
I6	9.23	10732.00	655
I7	10.66	11234.00	645
I8	11.22	11406.00	723
I9	9.11	10112.00	649
I10	13.80	13193.00	732
I11	5.86	9641.00	619
I12	8.25	9371.00	599
I13	5.75	9284.00	547
I14	6.74	9280.00	627
I15	6.19	9782.00	605
I16	3.17	5403.00	399
I17	7.89	10518.00	669
I18	7.01	10188.00	604
I19	6.02	9711.00	590
I20	5.49	8884.00	637
I21	5.77	9997.00	624
I22	9.58	11814.00	683
I23	7.61	9901.00	611
I24	7.20	9500.00	637
I25	4.88	7939.00	550
I26	6.68	10240.00	665
I27	5.36	8047.00	531
I28	4.52	7686.00	542
I29	6.54	8867.00	612
I30	4.49	7888.00	555
I31	4.44	7616.00	522
I32	7.20	11368.00	637
I33	7.22	11010.00	623

I34	5.27	8735.00	595
I35	9.91	11796.00	640
I36	6.48	10426.00	622
I37	8.21	10616.00	765
I38	6.54	9663.00	642
I39	6.02	7787.00	585
I40	5.94	8156.00	657
I41	8.20	10615.00	760
I42	5.95	8679.00	644
I43	3.90	5590.00	466
I44	6.03	7996.00	595
I45	8.20	10163.00	702
I46	6.42	9302.00	592
I47	5.87	8112.00	665
I48	3.81	5846.00	527
I49	5.22	7920.00	601
I50	6.69	9313.00	650
I51	7.25	9568.00	661
I52	4.92	6773.00	536
I53	7.31	9510.00	661
I54	6.25	8199.00	586
I55	6.85	10961.00	684
I56	6.74	9624.00	760
I57	7.75	11360.00	750
I58	6.84	9477.00	640
I59	6.35	9286.00	651
I60	6.76	9985.00	697
I61	6.79	8670.00	629
I62	5.70	7744.00	642
I63	6.64	9835.00	646
I64	6.53	9143.00	659
I65	6.80	9353.00	762
I66	8.19	10540.00	701
I67	8.29	9995.00	714
I68	6.84	9463.00	684
I69	9.41	11820.00	779
I70	7.22	10080.00	709
I71	5.41	7733.00	646
I72	5.83	7914.00	593

Table C.3: Computational efficiency results of the PSPMM formulation V4

C.2. Bianco–Caramia Formulation

C.2.1. Version V1

This table presents the detailed computational efficiency results for Version V1 of the Bianco–Caramia formulation, which represents the baseline implementation without preprocessing techniques or lexicographic optimization.

Instance	Runtime	Binary variables	Constraints
I1	9.03	12546.00	620
I2	6.90	12737.00	610
I3	8.82	15916.00	699
I4	5.88	10787.00	532
I5	8.46	15542.00	700

I6	7.26	14711.00	674
I7	7.50	14457.00	664
I8	8.53	16456.00	742
I9	11.53	15170.00	668
I10	14.25	17354.00	751
I11	8.18	13442.00	638
I12	11.09	14039.00	618
I13	8.69	11803.00	566
I14	7.42	14210.00	646
I15	7.60	13833.00	624
I16	2.51	7300.00	418
I17	8.65	15386.00	688
I18	6.53	13355.00	623
I19	6.29	13021.00	609
I20	9.07	13987.00	656
I21	6.67	14658.00	643
I22	10.11	15800.00	702
I23	7.98	14307.00	630
I24	7.34	13999.00	656
I25	5.02	12028.00	569
I26	6.45	14491.00	684
I27	6.38	11628.00	550
I28	4.28	11020.00	561
I29	8.03	13090.00	631
I30	7.38	11805.00	574
I31	4.06	11511.00	541
I32	8.20	14524.00	656
I33	6.85	14595.00	642
I34	5.93	13229.00	614
I35	7.53	14537.00	659
I36	7.39	14368.00	641
I37	7.16	18147.00	784
I38	9.21	14886.00	661
I39	9.98	13207.00	604
I40	10.04	15177.00	676
I41	8.33	18191.00	779
I42	11.01	15031.00	663
I43	9.66	9605.00	485
I44	9.76	13253.00	614
I45	11.68	16351.00	721
I46	17.24	13363.00	611
I47	7.68	15021.00	684
I48	12.27	11584.00	546
I49	9.66	13474.00	620
I50	10.73	15049.00	669
I51	14.17	15230.00	680
I52	8.86	11668.00	555
I53	13.11	15157.00	680
I54	10.74	13397.00	605
I55	11.31	16094.00	703
I56	9.02	18154.00	779
I57	9.10	17898.00	769
I58	8.52	14805.00	659
I59	8.10	14901.00	670
I60	12.07	16354.00	716
I61	10.77	13577.00	648

I62	12.20	14775.00	661
I63	11.50	14996.00	665
I64	11.26	15003.00	678
I65	13.09	18164.00	781
I66	11.08	16476.00	720
I67	13.07	16740.00	733
I68	7.94	16165.00	703
I69	12.51	18617.00	798
I70	11.65	16726.00	728
I71	10.91	14799.00	665
I72	8.25	13307.00	612

Table C.4: Computational efficiency results of the Bianco–Caramia formulation V1**C.2.2. Version V3**

This table presents the detailed computational efficiency results for Version V3 of the Bianco–Caramia formulation, which includes preprocessing techniques but does not apply lexicographic optimization.

