Scheduling Mitigations to Protect Critical Infrastructure under Resource Constraints

by

Ashley Peper

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Industrial and Systems Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2025

Date of final oral examination: 07/24/2025

The dissertation is approved by the following members of the Final Oral Committee:

James Luedkte, Professor, Industrial and Systems Engineering
Laura Albert, Professor, Industrial and Systems Engineering
Jeffrey Linderoth, Professor, Industrial and Systems Engineering
Michael Ferris, Professor, Computer Science

ACKNOWLEDGMENTS

I would first and foremost like to thank my advisors, Laura Albert and Jim Luedtke, for their guidance and support throughout my time at UW-Madison. I much appreciated all of the advice in research, writing, and future planning. I would also like to thank my other committee members, Jeff Linderoth and Michael Ferris, for taking the time to participate in improving my dissertation. I have also appreciated the support of other UW industrial engineering staff for helping me with many questions throughout my PhD. I would finally like to thank all my peers at UW-Madison for their company along the way.

I am grateful and fortunate to have had the wonderful support of my parents, John and Tracy, and my sister, Nicholle, throughout my time at UW-Madison. Particularly in helping me with responsibilities outside of my PhD, giving me more mental capacity to focus on writing a dissertation. I additionally appreciate my boyfriend, Eric, for helping me with every struggle and indecisive moment I had throughout my PhD, and especially for his unwavering support during these busiest last few months.

I lastly would like to thank my undergraduate professors at UWSP for encouraging me to pursue a PhD. In particular, Andy Felt for his continued support, including helping me find my PhD program at UW-Madison and, most importantly, for instilling in me a love of optimization.

This work was in part supported by the National Science Foundation Award 2000986. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This dissertation studies the problem of investing in security mitigations to protect a critical infrastructure system. We consider three different ways a system could be vulnerable to threats, and for each of these cases spend a chapter developing methods to model the deployment of mitigations to optimally protect the system. There exists literature to address the problem of selecting a portfolio of mitigations subject to a budget constraint in each of these cases. However, the literature does not address how to deploy the mitigations in resource-constrained settings over time. We consider the problem of choosing a portfolio of mitigations to implement while considering the time and resources required to implement the mitigations and precedence relationships between the mitigations. The first way we model threats to the system addresses overlapping effects of mitigations on different vulnerabilities of the system, allowing each vulnerability to be covered multiple times for diminishing returns. In the next chapter we consider the possibility of sophisticated attackers, rather than general vulnerabilities. We assume a set of attackers simultaneously work to complete their own project, where the actions they must take to complete these projects can be delayed by the defender's mitigations. The final way we model threats to the system is again by considering sophisticated attackers, but instead assuming the attacker have multiple ways to achieve their goal. While similar to the previous problem, this problem presents its own unique modeling challenges. In each case we introduce integer programming models that extend a resource constrained project scheduling problem (RCPSP) to both select and schedule mitigation tasks over a time horizon with the goal of maximally protecting the system. We additionally develop heuristic methods for each of these problems. We conclude by discussing the benefit we see in this research of considering the deployment of mitigations to protect critical infrastructure systems in resource-constrained settings over time.

1 INTRODUCTION

Critical infrastructure systems are prevalent throughout the world, and parts of these systems may be susceptible to attack by sophisticated adversaries, or simply become damaged, old, or otherwise in need of repair or replacement. Failure or exploitation of such vulnerable parts of the system may cause costly disruptions to the system, unintended use of the system, or other undesirable impacts. There are many examples of critical infrastructure systems that can benefit from the consideration of how to deploy security measures to protect them. Power systems may be protected by reinforcing or replacing components of a power grid to help avoid costly outages, transportation systems may be protected by replacing road infrastructure to improve traffic flow or making improvements to public transit options to improve safety and user satisfaction, and information technology systems may be protected by updating software or implementing security mechanisms to avoid data breaches or system down-time. This last example falls into the realm of cybersecurity, which is present in many areas of critical infrastructure given the prevalence of computer based control of modern infrastructure. Enayaty-Ahangar et al. (2020) provide an overview of papers that deal with optimization in cybersecurity. The proactive selection of security measures, or "mitigations", to decrease risks of attack or other vulnerabilities in a cybersystem, is one example of decisionmaking to secure critical infrastructure. However, as noted in Albert (2023), what is lacking in existing models is a consideration of the effort required to enact the plans they choose. While the selection of mitigations is a well studied area, planning the deployment of mitigations is a topic that remains unstudied. This remains true for critical infrastructure systems in general. While some aspect of timing may be considered in some models, e.g., (Borrero et al., 2016; Malaviya et al., 2012; Zheng and Castañón, 2012; Zhuo and Solak, 2014), there is a lack of models that consider resource based deployment over time of mitigations. A portfolio of mitigations chosen by one of these existing models may appear to be the best choice for protecting the system, however these mitigations may be difficult to implement simultaneously, resulting in delays in the completion of all mitigations. The time the system is vulnerable while mitigations are being implemented is worth considering. For this reason, we integrate a resource constrained project scheduling problem (RCPSP) to schedule mitigations into existing methods of mitigation selection.

In Chapter 2, we begin with a simple model of the system, where we model vulnerabilities in the system as a set of nodes that can be covered by mitigations. This extends a simple version of the problem proposed by Zheng et al. (2019). While their paper gets more complex by adding stochasticity to the problem, our more simple problem becomes difficult to solve via the addition of an RCPSP to the mitigation decisions. We analyze the benefit of integration of RCPSP into this model by comparing to preexisting methods that do not integrate scheduling and coverage as we do. We additionally develop heuristic methods to solve this problem more efficiently. We adapt an interval-based relaxation of Carrasco et al. (2022) to our problem as a fast heuristic method, and then use this as a baseline of a rolling horizon heuristic that provides good solutions in a scalable way. In computational studies we find that the integrated model has benefit over preexisting methods as the proposed alternative methods provide heuristic solutions with an average optimality gap of around 7%. Additionally the rolling horizon heuristic we present outperforms all other heuristic methods considered, finding the best solution among all methods given a 30 minute time limit on 33 of 35 larger instances.

In Chapter 3 we move from modeling threats as general vulnerabilities to more sophisticated attackers. These attackers are working to complete various attack projects and the defender can implement mitigations to delay steps of those projects. A example of this type of attacker model is that of Brown et al. (2009), where a defender is trying to delay steps in an adversary's nuclear arms project. We more specifically extend the problem proposed in Zheng and Albert (2019) that considers a more general set of attackers each completing projects. In a mitigation selection models such as these, the assumption is made that the defender acts first. However, in reality, mitigations take time to implement, and attackers may already be working to complete their projects in the meantime. The consideration of the defender's scheduling decisions becomes especially important in this context, as the timing of when mitigations are completed affects whether or not they are even useful. If the adversary has already completed an action, delaying that action has no benefit. This is something that is not fully considered in Chapter 2. Thus in Chapter 3 we address this interplay of timing between the completion of attacker and defender projects. We develop an integrated integer programming model of this bilevel problem by using a time-indexed network that allows us to keep track of attackers' arc lengths over time as determined by the defender's choices. We simplify this time-indexed network to reformulate the model

to be easier to solve and improve linear programming relaxation bounds. We present additional heuristic methods that allow us to analyze the benefit of this integrated model over models that either disregard the attacker network or assume the defender acts first. In computational experiments we find that our reformulations greatly improve the solvability of the model. In addition, we see that our heuristic methods average a 5-10% optimality gap, demonstrating a benefit to the integrated model.

In Chapter 4 we consider a third model of cybersecurity threats that we can integrate the idea of scheduling into. Similar to the previous chapter, we consider more sophisticated attacks, however here we look at an attacker choosing the fastest method of attack via a shortest path along an attack graph. Mitigations again affect the edges of the attacker graph, increasing the time it takes for the attacker to complete an action. This is an example of a shortest path network interdiction problem as seen in papers such as Israeli and Wood (2002). This is a common way to model attacks in areas such as cybersecurity, where an attacker needs to only find a single way to breach into the system. We can use similar techniques to model this as in the previous chapter, however this problem becomes more difficult to model since its natural structure is that of a max-min problem. In this chapter we develop a model for this problem and develop a similar simpler model to approximately solve it. Our computational experiments demonstrate that this model suffers from the same computational difficulties that are seen in shortest path interdiction problems such as in Israeli and Wood (2002), while also being more complex in the defender's problem. We find that our proposed approximate model generally finds optimal or near optimal solutions. This shows that a full consideration of simultaneous scheduling between the attacker and defender may not be necessary, and a model that partially considers this may be sufficient in most cases.

Finally, in Chapter 5, we compare the models developed in all three chapters. We see that each model has similar properties, however each are useful in different cases. While a more general model like in 2 may be useful, considering more specific types of attacks as in 3 and 4 may be more appropriate in certain situations. We analyze the differences in solutions for each of these models on a single example, adapted to each model type. We conclude by discussing future directions for research in this area.

2 SELECTING AND SCHEDULING MITIGATIONS UNDER RESOURCE CONSTRAINTS

2.1 Introduction

Critical infrastructure systems are prevalent throughout the world, and parts of these systems may be susceptible to attack by sophisticated adversaries, or simply become damaged, old, or otherwise in need of repair or replacement. Failure or exploitation of such vulnerable parts of the system may cause costly disruptions to the system, unintended use of the system, or other undesirable impacts. We study the problem of a manager of a critical infrastructure system who is aware of the various vulnerabilities of the system, and the actions that can be taken to mitigate those vulnerabilities. Due to limited resources, the system manager may not be able to implement all of the mitigation options. Papers such as Zheng et al. (2019) address this, but none have considered the resources required over time to deploy these mitigations. As a result, it may not be possible to implement all of the chosen mitigations simultaneously and this can mean some important selected mitigations end up being implemented later than desired, leaving the system vulnerable in the meantime. To address this challenge, we investigate the problem of selecting *and scheduling* mitigations to best protect a system of critical infrastructure while considering resource constraints over time and precedence relations between the mitigation tasks.

There are many examples of critical infrastructure systems that can benefit from the proposed model. Power systems may be protected by reinforcing or replacing components of a power grid to help avoid costly outages, transportation systems may be protected by replacing road infrastructure to improve traffic flow or making improvements to public transit options to improve safety and user satisfaction, and information technology systems may be protected by updating software or implementing security mechanisms to avoid data breaches or system down-time. This last example falls into the realm of cybersecurity, which is present in many areas of critical infrastructure given the prevalence of computer based control of modern infrastructure. As may then be expected, cybersecurity is frequently used as the main application in the literature of mitigation selection.

In this chapter , we address the problem of choosing and scheduling mitigations that

provide protection to a system. We assume the manager of the system is aware of areas in the system that may be vulnerable to attack or cause other potential problems in the system. We denote these as the vulnerability “nodes" of the system. When the system’s manager implements a mitigation, it reduces the vulnerability of various nodes, thereby providing “coverage" to those nodes. In more basic node coverage applications, this may be a binary notion: a node can either be covered or not. We consider the more general notion of “multiple coverage", that is, we can continue to reduce the vulnerability of a node by implementing multiple mitigations that affect it. Covering a vulnerability that has already been covered by other mitigations may be less useful, so we assume there are diminishing returns for individual node coverage.

We also address resource usage in implementing these mitigations. Similar to previous models of mitigation selection, we assume there is an overall fixed cost to implement mitigations, so we include an overall maximum budget for the selected mitigations. However, we also consider time-dependent resources that affect when we can complete each mitigation. Such resources may include labor for various personnel types, computational resources, or additional time-dependent monetary resources. Additionally, we allow mitigations to be composed of a set of jobs that reflect the multiple requirements to deploy a mitigation. We assume implementing a mitigation uses resources for given amount of time, and we want to schedule the selected mitigations—and its associated jobs—over a discrete time horizon to maximize coverage of vulnerabilities while also considering the desire to have coverage as early as possible.

One key component of the problem we present is the selection of mitigations to cover vulnerabilities; an area which researchers have studied in multiple contexts. This is a case of a maximum coverage problem subject to a budget constraint as is presented in Khuller et al. (1999). This presents a base model, which other literature in this area extend. Zheng et al. (2019) explore the context of multiple coverage, which allows vulnerabilities to be covered multiple times, where better objective values can be attained with more coverage. They develop integer programming formulations and approximation algorithms for a few variations of this multiple coverage mitigation selection problem, utilizing the multiple coverage structure to also address the possibility of a mitigation failing. They consider vulnerabilities in a set of attack paths, where multiple mitigations can cover each vulnerability in each attack path. A non-decreasing concave function of the coverage of a

path is then used in the objective, summed over all paths, to model the benefit of multiple coverage. Zhuo and Solak (2014) consider a multiple coverage cybersecurity investment problem that considers the potential uncertainty in mitigation effectiveness by using a two-stage stochastic programming model. In this stochastic model, the effectiveness over time of potential investments varies by scenario. While this approach addresses some time aspects of mitigation deployment, it does not address the scheduling implications of limited resources and precedence constraints. Khouzani et al. (2019) develop deterministic and stochastic models to select mitigations to minimize attacker success based on probabilistic attack graphs. Schmidt et al. (2021) and Zheng and Albert (2019) also address problems in the area of mitigation selection with integer programming techniques, considering ideas of achieving a threshold difficulty level or maximally delaying attackers on attack paths and graphs, respectively. Schilling and Werners (2016) develop a model for mitigation selection based on data from common IT security practices and standards. Numerous additional related applications of optimization to cybersecurity exist in the literature, as are presented in Enayaty-Ahangar et al. (2020). Various papers consider other specific threat modeling techniques and how to invest in security measures to minimize those threats (Vorobeychik and Letchford, 2015; Fila and Wideł, 2020; Nagurney and Shukla, 2017) However, while several methods exist to select mitigations to improve security under different goals and attacker assumptions, there is a lack of literature that considers the deployment of these mitigations over time, particularly in resource constrained settings (Albert, 2023). Malaviya et al. (2012) develop a multiperiod interdiction model that considers resource usage over a series of time periods. While this model addresses the timing aspect of resource investment into security, it is tailored to the network interdiction structure while we consider coverage of vulnerabilities.

A class of problems that can address this notion of mitigation deployment over time is the Resource Constrained Project Scheduling Problem (RCPSP). The RCPSP involves a series of jobs that must be scheduled, where each job utilizes resources over time and resources are available over time. The jobs must then be scheduled in a way that satisfies these resource constraints as well as precedence relations between the jobs. Each job takes a set amount of time to complete, and the goal is to find a schedule that finishes the last job as early as possible. Originally modeled as a binary integer programming problem (Pritsker et al., 1969), the RCPSP is commonly studied in the integer programming context, e.g.,

(Artigues et al., 2013; Talbot and Patterson, 1978; Mingozzi et al., 1998; Hill et al., 2022). As this is an NP hard problem, many heuristic solutions have been developed for this problem and its variants e.g., (Kolisch and Hartmann, 1999, 2006; Carrasco et al., 2022). Hartmann and Briskorn (2022) provide a survey of the current variants and extensions of the RCPSP. Traditionally, the RCPSP considers an objective that minimizes the makespan of the project. This is quite different from our objective that maximizes coverage of vulnerabilities over time. There are a few variants that have alternate objectives. One that is more comparable to our problem is the RCPSP with net present value (NPV) objective (Yang et al., 1993; Vanhoucke et al., 2001). Scheduling problems such as the RCPSP are often solved using constraint programming techniques, e.g., (Liess and Michelon, 2008; Berthold et al., 2010). Schutt et al. (2012) develop techniques for solving an RCPSP with NPV objective with constraint programming methods. Integer programming methods are also commonly used for this type of RCPSP. Most related to our work, Carrasco et al. (2022) provide a heuristic approach to an RCPSP with an NPV objective, where a smaller integer program is developed by combining time periods into a much smaller set of time "intervals". In the setting where resources are cumulative and can be used in future time periods, they can use a solution to this smaller model to build a true solution with an approximation guarantee. While we could try to approximate a solution using a RCPSP model with a time-discounted objective, it does not allow us to model an objective of diminishing returns for multiple coverage of nodes. By including the effect of diminishing returns for node coverage, we deviate from an RCPSP in that the effect of completing a job is dependent on what other jobs have been completed. Our model extends beyond the currently existing framework of the RCPSP in order to achieve this goal. We note, however, that we make use of ideas developed by Carrasco et al. (2022) to derive an effective heuristic for our model.

The first contribution of this chapter is to introduce a new integer programming model (in Section 2.2) that combines a vulnerability coverage problem with an RCPSP, in order to find a mitigation implementation schedule that maximizes coverage over time. This model adds complexity to existing mitigation selection models by considering timing and resource constraints when selecting mitigations. In addition, it expands upon existing RCPSP literature by considering a more complex objective that considers the impact completed jobs have on covering vulnerabilities, and the diminishing returns of multiple coverage. Since our problem can be difficult to solve exactly, in Section 2.3 we propose heuristic

methods to find good approximate solutions to this problem more efficiently. Here we adapt an existing heuristic method to our problem, as well as develop a new rolling horizon heuristic that repeatedly solves an updated version of the underlying approximate model. In Section 2.4 we present results of a computational study to demonstrate the benefit of our model and the quality of the solutions generated by the proposed heuristics. In Appendix A.1, we provide a description of alternate methods to solve this problem that do not combine selection and scheduling in a single model, which we use as benchmarks demonstrate the benefit of our model. We find that integrating scheduling and mitigation selection leads to solutions that improve by 7% on average over methods that either consider mitigation selection without considering scheduling, or consider scheduling without considering the overlapping impacts of the selected mitigations. In addition, for larger problems our proposed heuristic technique enables finding high-quality solutions quickly, and in particular yields solutions to the proposed IP model that are better than those obtained by a general-purpose MIP solver in a similar amount of time.

2.2 Problem Description and Integer Programming Formulation

Problem Description

We consider the implementation of a set of potential mitigations $j \in J$ to provide protection against a set of vulnerabilities $n \in \mathcal{N}$. Following the literature on resource constrained project scheduling, we will interchangeably refer to mitigations as jobs. These mitigations/jobs are deployed over a discrete time planning period consisting of T periods, $\mathcal{T} = \{1, ..., T\}$. When a job $j \in J$ is completed it provides $w_{jn} \geqslant 0$ units of coverage of vulnerability $n \in \mathcal{N}$. Some jobs $j \in J$ might not provide coverage to any vulnerability $n \in \mathcal{N}$ (i.e., $w_{jn} = 0$ for all $n \in \mathcal{N}$) but may need to be completed before other jobs that do provide coverage can be performed. To model this structure, we define a set of job precedences P, where $(i, j) \in P$ implies that job i must be completed before job j can be started. A job $j \in J$ has a duration of $\tau_j \in \mathbb{Z}_+$ time periods. While job $j \in J$ is in process, it requires c_{jr} units of resource $r \in R$, where R is the set of renewable resources. We define b_{rt} as the amount of resource $r \in R$ available in time period $t \in \mathcal{T}$. If selected, job $j \in J$ also consumes C_m units

of a non-renewable resource, for which the total available budget is B units.

We desire to maximize time-weighted coverage, adjusted for the diminishing returns of covering a node multiple times. We model this using piecewise linear "coverage reward" functions f_n that take as input the amount of coverage of node $n \in \mathcal{N}$ and output the reward for covering the node at that level. The function f_n is assumed to be concave, reflecting the potential diminishing returns of multiple coverage. We also define a time-weighting parameter $\alpha_t \geqslant 0$, that represents the value of having coverage in time period $t \in \mathcal{T}$. We discuss some specific options for these time-weights α_t in Section 2.2.

Table 2.1: Sets and parameters used to describe a problem instance and decision variables for the IP model.

Sets	
$\mathcal{T}$	set of time periods, $\{1, ..., T\}$
$\mathcal{N}$	set of nodes
J	set of jobs
R	set of resources
P	set of precedence relations
Parameters	
w_{jn}	coverage of node $n \in \mathcal{N}$ gained by completing job $j \in J$
τ_j	time required to complete job $j \in J$
c_{jr}	usage of resource $r \in R$ when job $j \in J$ is in process
C_m	cost of job $j \in J$ for cumulative monetary budget
b_{rt}	availability of resource $r \in R$ in time period $t \in \mathcal{T}$
B	cumulative monetary budget
α_t	time-weighting parameter for objective function for period $t \in \mathcal{T}$
f_n	piecewise linear function that takes in coverage for node $n \in \mathcal{N}$, and outputs an objective value adjusted for diminishing returns
Decision Variables	
z_{nt}	amount of coverage for node $n \in \mathcal{N}$ at time $t \in \mathcal{T}$
x_{jt}	binary indicator for if job $j \in J$ finishes at time $t \in \mathcal{T}$

Integer Programming Formulation

We formulate this problem as an integer program. Variables x_{jt} determine at which time period t a job $j \in J$ is completed, with $x_{jt} = 1$ if job j is completed in time period t and $x_{jt} = 0$ otherwise. Variables z_{nt} indicate the amount of coverage node $n \in \mathcal{N}$ has at time $t \in \mathcal{T}$.

These variables have the property that $z_{nt} \geqslant z_{n,t-1}$, in that, once node n gains coverage from the completion of a job, it has that coverage for the rest of the time horizon. For the objective, these z_{nt} variables are then used as input into the piecewise linear functions $f_n(z_{nt})$ for each $n \in \mathcal{N}$. Table 2.1 provides a summary of the sets, parameters, and variables described in this section.

The integer programming formulation is then:

$$y = \max \sum_{t \in \mathcal{T}} \sum_{n \in \mathcal{N}} \alpha_t f_n(z_{nt}) \tag{2.1a}$$

$$\text{s.t.} \sum_{j \in J} \sum_{s=t}^{t+\tau_j - 1} c_{jr} x_{js} \leqslant b_{rt} \qquad \forall t \in \mathcal{T},\ r \in R \tag{2.1b}$$

$$\sum_{t \in \mathcal{T}} \sum_{j \in J} C_m x_{jt} \leqslant B \tag{2.1c}$$

$$\sum_{s=1}^{t} x_{js} \leqslant \sum_{s=1}^{t-\tau_j} x_{is} \qquad \forall (i,j) \in P,\ t \in \mathcal{T} \tag{2.1d}$$

$$\sum_{t \in \mathcal{T}} x_{jt} \leqslant 1 \qquad \forall j \in J \tag{2.1e}$$

$$x_{jt} = 0 \qquad \forall j \in J, t = 1, .., \tau_j - 1 \tag{2.1f}$$

$$z_{nt} = \sum_{j \in J} w_{jn} x_{jt} + z_{n,t-1} \qquad \forall t \in \mathcal{T},\ n \in \mathcal{N} \tag{2.1g}$$

$$x_{jt} \in \{0, 1\} \qquad \forall j \in J,\ t \in \mathcal{T} \tag{2.1h}$$

$$z_{nt} \geqslant 0 \qquad \forall n \in \mathcal{N},\ t \in \mathcal{T}, \tag{2.1i}$$

where we define $z_{n0} := 0$ for $n \in \mathcal{N}$. The objective (2.1a) maximizes the total coverage rewards across nodes and time periods. Constraints (2.1b) and (2.1c) give the budget constraint for each time period for the renewable resources and the overall budget for the non-renewable monetary resource, respectively. Constraint set (2.1d) enforces the precedence constraints. Constraint set (2.1e) enforces that each job may be completed at most once and constraints (2.1f) enforces that a job cannot be completed earlier than the amount of time needed to complete it. Constraints (2.1g) set the variables z_{nt} by adding the weighted coverage gained from jobs completed in period t to the coverage for node n from the previous time period. Lastly, constraints sets (2.1h) and (2.1i) define the binary

and non-negativity restrictions of the decision variables.

Objective Function Time Weighting

The objective function (2.1a) weights by time with parameters α_t for $t \in \mathcal{T}$. We thus consider how to define α_t, for $t \in \mathcal{T}$, to achieve a desired time-weighting.

In the objective (2.1a), since $z_{nt} \geqslant z_{n,t-1}$ for all t and f_n is a nondecreasing function, it appears that coverage is rewarded repeatedly once earned. In this way, we implicitly weight by time. Alternatively, we can try to achieve more control over the time weighting by using an objective that only achieves a reward from a job for the period in which it is completed. Thus we evaluate rewards based on how much coverage was gained in each time period after adjusting for diminishing returns. This yields the following alternative form of the objective,

$$\sum_{n\in\mathcal{N}} \sum_{t=1}^{T} a_t \big(f_n(z_{nt}) - f_n(z_{n,t-1})\big), \tag{2.2}$$

where the time-weighting coefficients $a_t \geqslant 0$ give the value of gaining coverage in period $t \in \mathcal{T}$. Observe that if we define $a_{T+1} = 0$ and then set $\alpha_t = a_t - a_{t+1}$ for $t \in \mathcal{T}$, we can recover the original objective function (2.1a). Indeed we have:

$$\begin{aligned}
&\sum_{n\in\mathcal{N}} \sum_{t=1}^{T} a_t \big(f_n(z_{nt}) - f_n(z_{n,t-1})\big) \\
&\qquad = \sum_{n\in\mathcal{N}} \Big(\sum_{t=1}^{T} a_t f_n(z_{nt}) - \sum_{t=1}^{T} a_t f_n(z_{n,t-1}) \Big) \\
&\qquad = \sum_{n\in\mathcal{N}} \Big(\sum_{t=1}^{T} a_t f_n(z_{nt}) - \sum_{t=0}^{T-1} a_{t+1} f_n(z_{nt}) \Big) \\
&\qquad = \sum_{n\in\mathcal{N}} \sum_{t=1}^{T} (a_t - a_{t+1}) f_n(z_{nt}) + \sum_{n\in\mathcal{N}} a_1 f_n(0).
\end{aligned} \tag{2.3}$$

The final equality holds because $z_{n0} = 0$. Since $\sum_{n\in\mathcal{N}} a_1 f_n(0)$ is a constant, it can be dropped from the objective function. We can see then that (2.3) fits the form of objective function (2.1a), with $\alpha_t = a_t - a_{t+1}$. Note that since in (2.2) we only receive a reward for coverage in the period it is gained, to promote completing jobs earlier we would set

$a_t \geqslant a_{t+1}$ for each $t \in \mathcal{T}$. This is consistent with our assumption that $\alpha_t \geqslant 0$ for each $t \in \mathcal{T}$.

This alternative view of the objective function can provide insight into how to choose the weights α_t, by first choosing a_t and defining α_t accordingly. We discuss three options to model a variety of situations.

- Exponential decay: $a_t = \gamma^t$ for some $\gamma < 1$, $t \in \mathcal{T}$: This represents the case where priority is placed on earlier periods via some discount factor. Plugging this into (2.1a) yields $\alpha_t = (1-\gamma)\gamma^t$, for $t = 1, ..., T-1$, and $\alpha_T = \gamma^T$. Note this difference in the last term comes from the definition $a_{T+1} = 0$. This difference is logical in that while we desire to complete mitigations earlier in the horizon, these mitigations will remain in effect after the horizon. This increase in the last term then represents the benefit of completing a mitigation at any point in the horizon.

- Duration-to-horizon reward: $a_t = (T - t + 1)$, $t \in \mathcal{T}$: In this model the reward is set according to how many remaining periods in the horizon the coverage is active for. This equates to setting $\alpha_t = 1$ for all $t \in \mathcal{T}$. Thus in this case, we add no explicit time-weighting to the objective (2.1a), but there is implicit time-weighting gained from mitigations that are in effect for a longer duration.

- No time-weighting: $a_t = 1$, $t \in \mathcal{T}$: If timing of mitigation completion is not important, time weighting may not be needed at all. This could be the case with a very short time horizon, where all that matters is gaining as much coverage in the time horizon as possible. This is achieved by setting $a_t = 1$. That is, a reward is gained from the job when it is completed, and the time period it is completed in has no effect on that reward. This yields $\alpha_t = 0$ for $t = 1, ..., T-1$ and $\alpha_T = 1$.

