ALGORITHMS AND COMPLEXITY ANALYSIS FOR INTEGER MULTICOMMODITY NETWORK FLOW AND ROBUST SINGLE-MACHINE SCHEDULING PROBLEMS

By
BITA TADAYON

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
UNIVERSITY OF FLORIDA
2014
To my parents, Nader and Nooshin, for their endless love, support, and encouragement
ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Dr. J. Cole Smith, for his valuable support, guidance, and encouragement, throughout the course of my studies at University of Florida. Without his help and guidance, this dissertation would not be possible. He continually conveyed a spirit of adventure in regard to research and made this journey an exciting path toward becoming the researcher I am today. I am beyond grateful for having the opportunity to work with him and could not have wished for a better advisor. I would also like to thank Dr. Joseph Geunes, Dr. Yongpei Guan, and Dr. Sanjay Ranka for serving on my Ph.D. committee and for their insightful comments and suggestions.

During my years at University of Florida, I had the pleasure of meeting many amazing graduate student colleagues who became great friends of mine. I would like to thank my officemates, Soheil Hemmati, Mike Prince, and Deon Burchett for all the help and support they have given me during the last three years. I would also like to thank my dear friends Shantih Spanton, Cinthia Perez, Ayse Nur Arslan, Zehra Melis Teksan, Svetlana Moiseeva, Saeed Ghadimi, Andrew Romich, Jing Ma, Vijay Pappu, Chrysafis Vogiatzis, Jose Walteros, Jorge A. Sefair, Dmytro Korenkevych, Yen Tang, Chaoyue Zhao, Gregory Pastukhov, and Oleg Shirokikh for creating so many wonderful memories for me.

Last but certainly not least, my deepest gratitude goes to my dearest parents, Nader and Nooshin, who taught me how to follow my dreams and encouraged me in every decision I made in my life. I want to thank my dear uncle and aunt, Nasser and Fautee, for always being there for me and making me feel home while away from home. Finally I want to thank my boyfriend, Roosbeh, for his love and support and all the laughter he gave me throughout these years.
TABLE OF CONTENTS

ACKNOWLEDGMENTS ................................................................. 4
LIST OF TABLES ................................................................. 7
LIST OF FIGURES ............................................................... 8
ABSTRACT ............................................................................. 9

CHAPTER

1 INTRODUCTION ................................................................. 11

2 ALGORITHMS FOR AN INTEGER MULTICOMMODITY NETWORK FLOW PROBLEM WITH NODE RELIABILITY CONSIDERATIONS ............................................ 14

2.1 Motivation and Literature Review ........................................... 14
2.2 Problem Statement and Complexity ...................................... 17
  2.2.1 Notation and Description .............................................. 18
  2.2.2 Nonlinear Formulation of MCFNR .................................... 21
  2.2.3 Complexity Analysis ....................................................... 24
2.3 Linearization of the Mathematical Model ............................... 29
  2.3.1 Approach 1 .................................................................. 30
  2.3.2 Approach 2 .................................................................. 32
2.4 Lower- and Upper-Bounding Scheme ................................... 34
2.5 Cutting-Plane Algorithm ..................................................... 36
2.6 Computational Experiments ................................................ 38
  2.6.1 Test Problem Generation .............................................. 38
  2.6.2 Results ....................................................................... 40

3 A SURVEY OF ROBUST SINGLE-MACHINE SCHEDULING PROBLEM ...... 48

3.1 Motivation ........................................................................ 48
3.2 Robustness and Uncertainty Definitions ................................ 51
  3.2.1 Robustness Measures ................................................... 51
  3.2.2 Uncertainty Representation ........................................... 54
3.3 Robust Single-Machine Scheduling Literature Classification .... 56
3.4 Robust Optimization in Other Scheduling Problems .............. 61

4 ALGORITHMS AND COMPLEXITY ANALYSIS FOR ROBUST SINGLE-MACHINE SCHEDULING PROBLEMS ...................................................... 65

4.1 Motivation ....................................................................... 65
4.2 Problem Definition and Notation ........................................ 67
4.3 Complexity Results and Algorithms .................................... 69
  4.3.1 Minimizing Total Completion Time ................................ 70
4.3.1.1 Scenario Generation Problem ........................................ 70
4.3.1.2 Robust Optimization Problem ..................................... 74
4.3.2 Minimizing Total Weighted Completion Time ....................... 78
  4.3.2.1 Scenario Generation Problem .................................... 78
  4.3.2.2 Robust Optimization Problem .................................... 79
4.3.3 Minimizing Maximum Lateness/Tardiness ............................ 84
  4.3.3.1 Scenario Generation Problem .................................... 84
  4.3.3.2 Robust Optimization Problem .................................... 86
4.3.4 Minimizing Number of Late Jobs .................................... 87
  4.3.4.1 Scenario Generation Problem .................................... 87
  4.3.4.2 Robust Optimization Problem .................................... 91

5 COMPUTATIONAL ANALYSIS FOR A SPECIAL CASE OF ROBUST SINGLE-
MACHINE SCHEDULING PROBLEM ........................................... 106
  5.1 Approximate Solutions .................................................. 106
    5.1.1 Upper Bound ....................................................... 106
    5.1.2 Lower Bounds ..................................................... 108
      5.1.2.1 Single-Stage Approaches .................................... 108
      5.1.2.2 Multi-Stage Approaches ..................................... 110
    5.1.3 Test Problem Generation ......................................... 111
    5.1.4 Results ............................................................. 112
  5.2 Exact Solution ........................................................... 112

6 CONCLUSIONS AND FUTURE RESEARCH ................................. 120
REFERENCES ................................................................. 123
BIOGRAPHICAL SKETCH .................................................... 130
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Comparison of cut-generation strategies 1, 2, and 3</td>
<td>42</td>
</tr>
<tr>
<td>2-2</td>
<td>Comparison of GloMIQO and the S1 cutting-plane algorithm</td>
<td>44</td>
</tr>
<tr>
<td>2-3</td>
<td>Comparison of the three $U_i$- and $R_i$-value selection strategies</td>
<td>45</td>
</tr>
<tr>
<td>2-4</td>
<td>Comparison of lower- and upper-bounding models</td>
<td>47</td>
</tr>
<tr>
<td>3-1</td>
<td>Complexity results obtained in the literature for robust SMSP</td>
<td>60</td>
</tr>
<tr>
<td>4-1</td>
<td>Complexity results for robust SMSP under budgeted uncertainty</td>
<td>70</td>
</tr>
<tr>
<td>5-1</td>
<td>Evaluation of lower- and upper- bounding heuristic algorithms</td>
<td>113</td>
</tr>
<tr>
<td>5-2</td>
<td>Evaluation of exact solution method ($2 \leq n \leq 10$)</td>
<td>116</td>
</tr>
<tr>
<td>5-3</td>
<td>Evaluation of exact solution method ($n = 10$)</td>
<td>117</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Problem illustration</td>
<td>19</td>
</tr>
<tr>
<td>2-2</td>
<td>Converting the reliability on arc to the reliability on node</td>
<td>21</td>
</tr>
<tr>
<td>2-3</td>
<td>An example transformation from 3DM</td>
<td>27</td>
</tr>
<tr>
<td>2-4</td>
<td>Solution time of instances using three cutting-plane generation strategies</td>
<td>43</td>
</tr>
<tr>
<td>4-1</td>
<td>Creating sequence $\pi''$ from $\pi'$</td>
<td>74</td>
</tr>
<tr>
<td>4-2</td>
<td>Dynamic-programming SGP network with $\sum_{j \in J} U_j$ criterion and US2</td>
<td>90</td>
</tr>
<tr>
<td>4-3</td>
<td>SGP network for the $\sum_{j \in J} U_j$ criterion under US2</td>
<td>101</td>
</tr>
</tbody>
</table>
In this dissertation we address several optimization problems with uncertain data elements. For each problem we apply an appropriate method for uncertainty representation, propose mathematical formulations, and present solution methods. We first consider a multicommodity network flow problem in which intermediate nodes (hubs) may fail to successfully relay the flow. We model this uncertainty by associating each hub node with a reliability function, which depends on the total flow that crosses that hub. The probability that each commodity is successfully transferred from its origin to its destination is given by the product of hub reliabilities on the commodity’s path. We seek to find minimum-cost commodity paths such that each commodity reaches its destination with a sufficiently large probability. We first formulate the problem as a nonlinear multicommodity network flow problem and prove that it is strongly NP-hard. We then present two linearization techniques, and propose a pair of lower- and upper-bounding formulations, which can then be used within an exact cutting-plane algorithm to solve the problem. Finally, we analyze the computational effectiveness of our proposed strategies on a set of randomly generated instances.

As our second line of research we consider single-machine scheduling problems with uncertainty in their parameter values. We focus on robust optimization as an appropriate method of dealing with uncertainty in several scheduling environments. We
first present a comprehensive survey of robust single-machine scheduling problems, classify the literature, and introduce open problems in this area. This survey proposes the possibility of improving existing robust scheduling models by applying recent developments in robust optimization in this area. Accordingly, as the next step of our research we study a robust single-machine scheduling problem where job processing times are subject to uncertainty with their values belonging to independent continuous intervals. We consider four alternative optimization criteria and apply state-of-the-art robust optimization methods to define three different uncertainty sets. Then, given each combination of objective function and uncertainty set, we explore the problem of determining the worst-case scenario (job processing-time values) corresponding to a given job schedule, and analyze the problem of scheduling jobs to minimize the worst-case objective.
CHAPTER 1
INTRODUCTION

In the process of developing mathematical representations of real-world optimization problems, the ultimate goal is to obtain the simplest model that expresses the problem as accurately as possible. Researchers face a wide variety of challenges in problem representation, especially in the presence of uncertainty. Due to variations in process and environmental data, uncertainty is common in many practical problems. Based on available information about the behavior of uncertain parameters and the desired performance level, one must select an uncertainty representation method and prescribe an approach for solving the problem.

In our research we study the effect of uncertainty in two well-known integer programming problems: The multicommodity network flow problem and the single-machine scheduling problem. For each problem, we select an appropriate method to represent uncertainty, and explore alternative formulation strategies for the problem. We then investigate the problem complexity and prescribe algorithms for solving each problem.

This dissertation begins by considering the problem of sending a set of multiple commodities from their origin to destination nodes on a network in which intermediate nodes (hubs) may fail to correctly deliver flows. In several real-world applications, such as communication or transportation networks, the risk of failure in flow transmission through a hub node increases as the traffic passing through the hub increases. Accordingly, in Chapter 2 we model the uncertainty in flow transmission by defining the reliability of each hub node (i.e., the probability that it correctly relays each commodity flow that passes through it) as a nonincreasing function of the total flow that crosses that hub. Additionally, in many applications, such as transferring individual messages in a communication network or transporting goods in a railway network, the flow of each commodity cannot be split among different paths. Therefore, in Chapter 2 we require the flow between a commodity’s origin and destination to follow a single path...
A multicommodity flow problem with this assumption is referred to as an integer multicommodity flow problem. The probability that each commodity is successfully relayed from its origin to its destination is then given by the product of hub reliabilities on the commodity’s path. The problem seeks to find minimum-cost commodity paths such that each commodity reaches its destination with a sufficiently large probability. We formulate this problem as a nonlinear mixed-integer programming problem and prove that it is strongly NP-hard. We then present two linearization techniques for this formulation, and propose a pair of lower- and upper-bounding formulations, which can then be used within an exact cutting-plane algorithm to solve the problem.

The second research focus in this dissertation studies single-machine scheduling problems in the presence of uncertainty. In Chapter 3, after presenting an overview of different approaches to hedge against data uncertainty in scheduling problems, we focus on robust optimization as an appropriate method of dealing with uncertainty in several scheduling environments. In this chapter we present a comprehensive survey of robust optimization applications in solving single-machine scheduling problems. We introduce three different robustness measures and four different uncertainty sets that have been applied in the robust scheduling literature. We then classify the results obtained for the specific case of robust single-machine scheduling problems with different objective functions, under each combination of uncertainty set and robustness measure, and introduce open problems in this area. Our ultimate goal in this chapter is to address the growing gap between the literature of robust optimization and robust scheduling, and encourage the closure of this gap.

To take a step towards improving existing robust scheduling models, we apply state-of-the-art robust optimization techniques in Chapter 4 to define and solve a single-machine scheduling problem in which job processing times are uncertain. We assume that job processing-time values can be represented as independent continuous intervals and we seek to guarantee a minimum quality for the objective value in the
worst-case scenario. We define the problem as a robust optimization problem and introduce three alternative uncertainty sets to control the level of conservatism and moderate the worst-case scenario in the problem. We then study the problem under four alternative optimization criteria, specifically, minimizing total completion time, minimizing total weighted completion time, minimizing maximum lateness, and minimizing number of late jobs.

In Chapter 5 we focus on one specific case of the robust single-machine scheduling problem introduced in Chapter 4, where we seek to minimize number of late jobs in the worst-case scenario in which no more than a certain number of jobs can take on their worst-case processing-time values. We implement the mixed-integer programming (MIP) formulation presented for this problem in Chapter 4 and solve a set of randomly-generated test problems using CPLEX default MIP solvers. We also introduce an upper bound and alternative lower bounds for the problem and test their performance by applying them on a set of randomly-generated instances.
2.1 Motivation and Literature Review

The multicommodity flow (MCF) problem seeks to satisfy demands among a set of commodities at minimum cost, across a directed network having capacitated arcs. The commodities are associated with an origin and destination node, and with a demand quantity. Additionally, many applications require flow between a commodity’s origin and destination to follow a single path. Given flow costs for each arc on the network, the problem of simultaneously shipping all commodity demands on the network at a minimum cost where each commodity’s flow follows a single path is referred to as the integer multicommodity flow problem [10, 11].

This chapter examines integer MCFs on networks, in which intermediate nodes on an origin-destination path may fail to correctly deliver flows. For this problem, we assume that when a node fails to properly relay a commodity flow, the flow itself is propagated through the network as desired, but the contents of the flow have somehow been damaged. This may be the case in shipping fragile contents or in relaying information in a communication network. The reliability of each node (i.e., the probability that it correctly relays each commodity flow that passes through it) is modeled as a nonincreasing function of the load assigned to it, where load is given by the total amount of flow that crosses the node. Given an origin-destination route for each commodity, there exists a Boolean random variable corresponding to each commodity/node pair, which specifies whether or not the node will successfully relay the commodity flow. (For any node that does not serve as an intermediate node on the commodity’s path, the random variable is irrelevant.) We assume that these random variables are mutually independent, and so the probability of successfully transmitting flow on a path is calculated as the product of node reliabilities lying on the path. (For instance, if two commodities both send flow through a common intermediate node, then the probability
that this node successfully relays flow from the first commodity is independent of the probability that it successfully relays flow from the second commodity. Also, if two nodes both lie on some commodity’s path, then the probability that the first node successfully relays the commodity’s flow is independent of the probability that the second node successfully relays the flow.) The problem we examine is the integer MCF, subject to the restriction that each commodity must be successfully delivered with a sufficiently large (specified) probability. We call this the multicommodity flow problem with node reliability constraints (MCFNR).

Ahuja et al. [1] provide a general discussion of MCF models and algorithms. See also [5, 47] for comprehensive surveys on classical MCF research, and the recent survey by Ouorou et al. [67] which focuses on algorithms for solving nonlinear convex MCF problems.

Arc-based and path-based formulations are two commonly used strategies for modeling MCFs. Arc-based formulations include decision variables that determine how much flow for each commodity is shipped on each arc, and result in a polynomial number of constraints and variables. By contrast, path-based formulations require fewer constraints, but an exponential number of variables, one for every path connecting a commodity origin and destination. To solve the integer MCF using a path-based formulation, Barnhart et al. [10, 11] present a column-generation model for solving linear programming relaxations for the integer MCF, and prescribe a branch-and-price-and-cut approach for solving the problem. Brunetta et al. [20] study several classes of valid inequalities obtained from alternative formulations of the problem, and propose a branch-and-cut algorithm for solving the problem. They also study the polyhedral structure of the MCF polytope for the special case in which all flows are integer, all commodities have unit demand, and all arcs have unit capacities.

Similar to our problem, many MCF applications involve side constraints that must be satisfied in addition to the standard MCF constraints. Holmberg and Yuan [37] examine
time-delay and reliability side constraints for a communication network, in which the reliability of each path is calculated based on the arc failure rates that lie on that path. Distinct from our problem, the arc failure rates do not depend on the amount of traffic crossing the arc; moreover, their problem permits the use of multiple paths to send flows between each origin-destination pair and requires all paths to (independently) satisfy the minimum reliability requirement. To solve this problem, the authors prescribe a column-generation algorithm for solving a path-based model.

Network flow reliability problems have received much attention in the network flow optimization literature, particularly when arc or node reliabilities are not functions of the flow they transmit, as is the case in our study. In these studies, reliability is often defined as the probability of satisfying all commodity demands, given uncertain arc capacities. The problem of evaluating network reliability in this context is NP-hard [21]. Two primary approaches used to evaluate network reliability in the literature employ the concepts of Minimal Paths (MPs) and Minimal Cuts (MCs). Foundational approaches using MPs can be found in [6, 38, 55, 57, 60, 68, 82, 85]. Yeh generalizes the network reliability problem to consider cases with unreliable nodes [86] and with a budget constraint [88]. Lin [58, 59] extends the reliability problem to accommodate multicommodity cases. The MC approach has been used in [42, 74, 86, 87], and is closely related to MP algorithms by the max-flow min-cut theorem [30].

Another line of research regards network design problems in which a network’s topology and arc capacities are determined in a way that demands can be satisfied via a cost-efficient routing scheme [9, 19, 22, 32, 34, 42]. For instance, Gavish et al. [33] address the problem of designing a reliable network with minimum possible cost. They assume that arcs and nodes in the network have specific failure rates and the network state fluctuates based on the status of the arcs and nodes. The goal is to design a network such that the capacity assigned to each arc is large enough to satisfy the demands in all possible network states.
In contrast to the papers discussed above, MCFNR considers the situation in which node reliabilities depend on the amount of flow they transmit. To the best of our knowledge, the effect of congestion on the reliability of MCF problems as defined in our study has not been considered in the literature. However, in transportation applications the effect of congestion on travel time (which is analogous to our reliability analysis) has been studied comprehensively. Jahn et al. [40] consider the problem of routing flow in a transportation network in which all vehicles having a common origin and destination are assumed to be one commodity. In their model, travel time on each arc is assumed to be a differentiable nondecreasing function of the rate of traffic on that arc. Accordingly, traffic load on each arc affects the minimum total travel time. More work in the area of traffic congestion in transportation networks can be found in [79, 83, 89].

The remainder of this chapter is organized as follows. In Section 2.2, we develop a mathematical programming formulation for MCFNR and analyze the complexity of the problem. In Section 2.3, we introduce two approaches to linearize the mathematical formulation proposed in Section 2.2. We then present lower- and upper-bounding schemes for the problem in Section 2.4, and apply those bounds within a cutting-plane algorithm to solve the problem in Section 2.5. In Section 2.6, we employ our cutting-plane algorithm to solve a set of randomly generated instances of different sizes. We also compare the objective function values obtained from solving the lower- and upper-bounding models on our test problems.

2.2 Problem Statement and Complexity

We provide a formal description of MCFNR in Section 2.2.1, and present a mathematical programming formulation in Section 2.2.2. We then show that MCFNR is strongly NP-hard in Section 2.2.3, even under data assumptions that simplify the problem.
2.2.1 Notation and Description

We begin by providing notation and assumptions used for MCFNR. Define $K$ to be the set of commodities. For each commodity $k \in K$, define $O(k)$ as its origin node, $D(k)$ as its destination node, and $d_k$ as its demand quantity. Commodity $k \in K$ must be successfully transmitted to its destination with probability at least $\tau_k$, which is a parameter bounded by $0 < \tau_k \leq 1$. The cost of sending each unit of commodity $k$ on some arc $(i,j)$ in the network is denoted by $C_{ij}^k$.

We consider problems in which the node set $N$ is partitioned into terminals, $T$, and hubs, $H$. Set $T$ consists (only) of all nodes that serve as a commodity origin or destination node, i.e., $T = \bigcup_{k \in K} \{O(k), D(k)\}$. The network topology is such that all intermediate nodes in any $O(k)$–$D(k)$ path ($\forall k \in K$) belong to $H$. Furthermore, each feasible commodity path must consist of at least one hub, i.e., terminals cannot communicate directly with each other. (This assumption is not restrictive, and can be relaxed in the following analysis; however, it allows us to simplify our exposition.) Next, assume that all routes to and from a node $p \in T$ must transit through a single node $i \in H$; we henceforth say that node $i$ is assigned to terminal $p$. In many practical problem settings, it is necessary to limit the number of terminals to which nodes in $H$ can be assigned; we let $\gamma_i$, $\forall i \in H$, denote this limit. We now illustrate the MCFNR using the following example.

**Example 2.1.** Consider the two-commodity MCFNR instance depicted in Figure 2-1. The origin, destination, and demand values for each commodity, as well as the per-unit flow costs $C_{ij}^k$ on each arc, are specified in the figure (for simplicity, we assume that $C_{ij}^1 = C_{ij}^2$ for each arc $(i,j)$). The reliability of nodes $i = 1, \ldots, 4$ are given by $[1 - (\text{load of node } i)^2/100]$, and all arcs are uncapacitated. In this instance, if $\tau_1 = \tau_2 = 0.5$, then both paths use intermediate nodes 1 and 4, with a cost of 15: Five units of flow are shipped across nodes 1 and 4, and thus their reliabilities both equal 0.75. Independence of these reliabilities imply that the probability of transferring each
commodity successfully is 0.56, which satisfies the minimum probability requirement. However, if \( \tau_1 = \tau_2 = 0.6 \), then the previous solution is no longer feasible. For this new instance, commodity 1 uses intermediate nodes 1 and 4, but commodity 2 uses intermediate nodes 3 and 4 at optimality. The loads of nodes 1, 3, and 4 are then 3, 2, and 5, respectively, yielding corresponding node reliabilities of 0.91, 0.96, and 0.75. The reliability of the commodity 1 path is 0.68, and the reliability of the commodity 2 path is 0.72. Both probabilities satisfy the minimum requirement, and the total cost is 17.

![Figure 2-1. Problem illustration](image)

**Remark 2.1.** Without loss of generality, we assume that the terminals are perfectly reliable, whereas the reliability of each hub is a nonincreasing function of the amount of flow passing through the hub. Observe that the case of unreliable terminals can be handled by adjusting the \( \tau_k \)-values based on the total (fixed) flow originating from or arriving to each terminal. This generalization is valid because the load of each terminal is constant in any feasible solution, and is determined by the total demand of commodities that originate from or arrive at that terminal. For instance, suppose in Example 2.1 that the reliability of all nodes (hubs and terminals) are defined using the same reliability function \((1 - (\text{load of node } i)^2/100)\). In any feasible solution, each commodity passes through its origin and destination terminals, whose reliabilities would be 0.91 for \( O(1) \) and \( D(1) \) and 0.96 for \( O(2) \) and \( D(2) \). Now suppose that we desire
each commodity to be successfully transmitted to its destination with probability 0.5. We can treat the terminals as being perfectly reliable, and scale the reliability thresholds as $\tau_1 = 0.5/(0.91^2)$ and $\tau_2 = 0.5/(0.96^2)$. In this manner, all terminals can equivalently be treated as having perfect reliability.

The arc set in MCFNR is partitioned into arcs $A \subseteq \{(i, j)|i, j \in H\}$ having both incident nodes in $H$, and arcs $\tilde{A} \subseteq \{(i, p) \cup (p, i)|i \in H, p \in T\}$ that include all eligible terminal-hub assignments. Note that assignments are undirected, i.e., $(i, p) \in \tilde{A}$ if and only if $(p, i) \in \tilde{A}, \forall i \in H, p \in T$. Recall that the cost of transferring each unit of commodity $k \in K$ on arc $(i, j) \in A \cup \tilde{A}$ is given by $C^k_{ij}$. The problem is therefore to find an assignment of terminals to hubs, and a set of paths to transmit all commodities, at a minimum total cost, while guaranteeing that each commodity $k \in K$ is successfully transmitted to its destination with at least a probability of $\tau_k$.

**Remark 2.2.** The MCFNR assumes that nodes may fail to correctly relay flow, but arcs are perfectly reliable. This assumption is reasonable, for instance, when examining a network of people communicating electronically in an organization. Nodes represent the people, and arcs determine possible communications between those people. The probability of mistakes made by each person who is responsible for directing messages through the system increases as the workload on that person increases. Compared to human error rate, electronic communication (on the arcs) is virtually flawless. In other applications, however, it is not reasonable to assume that the arcs are perfectly reliable. Our model can easily be adjusted to cover those instances as well, by replacing each unreliable arc $(i, j)$ with two perfectly reliable arcs $(i, ij)$ and $(ij, j)$, where $ij$ is a dummy node whose reliability is defined using the initial arc’s reliability function. After this modification, the flow passing the dummy node is exactly the same as the initial arc flow, which yields an equivalent problem having perfectly reliable arcs. This modification is illustrated in Figure 2-2.
Generally, the assumption of having reliability functions on nodes results in a model which is more comprehensive compared to the one with arc reliability functions. This is true since the discussed modification can be applied to incorporate arcs with reliability functions into the model, even when the network is undirected. On the other hand, if arc reliability functions are used to build the model a similar modification can be used to incorporate the nodes with reliability functions into the model by splitting the node into two and putting a dummy arc with the same reliability function in between for directed networks. However, the modification technique fails when the network is undirected.

2.2.2 Nonlinear Formulation of MCFNR

We now formulate MCFNR as a mixed-integer nonlinear programming (MINLP) problem. We begin by addressing the hub reliability functions, which are nonincreasing functions of the hub’s load. We allow the hub reliability function to take on any form (assuming that reliability is a nonincreasing function of load). However, one recurring example in this chapter examines the case in which the reliability function of hub $i \in H$ is given by $1 - [(\text{load of node } i)^2 / m_i^2]$, where parameter $m_i$ is the maximum load that hub $i$ can take before its reliability drops to zero. This particular function captures the case in which reliability degrades at an increasing rate as its load increases (i.e., as a concave function of load), up until the point that the hub reliability drops to zero. (Note that this function necessitates additional constraints that restrict the load of node $i$ to be no more than $m_i$.)

To formulate MCFNR, we first define the following sets that appear in our formulation. For each $i \in H$, define the forward star, $FS(i) = \{j \in H : (i, j) \in A\}$ and the reverse
star, $RS(i) = \{h \in H : (h, i) \in A\}$. Similarly, let $\tilde{FS}(p) = \{i \in H : (p, i) \in \tilde{A}\}$ represent the set of hubs that can be assigned to terminal $p \in T$, and also define $\tilde{RS}(i) = \{p \in T : (p, i) \in \tilde{A}\}$ as the set of terminals that can be assigned to hub $i \in H$. For every $p \in T$, define $\kappa(p) = \{k \in K : O(k) = p \text{ or } D(k) = p\}$ as the set of commodities whose origin or destination terminal is $p$.

Our formulation utilizes the following decision variables.

$$Y_{pi} = \begin{cases} 1, & \text{if hub } i \text{ is assigned to terminal } p \\ 0, & \text{otherwise,} \end{cases} \quad \forall (p, i) \in \tilde{A}$$

$$X^k_{ij} = \begin{cases} 1, & \text{if arc } (i, j) \text{ is used to transfer commodity } k \\ 0, & \text{otherwise,} \end{cases} \quad \forall (i, j) \in A, \ k \in K$$

$$U_i = \text{total amount of flow passing through hub } i, \forall i \in H$$

$$R_i = \text{reliability of hub } i, \forall i \in H$$

We propose the following MINLP formulation for MCFNR, given the capacity functions described above, where we define $Y_{pi} \equiv 0, \forall p \in T, i \in H : (p, i) \notin \tilde{A}$.

$$\text{Min} \quad \sum_{(i,j) \in A} \sum_{k \in K} C^k_{ij} d_k X^k_{ij} + \sum_{(p,i) \in \tilde{A}} \sum_{k \in \kappa(p)} C^k_{pi} d_k Y_{pi} \quad (2-1)$$

subject to:

$$\sum_{i \in FS(p)} Y_{pi} = 1, \quad \forall p \in T \quad (2-2)$$

$$\sum_{p \in RS(i)} Y_{pi} \leq \gamma_i, \quad \forall i \in H \quad (2-3)$$

$$\sum_{j \in FS(i)} X^k_{ij} - \sum_{h \in RS(i)} X^k_{hi} = Y_{O(k), i} - Y_{D(k), i}, \quad \forall k \in K, \ i \in H \quad (2-4)$$

$$U_i = \sum_{k \in K} d_k \left( \sum_{j \in FS(i)} X^k_{ij} + Y_{D(k), i} \right), \quad \forall i \in H \quad (2-5)$$

$$R_i = 1 - \frac{U_i^2}{m_i^2}, \quad \forall i \in H \quad (2-6)$$
The objective function (2–1) minimizes the cost of transferring all commodities' flows among the hubs (the first term) and between each assigned terminal-hub pair (the second term). Constraints (2–2) ensure that exactly one hub is assigned to each terminal and constraints (2–3) impose an upper bound, $\gamma_i$, on the number of terminals that can be assigned to hub $i \in H$. The flow-balance constraints corresponding to each commodity $k \in K$ at each hub $i \in H$ are stated by constraints (2–4). For every hub $i \in H$, constraint (2–5) defines the hub load, $U_i$, as the sum of the demand values over all commodities that pass through hub $i$. The reliability of each hub $i \in H$ is then defined in constraint (2–6). Constraint (2–7) guarantees that the hub load is no more than $m_i$ for each hub $i \in H$.

Constraints (2–8) state a reliability threshold inequality for every commodity. In these constraints, for each commodity $k \in K$, the statement in parentheses represents the reliability of the first hub visited in the path of that commodity. This value is then multiplied by reliability of all other hubs on the path for $k$ to calculate the probability of successfully transferring commodity $k$ to its destination. Finally, constraints (2–9) and (2–10) represent logical restrictions on $X$- and $Y$-variables, respectively. Note that the nonnegativity of the $U$- and $R$-variables is implied by (2–5), (2–6), and (2–7), along with the nonnegativity of the $d$-values, $X$-variables, and $Y$-variables. We now employ a similar approach used by Andreas and Smith [4] to reformulate constraints (2–8). Define $s^k_i$ to be the probability that commodity $k$ successfully reaches hub $i$ from its origin terminal, for $k \in K$, given that its path visits hub $i$. Constraints (2–8) can be substituted
by the following inequalities:

\[
\begin{align*}
    s^k_j & \leq R^k_j s^k_i + (1 - X^k_j), & \forall k \in K, (i,j) \in A & \quad (2-11) \\
    s^k_i & \leq 1 - (1 - R^k_i) Y^k_{O(k),i}, & \forall k \in K, i \in \tilde{FS}(O(k)) & \quad (2-12) \\
    s^k_i & \geq \tau^k, & \forall k \in K, i \in H. & \quad (2-13)
\end{align*}
\]

Constraints (2–11) and (2–12) can be tightened by noting that if hub \( i \) lies on the (unique) path for commodity \( k \), then at least \( d^k_i \) units of flow passes through this node, and so \( s^k_i \) may not exceed \( \alpha^k = 1 - d^k_i / m^k_i \). Moreover, if we know that \( i \) is not the first hub visited in the path for commodity \( k \), then we also know that the message has already passed at least one other hub. Therefore, for these hubs the upper bound on \( s^k_i \) will be \( (\alpha^k)^2 \). As a result, constraints (2–11) and (2–12) can be revised as:

\[
\begin{align*}
    s^k_j & \leq R^k_j s^k_i + (\alpha^k)^2 \left( 1 - X^k_j - \left[ 1 - \frac{1}{\alpha^k} \right] Y^k_{O(k),j} \right), & \forall k \in K, (i,j) \in A & \quad (2-14) \\
    s^k_i & \leq (\alpha^k)^2 - ((\alpha^k)^2 - R^k_i) Y^k_{O(k),i}, & \forall k \in K, i \in \tilde{FS}(O(k)). & \quad (2-15)
\end{align*}
\]

Note that in our MINLP formulation, variables \( U_i \) and \( R_i \), as well as constraints (2–5) and (2–6) are included in the model for the sake of simplicity and can be removed by substituting the corresponding values in the remaining constraints. Our MINLP formulation thus contains \( O(|\tilde{A}| + |H||K| + |A||K|) \) variables and \( O(|T| + |H||K| + |A||K|) \) constraints.