Instance	Runtime	Binary variables	Constraints
I1	10.62	9207.00	620
I2	7.59	8419.00	610
I3	10.64	11810.00	699
I4	6.12	7271.00	532
I5	13.75	10420.00	700
I6	8.61	10495.00	674
I7	10.03	11025.00	664
I8	8.30	11172.00	742
I9	9.22	9943.00	668
I10	15.18	12823.00	751
I11	6.44	9431.00	638
I12	8.40	9111.00	618
I13	5.96	9079.00	566
I14	6.27	9109.00	646
I15	6.15	9603.00	624
I16	3.22	5229.00	418
I17	6.84	10239.00	688
I18	5.97	9968.00	623
I19	6.73	9508.00	609
I20	7.93	8691.00	656
I21	5.39	9810.00	643
I22	6.80	11508.00	702
I23	7.32	9715.00	630
I24	6.04	9235.00	656
I25	3.73	7804.00	569
I26	6.08	10023.00	684
I27	5.72	7815.00	550
I28	3.66	7498.00	561
I29	6.46	8690.00	631
I30	6.33	7688.00	574
I31	3.09	7441.00	541
I32	5.71	11117.00	656
I33	5.63	10791.00	642
I34	4.45	8527.00	614
I35	6.62	11542.00	659
I36	5.22	10221.00	641

I37	4.73	10516.00	784
I38	4.55	9563.00	661
I39	3.57	7680.00	604
I40	3.67	8068.00	676
I41	5.50	10507.00	779
I42	3.66	8584.00	663
I43	2.28	5498.00	485
I44	3.19	7905.00	614
I45	4.49	10073.00	721
I46	6.00	9194.00	611
I47	3.25	8010.00	684
I48	2.06	5764.00	546
I49	3.12	7805.00	620
I50	3.87	9190.00	669
I51	4.25	9450.00	680
I52	2.65	6667.00	555
I53	4.36	9415.00	680
I54	3.15	8099.00	605
I55	5.18	10872.00	703
I56	4.51	9519.00	779
I57	4.77	11251.00	769
I58	3.81	9378.00	659
I59	3.53	9197.00	670
I60	4.21	9903.00	716
I61	5.68	8566.00	648
I62	3.03	7659.00	661
I63	3.96	9751.00	665
I64	4.30	9051.00	678
I65	4.00	9263.00	781
I66	4.39	10435.00	720
I67	4.91	9878.00	733
I68	4.23	9342.00	703
I69	8.25	11714.00	798
I70	4.09	9986.00	728
I71	3.08	7630.00	665
I72	3.04	7801.00	612

Table C.5: Computational efficiency results of the Bianco–Caramia formulation V3

C.2.3. Version V4

This table presents the detailed computational efficiency results for Version V4 of the Bianco–Caramia formulation, which combines preprocessing techniques with lexicographic optimization and represents the fully enhanced implementation.

Instance	Runtime	Binary variables	Constraints
I1	16.39	9207.00	621
I2	10.96	8419.00	611
I3	21.51	11810.00	700
I4	8.74	7271.00	533
I5	20.97	10420.00	701
I6	14.07	10495.00	675
I7	13.97	11025.00	665
I8	14.58	11172.00	743
I9	10.79	9943.00	669
I10	22.92	12823.00	752

I11	8.23	9431.00	639
I12	18.08	9111.00	619
I13	10.40	9079.00	567
I14	9.95	9109.00	647
I15	11.70	9603.00	625
I16	4.07	5229.00	419
I17	12.86	10239.00	689
I18	13.66	9968.00	624
I19	12.40	9508.00	610
I20	10.61	8691.00	657
I21	9.13	9810.00	644
I22	16.02	11508.00	703
I23	15.02	9715.00	631
I24	10.36	9235.00	657
I25	6.51	7804.00	570
I26	11.52	10023.00	685
I27	11.09	7815.00	551
I28	7.03	7498.00	562
I29	11.31	8690.00	632
I30	10.04	7688.00	575
I31	5.51	7441.00	542
I32	10.59	11117.00	657
I33	12.62	10791.00	643
I34	7.57	8527.00	615
I35	18.33	11542.00	660
I36	13.48	10221.00	642
I37	10.27	10516.00	785
I38	8.84	9563.00	662
I39	7.27	7680.00	605
I40	6.95	8068.00	677
I41	10.78	10507.00	780
I42	7.24	8584.00	664
I43	4.18	5498.00	486
I44	6.13	7905.00	615
I45	8.13	10073.00	722
I46	10.36	9194.00	612
I47	6.28	8010.00	685
I48	4.39	5764.00	547
I49	6.50	7805.00	621
I50	8.55	9190.00	670
I51	8.69	9450.00	681
I52	5.02	6667.00	556
I53	8.37	9415.00	681
I54	6.85	8099.00	606
I55	10.50	10872.00	704
I56	8.47	9519.00	780
I57	9.81	11251.00	770
I58	7.82	9378.00	660
I59	7.19	9197.00	671
I60	8.14	9903.00	717
I61	10.27	8566.00	649
I62	6.34	7659.00	662
I63	7.88	9751.00	666
I64	8.11	9051.00	679
I65	8.03	9263.00	782
I66	9.15	10435.00	721