In our analyses we use the first proposed option of exponential time weighting. As we illustrate in our computational study in Section 2.4, this works as a flexible method in which modifying the parameter γ can give more or less priority on the timing of mitigation completion.

2.3 Interval Based Heuristics

As we see in our computational study in Section 2.4, for large instances, it can take a long time to find a high-quality solution to the model (2.1) using a general purpose MIP solver. We thus investigate heuristic approaches to find good approximate solutions more efficiently for such large-scale instances. In the appendix, we adapt two existing models as methods to find heuristic solutions to (2.1). These methods consider scheduling and coverage separately, and they provide a benchmark for the quality of solutions attainable without modeling scheduling and coverage together. Here, we instead develop heuristic methods that simultaneously consider selection and scheduling, but in an approximate way by grouping time periods into a series of intervals.

Interval Based Relaxation

We begin by developing a model that groups time periods into intervals and approximates constraints in a way that provides a relaxation to the full model (2.1). This interval approach has been used previously by Hall et al. (1997) for job scheduling problems and adapted by Carrasco et al. (2022) to solve a version of an RCPSP. Carrasco et al. (2022) solve an RCPSP with an NPV objective, where resources are cumulative, i.e., unused resources from one time period carry over to subsequent periods. This shares many similarities with our problem and thus we use the interval-based reformulations that Carrasco et al. (2022) present as a basis for our interval-based model. Carrasco et al. (2022) find that their interval model is a relaxation of the RCPSP in consideration. By addressing a few differences in our problem from theirs, we also develop an interval model that provides a relaxation for our model (2.1). In addition Carrasco et al. (2022) utilize a heuristic algorithm for scheduling jobs based on the solution to their interval model, and find that this solution has an approximation bound. This approximation bound is specific to an RCPSP with cumulative resources, and cannot be applied to our case of non-cumulative resources. Nevertheless, the interval-based reformulation can provide a fast solution and we use it as the basis for our heuristic methods.

Let $\mathcal{K} = \{1, ..., K\}$ be the set of time intervals, and $\mathcal{T} = \{\hat{t}_k \ : \ k \in \mathcal{K}\}$ be the set of final periods for each interval. Then each interval $k \in \mathcal{K}$ consists of periods $[\hat{t}_{k-1} + 1, \hat{t}_k]$, where $\hat{t}_0 := 0$. For each $k \in \mathcal{K}$ let $\hat{T}_k = \hat{t}_k - \hat{t}_{k-1}$ be the length of interval k. We also define $\kappa(t)$ as

the interval containing period $t \in \mathcal{T}$. While any interval sizes can be used in this formulation, we use exponentially increasing sizes of intervals to decrease the size of the problems in a well-scaling way. This also is reasonable with the use of exponential weighting in our objective as we make intervals larger where the time periods carry less objective weight. To do so, let $\hat{T}_k = \lfloor \beta^{k-1} \rfloor$ for $k = 1, ..., K$, for some $\beta \geqslant 1$. Note, this definition of interval sizes differs slightly from Carrasco et al. (2022), where each interval *end period* is exponentially increasing, rather than each interval length exponentially increasing in our model. The two methods are similar and have the desirable qualities mentioned above. We use this alternative version to obtain parameter interpretability in that $\beta = 1$ gives the original time periods, so that the closer β is to 1, the closer the interval-based model is to the the original period-based model (2.1).

We can then develop the model using the set $\mathcal{K}$ of time intervals instead of the original set of time periods. As in Carrasco et al. (2022), our goal for this model is for it to be a relaxation of (2.1) in the sense that any feasible solution of (2.1) can be mapped to a solution of the interval model having equal or smaller cost. For use in algorithms, we parameterize this model by the set $\mathcal{T}$ of interval end times, as this determines the structure of the intervals used in the model. Let x_{jk} be a binary variable that equals 1 if job $j \in J$ is completed in interval $k \in \mathcal{K}$, and let z_{nk} be the amount of coverage gained for node n by the end of interval k. Most of the interval model is similar to the original model (2.1), where we simply replace time period indices with interval indices. However, the resource and precedence constraints require more attention.

For the precedence constraints, as is described in Carrasco et al. (2022), we must add more precedence relations to our existing set P. To understand why, suppose we imposed some precedences $(i, j), (j, \ell) \in P$, where each job takes two time periods to complete. We find an interval k with $\hat{T}_k = 3$ such that jobs i and j could both be finished in k and the precedence would hold, and additionally jobs j and ℓ could both be finished in k without violating the corresponding precedence constraint. However for interval k to contain all three of i, j, and k, it would need to be at least four periods long. Since $\hat{T}_k = 3$, this is not true, and all three jobs cannot be scheduled in this interval. Thus, considering only precedences in P allows infeasible solutions (2.1) to be feasible to the interval model. We address this with the same approach as Carrasco et al. (2022), in defining an extended precedence set $\hat{P}$ that contains all pairs of jobs (i, j) such that i is connected to, and precedes,

j on the graph induced by P. Then for each $(i, j) \in \hat{P}$, we define $\rho_{i,j}$ as the longest path from i to j on the graph defined by P, with the edge weights of this graph given by τ_ℓ for $(h, \ell) \in P$. Since this implies that for any pair of jobs $(i, j) \in \hat{P}$, job i must complete at least $\rho_{i,j}$ periods before job j, our new precedence constraints are as follows:

$$\sum_{\ell=1}^{k} x_{j\ell} \leqslant \sum_{\ell=1}^{\kappa(\hat{t}_k - \rho_{i,j})} x_{i\ell} \qquad \forall\, (i, j) \in \hat{P},\, k \in \mathcal{K}.$$

These constraints match the format given by Carrasco et al. (2022). These constraints fix the issue in the given example, since there would be a precedence relation $(i, \ell) \in \hat{P}$ that enforces that it be possible to have four periods in between these two jobs. Scheduling both jobs i and ℓ in an interval of length three would no longer be possible.

For the resource constraints, we can aggregate resource availability over time periods in each time interval (this is similar to the select-then schedule model presented in Appendix A.1). For each resource $r \in R$ and interval interval $k \in \mathcal{K}$, we have a budget of $\hat{b}_{rk} = \sum_{t=\hat{t}_{k-1}+1}^{\hat{t}_k} b_{rt}$. We then use the following resource constraints:

$$\sum_{j \in J} \left(x_{jk} c_{jr} + \sum_{\ell=k+1}^{\kappa(\hat{t}_k + \tau_j) - 1} x_{j\ell} c_{jr} \min\left(\tau_j - (\hat{t}_\ell - \hat{t}_k), \hat{T}_k\right) \right) \leqslant \hat{b}_{rk} \qquad \forall r \in R, k \in \mathcal{K}.$$

These constraints mostly match the format of Carrasco et al. (2022) but differ for jobs completed in interval k. To assure that every feasible solution of the original model (2.1) is feasible to this constraint, it counts resources used in each interval k by assuming the least possible resource usage of any job for that interval. As seen in the version of these constraints used by Carrasco et al. (2022), if a job is completed after interval k, we assume for the constraints for interval k that the job is completed at the end of its scheduled interval ℓ so that the number of periods in which it consumes resources in interval k is the minimum of the remaining duration, $\tau_j - (\hat{t}_\ell - \hat{t}_k)$, and the length, $\hat{T}_k$, of interval k. However, if a job is completed in interval k, we instead assume in the constraints for interval k that it is completed in the first period of the interval and thus takes a single period of resources. Thus, no job accounts for more resources from an interval k using this constraint than it would truly take from those intervals given an actual schedule.

Although similar to the interval resource constraints developed in Carrasco et al. (2022), there is an important difference. In their paper, for the resource constraints, they assume all jobs (including those completed in interval k) are completed at the end of the interval. This works for their case of cumulative constraints since, in this case, any resources not used in one period are available in any future period, but this is not the case for the non-cumulative resources in our model. To understand why this is important, consider the following example with a schedule of jobs j and ℓ with $T = 6$ and a single resource r, where $b_{rt} = 1$ for all $t \in \mathcal{T}$. Job j is scheduled to complete in period four and consumes one unit of resource r each period for $\tau_j = 3$ periods. Job ℓ is scheduled to complete in period 6 and consumes 1 unit of resource r each period for $\tau_\ell = 2$ periods. Consider translating this to an interval model. To do so, split the horizon into two intervals, with $\hat{T}_1 = 2$ and $\hat{T}_2 = 4$. Both jobs complete in interval two. Job j uses two units of resource r in interval two and one unit of resource r in interval one, while job ℓ uses two units of resource r in interval two. Interval two has four units of resource r available, and all are consumed. However, if we were to use the resource constraints given by Carrasco et al. (2022), we would assume both jobs are completed at the end of interval two and thus all five units of resource r for the two jobs would be counted in interval two. This valid schedule would be considered infeasible to the interval model, and hence the interval model would not be a relaxation of the true model. By instead assuming both jobs are completed at the beginning of interval two when counting the resources used for interval two, we get an underestimate of two units of resource r used in interval two, making the solution feasible to the interval model. The version used by Carrasco et al. (2022) is not an issue with cumulative resources, since the one unit of resource r that should have been used in interval one, is still available in interval two where it is instead counted. This does not hold true of non-cumulative resources, so we cannot use this method. Thus, we assume the least resource usage for every interval to avoid overestimation of resource usage that would cause valid schedules to be infeasible to the interval model. This allows this model to be a relaxation of (2.1), in the case we study of non-cumulative resource constraints.

Lastly, we consider the objective. Since Carrasco et al. (2022) develop their interval-based relaxation for an RCPSP, they do not have the notion of coverage in their model so we derive this ourselves. As before, the z variables capture coverage, so we can continue to use the piecewise-linear functions f_n, with z_{nk} as inputs. Since we are then counting

gained coverage only for each interval rather than for each period, to allow this to be a relaxation, we assume for the objective that all jobs are completed at the beginning of the interval they are completed in. Thus, we set $\hat{a}_k = a_{\hat{t}_{k-1}+1}$.

The interval model, $\mathbf{IM}(\mathcal{T})$, is then as follows:

$$\max \sum_{k\in\mathcal{K}} \sum_{n\in\mathcal{N}} (\hat{a}_k - \hat{a}_{k-1}) f_n(z_{nk}) \tag{2.4a}$$

$$\text{s.t.} \sum_{j\in J} \Big(x_{jk} c_{jr} + \sum_{\ell=k+1}^{\kappa(\hat{t}_k+\tau_j)-1} x_{j\ell} c_{jr} \min(\tau_j - (\hat{t}_\ell - \hat{t}_k), \hat{T}_k) \Big) \leqslant \hat{b}_{rk} \qquad \forall r \in R,\ k \in \mathcal{K} \tag{2.4b}$$

$$\sum_{j\in J} \sum_{k\in\mathcal{K}} C_m x_{jk} \leqslant B \tag{2.4c}$$

$$\sum_{\ell=1}^{k} x_{j\ell} \leqslant \sum_{\ell=1}^{\kappa(\hat{t}_k-\rho_{i,j})} x_{i\ell} \qquad \forall\, (i,j) \in \hat{P},\ k \in \mathcal{K} \tag{2.4d}$$

$$\sum_{k\in\mathcal{K}} x_{jk} \leqslant 1 \qquad \forall j \in J \tag{2.4e}$$

$$x_{jk} = 0 \qquad \forall j \in J,\ k < \kappa(\tau_j) \tag{2.4f}$$

$$z_{nk} = \sum_{j\in J} w_{jn} x_{jk} + z_{n,k-1} \qquad \forall n \in \mathcal{N},\ k \in \mathcal{K} \tag{2.4g}$$

$$z_{nk} \geqslant 0 \qquad \forall n \in \mathcal{N},\ k \in \mathcal{K} \tag{2.4h}$$

$$x_{jk} \in \{0,1\} \qquad \forall j \in J,\ k \in \mathcal{K}, \tag{2.4i}$$

where $z_{n0} := 0$ for $n \in \mathcal{N}$.

We next verify that this formulation provides a relaxation of the original model (2.1), noting that this requires an independent proof due to the differences between our model

and that of Carrasco et al. (2022).

Theorem 2.1. *Assume* $a_t \geqslant a_{t+1}$ *for* $t = 1, ..., T$ *and* $f_n \geqslant 0$ *and non-decreasing for all* $n \in \mathcal{N}$. *Then* (2.4) *is a relaxation of* (2.1). *Specifically, for every* $(\bar{x}, \bar{z})$ *feasible to* (2.1), *there exists a feasible solution* $(\hat{x}, \hat{z})$ *to* (2.4) *for which* $(\hat{x}, \hat{z})$ *evaluated in the objective* (2.4a) *is at least as large as the value of* $(\bar{x}, \bar{z})$ *evaluated in* (2.3).

Proof. Let $(\bar{x}, \bar{z})$ be a feasible solution to (2.1) and $\bar{y}$ be its corresponding objective value evaluated in (2.3). Let $\hat{x}$ be defined such that for each $j \in J$, if $\bar{x}_{jt} = 1$ for some t, then $\hat{x}_{j\kappa(t)} = 1$, and $\hat{x}_{jk} = 0$ for all $k \neq \kappa(t) \in \mathcal{K}$. Let $\hat{z}$ be defined such that for each n, $\hat{z}_{n1} = \sum_{j\in J} w_{jn}\hat{x}_{j1}$ and $\hat{z}_{nk} = \sum_{j\in J} w_{jn}\hat{x}_{jk} + \hat{z}_{n,k-1}$ for $k = 2, \ldots, K$. Define $\hat{y}$ according to (2.4a), that is $\hat{y} = \sum_{k\in\mathcal{K}} \sum_{n\in\mathcal{N}} (\hat{a}_k - \hat{a}_{k-1}) f_n(\hat{z}_{nk})$. Note that since $\bar{x}$ is binary, by (2.1e), for any $j \in J$, $k \in \mathcal{K}$ we can write $\hat{x}_{jk} = \sum_{\hat{t}_{k-1}+1}^{\hat{t}_k} \bar{x}_{jt}$. We want to show that $(\hat{x}, \hat{z})$ is feasible for (2.4) and that $\hat{y} \geqslant \bar{y}$.

We begin by showing that $\hat{y} \geqslant \bar{y}$. Recall earlier we showed that (2.3) is equivalent to (2.2), so that we can alternatively write $\bar{y} = \sum_{n\in\mathcal{N}} \sum_{t=1}^{T} a_t (f_n(\bar{z}_{nt}) - f_n(\bar{z}_{n,t-1}))$. Similarly, since $\hat{z}_{n0} = 0$ for all $n \in \mathcal{N}$, we can apply this same equivalence to $\hat{y}$ to get $\hat{y} = \sum_{n\in\mathcal{N}} \sum_{t=1}^{T} \hat{a}_k (f_n(\hat{z}_{nt}) - f_n(\hat{z}_{n,t-1}))$. Next, (2.1g) implies $\bar{z}_{nt} - \bar{z}_{n,t-1} = \sum_{j\in J} w_{jn}\bar{x}_{jt}$ for each $n \in \mathcal{N}$ and $t \in \mathcal{T}$ and summing this over the periods in an interval k yields:

$$\begin{aligned} \sum_{t=\hat{t}_{k-1}+1}^{\hat{t}_k} (\bar{z}_{nt} - \bar{z}_{n,t-1}) &= \sum_{t=\hat{t}_{k-1}+1}^{\hat{t}_k} \sum_{j\in J} w_{jn}\bar{x}_{jt} \\ \implies \bar{z}_{n\hat{t}_k} - \bar{z}_{n\hat{t}_{k-1}} &= \sum_{j\in J} w_{jn}\hat{x}_{jt} = \hat{z}_{nk} - \hat{z}_{n,k-1}. \end{aligned} \tag{2.5}$$

Since $\bar{z}_{n0} = \hat{z}_{n0} = 0$, we have $\bar{z}_{n,\hat{t}_1} = \hat{z}_{n1}$. Inductively then assume $\bar{z}_{n\hat{t}_k} = \hat{z}_{nk}$ for some k, then for $k+1$ we have $\bar{z}_{n\hat{t}_{k+1}} - \bar{z}_{n\hat{t}_k} = \hat{z}_{n,k+1} - \hat{z}_{nk}$, so that $\bar{z}_{n\hat{t}_{k+1}} = \hat{z}_{n,k+1}$. Then, using the

telescoping nature of the objective summation, we have:

$$\begin{aligned}
\hat{y} &= \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{K}} \hat{a}_k\big(f_n(\hat{z}_{nk}) - f_n(\hat{z}_{n,k-1})\big) \\
&= \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{K}} \hat{a}_k\big(f_n(\bar{z}_{n\hat{t}_k}) - f_n(\bar{z}_{n\hat{t}_{k-1}})\big) \\
&= \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{t=\hat{t}_{k-1}+1}^{\hat{t}_k} \hat{a}_k\big(f_n(\bar{z}_{nt}) - f_n(\bar{z}_{n,t-1})\big) \\
&= \sum_{n\in\mathcal{N}}\sum_{t=1}^{T} \hat{a}_{\kappa(t)}\big(f_n(\bar{z}_{nt}) - f_n(\bar{z}_{n,t-1})\big) \\
&\geqslant \sum_{n\in\mathcal{N}}\sum_{t=1}^{T} a_t\big(f_n(\bar{z}_{nt}) - f_n(\bar{z}_{n,t-1})\big) = \bar{y}. \qquad (2.6)
\end{aligned}$$

Inequality (2.6) follows because $\hat{a}_k = a_{\hat{t}_k} \geqslant a_t$ for all t such that $\kappa(t) = k$ and $f_n(\bar{z}_{nt}) \geqslant f_n(\bar{z}_{n,t-1}) \geqslant 0$ as f_n is non-decreasing and $\bar{z}_{nt} \geqslant \bar{z}_{n,t-1}$ for all $n \in \mathcal{N}$, $t \in \mathcal{T}$.

Next, we show that $(\hat{x}, \hat{z})$ is feasible for (2.4). Constraints (2.4g) are satisfied by our definition of $\hat{z}$. Also $w_{jn} \geqslant 0$ and $\hat{x}_{jk} \geqslant 0$ for all $j \in J, k \in \mathcal{K}, n \in \mathcal{N}$ imply that (2.4h) is satisfied. It only remains to show that the constraints involving $\hat{x}$ are satisfied. The relationship $\hat{x}_{jk} = \sum_{t=\hat{t}_{k-1}+1}^{\hat{t}_k} \bar{x}_{jk}$ and $\bar{x}$ being binary and satisfying $\sum_{t\in\mathcal{T}} \bar{x}_{jt} \leqslant 1$ by (2.1e) imply that $\hat{x}_{jk}$ is binary and satisfies (2.4i). It follows from $\sum_{t\in\mathcal{T}} \bar{x}_{jt} = \sum_{k\in\mathcal{K}} \hat{x}_{kt}$ and $\bar{x}$ satisfying (2.1c) that $\hat{x}$ satisfies (2.4c). Similarly, $\hat{x}$ satisfies (2.4e) follows because $\bar{x}$ satisfies (2.1e).

For constraints (2.4d) we need that for each $k \in \mathcal{K}$ and $(i,j) \in \hat{P}$, the inequality $\sum_{\ell=1}^{k} \hat{x}_{j\ell} \leqslant \sum_{\ell=1}^{\kappa(\hat{t}_k - \rho_{ij})} \hat{x}_{i\ell}$ holds. By definition of ρ_{ij}, for any $(i,j) \in \hat{P}$ we have a corresponding path of precedences $i = u_0, u_1, u_2, ..., u_M = j$ with $\sum_{m=1}^{M} \tau_{u_m} = \rho_{ij}$. Then we have by (2.1d) that $\sum_{s=1}^{\hat{t}_k} \bar{x}_{js} \leqslant \sum_{s=1}^{\hat{t}_k - \tau_j} \bar{x}_{u_m s}$ and inductively, $\sum_{s=1}^{\hat{t}_k - \sum_{p=m+1}^{M} \tau_{u_p}} \bar{x}_{u_m s} \leqslant \sum_{s=1}^{\hat{t}_k - \sum_{p=m}^{M} \tau_{u_p}} \bar{x}_{u_{m-1} s}$ for $1 \leqslant m \leqslant M$. Then, $\sum_{\ell=1}^{k} \hat{x}_{j\ell} = \sum_{s=1}^{\hat{t}_k} \bar{x}_{js} \leqslant \sum_{s=1}^{\hat{t}_k - \rho_{ij}} \bar{x}_{is} \leqslant \sum_{\ell=1}^{\kappa(\hat{t}_k - \rho_{ij})} \hat{x}_{i\ell}$ as desired.

Lastly, we consider the resource constraints (2.4b) for a fixed $r \in R$ and $k \in \mathcal{K}$. For $j \in J$,

define

$$\hat{u}_j = \hat{x}_{jk} + \sum_{\ell=k+1}^{\kappa(\hat{t}_k+\tau_j)-1} \min(\tau_j - (\hat{t}_\ell - \hat{t}_k), \hat{T}_k))\hat{x}_{j\ell} \quad \text{and} \quad \bar{u}_j = \sum_{t=\hat{t}_{k-1}+1}^{\hat{t}_k} \sum_{s=t}^{t+\tau_j-1} \bar{x}_{js}.$$

We show that $\hat{u}_j \leqslant \bar{u}_j$ for each $j \in J$, from which the result follows because then

$$\begin{aligned}
\sum_{j\in J} \big(\hat{x}_{jk}c_{jr} + \sum_{\ell=k+1}^{\kappa(\hat{t}_k+\tau_j)-1} \hat{x}_{j\ell}c_{jr} \min(\tau_j - (\hat{t}_\ell - \hat{t}_k), \hat{T}_k)\big) &= \sum_{j\in J} c_{jr}\hat{u}_j \\
&\leqslant \sum_{j\in J} c_{jr}\bar{u}_j \\
&= \sum_{j\in J} c_{jr} \sum_{t=\hat{t}_{k-1}+1}^{\hat{t}_k} \sum_{s=t}^{t+\tau_j-1} \bar{x}_{js} \\
&\leqslant \sum_{t=\hat{t}_{k-1}+1}^{\hat{t}_k} b_{rt} = \hat{b}_{rk},
\end{aligned}$$

where the inequality follows because $\bar{x}$ satisfies (2.1b). We thus consider a fixed job $j \in J$. If $\sum_{k\in\mathcal{K}} \hat{x}_{jk} = 0$, then $\hat{u}_j = 0 \leqslant \bar{u}_j$. We thus assume $\sum_{k\in\mathcal{K}} \hat{x}_{jk} = 1$ and let p be the interval such that $\hat{x}_{jp} = 1$ and let q be the time period with $\bar{x}_{jq} = 1$. (Note that p and q depend on the job j, but we suppress this to simplify the notation since j is fixed in this argument.) Now, if either $p < k$ or $p \geqslant \kappa(\hat{t}_k + \tau_j)$ then $\hat{u}_j = 0 \leqslant \bar{u}_j$. If $p = k$, then $\hat{u}_j = 1 \leqslant \bar{u}_j$ because $\bar{x}_{jq} = 1$ and $\hat{t}_{k-1} + 1 \leqslant q \leqslant \hat{t}_k$. Thus, consider the final case with $k < p < \kappa(\hat{t}_k + \tau_j)$. Note that $\bar{u}_j$ records the number of periods the solution $\bar{x}$ indicates that job j uses resource r during interval k. The job starts at period $q - \tau_j + 1$ and we determine $\bar{u}_j$ based on whether or not $q - \tau_j + 1 \leqslant \hat{t}_{k-1}$. If $q - \tau_j + 1 \leqslant \hat{t}_{k-1}$, then job j starts before interval k and ends after interval k, and hence $\bar{u}_j = \hat{T}_k$, the total number of periods in interval k. If $q - \tau_j + 1 > \hat{t}_{k-1}$, then job j uses resource r for $\hat{t}_k - (q - \tau_j)$ periods during interval k. Since $\hat{T}_k = \hat{t}_k - \hat{t}_{k-1} \leqslant \tau_j - (q - \hat{t}_k)$ if and only if $q - \tau_j \leqslant \hat{t}_{k-1}$, these cases can be combined with the statement that $\bar{u}_j = \min\{\hat{T}_k, \tau_j - (q - \hat{t}_k)\}$. Finally, the result follows because

$$\hat{u}_j = \min\{\hat{T}_k, \tau_j - (\hat{t}_p - \hat{t}_k)\} \leqslant \min\{\hat{T}_k, \tau_j - (q - \hat{t}_k)\} = \bar{u}_j,$$

where the inequality follows because $q \leqslant \hat{t}_p$.

Therefore all constraints of (2.4) are satisfied by $(\hat{x}, \hat{z})$ so that (2.4) is a valid relaxation of (2.1). □

Interval Model Solution Algorithm

Given a solution $(\hat{x}, \hat{z})$ to the interval model, $\mathbf{IM}(\mathcal{T})$, we can create a solution to the full model (2.1), as described in Carrasco et al. (2022). We detail this process in Algorithm 1, which is adapted from Carrasco et al. (2022) to fit our notation. Given a solution $(\hat{x}, \hat{z})$ to (2.4), Algorithm 1 outputs finishing times $\bar{t}_j$ for each job $j \in J$, where $\bar{t}_j = 1$ indicates that job j is not scheduled. Using this we define a solution $(\bar{x}, \bar{z})$ to (2.1) by setting $\bar{x}_{j\bar{t}} = 1$ for each $j \in J$ with $\bar{t}_j \neq -1$, and then setting $\bar{z}$ to satisfy constraints (2.1g). In Algorithm 1, the directed graph induced by the precedence relations is used to create a topological order of the jobs. It then iterates through each interval and schedules jobs assigned to that interval, iterating through jobs according to the topological order. The algorithm schedules each job to complete in the earliest period possible given resource and precedence constraints, no earlier than the first period of the interval it was assigned to. Iterating through all the intervals yields a schedule that satisfies the resource and precedence constraints. One main difference in Algorithm 1 from the one presented in Carrasco et al. (2022) is that it is possible for some jobs to not be completed in the time horizon, and therefore we modify the algorithm to add a flag for incomplete jobs to the output.

Rolling Horizon Heuristic

In order to improve upon solutions found using the interval-based model, we exploit the fast solving nature of Algorithm 1, by integrating this method into a rolling horizon heuristic. Since we do not have cumulative resource constraints as in Carrasco et al. (2022), we cannot obtain an approximation guarantee as they have. In addition, while Algorithm 1 provides a solution quickly, in preliminary computational studies, we found that directly using the solution obtained from Algorithm 1 with the solution from the interval model as input generally yields relatively poor quality solutions. Since this method does provide solutions quickly, we develop a rolling horizon heuristic where we solve the interval model across multiple iterations to find better solutions.

Algorithm 1 Interval Model Solution Algorithm

Input: $\omega(J) \leftarrow$ topological order of J according to P, $\mathcal{T} = \{\hat{t}_k : k \in \mathcal{K}\} \leftarrow$ set of interval end times, $(\hat{x}, \hat{z}) \leftarrow$ solution to **IM**$(\mathcal{T})$
Output: Finishing time $\bar{t}_j$ for each job $j \in J$ ($\bar{t}_j = -1$ if job not completed)

$\bar{t}_j \leftarrow -1 \; \forall j \in J$
for $k = 1, \ldots, K$ **do**
  for $j \in \omega(J)$ such that $\hat{x}_{jk} = 1$, following the topological order **do**
    $t \leftarrow \max\{\hat{t}_{k-1} + 1, \max\{\bar{t}_i + \tau_j : (i,j) \in P\}\}$
    while not enough resources to assign job j to $[t - \tau_j, t]$ and $t \leqslant T$ **do**
      $t \leftarrow t + 1$
    end while
    if $t \leqslant T$ **then**
      $\bar{t}_j \leftarrow t$
    end if
  end for
end for

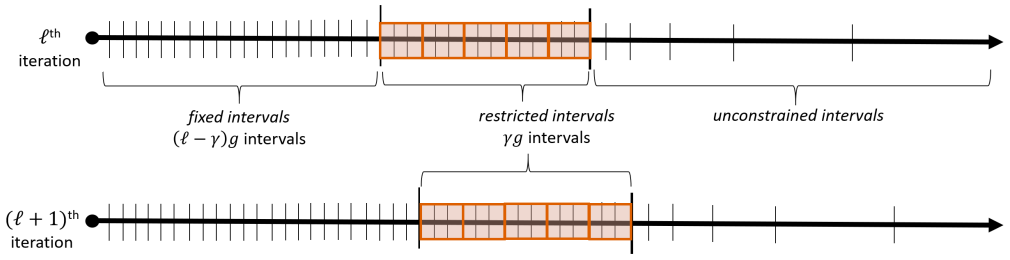


Figure 2.1: Interval set up for ℓth and $(\ell + 1)$th iterations of Rolling Horizon Heuristic.