**2.2.3 Complexity Analysis**

Because the integer MCF is strongly NP-hard, it is not surprising that MCFNR is also strongly NP-hard. In fact, we show in this section that MCFNR remains strongly NP-hard even for the simplified case in which all commodities have unit demands, all hubs are subject to a common affine reliability function, the minimum required probability of successfully transmitting each commodity is the same value \( \overline{\tau} \) for all commodities, and there is no cost for transferring flow in the network.
Theorem 2.1. The special case of MCFNR in which \( d_k = 1 \) and \( \tau_k = \bar{\tau}, \forall k \in K; C^k_{ij} = 0, \forall (i,j) \in A \cup \tilde{A}, k \in K; \) and \( R_i = f(U_i), \forall i \in H, \) is NP-hard, where \( f \) is an affine function and \( 0 < \bar{\tau} < 1 \) is a constant parameter.

**Proof.** We begin by defining the decision problem, MCFNRD, corresponding to MCFNR as follows: Does there exist a feasible MCFNR solution, regardless of cost? MCFNRD belongs to NP because we can verify if a routing scheme is feasible in polynomial time by enumerating the set of commodities passing through each hub, calculating the load of each hub, and then determining the probability of successfully delivering each commodity. This procedure requires \( O(|H||K|) \) steps, and so MCFNRD belongs to NP.

Next, we show that MCFNRD is NP-complete. Our proof employs a polynomial transformation from the 3-DIMENSIONAL MATCHING (3DM) problem (known to be strongly NP-complete \([31]\)) to an equivalent instance of MCFNRD.

**3DM:** Let \( W, X, \) and \( Y \) be finite, disjoint sets with \( |W| = |X| = |Y| = \delta, \) and define \( Z \subseteq W \times X \times Y \) as a set of triples \((w, x, y)\) such that \( w \in W, \)
\( x \in X, \) and \( y \in Y. \) (We assume \( \delta \geq 2. \)) A 3DM solution consists of a subset \( M \subset Z \) such that \( |M| = \delta \) and for any two distinct triples \((w_1, x_1, y_1) \in M \) and \((w_2, x_2, y_2) \in M, \) we have \( w_1 \neq w_2, x_1 \neq x_2, \) and \( y_1 \neq y_2. \)

To transform an arbitrary 3DM instance to an equivalent MCFNRD instance, we create a network consisting of \( \delta \) groups of “regular” hubs, plus an additional group of “dummy” hubs. The \( i^{th} \) regular group includes \( 2\delta \) regular hubs: \( \delta \) hubs \( \{x^i_1, \ldots, x^i_\delta\} \) corresponding to \( x_i \in X \) and \( \delta \) hubs \( \{y^i_1, \ldots, y^i_\delta\} \) corresponding to \( y_i \in Y. \) There are also \( 2\delta \) dummy hubs, given by \( \{x^d_1, \ldots, x^d_\delta\} \) and \( \{y^d_1, \ldots, y^d_\delta\}. \)

We next introduce a set of \( 4\delta \) commodities, each having a unit demand, which for ease in exposition we classify into three categories: primary (\( \delta \) commodities), secondary (\( 2\delta \) commodities), and dummy (\( \delta \) commodities). Set \( T \) consists of \( 2\delta + 6 \) terminals. Primary commodity \( i \in \{1, \ldots, \delta\} \) has an origin terminal \( w_i \) and a destination terminal \( t_i. \) For the secondary commodities, \( \delta \) commodities have a common origin \( O_x \) and
destination $D_x$, and the other $\delta$ commodities have a common origin $O_y$, and destination $D_y$. Finally, all dummy commodities have a common origin $w_d$ and destination $t_d$.

Next, we construct the set $\tilde{A}$ in the transformed MCFNRD instance by specifying $\tilde{FS}(p)$ for every $p \in T$. First, $\tilde{FS}(w_i), \forall i = 1, \ldots, \delta,$ is the set of all $x_i^j$ such that $(w_i, x_i, y_k) \in Z$ for some $y_k \in Y$. Also,

$$\tilde{FS}(t_i) = \{y_i^1, \ldots, y_i^\delta\}, \forall i = 1, \ldots, \delta.$$ 

For the secondary terminals, we have:

$$\tilde{FS}(O_x) = \{x_1^1, \ldots, x_1^\delta\},$$

$$\tilde{FS}(O_y) = \{y_1^1, \ldots, y_1^\delta\},$$

$$\tilde{FS}(D_x) = \{x_\delta^1, \ldots, x_\delta^\delta\},$$

and

$$\tilde{FS}(D_y) = \{y_\delta^1, \ldots, y_\delta^\delta\}.$$ 

Finally, $\tilde{FS}(w_d) = \{x_d^1, \ldots, x_d^\delta\}$, and $\tilde{FS}(t_d) = \{y_d^1, \ldots, y_d^\delta\}$. We now list all arcs that belong to $A$.

- for each $i \in \{1, \ldots, \delta - 1\}$ an arc exists from $x_i^j$ to $x_{i+1}^j$ and from $y_i^j$ to $y_{i+1}^j$,
- for each $(w_i, x_i, y_k) \in M$, there exists an arc from $x_i^j$ to $y_i^k$,
- for each $i \in \{1, \ldots, \delta\}$, arcs $(x_i^j, x_i^{j+1}), (x_i^{j+1}, y_i^k)$, and $(y_i^k, y_i^{j+1})$ exist.

Note that only $\delta$ paths, each of the form $O_x-x_1^1-\ldots-x_\delta^1-D_x$, $\forall i \in \{1, \ldots, \delta\}$, connect $O_x$ and $D_x$; only $\delta$ paths, $O_y-y_1^1-\ldots-y_\delta^1-D_y$, $\forall i \in \{1, \ldots, \delta\}$, connect $O_y$ and $D_y$; and only $\delta$ paths, $w_d-x_d^1-\ldots-x_d^\delta-t_d$, $\forall i \in \{1, \ldots, \delta\}$, connect the dummy terminals.

We choose the hub reliability function and threshold reliability values so that each path in a feasible MCFNRD solution visits at most two hubs having a load of two, with all other nodes visited by the path having a load of one. We use a linear hub reliability function given by $R_i = 1 - \varepsilon(U_i - 1), \forall i \in H$, for some $0 < \varepsilon \leq 1/2$. Having a minimum required probability of $\tau_i = (1 - \varepsilon)^2, \forall i \in H$, will then ensure that the specified conditions are satisfied: Because $1 - 2\varepsilon < (1 - \varepsilon)^2$, no hub can have a load exceeding two in a
feasible solution, and because $R_i = (1 - \varepsilon)$ when $U_i = 2$, each path visits no more than two hubs having a load of two.

Figure 2-3 illustrates the transformation from a 3DM instance with $\delta = 3$ and $Z = \{(w_1, x_2, y_3), (w_1, x_1, y_3), (w_2, x_1, y_3), (w_3, x_3, y_3), (w_3, x_1, y_1)\}$. Squares represent the primary and dummy terminals, rounded squares depict the secondary terminals, and circles symbolize the hubs. Primary and dummy origin-destination pairs are drawn vertically, and the secondary pairs are depicted horizontally. The 3DM solution is given by $M = \{(w_1, x_2, y_3), (w_2, x_3, y_2), (w_3, x_1, y_1)\}$. Each of the $2\delta$ unique paths between the secondary terminals ($\delta$ connecting $O_x$ and $D_x$, and $\delta$ connecting $O_y$ and $D_y$) is used to transfer one secondary commodity flow, and each dummy commodity is transferred via one of the $\delta$ paths from $w_d$ to $t_d$. The selected primary paths are marked by dashed arcs in Figure 2-3.

Figure 2-3. An example transformation from 3DM
To prove that the transformed MCFN RD instance is equivalent to the 3DM instance, we show that there exists a 3DM solution if and only if there exists an MCFN RD solution. First, suppose that a solution $M$ to the 3DM instance exists. A feasible routing scheme can be obtained by sending the flow of each primary commodity $i$ through the path $w_i - x_{ij}^i - y_{ik}^i - t_i$, corresponding to the triple $(w_i, x_{ij}, y_{ik}) \in M$. Each dummy commodity $i \in \{1, \ldots, \delta\}$ is routed on the path $w_d - x_{dj}^i - y_{dk}^i - t_d$. For each $j \in \{1, \ldots, \delta\}$, one secondary commodity can be routed along the path $O_x - x_{1i}^j - \cdots - x_{\delta}^j - x_d - D_x$ and another along the path $O_y - y_{1i}^j - \cdots - y_{\delta}^j - y_d - D_y$. To see that this routing scheme is feasible for the MCFN RD instance, first note that all dummy hubs have a load of two (because one secondary and one dummy commodity use each dummy node). Each regular hub has a load of at least one because of the secondary commodity flows, and for each $(w_i, x_{ij}, y_{ik}) \in M$, regular hubs $x_{ij}^i$ and $y_{ik}^i$ have a load of two because of the primary commodity flow on the path $w_i - x_{ij}^i - y_{ik}^i - t_i$. We now show that at most two hubs with a load of two exist on each path, and so the threshold reliability constraints are satisfied. The only paths that visit more than two hubs are the ones corresponding to the secondary commodities; these paths include two hubs with a load of two: Either $x_{ij}^i$ and $x_{dj}^i$, or $y_{ij}^i$ and $y_{dj}^i$, for some $1 \leq i \leq \delta$. If both $x_{ij}^i$ and $x_{dj}^i$ ($y_{ij}^i$ and $y_{dj}^i$) have a load of two for $1 \leq i < k \leq \delta$, then $x_j$ ($y_j$) appears in two different triples in $M$, which is not possible.

Now, suppose that there exists a feasible solution to the MCFN RD instance. First, recall that there are $\delta$ distinct paths between each secondary origin-destination pair. Because $\delta$ secondary commodities with unit demand have to be transferred between each pair, and at most two hubs with a load of two are allowed on each path, each commodity needs to take a separate path. Each of the $\delta$ paths $w_d - x_{dj}^i - y_{dj}^i - t_d$, $\forall i \in \{1, \ldots, \delta\}$ must be used to route the dummy commodities, or else the load of some node $x_{dj}^i$ (and $y_{dj}^i$) would be at least three (due to the secondary commodity flow). We have thus established that the number of units of secondary and dummy
commodity flow through $x_i^j$ and $y_i^j$ is one, $\forall i$ and $j$ in $\{1, \ldots, \delta\}$, and through $x_i^d$ and $y_i^d$ is two, $\forall j \in \{1, \ldots, \delta\}$.

Because each secondary commodity uses a dummy node (with a load of two), only one regular hub on the secondary path can relay a primary commodity. That is, at most one node in $\{x_1^i, \ldots, x_\delta^i\}$, and one node in $\{y_1^i, \ldots, y_\delta^i\}$, can be used in a primary commodity path. To generate a 3DM solution, we thus let $M$ consist of all $(w_i, x_j, y_k)$ triples that correspond to a primary path for commodity $i$ of the form $w_i-x_i^j-y_k^i-t_i$. As a result, each element of $w, x, \text{ and } y$ appears exactly in one triple contained in $M$, and so $M$ provides a feasible 3DM solution.

Finally, note that the transformed network has a polynomial number of nodes and arcs, and a polynomial number of commodities need to be transferred via the network. Moreover, the numerical data for the transformed problem is limited to the only two relevant values of $R$, that need to be stored (1 for a hub having a load of one, and $1 - \varepsilon$ for those having a load of two) along with $\bar{\tau} = (1 - \varepsilon)^2$. By setting, e.g., $\varepsilon = 1/2$, all data can be represented using a constant number of bits. The transformation therefore shows that MCFNRD is strongly NP-complete.

Before concluding this section, we note that MCFNRD remains strongly NP-complete specifically for the hub reliability functions given in (2–6). We can use such a function in the proof by choosing $m_i = 8\delta, \forall i \in H$, which is twice the total demand value.

In order to satisfy the conditions, the minimum probability in this case would be

$$\bar{\tau} = (1 - 2^2/(8\delta)^2)^2(1 - 1/(8\delta)^2)^{(\delta-1)},$$

because the length of the longest path in the transformed instance is $\delta + 1$, and at most two hubs on this path may carry two units of flow. We omit further details of how this choice of $\bar{\tau}$ forces MCFNRD solutions to correspond to 3DM instances for brevity.

### 2.3 Linearization of the Mathematical Model

In the mathematical model presented in the previous section, constraints (2–14) and (2–15) include bilinear terms that make the problem non-convex. Therefore, to
assist us in finding a global optimal solution, we reformulate the problem using an
equivalent mixed-integer linear program (MILP). The difficulty in linearizing this model
stems from the fact that neither $R_j$ nor $s^k_i$ in the term $R_j s^k_i$ is binary-valued, which then
prohibits the use of standard linearization techniques for quadratic programs. However,
based on the assumption that each commodity is routed on a single path, there exist
finitely many possible values for $U_i$, and by extension for $R_i$. Using this fact and the
linearization method given in [64], we provide two different approaches for obtaining
MILP formulations in Sections 2.3.1 and 2.3.2. Furthermore, the first approach that
we provide specifically assumes the reliability function given in (2–6), while the second
approach is valid for general (non-increasing) reliability functions.

2.3.1 Approach 1

Define binary variables $Q^k_i$, $\forall i \in H, k \in K$, equal to one if hub $i$ is used to transfer
commodity $k$ and zero otherwise. These variables can be defined by the following
equalities:

$$Q^k_i = \sum_{j \in FS(i)} X^k_{ij} + Y^D(k), i \in H, k \in K. \quad (2–16)$$

Also, noting that $U_i = \sum_{k \in K} d_k Q^k_i$, constraints (2–6) simplify to:

$$R_i = 1 - \frac{1}{m_i^2} \left[ \sum_{k \in K} (d_k)^2 Q^k_i + 2 \sum_{1 \leq k_1 < k_2 \leq |K|} d_{k_1} d_{k_2} Q^{k_1}{i} Q^{k_2}{i} \right]. \quad (2–17)$$

where the bracketed expression is equal to $(\sum_{k \in K} d_k Q^k_i)^2$, noting that $(Q^k_i)^2 = Q^k_i$. This
expression can now be linearized because each quadratic term is the product of two
binary variables. Define new (continuous) variables $w_{i}^{k_1,k_2} = Q^{k_1}{i} Q^{k_2}{i}, \forall i \in H, 1 \leq k_1 < k_2 \leq |K|$. Then, we substitute constraints (2–5) and (2–6) with the following constraints:

$$R_i = 1 - \frac{1}{m_i^2} \left[ \sum_{k \in K} (d_k)^2 Q^k_i + 2 \sum_{1 \leq k_1 < k_2 \leq |K|} d_{k_1} d_{k_2} w_{i}^{k_1,k_2} \right], \quad \forall i \in H \quad (2–18)$$

$$w_{i}^{k_1,k_2} \geq 0, \quad \forall i \in H, 1 \leq k_1 < k_2 \leq |K| \quad (2–19)$$
\[ w_{i_{1}i_{2}}^{k_{1}k_{2}} \geq Q_{i_{1}}^{k_{1}} + Q_{i_{2}}^{k_{2}} - 1, \quad \forall i \in H, \ 1 \leq k_{1} < k_{2} \leq |K| \]  \(2–20\)

\[ w_{i_{1}i_{2}}^{k_{1}k_{2}} \leq Q_{i_{1}}^{k_{1}}, \quad \forall i \in H, \ 1 \leq k_{1} < k_{2} \leq |K| \]  \(2–21\)

\[ w_{i_{1}i_{2}}^{k_{1}k_{2}} \leq Q_{i_{2}}^{k_{2}}, \quad \forall i \in H, \ 1 \leq k_{1} < k_{2} \leq |K|. \]  \(2–22\)

Note that \((2–18)\) is a linear function defining \(R_{i}\)-variables, although it is valid specifically because of the reliability function form \((2–6)\). Here, constraints \((2–21)\) and \((2–22)\) are not necessary, because \(w_{i_{1}i_{2}}^{k_{1}k_{2}}\) will take its smallest possible value to satisfy the threshold reliability constraints in some optimal solution. Define \(\mathcal{X} = \{ (X, Y) \) that satisfy \((2–2)\), \((2–3)\), \((2–4)\), \((2–9)\), and \((2–10)\}\), and \(Q = \{ (w, Q) \) that satisfy \((2–16)\), \((2–19)\), and \((2–20)\}\). Then, applying the above modifications, the formulation becomes

\[
\text{Min} \sum_{(i,j) \in A} \sum_{k \in K} C_{ij}^{k} d_{k} X_{ij}^{k} + \sum_{(p,i) \in A} \sum_{k \in K(p)} C_{pi}^{k} d_{k} Y_{pi} \quad (2–23)
\]

subject to:

\[
(X, Y) \in \mathcal{X} \quad (2–24)
\]

\[
(w, G) \in Q \quad (2–25)
\]

\[
\sum_{k \in K} d_{k} Q_{i}^{k} \leq m_{i}, \quad \forall i \in H \quad (2–26)
\]

\[
s_{j}^{k} \leq \left( 1 - \frac{1}{m_{j}} \left[ \sum_{k \in K} (d_{k})^{2} Q_{j}^{k} + 2 \sum_{1 \leq k_{1} < k_{2} \leq |K|} d_{k_{1}} d_{k_{2}} w_{j_{1}j_{2}}^{k_{1}k_{2}} \right] \right) s_{i}^{k} + (\alpha_{k})^{2} \left( 1 - X_{ij}^{k} - \left[ 1 - \frac{1}{\alpha_{k}} \right] Y_{O(k),j} \right), \quad \forall k \in K, \ (i,j) \in A \quad (2–27)
\]

\[
s_{i}^{k} \leq (\alpha_{k})^{2} - \left[ (\alpha_{k})^{2} - \left( 1 - \frac{1}{m_{i}} \left[ \sum_{k \in K} (d_{k})^{2} Q_{i}^{k} \right. \right. \right. \left. \left. \left. + 2 \sum_{1 \leq k_{1} < k_{2} \leq |K|} d_{k_{1}} d_{k_{2}} w_{i_{1}i_{2}}^{k_{1}k_{2}} \right] \right) \right] Y_{O(k),i}, \quad \forall k \in K, \ i \in FS(O(k)) \quad (2–28)
\]

\[
s_{i}^{k} \geq \tau_{k}, \quad \forall k \in K, \ i \in H \quad (2–29)
\]

All constraints in the foregoing model are linear except for \((2–27)\) and \((2–28)\). However, all nonlinear terms in these two constraints are the product of two bounded variables,
at most one of which is continuous. Therefore, they can be linearized using the same technique discussed before.

Linearization using this approach requires a total of \( O(|A||K|^3) \) variables and the same order of constraints, assuming that \(|A| \geq |H|\). As a result, this linearization method (potentially) increases the number of variables by a factor of \( O(|K|^2) \) over the nonlinear formulation.

2.3.2 Approach 2

We now revisit the original formulation given by (2–1)–(2–10). Because \( U_i \) can take only a finite number of values, \( \forall i \in H \), we can determine all potential values for \( U_i \) via the following two-stage approach.

- The first stage determines which commodities can send flows through hub \( i \). This stage first executes a depth-first search starting at hub \( i \), using arcs in the reverse direction, to determine which origin nodes can send flow to node \( i \). Similarly, we then execute depth-first search starting at hub \( i \), using arcs in the forward direction, to determine which destination nodes can be reached from hub \( i \). The set of commodities \( k \) whose origins can reach \( i \) and whose destinations are reachable from \( i \) is denoted by \( K' \). The complexity of this step is \( O(|A| + |\bar{A}| + |K|) \).

- In the second stage, we employ dynamic programming to determine all possible sums of the form \( \sum_{k \in K''} d_k \), for each \( K'' \subseteq K' \), which in turn yield all potential values for \( U_i \). To achieve this, index the commodities in \( K' \) as 1, ..., \( |K'| \). For \( j = 0, ..., |K'| \) and \( s = 0, ..., \min\{m, \sum_{k=1}^{|K'|} d_k\} \), define \( \theta_{js} \) as a binary variable that equals 1 if and only if there exists a subset \( \Psi \) of \( \{d_1, ..., d_j\} \) such that the sum of elements in \( \Psi \) equals \( s \). Initially, set all \( \theta \)-variables to 0, except \( \theta_{00} = 1 \). (Also, define \( \theta_{js} \equiv 0 \) if \( s < 0 \), for every \( j \).) Then, for each \( j = 1, ..., |K'| \), the algorithm considers every value \( s = 0, ..., \min\{m, \sum_{k=1}^{|K'|} d_k\} \) in order, and sets

\[
\theta_{js} = \max\{\theta_{j-1,s}, \theta_{j-1,s-d_j}\}.
\]

At the end of this process, all potential \( U_i \)-values coincide with those \( s \) for which \( \theta_{|K'|s} = 1 \). The complexity of this step is \( O(|K|\min\{m, \sum_{k=1}^{|K'|} d_k\}) \).

If \(|K| \geq |A| + |\bar{A}|\), then the second step dominates the complexity. Else, if \(|A| + |\bar{A}| > |K|\), then either step could dominate; hence, the overall algorithm complexity is \( O(|A| + |\bar{A}| + |K|\min\{m, \sum_{k \in K} d_k\}) \).
Assuming that \( G_i + 1 \) possible \( U_i \)-values are generated, the set of all possible values for \( U_i \) can be represented as \( \{ V_0, \ldots, V_{G_i} \} \). Accordingly, for each possible value of \( U_i \), the corresponding value for \( R_i \) can be calculated using equation (2–6). Observe that in this case, our approach does not depend on the quadratic form of (2–6), and hence, any valid reliability function can be used within Approach 2. Let \( \{ F_0, \ldots, F_{G_i} \} \) be the set of possible values for \( R_i \), \( \forall i \in H \). To substitute \( U_i \) and \( R_i \) in the model by discrete variables, we define new binary variables \( u_{i}^{g}, \forall i \in H, g \in \{0, \ldots, G_i\} \), which equal one if \( U_i = V_g \) (and \( R_i = F_g \)), and equal zero otherwise. We can now formulate an MILP based on our MINLP formulation in which constraints (2–5) and (2–6) are revised as:

\[
\sum_{g=0}^{G_i} V_g u_{i}^{g} = \sum_{k \in K} d_k \left( \sum_{j \in FS(i)} X_{ij}^k + Y_{D(k),i} \right), \quad \forall i \in H \tag{2–30}
\]

\[
R_i = \sum_{g=0}^{G_i} F_g u_{i}^{g}, \quad \forall i \in H \tag{2–31}
\]

\[
\sum_{g=0}^{G_i} u_{i}^{g} = 1, \quad \forall i \in H \tag{2–32}
\]

\[
u_{i}^{g} \in \{0, 1\}, \quad \forall i \in H, g \in \{0, \ldots, G_i\}. \tag{2–33}
\]

The constraints of this revised formulation include \((X, Y) \in \mathcal{X}^{*}\), constraints (2–13), (2–14), (2–15), and (2–30)–(2–33).

With the above modifications, both \( U_i \) and \( R_i \) can be substituted by linear functions of discrete variables, and so all terms in (2–14) and (2–15) are now either linear or quadratic. Also, because each quadratic term is the product of two bounded variables, at most one of which is continuous, they can be linearized by defining a new variable for each product and using the same method used in Section 2.3.1.

To linearize our MINLP using this approach, we need a total of \( O(|\tilde{A}|G + |H|(|K| + G) + |A||K|G) \) variables and \( O(|\tilde{A}|G + |H||K| + |A||K|G) \) constraints, where \( G = \max\{G_i, \forall i \in H\} \). Because \( G \) is exponentially large in general, this formulation may
be too large to be solved within practical computational limits. We thus explore an alternative solution methodology based on this formulation in the next two sections.

2.4 Lower- and Upper-Bounding Scheme

Since MCFNR is NP-hard, and our exact formulations tend to be intractable due to their size, we instead investigate lower- and upper-bounding schemes for MCFNR by solving polynomial-size MILP formulations.

We first discuss our lower-bounding model, which is similar to Approach 2 presented in Section 2.3.2. However, here we only consider a subset of possible $U_i$-values (and corresponding $R_i$-values). Our lower-bounding model then sets $U_i$ to equal the largest value in this subset that does not exceed the load of node $i$ (i.e., $U_i$ is “rounded down” to the nearest value in the subset of possible $U_i$-values). The value for $R_i$ then corresponds to the estimated $U_i$-value, and is thus possibly an overestimation of the true reliability of hub $i$. Using this strategy yields a lower bound on the optimal objective function value, because if the reliability constraints (constraints (2–13), (2–14), and (2–15)) are satisfied by the hub reliability values, then they will be satisfied using the overestimated values for $R_i$ as well. As a result, the lower-bounding problem (in which we overestimate $R_i$-values) is a relaxation of the original problem.

To implement this idea, we pick $\beta + 1$ possible values for $U_i$ from the set $\{V_0, \ldots, V_{G_i}\}$, and index these values as $\{V_{i0}, \ldots, V_{i\beta}\}$, where $V_{ij} < V_{i(j+1)}$, $\forall j = 0, \ldots, \beta - 1$. Note that the smallest possible $V_i$-value is 0 and the largest is $\sum_{k \in K} d_k$, because each commodity visits every hub at most once. Hence, we select $V_{i0} = 0$ and $V_{i\beta} = \sum_{k \in K} d_k$, which guarantees that the actual load of node $i$ is contained in the interval $[V_{i0}, V_{i\beta}]$. We then calculate the $R_i$-value corresponding to $V_{i\beta}$, for $i = 0, \ldots, \beta$, using equation (2–6) and maintain the previous index order to form the set $\{F_{i0}, \ldots, F_{i\beta}\}$. Obviously, $F_{i0} = 1$ and $F_{ig} \geq F_{ig+1}$, for $g = 0, \ldots, \beta - 1$. Now, define
new binary variables $\tilde{u}_i^g$ as follows:

$$
\tilde{u}_i^g = \begin{cases} 
1, & \text{if } U_i \geq V_{(g)} \\
0, & \text{otherwise}
\end{cases} \quad \forall i \in H, \ g \in \{1, \ldots, \beta\}.
$$

We then need to add the following constraints to the problem:

$$
\tilde{u}_i^g \geq \frac{U_i - V_{(g)} + 1}{\sum_{k \in K} d_k - V_{(g)} + 1}, \quad \forall i \in H, \ g \in \{1, \ldots, \beta\}. \tag{2–34}
$$

The estimated value for $R_i$ in the lower-bounding problem is given as:

$$
R_i^L = 1 + \sum_{g=1}^{\beta} \tilde{u}_i^g (F_{(g)} - F_{(g-1)}), \quad \forall i \in H. \tag{2–35}
$$

Similarly, we can obtain a reduced-size model in which we underestimate the $R_i$-values (and overestimate the $U_i$-values), which then yields an upper bound for the problem. In this case, we modify the definition of $\tilde{u}_i^g$ so that it equals one if and only if $U_i > V_{(g)}$ (as a strict inequality), for each $i \in H$ and $g = 0, \ldots, \beta - 1$. We enforce this relationship by the constraints:

$$
\tilde{u}_i^g \geq \frac{U_i - V_{(g)}}{\sum_{k \in K} d_k - V_{(g)}}, \quad \forall i \in H, \ g \in \{0, \ldots, \beta - 1\}. \tag{2–36}
$$

and constrain the estimated $R_i$-values in the upper-bounding problem as:

$$
R_i^U = 1 + \sum_{g=0}^{\beta-1} \tilde{u}_i^g (F_{(g+1)} - F_{(g)}). \tag{2–37}
$$

We can now formulate the lower-bounding and upper-bounding problems by revising model (2–1)–(2–10), in which (2–8) is replaced with (2–13), (2–14), and (2–15), as follows. First, substitute variable $R_i$ in model (2–1)–(2–10) by $R_i^L$ (or $R_i^U$ for the upper-bounding model) and replace constraints (2–6) by the expression defining $R_i^L$ in (2–35) (or $R_i^U$ in (2–37)). Next, add constraints (2–34) (or (2–36)) to the model along with binariness restrictions on the $\tilde{u}$-variables. The same methods discussed in Section 2.3 can then be implemented to linearize the lower- and upper-bounding models.
2.5 Cutting-Plane Algorithm

In this section, we present a method for obtaining an optimal solution for MCFNR by appending cutting planes to the lower-bounding formulation in Section 2.4. First, we start with subsets \( \{V_{\{0\}}, \ldots, V_{\{g\}}\} \) and \( \{F_{\{0\}}, \ldots, F_{\{g\}}\} \), and formulate the lower-bounding problem described in Section 2.4. We then execute branch-and-bound on the lower-bounding problem until an integer-feasible solution, \((\hat{X}, \hat{Y})\), is found. Given the multicommodity flow routes prescribed by \((\hat{X}, \hat{Y})\), we calculate the actual \(U_i\)- and \(R_i\)-values using equations (2–5) and (2–6), respectively, and compute the reliability of each path.

Because the MINLP feasible region is a subset of the lower-bounding problem’s feasible region, the actual reliability of a path for some commodity \(k \in K\) might be less than \(\tau_k\) in the solution given by \((\hat{X}, \hat{Y})\). If so, this commodity is said to be a “violated” commodity. Define \(\tilde{K} \subseteq K\) as the set of all violated commodities. If no violated commodities exist (\(\tilde{K} = \emptyset\)), then the actual reliability values given by the solution \((\hat{X}, \hat{Y})\) are all at least \(\tau_k\), \(\forall k \in K\), and \((\hat{X}, \hat{Y})\) is feasible to MCFNR. Otherwise, for each lower-bounding solution having violated commodities, we cut off the current solution via a cutting plane, as described below.

To generate a cutting plane, consider a set \(\tilde{K}\) of violated commodities. For each violated commodity path, we need to either revise the path, or reduce the load of a hub visited by the path. For \(k \in \tilde{K}\), let \(P_k\) be the set of hubs that lie on commodity \(k\)’s path. Moreover, define \(I_k\) as the set of all incoming arcs to the hubs in \(P_k\) whose corresponding \(X\)-variables equal 1 in the current solution. Similarly, let \(\tilde{I}_k\) be the set of all terminal-hub pairs assigned in the current lower-bounding solution, such that the hub belongs to \(P_k\). Formally, given a current solution with flow and assignment values \((\hat{X}, \hat{Y})\), and violated commodity \(k \in \tilde{K}\):

\[
I_k = \{(i, j, k') \mid (i, j) \in A, j \in P_k, k' \in K, \hat{X}_{i,j}^{k'} = 1\},
\]

\[
\tilde{I}_k = \{(p, i) \in \tilde{A} \mid i \in P_k, \hat{Y}_{pi} = 1\}.
\]
One necessary condition for feasibility is to require at least one $X_{ij}^{k'}$-variable to equal zero, for $(i, j, k') \in I_k$, or at least one $Y_{pi}$-variable to equal zero, for $(p, i) \in I_k$. This condition is enforced by the following inequality:

$$\sum_{(i,j,k') \in I_k} (1 - X_{ij}^{k'}) + \sum_{(p,i) \in I_k} (1 - Y_{pi}) \geq 1. \tag{2–38}$$

Inequality (2–38) is valid, because if the left-hand-side is zero, then commodity $k$ continues to use the same path as before, and the load on every hub in $P_k$ is at least as large as it was in the previous solution (which led to commodity $k$ being a violating commodity). Thus, any solution in which the left-hand-side of (2–38) equals zero must be infeasible. Moreover, (2–38) is a cutting plane because its left-hand-side evaluates to zero in the current infeasible solution ($\bar{X}, \bar{Y}$).

We now show that the cutting-plane algorithm converges finitely. Note that each feasible solution to the lower-bounding problem corresponds to a distinct set of $X$- and $Y$-variables. As a result, the number of different solutions cannot be more than $2^{\vert A \vert \vert K \vert + \vert \bar{A} \vert}$. Moreover, cutting-plane algorithm visits each integer solution at most once in the branch-and-bound tree, and so the given algorithm converges finitely.

**Remark 2.3.** There are various implementation options for the cutting-plane algorithm based on the number of cuts we add at each node of the branch-and-bound tree for which inequalities of the form (2–38) are generated. Here, we implement three different strategies to cut off a solution having violated commodities $\bar{K}$.

**Strategy 1.** Generate $\vert \bar{K} \vert$ cutting-plane inequalities of the form (2–38), one for each violated commodity.