I67	9.50	9878.00	734
I68	8.54	9342.00	704
I69	14.29	11714.00	799
I70	8.40	9986.00	729
I71	6.72	7630.00	666
I72	6.07	7801.00	613

Table C.6: Computational efficiency results of the Bianco–Caramia formulation V4

D

Pyomo Model Implementations

This appendix presents the complete Pyomo implementation of the PSPMM formulation and the Bianco and Caramia formulation

D.1. PSPMM Formulation

```
1 import pyomo.environ as pyo
2 from dataset_augmented import load_augmented
3 from pathlib import Path
4 from pyomo.opt import SolverStatus, TerminationCondition
5 import time
6
7 # Load all instances
8 base = Path(__file__).parent
9 instance_folder = base / "generated_instances_20"
10 instance_files = sorted(instance_folder.glob("*.json"))
11
12 results_table = []
13
14 for file_path in instance_files:
15     print("\n")
16     print("=" * 70)
17     print("INSTANCE:", file_path.name)
18     print("=" * 70)
19     data = load_augmented(str(file_path))
20
21     # Extract data
22     activities = data["activities"]
23     resources = data["resources"]
24     modes = data["modes"]
25     durations = data["durations"]
26     costs = data["costs"]
27     resource_usage = data["resource_usage"]
28     resource_capacities = data["resource_capacities"]
29     predecessors = data["predecessors"]
30
31     # Critical path
32     fast_mode = {}
33     slow_mode = {}
34     for j in activities:
35         fast_mode[j] = min(modes[j], key=lambda m: durations[j,m])
36         slow_mode[j] = min(modes[j], key=lambda m: costs[j,m])
37
38     def critical_path_duration(selected_modes):
39         earliest_finish = {}
40         for j in activities:
41             duration = durations[j, selected_modes[j]]
42             if len(predecessors[j]) == 0:
43                 earliest_start = 0
```

```

44         else:
45             earliest_start = max(earliest_finish[h]
46                                 for h in predecessors[j])
47             earliest_finish[j] = (earliest_start+duration)
48             final_activity = max(activities)
49             return earliest_finish[final_activity]
50
51     CP_min = critical_path_duration(fast_mode)
52     CP_max = critical_path_duration(slow_mode)
53     print("Shortest_CP:", CP_min)
54     print("Longest_CP:", CP_max)
55
56     # Deadline & alpha
57     C_min = sum(costs[j, fast_mode[j]])
58     for j in activities:
59         C_max = sum(costs[j, slow_mode[j]])
60         for j in activities:
61             print("Fast_CPcost:", C_min)
62             print("Slow_CPcost:", C_max)
63
64     beta = 0.5 # Midpoint between shortest and longest critical path
65     # beta < 0.5 for aggressive deadline
66     # beta > 0.5 for relaxed deadline
67     deadline = int(CP_min+beta*(CP_max-CP_min))
68     print("Deadline:", deadline)
69
70     average_cost = (C_min+C_max)/2
71     average_cp = (CP_min+CP_max)/2
72     alpha = average_cost/average_cp
73     print("Alpha:", alpha)
74
75     # Time horizon
76     UB = CP_max
77     time_periods = range(UB + 1)
78
79     # Create Pyomo model
80     model = pyo.ConcreteModel()
81
82     # Remove dummy activities from time-indexed variables
83     real_activities = []
84     for j in activities:
85         all_zero = True
86         for m in modes[j]:
87             if durations[j,m] > 0:
88                 all_zero = False
89                 break
90         if not all_zero:
91             real_activities.append(j)
92
93     # Compute earliest finish times
94     earliest_finish = {}
95     for j in activities:
96         if len(predecessors[j]) == 0: earliest_start = 0
97         else:
98             earliest_start = max(earliest_finish[h]
99                                 for h in predecessors[j])
100             shortest_mode = min(durations[j,m]
101                                for m in modes[j])
102             earliest_finish[j] = (earliest_start + shortest_mode)
103
104     # Create feasible variables only
105     feasible_indices = []
106     for j in activities:
107         for m in modes[j]:
108             duration = durations[j,m]
109             # Dummy activity
110             if duration == 0:
111                 feasible_indices.append((j,0,m))
112             else:
113                 for t in time_periods:
114                     if t >= duration and t >= earliest_finish[j]:

```

```

115         feasible_indices.append((j,t,m))
116
117     # Precomputed index sets
118     activity_vars = {j: [] for j in activities}
119     activity_mode_vars = {}
120     for j in activities:
121         for m in modes[j]:
122             activity_mode_vars[j,m] = []
123     for (j,t,m) in feasible_indices:
124         activity_vars[j].append((t,m))
125         activity_mode_vars[j,m].append(t)
126
127     # Binary variables
128     model.x = pyo.Var(feasible_indices, within=pyo.Binary)
129
130     # Fast acces structures
131     activity_vars = {j: [] for j in activities}
132     activity_mode_vars = {}
133     for (j,t,m) in feasible_indices:
134         activity_vars[j].append((t,m))
135         if (j,m) not in activity_mode_vars:
136             activity_mode_vars[j,m] = []
137         activity_mode_vars[j,m].append(t)
138
139     # Tardiness variable
140     model.L = pyo.Var(within=pyo.NonNegativeReals)
141
142     # Finish times
143     def finish_time_rule(model,j):
144         return sum(t * model.x[j,t,m]
145                    for (t,m) in activity_vars[j])
146     model.F = pyo.Expression(activities, rule=finish_time_rule)
147
148     # Objective
149     def objective_rule(model):
150         mitigation_cost = sum(costs[j,m]*model.x[j,t,m]
151                               for (j,t,m) in feasible_indices)
152         return mitigation_cost + alpha * model.L
153     model.objective = pyo.Objective(rule=objective_rule, sense=pyo.minimize)
154
155     # Execute once
156     model.execute_once = pyo.ConstraintList()
157     for j in activities:
158         model.execute_once.add(sum(model.x[j,t,m]
159                                   for (t,m) in activity_vars[j]) == 1)
160
161     # Precedence constraints
162     model.precedence_constraints = pyo.ConstraintList()
163     for j in activities:
164         # finish time of j
165         finish_j = sum(t * model.x[j,t,m]
166                        for (t,m) in activity_vars[j])
167         # duration of j
168         duration_j = sum(durations[j,m] * sum(model.x[j,t,m]
169                                                for t in activity_mode_vars[j,m]) for m in modes[j])
170         # start time of j
171         start_j = finish_j - duration_j
172         for h in predecessors[j]:
173             # finish time of predecessor
174             finish_h = sum(t * model.x[h,t,m]
175                            for (t,m) in activity_vars[h])
176             model.precedence_constraints.add(start_j >= finish_h)
177
178     # Precompute active variables
179     active_by_time = {t: [] for t in time_periods}
180     for (j, tau, m) in feasible_indices:
181         duration = durations[j,m]
182         start = tau - duration
183         for t in range(start, tau):
184             active_by_time[t].append((j,tau,m))
185

```

```

186 # Precompute resource terms
187 resource_terms = {}
188 for k in resources:
189     for t in time_periods:
190         terms = []
191         for (j,tau,m) in active_by_time[t]:
192             usage = resource_usage.get((j,m,k), 0)
193             if usage > 0:
194                 terms.append(usage * model.x[j,tau,m])
195         resource_terms[k,t] = terms
196
197 # Resource constraints
198 model.resource_constraints = pyo.ConstraintList()
199 for k in resources:
200     capacity = resource_capacities[str(k)]
201     for t in time_periods:
202         terms = resource_terms[k,t]
203         if terms:
204             model.resource_constraints.add(sum(terms) <= capacity)
205
206 # Tardiness constraint
207 final_activity = max(activities)
208 model.tardiness_constraint = pyo.Constraint(
209     expr = model.L >= model.F[final_activity]-deadline)
210
211 solver = pyo.SolverFactory("gurobi")
212
213 # Gurobi settings
214 solver.options["Presolve"] = 2
215 solver.options["Cuts"] = 2
216 solver.options["Heuristics"] = 0.2
217 solver.options["MIPFocus"] = 1
218 solver.options["TimeLimit"] = 600
219
220
221 # Using lexicographic optimization
222 # First optimization: minimize cost + tardiness
223 start_time_1 = time.time()
224 results = solver.solve(model, tee=True)
225 runtime_1 = time.time() - start_time_1
226
227 acceptable = [
228     TerminationCondition.optimal,
229     TerminationCondition.feasible,
230     TerminationCondition.maxTimeLimit]
231
232 # Otherwise entire experiment stops after the first infeasible instance
233 if results.solver.termination_condition not in acceptable:
234     print("No feasible solution.")
235     results_table.append({
236         "instance": file_path.name,
237         "activities": len(activities),
238         "objective": None,
239         "makespan": None,
240         "tardiness": None,
241         "runtime": None,
242         "binary_variables": None,
243         "constraints": None,
244         "status": str(results.solver.termination_condition)})
245     continue
246
247 best_objective = pyo.value(sum(costs[j,m] * model.x[j,t,m]
248     for (j,t,m) in feasible_indices) + alpha * model.L)
249
250 # Second optimization: minimize makespan among optimal solutions
251 model.fix_objective = pyo.Constraint(
252     expr=sum(costs[j,m] * model.x[j,t,m]
253     for (j,t,m) in feasible_indices)+alpha * model.L <= best_objective + 1e-6)
254
255 model.objective.deactivate()
256 # Remove old objective if rerunning in Spyder