At each iteration of the rolling horizon heuristic, we solve an interval model with a time horizon closer to the original periods, but use knowledge from past iterations to strategically add constraints to preserve the fast-solving nature of the model while avoiding significant sacrifices in solution quality.

The rolling horizon heuristic is described in Algorithm 2. The heuristic uses two integer parameters to divide the planning horizon into three segments: group size $g \geqslant 1$ and restricted segment size $\gamma \geqslant 1$. For iterations $\ell \geqslant \gamma$, the first segment consists of periods 1 to $(\ell - \gamma)g$, the second segment consists of the next γg periods, and the final segment covers the remaining periods. Figure 2.1 illustrates these segments for iterations ℓ and $\ell + 1$, for the case $\ell \geqslant \gamma$. When $\ell < \gamma$ the first segment is empty, and the second segment consists of

ℓg periods.

The model solved in the initial iteration $\ell = 0$ is the interval model as described in Section 2.3, with base β to define the interval sizes. Algorithm 1 then provides a feasible solution, which we can use as a reference solution for the next iteration.

Then for each subsequent iteration $\ell \geqslant 1$, a reference solution $(\bar{x}, \bar{z})$ determined from the previous iterations is used to define the model solved. The intervals in the iteration ℓ model are defined by sets, $\mathcal{T}_\ell$, of interval end points $\hat{t}_k$ given as:

$$\mathcal{T}_\ell = \left\{ \hat{t}_k = \begin{cases} k & \text{if } k \leqslant g\ell \\ \min\{T, g\ell + \sum_{i=g\ell+1}^{k} \lfloor \beta^{i-1} \rfloor\} & \text{if } k > g\ell \end{cases} : k \in \mathcal{K}_\ell \right\}.$$

These interval end points are set to be equal to each period for the first $g\ell$ intervals, and then exponentially increasing from interval $g\ell+1$ to the end of the horizon. The index set of intervals for iteration ℓ is then $\mathcal{K}_\ell = \left\{1, ..., K_\ell = \min\{T, g\ell + \lceil \log_\beta ((\beta-1)(T-g\ell)+1) \rceil\}\right\}$, where K_ℓ is determined by the number of intervals necessary to cover the remaining horizon after the first $g\ell$ intervals, given the exponentially increasing size of the unconstrained intervals. In the first segment of the planning horizon (non-empty only when $\ell > \gamma$) the model enforces that the scheduling variables are *fixed* to match the reference solution $\bar{x}$ in those time periods. Let $\bar{t}_j$ be the finishing time of job $j \in J$ in the reference solution and define $\bar{J}_{fix} = \{j \in J : 0 \leqslant \bar{t}_j \leqslant g(\ell - \gamma)\}$ as the set of jobs fixed to start in this first segment. Thus, we enforce the constraints:

$$x_{j\bar{t}_j} = 1 \qquad \forall j \in \bar{J}_{fix} \tag{2.7}$$

$$\sum_{k=1}^{g(\ell-\gamma)} x_{jk} = 0 \qquad \forall j \in J \setminus \bar{J}_{fix}. \tag{2.8}$$

Constraints (2.7) enforce jobs completed in the first $g(\ell - \gamma)$ periods of the reference solution are completed in the corresponding single-period interval and constraints (2.8) enforce that no other jobs are completed in the fixed intervals. In the second segment of the planning horizon, the model enforces that some proportion $\eta \in [0, 1]$ of jobs scheduled in that segment in the reference solution should be scheduled at a similar time. Let $\bar{J}_{res} := \{j \in J : g(\ell - \gamma) < \bar{t}_j < g\ell\}$ be the set of jobs scheduled in the second segment according to the reference solution. Let $\nu_j = \lfloor \frac{\bar{t}_j - 1}{g} \rfloor g$ and define the fixing group for a job

$j \in \bar{J}_{res}$ as $[\nu_j + 1, \nu_j + g]$. Then the model also adds the constraints

$$\sum_{j \in \bar{J}_{rel}} \sum_{k=\nu_j+1}^{\nu_j+g} x_{jk} \geqslant \eta |\bar{J}_{res}| \quad . \tag{2.9}$$

which enforces that at least a proportion η of jobs $j \in \bar{J}_{res}$ are completed in the same fixing group as in the reference solution. Nothing is fixed in the final segment, and thus it matches the interval model of Section 2.3.

Define $\mathbf{CIM}(\mathcal{T}_\ell, \bar{J}_{fix}, \bar{J}_{res},)$ to be the model $\mathbf{IM}(\mathcal{T}_\ell)$ with the constraints (2.7)–(2.9) added. Solving $\mathbf{CIM}(\mathcal{T}_\ell, \bar{J}_{fix}, \bar{J}_{res},)$ yields an interval model solution $(\hat{x}^\ell, \hat{z}^\ell)$ with objective $\hat{y}^\ell$, that we then pass to Algorithm 1 to generate a solution (x^ℓ, z^ℓ) to the original model (2.1) and use this to calculate the objective value y^ℓ using (2.1a). If $y^\ell > \bar{y}$, where $\bar{y}$ is the objective of our current reference solution, we update our reference solution $(\bar{x}, \bar{z})$ to be this new solution.

This rolling horizon method can be iterated for up to $\lfloor \frac{T}{g} \rfloor$ iterations. However, as we fix more jobs, the improvement on solutions decreases, so that we can also choose to stop once the current solution has a small optimality gap in the constrained interval model. That is, if we create a solution to the current iteration's model from the current best solution as we do in Theorem 2.1, we can consider how close this is to optimal for the current model. If it is close to optimal, there is little potential improvement that can be made from our current solution. In order to determine the optimality gap of our reference solution in the model $\mathbf{CIM}(\mathcal{T}_\ell, \bar{J}_{fix}, \bar{J}_{res},)$, we must first convert it into a solution $(\hat{x}, \hat{z})$ to this model. We can do so as in Theorem 2.1, which we detail in lines 20 to 22 of Algorithm 2. This gives an objective for the reference solution in the constrained interval model as $\hat{y}$. Since the constrained interval model changes each iteration, this optimality gap is not strictly decreasing. Thus, to avoid stopping too early, we choose to stop iterating once this optimality gap is below a certain threshold ϵ for a given number of iterations SC. Then if the objective $\hat{y}^\ell$ we obtain from solving the model satisfies $\hat{y}^\ell - \hat{y} < \epsilon$, we increment a counter, sc. Once $sc \geqslant SC$, we terminate the algorithm. We note also that the solution $(\hat{x}, \hat{z})$ generated in this process can be used as a warm start for the solver to improve run-time when solving the model $\mathbf{CIM}(\mathcal{T}_\ell, \bar{J}_{fix}, \bar{J}_{res},)$.

By continually constraining the model further as the intervals become closer to the

Algorithm 2 Interval Based Rolling Horizon Heuristic

Input: $\omega(J) \leftarrow$ topological order of J according to P, $\beta \leftarrow$ interval size generation parameter, $g \leftarrow$ group size, $\gamma \leftarrow$ fixed/restricted interval split parameter, $\epsilon \leftarrow$ termination threshold, $\eta \leftarrow$ restricted job proportion parameter, $SC \leftarrow$ termination value of sc

Initialize: $sc \leftarrow 0$, $(\bar{x}, \bar{z}, \bar{y}) \leftarrow (\mathbf{0}, \mathbf{0}, 0)$, $\bar{t}_j \leftarrow -1 \; \forall \, j \in J$.
for $\ell = 0, 1, 2, \ldots$ **do**
  $\bar{J}^{\ell}_{fix} \leftarrow \{j \in J : 0 \leqslant \bar{t}_j \leqslant g(\ell - \gamma)\}$
  $\bar{J}^{\ell}_{res} \leftarrow \{j \in J : g(\ell - \gamma) < \bar{t}_j \leqslant g\ell\}$
  $\tilde{x}^{\ell}_{jk} \leftarrow \sum_{t=\hat{t}_{k-1}+1}^{\hat{t}_k} \bar{x}_{jt}$ for each $j \in J$, $k \in \mathcal{K}_\ell$
  $\tilde{z}^{\ell}_{nk} \leftarrow \sum_{s=1}^{k} \sum_{j \in J} w_{jn} \tilde{x}^{\ell}_{js}$ for each $n \in \mathcal{N}$, $k \in \mathcal{K}_\ell$
  $\tilde{y}^{\ell} \leftarrow$ objective (2.4a) of $IM(\mathcal{T}_\ell)$ evaluated at $(\tilde{x}^{\ell}, \tilde{z}^{\ell})$
  $(\hat{x}^{\ell}, \hat{z}^{\ell}, \hat{y}^{\ell}) \leftarrow$ solution and objective value of **CIM**$(\mathcal{T}_\ell, \bar{J}^{\ell}_{fix}, \bar{J}^{\ell}_{res})$
  $t^{\ell}_j \leftarrow$ completion time for $j \in J$ output from Algorithm 1 with inputs $\mathcal{T}_\ell$ and $(\hat{x}^{\ell}, \hat{z}^{\ell})$
  $x^{\ell}_{jt} \leftarrow \begin{cases} 1 & \text{if } t = t^{\ell}_j \\ 0 & \text{o.w} \end{cases}$ for $t \in \mathcal{T}$
  $z^{\ell}_{nt} \leftarrow \sum_{s=1}^{T} w_{jn} x_{js}$ for each $n \in \mathcal{N}$, $t \in \mathcal{T}$
  $y^{\ell} \leftarrow \sum_{n \in \mathcal{N}} \sum_{t \in \mathcal{T}} \alpha_t f(z_{nt})$
  if $y^{\ell} \geqslant \bar{y}$ **then**
    $(\bar{x}, \bar{z}, \bar{y}) \leftarrow (x^{\ell}, z^{\ell}, y^{\ell})$
    $\bar{t}_j \leftarrow t^{\ell}_j$
  end if
  if $\hat{y}^{\ell} - \tilde{y}^{\ell} < \epsilon$ **then**
    $sc \leftarrow sc + 1$
  end if
  if $\ell > \frac{T}{g}$ or $sc \geqslant SC$ **then**
    return $(\bar{x}, \bar{z}, \bar{y})$ as final solution
  end if
end for

original periods, we generally can maintain the fast-solving nature of the interval model over all iterations. In addition, setting a time limit or using a loose relative optimality tolerance when solving the interval models can further reduce the time for running this heuristic. Another property of this method is that the first iteration yields the solution from running the interval model and finding a corresponding schedule. Thus, this method still provides a reasonable feasible solution very quickly, but also can provide improving solutions as it proceeds through the iterations.

2.4 Computational Results

We perform a series of computational experiments to analyze the benefit of our model and the performance of the proposed heuristic methods. We describe our method for generating synthetic test instances in Section 2.4, and in Section 2.4 we discuss details of the parameters of the rolling horizon heuristic presented in Section 2.3. In Section 2.4 we report results on instances that are small enough that the integrated model can be solved to optimality using a MIP solver. In Section 2.4 we report results on a test set of larger instances in order to test the scalability of our methods.

We compare the performance of the full model (2.1) on these instances to the heuristics presented in Section 2.3 as well as the two heuristic methods derived from existing models detailed in the appendix. The first of these (*RCPSP*) is based on solving a RCPSP model that considers the timing and vulnerability coverage benefit of job completions, but ignores the impact of potential diminishing returns from covering vulnerabilities multiple times. The second modified existing method, select-then-schedule (STS), first uses the model of Zheng et al. (2019) to select a set of jobs taking into consideration the diminishing returns, and then uses an RCPSP model to schedule the selected jobs. Table 2.2 summarizes all the methods we test in our experiments: solving the full model (*OPT*) given by (2.1), the RCPSP model (*RCPSP*), the select-then-schedule method (*STS*), the Interval Model Solution Algorithm (*Int-fast*) of Section 2.3, and the Interval-based Rolling Horizon heuristic (*Int-roll*) of Section 2.3. Method *Opt-TL* refers to solving the full model and reporting the best solution the solver takes within the a given time limit.

All IP models are solved using the commercial MIP solver Gurobi 10.0.0, and our experiments are conducted on a machine with 16GB RAM and an Intel Core i7-7660U

Table 2.2: Methods used in Section 2.4

Method	**Abbreviation**
Full Model (2.1)	*Opt*
RCPSP Model (Appendix A.1)	*RCPSP*
Select-then-Schedule Method (Appendix A.1)	*STS*
Interval Model Fast Algorithm (Section 2.3)	*Int-fast*
Interval Model Based Rolling Horizon Algorithm (Section 2.3)	*Int-roll*
Time-limited Full Model	*Opt-TL*

processor at 2.5GHz.

Data Generation

We summarize here the parameters used to generate the synthetic data. A more detailed description of the data generation is given in the Appendix, and the data for all our test instances is in the Github repository (Peper, 2024). For creating the jobs and resource data, we use two methods. For some of our instances we use the j120 instances from PSPlib to generate the project data. We also supplement the test instances with project data we created to have instances with a wider variety of structures, generally following the format presented in Kolisch and Sprecher (1997) for RCPSP data generation. In both cases, we then generate coverage data to create a full instance of our problem.

For the project data we generate, we create the set of jobs by creating a collection of projects, each of which has some number of jobs, including “start" and “finish" jobs that have no predecessors or successors, respectively. The generation process is parameterized by the number of projects Φ, a range for the number of jobs in each project, $[J_{min}, J_{max}]$, and the complexity, ν, of the precedence network determined by the average number of arcs exiting each node. The per period budget of each resource is determined by a “resource scarcity" parameter, RS, by multiplying RS by the highest amount of the resource used by any one job. Setting $RS \geqslant 1$, ensures that all jobs can be done individually. The PSPlib data does not include an overall budget, so for all instances this is determined using a “budget factor" parameter Bud that gives the expected percentage of jobs that can be completed.

Each project contains a “finish job”, which is the last job in that project, and we enforce that all finish jobs j have $w_{jn} > 0$ for some vulnerability $n \in \mathcal{N}$ (i.e., they provide coverage

of at least one vulnerability). We use a parameter ML to determine what percent of non-finish jobs also provide positive coverage to some vulnerability. For nodes that provide positive coverage to some vulnerability, the average amount of nodes a job covers is given by a "coverage factor" parameter, CF. The amount of coverage each such job provides a node is in part determined by how much work that job requires to be completed, where the amount of work is determined by the total resources required by the job and the jobs that precede it. A "scaling factor" parameter, SF, is used to determine how much impact work required has on a job's coverage. The objective function is created as a piecewise linear function for each node, with a parameter, ES, that is multiplied by the initial slope for each node's function to determine that function's end slope. If ES is close to one, the objective is close to linear, whereas $ES = 0$ enforces that each node has a point at which coverage no longer provides any benefit.

Rolling Horizon Heuristic Parameter Specifications

We choose parameters for *Int-roll* that produce relatively good solutions without significantly sacrificing run-time by performing some preliminary analysis on alternative instances. We begin by choosing values for β (determines interval sizes), g (determines the fixing group size), γ (determines the split between fixed and restricted intervals), and η (determines the percent of jobs fixed in the restricted intervals). We set $\beta = 2$, $g = 5$, $\gamma = 2$, and $\eta = 0.7$, which generally provide a good balance between run-time and solution quality in most instances. We note that small adjustments in β generally have little impact on the final solution quality, and small increases in γ generally result in large increases in run-time. While the first solution to *Int-roll* is the solution to *Int-fast* with $\beta = 2$, we also run *Int-fast* separately, with a smaller $\beta = 1.4$ to promote better solutions, since the slightly larger run-time is not as impactful as when iterated in *Int-roll*.

We use the stopping early mechanism described in Section 2.3, but with slight modifications to improve run-time. We set $SC = 5$ and $\epsilon = 0.005$. However, we make a small addition, if a solution is identical to the warm start solution, we then add 2.5 to sc instead of 1, so that if the interval model finds no way to improve the current solution more than once we stop iterating.

We add a time limit of three minutes for solving the IP for each iteration. In our

preliminary analysis of this method with large instances we generally found that the solver found improved solutions very quickly, but sometimes took much longer to prove optimality. Since this is part of a heuristic, proving optimality is less important, so that stopping short of an optimal solution has little to no effect on the overall solution quality.

Lastly we improve *Int-fast* (Algorithm 1) by considering multiple orderings of the jobs. This algorithm takes as input a topological sort on the set of jobs. Our data generation process naturally gives us one topological sort of the jobs, but alternatives might provide a better solution. We experimented with many options and found no option in particular that consistently performed better than others. Since this algorithm runs so quickly, rather than using just one topological sort, we run the Interval Model Solution Algorithm with all of our options and take the best solution found. We do this at every iteration of *Int-roll* as well as for *Int-fast*. Additional topological sorts used involve ranking jobs by the amount of node coverage they or their preceding jobs gain, or by the amount the objective (2.4a) decreases if we do not complete the job and those it precedes.

Small Instance Analysis

We first compare our methods on a set of test instances that are designed to be small enough that we can obtain the true optimal solution with method *Opt* within a three-hour time limit. This enables assessing how far from optimal the solutions obtained by the heuristic methods are.

To consider how these methods compare across a variety of instances, we choose a base set of parameters for data generation and then individually vary each of these. This data set includes both the instances based on PSPlib data, as well as instances using our own generated project data. For the PSPlib data we include five instances with additional data generated using varied parameters given in Table 2.4, using two different seeds each. We also include these and an extra five PSPlib instances with a shorter timeline, since given the budget constraint, the full time provided by the PSPlib data is generally not needed. This results in 130 instances of this form. For the project data we create, we then use six different seeds to create instances based on each of these sets of parameters. We have 18 different parameter sets, so this yields a total of 108 instances. The parameters we vary and the values used for each parameter are given in Table 2.3 in addition to those in Table

Table 2.3: Project parameters in small test instances.

Parameter	Base value	Vary values
Number of projects, Φ	8	4, 12
Job duration range, $[J_{min}, J_{max}]$	$[2, 6]$	$[1, 7]$, $[3, 5]$
Resource scarcity, RS	1.25	1.0 ,1.5
Network complexity, ν	1.4	1.1, 1.8

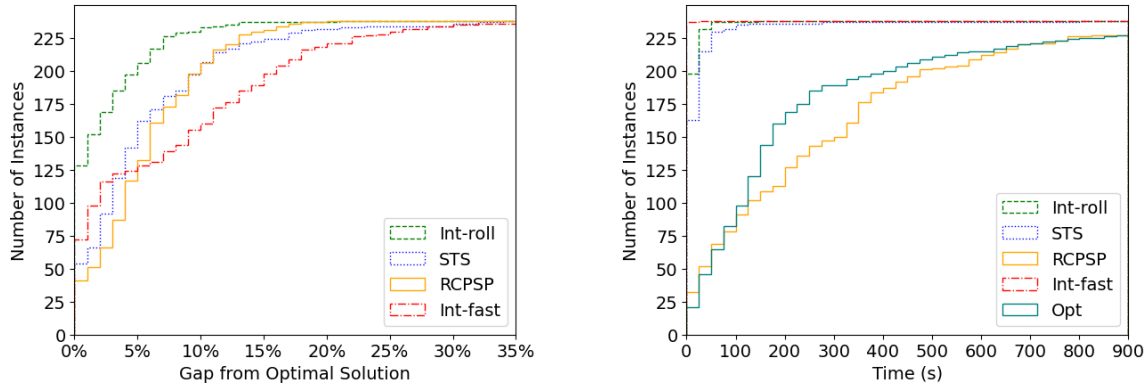
Table 2.4: Additional instance parameters in small test instances.

Parameter	Base value	Vary values
Number of vulnerabilities, $\lvert\mathcal{N}\rvert$	20	10, 30
Exponential time-weighting base, α	0.95	0.9, 0.97, 0.99
Objective linearity, ES	0	0.2, 0.7
Coverage scaling Factor, SF	0.5	0.2, 0.8

2.4. In this table, the column 'Base value' provides the parameter used in the base instance and the column 'Vary values' describe how that parameter is changed to create different instances. For example, the number of projects Φ is 8 in the base set of six instances, and two separate sets of instances are created by setting $\Phi = 4$ and $\Phi = 12$ (but keeping all the rest of the parameters as in the 'Base value' column). When varying the number of projects we also vary project sizes to get an expected similar number of jobs. We vary most of our parameters up and down from the base. One exception to this is our objective linearity parameter. In our problem it makes most sense to have a final slope of zero, as in coverage applications we expect gains of coverage to eventually become negligible once a vulnerability is highly covered.

We run each of our methods with a three-hour time limit for each instance, in order to compare solution quality. The only method to reach this time limit was *RCPSP* on one of the instances with $\alpha = 0.99$.

In Figure 2.2a we display a cumulative distribution plot of the optimality gaps of each method on these instances. Each curve in this graph represents the number of instances for which a method yielded a solution that has optimality gap no more than the amount listed on the x-axis. We first analyze the performance of the methods, *STS* and *RCPSP*, that do not integrate selection and scheduling. Our first observation is that both *STS* and *RCPSP* have significant optimality gaps on many instances – e.g., *STS* and *RCPSP* each



(a) Optimality gaps of heuristic methods on small instances.

(b) Run times of heuristic methods on small instances.

Figure 2.2: Heuristic performance on small instances.

have optimality gaps above 10% on approximately 40 of the instances. On average we find that the *STS* optimality gap is 5.6% and the *RCPSP* optimality gap is 5.9%. These results demonstrate the value of integrating the selection and scheduling of the mitigations in our proposed model. Considering now the other heuristics, we find that *Int-roll* has the best performance in terms of optimality gaps. We additionally see that *Int-fast* performs worse than the other methods, achieving an average of a 7.7% optimality gap. We note that the instances *Int-fast* generally performs well on are those created from PSPlib instances. The main difference between these instances and the instances using project networks we generated, is that the PSPlib instances generally had one large project, rather than multiple smaller projects. This gives insight on when *Int-fast* may be a reasonable solution method for this problem.

To analyze the running time of the heuristics, Figure 2.2b displays the number of instances for each method that complete within the specified time along the x-axis. We limit the range of the x-axis to 10 minutes to better see the differences in the methods with shorter run-times. We can see in this figure that both *RCPSP* and *Opt* take longer to run, while *Int-roll*, *STS*, and *Int-fast* all complete relatively quickly. Based on these two figures, we find that *RCPSP* appears to be ineffective as a heuristic method as it has comparable run times to solving *Opt*.

To better understand when the integrated model is most useful, we studied trends in

solution quality of the *STS* and *RCPSP* methods based on our parameter variation. The only parameter that we observed demonstrated a noticeable trend was objective linearity ES. In Figure 2.3, we display the average optimality gap of these two methods for the three ES values we tested $(0, 0.2, \text{and } 0.7)$. *RCPSP* performs considerably better on instances with higher values of this parameter, often finding the optimal solution when $\text{ES} = 0.7$. We also see that *STS* is not affected as significantly by ES, so that *RCPSP* yields smaller optimality gaps than *STS* for any values when $\text{ES} \geqslant 0.2$. This is not surprising, since with a completely linear objective the RCPSP model is equivalent to our full model.

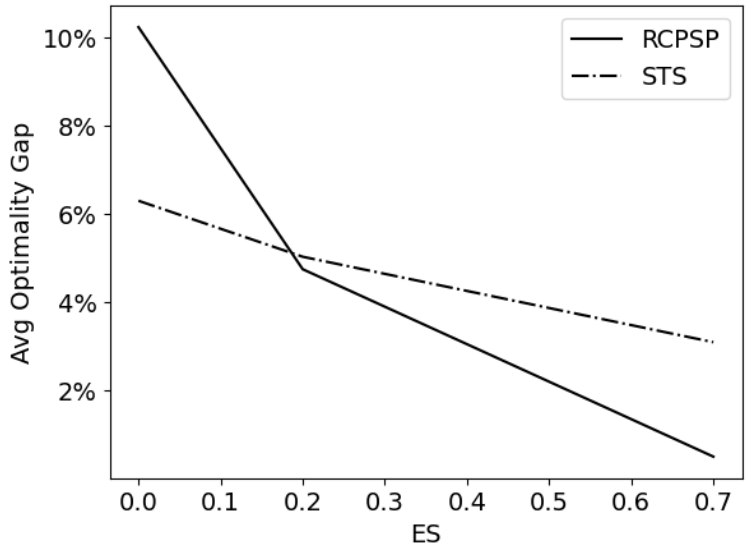


Figure 2.3: Avereage ptimality gaps for RCPSP and Select-then-Schedule Methods varied by linearity parameter (ES)

We additionally consider the effect of the integrated model on what mitigations are chosen and the schedule of their implementation. We show this with two figures. Figure 2.4, shows how much each method covers individual nodes. In this histogram we look at how much coverage has been given to each node in the solutions for the base instances, averaged over all seeds. The x axis provides various amounts of node coverage, and the height of each bar depicts how many nodes are covered that amount. Here we can see how *RCPSP* does not account as well as the other methods for diminishing returns, in that it generally has more nodes with higher coverages than the integrated model. Figure 2.5 shows coverage gained over time by the solutions generated by each method. This figure

plots for each time period the total function-adjusted coverage gained by that time. Here we see that while at the beginning of the horizon *STS* does well at completing jobs that provide good coverage, its lack of consideration of scheduling in the first phase causes the amount of coverage gains it achieves to slow down relative to the integrated model as time goes on. With *RCPSP* we see that it generally gains less coverage than *Opt* by nearly any point in time.

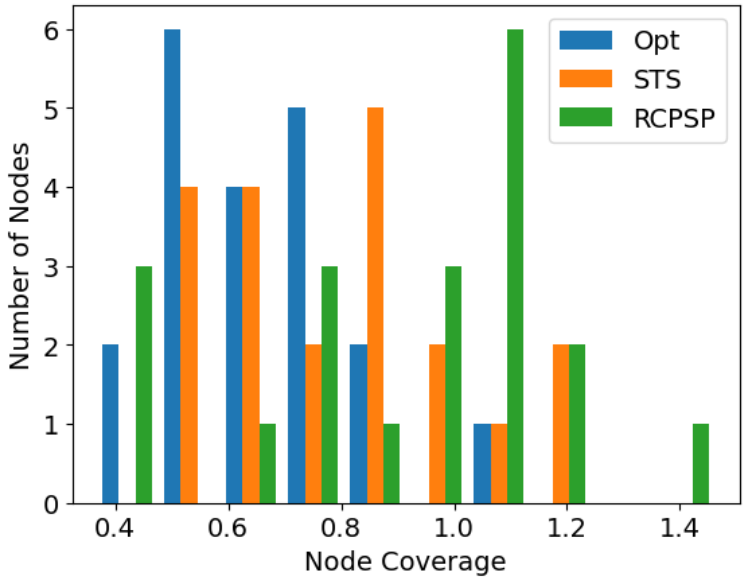


Figure 2.4: Distribution of coverage per node, for base instances averaged over all seeds. Bars give the number of nodes that achieve the amounts of coverage given on the x-axis.