**Strategy 2.** Generate a single cut by aggregating cutting-plane constraints (2–38) for all violated commodities.

**Strategy 3.** Define a “most violated commodity” ($k_m$) as a violated commodity for which the difference between the reliability threshold value and the actual probability of
successful delivery is the largest among all violated commodities. Generate a single cut (2–38) corresponding only to commodity \( k_m \).

Note that the last two strategies generate fewer cutting planes than the first strategy. However, strategy 1 tends to cut off more infeasible solutions than the other two strategies, thus reducing the size of the branch-and-bound tree explored by the algorithm.

2.6 Computational Experiments

In this section we evaluate the computational efficiency of our cutting-plane algorithms for solving MCFNR by testing them on 40 randomly generated instances. We first present our test problems and describe our random generation routine in Section 2.6.1. Then, in Section 2.6.2, we implement all three cutting-plane strategies using CPLEX 12.2 via ILOG Concert Technology, along with a brief comparison to a premier quadratic optimization solver. All computations were performed on an Intel Core i5 with a 2.40 GHz processor and 4.0 GB RAM.

2.6.1 Test Problem Generation

We first describe the scheme that we use to randomly generate MCFNR instances. Because MCFNR is a new problem in the literature, we have also made our test instances available at \http://www.ise.ufl.edu/cole\). The instances are classified into eight categories based on the combination of parameter values \( |T| \in \{5, 10\} \), \( |H| \in \{10, 20\} \), and \( |K| \in \{5, 10\} \), with one exception. When \( |T| = 10 \) and \( |K| = 5 \), all communications would be unidirectional, i.e., each terminal serves as either an origin or a destination for exactly one commodity. To avoid this situation (which would seem to be rare in practice), we modify \( |T| \in \{5, 7\} \) when \( |K| = 5 \).

For each combination of \( |T|, |H|, \) and \( |K| \), we randomly generate five instances. To form the set \( A \), for each pair of hubs \( i, j \in H, i \neq j \), we generate arc \((i, j)\) with probability 0.5. The set \( \tilde{A} \) is generated by applying the same method for each undirected arc \((p, i)\) such that \( p \in T \) and \( i \in H \). We then generate an arc from each isolated hub (a hub
with no incoming or outgoing arcs) to a random hub. An integer random number with
discrete uniform distribution between 1 and 5 is generated as the cost of each arc in
$A$. The same method is used to generate the cost of arcs in $\tilde{A}$, except that the integer
values are generated uniformly between 1 and 15.

For each commodity, the origin terminal is randomly selected from the set $T$. The
destination terminal is then generated via the same method, except that the process is
repeated until the commodity’s destination is different from its origin, and the generated
origin-destination pair is not the same as any other commodity’s origin-destination pair.
Next, $\gamma_i$-values (for all $i \in H$) are generated randomly between 1 and $|T|$. We then
ensure that every terminal is an origin or destination for at least one commodity, and that
all terminals can be assigned to the hubs, i.e., $\sum_{i \in H} \gamma_i \geq |T|$. The demand value for
each commodity is produced by generating a random number with uniform distribution
between 1 and 5.

We set $m_i = 2 \sum_{k \in K} d_k$, $\forall i \in H$, which thus obviates the need for constraints limiting
$U_i \leq m_i$, for each $i \in H$. We then generate $\tau_k$-values by first defining a new parameter $\chi$
as the largest terminal load, i.e., $\chi = \max_{p \in T} \sum_{k \in k(p)} d_k$. Then we define $\tau_k$ as follows:

$$
\tau_k = \left(1 - \frac{\chi^2}{m^2}\right)^2, \quad \forall k \in K, \tag{2–39}
$$

where $m$ is the common value assigned to all $m_i$-values in the instance. In this manner,
all commodity reliability thresholds equal a common value, $\tau$. Setting $\tau$ as such tends
to generate instances that are challenging to solve, and do not generally admit optimal
solutions in which all commodity pairs simply use a shortest path connecting their
origins and destinations. More specifically, our choice of $\tau$ guarantees the feasibility of
solutions for which each terminal is assigned to a separate hub, and each commodity
path passes through at most two hubs (each of which would therefore have a load of
at most $\chi$). Several of our instances do not satisfy this criteria, and so these instances
may or may not be feasible. Each infeasible instance generated in this manner was discarded, so that all instances in our test set have optimal solutions.

To generate the set of possible values for $U_i$, and by extension for $R_i$, recall that we require $V_0 = 0$ ($F_0 = 1$) and $V_\beta = \sum_{k \in K} d_k$ ($F_\beta = 0.75$, noting that each $m_i = 2 \sum_{k \in K} d_k$). Our strategy for selecting the remaining $V$- and $F$-values is inspired from the shape of the reliability function defined in (2–6), which is concave and decreasing. First, we set $\beta = |K|$. To obtain a tighter relaxation of this function, we may wish to select more $V$-values corresponding to higher hub loads. On the other hand, because we are unlikely to relay a large portion of the total commodities’ loads through a single hub, selecting several large values for $U_i$ is not reasonable. Therefore, we tend to distribute the values more densely around the center of the reliability function’s range. To accomplish this goal, we first sort the commodities in nondecreasing order of their demand values, and obtain the ordered set $\{d_{(1)}, \ldots, d_{(|K|)}\}$. Then, we define

$$V_{i+1} = \begin{cases} 
V_i + d_{\lfloor |K|/2 \rfloor - i + 1}, & \forall i \in \{0, \ldots, \lfloor |K|/2 \rfloor \} \\
V_i + d_{i+1}, & \forall i \in \{\lfloor |K|/2 \rfloor + 1, \ldots, |K| - 1\}.
\end{cases}$$

Note that the number of selected values in this method is $|K| + 1$.

### 2.6.2 Results

We present computational results regarding the comparison of cut-generation strategies in Table 2-1, where all times are reported in CPU seconds. We allow a one-hour (3600 seconds) time limit, and for the instances that exceed this time limit, we report the relative optimality gap that was achieved.

In Table 2-1, S1, S2, and S3 refer to column-generation strategies 1, 2, and 3, respectively. The column labeled “Instance” describes the instances as “XYZ-N” in which “X”, “Y”, and “Z” denote the level (“L” for large, and “S” for small) of parameters $|T|$, $|H|$, and $|K|$, respectively, and “N” is the instance number within the category specified by “XYZ”. In the second column (“Time (Gap%)”), we report the solution time or relative
optimality gap for each of the strategies, depending on whether the instance has been solved to optimality within the time limit or not, respectively. For each strategy, the number of cuts generated and number of nodes in the branch-and-bound tree for each strategy are presented in columns “Cuts” and “Nodes”, respectively.

Table 2-1 shows that the number of instances that can be solved within one hour using strategies 1, 2, and 3 are 38, 36, and 35, respectively. Also, the average solution time for the instances that have been solved using all three strategies (35 instances) are 167, 184, and 226 using strategies 1, 2, and 3, respectively. Figure 2-4 depicts the number of solved instances by increasing the time limit, using each strategy. The horizontal axis in this figure represents the limit we allow for the solution time and the graph represents the number of instances that have been solved to optimality within that time limit, using the corresponding strategy. Based on this graph, strategy 1 appears to be the most effective strategy overall, although the differences in the performances of these strategies appear to be small. As a result, we use strategy 1 to generate cuts in our remaining experiments.

Recall that because we selected a quadratic capacity function in these experiments, we could alternatively solve these MCNF instances via a mixed-integer quadratic optimizer. Accordingly, we modeled the MCNF in GAMS 24.1.1 and solved the MINLP formulation directly using the GloMIQO 2 solver. We adjusted the relative optimality gap parameter (optcr) to 0.001 in order to obtain optimal solutions for these instances. Table 2-2 presents the results of using this solver (in the column marked “GloMIQO”) to solve the foregoing instances as before (again with the one hour time limit), and compares them with the results obtained from our cut-generation strategy 1 (“S1”). The column (“Time (Gap%)”) gives the execution times for instances solved to optimality by the methods, or the relative optimality gaps for instances that reached the time limit. The third (“LB”) and fourth (“UB”) columns respectively depict the lower and upper bound values obtained using each method. The cells corresponding to the instances in which
<table>
<thead>
<tr>
<th>Instance</th>
<th>Time (Gap%)</th>
<th>Cuts</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
</tr>
<tr>
<td>LLL-1</td>
<td>139</td>
<td>149</td>
<td>137</td>
</tr>
<tr>
<td>LLL-2</td>
<td>13</td>
<td>28</td>
<td>12</td>
</tr>
<tr>
<td>LLL-3</td>
<td>2918</td>
<td>(5.42%)</td>
<td>(1.48%)</td>
</tr>
<tr>
<td>LLL-4</td>
<td>80</td>
<td>161</td>
<td>84</td>
</tr>
<tr>
<td>LLL-5</td>
<td>9</td>
<td>21</td>
<td>9</td>
</tr>
<tr>
<td>LLS-1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>LLS-2</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>LLS-3</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>LLS-4</td>
<td>4</td>
<td>41</td>
<td>10</td>
</tr>
<tr>
<td>LLS-5</td>
<td>5</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>LSL-1</td>
<td>113</td>
<td>150</td>
<td>90</td>
</tr>
<tr>
<td>LSL-2</td>
<td>1801</td>
<td>1274</td>
<td>2723</td>
</tr>
<tr>
<td>LSL-3</td>
<td>1615</td>
<td>1660</td>
<td>(0.84%)</td>
</tr>
<tr>
<td>LSL-4</td>
<td>3048</td>
<td>3174</td>
<td>3384</td>
</tr>
<tr>
<td>LSL-5</td>
<td>413</td>
<td>405</td>
<td>389</td>
</tr>
<tr>
<td>LSS-1</td>
<td>21</td>
<td>79</td>
<td>164</td>
</tr>
<tr>
<td>LSS-2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LSS-3</td>
<td>31</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>LSS-4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LSS-5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SSL-1</td>
<td>18</td>
<td>92</td>
<td>19</td>
</tr>
<tr>
<td>SSL-2</td>
<td>(10.69%)</td>
<td>(11.54%)</td>
<td>(11.38%)</td>
</tr>
<tr>
<td>SSL-3</td>
<td>11</td>
<td>28</td>
<td>11</td>
</tr>
<tr>
<td>SSL-4</td>
<td>16</td>
<td>33</td>
<td>16</td>
</tr>
<tr>
<td>SSL-5</td>
<td>82</td>
<td>165</td>
<td>90</td>
</tr>
<tr>
<td>SLS-1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SLS-2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>SLS-3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>SLS-4</td>
<td>(7.14%)</td>
<td>(2.27%)</td>
<td>(17.11%)</td>
</tr>
<tr>
<td>SLS-5</td>
<td>881</td>
<td>(37.58%)</td>
<td>(3.62%)</td>
</tr>
<tr>
<td>SSL-1</td>
<td>845</td>
<td>456</td>
<td>599</td>
</tr>
<tr>
<td>SSL-2</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>SSL-3</td>
<td>47</td>
<td>64</td>
<td>90</td>
</tr>
<tr>
<td>SSL-4</td>
<td>8</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>SSL-5</td>
<td>17</td>
<td>19</td>
<td>15</td>
</tr>
<tr>
<td>SSS-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SSS-2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SSS-3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SSS-4</td>
<td>11</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>SSS-5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
GloMIQO fails to generate a feasible solution within the time limit are marked by “–”. We conclude that GloMIQO appears to be the more efficient solver on instances that are relatively easy to solve. Its potential advantages are underscored most dramatically in the LSL instances, except for LSL-5. However, for the most difficult instances, S1 is preferable. Note that GloMIQO failed to solve six instances to optimality, while S1 failed to solve two instances. Furthermore, GloMIQO fails to even find a feasible solution to three instances within the time limit. Again, it is worth noting that S1 can be applied for general capacity functions, and does not rely on quadratic structures.

We next study and compare two alternative methods in selecting \(V\)- and \(F\)-values to initialize our cutting-plane algorithm. The original method (in which \(U_i\)-values are chosen near the center of the reliability function range) is termed method 1. The second and third methods divide the interval \([0, \sum_{k \in K} d_k]\) into several subintervals of length...
<table>
<thead>
<tr>
<th>Instance</th>
<th>Time (Gap%)</th>
<th>LB</th>
<th>UB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLL-1</td>
<td>–</td>
<td>139</td>
<td>–</td>
</tr>
<tr>
<td>LLL-2</td>
<td>6</td>
<td>13</td>
<td>283</td>
</tr>
<tr>
<td>LLL-3</td>
<td>–</td>
<td>2918</td>
<td>–</td>
</tr>
<tr>
<td>LLL-4</td>
<td>(7.42%)</td>
<td>80</td>
<td>287</td>
</tr>
<tr>
<td>LLL-5</td>
<td>7</td>
<td>9</td>
<td>339</td>
</tr>
<tr>
<td>LLS-1</td>
<td>2</td>
<td>1</td>
<td>118</td>
</tr>
<tr>
<td>LLS-2</td>
<td>3</td>
<td>4</td>
<td>121</td>
</tr>
<tr>
<td>LLS-3</td>
<td>3</td>
<td>4</td>
<td>92</td>
</tr>
<tr>
<td>LLS-4</td>
<td>3</td>
<td>4</td>
<td>63</td>
</tr>
<tr>
<td>LLS-5</td>
<td>2</td>
<td>5</td>
<td>95</td>
</tr>
<tr>
<td>LSL-1</td>
<td>19</td>
<td>113</td>
<td>366</td>
</tr>
<tr>
<td>LSL-2</td>
<td>113</td>
<td>1801</td>
<td>269</td>
</tr>
<tr>
<td>LSL-3</td>
<td>149</td>
<td>1615</td>
<td>396</td>
</tr>
<tr>
<td>LSL-4</td>
<td>83</td>
<td>3048</td>
<td>374</td>
</tr>
<tr>
<td>LSL-5</td>
<td>2188</td>
<td>413</td>
<td>344</td>
</tr>
<tr>
<td>LSS-1</td>
<td>14</td>
<td>21</td>
<td>204</td>
</tr>
<tr>
<td>LSS-2</td>
<td>0</td>
<td>1</td>
<td>133</td>
</tr>
<tr>
<td>LSS-3</td>
<td>11</td>
<td>31</td>
<td>136</td>
</tr>
<tr>
<td>LSS-4</td>
<td>0</td>
<td>0</td>
<td>101</td>
</tr>
<tr>
<td>LSS-5</td>
<td>0</td>
<td>1</td>
<td>136</td>
</tr>
<tr>
<td>SLL-1</td>
<td>7</td>
<td>18</td>
<td>300</td>
</tr>
<tr>
<td>SLL-2</td>
<td>(2.42%)</td>
<td>(10.69%)</td>
<td>415.68</td>
</tr>
<tr>
<td>SLL-3</td>
<td>6</td>
<td>11</td>
<td>382</td>
</tr>
<tr>
<td>SLL-4</td>
<td>6</td>
<td>16</td>
<td>416</td>
</tr>
<tr>
<td>SLL-5</td>
<td>21</td>
<td>82</td>
<td>377</td>
</tr>
<tr>
<td>SLS-1</td>
<td>2</td>
<td>2</td>
<td>154</td>
</tr>
<tr>
<td>SLS-2</td>
<td>2</td>
<td>2</td>
<td>234</td>
</tr>
<tr>
<td>SLS-3</td>
<td>2</td>
<td>1</td>
<td>123</td>
</tr>
<tr>
<td>SLS-4</td>
<td>(7.14%)</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SLS-5</td>
<td>777</td>
<td>881</td>
<td>171</td>
</tr>
<tr>
<td>SSL-1</td>
<td>978</td>
<td>845</td>
<td>592</td>
</tr>
<tr>
<td>SSL-2</td>
<td>1</td>
<td>11</td>
<td>360</td>
</tr>
<tr>
<td>SSL-3</td>
<td>(1.65%)</td>
<td>47</td>
<td>418</td>
</tr>
<tr>
<td>SSL-4</td>
<td>11</td>
<td>8</td>
<td>395</td>
</tr>
<tr>
<td>SSL-5</td>
<td>2</td>
<td>17</td>
<td>460</td>
</tr>
<tr>
<td>SSS-1</td>
<td>0</td>
<td>0</td>
<td>191</td>
</tr>
<tr>
<td>SSS-2</td>
<td>0</td>
<td>0</td>
<td>230</td>
</tr>
<tr>
<td>SSS-3</td>
<td>1</td>
<td>1</td>
<td>176</td>
</tr>
<tr>
<td>SSS-4</td>
<td>8</td>
<td>11</td>
<td>273</td>
</tr>
<tr>
<td>SSS-5</td>
<td>1</td>
<td>1</td>
<td>123</td>
</tr>
</tbody>
</table>
\( d_{\text{med}} \) (median of demand values) and \( d_{\text{min}} \) (minimum of demand values), respectively. Method 2 thus sets \( \beta = \left\lceil \sum_{k \in K} d_k / d_{\text{med}} \right\rceil \). Note that the last subinterval has length \( \sum_{k \in K} d_k - d_{\text{med}}(\beta - 1) \leq d_{\text{med}} \), with strict inequality holding if \( \sum_{k \in K} d_k \) is not divisible by \( d_{\text{med}} \). These observations apply to method 3 as well, with \( d_{\text{med}} \) replaced by \( d_{\text{min}} \).

To test the three methods presented above, we randomly generated 20 instances for each of the eight categories discussed in Section 2.6.1. However, the demand values for commodities are now generated randomly between 1 and 100, in order to observe the differences between the methods. Table 2-3 presents the number of instances solved within one hour limit (“# solved”), and the average solution time (“Average Time”) for instances generated by each method (“M1”, “M2”, “M3”). Note that the number of selected values for \( U_i \) impacts the number of variables in the lower-bounding model, and tends to create large formulations for the third method in particular. The computational time limit remains one hour for these instances.

Table 2-3. Comparison of the three \( U_i \)- and \( R_i \)-value selection strategies

<table>
<thead>
<tr>
<th>Category</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLL</td>
<td>18</td>
<td>18</td>
<td>12</td>
<td>709</td>
<td>600</td>
<td>1843</td>
</tr>
<tr>
<td>LLS</td>
<td>19</td>
<td>20</td>
<td>20</td>
<td>5</td>
<td>8</td>
<td>72</td>
</tr>
<tr>
<td>LSL</td>
<td>17</td>
<td>19</td>
<td>15</td>
<td>752</td>
<td>499</td>
<td>1537</td>
</tr>
<tr>
<td>LSS</td>
<td>20</td>
<td>19</td>
<td>19</td>
<td>5</td>
<td>22</td>
<td>212</td>
</tr>
<tr>
<td>SLL</td>
<td>14</td>
<td>15</td>
<td>10</td>
<td>1149</td>
<td>1032</td>
<td>1779</td>
</tr>
<tr>
<td>SLS</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>543</td>
<td>543</td>
<td>610</td>
</tr>
<tr>
<td>SSL</td>
<td>16</td>
<td>16</td>
<td>10</td>
<td>1148</td>
<td>835</td>
<td>2043</td>
</tr>
<tr>
<td>SSS</td>
<td>20</td>
<td>20</td>
<td>19</td>
<td>4</td>
<td>7</td>
<td>210</td>
</tr>
</tbody>
</table>

Based on the data presented in Table 2-3, many of the instances generated by the third method cannot be solved due to memory or time limits. Those instances that were solved within the time limit by the third method require considerably more time to solve than the other two methods. The performance of the algorithm on the instances generated by methods 1 and 2 depends on the size of the instances and in particular, the size of parameter \(|K|\). For smaller values of \(|K|\), the first method appears to be
favorable. However, for the more challenging instances in which $|K|$ takes its upper bound value, the second method is preferable.

Next, we examine the strength of the lower and upper bounds for MCFNR, as presented in Section 2.4. We solve the lower- and upper-bounding models for the instances in Table 2-1 and compare the bounds with their optimal objective values in Table 2-4. We allow a one-hour time limit and present the calculated bound for the problem ("LB" for Lower Bound and "UB" for Upper Bound), the total solution time ("Time") in seconds, and the gap between the bound and the optimal objective function value ("Gap%") in Table 2-4. The last column of Table 2-4 lists the optimal objective function value of the instances that have been solved to optimality within the one-hour time limit using our cutting-plane algorithm. For instances in which we were unable to compute an upper bound within one hour, we report “–” in the table. The cells marked with “*” correspond to the instances for which the cutting-plane algorithm fails to terminate within one hour. Additionally, the upper-bounding model is infeasible for instance LLS-4, and is marked with “INF”.

According to Table 2-4 the quality of lower and upper bounds for the instances whose corresponding upper-bounding model have been solved within an hour are comparable. However, the upper-bounding model takes longer to terminate than the lower-bounding model, because finding feasible solutions for the upper-bounding model is more difficult than for the lower-bounding model.
<table>
<thead>
<tr>
<th>Instance</th>
<th>Lower-bounding model</th>
<th>Upper-bounding model</th>
<th>Cutting plane</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LB Time Gap% UB Time Gap% Optimal Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLL-1</td>
<td>309 22 0.96 312 47 0.00 312</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLL-2</td>
<td>283 13 0.00 283 11 0.00 283</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLL-3</td>
<td>327 200 2.97 340 2203 0.89 337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLL-4</td>
<td>287 18 2.05 293 45 0.00 293</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLL-5</td>
<td>339 14 0.00 339 8 0.00 339</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-1</td>
<td>118 2 0.00 118 2 0.00 118</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-2</td>
<td>121 5 0.00 121 3 0.00 121</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-3</td>
<td>84 2 8.70 – 3600 – 92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-4</td>
<td>61 2 3.17 INF 13 – 63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-5</td>
<td>90 2 5.26 106 6 11.58 95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-1</td>
<td>366 98 0.00 366 135 0.00 366</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-2</td>
<td>264 720 1.86 269 1385 0.00 269</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-3</td>
<td>388 63 2.02 – 3600 – 396</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-4</td>
<td>374 111 0.00 – 3600 – 374</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-5</td>
<td>343 59 0.29 360 379 4.65 344</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-1</td>
<td>165 0 19.12 228 25 11.76 204</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-2</td>
<td>133 1 0.00 – 3600 – 133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-3</td>
<td>128 6 5.88 136 1 0.00 136</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-4</td>
<td>101 1 0.00 118 2 16.83 101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLS-5</td>
<td>136 1 0.00 136 1 0.00 136</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-1</td>
<td>300 10 0.00 – 3600 – 300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-2</td>
<td>381 33 * 426 554 *</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-3</td>
<td>382 14 0.00 382 8 0.00 382</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-4</td>
<td>416 18 0.00 416 20 0.00 416</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-5</td>
<td>373 12 1.06 – 3600 – 377</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-1</td>
<td>154 2 0.00 – 3600 – 154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-2</td>
<td>234 3 0.00 – 3600 – 234</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-3</td>
<td>123 2 0.00 123 2 0.00 123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-4</td>
<td>170 1 * – 3600 –</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-5</td>
<td>158 3 7.60 – 3600 – 171</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-1</td>
<td>586 56 1.01 607 747 2.53 592</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-2</td>
<td>360 4 0.00 – 3600 – 360</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-3</td>
<td>418 11 0.48 420 17 0.00 420</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-4</td>
<td>390 5 1.27 416 19 5.32 395</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL-5</td>
<td>460 6 0.00 464 4 0.87 460</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSS-1</td>
<td>191 1 0.00 – 3600 – 191</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSS-2</td>
<td>230 1 0.00 – 3600 – 230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSS-3</td>
<td>176 1 0.00 185 2 5.11 176</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSS-4</td>
<td>269 2 1.47 309 125 13.19 273</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSS-5</td>
<td>123 1 0.00 146 2 18.70 123</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.1 Motivation

The class of single-machine scheduling problems we examine in this survey chapter seek to schedule nonpreemptive jobs on a single machine, which is capable of performing only one task at a time. Defining $J$ as the set of all jobs, each job $j \in J$ is typically associated with data attributes such as processing time ($p_j$), weight ($w_j$), and due date ($d_j$). The objective of a schedule depends on the completion time ($C_j$) of each job $j \in J$. Define $L_j$ (lateness) as $C_j - d_j$, $T_j$ (tardiness) as $\max \{0, C_j - d_j\}$, and $U_j$ as a binary indicator value equal to 1 if and only if $C_j - d_j > 0$. Common single-machine scheduling objectives include minimizing total completion time ($\sum_{j \in J} C_j$), total weighted completion time ($\sum_{j \in J} w_j C_j$), and the number of late jobs ($\sum_{j \in J} U_j$). Also, defining $L_{\max} = \max_{j \in J} \{L_j\}$ (and $T_{\max}$ and $C_{\max}$ analogously), other common objectives include minimizing $L_{\max}$, $T_{\max}$, and $C_{\max}$. (The latter objective is also referred to as makespan.)

Deterministic scheduling problems assume that the exact values of all parameters are known. As a result, it is straightforward to calculate the job completion times and objective function value corresponding to each sequence. Moreover, each problem listed above is polynomially solvable using deterministic data [50, 66, 73].

Due to variability of process and environmental data, uncertainty is common in many practical scheduling problems. Therefore, several researchers have developed methods to hedge against data uncertainty in this area. Reactive (online) scheduling deals with adjusting job schedules as data is realized in order to reduce the effect of disruptions and unpredicted delays. Using this approach, the scheduler generates initial schedules without considering uncertainty, and then revises the schedule as disruptions occur. Online scheduling is specifically useful for situations in which limited information about uncertain data is available in advance, and it is practical for the scheduler to dynamically adjust the schedules. This reactive approach can accommodate a wide
variety of disruptions (such as machine breakdowns, process interruptions, or parameter value variations) and is suitable for problems in which disruptions are difficult to predict. For example, process scheduling in an operating system can be modeled as an online scheduling problem since the operating system typically does not know the execution time of a process before its completion, and it is possible for the operating system to interleave the execution of different tasks by temporarily interrupting a task and resuming its execution at a later time (preemption). Another example regards the scheduler component of a web server. The number of users and the length of each user’s tasks is unknown and very difficult to predict. Therefore, online scheduling is an appropriate method for modeling the problem of scheduling users’ web transactions. See [70] for an overview of online scheduling and a survey of results obtained in this area, and [61] for an example of online single-machine scheduling.

However, it is not always realistic to assume that schedules can be revised after disruptions. In these situations, a predictive (offline) scheduling approach prescribes solutions that are relatively insensitive to changes in input data. An implicit assumption made in offline scheduling is that all uncertain data values are realized after the decisions have been made. Stochastic programming and robust optimization are two important offline modeling schemes that have been applied for solving several scheduling problems under uncertainty. Stochastic optimization, which was first introduced by Dantzig [26], assumes knowledge of data probability distributions. This method seeks to optimize the expected objective function value, while ensuring that constraints are satisfied at least with sufficiently high probability. We refer the interested reader to textbooks [18, 36, 43] and the references therein. Pinedo and Schrage [69] present a survey of stochastic optimization applications in solving scheduling problems.

Robust optimization is an alternative approach for dealing with uncertain data [12, 72]. In robust optimization, data uncertainty is usually represented by continuous or discrete uncertainty sets and the feasibility of the solution is guaranteed with
respect to any possible data outcome within the uncertainty set. Two main factors motivate the use of robust optimization. First, it is not always possible to estimate data probability distributions with desired precision. Second, in stochastic programming, the problem size increases drastically with the number of uncertain parameters, which induces substantial computational challenges. Another essential difference between the stochastic programming and robust optimization approaches is that stochastic programming models typically aim at optimizing expected system performance, while robust optimization models focus on guaranteeing a minimum quality for the solution in the worst case.

In the deterministic scheduling literature, problems are classified into different categories based on the machine environment (\(\alpha\)), job characteristics (\(\beta\)), and the optimization criterion (\(\gamma\)), and we assume that the exact values of all parameters are known. Single machine (1), \(m\)-machine flow shop (\(F_m\)), and \(m\)-machine job shop (\(J_m\)) are examples of common machine environments (\(\alpha\)), while having precedence relations between jobs (prec), or having families of jobs with similar characteristics (\(fml\)) are examples of job characteristics (\(\beta\)). Any minimization criterion such as \(C_{\text{max}}\), \(\sum C_j\), and \(\sum U_j\), is an example of \(\gamma\). Accordingly, we can specify every deterministic scheduling problem using Graham's notation \(\alpha|\beta|\gamma\) suggested in [35]. In robust optimization, however, we assume that different job-related parameters such as \(p_j\), \(d_j\), and \(w_j\) are uncertain and we optimize with respect to the worst-case data realization using a min-max objective. Therefore, we also specify the uncertain parameters in the notation, denoted by \(\eta\). Finally, we specify the robustness measure for the problem, \(\nu\), which we explain in detail in Section 3.2.1. The overall instance description for a robust scheduling problem is then given by \(\text{MinMax}_\nu(\alpha|\beta|\gamma, \eta)\).

In this chapter we review the robust optimization techniques that have been used for solving standard scheduling problems under uncertainty. Note that the majority of research in this area is focused on single-machine scheduling problems (SMSP). Thus,
we explore and classify the literature of robust SMSP, while addressing some other standard scheduling problems that have been investigated.

The remainder of the chapter is organized as follows. In Section 3.2, we represent different robustness criteria and uncertainty representations that have been introduced in robust optimization literature and address the use of each method in scheduling. Then in Section 3.3, we discuss the details of existing research in the area of robust SMSP, classify them according to the categories presented in Section 3.2, and explore the existing gaps and open problems. We then address related robust scheduling results in Section 3.4.

### 3.2 Robustness and Uncertainty Definitions

To define a robust scheduling problem, we need to represent the potential values of uncertain parameters in the problem and specify a measure by which we evaluate the robustness of a particular solution. In this section, we present the most common robustness measures and uncertainty representations for robust optimization in general and for robust scheduling in particular. We also discuss the benefits, limitations, and potential applications for each scheme.

Define a sequence, $\pi$, as a permutation of jobs and denote the set of all possible permutations by $\Pi$. A scenario, $\phi$, is a particular realization of uncertain parameters, where $\Phi$ represents the (possibly infinite-cardinality) set of all possible scenarios. Let $Z_\pi^\phi$ be the objective function value of job sequence $\pi$ under data realization $\phi$. Define $Z^\phi$ as the best achievable (optimal) objective function value when data scenario $\phi$ happens. We use this notation to present robustness and uncertainty definitions in the following two subsections.

#### 3.2.1 Robustness Measures

We discuss the three most common robustness measures that arise in the scheduling field: Absolute robustness, robust deviation, and relative robust deviation [49]. (See also Sabuncuoglu and Goren [71] for a more detailed classification for
possible robustness and stability measures, along with applications of some measures in solving robust SMSPs.)

“Absolute robustness” seeks to minimize the maximum objective function value over all scenarios. That is, when dealing with absolute robustness measure, we consider the worst-case scenario corresponding to each sequence of jobs, and select the sequence whose worst-case objective value is minimum, compared to all other feasible sequences. We can mathematically state absolute robustness as \( \min_{\pi \in \Pi} \max_{\phi \in \Phi} Z_\phi^\pi \). This measure is particularly useful for situations in which one seeks to guarantee at least a certain quality for the solution over all possible scenarios.

To understand the “robust deviation” measure, we first introduce the concept of regret. In the context of robust scheduling, a decision maker chooses a schedule before observing the data values. After the data is observed, the decision maker’s regret is given by the difference between his/her chosen schedule’s objective and the objective of the retrospective optimal solution, i.e., the optimal solution given knowledge of the data outcome. Robust deviation seeks to minimize the largest possible regret, and can be stated as \( \min_{\pi \in \Pi} \max_{\phi \in \Phi} (Z_\phi^\pi - Z^\phi) \). Minimizing robust deviation may be appropriate when determining a schedule whose performance, compared to the corresponding optimal performance, is relatively insensitive to data realization. Solutions that minimize robust deviation can also be interpreted as \textit{uniformly suboptimal} solutions, i.e., \( \epsilon \)-optimal solutions for all data realizations, with \( \epsilon \) as small as possible.