```

```

257     if hasattr(model, "makespan_objective"):
258         model.del_component(model.makespan_objective)
259
260     model.makespan_objective = pyo.Objective(
261         expr=model.F[final_activity], sense=pyo.minimize)
262
263     # Resolve
264     start_time_2 = time.time()
265     results = solver.solve(model, tee=True)
266     runtime_2 = time.time() - start_time_2
267
268     # Total combined solver runtime
269     runtime = runtime_1 + runtime_2
270
271     # Computational metrics
272     num_binary_variables = sum(1 for v in model.component_data_objects(pyo.Var) if v.
273                                is_binary())
274     num_constraints = sum(1 for c in model.component_data_objects(pyo.Constraint, active=True
275                                                                    ))
276
277     # Print solution
278     if (results.solver.status == SolverStatus.ok and results.solver.termination_condition
279         in [TerminationCondition.optimal,
280             TerminationCondition.feasible,
281             TerminationCondition.maxTimeLimit]):
282         print("\nOPTIMAL SCHEDULE")
283         for (j,t,m) in feasible_indices:
284             if model.x[j,t,m].value is not None:
285                 if pyo.value(model.x[j,t,m]) > 0.5:
286                     duration = durations[j,m]
287                     start_time = t-duration
288                     print(
289                         f"Activity_{j:2d}_{t:2d}"
290                         f"Mode_{m}_{t:2d}"
291                         f"Start_{start_time:2d}_{t:2d}"
292                         f"Finish_{t:2d}")
293
294         print("\n")
295         print("Project_finish:", pyo.value(model.F[final_activity]))
296         print("Tardiness:", pyo.value(model.L))
297         print("Objective:", pyo.value(sum(costs[j,m] * model.x[j,t,m]
298                                         for (j,t,m) in feasible_indices) + alpha * model.L))
299     else:
300         print("\nNo feasible solution found.")
301         print("Status:", results.solver.status)
302         print("Termination:", results.solver.termination_condition)
303
304     results_table.append({
305         "instance": file_path.name,
306         "activities": len(activities),
307         "objective": pyo.value(sum(costs[j,m] * model.x[j,t,m]
308                                   for (j,t,m) in feasible_indices) + alpha * model.L),
309         "makespan": pyo.value(model.F[final_activity]),
310         "tardiness": pyo.value(model.L),
311         "runtime": runtime,
312         "binary_variables": num_binary_variables,
313         "constraints": num_constraints,
314         "status": str(results.solver.termination_condition)})
315
316     print("\n")
317     print("=" * 70)
318     print("SUMMARY")
319     print("=" * 70)

```

D.2. Bianco--Caramia Formulation

```

1 # Python import statements
2 import pyomo.environ as pyo
3 import time
4 from dataset_augmented import load_augmented
5 from pathlib import Path

```

```

6 from pyomo.opt import SolverStatus, TerminationCondition
7
8 # Load all instances
9 base = Path(__file__).parent
10 instance_folder = base / "generated_instances_20"
11 instance_files = sorted(instance_folder.glob("*.json"))
12
13 results_table = []
14
15 for file_path in instance_files:
16     print("\n")
17     print("=" * 70)
18     print("INSTANCE:", file_path.name)
19     print("=" * 70)
20     data = load_augmented(str(file_path))
21
22     # Extract data
23     activities = data["activities"]
24     resources = data["resources"]
25     modes = data["modes"]
26     durations = data["durations"]
27     costs = data["costs"]
28     resource_usage = data["resource_usage"]
29     resource_capacities = data["resource_capacities"]
30     predecessors = data["predecessors"]
31
32     # Critical path
33     fast_mode = {}
34     slow_mode = {}
35
36     for j in activities:
37         fast_mode[j] = min(modes[j], key=lambda m: durations[j, m])
38         slow_mode[j] = min(modes[j], key=lambda m: costs[j, m])
39
40     def critical_path_duration(selected_modes):
41         EF = {}
42         for j in activities:
43             d = durations[j, selected_modes[j]]
44             if len(predecessors[j]) == 0: ES = 0
45             else: ES = max(EF[h] for h in predecessors[j])
46             EF[j] = ES + d
47         return EF[max(activities)]
48
49     CP_min = critical_path_duration(fast_mode)
50     CP_max = critical_path_duration(slow_mode)
51
52     print("Shortest CP:", CP_min)
53     print("Longest CP:", CP_max)
54
55     # Deadline & alpha
56     beta = 0.5
57     deadline = int(CP_min + beta * (CP_max - CP_min))
58     print("Deadline:", deadline)
59
60     C_min = sum(costs[j, fast_mode[j]] for j in activities)
61     C_max = sum(costs[j, slow_mode[j]] for j in activities)
62     average_cost = (C_min + C_max)/2
63     average_cp = (CP_min + CP_max)/2
64     alpha = average_cost/average_cp
65     print("Alpha:", alpha)
66
67     # Time horizon
68     UB = CP_max
69     time_periods = range(UB + 1)
70     print("Time horizon:", UB)
71
72     # Create Pyomo model
73     model = pyo.ConcreteModel()
74
75     # Compute earliest start times
76     earliest_start = {}