Finally, we investigate different options for the time-weighting coefficients to use in the objective function. Recall that we propose the use of exponential time weighting and use as a base parameter value for our instances an exponential base of 0.95. We test this method against other exponential bases of 0.9, 0.97, and 0.99, as well as the linear implicit time weighting given by $\alpha_t = 1$, or the no time weighting given by $\alpha_t = 1$. Refer back to Section 2.2 for more details on these objective time-weighting options. We display the results of this analysis in Figure 2.6, which displays the cumulative coverage benefit over time when using different time-weighting options. We find that the option of no time weighting gives the best end of the horizon coverage, but not by a significant amount, and at the cost of a significant reduction in coverage earlier in the horizon. The other methods perform

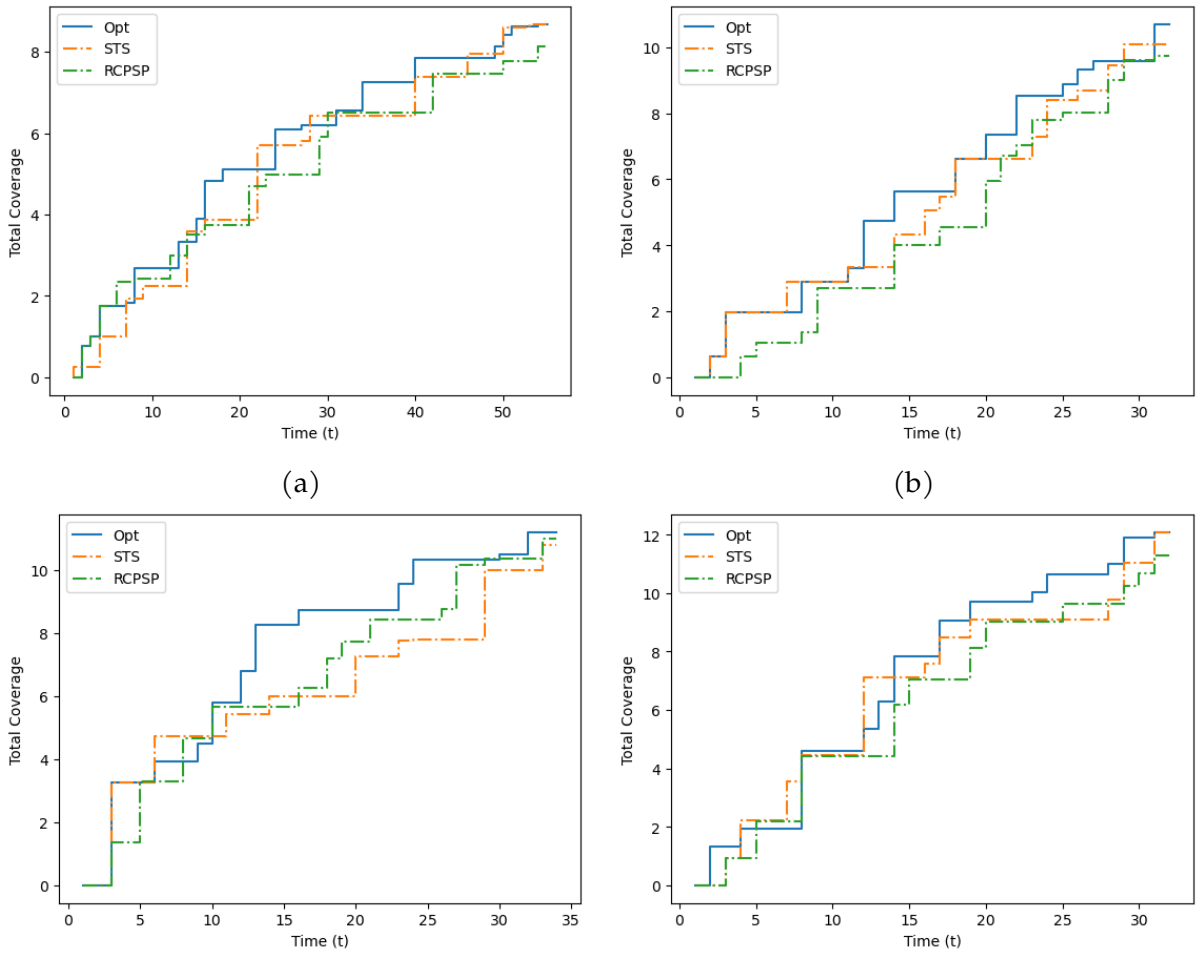


Figure 2.5: Coverage generated over time by each of the preexisting methods in comparison to the integrated model for four representative instances.

similarly to each other, where the linear time weighting and the smaller exponential base value of 0.9 prioritize the beginning of the horizon well, but often at the loss of overall coverage by the end of the horizon.

Large Instance Analysis

We next consider test instances that are so large that solving the full IP directly is time-consuming. In the analysis of the heuristic methods on the smaller instances, we found

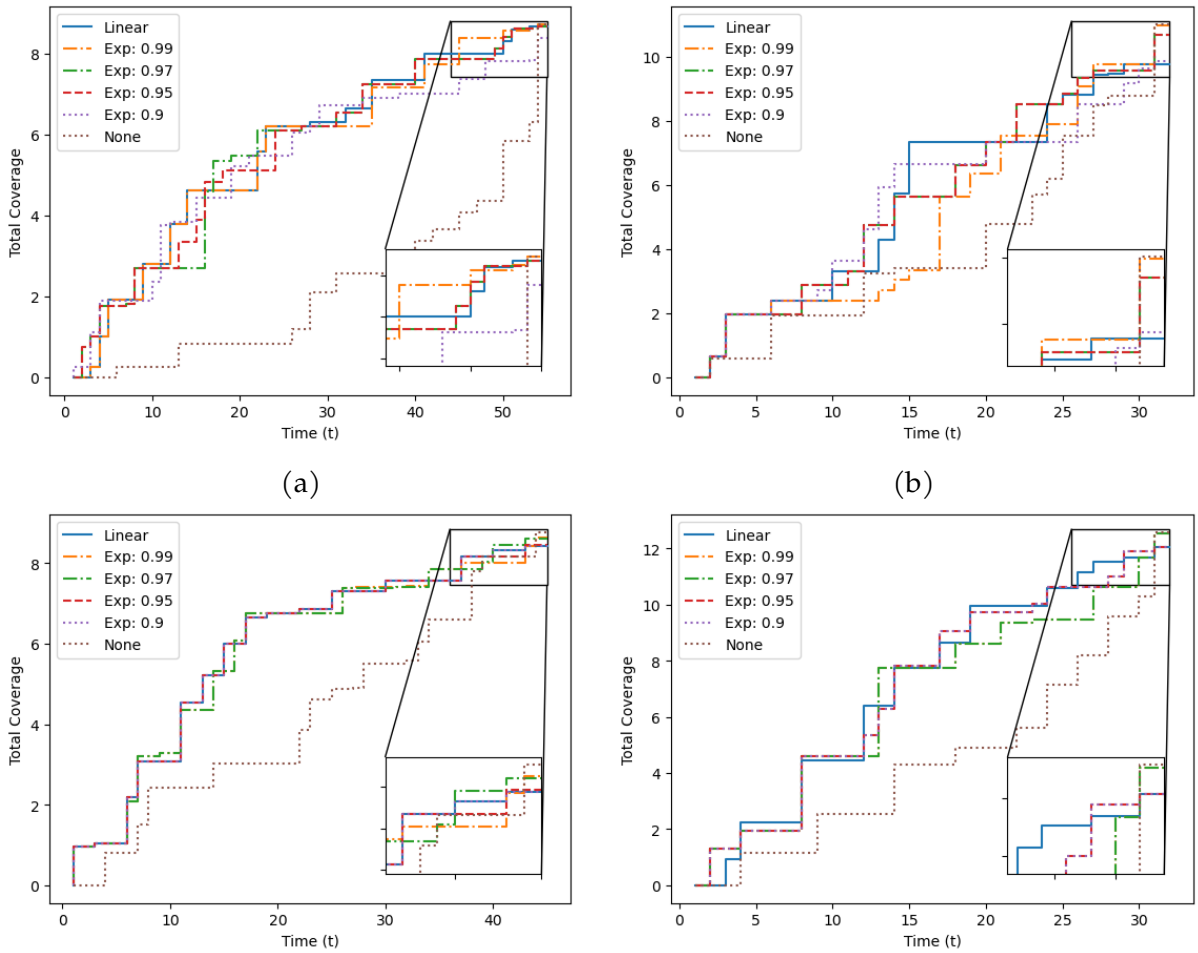


Figure 2.6: Coverage generated over time by solutions from the integrated model with varying time-weights in the objective for four representative instances. 'Linear' refers to $a_t = (T - t + 1)$ for $t \in \mathcal{T}$. Exp' refers to $a_t = \gamma^t$, with γ as specified. 'None' refers to $a_t = 1$ for $t \in \mathcal{T}$.

that *Int-Roll* and *STS* solved the larger of these instances quickly while also finding decent solutions. However, since *RCPSP* generally solved no faster than *Opt*, we do not consider *RCPSP* in this experiment. While other solution methods and heuristics exist for the RCPSP to improve solution times, the results seen in Figure 2.2 show that *RCPSP* does not generally outperform *STS* in terms of solution quality.

To generate a set of large instances, we choose a new set of parameters for our base

Table 2.5: Parameters and variations for large test instances. Uses same base values for parameters not varied as in the small instances, with the exception of including 10 projects with more jobs.

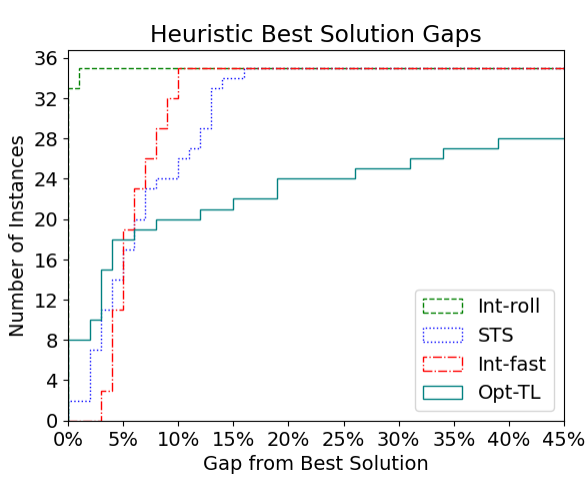
Parameter	**Base values**	**Additional values**
Budget factor, $\mathtt{Bud}$	0.2	0.1, 0.3
Job duration range, $[J_{min}, J_{max}]$	$[1, 10]$	$[4, 6]$
Network complexity, ν	1.4	2.4
Exponential time-weighting base, α	0.97	0.95, 0.99

instances that generate instances not solvable by *Opt* within a 30 minute time limit. We then vary parameters we expect may affect run-times. We already analyzed solution quality when using the solvable instances, so we leave off many parameter variations we used in those instances that appeared unlikely to significantly affect our large instance solutions. In Table 2.5 we display the base parameters used and their variations. We again generate 6 different instances for each parameter variation, leading to a total of 36 instances for these experiments.

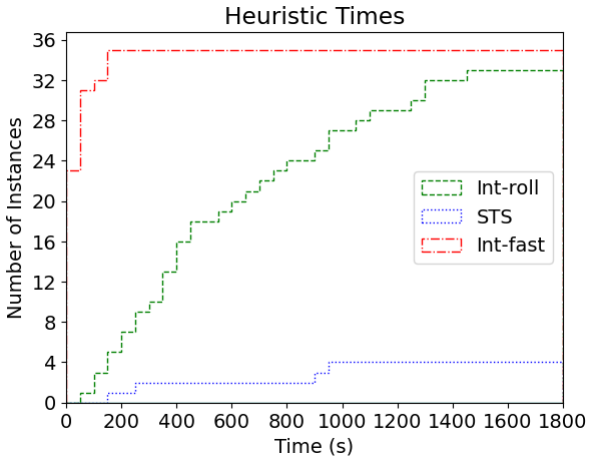
Figure 2.7 displays the performance of the heuristic methods on our large instances. Figure 2.7a displays the gaps from the best solution found by all methods in the 30 minute time limit. These gaps are given along the x-axis, while the number of instances with a gap less than or equal to this value is given along the y-axis. We see that *Int-roll* finds a solution that is within 1% of the best solution found for all but two instances. In fact, the solution found by *Int-roll* is the best solution found in 31 out of the 35 instances. Next, we see that *Int-fast* performs consistently well, with 3% to 10% gaps to the best found solution on all instances. Since we can generally assume our best solution is found by *Int-roll*, of which the *Int-fast* method is essentially the first step of, this also shows we can consistently expect *Int-roll* to improve by approximately 3-10% from its initial to final solution on large problems. Lastly we see that *Opt-TL* generally performs quite poorly. On only eight instances, it was able to find the best found solution in the time limit. On most instances *Opt-TL* provided the worst solution, demonstrating the value of our specialized heuristic *Int-roll* for finding high-quality solutions quickly.

Figure 2.7b displays the cumulative distribution plots of the run-time of each of these methods on the set of large instances. The x-axis displays run-times in seconds, and the y-axis is the number of instances that finished before that much time elapsed. We do not

include *Opt-TL* in this figure, since the 30 minute time limit was reached for all instances. We can see that *Int-fast* finishes very quickly for all instances, taking at most a few minutes in rare cases. On the other hand, *STS* generally did not finish, reaching the time-limit in all but four instances. Lastly we see that *Int-roll* lies in between these two. While slower than *Int-fast*, this method still manages to complete within the time limit on all but three instances. Additionally, in the few instances in which *Int-roll* does hit the time limit, as we will see an example of in Figure 2.8, it still finds better solutions than all other methods.



(a) Heuristic gaps from best solution found



(b) Heuristic run times, given an 30 minute time limit

Figure 2.7: Heuristic performance on large instances.

Another consideration in the performance of these methods is how the solutions develop over time. We display this for a few representative examples in Figure 2.8. In this figure we display the solutions and bounds generated by the IP solver on the full model, a running best solution found by the IP solver in the second phase of *STS*, a single point giving the single solution provided by *Int-fast* and its run-time, and lastly the running best solution found by *Int-roll*. Each line provides the best solution (or bound) from a method at a given point in its runtime, and ends when the model completes so as to provide a comparison of runtimes of each algorithm as well. *Opt-TL* reached the 30-minute time limit on each instance, thus we never see the bounds converge to the solution. The way the solutions improve over time and the relative quality of solutions between methods seen in Figures 2.8a and 2.8b are generally representative of the majority of instances, with *Int-roll* and

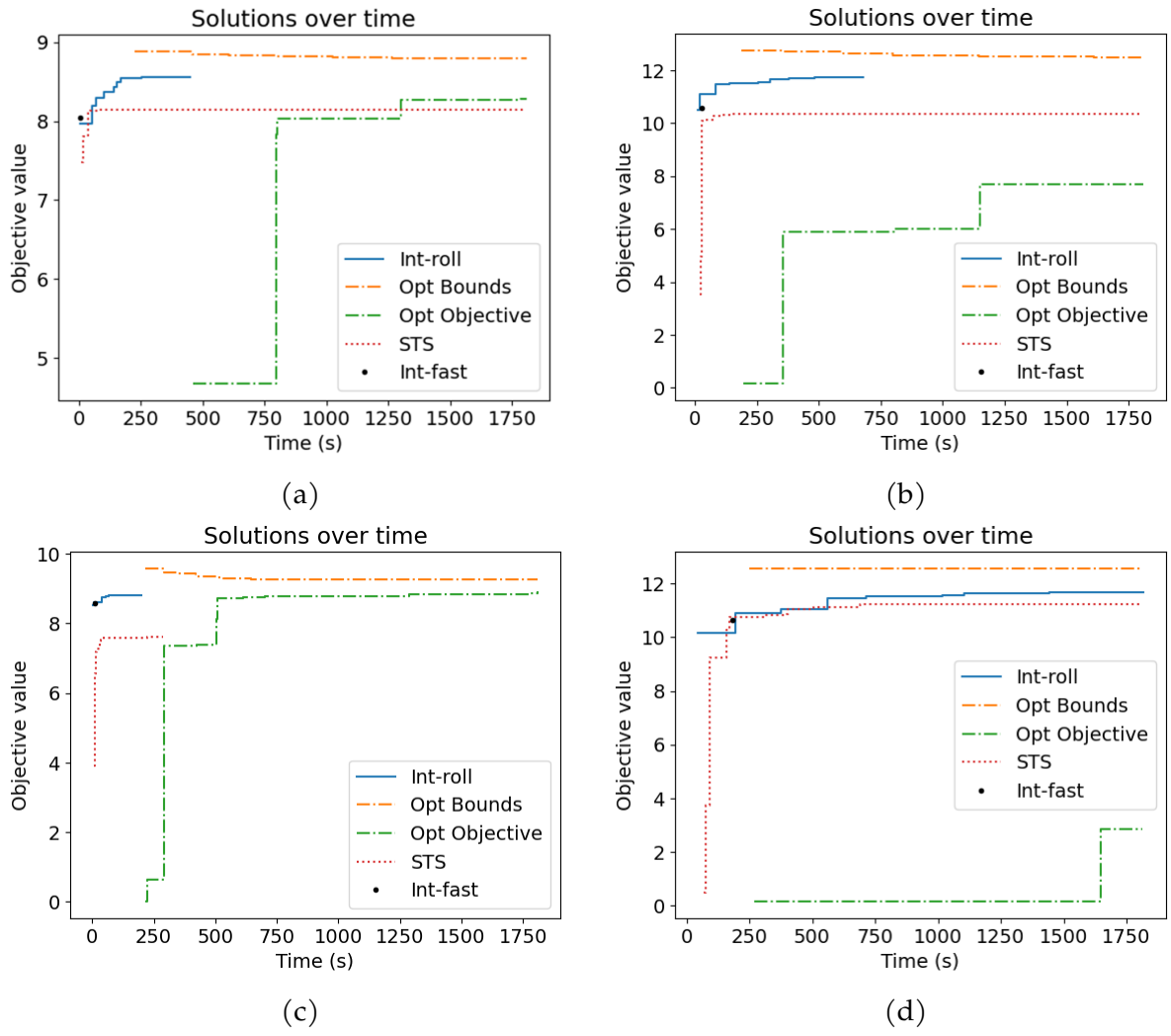


Figure 2.8: Solution generation over time by our heuristic methods, as well as bounds given by the IP solver for the full model, for some representative example instances. All methods were given a 30 minute time limit. *Int-fast* only provides one solution and is represented by a single data point.

STS finding good solutions quickly and *Opt* slowly developing solutions that generally do not surpass the quality of the solutions found by *Int-roll*. We see that at nearly any given point in time *Int-roll* has found the best solution. The only exceptions are short differences early in the time horizon with *Int-fast* or occasionally *STS* (as seen in Figure 2.8a), or in only a few cases, long after *Int-roll* has finished *Opt-TL* finds a slightly improved solution

as in Figure 2.8c. In Figure 2.8d we show an example of one of the two instances where *Int-roll* reaches the time limit. We can see that in this case, the method found much better solutions at any point in time, and reached a point where solutions stopped improving significantly before this time limit. In this way, we see that for instances large enough that *Int-roll* take a long time to solve, given any time limit it will still likely find a better solution than any other method we present.

2.5 Conclusion

In this chapter , we introduce a model that incorporates the scheduling and deployment of mitigations of system vulnerabilities into a model that selects mitigations while considering diminishing returns of multiple coverage. We formulate this as an IP model and demonstrate in a numerical study the benefit of jointly considering selection and scheduling. We also present additional methods for solving this problem for large instances. Through computational experiments, we demonstrate that our proposed interval-based rolling horizon heuristic works well to provide good solutions to difficult problems in a reasonable amount of time.

While we focus specifically on the problem of scheduling mitigations for multiple coverage, the heuristics we develop may also be extended to related problems. In particular, the interval-based model can easily be adapted to build upon the work of Carrasco et al. (2022) by incorporating non-cumulative resource constraints in an interval-based relaxation for an RCPSP. Additionally, because the proposed rolling horizon heuristic does not rely on the multiple coverage structure of our problem, it could be adapted as a heuristic for other project scheduling models.

Following the work of Zheng et al. (2019), further extensions to our model could be considered, such as the possibility that the effectiveness of a mitigation for protecting a vulnerable node is uncertain and hence modeled as a random event. Incorporating the scheduling the deployment of mitigations to the various other mitigation selection problems in the literature provides many other avenues for future work.

3 SCHEDULING MITIGATIONS TO DELAY ATTACKER PROJECTS

3.1 Introduction

Critical infrastructure systems are prevalent throughout the world, and threats to the proper functioning of these systems can significantly impact their users. Such threats may come in the form of sophisticated attackers, carrying out plans to disrupt the system in various potentially costly ways. It takes time for an attacker to complete all the steps necessary to reach their end goal, but it also takes time to implement mitigations to impede the attackers' progress. We consider the problem of implementing mitigations to maximally delay attackers, where we assume that the attackers are already working to breach the system as the mitigations are being implemented. This conservative assumption creates an interplay in timing that makes it important to consider the full deployment of mitigations, rather than just selecting which mitigations to implement.

The problem of delaying adversarial projects is an interdiction problem that has been previously studied. Smith et al. (2013) gives an overview of network interdiction literature. Given these problems can be difficult to solve, various techniques for solving network interdiction problems have been studied (Wei and Walteros, 2022, 2024; Nguyen et al., 2023; Tanınmış and Sinnl, 2022; Contardo and Sefair, 2022; Weninger and Fukasawa, 2025). In the literature, the interdicter is typically modeled with a set of one or more knapsack constraints providing their interdiction budget. The adversarial models, however, vary in complexity. Brown et al. (2005) provide models of interdiction problems for adversarial projects, starting with a basic longest path problem for the adversary, and eventually adding in variations such as task expediting and alternative technologies. They show that this problem can range from polynomially solvable to NP-hard depending on the how the adversarial problem is modeled. Zheng and Albert (2019) maintain more simple attack projects, but consider the problem of interdicting multiple attacker projects at once. In addition, they add an element of stochasticity to the problem in that the duration of attacker actions are uncertain. They develop a Lagrangian heuristic to solve this larger stochastic problem. The variation of considering more than a single attacker is a way this work differs from most other network interdiction literature, and is something we choose to address as well. Dimitrov and Morton (2012) provide a few applications of

network interdiction. One of these applications is also seen in Brown et al. (2009) and Gutin et al. (2015), where they consider the application of interdicting a nuclear weapons project. In Brown et al. (2009) this involves more sophisticated attack project modeling including task expediting and multiple forms of precedences. In their paper, they propose an integer programming formulation and a decomposition algorithm to solve it. Gutin et al. (2015) uses a Markov decision process to model this interdiction problem. Unlike these past works, we use a simple version of an attacker model, one that Brown et al. (2005) demonstrates is polynomially solvable given an interdicter (defender) with a single knapsack constraint. However, the problem we consider is more difficult, since we consider additional complexity to the defender's problem as well as interdicting a set of attackers, rather than a single attacker. There are papers that consider defender actions over time in network interdiction settings by taking a dynamic approach (Zheng and Castañón, 2012; Lunday and Sherali, 2010; Sefair and Smith, 2016; Gutin et al., 2015). These, however, still lack full consideration of resource and precedence constraints over time for the defender. One network interdiction paper that does consider a more complex defender is Malaviya et al. (2012). They consider a multiperiod network interdiction problem to interdict the maximum flow of a network over a time horizon, where the defender is subject to time-based resource constraints. While our work does share similar aspects to this, our model becomes much different in that both the attacker and defender problems are time-based.

In our problem, the defender must spend time and resources in order to implement mitigations to delay attackers. This takes the form of a Resource Constrained Project Scheduling Problem (RCPSP). The RCPSP involves a series of jobs that must be scheduled, where each job uses resources over time and resources are available by time. The jobs must then be scheduled in a way that satisfies these resource constraints as well as precedence relations between the jobs. Each job takes a set amount of time to complete, and the goal is to find a schedule that finishes the last job as early as possible. Originally developed as a zero-one programming model (Pritsker et al., 1969), the RCPSP is commonly studied in the integer programming context. As this is an NP hard problem, many heuristic solutions have been developed for this problem and its variants, e.g. (Carrasco et al., 2022). Hartmann and Briskorn (2022) provide a survey of the current variants and extensions of the RCPSP. Chapter 2 solves a mitigation scheduling problem that uses an RCPSP based model to determine the deployment of mitigations to cover a set of vulnerabilities. Our

model follows a similar approach for modeling the defender actions but uses a significantly different objective. We extend earlier work by considering the timing of both the defender and attackers' decisions. Our model is the first to integrate scheduling of mitigation implementation with a model that considers timing of attacker actions.

In this chapter , we model a defender of system of critical infrastructure who employs mitigations over time in order to maximally delay attackers trying to accomplish some goal to disrupt the system. We formulate this as a bilevel problem, which we then condense into one integer program. The first level is the defender's problem, which we formulate using the model given in Chapter 2 as a baseline. The objective of the defender's problem is a weighted average of the objective values of each attacker's problem. The second level captures the attackers' problems. A single attacker's problem is a critical path problem such as in Zheng and Albert (2019), where the attacker is trying to complete all tasks in their project network as fast as possible. The critical path is the longest sequence of tasks in the network, and thus the determining factor in when the attacker can achieve their goal. However, our problem differs from this paper in that we assume the attacker and defender are making time-based decisions simultaneously. If the defender implements a mitigation that affects an arc before the attacker reaches that arc, the arc is delayed for the attacker, but if the attacker has already started to traverse the arc by the time the defender implements a mitigation, the mitigation has no affect on that traversal. In this way, we have time varying arc lengths in the attackers' project networks, and thus must develop a new way to model the critical path problem for the attackers.

In Section 3.2, we introduce the problem as a network interdiction problem between a defender and set of attackers. We develop models for the defender and each attacker, and combine these into one full IP model that can be solved to find the optimal solution. In Section 3.3, we study a sequential model that heuristically solves the problem by assuming the defender acts first. Because of this assumption the sequential model is comparable to existing methods that ignore scheduling, and it provides a relaxation for our model. In Section 3.4, the sequential model motivates alternative exact formulations for the problem, that simplify the problem and improve LP relaxation bounds. Lastly, in Section 3.5, we perform a series of computational experiments comparing the proposed models against the heuristic of 3.3 and other RCPSP-based heuristics that simplify the problem by disregarding the attacker network, in order to analyze the benefit of an integrated model. We find around

a 10% optimality gap on average for our sequential model and simple RCPSP heuristic, and improve this to about a 5% optimality gap on average with a more sophisticated RCPSP heuristic. We also report the benefit of the proposed reformulations over our original presented model through improved run times, with a median run time of about 430 seconds for our original model and a median of about 70 seconds for our best reformulated model. We present additional results that show there is benefit gained from the integrated model as well as the reformulations we propose.

3.2 Integer Programming Formulation

We consider a problem in which a defender and multiple attacker are working simultaneously to complete projects, where the defender's actions can delay actions in the attacker's projects. This sounds like a multistage model where at each time period, the defender and attacker can react to the actions of the other. However, we note that it is sufficient to model this as a bilevel problem, where the defender chooses a schedule and then attacker completes their project based on the durations of their actions as affected by the defender's choices. This is because the attacker has no true decisionmaking, they only must start each action as soon as it is possible for them to do so. In this way, the attacker is completely predictable and the defender has no need to react to the attacker actions. Thus it is acceptable to say that the defender makes all decisions before the attacker begins their project, and we can model this as a bilivel problem.