“Relative robust deviation” minimizes the maximum relative (or percentage) deviation from optimality, i.e., \( \min_{\pi \in \Pi} \max_{\phi \in \Phi} ((Z_\phi^\pi - Z^\phi)/Z^\phi) \). In fact, relative robust deviation is a normalized robust deviation measure that seeks to minimize the relative regret in the problem. Since the value of relative regret only depends on the ratio of objective values, it can be used to create benchmarks to compare the quality of different problems’ solutions.
Robust deviation and relative robust deviation measures are appropriate for environments in which the quality of solutions are evaluated after the data is realized. In such cases the (relative) deviation or the relative deviation of the selected decision from the optimal decision for the realized scenario is a plausible quality measure. For highly competitive markets, where a firm needs to have a satisfactory performance compared to its competitors, under any potentially realizable scenario, the use of these measures is also appropriate [49].

To understand the difference between robustness measures, consider an example in which we seek to find a two-job schedule that minimizes total completion time, where \( p_1 \in [4, 5] \) and \( p_2 \in [1, 6] \). If we apply the absolute robustness measure, the worst-case scenario for any sequence occurs when \( p_1 = 5 \) and \( p_2 = 6 \). Therefore, the robust optimal solution is obtained by scheduling job 1 before job 2 (sequence 1, 2). However, when robust deviation is applied, the largest regret value equals 4 for sequence 1, 2 (when \( p_1 = 5 \) and \( p_2 = 1 \)) and equals 2 for sequence 2, 1 (when \( p_1 = 4 \) and \( p_2 = 6 \)). Therefore, scheduling job 2 before job 1 is favorable when robust deviation measure is used. Note that using relative robust deviation for this instance also results in sequence 2, 1 being optimal (largest relative regret value equals 4/7 for sequence 1, 2 and 2/14 for sequence 2, 1). However, robust deviation and relative robust deviation measures may result in different optimal solutions in other instances. For example, suppose that we modify the above example by letting \( p_1 \in [4, 5] \) and \( p_2 \in [2, 8] \). Given sequence 1, 2, the largest absolute regret value (3) is obtained when \( p_1 = 5 \) and \( p_2 = 2 \): The objective of sequence 1, 2 is 12, while the optimal objective value for this scenario is 9. Sequence 2, 1, on the other hand, results in the largest regret value of 4, and therefore sequence 1, 2 is preferable with respect to robust deviation measure. However, the relative regret value is 3/9 for sequence 1, 2 and 4/16 for sequence 2, 1, and so sequence 2, 1 is optimal under relative robust deviation measure.
3.2.2 Uncertainty Representation

In the robust optimization literature, several methods of expressing uncertain parameter values have been proposed. In this section, we discuss four main methods of uncertainty representation, present benefits and drawbacks of each method, and address some applications of each method in robust scheduling.

The most traditional method of presenting uncertainty in robust optimization problems was developed by Soyster [76]. Soyster assumes that each uncertain input data, independent of all the others, can take on any values within a continuous interval (for the sake of brevity, we call this representation “interval uncertainty”) and proposes a linear optimization model to solve the problem. The simplicity of this method and its resulting formulations motivates its extensive application in robust scheduling problems. See [7, 44, 62, 65] as examples of using interval uncertainty to represent the values of different job-specific parameters of SMSP having different optimization criteria.

Note that if data uncertainty is represented using Soyster’s method and we seek to minimize the absolute robustness, setting all parameters to their worst-case values will generate a worst-case scenario corresponding to any sequence of jobs. Therefore, robust SMSPs having the absolute robustness criterion under interval uncertainty is equivalent to a deterministic SMSP in which all data elements take on their worst-case values. However, generating a worst-case scenario when robust deviation or relative robust deviation measures are applied is not obvious. For example, suppose we seek to minimize total completion time in an SMSP with two jobs where $p_1 \in [1, 5]$ and $p_2 \in [3, 4]$ and we use robust deviation measure. The worst-case scenario for the sequence in which job 1 is processed before job 2 is when $p_1$ takes on its largest possible value (5) and $p_2$ takes on its smallest possible value (3), and so the regret is given by 2. Similarly, for the reverse sequence (job 2 before job 1), in the worst-case scenario we have $p_1 = 1$ and $p_2 = 4$, which results in robust deviation value of 3. Therefore, sequence
1, 2 minimizes robust deviation. Note that the optimal sequence for this problem with absolute robustness measure is the opposite (job 2 first and job 1 second).

An important disadvantage of this method is that correlations between the values of parameters is not addressed. One way to capture these correlations is to enumerate all possible scenarios and represent uncertain parameters as a set of discrete scenarios for their numerical values. We refer to this method as scenario-based uncertainty.

In the robust SMSP literature, using scenario-based uncertainty is common [3, 25, 27, 63]. This method allows the decision maker to consider the relationship between all uncertain factors in the scheduling environment and include all possible cases in the model. It also maintains control over the level of conservatism in the problem. However, this approach potentially requires enumeration of a large number of data outcomes, which is computationally intractable in some cases.

Although interval uncertainty results in simple robust formulations for several problems, it tends to generate schedules that are over-conservative in the case of absolute robustness. Ben-Tal and Nemirovski [13–15] and El-Ghaoui et al. [28, 29] address this issue and introduce a method to reduce the level of conservatism by confining the data to belong to uncertainty sets in the form of ellipsoids. They propose efficient algorithms to solve the resulting convex optimization problems.

Bertsimas and Sim [16, 17] propose budgeted uncertainty as an alternative approach to control the level of conservatism for general robust optimization problems, which produces a linear robust formulation. This method assumes that all uncertain parameters independently take on values according to a symmetric distribution with known mean values (their ideal values) and perturbation bounds. Since it is unlikely that all parameters fail to take on their predicted values, they limit the number of perturbed parameters (the ones with values other than their ideal values) in each constraint and in the objective function, separately. They also limit the amount of perturbation in the value of each individual parameter. Using this method to represent uncertain coefficients
of the objective function and constraints of an optimization problem, they propose a robust programming problem of moderately larger size. They prove that using budgeted uncertainty, a robust linear programming problem can still be solved as a linear program, and the robust counterpart of a polynomially solvable 0-1 linear optimization problem remains polynomially solvable.

To the best of our knowledge, budgeted uncertainty method is applied to robust SMSP in only one paper [77], which we discuss in Section 3.3. This method has also been directly applied in several other scheduling problems in the area of chemical process scheduling, as we discuss in Section 3.4.

### 3.3 Robust Single-Machine Scheduling Literature Classification

In this section, we review the robust single-machine scheduling literature and classify existing research in this area according to their robustness measure definition and uncertainty representation. Kasperski [45] summarizes some of the results obtained in the robust SMSP literature for different cases (specified by a robustness measure and an uncertainty representation) and introduces open cases in this area. A short survey of the results is also presented in [2]. Here, we provide an updated survey of the results obtained in each category of SMSPs.

As stated in Section 3.2.2, the case of absolute robustness when dealing with continuous intervals of uncertainty can be equivalently solved as a deterministic problem and therefore is out of the scope of this chapter. However, when parameter values are confined to belong to some uncertainty sets, as in budgeted uncertainty method, there is no obvious equivalent deterministic version of the problem. Tadayon and Smith [77] define three alternative uncertainty sets for the absolute robust SMSP when processing-time values are presented as independent continuous intervals. In uncertainty sets 1, 2, and 3, they respectively require the total delay, the number of delayed jobs, and the total ratio by which the processing times are increased to be no more than a constant value. They study the complexity of the problem MinMax(1 ||Z , p_j),
where $Z \in \{\sum C_j, \sum w_j C_j, L_{\text{max}}, T_{\text{max}}, \sum U_j\}$ under each uncertainty set, propose exact algorithms for polynomially solvable problems, and present mixed-integer programming formulations for the NP-hard problems and the ones with unknown complexity. We present the details and the results of this research in Chapter 4.

Absolute robust SMSP in presence of scenario-based uncertainty has been studied in several papers. Yang and Yu [84] prove that problem $\text{MinMax}_U(1 \mid \sum C_j, p_j)$ with scenario-based uncertainty is NP-hard for all three robustness measures (even in the special case of having only two scenarios). They also present an exact dynamic programming algorithm with exponential complexity and two polynomial time heuristics to solve this robust SMSP.

Aloulou and Della Croce [3] study several other absolute robust SMSPs with scenario-based uncertainty in different job-related parameters. When the scheduler seeks to minimize the number of late jobs, they prove that problems $\text{MinMax}(1 \mid \sum U_j, p_j)$ and $\text{MinMax}(1 \mid \sum U_j, p_j, d_j)$ are NP-hard, but the case of $\text{MinMax}(1 \mid \sum U_j, d_j)$ is still open. Also, when the objective function of the problem is minimizing $f_{\text{max}} \in \{C_{\text{max}}, L_{\text{max}}, T_{\text{max}}\}$ and precedence relations between jobs in the sequence are allowed, they propose polynomial algorithms for solving the problem in which $p_j$, $d_j$, or both are expressed as a set of discrete scenarios. In addition, they prove that problem $\text{MinMax}(1 \mid \sum w_j C_j, w_j)$ with scenario-based uncertainty is NP-hard. (Notice that NP-hardness of $\text{MinMax}(1 \mid \sum w_j C_j, p_j)$ also follows from NP-hardness of $\text{MinMax}(1 \mid \sum C_j, p_j)$, which was proved in [84]).

More recently, Mastrolilli et al. [63] seek to generate approximation schemes for problem $\text{MinMax}(1 \mid \sum w_j C_j, \{w_j, p_j\})$. They prove that the problem cannot be approximated within $O(\log^{1-\epsilon} n)$ for any $\epsilon > 0$, unless quasi-polynomial algorithms exist for NP. For the special case of unweighted jobs, they propose a 2-approximation algorithm for solving $\text{MinMax}(1 \mid \sum C_j, p_j)$ and prove that the problem is NP-hard to approximate within a factor less than 6/5.
Despite the extensive application of absolute robustness in the robust optimization literature, robust deviation has been investigated more widely in SMSP literature. When independent continuous intervals are used to represent uncertain processing times, Lebedev and Averbakh [51] prove that the general case of problem MinMax_{dev}(1 || \sum C_j, p_j) is NP-hard; however, they show that when all intervals of uncertainty have the same center, the problem can be solved polynomially (in O(n log n) time) if the number of jobs is odd, and is NP-hard otherwise. Montemanni [65] presents the first mixed-integer linear programming formulation for problem MinMax_{dev}(1 || \sum C_j, p_j) with interval uncertainty and translates some preprocessing rules into valid inequalities.

Kasperski and Zielinski [46] prove that any minmax robust deviation SMSP with total completion time criterion and interval uncertainty, whose equivalent deterministic problem is polynomially solvable, can be approximated within a factor of 2. A more general case of this problem with weighted sum of completion time criterion (MinMax_{dev}(1 || \sum w_j C_j, p_j)) has been studied in [75] in which some dominance relations are introduced and a heuristic algorithm is proposed to find a good nondominated sequence.

Robust deviation in the case of interval uncertainty is also considered in [7], [44], and [62] for different SMSPs. Averbakh [7] proves that MinMax_{dev}(1 || \max\{w_j T_j, w_j\}) is polynomially solvable by presenting an O(n^3) algorithm for the problem. Kasperski [44] proposes a polynomial algorithm (with O(n^4) complexity) for optimally solving MinMax_{dev}(1 | prec| L_{max}, \{p_j, d_j\}). Lu et al. [62] define a specific SMSP with total completion time objective function in which jobs are grouped into families and a sequence-dependent family setup time exists. They prove that when both job processing times and family setup times are presented as intervals of uncertainty, minimizing robust deviation in the corresponding problem is NP-hard. They then propose a simulated annealing-based algorithm to find good quality solutions for the problem.
Several researchers study robust deviation in SMSPs in which parameters are presented as discrete scenarios. Daniels and Kouvelis [25] examine problems $\text{MinMax}_{\text{dev}}(1 \parallel \sum C_j, p_j)$ and $\text{MinMax}_{\text{rel}}(1 \parallel \sum C_j, p_j)$ when the uncertainty set consists of discrete scenarios for the $p_j$-values. They prove that both problems are NP-hard and present some dominance relations that can be used to determine the relative job positions in a robust schedule. Then, they use these results to develop an exact branch-and-bound algorithm for solving the robust problem. They also present intuitive heuristic approaches and evaluate their efficiency and accuracy through a set of randomly-generated instances. Additionally, Daniels and Kouvelis prove that even when processing times are presented as independent continuous intervals, the worst-case scenario for each sequence $\pi$ belongs to the finite set of extreme-point scenarios (scenarios in which each parameter takes on its lower- or upper-bound values). According to this result, they claim that the formulations presented for scenario-based uncertainty are also valid for interval uncertainty.

As stated before, Yang and Yu [84] approach the same problem with robust deviation as one of the three robustness measures and present a different NP-hardness proof for the problem. A more general problem with the weighted sum of completion time criterion (using robust deviation measure and scenario-based uncertainty in job processing times) is studied in [27], where a set of valid inequalities for the convex hull of its feasible region is presented and is then utilized to design a cutting-plane algorithm for solving the problem.

To the best of our knowledge, relative robust deviation has been explicitly considered in only three SMSP papers so far. Averbakh [8] studies this measure of robustness for robust combinatorial optimization problems in general and for SMSP as a special case, and proves that the problem $\text{MinMax}_{\text{rel}}(1 \parallel \sum C_j, p_j)$ with interval uncertainty is NP-hard. Daniels and Kouvelis state that the analysis, results, and solution methods presented in [25] for the case of robust deviation can be applied for the relative robust deviation
measure with a slight modification. Yang and Yu \cite{84} prove the NP-hardness of problem MinMax$_v$(1 || \( \sum C_j, p_j \)) with scenario-based uncertainty for all three robustness measures (including relative robust deviation).

We summarize the results obtained in the literature of robust SMSP in Table 3-1. In this table, we specify the complexity of robust SMSP problems under each robustness measure (absolute robustness “abs. rob.”, robust deviation “rob. dev.”, and relative robust deviation “rel. rob. dev.”) and each uncertainty representation (budgeted uncertainty “budg. uncert.”, interval uncertainty “interval”, and scenario-based uncertainty “scenario”), as well as the reference from which the result is extracted. Each problem is specified by its objective function and uncertain parameter in the first column labeled as “(obj., param.)”. The complexity of the problems that are proved to be NP-hard are listed in the table as “NPH”. Note that budgeted uncertainty has been considered only for absolute robustness criterion. Among the results obtained in \cite{77}, we only include the ones about the first uncertainty set in this table, for the sake of brevity. For information about the complexity of each problem under the other two uncertainty sets, refer to Chapter 4. The cells whose corresponding problems have not been yet studied in the literature are marked by “-”.

**Table 3-1. Complexity results obtained in the literature for robust SMSP**

<table>
<thead>
<tr>
<th>(obj., param.)</th>
<th>abs. rob.</th>
<th>rob. dev.</th>
<th>rel. rob. dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\sum C_j, p_j))</td>
<td>(O(n \log n)) \cite{77}</td>
<td>NPH \cite{84}</td>
<td>NPH \cite{51}</td>
</tr>
<tr>
<td>((\sum w_j C_j, p_j))</td>
<td>Open \cite{77}</td>
<td>NPH \cite{84}</td>
<td>NPH \cite{51}</td>
</tr>
<tr>
<td>((\sum w_j C_j, w_j))</td>
<td>-</td>
<td>NPH \cite{84}</td>
<td>NPH \cite{51}</td>
</tr>
<tr>
<td>((\sum U_j, p_j))</td>
<td>(O(n \log n)) \cite{77}</td>
<td>NPH \cite{3}</td>
<td>-</td>
</tr>
<tr>
<td>((\sum U_j, d_j))</td>
<td>-</td>
<td>Open \cite{3}</td>
<td>-</td>
</tr>
<tr>
<td>((\sum U_j, {p_j, w_j}))</td>
<td>-</td>
<td>NPH \cite{3}</td>
<td>-</td>
</tr>
<tr>
<td>((L_{max}, p_j))</td>
<td>(O(n \log n)) \cite{77}</td>
<td>(O(n^2</td>
<td>S</td>
</tr>
<tr>
<td>((L_{max}, d_j))</td>
<td>-</td>
<td>(O(n^2 + n</td>
<td>S</td>
</tr>
<tr>
<td>((L_{max}, {p_j, d_j}))</td>
<td>-</td>
<td>(O(n^2</td>
<td>S</td>
</tr>
<tr>
<td>((T_{max}, p_j))</td>
<td>(O(n \log n)) \cite{77}</td>
<td>(O(n^2</td>
<td>S</td>
</tr>
<tr>
<td>((T_{max}, d_j))</td>
<td>-</td>
<td>(O(n^2 + n</td>
<td>S</td>
</tr>
<tr>
<td>((T_{max}, {p_j, d_j}))</td>
<td>-</td>
<td>(O(n^2</td>
<td>S</td>
</tr>
<tr>
<td>((C_{max}, p_j))</td>
<td>-</td>
<td>(O(n^2</td>
<td>S</td>
</tr>
<tr>
<td>((\max w_j T_j, w_j))</td>
<td>-</td>
<td>-</td>
<td>(O(n^3)) \cite{7}</td>
</tr>
</tbody>
</table>
3.4 Robust Optimization in Other Scheduling Problems

Although robust optimization, as defined in this chapter, has been most frequently studied for the single-machine scheduling environment, other scheduling problems have been also addressed in the literature. Moreover, the concept of robustness in some studies extends beyond the standard definitions presented earlier in this chapter, and include (a) the restriction that a schedule remains feasible with a certain probability [41], (b) that a schedule guarantees a certain performance level [24, 81]), or (c) the incorporation of different uncertainties in scheduling environments (e.g., unpredictable production interruptions [52]). In this section, we introduce some of the other areas of scheduling in which different variations of robust optimization has been applied to hedge against uncertainty in production environments.

Kouvelis et al. [48] apply a robustness definition similar to the one presented in this chapter to comply with uncertainty in a two-machine flow shop environment. They prove that problem MinMax_{dev}(F_2 | C_{max}, p_j) is NP-hard. They also discuss the properties of robust schedules and develop exact and heuristic solution approaches for the problem under both interval uncertainty and scenario-based uncertainty in jobs processing times.

Although ellipsoidal and budgeted uncertainty (presented by Ben-Tal and Nemirovski [13–15], El-Ghaoui et al. [28, 29], and Bertsimas and Sim [16, 17]) have not been widely addressed in robust single-machine scheduling literature yet, it has been applied to other scheduling problems, mainly in the area of chemical process scheduling.

Production scheduling, as one of the most important problems in industrial plant operations, has been studied in several research papers. Short-term production scheduling seeks to determine an optimal assignment of available resources to production tasks over time while satisfying the production requirements at due dates. Depending on the plant layout and the sequence of tasks, process scheduling is usually in the form of a standard flow shop or job shop scheduling problem. Ierapetritou and Floudas [39] propose a mixed-integer linear programming formulation for this problem.
Li and Ierapetritou [54] consider uncertainty in parameter values in the formulation presented in [39]. They implement the three main continuous uncertainty representation approaches for robust optimization, i.e., Soyster's method, Ben-Tal and Nemirovski's method, and Bertsimas and Sim's method (budgeted uncertainty), to formulate the robust process scheduling problem given uncertainty regarding product unit price, task processing times, and product demand values. They compare the three formulations by solving test instances from linear formulations using CPLEX and the ones from nonlinear formulations using the DICOPT solver via GAMS. The results of their computations show that Bertsimas and Sim's method yields the most appropriate formulation for the robust scheduling problem, according to the fact that this method does not increase the problem size substantially and maintains the linearity of the model.

Lin et al. [56] and Janak et al. [41] consider uncertainty in different parameters of the process scheduling problem as formulated in [39]. Uncertain parameters (price, processing times, and demands) are presented as independent continuous intervals in [56] and as random variables with known probability distributions in [41]. Janak et al. [41] define a robust solution to be one that is feasible in all constraints with a certain probability. Mathematical formulations for the robust problems are presented in both papers and the efficiency of models are investigated by solving test instances from linear formulations using CPLEX and from nonlinear models using DICOPT.

Li and Lerapetritou [53] review the literature of process scheduling under uncertainty. They introduce robust optimization as one of the alternative approaches in dealing with uncertainty in process scheduling and address the main advances and challenges in this area of research. Verderame et al. [80] provide a broader overview of planning and scheduling problems under uncertainty, across multiple sectors. Although the objective function and constraints of the model varies greatly among different sectors,
they address the common attribute of requiring a standard robustness definition and uncertainty representation in all areas and encourage interdisciplinary research.

Leon et al. [52] define a different type of uncertainty in a standard job shop environment in which disruptions occur randomly during the process. They seek to find a sequence of jobs that is robust under different scenarios of disruptions, assuming that preemption is not allowed (when a job’s process is interrupted, it has to be restarted), the performance measure is makespan, and the order of jobs cannot be revised after interruptions occur. They define the robustness measure as a randomly weighted sum of the expected makespan and the expected difference between the actual makespans and the deterministic makespan (without interruptions). Because the effect of each disruption depends on the outcome of all previous disruptions, they compute the robustness measure only in the case of a single interruption. For the case of several disruptions, they develop a surrogate method and embed it in a genetic algorithm to generate relatively robust schedules.

A robust solution in some applications is defined as a solution that guarantees a certain level of performance under all possible data realizations. Daniels and Carrillo [24] define a $\beta$-robust solution as a job sequence that maximizes the likelihood of achieving a total completion time value no greater than a constant value in a single-machine scheduling problem with uncertain processing times. They assume that jobs processing times are independent random variables with known means and variances that can be represented as discrete scenarios when each scenario occurs with a certain probability. They prove that the problem is NP-hard and develop an exact branch-and-bound algorithm and a polynomial heuristic for solving the problem. Wu et al. [81] consider a similar problem with independent normally distributed processing-time values and present three models: primal (maximizing the probability of obtaining a certain level of performance), dual (minimizing the level that can be achieved with a fixed probability), and hybrid. They determine the feasibility of each model for a set of
randomly generated problems and study the effect of some dominance rules in solving the problems.
We examine in this chapter a scheduling problem in which a set of jobs, \( J \), must be processed on a single machine, one at a time, without preemption. Every job \( j \in J \) requires a specific amount of time, \( p_j \), to be processed on the machine, and in some applications, is associated with a due date, \( d_j \). Also, for situations in which the jobs are not equally important, job \( j \) is associated with a weight, \( w_j \). We focus in this chapter on several alternative objective functions. Defining \( C_j \) to be the completion time of job \( j \), we consider the due-date independent objectives of minimizing total completion time of jobs (\( \sum C_j \)) and minimizing total weighted completion time of jobs (\( \sum w_j C_j \)). For cases in which due dates are relevant, we consider the problem of minimizing the number of late jobs (\( \sum U_j \), where \( U_j = 1 \) if \( C_j > d_j \) and is 0 otherwise), minimizing the maximum lateness (\( L_{\text{max}} = \max_j \{L_j\} \), where \( L_j = C_j - d_j \)), or minimizing the maximum tardiness (\( T_{\text{max}} = \max_j \{T_j\} \), where \( T_j = \max\{0, C_j - d_j\} \)).

Scheduling problems under uncertainty have received an extensive amount of attention during the last few decades. Schedules that are optimal with respect to deterministic data can be suboptimal in practice due to uncertainties in parameters such as processing times, due dates, and weights. Two approaches that can be used to model uncertainty in optimization problems include stochastic programming and robust optimization. Stochastic programming typically seeks to optimize a solution’s expected objective value. This approach requires some knowledge of the probability distribution for all nondeterministic parameters, which is often handled by sampling strategies (see, e.g., \([18, 43]\)).

Robust optimization is an alternative approach for dealing with uncertain data \([12, 72]\), which assumes that all uncertain data values are realized after the decisions have been selected. In robust optimization problems, the decision variables must remain
feasible under any data outcome. The objective (for minimization problems) seeks to minimize the maximum possible objective function value that could occur for the selected decision variables. In the context of our scheduling problems, a solution refers to a job permutation, which remains feasible for any data realization. The challenges that we face in these studies is to characterize which data outcomes result in worst-case objective function values (as a function of the chosen schedules), and to determine how to minimize those worst-case values.

Several criteria can be applied to measure the robustness of a particular solution. Kouvelis and Yu [49] introduce three general robustness measures called absolute robustness, robust deviation, and relative robust deviation. Absolute robustness is used when the goal is to minimize the objective function of the worst-case scenario, as we do in this chapter. Robust deviation (or absolute regret) seeks to minimize the largest possible difference between the observed objective function value and the optimal objective function value. Relative robust deviation (or relative regret) minimizes the largest possible ratio of robust deviation to the optimal objective function value.

Kaspersky [45] summarizes some results for robust scheduling problems (specified by a robustness measurement and an uncertainty representation) and introduces open cases in this area. Aissi et al. [2] present a survey of regret-based combinatorial optimization problems. See also [71] for a different categorization for robustness and stability measures, along with a review of single-machine scheduling problems (SMSP) in each category. For a more recent survey of the results obtained in the literature of robust single-machine scheduling, see Chapter 3 or refer to the technical report [78].

In this chapter we consider a budgeted uncertainty model in which processing times are uncertain and are confined to some specified interval, and where we limit the total magnitude of deviation from their ideal values. This approach, in a more general sense, constrains uncertain data to lie within some polyhedron. As opposed to interval uncertainty, our analysis prohibits data from simultaneously taking on worst-case values,
and instead concentrates on a less-conservative analysis of data realizations. See [17] for a thorough discussion of budgeted uncertainty models. In order to specify the problem in this chapter, we apply the same notation introduced in Chapter 3, i.e., \( \nu(\alpha | \beta | \gamma, \eta) \). The definition of each parameter (\( \nu, \alpha, \beta, \gamma \), and \( \eta \)) in this notation is presented in Section 3.1.

The remainder of this chapter is organized as follows. In Section 4.2 we provide the problem definition and discuss three different uncertainty sets that constrain total deviation in the ideal parameter values. In Section 4.3 we investigate absolute robustness in the SMSP under each uncertainty set with four commonly-used minimization criteria: total completion time (\( \sum C_j \)), total weighted completion time (\( \sum w_j C_j \)), maximum lateness/tardiness (\( L_{\text{max}} \) or \( T_{\text{max}} \)), and number of late jobs (\( \sum U_j \)).

### 4.2 Problem Definition and Notation

Let \( J = \{1, \ldots, n\} \) be the set of jobs to be processed. The ideal processing time of job \( j \in J \), given by \( p_j \), is defined as the (best-case) time required by the machine to process job \( j \), assuming that no other factors affect the process. The actual processing time of job \( j \) can be longer than the ideal processing time, in which case we say that job \( j \) is delayed. Denote the quantity of delay for job \( j \) by \( \delta_j \). Since in several applications, jobs having longer processing times are more likely to be delayed by a greater value, we assume that \( \delta_j \) is a proportion of \( p_j \) (i.e., \( \delta_j = k_j p_j \) for some nonnegative multiplier \( k_j \)) and limit the total delay for each job \( j \in J \) by requiring that \( k_j \leq K (\delta_j \leq Kp_j) \).

In addition to limiting the delay of each job, we also restrict the total amount of delay to control the level of uncertainty in the problem, i.e., \( (\delta_1, \ldots, \delta_n) \in S \), where \( S \) is the set of all possible \( \delta \)-values. In particular, we study three classes of uncertainty sets.

**Uncertainty set 1 (US1)** requires the total amount of delay to be no more than a constant, \( \Delta \), i.e., \( \sum_{j \in J} \delta_j \leq \Delta \). For the next uncertainty set, we define \( m_j \) as a binary variable that equals one if job \( j \) is delayed, and zero otherwise. **Uncertainty set 2 (US2)** limits the number of delayed jobs by an integer, \( M \), i.e., \( \sum_{j \in J} m_j \leq M \). Finally,
uncertainty set 3 (US3) ensures that the total ratio by which the processing times are increased cannot be more than a constant, $\kappa$, i.e., $\sum_{j \in J} k_j \leq \kappa$. Here, without loss of generality, we assume that $\Delta \leq \sum_{j \in J} Kp_j$, $M \leq n$, and $\kappa \leq Kn$.

We define a sequence, $\pi$, as a permutation of jobs and denote the set of all possible permutations by $\Pi$. The $j^{th}$ job in $\pi$ is denoted by $\pi_j$. A scenario, $\phi$, in this chapter is a particular realization of job processing times, where $\Phi$ represents the (infinite-cardinality) set of all scenarios in our uncertainty set. Given a job sequence $\pi$ and data scenario $\phi$, define $C_{\pi_j}^\phi$ to be the completion time of the $j^{th}$ job, and $Z_{\pi}^\phi$ to be the objective function value of the sequence. The actual processing time of the $j^{th}$ job in $\pi$ under scenario $\phi$ is denoted by $p_{\pi_j}^\phi$. For the ideal scenario in which all jobs take their ideal processing times, we denote the completion time of the $j^{th}$ job by $\tilde{C}_{\pi_j}$, and the objective value corresponding to this sequence as $Z_{\pi}$. Corresponding to each sequence $\pi$, a “worst-case scenario” ($\phi^*(\pi)$) is defined to be a scenario that maximizes the objective function given the job sequence $\pi$ (where for notation simplicity, $\phi^*(\pi)$ is replaced by $\phi^*$, unless it results in confusion).

To better understand the problem, we investigate the worst-case scenario and the optimal sequence for all three uncertainty sets in a problem instance presented in Example 4.1.

Example 4.1. Consider a two-job instance where $p_1 = 8$, $w_1 = 10$, $p_2 = 1$, and $w_2 = 1$. Let $K = 0.5$, and uncertainty budgets corresponding to US1, US2, and US3 are given by $\Delta = 4$, $M = 1$, and $\kappa = 0.5$, respectively. We seek to identify a sequence of jobs that minimizes total weighted completion time under the absolute robustness measure.

There are two possible sequences for this instance. The first sequence ($\pi^1$) schedules job 1 before job 2. By inspection, the worst-case scenario for all three uncertainty sets is achieved by increasing the processing time of job 1 by its maximum possible value (4). Therefore, $p_{\pi^1_1}^{\phi^*} = p_{\pi^1_1}^\phi = 12$, $p_{\pi^1_2}^{\phi^*} = p_{\pi^1_2}^\phi = 1$, and $Z_{\pi^1}^{\phi^*} = 10(12) + 1(12 + 1) = 133$ for all three uncertainty sets.
The second sequence ($\pi^2$) schedules job 2 first and job 1 second. For US1, the worst-case scenario increases the processing time of job 2 by 0.5 ($\min\{Kp_2, \Delta\}$), and the processing time of job 1 by 3.5 ($\min\{Kp_1, \Delta - 0.5\}$). Therefore, for US1, $p_{\pi^2_1}^{\phi^*} = p_{\pi^2_2}^{\phi^*} = 1.5$, $p_{\pi^2_1}^{\phi^*} = p_{\pi^2_2}^{\phi^*} = 11.5$, and $Z_{\pi^2}^{\phi^*} = 1(1.5) + 10(1.5 + 11.5) = 131.5$. For US2 and US3, the worst-case scenario increases the processing time of job 1 by 4. As a result, in those cases, $p_{\pi^2_1}^{\phi^*} = p_{\pi^2_2}^{\phi^*} = 1$, $p_{\pi^2_1}^{\phi^*} = p_{\pi^2_2}^{\phi^*} = 12$, and $Z_{\pi^2}^{\phi^*} = 1(1) + 10(1 + 12) = 131$. Hence, the second schedule is optimal, regardless of which uncertainty set we choose.

### 4.3 Complexity Results and Algorithms

In this section we examine our robust scheduling problem for each combination of objective functions described in Section 4.1 and uncertainty sets defined in Section 4.2. In each case, we first characterize the problem of generating the worst-case scenario for a given sequence $\pi$ (which we call the scenario-generation problem (SGP)), and then use the obtained results to determine the relative position of jobs in a robust schedule.

The SGP can be formulated as:

\[
\text{Max } R(\delta_{\pi_1}, \ldots, \delta_{\pi_n}) \tag{4-1}
\]

subject to:

\[
F(\delta_{\pi_1}, \ldots, \delta_{\pi_n}) \leq b \tag{4-2}
\]

\[
(\delta_{\pi_1}, \ldots, \delta_{\pi_n}) \in S, \tag{4-3}
\]

where $R(\delta_{\pi_1}, \ldots, \delta_{\pi_n})$ is the total increase in the ideal objective value (to which we refer as the total penalty) by delaying jobs and $F(\delta_{\pi_1}, \ldots, \delta_{\pi_n})$ is the amount of uncertainty budget that has been used due to job delays.