```

```

77     for j in activities:
78         if len(predecessors[j]) == 0:
79             earliest_start[j] = 0
80         else:
81             earliest_start[j] = max(earliest_start[h] + min(durations[h, m]
82                 for m in modes[h])
83                 for h in predecessors[j])
84
85     # Variables
86     # Sparse feasible variable generation
87     feasible_indices = []
88     for j in activities:
89         for m in modes[j]:
90             d = durations[j, m]
91             # Dummy activity
92             if d == 0:
93                 feasible_indices.append((j, 0, m))
94             else:
95                 latest_start = UB - d
96                 for t in range(latest_start + 1):
97                     # Earliest-start filtering
98                     if t >= earliest_start[j]:
99                         feasible_indices.append((j, t, m))
100 model.x = pyo.Var(feasible_indices, within=pyo.Binary)
101
102     # Fast access structures
103     activity_vars = {j: [] for j in activities}
104     activity_mode_vars = {}
105     for (j, t, m) in feasible_indices:
106         activity_vars[j].append((t, m))
107         if (j, m) not in activity_mode_vars:
108             activity_mode_vars[j, m] = []
109         activity_mode_vars[j, m].append(t)
110
111     # Finish times
112     model.F = pyo.Var(activities, within=pyo.NonNegativeReals)
113
114     # Tardiness variable
115     model.L = pyo.Var(within=pyo.NonNegativeReals)
116
117     # Objective
118     def objective_rule(model):
119         mitigation_cost = sum(costs[j, m] * model.x[j, t, m]
120             for (j, t, m) in feasible_indices)
121         return mitigation_cost + alpha * model.L
122     model.objective = pyo.Objective(rule=objective_rule, sense=pyo.minimize)
123
124     # Execute once
125     model.execute_once = pyo.ConstraintList()
126     for j in activities:
127         model.execute_once.add(sum(model.x[j, t, m]
128             for (jj, t, m) in feasible_indices if jj == j) == 1)
129
130     # Finish time definition
131     model.finish_constraints = pyo.ConstraintList()
132     for j in activities:
133         finish_expr = sum((t + durations[j, m]) * model.x[j, t, m]
134             for (t, m) in activity_vars[j])
135         model.finish_constraints.add(model.F[j] == finish_expr)
136
137     # Precedence constraints
138     model.precedence_constraints = pyo.ConstraintList()
139     for j in activities:
140         for h in predecessors[j]:
141             start_j = sum(t * model.x[j, t, m]
142                 for (t, m) in activity_vars[j])
143             model.precedence_constraints.add(model.F[h] <= start_j)
144
145     # Resource constraints
146     # Precompute active variables
147     active_by_time = {tau: [] for tau in time_periods}

```

```

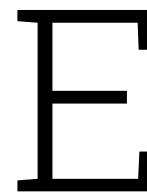
148     for (j, t, m) in feasible_indices:
149         d = durations[j, m]
150         for tau in range(t, t + d):
151             if tau in active_by_time:
152                 active_by_time[tau].append((j, t, m))
153     # Precompute resource terms
154     resource_terms = {}
155     for k in resources:
156         for tau in time_periods:
157             terms = []
158             for (j, t, m) in active_by_time[tau]:
159                 usage = resource_usage.get((j, m, k), 0)
160                 if usage > 0:
161                     terms.append(usage * model.x[j, t, m])
162             resource_terms[k, tau] = terms
163     # Resource constraints
164     model.resource_constraints = pyo.ConstraintList()
165     for k in resources:
166         capacity = resource_capacities[str(k)]
167         for tau in time_periods:
168             terms = resource_terms[k, tau]
169             if terms:
170                 model.resource_constraints.add(sum(terms) <= capacity)
171
172     # Tardiness constraint
173     final_activity = max(activities)
174     model.tardiness_constraint = pyo.Constraint(
175         expr=model.L >= model.F[final_activity] - deadline)
176
177     # Solver
178     solver = pyo.SolverFactory("gurobi")
179
180     solver.options["Presolve"] = 2
181     solver.options["Cuts"] = 2
182     solver.options["Heuristics"] = 0.2
183     solver.options["MIPFocus"] = 1
184     solver.options["TimeLimit"] = 600
185
186     # Lexicographic optimization:
187     # First optimization: minimize mitigation cost + tardiness
188     start_time_1 = time.time()
189     results = solver.solve(model, tee=True)
190     runtime_1 = time.time() - start_time_1
191
192     acceptable = [
193         TerminationCondition.optimal,
194         TerminationCondition.feasible,
195         TerminationCondition.maxTimeLimit]
196
197     # Otherwise entire experiment stops after the first infeasible instance
198     if results.solver.termination_condition not in acceptable:
199         print("No feasible solution.")
200         results_table.append({
201             "instance": file_path.name,
202             "activities": len(activities),
203             "objective": None,
204             "makespan": None,
205             "tardiness": None,
206             "runtime": None,
207             "binary_variables": None,
208             "constraints": None,
209             "status": str(results.solver.termination_condition)})
210         continue
211
212     best_objective = pyo.value(model.objective)
213
214     # Second optimization: minimize makespan among best objective solutions
215     model.fix_objective = pyo.Constraint(
216         expr= sum(costs[j, m] * model.x[j, t, m]
217                 for (j, t, m) in feasible_indices) + alpha * model.L <= best_objective + 1e-6)
218