Defender's Problem

We begin by modeling the defender's problem. We assume the defender has a set of jobs J they can complete over a time horizon of T time periods (we denote the set of time periods $\{1, ..., T\}$ as $\mathcal{T}$). These jobs are a series of tasks that comprise various mitigations that allow the defender to interdict arcs in attacker project networks, where each job can potentially interdict multiple arcs. Let $\mathcal{N}$ and $\mathcal{A}$ be the set of all attacker nodes and arcs, respectively. When a defender completes job $m \in J$ this increases the time it would take an attacker to move along each arc $(i, j) \in \mathcal{A}$ by δ_{ijm}. We assume the base duration of arc $(i, j) \in \mathcal{A}$ is d_{ij}. Not all jobs are mitigating jobs, so it is possible for some $m \in J$, $\sum_{(i,j)\in\mathcal{A}} \delta_{ijm} = 0$. Such

jobs are simply subtasks that must be completed in order to complete a job that interdicts some arcs. This notion gives rise to precedence constraints. We denote the set of all the defender's precedence constraints as P, where $(m, n) \in P$ if the defender must complete job $m \in J$ before they can start job $n \in J$. The defender can interdict an arc $(i, j) \in \mathcal{A}$ multiple times using multiple mitigations, but only up to some maximum possible delay of $\bar{\delta}_{ij}$. This encourages the defender to use a more diversified interdiction plan by approximating a notion of diminishing returns for interdicting an arc multiple times. We assume completing jobs takes time and resources. A job $m \in J$ must be worked on for τ_m periods, and it requires c_{mr} units of each resource $r \in R$ each period that it is in progress, where R is the set of all resources. In each period t a total of b_{rt} units of resource $r \in R$ are available. Each job $m \in J$ also comes with a fixed cost C_m, for which we have a total budget of B. Given these sets and parameters, we can begin modeling this as an integer program.

We introduce binary decision variables x_{mt} that indicate if job $m \in J$ has been completed at period $t \in \mathcal{T}$. We also define variables z_{ijt} that give the duration of arc (i, j) as of period t, as determined by the mitigations that have been completed by that time. While these two variables capture all the information we need, we cannot simply use z as durations in our attacker problem, since the duration values vary by time. To account for this, we create time-indexed attacker networks consisting of nodes $\mathcal{V}$ and arcs $\mathcal{E}$. We describe further how to develop the time-indexed network in Section 3.2 when we describe the attackers' problem. In this attacker network, nodes are of the form (i, t) for $i \in \mathcal{N}$ and $t \in \mathcal{T}$. Each arc $(i, j) \in \mathcal{A}$ gives rise to several arcs $((i, t), (j, s))$ that comprise the set $\mathcal{E}$, where $s - t$ gives a possible duration of the arc (i, j). With this setup, rather than lengthening arcs according to z, we enforce that an arc $((i, t), (j, s)) \in \mathcal{E}$ is only traversable by the attacker if $s - t \leqslant z_{ijt}$. That is, the defender can only force the attacker to take a longer arc if they have interdicted it to that length by the time the attacker begins traversing the arc. To do so we introduce binary variables ρ_{ijts} that can only equal 1 if arc $((i, t), (j, s))$ is traversable by an attacker. We enforce that, for each arc (i, j) starting at a given period t, only the variable ρ_{ijts} with the largest s such that $s - t \leqslant z_{ijt}$ is equal to 1. This represents the arc with the correct length given the defender's decisions.

Since our goal is to delay attackers as much as possible, we assume the attacker's project may take longer to complete than the time horizon in consideration for the defender. In addition, since the defender will likely revise their plans by re-solving the model in

the future in response to the latest circumstances and so may reasonably limit their time horizon, but would still need to consider the full extent of the attacker plans. Thus for the attacker horizon, we define T_A as the maximum time any attacker can take to complete their project given any set of mitigations. This can be found by solving a longest path problem on each attacker's network using arc lengths defined assuming all mitigations are implemented. We let $\mathcal{T}_A$ be the set $\{1, ..., T_A\}$ of all time periods for attackers. We index z_{ijt} over $t \in \mathcal{T}_A$.

Let A be the set of all attackers and define $Y^a(\rho)$ as the completion time of attacker $a \in A$ given the defender's choice of ρ. We consider how to compute $Y^a(\rho)$ in Section 3.2. Define p_a as the objective weight given to attacker a. This weight can account for things such as severity of the attack or likelihood of the attack. Using these weights for probabilities of a distribution of possible attackers can account for situations with uncertain attacker data.

This leads to the following formulation:

$$\max \sum_{a \in A} p_a Y^a(\rho) \tag{3.1a}$$

$$\text{s.t. } z_{ijt} \leqslant \sum_{m \in J} \delta_{ijm} x_{mt} + z_{i,j,t-1} \qquad \forall t \in \mathcal{T},\ (i,j) \in \mathcal{A} \tag{3.1b}$$

$$z_{ijt} \leqslant d_{ij} + \bar{\delta}_{ij} \qquad \forall t \in \mathcal{T},\ (i,j) \in \mathcal{A} \tag{3.1c}$$

$$z_{ij0} = d_{ij} \qquad \forall (i,j) \in \mathcal{A} \tag{3.1d}$$

$$z_{ijt} = z_{i,j,t-1} \qquad \forall (i,j) \in \mathcal{A}, t = T+1, ..., T_A \tag{3.1e}$$

$$\sum_{s:((i,t),(j,s)) \in \mathcal{E}} (s-t)\rho_{ijts} \leqslant z_{ijt} \qquad \forall (i,j) \in \mathcal{A}, t \in \mathcal{T}_A \tag{3.1f}$$

$$\sum_{s:((i,t),(j,s)) \in \mathcal{E}} \rho_{ijts} \leqslant 1 \qquad \forall (i,j) \in \mathcal{A}, t \in \mathcal{T}_A \tag{3.1g}$$

$$\sum_{t \in \mathcal{T}} x_{mt} \leqslant 1 \qquad \forall m \in J \tag{3.1h}$$

$$x_{mt} = 0 \qquad \forall m \in J, t = 1, .., \tau_m - 1 \tag{3.1i}$$

$$\sum_{m \in J} \sum_{s=t}^{t+\tau_m - 1} c_{mr} x_{ms} \leqslant b_{rt} \qquad \forall t \in \mathcal{T},\ r \in R \tag{3.1j}$$

$$\sum_{t\in\mathcal{T}}\sum_{m\in J} C_m x_{mt} \leqslant B \tag{3.1k}$$

$$\sum_{s=1}^{t} x_{ms} \leqslant \sum_{s=1}^{t-\tau_m} x_{\ell s} \qquad \forall (m,\ell)\in P,\ t\in\mathcal{T} \tag{3.1l}$$

$$x_{mt}\in\{0,1\} \qquad \forall m\in J,\ t\in\mathcal{T} \tag{3.1m}$$

$$z_{ijt}\geqslant 0 \qquad \forall (i,j)\in\mathcal{A},\ t\in\mathcal{T}_A \tag{3.1n}$$

$$\rho_{ijts}\in\{0,1\} \qquad \forall ((i,t),(j,s))\in\mathcal{E}. \tag{3.1o}$$

The objective function (3.1a) maximizes the weighted sum of completion times of each attacker's project. Constraints (3.1b) enforce that z_{ijt} updates each period $t\in\mathcal{T}$ based on mitigations completed that period. Constraints (3.1c) enforce the length of arc (i,j) is no more than the given maximum for that arc. Constraints (3.1d) enforce that all arcs begin at their base lengths at the start of the time horizon. Constraints (3.1e) set the z_{ijt} variables for all periods after the defender's time horizon. The values of these z_{ijt} variables no longer change each period since no more mitigations are being implemented. Constraints (3.1f) and (3.1g) set the ρ variables. With constraints (3.1f), for each arc $(i,j)\in\mathcal{A}$ and period $t\in\mathcal{T}_A$ we are allowed to set $\rho_{ijts}=1$ for any arcs $((i,t),(j,s))\in\mathcal{E}$ with length $s-t$ less than the current arc length z_{ijt}. Then constraints (3.1g) enforce that for each arc (i,j) and period t, at most one arc $((i,t),(j,s))\in\mathcal{E}$ can be chosen with $\rho_{ijts}=1$. Together with the fact that we are maximizing the longest path in the attacker networks, these enforce that the longest arc with length at most z_{ijt} will have $\rho_{ijts}=1$. Since z_{ijt} will be set to be a valid arc length, this implies that $\rho_{ijt,t+z_{ijt}}=1$. Constraints (3.1h) enforce each job can be completed at most once. Constraints (3.1i) enforce each mitigation cannot be completed earlier in the time horizon than its duration. Constraints (3.1j) enforce the resource constraints for each period and (3.1k) enforces the overall budget constraint. Lastly, constraints (3.1l) enforce the precedence constraints between mitigations for the defender.

Attacker's Problem

The objective of the defender is to maximize weighted sum of the times it takes the attackers to complete all the tasks in their project networks. In this section we describe the form of the attacker problem for an arbitrary attacker $a\in A$. The attacker's goal is to complete all tasks

in their project network defined by a set of nodes $\mathcal{N}_a$ and arcs $\mathcal{A}_a$ in the minimum amount of time. We assume the attacker starts at some beginning node $0 \in \mathcal{N}_a$ and reaching an end node $N \in \mathcal{N}_a$ represents the completion of the entire project. This problem is commonly known as the critical path problem. In this problem the attacker can traverse arcs (carry out its tasks) in parallel, so that the longest path in the network, the "critical path", is the one that determines how the overall project completion time. A key challenge in our attacker model that differentiates it from prior work (Brown et al., 2005; Zheng and Albert, 2019) is that in our model the arc durations depend not only on the defender decisions but also on *when* the arc is initiated, due the dependence on the timing of when defender mitigations are completed. Thus we need to introduce a time-indexed network for the attacker so that we can index our constraints by time as well to capture these time varying arc durations.

Let $\mathcal{V}_a$ and $\mathcal{E}_a$ be the nodes and arcs available to attacker a in the time indexed network. We have $\mathcal{V}_a \subseteq \{(i,t) : i \in \mathcal{N}_a, t \in \mathcal{T}_\Lambda\}$, and $\mathcal{E}_a \subseteq \{((i,t),(j,s)) : (i,j) \in \mathcal{A}_a, s,t \in \mathcal{T}_\Lambda, t < s\}$. While we could define these sets at equality, not all arcs in the full set correspond to possible arc durations that may be obtained from a defender action and given that, not all nodes are necessarily reachable. Thus to decrease the number of arcs and nodes in our problem, define $\{\hat{\delta}_{ij}^{\ell}\}_{\ell=1}^{L_{ij}}$ as the set of possible delays for $(i,j) \in \mathcal{A}$ given all mitigation combination options, where $0 = \hat{\delta}_{ij}^{1} < \hat{\delta}_{ij}^{2} < ... < \hat{\delta}_{ij}^{L_{ij}} = \bar{\delta}_{ij}$. Define also $\mathcal{L}_{ij} = \{1, ..., L_{ij}\}$ for each arc $(i,j) \in \mathcal{A}$. Then

$$\mathcal{E}_a \subseteq \left\{((i,t),(j,s)) : (i,j) \in \mathcal{A}_a,\ t,s \in \mathcal{T}_\Lambda,\ t = s + d_{ij} + \hat{\delta}_{ij}^{\ell},\ \ell \in \mathcal{L}_{ij}\right\}.$$

These arcs all correspond to possible task durations and we can define our formulation based on this set. However, since not all nodes are reachable we can further decrease the number of nodes and arcs. For example, suppose i is a node with only node 0 as a predecessor and $d_{0i} = 2$. Then node $(i,1)$ is not reachable. Several more nodes may be similarly unreachable. To create $\mathcal{V}_a$ and $\mathcal{E}_a$ then we can recursively build the network, starting with node $(0,0)$ and adding arcs $((0,0),(i, d_{0i} + \hat{\delta}_{0i}^{\ell}))$ for each $(0,i) \in \mathcal{A}_a$, $\ell \in \mathcal{L}_{0i}$. We can create a topological sort of the nodes in $\mathcal{N}_a$ according to $\mathcal{A}_a$ and consider each node, adding arcs as we did with the arcs from $(0,0)$. Since we go in the order of the topological sort, each time we do this for a node i, we know that this node will have no other ways to reach it, so that any (i,t) not reached by the time of processing node i, will

not be included in $\mathcal{V}$. We define also the end node $(N, T_A + 1)$, with arcs $((N, t), (N, T_A + 1)$ for each $(N, t) \in \mathcal{V}$. This is our end node for all attackers, used as an indicator that node N has been reached. The lengths for the arcs in this time-indexed network are defined as $\tilde{d}_{ijts} = s - t$, for $((i, t), (j, s)) \in \mathcal{E}$ for $s \neq T_A + 1$. For $((N, t), (N, T_A + 1)) \in \mathcal{E}$ we define $\tilde{d}_{N,N,t,T_A+1} = 0$ since these arcs are just recording that the end node has been reached and don't actually take any time. The minimum duration of the attacker project is then determined by the longest path in this time-indexed network, using arcs that are available based on the defender actions. We thus can formulate the attacker problem as a longest path problem in this network.

Define variables y^a_{ijts} as the flow variable on each arc $((i, t), (j, s)) \in \mathcal{E}_a$. Then, the attacker model is as follows:

$$
\begin{aligned}
Y^a(\rho) = \max \quad & \sum_{((i,t),(j,s)) \in \mathcal{E}_a} \tilde{d}_{ijts} y^a_{ijts} && && (3.2a)\\
\text{s.t.} \quad & \sum_{((j,s),(i,t)) \in \mathcal{E}_a} y^a_{jist} - \sum_{((i,t),(j,s)) \in \mathcal{E}_a} y^a_{ijts} = q_{it} && \forall (i,t) \in \mathcal{V}_a && (3.2b)\\
& y^a_{ijts} \leqslant \rho_{ijts} && \forall ((i,t),(j,s)) \in \mathcal{E}_a && (3.2c)\\
& y^a_{ijts} \geqslant 0 && \forall ((i,t),(j,s)) \in \mathcal{E}_a. && (3.2d)
\end{aligned}
$$

where $q_{N,T_A+1} = 1$, $q_{0,0} = -1$, and $q_{i,t} = 0$ otherwise.

The objective (3.2a) maximizes the total length of the attacker's path in the project network, which is equal to the time the attacker reaches node N. Constraints (3.2b) are the flow-balance constraints enforcing each attacker must leave node $(0, 0)$, enter node $(N, T_A + 1)$ and leave any node they enter along the way. Constraints (3.2d) enforce non-negativity of the flow variables. Lastly, where this model differs from a typical longest path problem model, is in constraint set (3.2c). Rather than enforcing $y \leqslant 1$, we upper bound y by the binary ρ. This enforces that an attacker cannot take an arc that is not available based on the defender's decisions.

Integrated Model

We are now ready to state the complete integrated model. Since both the defender and attacker models are maximization problems, we integrate the attacker model for each $a \in A$ directly into the defender model by introducing variables y^a and their associated constraints (3.2b) - (3.2d) for each attacker $a \in A$ into the defender's model, and replacing $Y^a(\rho)$ in the objective of the defender's model with the objective function (3.2a) of $Y^a(\rho)$.

The integrated model is then as follows:

$$\begin{aligned} \max \quad & \sum_{a \in A} p_a \sum_{((i,t),(j,s)) \in \mathcal{E}_a} \tilde{d}_{ijts} y^a_{ijts} && \text{(3.3a)} \\ \text{s.t.} \quad & (3.1b) - (3.1o) \\ & (3.2b) - (3.2d), \quad \forall a \in A. \end{aligned}$$

3.3 Sequential Model

Our computational experience (see Section 3.5) indicates that general purpose MIP solvers are unable to solve the MIP formulation for the integrated model in Section 3.2 for large instances. In particular, with many variables and constraints created by the time-indexed network, the integrated model becomes difficult to solve as the size of the attacker project networks grows. We thus explore an alternative *sequential* model that follows the assumption made in Zheng and Albert (2019) that the defender implements all their mitigations before the attacker starts their project. The motivation for studying this sequential model is threefold. First, as we will see, the optimal value of the sequential model provides an upper bound on the optimal value of our proposed integrated model (3.3). Second, the obtained defender decision can be interpreted as a heuristic solution, which can be evaluated in the true attacker model. Finally, comparing the quality of the obtained defender decision from this model to the optimal solution of the integrated model (3.3) can provide insight into the value of incorporating the timing of attacker actions into the defender model.

In the integrated model (3.3), attackers and defender act simultaneously, so that the length of an attacker's arc is dependent on when the attacker uses it. Here we assume that the defender acts first, so the length of the attacker's arc is chosen once for all time periods. As a result, we can eliminate the time indexed attacker networks we used in the original

model. However, we still must allow for different arc lengths, since the length of each arc is dependent upon what the defender chooses to do. In the problem setting whether the arc is delayed or not, past papers indicate that we can create two flow variables for each arc, one for if it is delayed and one for if it is not (Brown et al., 2005; Zheng and Albert, 2019). Then the flow on the delayed version of the arc is restricted by the defender's choices to mitigate that arc. Since we allow each arc to be delayed multiple times, we cannot directly use this formulation, however we can extend this idea. Instead of two flow variables, one for the original and one for the delayed version, we include multiple arcs, one for each possible arc duration. We defined earlier, $\{\hat{\delta}_{ij}^{\ell}\}_{\ell=1}^{L_{ij}}$ as the sequence of possible delays for arc $(i,j) \in \mathcal{A}$. The attacker flow variables for each attacker $a \in A$ are then $y_{ij\ell}^{a}$ for each arc $(i,j) \in \mathcal{A}_a$ and $\ell = 1, ..., L_{ij}$.

Given these flow variables, we can then include binary variables $\rho_{ij\ell}$ that connect the defender's solution to the attacker's problem by indicating if arc $(i,j) \in \mathcal{A}$ can be taken with delay $\hat{\delta}_{ij}^{\ell}$. The variables z_{ijt} in this formulation then capture the amount of delay on arc (i,j) as of period t. Lastly the binary variables x_{mt} retain the same definition as in the original model, and indicate if mitigation m is completed in period t.

For use in this and future sections, we define the set X as the set of x satisfying the RCPSP portions of the defender's problem. That is,

$$X = \left\{ x \in \{0,1\}^{|J|\times|T|} : (3.1h) - (3.1l) \right\}.$$

The sequential model is then as follows:

$$\max \sum_{a\in A} p_a \sum_{(i,j)\in\mathcal{A}_a} \sum_{\ell=1}^{L_{ij}} (d_{ij} + \hat{\delta}_{ij}^{\ell}) y_{ij\ell}^{a} \tag{3.4a}$$

$$\text{s.t. } x \in X$$

$$z_{ijt} \leqslant \sum_{m\in J} \delta_{ijm} x_{mt} + z_{i,j,t-1} \qquad \forall t \in \mathcal{T}, (i,j) \in \mathcal{A} \tag{3.4b}$$

$$z_{ijt} \leqslant \bar{\delta}_{ij} \qquad \forall t \in \mathcal{T}, (i,j) \in \mathcal{A} \tag{3.4c}$$

$$z_{ij0} = 0 \qquad \forall (i,j) \in \mathcal{A} \tag{3.4d}$$

$$\sum_{\ell\in\mathcal{L}_{ij}} \hat{\delta}_{ij}^{\ell} \rho_{ij\ell} \leqslant z_{ijT} \qquad \forall (i,j) \in \mathcal{A} \tag{3.4e}$$

$$\sum_{\ell \in \mathcal{L}_{ij}} \rho_{ij\ell} = 1 \qquad \forall (i,j) \in \mathcal{A} \tag{3.4f}$$

$$\sum_{j:(j,i)\in\mathcal{A}_a} \sum_{\ell \in \mathcal{L}_{ji}} y^a_{ji\ell} - \sum_{j:(i,j)\in\mathcal{A}_a} \sum_{\ell \in \mathcal{L}_{ij}} y^a_{ij\ell} = 0 \qquad \forall a \in A, i \in \mathcal{N}_a \setminus \{0, N\} \tag{3.4g}$$

$$\sum_{j:(j,N)\in\mathcal{A}_a} \sum_{\ell \in \mathcal{L}_{jN}} y^a_{jN\ell} = 1 \qquad \forall a \in A \tag{3.4h}$$

$$\sum_{j:(0,j)\in\mathcal{A}_a} \sum_{\ell \in \mathcal{L}_{0j}} y^a_{0j\ell} = 1 \qquad \forall a \in A \tag{3.4i}$$

$$y^a_{ij\ell} \leqslant \rho_{ij\ell} \qquad \forall a \in A, (i,j) \in \mathcal{A}_a, \ell \in \mathcal{L}_{ij} \tag{3.4j}$$

$$y^a_{ij\ell} \geqslant 0 \qquad \forall a \in A, \forall (i,j) \in \mathcal{A}_a \tag{3.4k}$$

$$z_{ijt} \geqslant 0 \qquad \forall (i,j) \in \mathcal{A},\ t \in \mathcal{T} \tag{3.4l}$$

$$\rho_{ij\ell} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}, \ell \in \mathcal{L}_{ij}. \tag{3.4m}$$

In this model, the objective (3.4a) again maximizes the weighted sum of longest path lengths of each attacker. We maintain similar z variable constraints, with constraints (3.4e) determining ρ based only on the z variable for period T. Flow balance constraints for the attacker are given by (3.4g) - (3.4i).

A solution to this model can be useful in two ways. First, a solution to the defender's problem only requires $x \in X$. Given this, a valid z and ρ can be calculated for use in the attacker's problem. Thus, since a solution to this model satisfies $x \in X$, it also provides a heuristic solution. We can then use (3.1b) - (3.1e) to calculate z by setting each z_{ijt} as large as possible, and (3.1f) - (3.1g) to determine ρ by setting these constraints to be satisfied at equality. Then given ρ, we can solve the attacker's problem for each $a \in A$ to calculate the true objective (3.1a). Second, since we make the optimistic assumption that the defender completes their scheduled jobs before the attackers begin their projects, this model is a relaxation of our original problem. We thus can use the objective value obtained as an upper bound on the true objective.

Proposition 3.1. *Then sequential model* (3.4) *is a relaxation of the integrated model* (3.3)*. Specifically, for every* $(\bar{x}, \bar{z}, \bar{\rho}, \bar{y})$ *feasible to* (3.3)*, there exists a feasible solution* $(\hat{x}, \hat{z}, \hat{\rho}, \hat{y})$ *to* (3.4) *for*

which $\hat{y}$ *evaluated in the objective* (3.4a) *is at least as large as the value of* $\bar{y}$ *evaluated in* (3.3a).

Proof. Let $(\bar{x}, \bar{z}, \bar{\rho}, \bar{y})$ be a feasible solution to (3.3). Set $\hat{x} = \bar{x} = x$ and $\hat{z}_{ijt} = \bar{z}_{ijt} - d_{ij}$ for $(i,j) \in \mathcal{A}$ and $t \in \mathcal{T}$. It can be easily seen that $\hat{x}$ and $\hat{z}$ satisfy constraints $\hat{x} \in X$ and (3.4b)-(3.4d).

Then set, for each $(i,j) \in \mathcal{A}$, $\hat{\rho}_{ij\hat{\ell}} = 1$ for the largest $\hat{\ell} \in \mathcal{L}_{ij}$ with $\hat{\delta}^{\ell}_{ij} \leqslant \hat{z}_{ijT}$, and $\hat{\rho}_{ij\ell} = 0$ for all other $\ell \in \mathcal{L}_{ij}$. Define $\hat{y}$ such that if given $\bar{y}$ attacker a takes arc (i,j) at any period, $\hat{y}^{a}_{ij\hat{\ell}} = 1$. This is valid since we know these arcs define a valid path, and by our definition of ρ, we have $\hat{y}_{ij\hat{\ell}} \leqslant \hat{\rho}_{ij\hat{\ell}} = 1$. Since $\bar{z}_{ijt}$ is non-decreasing with t, we know that $\hat{z}_{ijT} + d_{ij} = \bar{z}_{ijT} \geqslant \bar{z}_{ijt}$ for all $t \leqslant T$, and by (3.1e) we have $\bar{z}_{ijT} = \bar{z}_{ijt}$ for $t > T$. Thus $s - t \leqslant d_{ij} + \hat{\delta}^{\hat{\ell}}_{ij}$ for $\bar{\rho}_{ijts} = 1$. Thus for any $\bar{y}^{a}_{ijts} = 1$, we have $s - t \leqslant d_{ij} + \hat{\delta}^{\hat{\ell}}_{ij}$. So that (3.4a) evaluated at $\hat{y}$ is at least as large as (3.3a) evaluated at $\bar{y}$. □

3.4 Model Reformulations

We next describe some modifications to the integrated model formulation that are inspired by the sequential model of Section 3.3.

The sequential model (3.4) assumes the defender acts first, and because of this assumption, the time-indexed network is not needed for the attacker models. This reduces the problem size, since instead of having L_{ij} time-indexed arcs for every valid time period for each original arc $(i,j) \in \mathcal{A}$, we just have L_{ij} versions of arc (i,j). Our new observation is that if $T_A > T$, which is likely to occur when the defender is considering potential attacker actions that would take longer to complete than the defender planning horizon, then there are potentially many periods where the assumption that the defender acts first is essentially true. After the defender's time horizon ends, attacker arc lengths remain constant since no more jobs are being completed. We can thus make use of the simplification used in 3.3 for the periods in the range $[T, T_A]$ of the integrated model.

We develop a new reformulated model with a modified time-indexed network. In this modified time-indexed network, we let all of these periods $T, ..., T_A$ be represented by "period" T. We also include a separate ending period $T_A + 1$, so the set of periods for this model is $\mathcal{T} \cup \{T_A + 1\}$.

We then redefine the time-indexed arcs and nodes for the modified time-indexed network. Let $\Theta(t)$ be the set of original periods in the new period $t \in \mathcal{T} \cup \{T_A + 1\}$, i.e., $\Theta(t) = \{t\}$ for $t = 1, \ldots, T-1$ and $\Theta(T) = \{T, ..., T_A\}$. Let $\hat{\mathcal{E}} = \{((i,k),(j,h)) : \exists((i,t),(j,s)) \in \mathcal{E},\ t \in \Theta(k),\ s \in \Theta(h)\}$ be the modified set of time-indexed arcs. Then let $\hat{\mathcal{V}} = \{(i,k) : \exists((i,k),(j,h)) \in \mathcal{E}\} \cup \{(N, T_A + 1)\}$ be the set of nodes for the modified time-indexed network.

Define sets $\mathcal{L}_{ij}^{kh} \subseteq \{1, ..., L_{ij}\}$ as the index sets of valid arc length delays for $(i,j) \in \mathcal{A}$ going from period k to period h. For $k, h \neq T_A + 1$ with $h, k \notin \Theta(T)$, $\mathcal{L}_{ij}^{kh} = \{\ell : d_{ij} + \hat{\delta}_{ij}^{\ell} = h - k\}$ is a single element set. However, if $h = T$ or $k = T$, we have multiple periods represented by T, so that multiple arc durations could be valid. Thus in general, we have $\mathcal{L}_{ij}^{kh} = \{\ell : ((i,t),(j,t + d_{ij} + \hat{\delta}_{ij}^{\ell})) \in \mathcal{E}, t \in \Theta(k), t + d_{ij} + \hat{\delta}_{ij}^{\ell} \in \Theta(h)\}$. Let $\hat{\mathcal{E}}_a$ and $\hat{\mathcal{V}}_a$ be the subsets of these sets for each attacker.

We then index the appropriate variables of the reformulated model by both the time-indexed arcs and by the valid arc length set. Doing so results in more indices than in the original integrated model (3.3), but no more variables since for arcs in the first T periods there is only one valid arc length.