We summarize in Table 4-1 the results to be presented in this section. This table gives the algorithm complexity that we obtained for each problem (SGP and robust optimization problem) under each objective function ($\sum C_j$, $\sum w_j C_j$, $L_{\max}$ or $T_{\max}$, and $\sum U_j$) and each uncertainty set (US1, US2, and US3). For those problems where we could not identify a polynomial-time algorithm, we denote the complexity result.
by “Open(MIP)” if we specify a mixed-integer programming (MIP) formulation for the problem and by “Open” otherwise. Note that for these problems, we have also not identified an NP-hardness proof, which is why we specify that their complexity remains open.

Table 4-1. Complexity results for robust SMSP under budgeted uncertainty

<table>
<thead>
<tr>
<th>Objective</th>
<th>US1</th>
<th>US2</th>
<th>US3</th>
<th>Robust optimization problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sum_{j \in J} C_j)</td>
<td>(O(n))</td>
<td>(O(n \log n))</td>
<td>(O(n \log n))</td>
<td>(O(n \log n))</td>
</tr>
<tr>
<td>(\sum_{j \in J} w_j C_j)</td>
<td>(O(n))</td>
<td>(O(n \log n))</td>
<td>(O(n \log n))</td>
<td>Open(MIP)</td>
</tr>
<tr>
<td>(L_{\max}) or (T_{\max})</td>
<td>(O(n))</td>
<td>(O(n \log M))</td>
<td>(O(n \log \left\lceil \frac{K}{K} \right\rceil))</td>
<td>(O(n \log n))</td>
</tr>
<tr>
<td>(\sum_{j \in J} U_j)</td>
<td>(O(n))</td>
<td>(O(Mn^2))</td>
<td>Open(MIP)</td>
<td>(O(n^2))</td>
</tr>
</tbody>
</table>

4.3.1 Minimizing Total Completion Time

We seek to find a sequence \(\pi\) that minimizes the maximum total completion time of jobs \((\sum_{j \in J} C_{\pi_j})\) under all possible scenarios. The deterministic version of this problem is solvable by sequencing the jobs in nondecreasing order of their \(p\)-values, i.e., in shortest processing time (SPT) order [73]. We show in Section 4.3.1.1 how the SGP is solved in this case, and then prove that an SPT job ordering optimizes the robust scheduling problem under all three uncertainty sets.

4.3.1.1 Scenario Generation Problem

The SGP formulation corresponding to minimizing total completion time can be mathematically stated by (4–1)–(4–3) in which (4–2) can be equivalently restated as
\[ \sum_{j=1}^{n} F_j(\delta_{\pi_j}) \leq b, \]
where the specific form of each \(F_j\)-function and the value of \(b\) depend on which uncertainty set is used. We next prove that we can also express \(R(\delta_{\pi_1}, \ldots, \delta_{\pi_n})\) as a linear function of \(\delta\) regardless of the uncertainty set.

**Theorem 4.1.** For the problem of minimizing total completion time, the SGP objective function, \(R(\delta_{\pi_1}, \ldots, \delta_{\pi_n})\), can be expressed as \(\sum_{j=1}^{n} R_j(\delta_{\pi_j})\), where \(R_j(\delta_{\pi_j}) = (n - j + 1)\delta_{\pi_j}, \)
\[ \forall j = 1, \ldots, n. \]

**Proof.** We prove the theorem by induction on the number of jobs. First suppose that \(n = 1\). Assuming that we delay this job by \(\delta_{\pi_1}\), the total completion time is \(Z^\phi = \bar{C}_{\pi_1} + \delta_{\pi_1}\),
and so the total penalty is \( R(\delta_{\pi_1}) = R_1(\delta_{\pi_1}) = Z^\phi_{\pi} - \bar{C}_{\pi_1} = \delta_{\pi_1} \), which is equivalent to \((n - 1 + 1)\delta_{\pi_1}\) when \(n = 1\). Next, suppose by induction that the total penalty for a sequence of \(m\) jobs is given by \(\sum_{j=1}^{m}(m - j + 1)\delta_{\pi_j}\). We will show that for a sequence of \(m + 1\) jobs, the total penalty is given by \(R(\delta_{\pi_1}, \ldots, \delta_{\pi_{m+1}}) = \sum_{j=1}^{m+1}(m - j + 2)\delta_{\pi_j}\). Let \(\pi'\) be the sequence of the first \(m\) jobs of \(\pi\). The total completion time of \(\pi'\) is \(\sum_{j=1}^{m} \bar{C}_{\pi_j} + \sum_{j=1}^{m}(m - j + 1)\delta_{\pi_j}\) by the induction assumption. We now append job \(\pi_{m+1}\) to the end of \(\pi'\) to create \(\pi\). The total completion time of \(\pi\) under scenario \(\phi\), \(Z^\phi_{\pi}\), is given by \(\sum_{j=1}^{m} \bar{C}_{\pi_j} + \sum_{j=1}^{m}(m - j + 1)\delta_{\pi_j} + C^\phi_{\pi_{m+1}}\), because adding job \(\pi_{m+1}\) to the end of \(\pi\) does not affect the completion times of the preceding jobs. Note that \(C^\phi_{\pi_{m+1}} = C^\phi_{\pi_m} + p_{\pi_{m+1}} + \delta_{\pi_{m+1}}\) and \(\bar{C}_{\pi_m} = \bar{C}_{\pi_m} + \sum_{j=1}^{m} \delta_{\pi_j}\). Therefore,

\[
Z^\phi_{\pi} = \sum_{j=1}^{m} \bar{C}_{\pi_j} + \sum_{j=1}^{m}(m - j + 1)\delta_{\pi_j} + \bar{C}_{\pi_m} + \sum_{j=1}^{m} \delta_{\pi_j} + p_{\pi_{m+1}} + \delta_{\pi_{m+1}}.
\]

Since \(\bar{C}_{\pi_m} + p_{\pi_{m+1}} = \bar{C}_{\pi_{m+1}}\), we have

\[
Z^\phi_{\pi} = \sum_{j=1}^{m+1} \bar{C}_{\pi_j} + \sum_{j=1}^{m}(m - j + 1)\delta_{\pi_j} + \sum_{j=1}^{m+1} \delta_{\pi_j},
\]

so \(Z^\phi_{\pi} = \sum_{j=1}^{m+1} \bar{C}_{\pi_j} + \sum_{j=1}^{m+1}(m - j + 2)\delta_{\pi_j}\). The total penalty is therefore \(R(\delta_{\pi_1}, \ldots, \delta_{\pi_{m+1}}) = Z^\phi_{\pi} - \sum_{j=1}^{m+1} \bar{C}_{\pi_j} = \sum_{j=1}^{m+1}(m - j + 2)\delta_{\pi_j}\), and the proof is complete.

Next, we address the optimization of the SGP corresponding to each uncertainty set. For each uncertainty set, we prescribe an ordering, \(O\), of jobs to delay, such that the following greedy delay rule optimally solves the SGP:

For each \(j = 1, \ldots, n\), in this order, delay job \(O_j\) by the maximum amount allowed by (4–2) and (4–3).

**Lemma 1.** If \(O_j = \pi_j\), \(\forall j = 1, \ldots, n\), then the greedy delay rule yields an optimal solution for the SGP under US1.

**Proof.** For US1, delaying the \(j^{th}\) job by \(\delta_{\pi_j}\) uses \(\delta_{\pi_j}\) of the uncertainty budget, \(\Delta\).

Therefore, \(F_j(\delta_{\pi_j}) = \delta_{\pi_j}, \forall j = 1, \ldots, n\), and \(b = \Delta\) in (4–2). Moreover, \(\delta_{\pi_j}\) can take
on any value between 0 and $K\rho_j$. Hence, by Theorem 4.1, we can formulate the SGP under US1 as follows.

$$\begin{align*}
\text{Max} \sum_{j=1}^{n} (n - j + 1)\delta_{\pi_j} \\
\text{subject to:} \\
\sum_{j=1}^{n} \delta_{\pi_j} \leq \Delta \\
0 \leq \delta_{\pi_j} \leq K\rho_{\pi_j}, \quad \forall j = 1, \ldots, n.
\end{align*}$$

(4–4)

Because the objective coefficients of $\delta_{\pi_1}, \ldots, \delta_{\pi_n}$ form a decreasing sequence, the greedy delay rule under the ordering specified by Lemma (1) optimizes this problem. $\square$

Solving the SGP under US1 requires $O(1)$ operations for each $j = 1, \ldots, n$, and therefore, its total complexity is $O(n)$.

For our analysis regarding US2 and US3, we define $\bar{R}_j$ as the largest possible value that $R_j(\delta_{\pi_j})$ could take if unconstrained by (4–2), i.e., $\bar{R}_j = (n - j + 1)K\rho_{\pi_j}$. The following lemmas establish optimal SGP solutions under US2 and US3.

Lemma 2. Consider an ordering $\mathcal{O}$ obtained by sorting jobs $j = 1, \ldots, n$ in nonincreasing order of their $\bar{R}$-values. The greedy delay rule with this ordering yields an optimal solution under US2.

Proof. In US2, each job that is delayed by a positive amount consumes one unit of the total uncertainty budget, $M$. As a result, in (4–2), $b = M$ and $F_j(\delta_{\pi_j}) = m_{\pi_j}, \forall j = 1, \ldots, n,$ where $m_{\pi_j}$ is one when $\delta_{\pi_j}$ is positive and zero otherwise. Therefore, the SGP in this case can be stated as follows.

$$\begin{align*}
\text{Max} \sum_{j=1}^{n} (n - j + 1)\delta_{\pi_j} \\
\text{subject to:}
\end{align*}$$

(4–7)
\[ \sum_{j=1}^{n} m_{\pi_j} \leq M \]  \hfill (4–8)

\[ 0 \leq \delta_{\pi_j} \leq Kp_{\pi_j}m_{\pi_j}, \quad \forall j = 1, \ldots, n \]  \hfill (4–9)

\[ m_{\pi_j} \in \{0, 1\}, \quad \forall j = 1, \ldots, n. \]  \hfill (4–10)

An optimal solution for this problem selects \( M \) variables \( \delta_{\pi_j} \) to take on their upper bounds \( (Kp_{\pi_j}) \). In particular, the \( M \) variables having the largest \( \bar{R} \)-values will be chosen to equal their upper bounds, thus completing the proof. \( \square \)

**Lemma 3.** Consider the ordering \( O \) obtained by sorting jobs \( j = 1, \ldots, n \) in nonincreasing order of their \( \bar{R} \)-values. Given this ordering, the greedy delay rule yields an optimal solution under US3.

**Proof.** In US3, delaying the \( j \)th job by \( \delta_{\pi_j} \) uses \( k_{\pi_j} = \delta_{\pi_j}/p_{\pi_j} \) of the uncertainty budget, \( \kappa \). Therefore, substituting \( \delta_{\pi_j} = p_{\pi_j}k_{\pi_j} \) in Theorem 4.1, the SGP is given by

\[
\text{Max} \quad \sum_{j=1}^{n} (n - j + 1)p_{\pi_j}k_{\pi_j} \quad \hfill (4–11)
\]

subject to:

\[
\sum_{j=1}^{n} k_{\pi_j} \leq \kappa \quad \hfill (4–12)
\]

\[ 0 \leq k_{\pi_j} \leq K, \quad \forall j = 1, \ldots, n. \]  \hfill (4–13)

Observe that this problem is equivalent to maximizing \( (1/K) \sum_{j=1}^{n} \bar{R}_j k_{\pi_j} \). Therefore, we can generate an optimal solution for this problem via the greedy delay rule. This completes the proof. \( \square \)

According to Lemmas 2 and 3, to solve the SGP under US2 and US3, we sort the jobs by their \( \bar{R} \)-values \( (O(n \log n)) \), and perform the greedy delay rule \( (O(n)) \). Thus, the total complexity of solving the SGP under US2 and US3 is \( O(n \log n) \).
4.3.1.2 Robust Optimization Problem

Lemmas 1–3 permit us to characterize optimal solutions to our robust scheduling problem.

**Theorem 4.2.** Any SPT schedule is optimal for problem MinMax(1|| $\sum C_j$, $p_j$), under all three uncertainty sets.

**Proof.** Suppose that $\pi'$ is an optimal sequence of jobs that does not follow the SPT order, i.e., there exists some $j \in \{1, \ldots, n\}$ such that $p_{\pi'_j} > p_{\pi'_{j+1}}$. We show that we can improve the sequence by creating an alternative sequence $\pi''$, in which we swap the order of the $j^{th}$ and the $(j + 1)^{st}$ jobs in $\pi'$ and retain the ordering of all other jobs. Figure 4-1 illustrates the two sequences. For brevity in notation, we substitute $\delta_{\pi'}^{\phi^*(\pi')} (\delta_{\pi''}^{\phi^*(\pi''})$, $Z_{\pi'}^{\phi^*(\pi')} (\delta_{\pi''}^{\phi^*(\pi''})$ and $Z_{\pi'}^{\phi^*(\pi')} (\delta_{\pi''}^{\phi^*(\pi''})$ to represent job delays and total completion time in the worst-case scenarios corresponding to $\pi'$ ($\pi''$), respectively. We thus seek to show that

$$Z_{\pi'} - Z_{\pi''} = \sum_{q=1}^{n} (n - q + 1) \left[ (p_{\pi'_q} - p_{\pi'_{q+1}}) + (\delta_{\pi'_q}^{\phi^*(\pi'')} - \delta_{\pi''_{q+1}}^{\phi^*(\pi'')}) \right] > 0. \quad (4–14)$$

Because $\pi'$ and $\pi''$ are identical except for the $j^{th}$ and $(j + 1)^{th}$ jobs, (4–14) reduces to the following:

$$Z_{\pi'} - Z_{\pi''} = (p_{\pi'_j} - p_{\pi'_{j+1}}) + \sum_{q=1}^{n} (n - q + 1)(\delta_{\pi'_q}^{\phi^*(\pi'')} - \delta_{\pi''_{q+1}}^{\phi^*(\pi'')}) > 0.$$
Hence, because \( p_{n_j'} > p_{n_{j+1}'} \), it suffices to show that

\[
\sum_{q=1}^{n} (n - q + 1)(\delta_{n_q}^* - \delta_{n_q'}^*) \geq 0.
\] (4–15)

First consider US1, where Lemma 1 guarantees that the jobs are greedily delayed in the order \( \pi_1, \ldots, \pi_n \). As a result, \( \delta_{n_q}^* = \delta_{n_q'}^*, \forall q \in J \), where \( J = \{1, \ldots, n\} \setminus \{j, j + 1\} \). Therefore, it suffices to show that

\[(n - j + 1)(\delta_{n_j}^* - \delta_{n_{j+1}}^*) + (n - j)(\delta_{n_{j+1}}^* - \delta_{n_{j+1}'}^*) \geq 0.\] (4–16)

Next, note that \( \delta_{n_j}^* + \delta_{n_{j+1}}^* = \delta_{n_{j+1}}^* + \delta_{n_{j+1}'}^* \), because \( \delta_{n_q}^* = \delta_{n_q'}^* \) for all \( q \in J \), and \( \sum_{q=1}^{n} \delta_{n_q}^* = \sum_{q=1}^{n} \delta_{n_q'}^* \). Therefore, the left-hand-side of (4–16) reduces to \( \delta_{n_j}^* - \delta_{n_{j+1}'}^* \). Also, we have that \( \delta_{n_j}^* \geq \delta_{n_{j+1}'}^* \), because the largest delay for each job is a proportion of its processing time and \( p_{n_j'} > p_{n_{j+1}'} \). Thus, (4–16) holds true.

For US2 and US3, Lemmas 2 and 3 guarantee that the worst-case scenario delays jobs by their largest possible amount, in nonincreasing order of their \( \bar{R} \)-values. We focus on US3 here, with analysis for US2 following as a direct result. By Lemma 3, the condition given by (4–15) is equivalent to the following:

\[
\sum_{i=1}^{n} \bar{R}_i' k_{n_i'} - \sum_{i=1}^{n} \bar{R}_i'' k_{n_i''} \geq 0.
\] (4–17)

To prove the theorem for these uncertainty sets, we first establish the following facts.

**Fact 1.** \( \bar{R}_q' = \bar{R}_q'' \), \( \forall q \in J \), because \( \pi_q' = \pi_q'' \), \( \forall q \in J \).

**Fact 2.** \( \bar{R}_{j+1}' < \min\{\bar{R}_j', \bar{R}_j''\} \), and \( \max\{\bar{R}_j'', \bar{R}_j''+1\} < \bar{R}_j' \). Recalling that \( \bar{R}_j' = (n - j + 1) Kp_{n_j} \), \( \bar{R}_{j+1}' = (n - j) Kp_{n_{j+1}} \), \( \bar{R}_j'' = (n - j + 1) Kp_{n_j} \), and \( \bar{R}_j''+1 = (n - j) Kp_{n_j} \), the result holds because \( p_{n_j'} > p_{n_{j+1}'} \).

**Fact 3.** \( \bar{R}_j' + \bar{R}_{j+1}' - \bar{R}_j'' - \bar{R}_j''+1 > 0 \), because by substitution, we have \( \bar{R}_j' + \bar{R}_{j+1}' - \bar{R}_j'' - \bar{R}_j''+1 = K(p_{n_j'} - p_{n_{j+1}'}) > 0 \) (since \( p_{n_j'} > p_{n_{j+1}'} \)).
Let $\mathcal{D}' (\mathcal{D}'')$ be the set of job positions that are delayed in schedule $\pi' (\pi'')$, and define $l' \in \arg\min_{i \in \mathcal{D}'} \{\bar{R}_i\}$ ($l'' \in \arg\min_{i \in \mathcal{D}''} \{\bar{R}_i\}$). We can then establish the following fact.

**Fact 4.** In an optimal solution, $k_{\pi_i'} = K, \forall i \in \mathcal{D}' \setminus \{l'\}$ ($k_{\pi_i''} = K, \forall i \in \mathcal{D}'' \setminus \{l''\}$), $k_{\pi_i'} = 0, \forall i \in \{1, \ldots, n\} \setminus \mathcal{D}' (k_{\pi_i''} = 0, \forall i \in \{1, \ldots, n\} \setminus \mathcal{D}'')$, and $k_{\pi_i'} = k_{\pi_i''} = k$, for some $0 < k \leq K$.

We next explore different cases that may occur depending on whether or not the jobs in positions $j$ and $j + 1$ are delayed in each sequence, and then form subcases based on where $l'$ and $l''$ occur in the sequences. We then prove that (4–17) holds true for each case using Facts 1–4. For brevity in the following proofs, we refer to the left-hand-side of (4–17) as LHS.

- **Case 1.** $j \notin \mathcal{D}'$
  Due to Fact 2, $\bar{R}_j$ is larger than $\bar{R}_{j+1}$, $\bar{R}_j''$, and $\bar{R}_{j+1}''$. Therefore, $j + 1 \notin \mathcal{D}'$, and so $\mathcal{D}' \subseteq \tilde{J}$. By Fact 1, $\bar{R}_q' = \bar{R}_q''$, for each $q \in \mathcal{D}'$, implying that $\bar{R}_q'' > \max\{\bar{R}_q', \bar{R}_{q+1}''\}, \forall q \in \mathcal{D}'$. Hence, $j \notin \mathcal{D}''$ and $j + 1 \notin \mathcal{D}''$, and by Fact 4, LHS = 0.

- **Case 2.** $j \in \mathcal{D}'$, $j + 1 \notin \mathcal{D}'$, $j \notin \mathcal{D}'', j + 1 \notin \mathcal{D}''$
  For this case, note that because $\bar{R}_q' = \bar{R}_q'', \forall q \in \tilde{J}$, we have that $\mathcal{D}'$ differs from $\mathcal{D}''$ by only one element; in particular, $j \in \mathcal{D}', j \notin \mathcal{D}'', l' \notin \mathcal{D}'$, and $l'' \in \mathcal{D}''$.

  - **Case 2-1.** $l' = j$
    According to Facts 1 and 4, LHS = $k\bar{R}_j' - k\bar{R}_j''$. Note that $l'' \in \tilde{J}$, and so $\bar{R}_j'' = \bar{R}_j''$ (by Fact 1). Because $j \in \mathcal{D}'$ and $l'' \notin \mathcal{D}'$, we have $\bar{R}_j' \geq \bar{R}_j''$, and so LHS $\geq 0$.

  - **Case 2-2.** $l' \neq j$
    LHS = $K\bar{R}_j' + k\bar{R}_j'' - K\bar{R}_j'' - k\bar{R}_j''$. Since $\bar{R}_j'' = \bar{R}_j''$ (by Fact 1), we have LHS = $K\bar{R}_j' - (K - k)\bar{R}_j'' - k\bar{R}_j'' = (K - k)(\bar{R}_j' - \bar{R}_j'') + k(\bar{R}_j' - \bar{R}_j'')$, which is nonnegative since $K \geq k$, $\bar{R}_j' \geq \bar{R}_j''$, and $\bar{R}_j' \geq \bar{R}_j''$, by the same reasoning presented in Case 2-1.

- **Case 3.** $j \in \mathcal{D}', j + 1 \notin \mathcal{D}'$, $j \notin \mathcal{D}'', j + 1 \notin \mathcal{D}''$
  - **Case 3-1.** $l' = j, l'' = j$
    LHS = $k\bar{R}_j' - k\bar{R}_j''$, which is positive by Fact 2.
  - **Case 3-2.** $l' \neq j, l'' = j$
LHS = $K\bar{R}'_j + k\bar{R}''_j - K\bar{R}'''_j - k\bar{R}'''_j$. Since $\bar{R}'_j = \bar{R}'''_j$ (by Fact 1), LHS simplifies to $K\bar{R}'_j - (K-k)\bar{R}'''_j - k\bar{R}'''_j$, which is positive since $\bar{R}'_j > \bar{R}''_j$ by Fact 2, and $\bar{R}'_j \geq \bar{R}'''_j$, or else $l'$ would not be the lowest-priority job to be delayed in $D'$.  

- **Case 3-3.** $l'' \neq j$   
  Facts 1 and 2 imply that $\bar{R}'_j > \bar{R}'''_j$. Hence, if $l'' \neq j$, then $l'' \neq j$ as well. In this case, we have $l' = l''$ by Lemma 3 and Fact 1. Therefore, LHS = $K\bar{R}'_j - K\bar{R}'''_j > 0$.

- **Case 4.** $j \in D', j + 1 \notin D', j \notin D''$, $j + 1 \in D''$.  
The proof for this case is symmetric to that for case 3.

- **Case 5.** $j \in D', j + 1 \notin D', j \in D'', j + 1 \in D''$.  
  Note that for this case, we have $l' \neq j$, or else, by Lemma 3 it would be impossible to delay both $\pi''_j$ and $\pi''_{j+1}$ in $\pi''$ as assumed in this case.

  - **Case 5-1.** $l'' = j$  
    LHS = $K\bar{R}'_j + k\bar{R}''_j - K\bar{R}'''_{j+1} - k\bar{R}'''_{j+1}$. Because $\bar{R}'_j \geq \bar{R}'''_{j+1}$ (or else we would have delayed $\pi'_{j+1}$ instead of $\pi_j$), we conclude that LHS $\geq K\bar{R}'_j + k\bar{R}''_j - K\bar{R}'''_{j+1} - k\bar{R}'''_{j+1}$. Since $\bar{R}'''_{j+1} < \bar{R}'_j$ (by Fact 2), we obtain LHS $\geq K(\bar{R}'_j + \bar{R}''_j - \bar{R}''_{j+1} - \bar{R}''''_{j+1})$, which is positive by Fact 3.

  - **Case 5-2.** $l'' = j + 1$  
    Symmetric to Case 5-1 by exchanging $\bar{R}'_j$ and $\bar{R}'''_{j+1}$.

  - **Case 5-3.** $l'' \neq j$, $l'' \neq j + 1$  
    By Lemma 3, if the jobs in positions $j$ and $j + 1$ are delayed in $\pi''$, but only one of them is delayed in $\pi'$, then we must have that $l' \notin D''$ and $l'' \in D'$. Therefore, $l' \neq l''$ in this case, and both $l'$ and $l''$ belong to $D'$. Hence, LHS = $K\bar{R}'_j + K\bar{R}''_j + k\bar{R}''''_j - K\bar{R}''''_{j+1} - k\bar{R}''''_{j+1}$. Since $\bar{R}''''_j = \bar{R}''''_{j+1}$, we can write LHS as $K\bar{R}'_j + (K-k)\bar{R}''_j + k\bar{R}''''_j - K\bar{R}''''_{j+1} - k\bar{R}''''_{j+1}$; then, since $\bar{R}''''_j \geq \bar{R}''''_{j+1}$ (noting that if $\bar{R}''''_j < \bar{R}''''_{j+1}$, we would have delayed $\pi''_j$ instead of $\pi''''_j$), and if $\bar{R}''''_j < \bar{R}''''_{j+1}$, we would have delayed $\pi''''_{j+1}$ instead of $\pi''''_j$, the second and third terms of LHS are bounded as $(K-k)\bar{R}''''_j + k\bar{R}''''_j \geq (K-k)\bar{R}''''_{j+1} + k\bar{R}''''_{j+1} = K\bar{R}''''_{j+1}$. Thus, we conclude that LHS $\geq K(\bar{R}'_j + \bar{R}''_j - \bar{R}''''_{j+1} - \bar{R}''''_{j+1})$, which is positive by Fact 3.

- **Case 6.** $j + 1 \in D'$  
  Note that for this case, Fact 2 implies that $j \in D', j \in D''$, and $j + 1 \in D''$. This fact also establishes that $l' \neq j$, and furthermore, if $l' \neq j + 1$, then $l'' \neq j$ and $l'' \neq j + 1$.

  - **Case 6-1.** $l' = j + 1$, $l'' = j$  
    LHS = $K\bar{R}'_j + k\bar{R}''_{j+1} - K\bar{R}''''_{j+1} - k\bar{R}''''_{j+1}$. Because $\bar{R}''_{j+1} < \bar{R}''''_j$ by Fact 2, we conclude that LHS $\geq K(\bar{R}'_j + \bar{R}''_{j+1} - \bar{R}''''_{j+1} - \bar{R}''''_{j+1})$, which is positive by Fact 3.

  - **Case 6-2.** $l' = j + 1$, $l'' = j + 1$  
    Symmetric to Case 6-1 by exchanging $\bar{R}''''_j$ and $\bar{R}''''_{j+1}$.

  - **Case 6-3.** $l' \neq j + 1$ (which implies that $l'' \neq j$ and $l'' \neq j + 1$)
LHS = \( K \tilde{R}_j^j + K \tilde{R}_{j+1}^j - K \tilde{R}_{j+1}^j - K \tilde{R}_{j+1}^{j+1} \), which is positive by Fact 3.

Theorem 4.2 implies that one can solve the robust optimization problem under all three uncertainty sets by sorting the jobs by their processing times, which implies the worst-case complexity of \( \mathcal{O}(n \log n) \) for the problem.

### 4.3.2 Minimizing Total Weighted Completion Time

In this section, we consider the problem \( \text{MinMax}(\sum w_j C_j, p_j) \). The deterministic version of this problem can be solved by sequencing the jobs in nondecreasing order of the ratio \( p_j/w_j \), which forms a weighted shortest processing time (WSPT) order [73]. However, we will show in this section that the WSPT rule does not always create robust optimal schedules in the presence of uncertainty.

#### 4.3.2.1 Scenario Generation Problem

We first show that the corresponding SGP can be mathematically stated by (4–1)–(4–3), where both \( R(\delta_{1}, \ldots, \delta_{n}) \) and \( F(\delta_{1}, \ldots, \delta_{n}) \) can be expressed as the sum of separable functions.

**Theorem 4.3.** The total objective value increase \( R(\delta_{1}, \ldots, \delta_{n}) = \sum_{j=1}^{n} R_j(\delta_{1}) \), where \( R_j(\delta_{1}) = (\sum_{a=j}^{n} w_{a}) \delta_{1}, \forall j = 1, \ldots, n \).

**Proof.** Similar to the proof of Theorem 4.1.

As discussed before, \( F(\delta_{1}, \ldots, \delta_{n}) \) can be restated as \( \sum_{j=1}^{n} F_j(\delta_{1}) \) according to the uncertainty set. Note that the functions \( F_j(\delta_{1}), \forall j = 1, \ldots, n \), do not depend on the optimization criteria and therefore, the feasible region of the problem for US1, US2, and US3 are the same as the ones presented in formulations (4–4)–(4–6), (4–7)–(4–10), and (4–11)–(4–13), respectively. We define the objective function of the three problems according to Theorem 4.3 (where we substitute \( \delta_{1} \) by \( p_{1}/k_{1} \) for the case of US3 in (4–20) as we did in (4–11)). Therefore, the SGP can be stated as (4–18), (4–19), and
(4–20) for US1, US2, and US3, respectively.

\[
\text{Max } \sum_{j=1}^{n} \left( \sum_{q=j}^{n} w_{\pi_q} \right) \delta_{\pi_j}, \quad \text{subject to constraints } (4–5) \text{ and } (4–6) \quad (4–18)
\]

\[
\text{Max } \sum_{j=1}^{n} \left( \sum_{q=j}^{n} w_{\pi_q} \right) \delta_{\pi_j}, \quad \text{subject to constraints } (4–8)–(4–10) \quad (4–19)
\]

\[
\text{Max } \sum_{j=1}^{n} \left( \sum_{q=j}^{n} w_{\pi_q} \right) p_{\pi_j} k_{\pi_j}, \quad \text{subject to constraints } (4–12) \text{ and } (4–13) \quad (4–20)
\]

Adjusting our definition of \( \bar{R}_j \) as \( \bar{R}_j = \left( \sum_{q=j}^{n} w_{\pi_q} \right) K p_{\pi_j}, \forall j = 1, \ldots, n \), similar proofs demonstrate that Lemmas 1–3 hold for the problem of minimizing weighted completion time. Therefore, the complexity of solving SGP under each uncertainty set is similar to the ones presented in Section 4.3.1.1.

4.3.2.2 Robust Optimization Problem

Although solving the SGP is easy, creating a robust optimal sequence for this problem is not straightforward. Example 4.1 in Section 4.2 demonstrates a case in which the WSPT rule does not provide a robust optimal solution for the problem of minimizing total weighted completion time. Because \( p_1/w_1 < p_2/w_2 \) in this example, a WSPT sequence would schedule job 1 before job 2, which is suboptimal in all three uncertainty sets.

To further explore the problem of finding a robust optimal sequence in this case, we present a min-max mathematical programming formulation for the problem. We first define the following decision variables.

\[
x_{jq} = \begin{cases} 
1, & \text{if job } j \text{ is scheduled as the } q^{th} \text{ job in the sequence} \\
0, & \text{otherwise,}
\end{cases} \quad \forall j \in J, \; q = 1, \ldots, n
\]

\[
l_{ij} = \begin{cases} 
1, & \text{if job } j \text{ is scheduled after job } i \\
0, & \text{otherwise,}
\end{cases} \quad \forall i, j \in J
\]

\( \tilde{C}_j \) = completion time of job \( j \) where all processing times take on their ideal values, \( \forall j \in J \)
The objective function \((4–21)\) models the deterministic value of total weighted completion time plus the objective increase due to delayed jobs. Constraints \((4–22)\) and \((4–23)\) ensure that a unique position in the sequence is assigned to each job. Constraints \((4–24)\) force \(l_{ij}\) to equal one when \(j\) is scheduled after \(i\). Finally, Constraints \((4–25)\) calculate the completion time of each job. The remaining constraints define the type of decision variables in the model. Note that \(l_{ij}\) always tends to take on its smallest possible
value, which is either 0 or 1 according to Constraints (4–24). Therefore, we can relax the assumption of $l_{ij}$ being binary.

Next, we define $\theta(l)$ for each uncertainty set by adjusting models (4–18), (4–19), and (4–20) using the decision variables of the model presented in (4–21)–(4–27). Note that the terms $\delta_j$ in (4–18) and (4–19), and $p_jk_j$ in (4–20), are equivalent to $Kp_jy_j$. By substituting for $y$, we obtain the following three formulations for the SGP corresponding to each uncertainty set.