```

```

219 model.objective.deactivate()
220 model.makespan_objective = pyo.Objective(
221     expr=model.F[final_activity], sense=pyo.minimize)
222
223 # Resolve
224 start_time_2 = time.time()
225 results = solver.solve(model, tee=True)
226 runtime_2 = time.time() - start_time_2
227
228 # Total combined solver runtime
229 runtime = runtime_1 + runtime_2
230
231 # Computational metrics
232 num_binary_variables = sum(1 for v in model.component_data_objects(pyo.Var) if v.
    is_binary())
233 num_constraints = sum(1 for c in model.component_data_objects(pyo.Constraint, active=True
    ))
234
235 # Print results
236 if (results.solver.status == SolverStatus.ok and results.solver.termination_condition
237     in [
238         TerminationCondition.optimal,
239         TerminationCondition.feasible,
240         TerminationCondition.maxTimeLimit]):
241     print("\n")
242     print("=" * 60)
243     print("SOLUTION")
244     print("=" * 60)
245
246     for (j, t, m) in feasible_indices:
247         val = model.x[j, t, m].value
248         if val is not None and val > 0.5:
249             d = durations[j, m]
250             print(
251                 f"Activity_{j:2d}|_"
252                 f"Mode_{m}|_"
253                 f"Start_{t:3d}|_"
254                 f"Finish_{t+d:3d}|_"
255                 f"Duration_{d:3d}")
256     print("\n")
257     print("Project_finish:", pyo.value(model.F[final_activity]))
258     print("Tardiness:", pyo.value(model.L))
259     objective_value = pyo.value(sum(costs[j, m] * model.x[j, t, m]
260         for (j, t, m) in feasible_indices) + alpha * model.L)
261     print("Objective:", objective_value)
262 else:
263     print("\nNo feasible solution found.")
264     print("Status:", results.solver.status)
265     print("Termination:", results.solver.termination_condition)
266
267 results_table.append({
268     "instance": file_path.name,
269     "activities": len(activities),
270     "objective": pyo.value(sum(costs[j,m] * model.x[j,t,m]
271         for (j,t,m) in feasible_indices) + alpha * model.L),
272     "makespan": pyo.value(model.F[final_activity]),
273     "tardiness": pyo.value(model.L),
274     "runtime": runtime,
275     "binary_variables": num_binary_variables,
276     "constraints": num_constraints,
277     "status": str(results.solver.termination_condition)})
278
279 print("\n")
280 print("=" * 70)
281 print("SUMMARY")
282 print("=" * 70)