The reformulated model is then as follows:

$$\max \sum_{a \in \mathcal{A}} p_a \sum_{((i,k),(j,h)) \in \hat{\mathcal{E}}_a} \sum_{\ell \in \mathcal{L}_{ij}^{kh}} (d_{ij} + \hat{\delta}_{ij}^{\ell}) y_{ijhk\ell}^{a} \tag{3.5a}$$

$$\text{s.t. } x \in X$$

$$z_{ijt} \leqslant \sum_{m \in J} \delta_{ijm} x_{mt} + z_{i,j,t-1} \qquad \forall t \in \mathcal{T},\ (i,j) \in \mathcal{A} \tag{3.5b}$$

$$z_{ijt} \leqslant \bar{\delta}_{ij} \qquad \forall t \in \mathcal{T},\ (i,j) \in \mathcal{A} \tag{3.5c}$$

$$z_{ij0} = 0 \qquad \forall (i,j) \in \mathcal{A} \tag{3.5d}$$

$$\sum_{\ell \in \mathcal{L}_{ij}^{kh}} \hat{\delta}_{ij}^{\ell} \rho_{ijkh\ell} \leqslant z_{ijk} \qquad \forall ((i,k),(j,h)) \in \hat{\mathcal{E}} \tag{3.5e}$$

$$\sum_{h:((i,k),(j,h)) \in \hat{\mathcal{E}}} \sum_{\ell \in \mathcal{L}_{ij}^{kh}} \rho_{ijkh\ell} \leqslant 1 \qquad \forall (i,j) \in \mathcal{A}, k \in \mathcal{K} \tag{3.5f}$$

$$\sum_{((j,h),(i,k)) \in \hat{\mathcal{E}}_a} \sum_{\ell \in \mathcal{L}_{ji}^{hk}} y_{jihk\ell}^{a} \quad - \quad \sum_{((i,k),(j,h)) \in \hat{\mathcal{E}}_a} \sum_{\ell \in \mathcal{L}_{ij}^{kh}} y_{ijkh\ell}^{a} = 0$$

$$\forall a \in A, (i,k) \in \hat{V} \setminus \{(0,0),(N,K+1)\} \quad (3.5g)$$

$$\sum_{(N,k)\in\hat{V}_a} y^a_{N,N,k,K+1,0} = 1 \qquad \forall a \in A \quad (3.5h)$$

$$\sum_{((0,0),(j,h)\in\hat{\mathcal{E}}_a} \sum_{\ell\in\mathcal{L}^{0h}_{0j}} y^a_{0j0h\ell} = 1 \qquad \forall a \in A \quad (3.5i)$$

$$y^a_{ijkh\ell} \leqslant \rho_{ijkh\ell} \qquad \forall a \in A, ((i,k),(j,h)) \in \hat{\mathcal{E}}_a, \ell \in \mathcal{L}^{kh}_{ij} \quad (3.5j)$$

$$y^a_{ijkh\ell} \geqslant 0 \qquad \forall a \in A, ((i,k),(j,h)) \in \hat{\mathcal{E}}_a, \ell \in \mathcal{L}^{kh}_{ij} \quad (3.5k)$$

$$x_{mt} \in \{0,1\} \qquad \forall j \in J,\ t \in \mathcal{T} \quad (3.5l)$$

$$z_{ijt} \geqslant 0 \qquad \forall (i,j) \in \mathcal{A},\ t \in \mathcal{T} \quad (3.5m)$$

$$\rho_{ijkh\ell} \in \{0,1\} \qquad \forall ((i,k),(j,h)) \in \hat{\mathcal{E}}\ell \in \mathcal{L}^{kh}_{ij}. \quad (3.5n)$$

Relaxation Incorporated Reformulation

As we will see in Section 3.5, the sequential model (3.4) provided in Section 3.3 provides excellent upper bounds as a relaxation for the integrated model (3.3). The bounds given by even the LP relaxation of (3.4) are tighter than those of the LP relaxations of both the original model (3.3) and the reformulated model (3.5). Decreasing the upper bound is often a limiting factor on the solution times, of the original integrated model (3.3) in particular. This motivates the incorporation of the sequential model into the integrated model to improve the upper bounds found by the solver.

We develop a new model with the reformulated model (3.5) as a baseline. We add to this model variables and constraints from (3.4), as well as constraints to connect the variables from these two models. First we include the y^a variables of (3.4), which we will denote as $\tilde{y}^a$ in this model for each attacker $a \in A$. We include the flow balance constraints, (3.4g) - (3.4i), as well as nonegativity constraints. We also add in ρ variables from (3.4), denoted here as $\tilde{\rho}$. In the sequential model, these variables enforce that only one arc length is chosen for each arc $(i,j) \in \mathcal{A}$. This is valid for the sequential model, as arc lengths are determined after the defender completes their chosen jobs. However, in the reformulated model (3.5), each attacker may take arc (i,j) at a different time, and thus may take them with different lengths. We thus cannot enforce that each arc only has one valid length. However, we can enforce this per attacker, that is, each attacker will only take an arc (i,j)

at most once, and the delay on that arc must be at most the delay at the end of the horizon. We thus add an attacker index to each of the $\tilde{\rho}$ variables. We note that choosing only a single length for each arc is something the original model does not fully capture, and is likely a source of LP bound improvement from the sequential model.

Given the new variables $\tilde{y}^a$ and $\tilde{\rho}^a$, we need constraints to logically connect them to the current variables of the model y^a and ρ. We first include the constraint:

$$\tilde{\rho}^a_{ij\ell} \leqslant \sum_{((i,k),(j,h))\in\hat{\mathcal{E}}_a : \ell\in\mathcal{L}^{kh}_{ij}} \rho_{ijkh\ell} \qquad \forall a \in A, (i,j) \in \mathcal{A}_a, \ell = 1, ..., L_{ij} \tag{3.6}$$

$$\tilde{y}^a_{ij\ell} = \sum_{((i,k),(j,h))\in\hat{\mathcal{E}}_a : \ell\in\mathcal{L}^{kh}_{ij}} y^a_{ijkh\ell} \qquad \forall a \in A, (i,j) \in \mathcal{A}_a, \ell = 1, ..., L_{ij} \tag{3.7}$$

Constraints (3.6) enforce that an attacker a can only use arc (i,j) with delay $\hat{\delta}^\ell_{ij}$ if this is valid at some time index. The attacker path is determined by $\tilde{y}^a$, which in constraints (3.7) is restricted so that attacker a only uses arc (i,j) with delay $\hat{\delta}^\ell_{ij}$ if they do so at some time index.

We also must adjust constraints (3.4e) - (3.4f) from (3.4) to include the attacker index on $\tilde{\rho}$.

$$\sum_{\ell=1}^{L_{ij}} \hat{\delta}^\ell_{ij}\tilde{\rho}^a_{ij\ell} \leqslant z_{ijT} \qquad \forall a \in A, (i,j) \in \mathcal{A}_a \tag{3.8}$$

$$\sum_{\ell=1}^{L_{ij}} \tilde{\rho}^a_{ij\ell} = 1 \qquad \forall a \in A, (i,j) \in \mathcal{A}_a \tag{3.9}$$

Then the model is (3.5) subject to additional constraints: (3.4g), (3.6)-(3.9), in addition to nonnegativity constraints on $\tilde{y}^a$ and constraints to set $\tilde{\rho}^a$ between 0 and 1.

An additional small reformulation we make is to constraints (3.8). Let $tmax^a_i$ be the latest time attacker a can reach node i, which can be computed by solving a longest path problem on the attacker's graph under the assumption all mitigations are completed. Since we know that any delay added in periods after $tmax^a_i$ will not affect attacker a, in (3.8) we can determine the final coverage of (i,j) for attacker a to be $z_{ij\,\min\{T,tmax^a_i\}}$. Thus, we

use the modified constraint:

$$\sum_{\ell=1}^{L_{ij}} \hat{\delta}_{ij}^{\ell} \tilde{\rho}_{ij\ell}^{a} \leqslant z_{ij\,\min\{T, t\max x_i^a\}} \qquad \forall a \in A, (i,j) \in \mathcal{A}_a \tag{3.10}$$

3.5 Computational Results

In order to analyze the benefit of the integrated model, the quality of the heuristics, and the relative performance of the model reformulations, we perform and analyze computational experiments. Each model was solved using Gurobi 10.0.1 and a machine with Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz and was given a 30 minute time limit.

Test instances

We generated 60 instances to solve using each proposed method. This data was generated using similar methods as in Chapter 2. We created the attacker data by using an underlying project network generated in the same way as the defender's project network of Chapter 2. Each attacker then uses a subset of arcs from this underlying network, where every arc in the network is included in at least one attacker's arc set. Different attackers may have different resources and abilities so we assume they may take differing amounts of time to traverse each arc. We assign these times for each arc randomly for each attacker in a range $[dmin_a, dmax_a]$. We generate the project network for the defender in the same way as in Chapter 2. Each mitigation in the defender's project is assigned to delay a set of arcs of the attacker's underlying project. The amount of delay each arc gets from that mitigation is a random integer between 1 and max_delay, where max_delay is a parameter given as a proportion of the maximum arc duration parameter, $dmax_a$. We use two metrics: *attacker weight similarity,* AWS, and *attacker weight scaling factor*, ASF, to determine the relative importance of each attacker. We set $AWS \in [0,1]$ to give a value of how similar attacker weights are, with all identical weights having AWS closer to 1. Additionally, ASF gives extra priority to attackers with shorter projects to represent the urgency of defending against them. If p_a be the unmitigated project length of attacker $a \in A$ and $\bar{p}_a$ is the longest unmitigated project length of all attackers, then the attacker weight is generated as $(1 - AWS)((1 - ASF)\omega + ASF\frac{p_a}{\bar{p}_a}) + AWS$, where ω is a $U[0,1]$ random value.

Table 3.1: Instance characteristics for the 60 instances used.

Parameter	Min	Max	Mean	Median
$\|J\|$	103.0	263.0	142.7	137.5
B	20.0	52.0	28.2	27.0
$\|R\|$	1.0	3.0	2.0	2.0
$\|\mathcal{N}\|$	48.0	1107.0	152.7	85.0
T	26.0	52.0	37.4	35.0
T_a	42.0	217.0	91.3	76.0

Table 3.2: Parameters and variations for test instances.

Parameter	Base values	Additional values
# of attacker projects	2	1, 4
# of attackers, $\|A\|$	8	[4,100]
Attacker arc durations, $[dmin_a, dmax_a]$	[1, 10]	[1, 18]
Attacker network complexity, $comp_a$	1.5	2
Attacker weight similarity, AWS	0.7	[1]
Attacker weight scaling factor, ASF	0.2	[0]
Max delay per job	$\frac{1}{3}dmax_a$	$\frac{1}{6}dmax_a$, $\frac{3}{4}dmax_a$

In an attempt to choose instances that are complex enough but generally solvable in a 30 minute time limit, we use a defender network comparable to the smaller instances used in Chapter 2. Further instance characteristics are provided in Table 3.1. We vary parameters affecting the amount of delay given by mitigations, the range of arc durations, the number of attackers, and the structure of the attacker project network. We additionally consider a set of instances with a larger defender network. Table 3.2 details the parameters varied for these test instances. This results in 12 different instance types with 5 instances created for each by using difference seeds.

Methods

In our computational studies we analyze a few different categories of methods. First we have exact solution methods which include the following:

- *Opt-Orig*: The original integrated model (3.3) proposed in Section 3.2.
- *Opt-Reform*: The reformulated model (3.5) proposed in Section 3.4.

- *Opt+Seq*: The reformulated model (3.5) with the integration of parts of the sequential model (3.4), presented in Section 3.4.

In addition, we test heuristic methods derived from pre-existing methods. The first of these is:

- *Seq*: The sequential model (3.4) of Section 3.3. We use *Seq-Rel* to refer to its use as a relaxation, giving an upper bound on the true objective using the objective value given from the sequential model. We use *Seq-Heur* to refer to its use as a heuristic method, using the objective as computed given the defender solution to the sequential model.

Since the sequential model makes the same implicit assumption that existing mitigation scheduling models make, it can be seen as a benchmark of how well those models would perform for this problem.

We additionally consider the option of heuristically scheduling the jobs with an RCPSP without fully addressing the attackers' projects. We use two versions for this heuristic. In both versions of the RCPSP heuristic we use variables $x \in X$, and maximize an objective

$$\sum_{a \in A} p_a \sum_{m \in J} \sum_{t \in \mathcal{T}} \eta^a_{mt} x_{mt},$$

where η^a_{mt} gives some approximation of how much effect completing job $m \in J$ at period $t \in \mathcal{T}$ has on attacker $a \in A$.

- *RCPSP-1*: Here we disregard the attackers' networks and assume each job completed gives a reward of the total delay it adds to all arcs it affects. Note that the maximum amount of delay per arc is ignored, since we simply accrue a reward when a job is completed, and not considering any effect on arcs. We thus use the value $\sum_{(i,j) \in \mathcal{A}_a} \delta_{ijm}$ for each job $m \in J$ and attacker $a \in A$. Note that this value does not vary by time. We thus include a time weighting parameter α_t to prioritize completing jobs earlier. This yields $\eta^a_{mt} = \alpha_t \sum_{(i,j) \in \mathcal{A}_a} \delta_{ijm}$ for each $t \in \mathcal{T}$, $j \in J$, $a \in A$.

- *RCPSP-2*: In this version we incorporate information from the attackers' networks into the objective weights for each job. For an attacker $a \in A$, consider an arc $(i, j) \in \mathcal{A}_a$.

We know that this arc cannot be taken until node i has been reached. Let $tmin_i^a$ be the longest path length to node i, given no jobs are completed. This is the earliest attacker a can possibly reach node i. This means that a job that affects (i, j) before $tmin_i^a$ is guaranteed to impact the attacker. Similarly, $tmax_i^a$ is the longest path length to node i given all jobs are completed. This is the latest possible time node i can be reached by attacker a. Thus any jobs completed after $tmax_i^a$ have no effect on the arc for attacker a. We can then use this information to be more precise with the time weighting.

Let α_{it}^a be the time-weight for an arc originating from $i \in \mathcal{N}_a$. Then, for some $\beta \in [0, 1]$, $\alpha_{it}^a = 1$ if $t \leqslant tmin_i^a$, $\alpha_{it}^a = \beta^{t - tmin_i^a}$ if $tmin_i^a < t \leqslant tmax_i^a$, and $\alpha_{it} = 0$ if $t > tmax_i^a$. Then, $\eta_{mt}^a = \sum_{(i,j)\in\mathcal{A}_a} \alpha_{it}^a \delta_{ijm}$ for each $m \in J$, $a \in A$, $t \in \mathcal{T}$.

Results

Table 3.3: Average run time in seconds of each exact method over five instances for each specified variation. The number of instances that reached the 30 minute time limit is provided in parentheses when applicable.

Instances	***Opt-Orig***	***Opt-Reform***	***Opt+Seq***
Base	435.0 (1)	26.7	18.6
$\lvert P_a\rvert = 1$	46.1	30.3	26.0
$\lvert P_a\rvert = 4$	296.2	307.9	47.5
$comp_a = 2.0$	423.8 (1)	373.6 (1)	372.1 (1)
$\lvert A\rvert = 4$	437.8 (1)	116.7	160.5
$\lvert A\rvert = 100$	1800.0 (5)	1800.0 (5)	1637.8 (3)
$dmax_a = 18$	1097.2 (3)	209.4	213.2
max_delay $= 2$	79.6	24.9	10.1
max_delay $= 9$	738.5 (2)	123.9	91.7
ASF $= 0$	409.5 (1)	24.3	21.6
AWS $= 1$	406.2 (1)	23.3	19.0
$\lvert P_J\rvert = 10$	993.2 (2)	578.7	909.5
Average	**596.9 (17)**	**303.3 (6)**	**294.0 (4)**

Tables 3.3 – 3.6 report the performance of each of the methods of this chapter averaged over the 5 instances for each of the 12 different instance types. Table 3.3 provides the average run time for each exact method on each instance type. Table 3.3 provides the same

Table 3.4: Average run time in seconds of each heuristic method over five instances for each specified variation. Number of instances that reached the 30 minute time limit is provided in parentheses when applicable.

Instances	***Seq***	***RCPSP-1***	***RCPSP-2***
Base	43.9	5.2	3.6
$\lvert P_a \rvert = 1$	38.0	17.3	9.5
$\lvert P_a \rvert = 4$	23.5	17.3	12.2
$comp_a = 2.0$	38.7	13.3	7.5
$\lvert A \rvert = 4$	168.7	30.4	15.3
$\lvert A \rvert = 100$	1119.2 (1)	20.6	30.4
$dmax_a = 18$	529.1 (1)	17.8	69.5
max_delay $= 2$	9.4	10.5	2.3
max_delay $= 9$	381.8 (1)	5.6	4.6
ASF $= 0$	38.4	4.8	3.9
AWS $= 1$	36.2	5.4	3.6
$\lvert P_J \rvert = 10$	928.0 (2)	794.2 (2)	487.9 (1)
Average	**279.6 (5)**	**78.5 (2)**	**54.2 (1)**

data for each heuristic method. If any instance was not solvable within the time limit for an instance type, a count of how many (out of 5) instances the method reached the time limit on for that instance type is recorded in parentheses next to the average run time. Such instances contribute the time limit of 1800 seconds to the average, so note that these values may be biased lower than they would be with no time limit. Table 3.5 provides the optimality gaps of the solutions provided by the heuristic methods. In the few cases where we did not find an optimal solution within the time limit, we compute the gap to the best found solution. As can be seen in Table 3.3, this only happens with all instances with 100 attackers and a single instance with attacker network complexity of 2. An idea of how far off these values are can be given by the bounds given in Table 3.6. Table 3.6 provides the gaps for two upper bounds for each method. Similarly to Table 3.5, these gaps are computed relative to the best solution found. The first value reported is from the bound given by the LP relaxation of each model. The second value is the best bound found by solving the model. For the exact solution methods, this is 0% whenever the instance was solved to optimality. In cases where the exact solution methods reached the time limit, this is the best bound reported by Gurobi at the 30 minute time limit. For *Seq-Rel*, as a relaxation, the best bound it provides is simply the objective value for this model.

Table 3.5: Average optimality gap for each heuristic method over five instances for each specified variation. Gap to best solution found computed instead when all exact methods reached time limit.

Instances	***Seq-Heur***	***RCPSP-1***	***RCPSP-2***
Base	9.3%	9.9%	5.3%
$\|P_a\| = 1$	12.0%	7.7%	3.3%
$\|P_a\| = 4$	7.8%	9.0%	5.8%
$comp_a = 2.0$	8.6%	9.7%	4.8%
$\|A\| = 4$	14.9%	13.2%	8.2%
$\|A\| = 100$	0.9%	4.1%	1.9%
$dmax_a = 18$	11.2%	17.2%	9.6%
max_delay $= 2$	2.1%	3.6%	1.8%
max_delay $= 9$	20.1%	18.4%	11.5%
ASF $= 0$	8.9%	9.8%	5.2%
AWS $= 1$	9.1%	9.9%	5.3%
$\|P_J\| = 10$	9.9%	12.6%	6.9%
Average	**9.6%**	**10.4%**	**5.8%**

We first compare the exact solution methods. We see that in most cases *Opt-Seq* performs the best out of the three methods. In Table 3.3, we see that *Opt-Orig* generally has significantly longer run times than the other methods, whereas *Opt-Reform* and *Opt-Seq* perform similarly. Some notable differences are *Opt-Reform* performing better with fewer attackers and with a larger defender project. Additionally *Opt-Seq* performs significantly better on the instances with an attacker network with more smaller projects. Looking at Table 3.6, we see that each of the reformulations improve the initial bounds. In particular, *Seq-Rel* has initial LP relaxation bounds better than those of *Opt-Reform*, and the combination of these models, *Opt+Seq*, has the best initial bounds. Additionally, we can see that *Opt+Seq* outperforms the other methods in terms of efficiently reducing the upper bound in that this method has the lowest final bound gap for all instance types.

When examining the heuristic methods, we see there is a solution gap that is addressed by an integrated model. Table 3.5 reports that both *Seq-Heur* and *RCPSP1* average around a 10% optimality gap, while *RCPSP-2* improves upon this with an average optimality gap of about 6%. Furthermore, Table 3.4 suggests that these heuristics may be most useful in cases with several attackers. This additional model complexity does not affect the complexity of the RCPSP models, allowing them to still be solved quickly. In addition, with more

Table 3.6: Average gap between best solution found and LP relaxation bound (Initial) and best solution found and best bound found after 30 minute time limit (Best), for each method that provides bounds.

	Opt-Orig		*Opt-Reform*		*Opt+Seq*		*Seq-Rel*	
Instance type	**Initial**	**Best**	**Initial**	**Best**	**Initial**	**Best**	**Initial**	**Best**
Base	15.3%	0.7%	10.9%	0%	4.0%	0%	4.9%	2.3%
$\|P_a\| = 1$	19.0%	0%	15.0%	0%	8.9%	0%	10.2%	3.7%
$\|P_a\| = 4$	18.9%	0%	14.8%	0%	8.8%	0%	9.8%	2.4%
$comp_a = 2.0$	15.6%	0.9%	9.3%	0.9%	5.1%	0.4%	5.7%	1.8%
$\|A\| = 4$	22.8%	0.4%	20.1%	0%	13.5%	0%	14.7%	4.5%
$\|A\| = 100$	18.4%	7.1%	14.2%	5.4%	5.3%	1.4%	5.6%	2.3%
$dmax_a = 18$	19.3%	5.1%	12.2%	0%	7.3%	0%	10.7%	6.0%
max_delay $= 2$	7.1%	0%	5.0%	0%	2.0%	0%	2.2%	0.4%
max_delay $= 9$	28.2%	3.9%	21.8%	0%	13.7%	0%	17.1%	7.6%
ASF $= 0$	17.4%	0.7%	13.2%	0%	6.4%	0%	7.2%	2.3%
AWS $= 1$	16.5%	0.8%	12.2%	0%	5.3%	0%	6.1%	2.2%
$\|P_J\| = 10$	18.8%	0.7%	14.6%	0%	7.0%	0%	8.0%	3.9%
Average	18.1%	1.7%	13.6%	0.5%	7.3%	0.2%	8.5%	3.3%

attackers, the impact of approximating the objective by not properly considering attackers becomes less significant.

We additionally analyze cases where the instance structure is more or less suited for heuristic methods. We find that one parameter, the maximum arc delay for each mitigation, has a relatively significant impact on heuristic performance. We display the average optimality gap for each heuristic method on these instances in Figure 3.1. We varied this parameter between $\frac{1}{6}$, $\frac{1}{3}$, and $\frac{3}{4}$ of the maximum arc duration parameter 12, and found that when a mitigation is only allowed to affect each arc a small amount, the heuristic methods generally perform better. Similarly, when a mitigation can greatly affect arc lengths, the heuristics perform poorly. This makes sense as when mitigations have larger impacts on individual arcs, the importance of choosing good arcs to interdict increases. Conversely, this shows that when mitigations have small impacts on the arcs they interdict, it is more reasonable to use a heuristic method.

To further analyze the benefit of the integrated model over heuristics based on preexisting methods, we consider two metrics of solution quality. The first of these is the number of times an attacker uses an arc on their critical path before the defender interdicts that

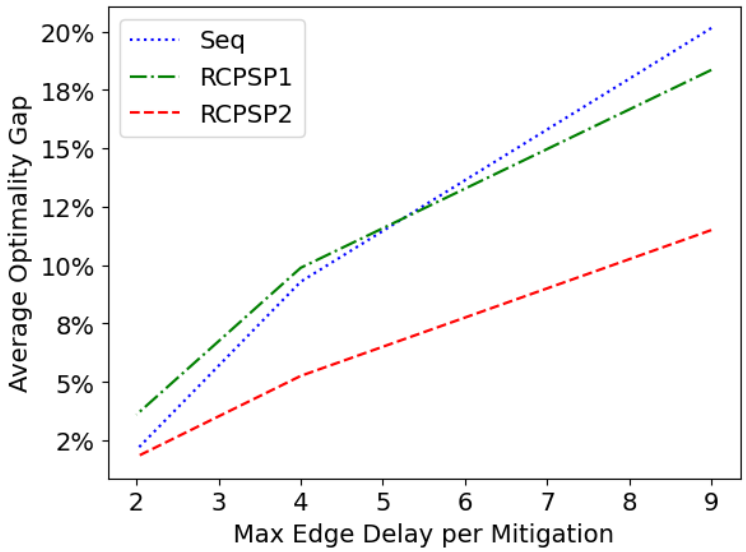
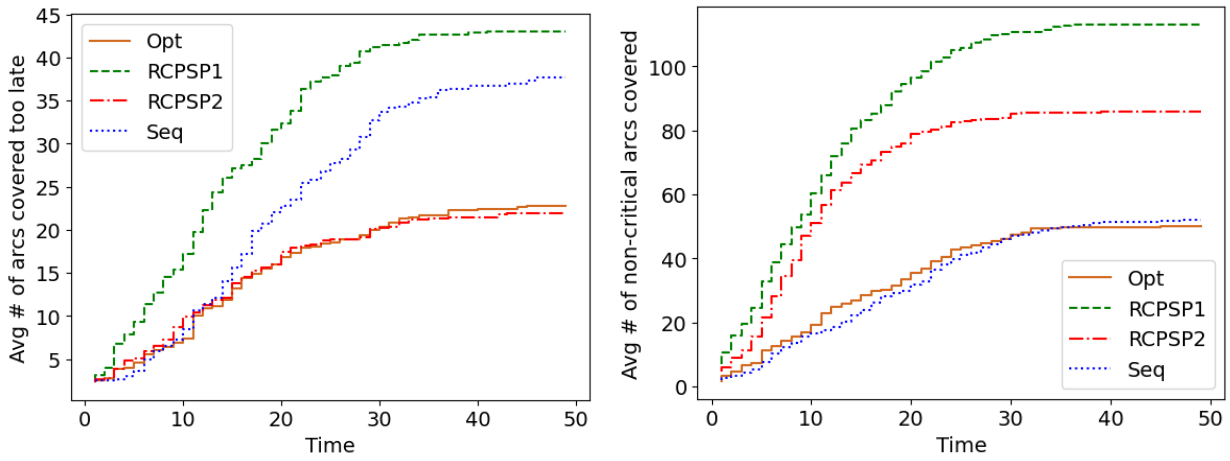


Figure 3.1: Heuristic performance on instances varied by maximum arc delay

arc. Methods that have a high number for this demonstrate that despite possibly choosing good arcs to interdict, they fail to properly prioritize timing. The second metric is the number of arcs interdicted that are not on the critical path. While, given multiple attackers, interdicting a variety of arcs is useful, arcs interdicted that are not on an attacker's critical path ultimately do not affect the completion time for that attacker. Thus a reduction in this number, while maintaining a good objective value indicates a method that well prioritizes arcs. A method with a relatively high number for this metric demonstrates that it does not do as well at choosing important arcs to delay.

We display the accumulation of these two values over time for each heuristic method solution as well as the optimal solution in Figure 3.2. The results are averaged over all instances except the variations with 100 attackers, which we display separately in Figure 3.3. Because the instances with 100 attackers have significantly more attackers, the metrics we are considering vary slightly less across methods, and also naturally take larger values. This causes these instances to disproportionately affect the averages when included. We thus keep the instances with 100 attackers separate to better demonstrate the performance on all other instances. In Figure 3.2, we can see that the simple RCPSP method does poorly for both metrics. The sophisticated RCPSP does well at timing when mitigations are completed, but poorly considers which mitigations to choose. The sequential model reverses this by

choosing good mitigations while doing poorly at scheduling those mitigations. We finally see that the integrated model does well in both of these metrics, demonstrating that it overcomes the issues other preexisting methods have for this problem. In Figure 3.3, we see similar results, with the notable exception of the sequential model performing well on both metrics.