For US1, we have:

$$\theta(l) = \max \sum_{j \in J} \left( \left( w_j + \sum_{i \in J} w_i l_{ij} \right) Kp_j \right) y_j$$

subject to:

$$\sum_{j \in J} Kp_j y_j \leq \Delta$$

$$0 \leq y_j \leq 1, \quad \forall j \in J.$$  \hspace{1cm} (4–28)

For US2, the model is defined as follows:

$$\theta(l) = \max \sum_{j \in J} \left( \left( w_j + \sum_{i \in J} w_i l_{ij} \right) Kp_j \right) y_j$$

subject to:

$$\sum_{j \in J} m_j \leq M$$

$$0 \leq y_j \leq m_j, \quad \forall j \in J$$

$$m_j \in \{0, 1\}, \quad \forall j \in J.$$  \hspace{1cm} (4–30)

Finally, for US3, we define:

$$\theta(l) = \max \sum_{j \in J} \left( \left( w_j + \sum_{i \in J} w_i l_{ij} \right) Kp_j \right) y_j$$

subject to:

$$\sum_{j \in J} m_j \leq M$$

$$0 \leq y_j \leq m_j, \quad \forall j \in J$$

$$m_j \in \{0, 1\}, \quad \forall j \in J.$$  \hspace{1cm} (4–35)
\[
\sum_{j \in J} Ky_j \leq \kappa
\]  \quad (4–36)

\[
0 \leq y_j \leq 1, \quad \forall j \in J.
\]  \quad (4–37)

Next, we will show how to convert the model presented in (4–21)–(4–27) to an MIP. For US1 and US3, note that because (4–28)–(4–30) and (4–35)–(4–37) are linear programs, we can replace \( \theta(I) \) by the optimal objective function to their dual formulations (due to the strong duality theorem). For US2, though, the formulation presented in (4–31)–(4–34) is an MIP. Theorem 4.4 presents a linear program that is equivalent to (4–31)–(4–34), thus allowing us to employ the strong duality theorem to formulate (4–21)–(4–27) as an MIP for US2.

**Theorem 4.4.** The optimal value of the following linear program equals the optimal value of the problem formulated in (4–31)–(4–34).

\[
\theta(I) = \text{Max} \sum_{j \in J} \left( w_j + \sum_{i \in J} w_{ji} l_{ji} \right) Kp_j y_j
\]  \quad (4–38)

subject to:

\[
\sum_{j \in J} y_j \leq M
\]  \quad (4–39)

\[
0 \leq y_j \leq 1, \quad \forall j \in J.
\]  \quad (4–40)

**Proof.** The coefficient matrix defining the constraint set for the problem formulated in (4–38)–(4–40) is totally unimodular, which thus implies that an optimal solution to (4–38)–(4–40) must exist in which all \( y \)-variables are binary valued (noting that the feasible region of the problem presented in (4–38)–(4–40) is nonempty and bounded). Also, because the \( m \)-variables only appear in (4–32) and (4–33), the structure of those constraints guarantees that an optimal solution exists in which \( m_j = 1 \) only if \( y_j = 1, \forall j \in J \). This fact, combined with (4–33), ensures that \( y_j = m_j \). Substituting out the \( m \)-values, the formulations presented in (4–31)–(4–34) and in (4–38)–(4–40) become identical.  \[\square\]
Next, we present a robust MIP formulation corresponding to each uncertainty set.

Define $u$ as the dual variable associated with the weight constraint (Constraint (4–29), (4–39), and (4–36) for US1, US2, and US3, respectively), and $v_j$ as the dual variable associated with bounding constraints corresponding to $j \in J$ (Constraints (4–30), (4–40), and (4–37) for US1, US2, and US3, respectively).

The MIP formulation for US1 is presented below.

$$\begin{align*}
\text{Min} & \sum_{j \in J} w_j \tilde{c}_j + \Delta u + \sum_{j \in J} v_j & (4–41) \\
\text{subject to:} & \\
\text{Constraints (4–22)–(4–27),} & \\
Kp_j u + v_j & \geq \left( w_j + \sum_{i \in J} w_i l_{ji} \right) Kp_j, \quad \forall j \in J & (4–42) \\
u & \geq 0, \quad v_j & \geq 0, \quad \forall j \in J. & (4–43)
\end{align*}$$

For US2, the model is presented below.

$$\begin{align*}
\text{Min} & \sum_{j \in J} w_j \tilde{c}_j + Mu + \sum_{j \in J} v_j & (4–44) \\
\text{subject to:} & \\
\text{Constraints (4–22)–(4–27),} & \\
u + v_j & \geq \left( w_j + \sum_{i \in J} w_i l_{ji} \right) Kp_j, \quad \forall j \in J & (4–45) \\
u & \geq 0, \quad v_j & \geq 0, \quad \forall j \in J. & (4–46)
\end{align*}$$

Finally, for US3, we have the following MIP formulation.

$$\begin{align*}
\text{Min} & \sum_{j \in J} w_j \tilde{c}_j + \kappa u + \sum_{j \in J} v_j & (4–47) \\
\text{subject to:} & \\
\text{Constraints (4–22)–(4–27),} &
\end{align*}$$
\[ Ku + v_j \geq \left( w_j + \sum_{i \in J} w_i l_{ji} \right) K p_j, \quad \forall j \in J \]  
\[ u \geq 0, \quad v_j \geq 0, \quad \forall j \in J. \]

Linearity of formulations presented in (4–41)–(4–43), (4–44)–(4–46), and (4–47)–(4–49) implies that we can solve each problem using a standard MIP solver.

### 4.3.3 Minimizing Maximum Lateness/Tardiness

When minimizing the maximum lateness or tardiness among all jobs in a schedule, given deterministic data, the problem can be solved optimally by sequencing the jobs in nondecreasing order of their due dates \((d_j)\). These schedules are known as earliest due date (EDD) schedules, due to Lawler \([50]\). We show in this section that EDD schedules remain optimal under all three uncertainty sets.

#### 4.3.3.1 Scenario Generation Problem

We begin by stating a simple \(O(n)\) algorithm for solving SGP under US1.

**Theorem 4.5.** For US1, delaying jobs by their largest possible values, in the order that they appear in \(\pi\), solves the SGP corresponding to \(\pi\).

**Proof.** We prove that the strategy described in Theorem 4.5 maximizes the completion time for each job in the sequence, and therefore maximizes the largest lateness/tardiness occurring in the sequence. By contradiction, suppose that in the worst-case scenario \(\phi^*\) corresponding to a sequence \(\pi\), the \(j^{th}\) job is delayed \((p_{\pi_j}^{\phi^*} \geq p_{\pi_j} + \epsilon, \text{for some } \epsilon > 0)\) while there is a job in position \(i < j\) that is not delayed by its maximum possible value \((p_{\pi_i}^{\phi^*} \leq (1 + K)p_{\pi_i} - \epsilon)\). Note that decreasing \(p_{\pi_j}^{\phi^*}\) and increasing \(p_{\pi_i}^{\phi^*}\) by \(\epsilon\) is a feasible action, according to both the largest delay limitation for each job and uncertainty budget constraint, and increases the completion time of jobs in positions \(i, \ldots, j - 1\) without changing \(C_{\pi_1}^{\phi^*}, \ldots, C_{\pi_{i-1}}^{\phi^*}\) and \(C_{\pi_i}^{\phi^*}, \ldots, C_{\pi_{j-1}}^{\phi^*}\). As a result, the completion time of every job in the modified schedule is not smaller than its completion time in scenario \(\phi^*\). This completes the proof. \(\square\)
For US2, an optimal solution to the SGP can be obtained using the following polynomial algorithm.

For each job position \( j = 1, \ldots, n \) in \( \pi \), define \( G_j \) as follows. If \( j \leq M \), then \( G_j = \{ \pi_1, \ldots, \pi_j \} \), and if \( j > M \), then \( G_j \) contains \( M \) jobs having the longest processing times among all jobs in positions \( 1, \ldots, j \). (That is, for the case in which \( j > M \) we have that \( \pi_i \in G_j \) for \( 1 \leq i \leq j \) implies that \( p_{\pi_i} \geq p_{\pi_q} \) for every \( q \) such that \( 1 \leq q \leq j \) and \( \pi_q \notin G_j \).) Recall that \( \tilde{C}_{\pi_j} \) is the ideal completion time of the \( j^{\text{th}} \) job in \( \pi \) (when all jobs take on their ideal processing time). Define \( R'_j = K \sum_{\pi_i \in G_j} p_{\pi_i} + \tilde{C}_{\pi_j} - d_{\pi_j} \) as the maximum lateness for job \( \pi_j \) in schedule \( \pi \); this lateness is achieved by delaying all jobs in \( G_j \) by their largest possible amount. Hence, enumerating the maximum \( R'_j \)-value over all \( j = 1, \ldots, n \) identifies the worst-case objective function value corresponding to \( \pi \). Letting \( q \in \arg \max_{j \in \{1, \ldots, n\}} \{ R'_j \} \), the worst-case objective is \( R'_q \). An identical analysis holds for the case of maximum tardiness by setting \( R'_j = \max \{0, K \sum_{\pi_i \in G_j} p_{\pi_i} + \tilde{C}_{\pi_j} - d_{\pi_j} \} \).

For US3, we modify the approach used for US2. Define \( G_j, \forall j = 1, \ldots, n \), as before, except that this set now contains the \( \min \{ \lceil \kappa/K \rceil, j \} \) longest-processing-time jobs in \( \{ \pi_1, \ldots, \pi_j \} \). Also, denote \( \Theta \) as a job with the shortest processing time in \( G_j \) (\( \Theta \in \arg \min_{\pi_i \in G_j} \{ p_{\pi_i} \} \)). For each \( j = 1, \ldots, n \), compute

\[
R'_j = K \sum_{\pi_i \in G_j - \{ \Theta \}} p_{\pi_i} + (\kappa - K\lceil \kappa/K \rceil)p_{\Theta} + \tilde{C}_{\pi_j} - d_{\pi_j}
\]

for the maximum lateness case, and

\[
R'_j = \max \left\{ 0, K \sum_{\pi_i \in G_j - \{ \Theta \}} p_{\pi_i} + (\kappa - K\lceil \kappa/K \rceil)p_{\Theta} + \tilde{C}_{\pi_j} - d_{\pi_j} \right\}
\]

for the maximum tardiness case. By the same logic given for the case of US2, the worst-case objective is given by \( R'_q \) for some \( q \in \arg \max_{j \in \{1, \ldots, n\}} \{ R'_j \} \). This result is achieved by delaying jobs \( \pi_i \) by \( Kp_{\pi_i}, \forall \pi_i \in G_q - \{ \Theta \} \), and delaying job \( \Theta \) by \( (\kappa - K\lceil \kappa/K \rceil)p_{\Theta} \).
We now show that the complexity of the algorithm is $O(n \log M)$ for US2 and $O(n \log(\lceil \kappa/K \rceil))$ for US3. The main operation in the algorithm is forming the sets $G_j$, $\forall j = 1, \ldots, n$, and calculating the sum of processing times in each set. We store each set $G_j$ as a (sorted) binary tree and recursively calculate it using $G_{j-1}$. For US2, we create $G_j$ from $G_{j-1}$ by inserting job $\pi_j$ in the sorted binary tree $G_{j-1}$ ($O(\log M)$ operations) and add its processing time to $\sum_{\pi_i \in G_j} p_{\pi_i}$ ($O(1)$ operations). When $j > M$, we also remove a job having the shortest processing time in $G_j$ (to keep the cardinality of $G_j$ equal to $M$). This requires $O(1)$ operations to locate and remove this job, and to subtract its processing time from $\sum_{\pi_i \in G_j} p_{\pi_i}$. Finding the maximum lateness/tardiness requires $O(n)$ operations in a postprocessing step. Thus, the total complexity of the algorithm for US2 is $O(n \log M)$. A similar discussion for US3 establishes a complexity of $O(n \log(\lceil \kappa/K \rceil))$ for the algorithm.

### 4.3.3.2 Robust Optimization Problem

We now show that EDD scheduling results in an optimal algorithm for the robust optimization problem discussed in this subsection, under all three uncertainty sets.

**Theorem 4.6.** A schedule formed by the EDD rule is optimal for the robust optimization problem of minimizing maximum lateness/tardiness in SMSP, under all three uncertainty sets.

**Proof.** Consider an EDD sequence $\pi$, and suppose that $\pi_j$ has the maximum lateness or tardiness in the worst-case scenario of sequence $\pi$ ($L_{\max} = L_{\pi_j} = C_{\pi_j} - d_{\pi_j}$ or $T_{\max} = T_{\pi_j} = \max\{0, C_{\pi_j}^* - d_{\pi_j}\}$). Suppose that $\pi$ is not optimal, and that $\pi'$ is an optimal sequence. First, note that at least one job appearing before $\pi_j$ in schedule $\pi$ must appear after $\pi_j$ in $\pi'$ (or else, applying the same delays to jobs in schedule $\pi'$ as applied in the worst case for schedule $\pi$ yields at least as much lateness (or tardiness) for job $\pi_j$ in schedule $\pi'$). Let job $\pi_q'$ be the latest scheduled job in $\pi'$ among those jobs scheduled before $\pi_j$ in $\pi$. In schedule $\pi'$, job $\pi_j$, along with all jobs scheduled prior to $\pi_j$ in $\pi$, are scheduled prior to $\pi_q'$ in $\pi'$. Therefore, using the same delays in $\pi'$ as in
\( \pi, \) job \( \pi'_q \) completes in \( \pi' \) after job \( \pi_j \) completes in \( \pi. \) Moreover, \( d_{\pi'_q} \leq d_{\pi_j} \) since \( \pi \) is an EDD sequence and \( \pi'_q \) is scheduled before \( \pi_j \) in \( \pi. \) As a result, the value of \( L_{\max} \) \((T_{\max})\) corresponding to \( \pi' \) in the worst-case scenario is at least as large as the one corresponding to \( \pi. \) This contradicts the assumption that \( \pi \) is suboptimal (if \( \pi' \) is optimal) and completes the proof. \( \square \)

Theorem 4.6 implies that solving the robust optimization problem, under all three uncertainty sets, requires \( O(n \log n) \) operations.

4.3.4 Minimizing Number of Late Jobs

Minimizing the number of late jobs with deterministic data is polynomially solvable by Moore’s algorithm [66], which works as follows. We first schedule the jobs in EDD order to form \( \pi. \) Then, we investigate the jobs in the order that they appear in \( \pi \) until we find the first late job, say \( \pi_j. \) We then remove (or “reject”) a job in \( \{\pi_1, \ldots, \pi_j\} \) having the longest processing time, and continue to the next late job until all jobs have been scheduled in \( \pi \) or removed. We then schedule any removed jobs at the end of \( \pi, \) in any arbitrary order, to form an optimal solution.

This algorithm must be modified in the presence of uncertainty to yield an optimal schedule. For example, consider a two-job instance where \( p_1 = p_2 = 1, d_1 = 1, \) and \( d_2 = 2. \) Let \( K = 1, \) and let uncertainty budgets be given by \( \Delta = M = K = 1 \) for the three uncertainty models. Scheduling job 1 before job 2 produces no late jobs assuming deterministic data. However, this sequence results in two late jobs in presence of uncertainty (for all three uncertainty sets) by delaying job 1. The reverse sequence (job 2 before job 1) yields only one late job in the worst case in all three uncertainty models.

4.3.4.1 Scenario Generation Problem

For US1, the SGP is solved by the same method as the one presented in Section 4.3.3.1 for US1 (delaying jobs by the largest possible amount in the order that they appear in \( \pi). \) As discussed in the proof of Theorem 4.5, this strategy results in the largest possible completion time for every job in the sequence. Accordingly, it
creates the maximum number of delayed jobs in the sequence. It then follows that the complexity of the SGP problem under US1 is $O(n)$.

For US2, we construct a dynamic-programming algorithm for the SGP. Define $f_j(l, r)$ as the maximum completion time of the $j^{th}$ job in $\pi$, when we delay $r$ of the first $j$ jobs by their largest possible value and create $l$ late jobs (among the first $j$ jobs) by this action. If it is impossible to create $l$ late jobs by delaying $r$ jobs (among the first $j$ jobs in $\pi$), then $f_j(l, r) = 0$. A worst-case scenario corresponds to the largest possible value of $l$ for which $f_n(l, M)$ is positive. We start by initializing $f_j(l, r) = 0$, $\forall j, l = 0, \ldots, n$, and $r = 0, \ldots, M$. To describe our recursion, we define an indicator function, $I$, such that $I{\cdot} = 1$ if $\cdot$ is true, and $I{\cdot} = 0$ otherwise. The following recursion considers four possible cases, one corresponding to each combination of whether or not $\pi_j$ will be late, and whether or not $\pi_j$ is delayed. For each $j, l = 1, \ldots, n$ and $r = 0, \ldots, M$, we have:

$$f_j(l, r) = \max \left\{ \begin{array}{l}
(f_{j-1}(l, r) + p_{\pi_j}) I\{f_{j-1}(l, r) + p_{\pi_j} \leq d_{\pi_j}\} \\
(f_{j-1}(l-1, r - 1) + p_{\pi_j}(1 + K)) I\{f_{j-1}(l-1, r - 1) + p_{\pi_j}(1 + K) \leq d_{\pi_j}\} \\
(f_{j-1}(l-1, r) + p_{\pi_j}) I\{f_{j-1}(l-1, r) + p_{\pi_j} > d_{\pi_j}\} \\
(f_{j-1}(l-1, r - 1) + p_{\pi_j}(1 + K)) I\{f_{j-1}(l-1, r - 1) + p_{\pi_j}(1 + K) > d_{\pi_j}\}.
\end{array} \right. \quad (4–50)$$

The first two cases correspond to the event in which $\pi_j$ is not late, and the last two cases correspond to the event in which $\pi_j$ is late. The first and third (second and fourth) cases correspond to the event in which $\pi_j$ is not (is) delayed. Observe that if one of the cases does not apply, the corresponding indicator function equals zero and the case is ignored in the computation of $f_j(l, r)$.

The algorithm starts by setting $f_0(0, 0) = 0$. The procedure computes $f_1(l, r)$ for all combinations of $l$ and $r$, then $f_2(l, r)$ for all combinations of $l$ and $r$, and so on, up to $f_n(l, r)$ for all combinations of $l$ and $r$. After identifying an optimal objective function value (largest $l$ for which $f_n(l, r) > 0$), the solution leading to this value can be found by backtracking. The complexity of this algorithm is $O(Mn^2)$, since $O(Mn^2)$ $f$-values must be computed, and (4–50) requires $O(1)$ effort for each $f$-value. The algorithm is illustrated by the following example.
Example 4.2. Consider a four-job sequence \( \pi = (1, 2, 3, 4) \) where

\[
\begin{align*}
    p_1 &= 4, & p_2 &= 6, & p_3 &= 2, & p_4 &= 10, \\
    d_1 &= 5, & d_2 &= 12, & d_3 &= 15, & d_4 &= 30, \\
    K &= 0.5, \text{ and } M &= 3.
\end{align*}
\]

We seek to identify job processing time values that maximize the number of late jobs in \( \pi \).

Figure 4-2 demonstrates the calculation of the \( f \)-values in (4–50), in a forward propagation manner. In this figure, if we encounter multiple candidates for some \( f_j(l, r) \), then only one node corresponding to largest values of \( f_j(l, r) \) is retained (deleted nodes are shaded gray). Also, there is no “delay 4” branch emerging from node “\( f_3(3, 3) = 18 \),” because three jobs have already been delayed at this node, which is the maximum number of delayed jobs. The worst-case scenario is given by delaying jobs 1, 2, and 4 by their largest possible value (\( p_1^{\ast} = 6, p_2^{\ast} = 9, p_3^{\ast} = 2, p_4^{\ast} = 15 \)), which results in four late jobs. The node containing \( f_n(l, r) \) having the largest value of \( l \), and the path from \( f_0(0, 0) = 0 \) to this node, are displayed using thick arrows.

Note that we cannot directly extend the proposed dynamic programming algorithm in order to solve SGP under US3: \( r \) is no longer integral in that case, and it is not clear how to obtain a finite state space over which our recursion takes place. Therefore, we leave open the question of whether SGP is polynomially solvable for this problem under US3.

As an alternative, we propose an MIP formulation for the corresponding SGP under US3. We first define the SGP decision variables corresponding to sequence \( \pi \).

\[
U_{\pi_j} = \begin{cases} 
1, & \text{if job } \pi_j \text{ is late} \\
0, & \text{otherwise},
\end{cases} \quad \forall j = 1, \ldots, n
\]

\[
C_{\pi_j} = \text{completion time of job } \pi_j, \forall j = 1, \ldots, n
\]
Figure 4-2. Dynamic-programming SGP network with $\sum_{j \in J} U_j$ criterion and US2

$k_{\pi_j} =$ the proportion of $p_{\pi_j}$ by which we delay job $\pi_j$, $\forall j = 1, \ldots, n$

Our MIP formulation for the SGP under US3 is given as follows.

Max $\sum_{j \in J} U_{\pi_j}$ \hspace{1cm} (4–51)

subject to:

$C_{\pi_j} = \sum_{i=1}^{j} (p_{\pi_i} (1 + k_{\pi_i}))$, \hspace{1cm} $\forall j = 1, \ldots, n$, \hspace{1cm} (4–52)

$U_{\pi_j} \leq 1 + \frac{C_{\pi_j} - d_{\pi_j} - \epsilon_j}{d_{\pi_j} + \epsilon_j + C_{\pi_j}}$, \hspace{1cm} $\forall j = 1, \ldots, n$, \hspace{1cm} (4–53)

$0 \leq k_{\pi_j} \leq K$, \hspace{1cm} $\forall j = 1, \ldots, n$, \hspace{1cm} (4–54)
\[ \sum_{j=1}^{n} k_{\pi_j} \leq \kappa \]  \hspace{1cm} (4–55)

\[ U_{\pi_j} \in \{0, 1\}, \quad \forall j = 1, \ldots, n, \]  \hspace{1cm} (4–56)

where \( \epsilon_j \) is the smallest value by which job \( \pi_j \) can be late. Note that we can assume, without loss of generality, that \( d_{\pi_j} + \epsilon_j - \tilde{C}_{\pi_j} > 0 \) (if \( d_{\pi_i} + \epsilon_i - \tilde{C}_{\pi_i} \leq 0 \) for some \( \pi_i \in J \), then \( \pi_i \) is late, regardless of the delay scenario. In that case, we fix \( U_{\pi_i} = 0 \) and remove Constraint (4–53) where \( j = i \).) It then follows that if job \( \pi_j \) is not late, we have \(-1 \leq (C_{\pi_j} - d_{\pi_j} - \epsilon_j)/(d_{\pi_j} + \epsilon_j - \tilde{C}_{\pi_j}) < 0 \) and so Constraint (4–53) forces \( U_{\pi_j} = 0 \). On the other hand, when \( \pi_j \) is late, the right-hand-side of Constraint (4–53) is greater than 1 and so \( U_{\pi_j} = 1 \) at optimality.

In practice, because the \( k \)-variables are continuous, it is necessary to use very small values for \( \epsilon \)-constants in this model. Practically speaking, one might set \( \epsilon_j \) as the smallest detectable value that causes a job to be late. For instance, if processing times are measured in minutes, and job \( \pi_j \) is not practically late until it is five minutes past due, then \( \epsilon_j = 5 \).

### 4.3.4.2 Robust Optimization Problem

We first present a modification of Moore’s algorithm for solving the problem of minimizing the number of late jobs in a SMSP under uncertainty. Then, in Theorem 4.7, we prove that the proposed algorithm generates an optimal robust sequence under US1.

**Modified Moore’s (MM) Algorithm.**

**Step 0.** Initialize \( \bar{\Delta} = \Delta \) as the remaining uncertainty budget. Let \( \bar{\pi} \) be an EDD schedule of jobs and \( \pi \) (initially empty) be the schedule that we construct using this algorithm. Also, let \( \mathcal{R} \) (initially empty) be the set of rejected jobs and \( r \) be the last rejected job. Define \( I_i \) to be the subset of jobs that complete before their deadlines when we use the MM algorithm to schedule jobs \( \{ \bar{\pi}_1, \ldots, \bar{\pi}_i \} \) (with \( I_0 = \emptyset \)). Initialize \( i = j = 1 \), where \( i \) is the job position currently under examination in \( \bar{\pi} \) and \( j \) is the job position being scheduled in \( \pi \).
Step 1. Tentatively schedule $\pi_i$ in the $j^{th}$ position of $\pi$ $(\pi_j = \pi_i)$ and set $p^{\phi^*}_{\pi_j} = p_{\pi_j} + \min\{\Delta, Kp_{\pi_j}\}$. Update the value of $\Delta$ to equal $\Delta - \min\{\Delta, Kp_{\pi_j}\}$. If $\pi_j$ is late in $\pi$, then go to Step 2; otherwise, go to Step 3.

Step 2. Adjust the schedule of jobs in $\pi$ as follows:

Step 2-1. Find $q \in \arg \max_{s \in \{1,...,j\}} \{p_{\pi_s}\}$, and choose $r = \pi_q$ to be the next rejected job. Add $\pi_q$ to $R$, update $\Delta$ to equal $\Delta + p^{\phi^*}_{\pi_q} - p_{\pi_q}$, and set $p^{\phi^*}_{\pi_q} = p_{\pi_q}$. Go to Step 2-2.

Step 2-2. If $q = j$, then go to Step 4; otherwise, go to Step 2-3.

Step 2-3. Set $\pi_q = \pi_{q+1}$. If $\Delta > 0$ and $p^{\phi^*}_{\pi_q} < (1 + K)p_{\pi_q}$, then go to Step 2-4; otherwise, go to Step 2-5.

Step 2-4. Update the value of $p^{\phi^*}_{\pi_q}$ to equal $p^{\phi^*}_{\pi_q} + \min\{\Delta, (1 + K)p_{\pi_q} - p^{\phi^*}_{\pi_q}\}$ and update the value of $\Delta$ to equal $\Delta - \min\{\Delta, (1 + K)p_{\pi_q} - p^{\phi^*}_{\pi_q}\}$. Go to Step 2-5.

Step 2-5. Increment the value of $q$ by one and go to Step 2-2.

Step 3. Set $l_i = l_{i-1} \cup \{\pi_j\}$. Increment $i$ and $j$ by one and go to Step 5.

Step 4. Set $l_i = l_{i-1} \cup \{\pi_j\} \setminus \{r\}$. Increment $i$ by one and go to Step 5.

Step 5. If $i \leq n$, then go to Step 1; otherwise, schedule the jobs in $R$ in positions $j, ..., n$ of $\pi$, in any order, and terminate.

In order to prove the optimality of the MM algorithm, we first present some definitions and lemmas to facilitate the proof. Given any set of jobs $S$, we denote an EDD sequence of jobs in $S$ as $\text{EDD}(S)$ and define the length of $S$ (denoted by $C^*_\text{max}(S)$) as the worst-case makespan value of jobs in $S$ under US1 (i.e., $C^*_\text{max}(S) = \sum_{j \in S} p^*_j$). When all jobs in $S$ take on their ideal processing times, makespan is denoted by $\bar{C}^*_\text{max}(S) = \sum_{j \in S} p_j$.

**Lemma 4.** $C^*_\text{max}(S_1) \diamond C^*_\text{max}(S_2)$ if and only if $\bar{C}^*_\text{max}(S_1) \diamond \bar{C}^*_\text{max}(S_2)$, where $\diamond$ refers to any of the following operations: $<$, $\leq$, and $=$.

**Proof.** We prove the lemma for the case in which $\diamond$ is the $<$ operator, with all other cases established by a symmetric argument. For $i = 1, 2$, define $\Delta_{S_i}$ as the total delay...
of jobs in the worst-case scenario corresponding to an arbitrary sequence of jobs in \( S_i \).

Therefore, we have \( C^*_i(S_i) = \tilde{C}_i(S_i) + \tilde{\Delta}_S, \) for \( i = 1, 2 \). Note that the inequalities

\[ \delta_j \leq Kp_j, \forall j \in S_1 \cup S_2, \text{ and } \tilde{\Delta}_S \leq \Delta, \text{ for } i = 1, 2, \]

imply that \( \tilde{\Delta}_S = \min\{\Delta, K\tilde{C}_i(S_i)\} \), for \( i = 1, 2 \). We can therefore conclude that if \( \tilde{C}_i(S_1) < \tilde{C}_i(S_2) \), we have \( \tilde{\Delta}_S_1 \leq \tilde{\Delta}_S_2 \) and therefore \( C^*_i(S_1) < C^*_i(S_2) \). By the same argument, if \( \tilde{C}_i(S_1) \geq \tilde{C}_i(S_2) \), then \( \tilde{\Delta}_S_1 \geq \tilde{\Delta}_S_2 \) as well, which implies that \( C^*_i(S_1) \geq C^*_i(S_2) \). This completes the proof. \( \square \)

**Lemma 5.** If job \( q \) appears in two sequences, \( \pi^1 \) and \( \pi^2 \), with \( C^*_q(\pi^1) \leq C^*_q(\pi^2) \), then \( p^*_q(\pi^1) \geq p^*_q(\pi^2) \).

**Proof.** For each \( i = 1, 2 \), let \( S_i \) be the set of jobs that are scheduled before job \( q \) in \( \pi^i \), and define \( \tilde{\Delta}_S \) as presented in the proof of Lemma 4. Note that \( C^*_q(\pi^i) = C^*_i(S_i \cup \{q\}) \), for \( i = 1, 2 \), and so, Lemma 4 implies that \( \tilde{C}(S_1 \cup \{q\}) \leq \tilde{C}(S_2 \cup \{q\}) \). Subtracting \( p_q \) from both sides, we obtain \( \tilde{C}(S_1) \leq \tilde{C}(S_2) \). As established in the proof of Lemma 4, the latter inequality implies that \( \tilde{\Delta}_S_1 \leq \tilde{\Delta}_S_2 \). Also, recall from Section 4.3.4.1 that under US1, SGP delays jobs by the largest possible amount in the order that they appear in the sequence. We therefore conclude that \( \delta^*_q(\pi^i) = \min\{\Delta - \tilde{\Delta}_S, Kp_q\} \) for \( i = 1, 2 \). Thus, \( \delta^*_q(\pi^1) \geq \delta^*_q(\pi^2) \), which implies that \( p^*_q(\pi^1) \geq p^*_q(\pi^2) \) and completes the proof. \( \square \)

A subset of jobs is “feasible” if they can all be scheduled on time, even in the worst-case scenario. Given Lemmas 4 and 5, we now state the following theorem.

**Theorem 4.7.** The MM algorithm generates a schedule that minimizes the number of late jobs in the robust SMSP under US1.

**Proof.** We prove this theorem by adapting the proof used to show the optimality of Moore’s algorithm for the deterministic version of this problem. To prove Theorem 4.7, it is sufficient to prove the following condition: For each \( i = 1, \ldots, n \), among all maximum-cardinality feasible subsets of \( \{\tilde{\pi}_1, \ldots, \tilde{\pi}_i\} \), \( I_i \) is a subset having minimum length, i.e.,

(a) \( I_i \subseteq \{\tilde{\pi}_1, \ldots, \tilde{\pi}_i\} \) is feasible,
(b) \(|l_i| \geq |S|\), \(\forall\) feasible \(S \subseteq \{\bar{x}_1, ..., \bar{x}_i\}\), (4–57)

(c) \(C^*_\text{max}(l_i) \leq C^*_\text{max}(S)\), \(\forall\) feasible \(S \subseteq \{\bar{x}_1, ..., \bar{x}_i\}\) such that \(|S| = |l_i|\).