```



Dataset Generation and Preprocessing

E.1. Dataset Augmentation

The following script was used to create augmented benchmark datasets and generate renewable resource data.

```
1 from pathlib import Path
2 import math
3 import json
4 import random
5
6 # Paths
7 SOURCE = (
8     Path(__file__).parent
9     / "DTCTP_test_beds"
10    / "test_beds_2"
11    / "net0_2")
12
13 TARGET = (Path(__file__).parent
14           / "generated_instances_20")
15
16 TARGET.mkdir(exist_ok=True)
17
18 random.seed(42)
19
20 TARGET_NODES = 20
21
22 # Main loop
23 for file in SOURCE.glob("*"):
24     if not file.is_file():
25         continue
26     print("Processing:", file.name)
27     with open(file, "r") as f:
28         lines = [line.strip()
29                  for line in f
30                  if line.strip()]
31     ptr = 0
32     n_nodes = int(lines[ptr])
33     ptr += 1
34     original_activities = list(range(n_nodes))
35     predecessors = {j: [] for j in original_activities}
36     successors = {}
37
38     # Precedence
39     for _ in range(n_nodes):
40         row = list(map(int, lines[ptr].split()))
41         ptr += 1
42         node = row[0]
43         n_succ = row[1]
44         succ = row[2:2+n_succ]
```

```

45     successors[node] = succ
46     for s in succ:
47         predecessors[s].append(node)
48
49 # Modes
50 modes = {}
51 durations = {}
52 costs = {}
53 for _ in range(n_nodes):
54     row1 = list(map(int, lines[ptr].split()))
55     ptr += 1
56     node = row1[0]
57     n_modes = min(3, row1[1])
58     row2 = list(map(int, lines[ptr].split()))
59     ptr += 1
60     modes[node] = list(range(n_modes))
61     for m in range(n_modes):
62         # SCALE DOWN DURATIONS
63         duration = max(1, int(row2[2*m] / 4))
64         # SCALE COSTS
65         cost = max(1, int(row2[2*m+1] / 10))
66         durations[node, m] = duration
67         costs[node, m] = cost
68
69 # Keep only 20 activities
70 start_dummy = 0
71 end_dummy = n_nodes - 1
72 removable = list(range(1, n_nodes - 1))
73 n_remove = n_nodes - TARGET_NODES
74 remove_nodes = set(random.sample(removable, n_remove))
75 kept_nodes = [j for j in original_activities
76               if j not in remove_nodes]
77
78 # Reconnect network
79 new_successors = {j: [] for j in kept_nodes}
80 for j in kept_nodes:
81     frontier = successors[j][:]
82     visited = set()
83     while frontier:
84         s = frontier.pop(0)
85         if s in visited:
86             continue
87         visited.add(s)
88         if s in kept_nodes:
89             if s not in new_successors[j]:
90                 new_successors[j].append(s)
91         else:
92             frontier.extend(successors[s])
93
94 # Rebuild predecessors
95 new_predecessors = {j: [] for j in kept_nodes}
96 for j in kept_nodes:
97     for s in new_successors[j]:
98         new_predecessors[s].append(j)
99
100 # Renumber
101 mapping = {}
102 for idx, old in enumerate(sorted(kept_nodes)):
103     mapping[old] = idx
104 activities = list(range(TARGET_NODES))
105 predecessors_final = {}
106 modes_final = {}
107 durations_final = {}
108 costs_final = {}
109 for old_j in kept_nodes:
110     new_j = mapping[old_j]
111     predecessors_final[new_j] = [mapping[h] for h in new_predecessors[old_j]]
112     modes_final[new_j] = modes[old_j]
113     for m in modes[old_j]:
114         durations_final[new_j, m] = durations[old_j, m]
115         costs_final[new_j, m] = costs[old_j, m]

```

```

116
117 # Resources
118 resources = [1, 2]
119 resource_usage = {}
120 for j in activities:
121     pmax = max(durations_final[j, m] for m in modes_final[j])
122     # Much smaller resource usage
123     base_worker = max(1, round(pmax / 20))
124     base_machine = max(1, round(pmax / 30))
125     for m in modes_final[j]:
126         duration = durations_final[j, m]
127         if duration == 0:
128             resource_usage[j, m, 1] = 0
129             resource_usage[j, m, 2] = 0
130         else:
131             factor = pmax / duration
132             # Softer scaling
133             resource_usage[j, m, 1] = max(1, math.ceil(base_worker * factor * 0.5))
134             resource_usage[j, m, 2] = max(1, math.ceil(base_machine * factor * 0.5))
135
136 # Resource capacities
137 # Much looser capacities
138 max_r1 = max(resource_usage[j, m, 1]
139               for j in activities
140               for m in modes_final[j])
141 max_r2 = max(resource_usage[j, m, 2]
142               for j in activities
143               for m in modes_final[j])
144 cap1 = math.ceil(8 * max_r1)
145 cap2 = math.ceil(8 * max_r2)
146 resource_capacities = {1: cap1, 2: cap2}
147
148 # Save
149 out = TARGET / (file.stem + "_20.json")
150 instance = {
151     "activities": activities,
152     "resources": resources,
153     "modes": modes_final,
154     "durations": {str(k): v for k, v in durations_final.items()},
155     "costs": {str(k): v for k, v in costs_final.items()},
156     "resource_usage": {str(k): v for k, v in resource_usage.items()},
157     "resource_capacities": {1: cap1, 2: cap2},
158     "predecessors": predecessors_final}
159 with open(out, "w") as f:
160     json.dump(instance, f, indent=4)
161 print("\nFinished generating stabilized instances.")

```

E.2. Data Loading

```

1 import json
2
3 def load_augmented(filename):
4     with open(filename, "r") as f: data = json.load(f)
5     durations = {eval(k):v
6                  for k,v in data["durations"].items()}
7     costs = {eval(k):v
8              for k,v in data["costs"].items()}
9     resource_usage = {eval(k):v
10                       for k,v in data["resource_usage"].items()}
11
12     data["durations"] = durations
13     data["costs"] = costs
14     data["resource_usage"] = resource_usage
15     data["activities"] = [int(j) for j in data["activities"]]
16     data["resources"] = [int(k) for k in data["resources"]]
17     data["modes"] = {int(j):v for j,v in data["modes"].items()}
18     data["predecessors"] = {int(j):v for j,v in data["predecessors"].items()}
19
20     return data

```