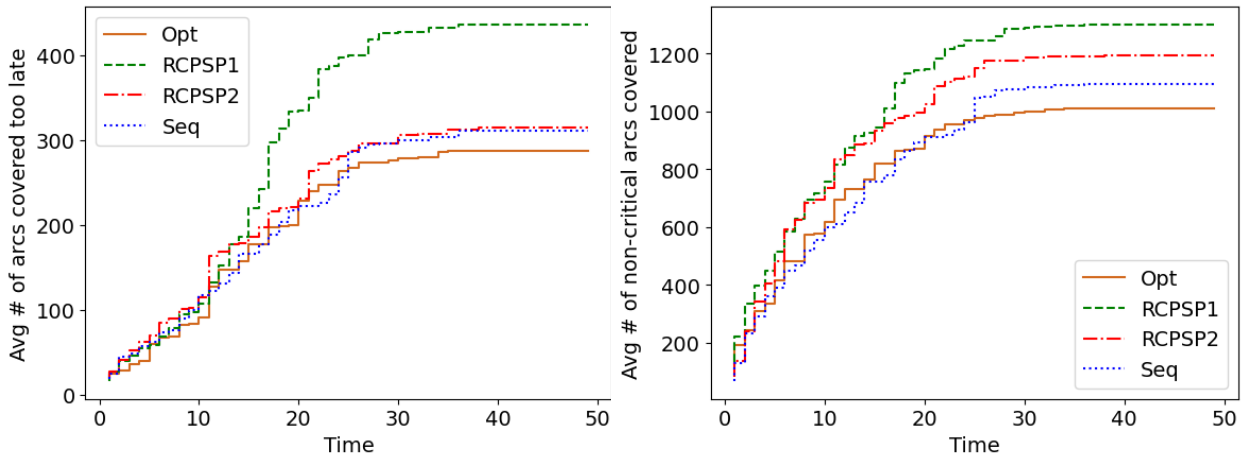


(a) Arcs used by attacker before interdiction (b) Arcs interdicted but not used by attacker

Figure 3.2: Two metrics over time for each proposed method averaged over all instances except those with the 100 attacker variation.

3.6 Conclusions and Future Directions

In this chapter we approach a network interdiction problem of delaying attacker projects, with the added complexity of the defender needing to schedule their actions with resources over time. This notion of simultaneous scheduling between the defender and attackers necessitated the development of a new model for this problem, which we demonstrated methods of reformulation in order to improve. This work thus provides useful techniques for formulating any IP models for problems that involve both an attacker and defender simultaneously completing time-based actions. Our computational studies suggest that this could be a useful model, and time saved from our reformulations are necessary to solve more difficult instances in a reasonable amount of time.



(a) Arcs used by attacker before interdiction (b) Arcs interdicted but not used by attacker

Figure 3.3: Two metrics over time for each proposed method, on averaged over instances with 100 attackers.

Our computational studies demonstrate a limitation to solvable instance size. Even with reformulations, we find that scaling up the number of attackers or the size of the defender network leads to longer run times. With a combination of these larger variations, instances may no longer be solvable in a reasonable amount of time. We attempted to address this with various decomposition methods, but the *Opt+Seq* model outperformed all decomposition methods tested. Thus this work could be extended by developing alternative solution methods or heuristic approaches for harder problems.

4 SCHEDULING MITIGATIONS TO DELAY SHORTEST PATH ATTACKS

4.1 Introduction

Protecting a system of critical infrastructure from threats can require considering multiple different ways the system can be disrupted. An adversary trying to disrupt the system may have several options of ways to exploit the system to complete their attack. A defender of this system must be strategic about what adversary actions to secure against to best avoid attack. In this chapter, we explore a model in which a defender of a system of critical infrastructure must decided how to deploy mitigations to delay a set of attackers each looking for the fastest method to disrupt the system. Each attacker's problem is modeled as an attack graph that provides multiple ways the attacker can achieve an end goal. In this model, an attacker is able to choose a single route along the attack graph to reach their goal, and thus the attacker's problem is modeled as a shortest path problem. The defender's goal is then to maximize the shortest paths of each attacker rather than the longest paths. This is a shortest path network interdiction problem, however, we extend beyond current literature in this area by considering multiple attackers and resource constrained project scheduling to model the defender's decisions.

This method of attack modeling where the attacker needs only find a single path through an attack graph is seen in papers such as Khouzani et al. (2019) and Kordy et al. (2014). In Kordy et al. (2014) we see a variety of methods of attack modeling, with attack graphs being one of them. Nodes of an attack graph represent states of an attack and edges represent actions that can take them from one state to the other. Often, as seen in Khouzani et al. (2019), the edges have associated probabilities and the attacker seeks a maximum reliability path, representing the most likely way for them to succeed at an attack. This can be modeled as a shortest path, if the probabilities for each edge are independent. We see this in DuBois et al. (2023), where multiple attackers looking to complete a maximum reliability path are considered. However, as mentioned in Kordy et al. (2014), these edges could also be associated with an amount of time, and the attacker's shortest path is the fastest way they can complete an attack. In this chapter we consider attack graphs with time-based edges and assume the defender can delay these edges by implementing mitigations. This is an example of a shortest path network interdiction problem such as seen in Israeli and

Wood (2002). Here they assume a leader can interdict arcs to either increase their length or remove them entirely, in order to maximize the shortest path of the follower. We look at a similar problem, but add to the leader's problem by considering more complex interdiction actions that need to be implemented with resources over time.

Network interdiction is a well-studied field. Smith et al. (2013) gives an overview of network interdiction literature. Given these problems can be difficult to solve, various techniques for solving network interdiction problems have been studied (Wei and Walteros, 2022, 2024; Nguyen et al., 2023; Tanınmış and Sinnl, 2022; Contardo and Sefair, 2022; Weninger and Fukasawa, 2025). Shortest path interdiction in particular is a well studied network interdiction problem, with literature including several variations of shortest path network interdiction and methods of solving these problems (Israeli and Wood, 2002; Holzmann and Smith, 2021; Song and Shen, 2016; Sefair and Smith, 2016). In the literature, the interdicter is typically modeled with a set of one or more knapsack constraints providing their interdiction budget. We however, expand this idea by considering resource-constrained defender projects over time. There exists network interdiction literature that studies time-based interaction between the defender and attacker by considering a more dynamic approach (Lunday and Sherali, 2010; Sefair and Smith, 2016; Gutin et al., 2015; Borrero et al., 2016). However, these do not include the complexity of multiple resource and precedence constraints connecting the jobs of the defender in combination with a set of multiple attackers to delay.

The problem of delaying the shortest paths of a set of attackers, with the defender subject to resource constrained project scheduling constraints, shares many similarities with Chapter 3. In Chapter 3, by delaying attacker projects, our defender is delaying the attackers' longest path, which results in a very similar attacker model. We thus can use many of the formulation techniques as in Chapter 3 as a starting point. However, because the attackers' problem is now a minimization problem instead of a maximization problem, developing a single model for the problem in this chapter becomes more difficult. Simply taking the dual of the attackers' problem results in variable multiplication in the objective when combining it with the defender model. We thus explore ways to best formulate the model in a way that achieves a linear objective. Additionally, a question arises in whether a bilevel model as we used in Chapter 3 is appropriate in the case of the attackers searching for a shortest path. We address the potential shortcomings of the bilevel approach and

justify why using this modeling method still gives interesting solutions. Finally, while shortest path network interdiction literature already exists to help address some of these issues, by including multiple attackers and a time-based defender, the problem we discuss in this chapter requires additional modeling considerations that cannot be fully addressed by the previous work.

In this chapter we begin by introducing the problem, and discussing modeling approaches and the assumptions we make in Section 4.2. We then develop a two stage integer programming model, and discuss ways to combine this min-max problem into a single IP model, in Section 4.3. In Section 4.4, we develop valid inequalities to improve the LP relaxation bounds for the models we propose. In Section 4.5 we develop an approximate model for this problem that can be useful as a heuristic. Finally we perform computational experiments to evaluate the benefit of our model and approaches to solving this model in Section 2.4.

4.2 Problem Description and Modeling Considerations

We consider a defender and a set A of attackers. Each attacker $a \in A$ has various actions they can complete in order to complete an attack on the defender's system. These actions are represented by an attack graph with arcs $\mathcal{A}^a$ and nodes $\mathcal{N}^a$, where each arc $(i,j) \in \mathcal{A}^a$ has cost of the duration d_{ij} of the associated action. A path along this graph gives an attack plan with cost of the amount of time required to complete the attack. Each attacker seeks the shortest path along their attack graph.

The defender has a set J of jobs they can complete in order to interdict attacker arcs. Each job can interdict multiple arcs, and each arc can be interdicted by multiple jobs. We denote the amount of delay job $m \in J$ inflicts upon arc (i,j) as δ_{ijm}, and assume an arc can only be delayed by a maximum of $\bar{\delta}_{ij}$. In order to complete these jobs, the defender must invest resources over time. The defender has a set R of resources and each job m requires c_{ijr} per period over τ_m time periods. We also assume jobs are related through precedence relationships given by P; a precedence $(m,n) \in P$ gives that job m must be completed before job n can begin. We finally assume that the defender has a non-time-dependent budget of B as well, where each job m has cost C_m.

We assume the defender and attacker both work to complete their jobs and actions

simultaneously. In the previous chapter, we were able to model this as a bilevel problem, where the defender chooses a schedule and the attack completes their project based on the resulting arc durations as the defender's solution affects them each period. This works because the attacker has no true decision making. They begin each action as soon as is possible, thus their solution is entirely predictable and we need not consider the defender's response to the attacker's actions at each period.

In this chapter, we consider attackers who get to choose a path to take along their graph. This means that an attacker could take an unexpected path, resulting in the defender being able to adapt their scheduling decisions to better mitigate the newly expected actions of the attacker. As such, this problem becomes a multistage problem, where decisions are reevaluated at each period based on the actions of the opposing player(s). Nonetheless, we still choose to model this as a bilevel problem, where the defender chooses a schedule and the attacker then chooses a path based on this schedule. This still captures the interplay of timing between the defender and the attackers, since the attackers' arc durations still depend on when mitigations are completed by the defender. A bilevel model only misses out on the ability of the defender to adapt their plans in the case the attacker chooses an unexpected path, and as a result the attacker has no need to adapt their paths based on defender actions since they know the defender's plan from the start. We nonetheless use this approach with the idea that it is a more pessimistic approach for the defender.

To see how this is pessimistic for the defender, consider the bilevel formulation. We assume the defender chooses a schedule first. Each attacker then knows this schedule and chooses the best path through their attack graph, given the interdiction plan made by the defender. Here we optimize assuming that the attacker makes the best decisions possible with complete information about the schedule the defender has chosen. Thus, although the attackers may actually make different decisions than we assume with this model (because the attacker does not actually have such information), they cannot do better than the objective assumes the attacker will be able to achieve. The model is pessimistic from the defender's perspective relative to a dynamic model, as the defender chooses their entire plan first without information about the attackers' true decisions, whereas the attackers are assumed to be aware of the full plan of the defender before beginning their attack. In this way, the defender has less information than they would in reality, while each attacker has more information. Note that when applying this model to a real problem, the defender

could resolve an updated model after observing the attacker actions and adjust any plans as possible. However, sticking with the original plan given by the originally solved bilevel model can result in no worse of an objective, since the model assumes the attacker acts optimally for the given plan. Being able to change the plan after a period time can only give a better expected result, as otherwise the plan would not be changed. Thus, by starting with the plan given by the bilevel model, the defender guarantees themself at least that good of an objective.

4.3 Model

We develop a bilevel integer programming model of this problem. The upper level is the defender's problem. This is very similar to the defender's problem in Chapter 3, with only some minor adjustments. We first introduce binary decision variables x_{mt} that indicate if job $m \in J$ has been completed at period $t \in \mathcal{T}$, where $\mathcal{T}$ is the time horizon for the defender to complete jobs in. We can also define variables z_{ijt} that give the delay on arc (i, j) as of period t. While these two variables capture all the information we need, we cannot simply use z to define durations in our attacker problem as we need to deal with the fact that these values change over time. To account for this, we create a time-indexed attack graph, which we will describe further in Section 3.2 when we describe the attacker's problem. In this attack graph, each arc $(i, j) \in \mathcal{A}$ gives rise to several arcs $((i, t), (j, s))$ that comprise a set $\mathcal{E}$, where $s - t$ gives a possible duration of the arc (i, j). With this setup, rather than lengthening arcs according to z, we enforce that an arc $((i, t), (j, s)) \in \mathcal{E}$ is only traversable by the attacker if $s - t \leqslant d_{ij} + z_{ijt}$. That is, the defender can only force the attacker to take a longer arc if they have interdicted it to that length by the time the attacker begins traversing the arc.

Since our goal is to delay the attacker as much as possible, we assume it is possible the attacker's shortest path may be longer than the time horizon in consideration for the defender. Thus we define T_A as the longest path in the graph given all mitigations are completed. This insures that all possible paths exist in the time indexed network. We let $\mathcal{T}_A$ be the set $\{1, ..., T_A\}$ of all time periods for the attacker.

As seen in Chapter 3, reformulation to remove extra time indexing after the end of the defender's time horizon can be useful, so we use that reformulated model here. We describe

how the time indexed network is designed in further detail in the attackers' problem. In order to use this model, we must include an extra index for every variable indexed by a time indexed arc, to represent the arc length. Define $\{\hat{\delta}_{ij}^{\ell}\}_{\ell=1}^{L_{ij}}$ as the sequence of possible delays for $(i,j) \in \mathcal{A}$ given all mitigation combination options, where $0 = \hat{\delta}_{ij}^{1} < \hat{\delta}_{ij}^{2} < ... < \hat{\delta}_{ij}^{L_{ij}} = \bar{\delta}_{ij}$. Then $\mathcal{L}_{ij}^{ts} \subset \{1, ..., L_{ij}\}$ is the indices of valid delays for time indexed arc $((i,t),(j,s))$. We note that before the end of the time horizon, that is, $s \neq T$, then $|\mathcal{L}_{ij}^{ts}| = 1$ as it contains only the single $\hat{\delta}_{ij}^{\ell}$ such that $d_{ij} + \hat{\delta}_{ij}^{\ell} = s - t$.

We use the following variables in the defender's model:

- $x_{mt} = 1$ if job $m \in J$ is schedule in period $t \in \mathcal{T}$, else 0
- z_{ijt} = delay on arc $(i,j) \in \mathcal{A}$ as of period $t \in \mathcal{T}$
- $\rho_{ijts\ell} = 1$ if the time indexed arc $((i,t),(j,s)) \in \mathcal{E}$ with delay $\hat{\delta}_{ij}^{\ell}$ is the correct arc given the amount of delay z_{ijt} on arc (i,j) at period $t \in \mathcal{T}$.

The defender's RCPSP based constraints to schedule the mitigations are as follows:

$$\sum_{t\in\mathcal{T}} x_{mt} \leqslant 1 \qquad \forall m \in J \tag{4.1a}$$

$$x_{mt} = 0 \qquad \forall m \in J, t = 1, .., \tau_m - 1 \tag{4.1b}$$

$$\sum_{m\in J} \sum_{s=t}^{t+\tau_m-1} c_{mr} x_{ms} \leqslant b_{rt} \qquad \forall t \in \mathcal{T},\ r \in R \tag{4.1c}$$

$$\sum_{t\in\mathcal{T}} \sum_{m\in J} C_m x_{mt} \leqslant B \tag{4.1d}$$

$$\sum_{s=1}^{t} x_{ms} \leqslant \sum_{s=1}^{t-\tau_m} x_{ns} \qquad \forall (m,n) \in P,\ t \in \mathcal{T}. \tag{4.1e}$$

Since these constraints are common to all formulations in this chapter, we define X as the set of x that satisfies the constraints (4.1).

Define $Y(\rho)$ as the solution to the attacker's problem given the defender's choice of ρ. We then have the following formulation of the defender's problem:

$$\max \sum_{a \in \mathcal{A}} p_a Y^a(\rho) \tag{4.2a}$$

$$\text{s.t. } x \in X$$

$$z_{ijt} \leqslant \sum_{m \in J} \delta_{ijm} x_{mt} + z_{i,j,t-1} \qquad \forall t \in \mathcal{T},\ (i,j) \in \mathcal{A} \tag{4.2b}$$

$$z_{ijt} \leqslant \bar{\delta}_{ij} \qquad \forall t \in \mathcal{T},\ (i,j) \in \mathcal{A} \tag{4.2c}$$

$$z_{ij0} = 0 \qquad \forall (i,j) \in \mathcal{A} \tag{4.2d}$$

$$\sum_{\ell \in \mathcal{L}_{ij}^{kh}} \hat{\delta}_{ij}^{\ell} \rho_{ijkh\ell} \leqslant z_{ijk} \qquad \forall ((i,k),(j,h)) \in \mathcal{E} \tag{4.2e}$$

$$\sum_{h:((i,k),(j,h)) \in \hat{\mathcal{E}}} \sum_{\ell \in \mathcal{L}_{ij}^{kh}} \rho_{ijkh\ell} = 1 \qquad \forall (i,k) \in \mathcal{V}, j \in \mathcal{N} : (i,j) \in \mathcal{A} \tag{4.2f}$$

$$x_{mt} \in \{0,1\} \qquad \forall j \in J,\ t \in \mathcal{T} \tag{4.2g}$$

$$z_{ijt} \geqslant 0 \qquad \forall (i,j) \in \mathcal{A},\ t \in \mathcal{T} \tag{4.2h}$$

$$\rho_{ijkh\ell} \in \{0,1\} \qquad \forall ((i,k),(j,h)) \in \hat{\mathcal{E}} \ell \in \mathcal{L}_{ij}^{kh}. \tag{4.2i}$$

Here the defender's objective function (4.2a) maximizes the solution to the attacker's problem. Constraints (4.2b) enforce that z_{ijt} updates each period $t \in \mathcal{T}$ based on mitigations implemented that period. Constraints (4.2c) enforces the delay of arc (i,j) is no more than the given maximum for that arc. Constraints (4.2d) enforce that all arc delays begin at zero at the start of the time horizon. Constraints (4.2e) and (4.2f) set the ρ variables. With constraints (4.2e), for each arc $(i,j) \in \mathcal{A}$ and period $t \in \mathcal{T}_A$ we are allowed to set $\rho_{ijts} = 1$ for any arcs $((i,t),(j,s)) \in \mathcal{E}$ with delay $\hat{\delta}_{ij}^{\ell}$ less than the current delay z_{ijt}. Then constraints (4.2f) enforce that for each arc (i,j) and period t, exactly one arc $((i,t),(j,s)) \in \mathcal{E}$ can be chosen with $\rho_{ijts} = 1$. Together with the fact that this model is maximizing the longest path in the attacker graphs, these enforce that the longest arc $((i,t),(j,s))$ with delay $\hat{\delta}_{ij}^{\ell} \in \mathcal{L}_{ij}^{ts}$ at most z_{ijt} will have $\rho_{ijts\ell} = 1$. Since z_{ijt} will be set to be a valid amount of delay, this implies that $\rho_{ijtsz_{ijt}} = 1$. Constraints (4.1a) enforce each job can be completed at most once. Constraints (4.1b) enforce each mitigation cannot be completed earlier in the time horizon than its duration. Constraints (4.1c) enforce the resource constraints for each period and (4.1d) enforces the overall budget constraint. Lastly, we have (4.1e) enforcing

the precedence constraints for the defender.

The objective of the defender's problem is to maximize the time it takes for the attacker to complete a path in the attack graph. The attacker's goal is to find the shortest path through the attack graph defined by a set of nodes $\mathcal{N}$ and arcs $\mathcal{A}$. We assume the attacker starts at some beginning node 0 and reaching an end node N represents the completion of the entire graph. As in Chapter 3, the defender's actions result in the attackers' graphs having arc lengths that depend on when they reach them. This leads us to use a time-indexed attacker network. We can then model an attacker's problem as a shortest path problem along a time indexed network. Again, we use the reformulated version of the time-indexed network from Chapter 3.

Let $\mathcal{V}$ and $\mathcal{E}$ be the nodes and arcs available to the attacker in the time indexed graph. We have $\mathcal{V} \subseteq \{(i,t) : i \in \mathcal{N}, t \in \mathcal{T}\} \cup \{(N, T_A + 1)\}$, where node (i,T) represents being at node $i \in \mathcal{N}$ at any period between T and T_A. Define $\kappa(t)$ for $t \in \mathcal{T}_A$ as the time indexed node that represents period t. We'll have $\kappa(t) = t$ for $t < T$ and $\kappa(t) = T$ for $t \geqslant T$. Then $\mathcal{E} = \{((i,t),(j,s)) : (i,j) \in \mathcal{A}, (i,t),(j,s) \in \mathcal{N}, \exists\, \ell \text{ s.t. } \kappa(t + \hat{\delta}^{\ell}_{ij}) = s\}$. These are all valid arcs and we can define our formulation based on this set, however since not all nodes are reachable we can further decrease the number of nodes and arcs. For example, suppose i is a node with only node 0 as a predecessor and $d_{0i} = 2$, node $(i,1)$ is then not reachable. Several more nodes may be similarly unreachable. To create $\mathcal{V}$ and $\mathcal{E}$ then we can recursively build the graph, starting with node $(0,0)$ and adding arcs $((0,0),(i,\kappa(d_{0i} + \hat{\delta}^{\ell}_{0i})))$ for each $(0,i) \in \mathcal{A}$, $\ell = 1, ..., L_{0i}$. We can create a topological sort of the nodes in $\mathcal{N}$ according to $\mathcal{A}$ and consider each node, adding arcs as we did with the arcs from $(0,0)$. Since we go in the order of the topological sort, each time we do this for a node i, we know that this node will have no other ways to reach it, so that any (i,t) not reached by the time of processing node i, will not be included in $\mathcal{V}$. We define also the end node $(N, T_A + 1)$, with arcs $((N,t),(N,T_A + 1)$ for each $t \in \mathcal{T}$. This is our end node, used as an indicator that node N has been reached.

We can then formulate this as a shortest path problem over the time-indexed graph, where only arcs allowed given the defender's choices can be used by the attacker. This results in the following model for each attacker $a \in A$ using flow variables $y^{a}_{ijts\ell}$ to represent if attacker a uses the time indexed arc $((i,t),(j,s)) \in \mathcal{E}_a$ with delay $\hat{\delta}^{\ell}_{ij}$ for $\ell \in \mathcal{L}^{ts}_{ij}$:

$$\Upsilon^a(\rho) = \tag{4.3a}$$

$$\min \sum_{((i,t),(j,s))\in\mathcal{E}_a} \sum_{\ell\in\mathcal{L}_{ij}^{ts}} (d_{ij} + \hat{\delta}_{ij}^{\ell}) y_{ijts\ell}^a \tag{4.3b}$$

$$\text{s.t.} \sum_{(j,s):((j,s),(i,t))\in\mathcal{E}_a} \sum_{\ell\in\mathcal{L}_{ji}^{st}} y_{jist\ell}^a - \sum_{(j,s):((i,t),(j,s))\in\mathcal{E}_a} \sum_{\ell\in\mathcal{L}_{ij}^{ts}} y_{ijts\ell}^a = q_{it} \qquad \forall (i,t)\in\mathcal{V}_a \tag{4.3c}$$

$$y_{ijts\ell}^a \leqslant \rho_{ijts\ell} \qquad \forall ((i,t),(j,s))\in\mathcal{E}_a, \ell\in\mathcal{L}_{ij}^{ts} \tag{4.3d}$$

$$y_{ijts\ell}^a \geqslant 0 \qquad \forall ((i,t),(j,s))\in\mathcal{E}_a, \ell\in\mathcal{L}_{ij}^{ts}, \tag{4.3e}$$

where $q_{N,T_A+1} = 1$, $q_{0,0} = -1$, and $q_{i,t} = 0$ otherwise.

Here the attacker's objective (4.3b) is to minimize the length of the arcs taken, where the length of an arc corresponding to $y_{ijts\ell}$ is given by the sum of its initial duration d_{ij} and the amount of imposed delay $\hat{\delta}_{ij}^{\ell}$. Constraints (4.3c) are flow balance constraints where each arc is represented by not only its start and end node but also its length. Finally, constraints (4.3d) enforce that arcs that are shorter than the true lengths given by the defender's problem are not usable by the attacker.

Currently, we have a maximization problem for the defender and a minimization problem for the attackers. In order to combine the attacker and defender models into one combined model, we need to have a maximization problem for the attackers. To do so, we can take the dual of the attackers problem. Let γ^a be the dual variables associated with constraints (4.3d) and π^a be the dual variables associated with the constraints (4.3c). Then we have the following dual:

$$\max \pi_{N,T_A+1}^a - \pi_{0,0}^a + \sum_{((i,t),(j,s))\in\mathcal{E}_a} \sum_{\ell\in\mathcal{L}_{ij}^{ts}} \rho_{ijts}\gamma_{ijts\ell}^a \tag{4.4a}$$

$$\text{s.t. } \gamma_{ijts\ell}^a - \pi_{it}^a + \pi_{js}^a \leqslant d_{ij} + \hat{\delta}_{ij}^{\ell} \qquad \forall ((i,t),(j,s))\in\mathcal{E}_a, \ell\in\mathcal{L}_{ij}^{ts} \tag{4.4b}$$

$$\gamma_{ijts\ell}^a \leqslant 0 \qquad \forall ((i,t),(j,s))\in\mathcal{E}_a. \tag{4.4c}$$

This dual matches the form of typical shortest path problem duals, with the addition

of dual variables γ^a corresponding with the arc removal constraints. We note, that as is generally done for duals of shortest path problems, we can set $\pi^a_{0,0} = 0$, and will do this for all future formulations. With the dual formulation we now have a maximize problem which we could combine with our defender's maximize problem. However, here we are multiplying a defender variable ρ by an attacker variable γ. Thus if we were to combine the two problems as is we would have a non-linear model. We prefer to deal with linear models, so we continue to investigate ways to reformulate this attacker model.

4.4 MIP Formulations

We desire to combine the attacker and defender models in to a single combined model that can be solved with an IP solver. However, with the proposed forms of the attacker and defender problems (4.2) and (4.4), combining them would result in variable multiplication. We thus explore methods to fix or avoid this issue. Our first approach uses McCormick linearization to directly deal with the variable multiplication. We then simplify this into an approach used in Israeli and Wood (2002), that adjusts modeling in the attackers' primal to avoid variable multiplication in the dual. We finally propose a simpler heuristic model that uses an approximation of the attacker's problem to provide ideally close to accurate objectives without the big-M type of constraints needed in the previous two models.

McCormick Linearization

We begin by reformulating the model for each attacker $a \in A$ with a natural approach to variable multiplication: McCormick Relaxation. Since our ρ variables are binary, this will exactly model the multiplied variables, and does not result in only a relaxation. In order to do this, we need to have bounds on the γ^a variables. We already have an upper bound of zero for each of these variables from (4.4c), so we need only derive a lower bound. We do so by considering the role of these variables in the attackers' dual problem (4.4), given how it compares to a typical shortest path dual formulation.

If $\rho_{ijts\ell} = 1$ for all $((i,t),(j,s)) \in \mathcal{E}_a$, then we would have a normal shortest path formulation, the dual of which could be obtained from (4.4) by setting $\gamma^a_{ijts\ell} = 0$ for all $((i,t),(j,s)) \in \mathcal{E}_a$, $\ell \in \mathcal{L}^{ts}_{ij}$. But if $\rho_{ts\ell} = 0$ for any $((i,t),(j,s)) \in \mathcal{E}_a$, then for all $\ell \in \mathcal{L}^{ts}_{ij}$, we

can set $\gamma^a_{ijts\ell} = 0$ without directly affecting the objective. Thus this allows us to set $\gamma^a_{ijts\ell}$ as negative as necessary to satisfy the corresponding constraint. In this way, the γ^a dual variables can be seen as a way to ensure the constraint corresponding to an arc that has been interdicted does not affect the attacker's solution.