First, note that for a single job (\(\bar{x}_1\)), the MM algorithm generates \(l_1 = \{\bar{x}_1\}\) if \(\rho_{\bar{x}_1}^* \leq d_{\bar{x}_1}\) and \(l_1 = \emptyset\) otherwise. Hence (4–57) holds for \(i = 1\). For \(i \geq 2\), we prove that (4–57) holds by contradiction. Let \(q (\geq 2)\) be the smallest value of \(i\) for which (4–57) does not hold. Let \(D_q\) be a minimum-length feasible subset of \(\{\bar{x}_1, ..., \bar{x}_q\}\) among all maximum-cardinality feasible subsets of these jobs. Noting that \(l_q\) is feasible by construction, then one of the following two conditions holds if \(l_q\) does not satisfy (4–57):

**Case 1.** \(|l_q| < |D_q|\).

**Case 2.** \(|l_q| = |D_q|\), but \(C^*_\text{max}(l_q) > C^*_\text{max}(D_q)\).

To analyze these cases, we first observe that in every step of the MM algorithm, at most one more job will be added to the set \(I\), i.e., \(|I_{q-1}| \leq |l_q| \leq |I_{q-1}| + 1\). Furthermore, if \(|l_q| = |I_{q-1}| + 1\), then \(l_q = I_{q-1} \cup \bar{x}_q\). Consider Case 1 first. We must have that \(\bar{x}_q \in D_q\) and \(|D_q| = |l_q| + 1\). To see this, note that if \(\bar{x}_q \not\in D_q\), then \(D_q\) is a feasible subset of \(\{\bar{x}_1, ..., \bar{x}_{q-1}\}\), which contradicts the assumption that (4–57) holds for \(l_{q-1}\) since \(|D_q| > |l_q| \geq |l_{q-1}|\). Also, if \(|D_q| \geq |l_q| + 2\), then it follows that \(|D_q \setminus \{\bar{x}_q\}| > |l_{q-1}|\), which again contradicts the assumption that (4–57) holds for \(l_{q-1}\). Define \(D_{q-1} = D_q \setminus \{\bar{x}_q\}\), and note that \(D_{q-1}\) is a feasible subset with \(|l_{q-1}| = |D_{q-1}|\) (\(|l_{q-1}| < |D_{q-1}|\) is impossible since (4–57) holds for \(l_{q-1}\), whereas \(|l_{q-1}| > |D_{q-1}|\) is impossible because \(|D_q| = |D_{q-1}| + 1\) and \(|D_q| > |l_q| \geq |l_{q-1}|\)). Therefore, \(|D_q| = |l_{q-1}| + 1\) and so \(|l_q| = |l_{q-1}|\). By induction, \(C^*_\text{max}(l_{q-1}) \leq C^*_\text{max}(D_{q-1})\), which implies that \(C^*_\text{max}(l_{q-1}) \leq C^*_\text{max}(D_{q-1})\) by Lemma 4. Adding \(\rho_{\bar{x}_q}\) to both sides of the latter inequality yields \(\tilde{C}^*_\text{max}(l_{q-1} \cup \{\bar{x}_q\}) \leq \tilde{C}^*_\text{max}(D_q)\) and by Lemma 4, \(C^*_\text{max}(l_{q-1} \cup \{\bar{x}_q\}) \leq C^*_\text{max}(D_q)\). Thus, adding \(\bar{x}_q\) to the set \(l_{q-1}\) creates a feasible set \(l_q^\prime\) with \(|l_q^\prime| = |l_{q-1}| + 1\), which contradicts the assumption that \(l_q\) was generated by the MM algorithm since \(l_q = l_{q-1}\).
Hence, for the remainder of this proof, assume that Case 2 holds, but Case 1 does not. One of the following two cases must arise: (2a) $|l_q| = |l_{q-1}| + 1$; or (2b) $|l_q| = |l_{q-1}|$.

In Case (2a), $\bar{\pi}_q \in I_q$ and $\bar{\pi}_q \in D_q$ (or else the induction assumption is contradicted).

Defining $D_{q-1}$ as above, we have that $l_{q-1} = l_q \setminus \{\bar{\pi}_q\}$ and $|D_{q-1}| = |l_{q-1}|$. But then having $C^*_\text{max}(l_q) > C^*_\text{max}(D_q)$ implies that $\tilde{\text{C}}_{\text{max}}(l_q) > \tilde{\text{C}}_{\text{max}}(D_q)$ (by Lemma 4). Subtracting $p_{\bar{\pi}_q}$ from both sides of the latter inequality yields $\tilde{\text{C}}_{\text{max}}(l_{q-1}) > \tilde{\text{C}}_{\text{max}}(D_{q-1})$. Lemma 4 then guarantees that $C^*_\text{max}(l_{q-1}) > C^*_\text{max}(D_{q-1})$, contradicting the assumption that (4–57) holds for $l_{q-1}$.

In Case (2b), $|l_q| = |l_{q-1}|$ implies that $l_{q-1} \cup \{\bar{\pi}_q\}$ is infeasible, and so the MM algorithm adds $\pi_q$ to $l_{q-1}$ and removes a job $s_i$ having the longest processing time ($s_i \in \arg \max\{p_s|s \in l_{q-1} \cup \{\bar{\pi}_q\}\}$) from $l_{q-1}$ to form $l_q$. Therefore, $C^*_\text{max}(l_q) \leq C^*_\text{max}(l_{q-1})$. Since $C^*_\text{max}(D_q) < C^*_\text{max}(l_q)$, we conclude that $\bar{\pi}_q \in D_q$ (since otherwise, $D_q$ will be a feasible subset of $\{\bar{\pi}_1, \ldots, \bar{\pi}_{q-1}\}$ and $C^*_\text{max}(D_q) < C^*_\text{max}(l_q) \leq C^*_\text{max}(l_{q-1})$ contradicts the assumption that (4–57) holds for $l_{q-1}$). Also, by Lemma 4, $\tilde{\text{C}}_{\text{max}}(D_q) < \tilde{\text{C}}_{\text{max}}(l_q) \leq \tilde{\text{C}}_{\text{max}}(l_{q-1})$.

Define $\ell = \max\{|i|\bar{\pi}_i \in l_{q-1} \setminus D_q\}$ ($\ell$ exists since $l_{q-1} \subseteq D_q$ otherwise, but $\bar{\pi}_q \in D_q$ would imply $l_{q-1} \subset D_q$, contradicting $|l_{q-1}| = |D_q|$). Define $D'_{q-1} = D_q \cup \{\bar{\pi}_\ell\} \setminus \{\bar{\pi}_q\}$ and note that $|D'_{q-1}| = |l_{q-1}|$. Our proof will show that $D'_{q-1}$ is feasible, and that its length is smaller than the length of $l_{q-1}$, which contradicts the induction assumption. We first note that:

$$
\tilde{\text{C}}_{\text{max}}(D'_{q-1}) = \tilde{\text{C}}_{\text{max}}(D_q) + p_{\bar{\pi}_\ell} - p_{\bar{\pi}_q} < \tilde{\text{C}}_{\text{max}}(l_q) + p_{\bar{\pi}_\ell} - p_{\bar{\pi}_q}
\leq \tilde{\text{C}}_{\text{max}}(l_q) + p_{s_i} - p_{\bar{\pi}_q} = \tilde{\text{C}}_{\text{max}}(l_{q-1}).
$$

This inequality chain, along with Lemma 4, implies that $C^*_\text{max}(D'_{q-1}) < C^*_\text{max}(l_{q-1})$.

Therefore, to achieve the desired contradiction, we need to show that $D'_{q-1}$ is feasible. Define the set $S = \{\bar{\pi}_i \in l_{q-1}|i > \ell\}$ and note that $S \subset D_q$ (according to the definition of $\ell$). Let $\pi(D)$ be a feasible sequence of jobs in $D_q$ and form a new sequence $\pi'(D)$ by removing job $\bar{\pi}_q$ and the jobs in $S$ from $\pi(D)$, processing the remaining jobs as early
as possible (in the same order relative to each other as before), and then processing
job $\pi_\ell$, followed by the jobs in $S$ in EDD order. Note that $\pi'(D)$ is a possible schedule
for the jobs in $D'_{q-1}$. We claim that $\pi'(D)$ contains no late jobs. To prove this claim,
note that $C_i^{\phi^*(\pi(D))} \leq C_i^{\phi^*(\pi'(D))}$, $\forall i \in D_q \setminus (S \cup \{\pi_q\})$, since we schedule these jobs in
$\pi'(D)$ no later than they were scheduled in $\pi(D)$, which feasibly scheduled these jobs.
Let $m = |l_{q-1}| = |D'_{q-1}|$ and recall that $\pi'(D)_m = \text{EDD}(l_{q-1})_m \in S \cup \{\pi_\ell\}$. Also note
that $C_{\pi(D)}^\phi(D)_m < C_{\text{EDD}(l_{q-1})}^\phi(D)_m$ by $(4\mathcal{–}58)$ and Lemma 4. It then follows that the $m$th job in
$\pi'(D)$ is not late since $l_{q-1}$ is feasible by assumption. By Lemma 5, we conclude that
$p_{\pi(D)}^\phi(D)_m \geq p_{\text{EDD}(l_{q-1})}^\phi(D)_m$, which then implies that $C_{\pi(D)}^\phi(D)_m < C_{\text{EDD}(l_{q-1})}^\phi(D)_m$. Continuing this
procedure proves that $C_{\pi'(D)}^\phi(D)_s < C_{\text{EDD}(l_{q-1})}^\phi(D)_s$, $\forall s \in S \cup \{\pi_\ell\}$, and therefore all jobs in
$S \cup \{\pi_\ell\}$ complete on time in schedule $\pi'(D)$. Thus, $D'_{q-1}$ is feasible, contradicting the
induction assumption. This completes the proof. \hfill \Box

We now show that the complexity of the MM algorithm is $O(n^2)$. Note that Step 0 is
executed only once and requires generating $\pi$ by sorting the jobs in EDD order; so the
complexity of this step is $O(n \log n)$. Each one of the Steps 1, 3, and 4 requires $O(1)$
operations and is performed no more than $O(n)$ times for solving a problem with $n$ jobs.
Step 2 is performed a maximum of $O(n)$ times; moreover, for each $j = 1, \ldots, n$, this step
requires finding a job having the longest processing time ($O(j))$, and adjusting $\Delta$ and the
processing times of jobs in $\{\pi_1, \ldots, \pi_j\}$ ($O(j)$). Thus, worst-case complexity of this step
is $O(n^2)$. Finally, Step 5 requires $O(n)$ operations over the course of the algorithm. The
MM algorithm thus has a complexity of $O(n^2)$.

For US2, we present a min-max mathematical formulation of the problem. The
job-ordering decision variables are:

$$
x_{jk} = \begin{cases} 
1, & \text{if job } j \text{ is scheduled as the } k^{\text{th}} \text{ job in the sequence} \\
0, & \text{otherwise,} 
\end{cases} \quad \forall j \in J, \ k = 1, \ldots, n.
$$

Next, we define decision variables that mimic the SGP dynamic programming process
explained in Section 4.3.4.1. The following variables are defined with respect to
combinations of \( k \)-, \( l \)-, and \( r \)-values and signify that \( l \) of the first \( k \) jobs will be late, given that \( r \) of those \( k \) jobs have been maximally delayed. For brevity, we simply refer to a delay triple \((k, l, r)\) in the definitions below.

\[
f_{klr} = \text{largest completion time of the } k^{\text{th}} \text{ job in } \pi, \text{ given delay triple } (k, l, r)\]
\[
\forall k = 0, \ldots, n, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\}
\]

\[
u_{klr1} = \begin{cases} 
1, & \text{if the delayed version of the } (k+1)^{\text{st}} \text{ job in } \pi \text{ would finish after its deadline, given delay} \\
n\text{triple } (k, l, r) & \text{triple } (k, l, r) \\
0, & \text{otherwise,} \\
n\otherwise, & \forall k = 0, \ldots, n-1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M-1\} 
\end{cases}
\]

\[
u_{klr0} = \begin{cases} 
1, & \text{if the non-delayed version of the } (k+1)^{\text{st}} \text{ job in } \pi \text{ would finish after its deadline, given delay} \\
n\text{triple } (k, l, r) & \text{triple } (k, l, r) \\
0, & \text{otherwise,} \\
n\otherwise, & \forall k = 0, \ldots, n-1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\} 
\end{cases}
\]

Intuitively, our formulation specifies a job sequence using \( x \)-variables and determines the \( f \)- and \( u \)-variables corresponding to the sequence. These values induce a network whose nodes correspond to \((k, l, r)\) delay triples, as depicted in Figure 4-3. Arcs in this network exist from \((n, l, r)\) to \(T\), for all \( l = 0, \ldots, n \) and \( r = 0, \ldots, \min\{k, M\} \), all with the length of 0. Every other arc in the network is from a node \((k, l, r)\) to \((k+1, l', r')\) for some \( k = 0, \ldots, n-1, \ l = 0, \ldots, k, \ l' = l, l + 1, \ r = 0, \ldots, \min\{k, M\}, \) and \( r' = r, r + 1 \), and its length is 0 if \( l' = l \) and 1 if \( l' = l + 1 \). Thus, the length of each path from \( S \) (corresponding to \((0, 0, 0)\)) to \( T \) determines the number of late jobs resulting from the delay strategy corresponding to the path. Accordingly, the longest path in this network yields the maximum number of late jobs, given the values of \( u \)-variables for a job sequence. To model the longest-path problem over this (acyclic) network, we define the following binary variables:
\[
\begin{align*}
y_{klr', r'} &= \begin{cases} 
1, & \text{if the longest path uses an arc connecting delay triple } (k, l, r) \text{ to } (k + 1, l', r') \text{ in the network} \\
0, & \text{otherwise,}
\end{cases} \\
\forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad l' = l + 1, \quad r = 0, \ldots, \min\{k, M\}, \\
r' = r & \text{if } r = M, \quad r' = r, r + 1 \text{ if } r < M
\end{align*}
\]

\[
\begin{align*}
y_{nlr, T} &= \begin{cases} 
1, & \text{if the longest path uses an arc connecting delay triple } (n, l, r) \text{ to node } T \text{ in the network} \\
0, & \text{otherwise,}
\end{cases} \\
\forall l = 0, \ldots, n, \quad r = 0, \ldots, M
\end{align*}
\]

Define \( \epsilon \) and \( \beta \) as arbitrarily small and large numbers, respectively, and let \( \theta(u) \) be the maximum possible number of late jobs, given \( u \). Furthermore, define \( d_{\min} = \min_{j \in J} \{d_j\} \), \( d_{\max} = \max_{j \in J} \{d_j\} \), and \( p_{\min} = \min_{j \in J} \{p_j\} \). The min-max formulation is presented below.

Min \( \theta(u) \) \hspace{2cm} (4–59)

subject to:

\[
\sum_{k=1}^{n} x_{jk} = 1, \quad \forall j \in J \quad (4–60)
\]

\[
\sum_{j \in J} x_{jk} = 1, \quad \forall k = 1, \ldots, n \quad (4–61)
\]

\[
\frac{f_{klr} + \sum_{j \in J} [p_j(1 + K) - d_j] x_{j(k+1)}}{\sum_{j \in J} p_j(1 + K) - d_{\min}} \leq u_{klr1} \leq 1 + \frac{f_{klr} + \sum_{j \in J} [p_j(1 + K) - d_j] x_{j(k+1)} - \epsilon}{d_{\max} - p_{\min}(1 + K) + \epsilon},
\]

\[
\forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M - 1\} \quad (4–62)
\]

\[
\frac{f_{klr} + \sum_{j \in J} [p_j - d_j] x_{j(k+1)}}{\sum_{j \in J} p_j(1 + K) - d_{\min}} \leq u_{klr0} \leq 1 + \frac{f_{klr} + \sum_{j \in J} [p_j - d_j] x_{j(k+1)} - \epsilon}{d_{\max} - p_{\min} + \epsilon},
\]

\[
\forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M\} \quad (4–63)
\]

\[
f_{(k+1)(l+1)(r+1)} \geq \left[ f_{klr} + (1 + K) \sum_{j \in J} p_j x_{j(k+1)} \right] - \beta(1 - u_{klr1}).
\]
\[ f_{k+1,l+1} \geq f_{klr} + (1 + K) \sum_{j \in J} p_j x_{j(k+1)} - \beta u_{klr1}, \]
\[ \forall k = 0, \ldots, n - 1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M - 1\} \] (4–65)

\[ f_{k+1,l+1} \geq f_{klr} + \sum_{j \in J} p_j x_{j(k+1)} - \beta (1 - u_{klr0}), \]
\[ \forall k = 0, \ldots, n - 1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\} \] (4–66)

\[ f_{k+1,l+1} \geq f_{klr} + \sum_{j \in J} p_j x_{j(k+1)} - \beta u_{klr0}, \]
\[ \forall k = 0, \ldots, n - 1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\} \] (4–67)

\[ f_{klr} \geq 0, \]
\[ \forall k = 0, \ldots, n, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\} \] (4–68)

\[ u_{klr0} \in \{0, 1\}, \]
\[ \forall k = 0, \ldots, n - 1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\} \] (4–69)

\[ u_{klr1} \in \{0, 1\}, \]
\[ \forall k = 0, \ldots, n - 1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M - 1\}. \] (4–70)

The objective function (4–59) minimizes \( \theta(u) \), which corresponds to the number of late jobs in the worst-case scenario. (The optimal value of SGP corresponds to \( \theta(u) \), which will be calculated via the model presented in (4–71)–(4–79) next.) Constraints (4–60) and (4–61) enforce a permutation schedule for the jobs. In Constraints (4–62) and (4–63) we can assume, without loss of generality, that the denominators are positive. To see this, note that \( \sum_{j \in J} p_j (1 + K) - d_{\text{min}} \leq 0 \) implies that no late job exists, regardless of the sequence and the delay scenario. On the other hand, \( d_{\text{max}} - \rho_{\text{min}} (1 + K) + \epsilon \leq 0 \) \( (d_{\text{max}} - \rho_{\text{min}} + \epsilon \leq 0) \) implies that the delayed (non-delayed) version of every job in \( J \) is late, regardless of the sequence. All of the aforementioned cases result in optimization
problems with trivial solutions and are excluded from our studies. Accordingly, the upper bound (lower bound) of both Constraints (4–62) and (4–63) take on values within \([0, 1)\) (within \((-\infty, 0]\)) when the \((k + 1)\)st job finishes on time, and within \([1, \infty)\) (within \((0, 1]\)) when it finishes after its deadline. Therefore, Constraints (4–62) (Constraints (4–63)) set \(u_{klr1}\) (\(u_{klr0}\)) to one if the delayed (non-delayed) version of the \((k + 1)\)st job in \(\pi\) finishes after its deadline, and to zero otherwise. Note that these constraints are similar to Constraints (4–53), with additional lower-bounding constraints (the lower-bounding constraints in this case are not automatically enforced as is the case in the formulation presented in (4–51)–(4–56) and must be explicitly added to the problem).

The next four constraints force the \(f\)-variables to take on their correct values. Given that the \(k\)th job in \(\pi\) finishes at \(f_{klr}\), Constraints (4–64) (Constraints (4–65)) calculate the completion time of delayed version of the \((k + 1)\)st job when \(u_{klr1} = 1\) (\(u_{klr1} = 0\)). Similarly, Constraints (4–66) (Constraints (4–67)) calculate the completion time of non-delayed version of the \((k + 1)\)st job, when \(u_{klr0} = 1\) (\(u_{klr0} = 0\)). Finally, Constraints (4–68)–(4–70) state logical restrictions on the model variables. Note that the value of \(\epsilon\) can be defined using the same logic discussed in Section 4.3.4.1 for defining \(\epsilon_j\) in Constraints (4–53).

Moreover, the largest possible completion time of jobs in \(J\) establishes a practical value for \(\beta\) in Constraints (4–64)–(4–67), i.e., \(\beta = (1 + K) \sum_{j \in J} p_j\).

The problem of finding a longest path is stated as follows.

\[
\theta(u) = \text{Max} \sum_{k=0}^{n-1} \sum_{l=0}^{k} \left[ \sum_{r=0}^{\min\{k,M\}} y_{klr(r+1)} + \sum_{r=0}^{\min\{k+1,M-1\}} y_{klr(l+1)(r+1)} \right] \tag{4–71}
\]

subject to:

\[
y_{klr(l+1)(r+1)} \leq u_{klr1}, \quad \forall k = 0, \ldots, n-1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k,M-1\} \tag{4–72}
\]

\[
y_{klr(r+1)} \leq 1 - u_{klr1}, \quad \forall k = 0, \ldots, n-1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k,M-1\} \tag{4–73}
\]

\[
y_{klr(r+1)r} \leq u_{klr0}, \quad \forall k = 0, \ldots, n-1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k,M\} \tag{4–74}
\]
Figure 4-3. SGP network for the $\sum_{j \in J} U_j$ criterion under US2. The value $[a, b]$ on each arc indicates that the arc has a cost of $a$ and a capacity of $b$.

\[ y_{k|l|r} \leq 1 - u_{k|l|0}, \quad \forall k = 0, \ldots, n - 1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\} \quad (4-75) \]

\[ y_{00011} + y_{00001} + y_{00010} + y_{00000} = 1, \quad (4-76) \]

\[ y_{k|l|l+1} + y_{k|l+1} + y_{k|l+1}|r + y_{k|l}|r \\
- y_{(k-1)|(l-1)|r} - y_{(k-1)|(l-1)|l} - y_{(k-1)|(l-1)|l} - y_{(k-1)|l|l} = 0, \quad \forall k = 1, \ldots, n-1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\} \quad (4-77) \]

\[ y_{l|l+1} - y_{(n-1)|(l-1)|l} - y_{(n-1)|(l-1)|l} - y_{(n-1)|(l-1)|l} - y_{(n-1)|l|l} = 0, \quad \forall l = 0, \ldots, n, \ r = 0, \ldots, \min\{n, M\} \quad (4-78) \]

\[ y_{k|l|l'} \geq 0, \quad \forall k = 0, \ldots, n-1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\} \quad (4-79) \]
The objective function (4–71) calculates \( \theta(u) \) as the maximum length of a path from 'S' to 'T' in the network shown in Figure 4-3. Constraints (4–72)–(4–75) ensure that the flow on each arc of the network is no more than its capacity. Constraints (4–76)–(4–78) state the flow-balance constraints for all nodes in the network (except for T, which is not necessary to include). Finally, Constraints (4–79) enforce variable nonnegativity. Note that because the coefficient matrix defining the constraint set (4–72)–(4–78) is totally unimodular, and because the network is acyclic, an optimal solution to the problem formulated in (4–71)–(4–79) must exist in which all \( y \)-variables are binary-valued.

Due to the convexity of the formulation presented in (4–71)–(4–79), we can employ the strong duality theorem to replace \( \theta(u) \) by the optimal objective of its dual formulation. Accordingly, as the next step of the research, we form the dual of the formulation presented in (4–59)–(4–70) and combine the two formulations ((4–59)–(4–70) and the dual of (4–71)–(4–79)) in order to obtain a single mathematical formulation for the robust problem of interest.

For each \( k = 0, \ldots, n - 1 \), \( l = 0, \ldots, k \), and \( r = 0, \ldots, \min\{k, M - 1\} \), define \( v_{klr}(l+1)(r+1) \), \( v_{klrl}(r+1) \), \( v_{klr} \), and \( v_{klrlr} \) as dual variables corresponding to Constraints (4–72), (4–73), (4–74), and (4–75), respectively. Also, define \( w_{klr} \), for \( k = 0, \ldots, n \), \( l = 0, \ldots, k \), and \( r = 0, \ldots, \min\{k, M - 1\} \) as dual variables corresponding to the flow-balance constraints (4–76)–(4–78). The dual formulation can therefore be stated as follows:

\[
\begin{align*}
\text{Min} & \quad \sum_{k=0}^{n-1} \sum_{l=0}^{k} \sum_{r=0}^{\min\{k,M-1\}} \left[ U_{klr} v_{klr}(l+1)(r+1) + (1 - U_{klr}) v_{klrl}(r+1) \right] \\
& \quad + \sum_{k=0}^{n-1} \sum_{l=0}^{k} \sum_{r=0}^{\min\{k,M\}} \left[ U_{klr0} v_{klr}(l+1) + (1 - U_{klr0}) v_{klrlr} \right] + w_{000}
\end{align*}
\]  

subject to:

\[ v_{klrl} + w_{klr} - w_{(k+1)lr} \geq 0, \quad \forall k = 0, \ldots, n - 1, \ l = 0, \ldots, k, \ r = 0, \ldots, \min\{k, M\}. \]
\( r' = r \) if \( r = M \), \( r' = r, r + 1 \) if \( r < M \) \hfill (4–82) \\
\begin{align*}
v_{klr(l+1)r'} + w_{klr} - w_{(k+1)(l+1)r'} &\geq 1, \\
\forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M\},
\end{align*}
\hfill (4–83) \\
\begin{align*}
w_{nlr} &\geq 0, \\
\forall l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M\},
\end{align*}
\hfill (4–84) \\
\begin{align*}
v_{klr'r'} &\geq 0, \\
\forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M\}, \\
l' = l, l + 1, \quad r' = r \text{ if } r = M, \quad r' = r, r + 1 \text{ if } r < M. \hfill (4–85)
\end{align*}

Next, we combine the dual formulation (4–81)–(4–85) with the primal formulation (4–59)–(4–70) and represent a mathematical formulation for the robust problem as follows:

\[
\begin{align*}
\text{Min} & \sum_{k=0}^{n-1} \sum_{l=0}^{k} \sum_{r=0}^{\min\{k, M-1\}} \left( u_{klr1} v_{klr(l+1)(r+1)} + (1 - u_{klr1}) v_{klr(l+1)} \right) \\
&+ \sum_{k=0}^{n-1} \sum_{l=0}^{k} \sum_{r=0}^{\min\{k, M\}} \left( u_{klr0} v_{klr(l+1)r} + (1 - u_{klr0}) v_{klr} \right) + w_{000} \\
\end{align*}
\hfill (4–86)
\]

subject to:

\[
(4–60)–(4–70) \text{ and } (4–82)–(4–85)
\]

We next show that the above formulation can be linearized and solved using a standard MILP solver. Note that each nonlinear term in (4–86) is a product of a \( u \)- and a \( v \)-variable, where \( u_{klr0}, u_{klr1} \in \{0, 1\} \) for all delay triples \((k, l, r)\). We claim that in an optimal solution to (4–86), \( v_{klr'l'} \in [0, 1] \) for all delay triples \((k, l, r), l' = l, l + 1, \) and \( r' = r, r + 1 \) (if \( r = M \), then \( r' = r \)). Note that when \( u \)-values are known, we can remove the arcs with capacity 0 in the network of Figure 4-3 and omit the corresponding capacity constraints in (4–72)–(4–75). Since the network is acyclic, the longest path problem can be solved by a standard search algorithm in which nodes are examined in the sequence of topological ordering of the nodes (see [1]). The effort required for this approach is proportional to the number of arcs in the network. Note that at optimality, the \( y \)-variables take on the reduced-cost values corresponding to the arcs. Therefore, we have \( y_{klr'l'} = \max\{0, w_{(k+1)(l+1)r'} - w_{klr} \} \) and \( y_{klr(l+1)r'} = \max\{0, 1 + w_{(k+1)(l+1)r'} - w_{klr} \} \). Since
every arc from a node \((k, l, r)\) points to a node \(((k + 1), l', r')\) in the network \((l' = l, l + 1,\) and \(r' = r, r + 1)\), and its cost is either 0 or 1, we have \(w_{(k+1,l',r')} - w_{klr} \in \{0, -1\}\), which then implies that \(y_{klr^l} \in \{0, 1\}\).

The above discussion shows that we can implement standard linearization techniques for quadratic programs to linearize (4–86), similar to the method implemented in Section 2.3 to linearize the quadratic terms in (2–17). Define \(v_{klr^l} = u_{klr} v_{klr^l}^0\) and \(v_{klr^l} = u_{klr} v_{klr^l}^0\) for each delay triple \((k, l, r)\), and \(l' = l, l + 1\). The MILP formulation of the robust problem is presented below.

\[
\begin{align*}
\text{Min} & \quad \sum_{k=0}^{n-1} \sum_{k'=0}^{k} \sum_{k''=0}^{k} \sum_{r=0}^{\min\{k, M-1\}} \left[ v_{klr^l+r} + v_{k'o} - v_{k'o} \right] + w_{klr^l} \\
\text{subject to:} & \quad v_{klr^l}^0 \leq u_{klr}^0, \quad \forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M\}, \quad l' = l, l + 1 \quad (4–88) \\
& \quad v_{klr^l}^0 \leq v_{klr^l}, \quad \forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M\}, \quad l' = l, l + 1 \quad (4–89) \\
& \quad v_{klr^l}^0 \geq u_{klr}^0 + v_{klr^l}^1 - 1, \quad \forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M\}, \quad l' = l, l + 1 \quad (4–90) \\
& \quad v_{klr^l}^0 \geq 0, \quad \forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M\}, \quad l' = l, l + 1 \quad (4–91) \\
& \quad v_{klr^l}^0 \leq u_{klr}, \quad \forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M - 1\}, \quad l' = l, l + 1 \quad (4–92) \\
& \quad v_{klr^l}^0 \leq v_{klr^l}^1, \quad \forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M - 1\}, \quad l' = l, l + 1 \quad (4–93) \\
& \quad v_{klr^l}^0 \geq u_{klr} + v_{klr^l}^1 - 1, \quad \forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M - 1\}, \quad l' = l, l + 1 \quad (4–94) \\
& \quad v_{klr^l}^0 \geq 0, \quad \forall k = 0, \ldots, n - 1, \quad l = 0, \ldots, k, \quad r = 0, \ldots, \min\{k, M - 1\}, \quad l' = l, l + 1 \quad (4–95) \\
\end{align*}
\]

(4–60)–(4–70) and (4–82)–(4–85)

Although the above discussion led us to an MILP formulation for the problem discussed in this section under US2, the problem complexity remains open. In Chapter 5, we investigate the level of difficulty of this problem by implementing the model presented in (4–87)–(4–96) and solving a set of randomly-generated instances using
CPLEX. We also explore several heuristic ideas to generate lower and upper bounds for the problem and improve the optimality gap.

As the last case introduced in our research, we consider the problem of robust SMSP with the objective of minimizing the number of late jobs under US3. Recall that we were unable to generate a polynomial algorithm to solve the SGP corresponding to this problem. Also, note that binary constraints in the MILP formulation presented in (4–51)–(4–56) implies nonconvexity of the SGP. Thus, we omit further discussion in this regard and leave this problem as an open question.
In this chapter we provide a more detailed study for a special case of robust single-machine scheduling problems where the objective is to minimize the number of late jobs in the worst-case scenario under uncertainty set 2 (US2), as defined in Section 4.3.4. We use the same notation and definitions as those presented in Chapter 4.

The remainder of this chapter is organized as follows. In Section 5.1 we present heuristic upper- and lower-bounding algorithms to generate and evaluate approximate solutions for a set of random instances. Then in Section 5.2 we test the MILP formulation presented in Section 4.3.4 (see (4–87)–(4–96)) by solving a set of randomly-generated instances using CPLEX, where we specify the upper-bounding solution presented in Section 5.1 as an incumbent solution to the problem. All computations in this chapter are performed on an Intel Core i7 with a 1.80 GHz processor and 8.0 GB RAM.

5.1 Approximate Solutions

In this section we present polynomial algorithms to find upper and lower bounds for the problem discussed in this chapter. We then solve a set of randomly generated instances using these algorithms and specify the optimality gap.

5.1.1 Upper Bound

In order to find an upper bound for the problem, we use a heuristic upper-bounding (HUB) algorithm to generate a sequence of jobs. Then, we solve the SGP corresponding to the created sequence using the dynamic-programming algorithm presented in Section 4.3.4.1. As the output of this process, we obtain the maximum number of late jobs corresponding to an arbitrary sequence of jobs, which yields an upper bound for the problem. The HUB algorithm is described as follows.

Step 0. Let \( \bar{x} \) be an EDD sequence of jobs and \( x \) (initially empty) be the sequence that we construct using this algorithm. Also, let \( R \) (initially empty) be the set of rejected jobs and \( r \) be the last rejected job. Define \( I \) to be the subset of jobs that complete
before their deadlines when we use this algorithm to schedule jobs \( \{\tilde{\pi}_1, \ldots, \tilde{\pi}_i\} \) (with \( l_0 = \emptyset \)). Also, define \( G_j \) to be the set of \( \min\{M, j\} \) jobs having the longest processing times among all jobs in positions \( 1, \ldots, j \). (That is, for the case in which \( j > M \), we have that \( \pi_s \in G_j \) for \( 1 \leq s \leq j \) implies that \( p_{\pi_s} \geq p_{\pi_q} \) for every \( q \) such that \( 1 \leq q \leq j \) and \( \pi_q \notin G_j \).) Initialize \( i = j = 1 \), where \( i \) is the job position currently under examination in \( \tilde{\pi} \) and \( j \) is the job position being scheduled in \( \pi \).

**Step 1.** Tentatively schedule \( \tilde{\pi}_i \) in the \( j^{th} \) position of \( \pi (\pi_j = \tilde{\pi}_i) \). Now, if \( \sum_{s=1}^{j} p_{\pi_s} + K(\sum_{s \in G_j} p_{\pi_s}) > d_{\pi_j} \), then go to Step 2; otherwise, go to Step 3.

**Step 2.** Adjust the schedule of jobs in \( \pi \) as follows:

**Step 2-1.** Find \( q \in \arg \max_{s \in \{1, \ldots, j\}} \{p_{\pi_s}\} \), and choose \( r = \pi_q \) to be the next rejected job. Add \( \pi_q \) to \( R \) and go to Step 2-2.

**Step 2-2.** If \( q = j \), then go to Step 4; otherwise, go to Step 2-3.

**Step 2-3.** Set \( \pi_q = \pi_{q+1} \). Go to Step 2-4.

**Step 2-4.** Increment the value of \( q \) by one and go to Step 2-2.

**Step 3.** Set \( I_i = I_{i-1} \cup \{\pi_j\} \). Increment \( i \) and \( j \) by one and go to Step 5.

**Step 4.** Set \( I_i = I_{i-1} \cup \{\pi_j\} \setminus \{r\} \). Increment \( i \) by one and go to Step 5.

**Step 5.** If \( i \leq n \), then go to Step 1; otherwise, schedule the jobs in \( R \) in positions \( j, \ldots, n \) of \( \pi \), in any order, and terminate.

After generating \( \pi \) using the HUB algorithm, we calculate the maximum number of late jobs in \( \pi \) by solving the corresponding SGP using the dynamic-programming algorithm presented in Section 4.3.4.1.

We next prove that the worst-case complexity of the presented upper-bounding procedure is \( \mathcal{O}(Mn^2) \). We first claim that the complexity of HUB algorithm is no more than \( \mathcal{O}(Mn^2) \). Note that Step 0 of HUB is executed only once and requires generating \( \tilde{\pi} \) by sorting the jobs in EDD order; so the complexity of this step is \( \mathcal{O}(n \log n) \). Step 1 is executed \( n \) times, and for each \( j = M + 1, \ldots, n \), it identifies \( M \) jobs having the longest processing times in \( \{\pi_1, \ldots, \pi_j\} \) to form the set \( G_j (\mathcal{O}(Mj)) \). Therefore, the
worst-case complexity of this step is $O(Mn^2)$. Step 2 is performed a maximum of $O(n)$ times; moreover, for each $j = 1, \ldots, n$, this step requires finding a job having the longest processing time ($O(j)$). Thus, the worst-case complexity of this step is $O(n^2)$. Steps 3 and 4 each require $O(1)$ operations and is performed no more than $O(n)$ times. Finally, Step 5 requires $O(n)$ operations over the course of the algorithm. As a result, the worst-case complexity of HUB algorithm is $O(Mn^2)$, and so the overall complexity of the upper-bounding procedure is $O(Mn^2)$ since the complexity of our dynamic-programming algorithm is also $O(Mn^2)$ (see Section 4.3.4.1).

5.1.2 Lower Bounds

In this section we consider single-stage and multi-stage approaches to generate lower bounds for the problem. We present the details of these approaches in the following subsections.

5.1.2.1 Single-Stage Approaches

In order to generate lower bounds for the problem, we assume that the set of delayed jobs is fixed, regardless of the sequence of jobs. We consider three different strategies for selecting the set of delayed jobs. For each case we first determine the job processing-time values via the corresponding strategy; then we find a sequence of jobs with minimum number of late jobs, given that processing times stay unchanged. Because the processing times are pre-determined for each case, finding a sequence that minimizes the number of late jobs is a deterministic problem and can be solved optimally using Moore’s algorithm [66], presented in Section 4.3.4. Note that this method obtains a lower bound for the problem because we replace the optimal objective value of the inner (maximization) problem in the min-max robust optimization formulation of the problem with the objective value of a feasible solution.

Next, we introduce three different strategies for selecting delayed jobs. Then, in Section 5.1.4, we calculate the lower bound corresponding to each strategy as discussed above, for a set of randomly-generated instances.
Lower bound 1 (LB1): To generate the first lower bound, we assume that $M$ jobs having longest processing-time values are delayed. In case several jobs having the same processing time exist, the jobs with earliest due dates are delayed.

Lower bound 2 (LB2): For our second lower bound, we first execute the HUB algorithm presented in Section 5.1.1 and determine the set of jobs that are delayed when the algorithm terminates. We then choose the same set of jobs to be delayed.

Lower bound 3 (LB3): To create our last single-stage lower bound, we use a heuristic lower-bound (HLB) algorithm to generate a set of delayed jobs ($D$) as follows.

**Step 0.** Let $\bar{\pi}$ be an EDD schedule of jobs and $\pi$ (initially empty) be the schedule that we construct through the course of this algorithm. Also, let $D$ (initially empty) be the set of delayed jobs. Initialize $i = j = 1$, where $i$ is the job position currently under examination in $\bar{\pi}$ and $j$ is the job position being scheduled in $\pi$.

**Step 1.** Tentatively schedule $\bar{\pi}_j$ in the $j^{th}$ position of $\pi$ ($\pi_j = \bar{\pi}_j$). If $\pi_j$ is late in $\pi$ without being delayed, then set $p^*_{\pi_j} = p_{\pi_j}$ and go to Step 2; otherwise, if either $\pi_j$ is not late in $\pi$ after being delayed or $|D| = M$, set $p^*_{\pi_j} = p_{\pi_j}$ and go to Step 3. Else ($\pi_j$ is late in $\pi$ only after being delayed and $|D| < M$), then set $p^*_{\pi_j} = p_{\pi_j} * (1 + K)$, add $\pi_j$ to $D$, and go to Step 2.

**Step 2.** Adjust the schedule of jobs in $\pi$ as follows:

**Step 2-1.** Find $q \in \arg\max_{s \in \{1, \ldots, j\}} \{p^*_{\pi_s}\}$, and go to Step 2-2.

**Step 2-2.** If $q = j$, then go to Step 4; otherwise, go to Step 2-3.

**Step 2-3.** Set $\pi_q = \pi_{q+1}$. Increment $q$ by one and go to Step 2-2.

**Step 3.** Increment $i$ and $j$ by one and go to Step 5.

**Step 4.** Increment $i$ by one and go to Step 5.

**Step 5.** If $i \leq n$, then go to Step 1; otherwise, go to Step 6.
**Step 6.** If $|D| < M$, form the set $G$ as the set of $M - |D|$ jobs in $J \setminus D$ having the longest processing times, set $p_j^* = p_j * (1 + K), \forall j \in G$, add the jobs in $G$ to the set $D$, and terminate the algorithm; otherwise, terminate the algorithm.

### 5.1.2.2 Multi-Stage Approaches

In this section we present two interactive lower-bounding procedures. For each procedure we solve the inner and outer problems recursively, where the output of one problem is considered to be the input of the other one, and vice-versa.

Our first procedure (LB4) starts by solving the inner problem (maximizing the number of late jobs in a given sequence by delaying $M$ jobs) corresponding to an EDD sequence of jobs using the dynamic-programming algorithm presented in Section 4.3.4.1. It then reschedules the jobs (solves the outer problem), given job processing times generated in the previous step, using Moore’s algorithm. As discussed before, the number of rejected jobs given by Moore’s algorithm forms a lower bound for the problem. This completes one iteration of the algorithm. To perform the next iteration, the sequence of jobs is fixed to be the one we generated in the previous iteration. We continue the procedure in a similar way and at the end of each iteration we update the lower bound if we find a greater value. We stop the procedure when the lower bound does not improve in 10 consecutive iterations.

Our second procedure (LB5) starts by solving the outer problem (finding a sequence of jobs having the minimum number of late jobs for a given set of processing-time values), assuming that $M$ jobs having the largest processing times are delayed, using Moore’s algorithm. The number of rejected jobs given by Moore’s algorithm forms a lower bound for the problem. (Note that this step is the same as the procedure we used to calculate UB1, so we expect the lower bound generated by this procedure to be at least as good as LB1.) In the next step, the procedure delays $M$ jobs in the generated sequence to maximize number of late jobs, using the dynamic programming algorithm presented in Section 4.3.4.1. This completes one iteration of the algorithm. To perform
the next iteration, we fix the processing times to be the ones generated in the previous iteration. We continue the procedure in a similar way, and at the end of each iteration we update the lower bound if we find a greater value. We stop the procedure when the lower bound does not improve in 10 consecutive iterations.

Note that although our multi-stage procedures require more effort compared to the single-stage procedures, they yield better bounds. In addition, the average solution time for the 100 instances having 100 jobs each is less than one second for both LB4 and LB5. The details of our test instances and the results of our computations are presented in Sections 5.1.3 and 5.1.4, respectively.

5.1.3 Test Problem Generation

We generate 100 random instances of the robust SMSP under US2, each having 100 jobs ($n = 100$), as follows. For each instance, we randomly generate an integer between 1 and $n - 1$ as the maximum number of delayed jobs ($M$). The value of parameter $K$ and the values of job processing times ($p_j, \forall j \in J$) are randomly drawn from the interval of $[0, 2]$ and $[0, 10]$, respectively.

In order to generate job due dates, we introduce a constant, $\gamma$, as a randomly generated value between 0 and 5. Then, for each job $j \in J$, we define $\Gamma_j$ as follows,

$$\Gamma_j = \gamma + \sum_{i=1}^{j} p_i (1 + K), \quad \forall j \in J,$$

and set the value of $d_j$ to a random number in the interval of $[p_j (1 + K), \Gamma_j]$. Using this method to generate due-date values tends to avoid instances having trivial solutions, i.e., instances in which $d_j$-values are either too small (e.g. $d_j < p_j (1 + K)$ implies that delaying job $j$ makes it late, regardless of the sequence), or too large (e.g. $d_j \geq (1 + K)(\sum_{i \in G} p_i) + \sum_{i \in J \setminus G} p_i$, where $G$ is the set of $M$ jobs in $J$ having the longest processing times, implies that job $j$ is never late, regardless of the sequence and the delay scenario).
5.1.4 Results

To compare and evaluate our upper and lower bounds, we solve the random instances presented in Section 5.1.3 using each algorithm. Table 5-1 presents the results of our computations. In this table we specify random instances by numbers between 1 and 100 in Column “Instance”. For each instance we present the objective value obtained by the lower-bounding Algorithm \( i \) (LB\( i \)) presented in Section 5.1.2 in Column “LB\( i \)” \( (i = 1, \ldots, 5) \), and the the one obtained by the upper-bounding algorithm (UB) presented in Section 5.1.1 in Column “UB”. We then evaluate the optimality gap for each instance by presenting the difference between the upper bound and the greatest lower bound in Column “UB-Max(LBs)”.

Based on the results presented in Table 5-1, multi-stage lower bounds 4 and 5 (LB4 and LB5) provide the best bounds in all of our instances, which is predictable due to their interactive nature. In 24 instances, we are able to close the gap between the lower and upper bounds and prove the optimality of the sequence generated by our upper-bounding algorithm. We further investigate the quality of our upper bound by comparing it to the optimal objective value of smaller instances in Section 5.2.

5.2 Exact Solution

In this section we implement the MILP formulation presented in Section 4.3.4 for solving the robust SMSP with the objective of minimizing the number of late jobs under US2, i.e., (4–87)–(4–96), using CPLEX 12.3 via ILOG Concert Technology. We first seek to estimate the size of instances that can be solved to optimality in a reasonable time using this formulation. To that end, we randomly generate 20 instances via the same method presented in Section 5.1.3, with a slight modification: we now assume that the number of jobs \( n \) in each instance is an integer random number with discrete uniform distribution between 2 and 10. (The value of \( \epsilon \) in our MILP formulation is set to 0.1 for all instances.) We then solve each instance using CPLEX as follows.
Table 5-1. Evaluation of lower- and upper- bounding heuristic algorithms

<table>
<thead>
<tr>
<th>Instance</th>
<th>LB1</th>
<th>LB2</th>
<th>LB3</th>
<th>LB4</th>
<th>LB5</th>
<th>UB</th>
<th>UB-Max(LBs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
<td>20</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>9</td>
<td>8</td>
<td>18</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>13</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>26</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>23</td>
<td>14</td>
<td>23</td>
<td>24</td>
<td>24</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>23</td>
<td>25</td>
<td>23</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>19</td>
<td>20</td>
<td>23</td>
<td>24</td>
<td>29</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>22</td>
<td>22</td>
<td>25</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>22</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>22</td>
<td>22</td>
<td>26</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>21</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>25</td>
<td>26</td>
<td>25</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>23</td>
<td>18</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>29</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>26</td>
<td>19</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>24</td>
<td>16</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>14</td>
<td>12</td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>22</td>
<td>22</td>
<td>28</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>14</td>
<td>15</td>
<td>14</td>
<td>16</td>
<td>15</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>15</td>
<td>23</td>
<td>24</td>
<td>24</td>
<td>26</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>14</td>
<td>23</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>10</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>19</td>
<td>7</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>22</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>27</td>
<td>17</td>
<td>17</td>
<td>19</td>
<td>20</td>
<td>20</td>
<td>29</td>
<td>9</td>
</tr>
<tr>
<td>28</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>19</td>
<td>21</td>
<td>20</td>
<td>22</td>
<td>22</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>15</td>
<td>20</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>10</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>32</td>
<td>25</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>26</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>25</td>
<td>26</td>
<td>25</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>23</td>
<td>15</td>
</tr>
<tr>
<td>35</td>
<td>10</td>
<td>4</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>25</td>
<td>25</td>
<td>27</td>
<td>27</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>37</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>24</td>
<td>24</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>38</td>
<td>13</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>39</td>
<td>21</td>
<td>17</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>12</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Table 5-1. Continued

<table>
<thead>
<tr>
<th>Instance</th>
<th>LB1</th>
<th>LB2</th>
<th>LB3</th>
<th>LB4</th>
<th>LB5</th>
<th>UB</th>
<th>UB-Max(LBs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>24</td>
<td>22</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>42</td>
<td>28</td>
<td>25</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td>23</td>
<td>12</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>26</td>
<td>3</td>
</tr>
<tr>
<td>45</td>
<td>15</td>
<td>14</td>
<td>16</td>
<td>17</td>
<td>16</td>
<td>23</td>
<td>6</td>
</tr>
<tr>
<td>46</td>
<td>25</td>
<td>14</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>47</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>48</td>
<td>20</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>49</td>
<td>23</td>
<td>13</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>28</td>
<td>5</td>
</tr>
<tr>
<td>50</td>
<td>18</td>
<td>19</td>
<td>18</td>
<td>20</td>
<td>20</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>51</td>
<td>19</td>
<td>22</td>
<td>20</td>
<td>23</td>
<td>23</td>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>52</td>
<td>12</td>
<td>9</td>
<td>12</td>
<td>14</td>
<td>14</td>
<td>23</td>
<td>9</td>
</tr>
<tr>
<td>53</td>
<td>19</td>
<td>17</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>54</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>21</td>
<td>7</td>
</tr>
<tr>
<td>55</td>
<td>26</td>
<td>24</td>
<td>26</td>
<td>25</td>
<td>26</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td>14</td>
<td>15</td>
<td>14</td>
<td>16</td>
<td>16</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>57</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>19</td>
<td>19</td>
<td>24</td>
<td>5</td>
</tr>
<tr>
<td>58</td>
<td>23</td>
<td>22</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>59</td>
<td>22</td>
<td>21</td>
<td>22</td>
<td>22</td>
<td>23</td>
<td>28</td>
<td>5</td>
</tr>
<tr>
<td>60</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>22</td>
<td>22</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>61</td>
<td>24</td>
<td>17</td>
<td>24</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>62</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>63</td>
<td>30</td>
<td>17</td>
<td>30</td>
<td>29</td>
<td>30</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>34</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td>65</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>66</td>
<td>25</td>
<td>19</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td>67</td>
<td>26</td>
<td>19</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>69</td>
<td>24</td>
<td>13</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>70</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>71</td>
<td>22</td>
<td>23</td>
<td>22</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>72</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>73</td>
<td>16</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>26</td>
<td>9</td>
</tr>
<tr>
<td>74</td>
<td>19</td>
<td>15</td>
<td>19</td>
<td>20</td>
<td>20</td>
<td>26</td>
<td>6</td>
</tr>
<tr>
<td>75</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>76</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>77</td>
<td>21</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>22</td>
<td>25</td>
<td>3</td>
</tr>
<tr>
<td>78</td>
<td>15</td>
<td>12</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>21</td>
<td>5</td>
</tr>
<tr>
<td>79</td>
<td>19</td>
<td>12</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>23</td>
<td>4</td>
</tr>
<tr>
<td>80</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 5-1. Continued

<table>
<thead>
<tr>
<th>Instance</th>
<th>LB1</th>
<th>LB2</th>
<th>LB3</th>
<th>LB4</th>
<th>LB5</th>
<th>UB</th>
<th>UB-Max(LBs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>81</td>
<td>21</td>
<td>17</td>
<td>21</td>
<td>22</td>
<td>21</td>
<td>29</td>
<td>7</td>
</tr>
<tr>
<td>82</td>
<td>25</td>
<td>18</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>83</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td>13</td>
<td>22</td>
<td>8</td>
</tr>
<tr>
<td>84</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>85</td>
<td>24</td>
<td>25</td>
<td>25</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>86</td>
<td>16</td>
<td>17</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>87</td>
<td>15</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>15</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>88</td>
<td>20</td>
<td>11</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>23</td>
<td>3</td>
</tr>
<tr>
<td>89</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>90</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>91</td>
<td>9</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>13</td>
<td>20</td>
<td>7</td>
</tr>
<tr>
<td>92</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>93</td>
<td>17</td>
<td>18</td>
<td>18</td>
<td>20</td>
<td>20</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>94</td>
<td>9</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>95</td>
<td>17</td>
<td>10</td>
<td>17</td>
<td>18</td>
<td>17</td>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>96</td>
<td>25</td>
<td>22</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>29</td>
<td>4</td>
</tr>
<tr>
<td>97</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>98</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>99</td>
<td>24</td>
<td>15</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>100</td>
<td>32</td>
<td>23</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>0</td>
</tr>
</tbody>
</table>

As a preprocessing step, we use the algorithm presented in Section 5.1.1 to generate an upper-bounding sequence and a delay scenario for each instance. We then add this solution as an incumbent solution to the corresponding instance in CPLEX. This preprocessing step is likely to expedite the process of optimally solving the problem by reducing the size of the branch-and-bound tree (fathoming branches whose corresponding objective values are greater than our upper-bound value) early on, which may dramatically reduce the solution time, especially when the upper bound is tight. We also set the optimality tolerance in CPLEX to 0.999, because the objective function of the problem (number of late jobs) is integer and any difference less than one between the lower and upper bound proves the optimality of our best incumbent solution. We then solve each instance using CPLEX until either the optimality tolerance, or the one-hour (3600 seconds) time limit is reached.
Table 5-2. Evaluation of exact solution method ($2 \leq n \leq 10$)

<table>
<thead>
<tr>
<th>Instance</th>
<th>$n$</th>
<th>$M$</th>
<th>$K$</th>
<th>UB</th>
<th>Time</th>
<th>Nodes</th>
<th>CPLEX LB</th>
<th>CPLEX UB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>6</td>
<td>0.960</td>
<td>3</td>
<td>3600</td>
<td>15651</td>
<td>1.500</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>0.285</td>
<td>3</td>
<td>3600</td>
<td>32756</td>
<td>1.047</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>4</td>
<td>0.411</td>
<td>3</td>
<td>404</td>
<td>8479</td>
<td>2.001</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>1</td>
<td>1.400</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.000</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>0.811</td>
<td>2</td>
<td>5</td>
<td>303</td>
<td>1.013</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>1.989</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.026</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>7</td>
<td>1.416</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0.002</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>1.256</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2.000</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>5</td>
<td>1.169</td>
<td>3</td>
<td>14</td>
<td>419</td>
<td>2.005</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1</td>
<td>1.282</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>1.570</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.013</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>4</td>
<td>0.229</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1.010</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>5</td>
<td>0.714</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0.002</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>3</td>
<td>0.639</td>
<td>3</td>
<td>12</td>
<td>591</td>
<td>2.017</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>8</td>
<td>1.488</td>
<td>3</td>
<td>2197</td>
<td>8023</td>
<td>2.001</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>3</td>
<td>1.892</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2.000</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>7</td>
<td>6</td>
<td>1.153</td>
<td>3</td>
<td>34</td>
<td>1508</td>
<td>2.004</td>
<td>3</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>1</td>
<td>1.564</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.007</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>7</td>
<td>4</td>
<td>1.843</td>
<td>3</td>
<td>209</td>
<td>7868</td>
<td>2.005</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td>5</td>
<td>0.883</td>
<td>2</td>
<td>16</td>
<td>1546</td>
<td>1.001</td>
<td>2</td>
</tr>
</tbody>
</table>

The results of the experiments discussed in this section are presented in Table 5-2. In this table for each instance number (1, ..., 20) listed in Column “Instance”, the corresponding parameter values, $n$, $M$, and $K$, are presented in Columns “n”, “M”, and “K”, respectively. The upper-bound value generated in the preprocessing step is listed in Column “UB”. After implementing the discussed solution method, we record the solution time (“Time”) in seconds, number of nodes in the branch-and-bound tree (“Nodes”), and the calculated bounds for the problem in CPLEX (“CPLEX LB” for Lower Bound and “CPLEX UB” for Upper Bound). Note that solution time of the instances in which CPLEX fails to solve the problem to optimality within the time limit are presented as “3600”.

According to the results presented in Table 5-2, most of the instances with up to 10 jobs are solvable in less than one hour; however, the solution time grows substantially by increasing the size of the problem. Therefore, we predict that our method will fail to solve most of the instances with 10 jobs or more in a one-hour time frame. Another
point worth noting here is that in all the instances that have been solved to optimality, the upper bound generated in the preprocessing step is optimal. This result implies that the upper-bounding algorithm presented in Section 5.1.1 is likely to provide a tight upper bound, even for larger instances.

To further evaluate the quality of our upper-bounding algorithm we randomly generate 20 other instances, each having 10 jobs, using the method presented in Section 5.1.3. As mentioned before, several of these instances might not be solved to optimality within one hour using our exact solution method, but we expect to solve some of them. Comparing the optimal objective of these relatively large instances to our upper bound will then provide a better evaluation of the upper bound quality. Table 5-3 presents the results of this study.

Table 5-3. Evaluation of exact solution method \((n = 10)\)

<table>
<thead>
<tr>
<th>Instance</th>
<th>M</th>
<th>K</th>
<th>UB</th>
<th>LB ≥ 1</th>
<th>Total time</th>
<th>Nodes</th>
<th>CPLEX LB</th>
<th>CPLEX UB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>4</td>
<td>1.170</td>
<td>4</td>
<td>1869</td>
<td>3600</td>
<td>8975</td>
<td>1.500</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0.114</td>
<td>2</td>
<td>82</td>
<td>143</td>
<td>3303</td>
<td>1.002</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0.185</td>
<td>2</td>
<td>208</td>
<td>214</td>
<td>4226</td>
<td>1.004</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
<td>1.606</td>
<td>4</td>
<td>572</td>
<td>3600</td>
<td>17422</td>
<td>1.043</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>3</td>
<td>1.444</td>
<td>3</td>
<td>670</td>
<td>3600</td>
<td>8043</td>
<td>1.526</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>3</td>
<td>1.780</td>
<td>3</td>
<td>484</td>
<td>3600</td>
<td>21624</td>
<td>1.213</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>2</td>
<td>1.442</td>
<td>2</td>
<td>622</td>
<td>2235</td>
<td>5096</td>
<td>1.003</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>3</td>
<td>1.138</td>
<td>3</td>
<td>424</td>
<td>3600</td>
<td>15645</td>
<td>1.625</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>3</td>
<td>1.219</td>
<td>3</td>
<td>356</td>
<td>3600</td>
<td>45928</td>
<td>1.070</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>2</td>
<td>0.760</td>
<td>2</td>
<td>250</td>
<td>363</td>
<td>6664</td>
<td>1.001</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>6</td>
<td>0.682</td>
<td>6</td>
<td>876</td>
<td>3600</td>
<td>6796</td>
<td>2.078</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>4</td>
<td>1.230</td>
<td>4</td>
<td>1221</td>
<td>3600</td>
<td>7810</td>
<td>1.117</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>3</td>
<td>0.009</td>
<td>3</td>
<td>114</td>
<td>3600</td>
<td>57742</td>
<td>1.778</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>3</td>
<td>1.555</td>
<td>3</td>
<td>558</td>
<td>3600</td>
<td>18182</td>
<td>1.055</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>4</td>
<td>0.687</td>
<td>4</td>
<td>747</td>
<td>3600</td>
<td>6832</td>
<td>1.277</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>2</td>
<td>0.181</td>
<td>2</td>
<td>673</td>
<td>718</td>
<td>5634</td>
<td>1.001</td>
</tr>
<tr>
<td>17</td>
<td>6</td>
<td>2</td>
<td>0.880</td>
<td>2</td>
<td>780</td>
<td>1322</td>
<td>6061</td>
<td>1.002</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>2</td>
<td>1.969</td>
<td>2</td>
<td>84</td>
<td>955</td>
<td>99563</td>
<td>1.001</td>
</tr>
<tr>
<td>19</td>
<td>6</td>
<td>3</td>
<td>1.837</td>
<td>3</td>
<td>523</td>
<td>3600</td>
<td>18592</td>
<td>1.051</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>3</td>
<td>0.443</td>
<td>3</td>
<td>437</td>
<td>3600</td>
<td>22884</td>
<td>1.333</td>
</tr>
</tbody>
</table>

The columns in Table 5-3 are similar to the ones in Table 5-2, except that Column “n” is omitted for brevity \((n = 10\) for all instances) and Column “Time” is substituted by
two columns: “LB ≥ 1”, which reports the time (seconds) it takes for the solver to find the first lower bound value of greater or equal to 1, and “Total time”, which contains the total solution time in seconds, similar to “Time” in Table 5-2.

According to the results obtained in Table 5-3, the only instances that are solved to optimality within one hour are the ones having the optimal value of 2. This implies that lower bound improvement happens very slowly and in all instances except one, its value lies in the interval of (1, 2) even after one hour; therefore, we reach optimality only in instances having upper-bound value of 2. It is also worth noting that our initial upper bound has not been improved in any of the instances, and that it is equal to the optimal value in all instances that have been solved to optimality. This result once again implies that our upper-bounding algorithm creates strong bounds for the problem even for larger instances.

As the last step of our research we investigate whether the values generated by our upper-bounding algorithm are locally optimal in the instances in which CPLEX fails to find the optimal solution within the time limit. To that end, we conduct 2-opt local search [23] for all such instances by swapping every pair of jobs in the sequence generated by our upper-bounding algorithm and calculating the maximum number of late jobs corresponding to each resulting sequence using the dynamic programming algorithm presented in Section 4.3.4.1. We observe that for each instance the maximum number of late jobs corresponding to every swapped sequence is at least as large as the upper-bound value for that instance. The result of this experiment implies that no improvement can be made by swapping job pairs in any of the instances and therefore our upper-bounding algorithm tends to be locally optimal.

**Remark 5.1.** Although the upper-bounding algorithm presented in Section 5.1.1 succeeds in finding an optimal solution in every instance that is solved to optimality in our experiments, and is locally optimum in other instances, we cannot conclude local optimality of this algorithm. Indeed this algorithm fails to find the optimal solution in
some other instances. For example, consider a three-job instance where the processing times are $p_1 = 2$, $p_2 = 10$, and $p_3 = 3$, and the due dates are $d_1 = 3$, $d_2 = 19$, and $d_3 = 19$. Also, let $K = M = 1$. The upper-bounding algorithm presented in Section 5.1.1 generates the sequence 3, 1, 2 (or 3, 2, 1) since both jobs 1 and 2 are rejected in Step 2 of the algorithm and are scheduled at the end of the sequence. This sequence results in two late jobs, if we delay job 2. However, the optimal sequence is 1, 3, 2 which contains no more than one late job, under any delay scenario. Moreover, sequence 3, 1, 2 is not locally optimal, since swapping the first two jobs results in an improved (and globally optimal) solution.
CHAPTER 6
CONCLUSIONS AND FUTURE RESEARCH

In Chapter 2 we considered a multicommodity network flow problem in which the probability of successful flow transmission through each hub node decreases as the flow passing through the hub increases. Our computational results led us to conclude that a powerful general-purpose solver, such as GloMIQO, is an appropriate tool for solving problem instances that are relatively small, if the nodal reliability functions are quadratic. However, for more difficult instances, or for instances that cannot be formulated as mixed-integer quadratic programs, we recommended the use of our cutting-plane procedure.

Future research in this area may focus on generating heuristic methods to improve the optimality gap for the instances that are not solvable in a reasonable time frame using our exact cutting-plane algorithm. Additionally, a key assumption in our model is that flows throughout the network are possibly ruined in transit, but are still passed along to the destination after they have been ruined. An alternative model may examine the case in which inspection of the cargo being shipped takes place at each node. If the cargo is found to be damaged, it can be immediately discarded instead of being forwarded onto its destination. As a result, the probabilities of successfully shipping commodity flows changes in this case, since the unsuccessful shipment of one commodity may now improve the odds that another commodity is successfully shipped. Other variations may consider the case in which flows can be misdirected, or when split flows are allowed. Each of these variations would require different approaches than the ones prescribed here.

In Chapter 3 we studied the taxonomy of the robust single-machine scheduling problems and presented a complete survey in this area. This study led us to conclude that despite the extensive use of discrete scenarios and independent continuous intervals in representing uncertain data in the literature of robust single-machine
scheduling, certain shortcomings exist in both representations. On one hand, it is not always easy to determine the exact possible scenarios for uncertain parameters in the problem, especially when the number of possible values is very large. On the other hand, in case of representing data as independent intervals of uncertainty (Soyster’s method), correlations among parameter values are not addressed and the obtained worst-case scenarios are very unlikely to happen in practice. Therefore, using methods similar to the ones suggested in [16, 77] to control the level of conservatism in scheduling problems with uncertain data is a promising area of future research.

In Chapter 4 we applied state-of-the-art robust optimization methods to define and solve a single-machine scheduling problem with uncertain job processing times, under four alternative optimization criteria. For each problem, the goal was to find a schedule that minimizes the worst-case objective function value. We assumed that job processing times can be represented using independent continuous intervals. We then moderated the level of conservatism in the problem by confining processing time values to belong to three alternative uncertainty sets. For each problem (distinguished by its optimization criterion and the uncertainty set), we studied the difficulty of the scenario-generation problem and the resulting robust optimization problem.

Results presented in Table 4-1 imply that although the problem of finding a worst-case scenario is polynomially solvable for almost every problem (expect for the problem of minimizing number of late jobs under uncertainty set 3, whose complexity remains unknown), their robust counter-parts are not obviously solvable in polynomial time (e.g., for the problem of minimizing weighted sum of completion times under all three uncertainty sets and for the problem of minimizing number of late jobs, under uncertainty sets 2 and 3). For solving the robust problems with a known worst-case scenario generation complexity, we either developed optimal algorithms, or derived a mixed-integer programming model. Future research in this area can be devoted to exploring the complexity of open problems introduced in this chapter.
In Chapter 5 we study the problem of minimizing number of late jobs in robust single-machine scheduling problem under uncertainty set 2, as defined in Chapter 4. We first present upper- and lower-bounding heuristic algorithms for the problem and evaluate them by solving a set of randomly-generated instances using each heuristic and comparing the bounds. We then implement the single-stage mixed-integer programming formulation presented for this problem in Chapter 4 by solving a set of random instances using CPLEX.

According to the results obtained in Chapter 5, instances with less than 10 jobs are likely to be solved using our mixed-integer programming formulation (4–87)–(4–96) in less than an hour. Moreover, the upper-bounding scheme presented in this chapter tends to generate good-quality solutions for all instances and can be used as a good heuristic algorithm for the problem. As a future research in this area, an effort to develop efficient exact algorithms for solving larger instances of the problem would be of interest. Also, tighter lower bounds can help reduce the gap and/or prove optimality in some instances.
REFERENCES


BIOGRAPHICAL SKETCH

Bita Tadayon was born in 1986 in Iran. She received her bachelor's degree in 2008 and her master's degree in 2010 from the Department of Industrial Engineering at Sharif University of Technology, Tehran, Iran. In August 2010, she joined the Ph.D. program in the Department of Industrial and Systems Engineering at the University of Florida. Her research interests lie in different areas of operations research, including integer programming, network optimization, decomposition approaches to large-scale optimization problems, and robust optimization. She received her Doctor of Philosophy degree in industrial and systems engineering from the University of Florida in the spring of 2014 and started her job as a Senior Operations Research Analyst at Sabre Airline Solutions in the summer of 2014.