We know that $\pi_{js} - \pi_{it} \leqslant T_A$. Since T_A is the attackers' time horizon, defined as the longest possible length of any path in any attacker's network, we know the time between the attacker reaching any two nodes must not be more than T_A. We can thus get a lower bound of $\gamma^a_{ijts\ell} \geqslant -T_a + d_{ij} + \hat{\delta}^\ell_{ij}$ for $((i,t),(j,s)) \in \mathcal{E}_a, \ell \in \mathcal{L}^{ts}_{ij}$.

Then we can apply McCormick linearization to this model by introducing a new set of variables u^a that represent the product of the ρ and γ^a variables. To do so we add the following inequalities and replace $\rho_{ijts\ell}\gamma^a_{ijts\ell} = u^a_{ijts\ell}$ for each $((i,t),(j,s)) \in \mathcal{E}_a, \ell \in \mathcal{L}^{ts}_{ij}$:

$$u^a_{ijts\ell} \geqslant (-T_a + d_{ij} + \hat{\delta}^\ell_{ij})\rho_{ijts} \qquad \forall((i,t),(j,s)) \in \mathcal{E}_a, \ell \in \mathcal{L}^{ts}_{ij} \tag{4.5a}$$

$$u^a_{ijts\ell} \geqslant \gamma^a_{ijts\ell} \qquad \forall((i,t),(j,s)) \in \mathcal{E}_a, \ell \in \mathcal{L}^{ts}_{ij} \tag{4.5b}$$

$$u^a_{ijts\ell} \leqslant (-T_a + d_{ij} + \hat{\delta}^\ell_{ij})\rho_{ijts\ell} + \gamma^a_{ijts\ell} + T_a - d_{ij} - \hat{\delta}^\ell_{ij} \qquad \forall((i,t),(j,s)) \in \mathcal{E}_a, \ell \in \mathcal{L}^{ts}_{ij} \tag{4.5c}$$

$$u^a_{ijts\ell} \leqslant 0 \qquad \forall((i,t),(j,s)) \in \mathcal{E}_a, \ell \in \mathcal{L}^{ts}_{ij} \tag{4.5d}$$

$$-T_a + d_{ij} + \hat{\delta}^\ell_{ij} \leqslant \gamma^a_{ijts\ell} \qquad \forall((i,t),(j,s)) \in \mathcal{E}_a, \ell \in \mathcal{L}^{ts}_{ij}. \tag{4.5e}$$

We can thus combine the defender and attackers' problems into a single MILP given by:

$$\max_{x,z,\rho,\pi,\gamma,u} \sum_{a\in A} p_a\Big(\pi^a_{N,T_A+1} + \sum_{((i,t),(j,s))\in\mathcal{E}_a} \sum_{\ell\in\mathcal{L}^{ts}_{ij}} u^a_{ijts\ell}\Big) \tag{4.6}$$

$$\begin{aligned}
&\text{s.t. } x \in X \\
&\quad (4.2b) - (4.2f) \\
&\quad (4.4b) \\
&\quad (4.5) \\
&\quad \pi^a_{0,0} = 0 && \forall a \in A \\
&\quad z_{ijt} \geqslant 0 && \forall (i,j) \in \mathcal{A},\ t \in \mathcal{T} \\
&\quad \rho_{ijkh\ell} \in \{0,1\} && \forall((i,k),(j,h)) \in \hat{\mathcal{E}}\ell \in \mathcal{L}^{kh}_{ij}.
\end{aligned}$$

Israeli and Wood Linearization

We can consider an alternate method of addressing the variable multiplication in the attackers' dual (4.4) by approaching the primal definition differently. We can model the removal of arcs by imposing a sufficiently large delay, rather than enforcing zero flow on that arc. This is the form used by Israeli and Wood (2002), where interdicting an arc can either impose a delay or remove it entirely. The attacker's time horizon T_A is a sufficient delay to effectively remove the arc from the attacker's consideration. With this, the attacker minimization model would be the following:

$$\min \sum_{((i,t),(j,s))\in\mathcal{E}_a} \sum_{\ell\in\mathcal{L}_{ij}^{ts}} (d_{ij} + \hat{\delta}_{ij}^{\ell} + T_A(1-\rho_{ijts\ell}))y_{ijts\ell}^{a} \tag{4.7a}$$

$$\text{s.t.} \sum_{(j,s):((j,s),(i,t))\in\mathcal{E}_a} \sum_{\ell\in\mathcal{L}_{ji}^{st}} y_{jist\ell}^{a} - \sum_{(j,s):((i,t),(j,s))\in\mathcal{E}_a} \sum_{\ell\in\mathcal{L}_{ij}^{ts}} y_{ijts\ell}^{a} = q_{it} \qquad \forall (i,t)\in\mathcal{V}_a \tag{4.7b}$$

$$y_{ijts\ell}^{a} \geqslant 0 \qquad \forall ((i,t),(j,s))\in\mathcal{E}_a, \ell\in\mathcal{L}_{ij}^{ts} \tag{4.7c}$$

The objective of this problem is nonlinear due to variable multiplication. However, the dual is the following:

$$\max \pi_{N,T_A+1}^{a} \tag{4.8a}$$

$$\text{s.t. } -\pi_{it}^{a} + \pi_{js}^{a} \leqslant d_{ij} + \hat{\delta}_{ij}^{\ell} + T_A(1-\rho_{ijts\ell}) \qquad \forall ((i,t),(j,s))\in\mathcal{E}_a, \ell\in\mathcal{L}_{ij}^{ts} \tag{4.8b}$$

$$\pi_{0,0}^{a} = 0. \tag{4.8c}$$

This model has linear constraints and a linear objective and the objective sense matches the defender's objective – hence, this can be combined with the defender's model for a full formulation.

Israeli and Wood (2002) notes that when using this technique, having large values for the delays results in poor LP relaxations. Thus, using a delay for the removed edges smaller than T_A would be beneficial.

Reconsider the original attacker problem (4.3). Here, when $\rho_{ijts\ell} = 1$, the constraint (4.3d) for $((i,t),(j,s)) \in \mathcal{E}_a$, $\ell \in \mathcal{L}_{ij}^{ts}$ is redundant as it is implied from the shortest path formulation. Thus we could remove this constraint and remove the corresponding variable $\gamma_{ijts\ell}^a$ from the dual, as can be achieved by setting $\gamma_{ijts\ell}^a = 0$, and have an equivalent formulation. Then in (4.4a) we would find that the summation term is always zero, since if $\rho_{ijts\ell} = 1$, then $\gamma_{ijts\ell}^a = 0$, and otherwise $\rho_{ijts\ell} = 0$. We can thus remove this term from the objective and set $\gamma_{ijts\ell}^a = (-T_a + d_{ij} + \hat{\delta}_{ij}^{\ell})(1 - \rho_{ijts\ell})$, so that it equals the lower bound we derived in 4.4 whenever it is not set equal to zero.

This gives the following formulation, which matches (4.8) with smaller delays for removed arcs:

$$\max \pi_{N,T_A+1}^a \tag{4.9a}$$

$$\text{s.t. } (-T_a + d_{ij} + \hat{\delta}_{ij}^{\ell})(1 - \rho_{ijts\ell}) - \pi_{it}^a + \pi_{js}^a \leqslant d_{ij} + \hat{\delta}_{ij}^{\ell} \quad \forall ((i,t),(j,s)) \in \mathcal{E}_a, \ell \in \mathcal{L}_{ij}^{ts} \tag{4.9b}$$

$$\pi_{0,0}^a = 0. \tag{4.9c}$$

The full Israeli and Wood Linearization model is then:

$$\begin{aligned} \max & \sum_{a \in A} p_a \pi_{N,T_A+1}^a && (4.10)\\ \text{s.t. } & x \in X \\ & (4.2b) - (4.2f) \\ & (4.9b) - (4.9c) \\ & z_{ijt} \geqslant 0 && \forall (i,j) \in \mathcal{A},\ t \in \mathcal{T} \\ & \rho_{ijkh\ell} \in \{0,1\} && \forall ((i,k),(j,h)) \in \hat{\mathcal{E}}\ell \in \mathcal{L}_{ij}^{kh}. \end{aligned}$$

We observe that this formulation has at least as good of LP relaxation bounds as the McCormick Linearization model (4.6).

Proposition 4.1. *The LP relaxation of* (4.10) *has optimal objective value no more than the optimal objective value of* (4.6).

Proof. We show this by demonstrating that for any solution to (4.10), there is a solu-

tion to (4.6) with equal objective. Let $(\hat{x}, \hat{z}, \hat{\rho}, \hat{\pi}^a)$ be a solution to (4.10), with objective $\sum_{a \in A} p_a \hat{\pi}^a_{N,T_A+1}$. Then set $\gamma^a_{ijts\ell} = (-T_a + d_{ij} + \hat{\delta}^\ell_{ij})(1 - \hat{\rho}_{ijts\ell})$ and $u^a_{ijts\ell} = 0$ for all $a \in A$, $((i,t),(j,s)) \in \mathcal{E}_a$, $\ell \in \mathcal{L}^{ts}_{ij}$. Then $(\hat{x}, \hat{z}, \hat{\rho}, \hat{\pi}, \gamma, u)$ provides a solution to (4.6), with objective $\sum_{a \in A} p_a \left(\hat{\pi}^a_{N,T_A+1} + \sum_{((i,t),(j,s)) \in \mathcal{E}_a} \sum_{\ell \in \mathcal{L}^{ts}_{ij}} u^a_{ijts\ell}\right) = \sum_{a \in A} p_a \hat{\pi}^a_{N,T_A+1}$. □

We conjecture that these LP relaxations in fact have equal optimal objective values, as is suggested by the computational results in Section 4.6.

Valid Inequalities

The model (4.9) provides a formulation without the extra variables needed for the McCormick linearization. However, these delays need to be relatively large to ensure it is not optimal for the attacker to use an arc that should not be feasible for them. Consider an arc $(i,j) \in \mathcal{A}_a$, and a period $t \in \mathcal{T}$. There may exist time indexed arcs $((i,t),(j,s))$ and $((i,t),(j,\hat{s}))$, $\hat{s} > s$, representing arc (i,j) at period t with varying lengths. Suppose $((i,t),(j,\hat{s}))$ is the arc chosen by the defender with the correct interdicted arc length at period t. A future arc $(j,k) \in \mathcal{A}_a$ may not be interdicted at period s, but is interdicted in period $\hat{s}$. Then if the delay imposed on arc $((i,t),(j,s))$ was too small, this model could give a shorter path for the attacker by taking arc $((i,t),(j,s))$ for the delayed amount instead of the arc that is actually available: $((i,t),(j,\hat{s}))$. By doing so, the attacker could then take arc (j,k) from period s for an uninterdicted length.

This issue prevents us from giving tight bounds for γ^a and forces us to use large delays in (4.8) that may result in poor LP relaxation bounds. However, this issue only arises for time indexed nodes before the end of the defender time horizon. Time-indexed nodes of the form (i,T), $i \neq N$, can only have time-indexed arcs to nodes of the form (j,T). Here the aforementioned issue cannot occur, and the delays imposed for removed arcs need only be large enough to enforce the length of the arc is the length provided by the defender problem.

Because this issue may only impact a small set of arcs, particularly in instances with relatively short defender time horizons, we develop a set of valid inequalities for the arcs not affected by this issue. These inequalities allow us to use the current arc duration to set the distance between time-indexed nodes $(i,t),(j,s) \in \mathcal{V}$ when there is only one possible time-indexed node that arc $(i,j) \in \mathcal{A}$ leads to in period $t \in \mathcal{T}$. We note that this is

the case for any arc that is not interdictable by the defender as well as any arc such that $t + d_{ij} \geqslant T$. This second case holds true because all time indexed arcs for (i, j) starting from time-indexed node (i, t) will go to the same time-indexed node (j, T), since this final period of the defender's horizon represents all times after T, and d_{ij} gives the shortest possible length of arc (i, j). We develop an approximate model in Section 4.5 that allows us to derive these constraints.

Define $\mathcal{S}$ to be the set of $(i, j) \in \mathcal{A}$, $t \in \mathcal{T}$ such that $|\{s : ((i, t), (j, s)) \in \mathcal{E}\}| = 1$, and for each $a \in A$, $\mathcal{S}_a \subset \mathcal{S}$ to be the subset that only includes arcs $(i, j) \in \mathcal{A}_a$.

We then propose that the following constraints are valid inequalities for the Israeli and Wood Linearization model (4.10):

$$-\pi^a_{it} + \pi^a_{js} \leqslant d_{ij} + z_{ijt} \qquad \forall a \in A, ((i, t), (j, s)) \in \mathcal{E}_a \text{ s.t.} (i, j, t) \in \mathcal{S}_a. \tag{4.11}$$

Proposition 4.2. *The inequalities* (4.11) *are valid for* (4.10).

Proof. For a given attacker $a \in A$, any arc $(i, j) \in \mathcal{A}_a$ and period $t \in \mathcal{T}$, with $|\{s : ((i, t), (j, s)) \in \mathcal{E}_a\}| = 1$, gives rise to constraints (4.4b) in the McCormick Linearization model (4.6):

$$\pi^a_{j,s} - \pi^a_{i,t} \leqslant d_{ij} + \hat{\delta}^\ell_{ij} - \gamma^a_{ijts\ell} \qquad \forall \ell \in \mathcal{L}^{ts}_{ij}, \tag{4.12}$$

where setting $\gamma^a_{ijts\ell} = (1 - \rho_{ijts\ell})(-T_a + d_{ij} + \hat{\delta}^\ell_{ij})$ gives us the Israeli and Wood Linearization model (4.10).

This gives that the time $(j, s) \in \mathcal{V}_a$ is reached must be $d_{ij} + \hat{\delta}^\ell_{ij} - \gamma^a_{ijts\ell}$ periods after (i, t) is reached. Because $|\{s : ((i, t), (j, s)) \in \mathcal{E}_a\}| = 1$, we know that either $s = T$ or arc (i, j) can only take on a single length. This is because if $s \neq T$, and $s \neq T_A + 1$, $\mathcal{L}^{ts}_{ij} = s - t$, if $s = T_A + 1$, $((i, t), (j, s))$ is an artificial arc created to signify the completion of the attackers path, and is not interdictable. In this case this set (4.12) of constraints exists for only one ℓ and can be simplified to $\pi^a_{j,s} - \pi^a_{i,t} \leqslant d_{ij}$. This is because $\hat{\delta}^\ell_{ij} = 0$ since only one arc length exists and it must be the original duration, and by (4.2f), $\rho_{ijts\ell} = 1$ so that $\gamma_{ijts\ell} = 0$.

If $s = T$, then $|\mathcal{L}^{ts}_{ij}| \geqslant 1$, that is, multiple arc durations may exist. However, since $|\{s : ((i, t), (j, s)) \in \mathcal{E}_a\}| = 1$, by (4.2f), we must have $\rho_{ijts\hat{\ell}} = 1$ for a singular $\hat{\ell} \in \mathcal{L}^{ts}_{ij}$. Thus $\gamma^a_{ijts\hat{\ell}} = 0$ and we have the constraint $\pi^a_{j,s} - \pi^a_{i,t} \leqslant d_{ij} + \hat{\delta}^{\hat{\ell}}_{ij}$. For all $\ell \in \mathcal{L}^{ts}_{ij}$, $\ell \neq \hat{\ell}$, $\gamma^a_{ijts\ell}$ is

set sufficiently negative to make the corresponding constraint (4.12) unrestrictive. Then the only important constraint (4.12) for arc $((i,t),(j,s))$, is the one corresponding to $\hat{\ell}$. In an optimal solution, $z_{ijt} = \hat{\delta}^{\hat{\ell}}_{ij}$, so that, we can use constraints (4.14b) in this case.

□

By including these valid inequalities (4.11) into the Israeli and Wood Linearization model (4.10), we can improve the LP relaxation. We note that since the variables ρ are only needed for the purpose of constraints (4.9b), this model can be simplified by removing $\rho_{ts\ell}$ and corresponding constraints whenever $(i,j,t) \in \mathcal{S}$. We use this simplification in when using this model in Section 4.6.

4.5 Approximate Model

We develop an approximate model that disregards the issue of attacker's using the incorrect time-indexed nodes mentioned in Section 4.4. By doing so we can impose the exact amount of delay set by the defender on all arcs. This provides the benefit of the valid inequalities of Section 4.4 on all arcs, rather than just a subset. In this section we derive those valid inequalities by first considering the attackers minimization problem in the case we allow the attacker to use the incorrect time-indexed arcs for interdicted lengths. This follows a similar idea as in Section 4.4, in that we have variable multiplication in the primal problem that does not result in variable multiplication in the dual.

We propose the following approximate model of the shortest path problem for attacker a given the defender's decisions.

$$\min \sum_{((i,t),(j,s))\in\mathcal{E}_a} (d_{ij} + z_{ijt}) y^a_{ijts} \tag{4.13a}$$

$$\text{s.t.} \sum_{(j,s):((j,s),(i,t))\in\mathcal{E}_a} y^a_{jist} - \sum_{(j,s):((i,t),(j,s))\in\mathcal{E}_a} y^a_{ijts} = q_{it} \qquad \forall (i,t) \in \mathcal{V}_a \tag{4.13b}$$

$$y^a_{ijts} \geqslant 0 \qquad \forall ((i,t),(j,s)) \in \mathcal{E}_a \tag{4.13c}$$

Here we directly use the variables z_{ijt} that capture the delay on arc $(i,j) \in \mathcal{A}$ at period $t \in \mathcal{T}$. Then any arc (i,j) taken at period t will have length $d_{ij} + z_{ijt}$. In this way, this model does suffer from the issue mentioned in Section 4.4. That is, attackers may be able

to use paths along earlier time-indexed nodes than should be possible given the defender's decisions. While the lengths of those paths as recorded in the objective would be correct, they can then avoid the interdiction of a future arc. This is thus a relaxation of the true attacker's primal problem, as the true solution for the attacker is still an option, but other shorter paths may exist as well. Because the defender is trying to maximize the shortest path of the attacker, this acts as a restriction to the defender. Thus using (4.13) in a combined model from the defender's perspective will result in objective values no greater than the true objective. However, it results in a much simpler model, which can be useful as a heuristic. We have the following dual of (4.13).

$$\max \pi^a_{N,T_A+1} \tag{4.14a}$$
$$\text{s.t. } -\pi^a_{it} + \pi^a_{js} \leqslant d_{ij} + z_{ijt} \qquad \forall((i,t),(j,s)) \in \mathcal{E}_a \tag{4.14b}$$
$$\pi^a_{0,0} = 0 \tag{4.14c}$$

This model eliminates the need for the defender's ρ variables, resulting in a simpler combined model of:

$$\max \sum_{a \in A} p_a \pi^a_{N,T_A+1} \tag{4.15a}$$
$$\text{s.t. } x \in X$$
$$z_{ijt} \leqslant \sum_{m \in J} \delta_{ijm} x_{mt} + z_{i,j,t-1} \qquad \forall t \in \mathcal{T},\ (i,j) \in \mathcal{A} \tag{4.15b}$$
$$z_{ijt} \leqslant \bar{\delta}_{ij} \qquad \forall t \in \mathcal{T},\ (i,j) \in \mathcal{A} \tag{4.15c}$$
$$z_{ij0} = 0 \qquad \forall (i,j) \in \mathcal{A} \tag{4.15d}$$
$$-\pi^a_{it} + \pi^a_{js} \leqslant d_{ij} + z_{ijt} \qquad \forall a \in A, ((i,t),(j,s)) \in \mathcal{E}_a \tag{4.15e}$$
$$\pi^a_{0,0} = 0 \qquad \forall a \in A \tag{4.15f}$$
$$x_{mt} \in \{0,1\} \qquad \forall j \in J,\ t \in \mathcal{T} \tag{4.15g}$$
$$z_{ijt} \geqslant 0 \qquad \forall (i,j) \in \mathcal{A},\ t \in \mathcal{T} \tag{4.15h}$$

Given a defender job scheduling solution x to this model (4.15), we can compute the

arc lengths to determine ρ and solve the original attackers problem (4.3) to get the true objective value.

4.6 Computational Studies

To compare the proposed formulations, and evaluate the usefulness of the proposed approximate model as a heuristic method, we test each model on a set of instances. The defender data in each of these instances is generated in the same way as in Chapter 3. We generate the attackers in a similar way, but design the network in a way more comparable to other shortest path interdiction literature. Israeli and Wood (2002) use a network described by m rows and n columns of nodes, where each node is connected to the nodes adjacent to it such that arcs go forward through the graph. We match this structure of nodes and columns for our underlying attack graph. However, to account for multiple attackers, we instead have each node of each column potentially connect to all nodes in the next column. We then assign a random subset of nodes from each column to each attacker, then assign the corresponding edges to that attacker. Once the attacker network is generated, we generate the rest of the attacker data in the same way as in Chapter 3.

We generate a set of 45 test instances in this way by varying select parameters and repeating for 5 different seeds over 9 different instance types. We list the parameters varied in table 4.1. We vary the attacker network structure, by adjusting the number of rows and columns of nodes. Because the attackers are finding shortest paths and the defenders are completing projects, we find that having longer paths for the attacker provide more interesting instances by giving the defender more time to act. For this reason we have as a base a longer 5×6 network, and defender job durations between 1 and 6 with attacker arc durations between 4 and 12. We vary each of these parameters as well to the values given in 4.1. We additionally vary the number of attackers and the maximum delay a single job can give an arc. These models can be difficult to solve, so we use smaller networks for our attackers and defender than we see in Israeli and Wood (2002) or Chapter 3. We use a defender with 8 projects and between 5 and 20 jobs per project.

Table 4.1: Parameters and variations for test instances.

Parameter	Base values	Additional values
Attacker network size, $m \times n$	5×6	6×5, 4×8
# of attackers, $\lvert A \rvert$	8	4, 20
Attacker arc durations, $[dmin_a, dmax_a]$	$[4, 12]$	$[2, 6]$, $[8, 18]$
Max delay per job	$\frac{1}{3} dmax_a$	$\frac{1}{6} dmax_a$, $\frac{2}{3} dmax_a$

Results

We test each proposed model on the set of 45 instances with a 20 minute time limit per instance. Each model was solved using Gurobi 10.0.1 and a machine with Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz. We test the following models and use the following abbreviations to refer to each.

- *Opt-MC*: The McCormick Linearization model (4.6). This is an exact solution method.
- *Opt-IW*: The Israeli and Wood Linearization model (4.10). This is an exact solution method.
- *Opt-VI*: The Israeli and Wood Linearization model (4.10), with the addition of the valid inequalities (4.11), and the removal of unnecessary variables ρ_{ij} and corresponding constraints as described at the end of Section 4.4. This is an exact solution method.
- *Approx*: The approximate model (4.15) presented in Section 4.5. This is a heuristic method.

In Table 4.2, we display the summary results for a few performance metrics for each method. In the first results column we display the average gap from the best solution found for each instances. For most instances where at least one exact method finished in the time limit, this is the gap between the value of the best found solution and the optimal value. Because some instances were not solved to optimality, our exact methods did not average a zero gap. However, *Opt-VI* most often found the best solution of the four methods. More notably, we see that *Approx* performs very well at finding optimal or near optimal solutions. This suggests that perfectly considering the simultaneous scheduling of the defender and attackers is not completely necessary, and a close approximation is

sufficient in these test instances. We note that *Approx* performs well on all instances types, ranging from a 0% to 0.2% best solution gap, achieving its worst gap on the instances where attacker arc durations are small. This makes sense, since with small arc durations attackers can generally complete their attacks before the end of the defender's time horizon. Since *Approx* is only inaccurate for the defender's time horizon, we expect *Approx* to perform worse in this scenario. Regardless, *Approx* still finds near optimal solutions even in this case.

Table 4.2: Averages for each method across objective gap, LP relaxation bound gap, and run time, as well as how many of the 45 instances failed to solve in a 20 minute time limit.

Model	Best Soln Gap	Final Bound Gap	LP Gap	Time (s)	# Exceeding Time Limit
Opt-MC	0.132%	0.729%	46.03%	660.2	19
Opt-IW	0.034%	0.161%	46.03%	369.8	9
Opt-VI	0.017%	0.114%	29.30%	398.3	9
Approx	0.084%	0.046%	0.96%	281.8	5

In the third and fourth columns of Table 4.2, we display the gaps for the final bound after the time limit and average LP relaxation bound for each method. For the exact methods, this is computed from the best solution found within the time limit by that method. For *Approx* this is computed for the objective of the approximation model, since the approximate model does not provide bounds for the true solution. We can see that all methods have small gaps at the end of the time limit, which shows that the best solution found is for each instance is usually very close to optimal. Looking at the LP relaxation bound gaps, we see that *Opt-MC* and *Opt-IW* both get the same large LP relaxation bound. As expected, the valid inequalities included in *Opt-VI* successfully decrease this bound. While the model *Approx* does not provide bounds on the true objective value, we see that its LP relaxation bound is very close to its objective value, motivating the benefit of the proposed valid inequalities in *Opt-VI*.

The last two columns of Table 4.2 give information about the runtime of each model averaged over the instances. In the final column, we provide the number of instances that were unable to be solved by the model in the 20 minute time limit. When computing the average run time for each model these instances contributed 1200 seconds, so we note that these average times may be skewed lower than they would be with no time limit. We see

that *Opt-IW* improves greatly upon *Opt-MC*. Additionally, while the first two columns of this table indicate that *Opt-VI* finds superior solutions in the time limit and begins with a lower LP relaxation bound, *Opt-VI* performs similarly to *Opt-IW* in run time. Finally, we see that *Approx* solves faster than the exact methods, reaching the time limit on fewer instances. As *Approx* finds near optimal solutions, this model appears to be a good choice for generating high-quality feasible solutions to this problem.

In Tables 4.3 and 4.4 we report the run times and gaps split up by each instance type. Each value in the table is averaged over the 5 instances for each instance type. Table 4.3 reports the run times in seconds for each method on each instance type. Reported in parentheses next to the run times are how many of the 5 instances reach the time limit. Any instance that reaches the 20 minute time limit contributes 1200 seconds to the average. Here we see that *Opt-IW* solves the fastest in a few cases, in particular when the maximum arc delay per job is high or when there is a small number of attackers. Conversely, we find a few cases where *Approx* is particularly useful, such as when there are a large number of attackers. *Approx* tends to stay more consistent in run-times across instances and is less prone to long run times with larger problems than the other methods.

Table 4.3: Average run times with count of instances exceeding limit in parentheses.

Instances	***Opt-MC***	***Opt-IW***	***Opt-VI***	***Approx***
Base	553.0 (2)	506.9 (2)	516.1 (1)	320.7 (1)
max delay $= \frac{1}{6}d_{max_a}$	663.4 (2)	398.6 (1)	323.9 (1)	322.3 (1)
max delay $= \frac{2}{3}d_{max_a}$	531.1 (1)	344.0 (1)	755.7 (2)	673.4 (2)
$\|A\| = 5$	662.3 (2)	168.7	315.4 (1)	256.6
$\|A\| = 20$	1206.6 (5)	1060.2 (4)	838.2 (3)	265.0
$d_{ij} \in [2, 6]$	395.8 (1)	117.6	70.0	9.9
$d_{ij} \in [8, 18]$	1001.8 (4)	512.9 (1)	461.2 (1)	393.2 (1)
6×5	276.4 (1)	134.9	87.1	104.2
4×8	687.6 (1)	122.5	252.5	253.2

Table 4.4 reports the LP relaxation bound gaps averaged over the five instances for each of the nine instance types. Here we see that *Opt-VI* gives a greater decrease in LP bound gap on instances with longer attacker time horizons such as with a longer attack graph or with longer arc durations. This is what we expect as there may be more arcs after the defender's time horizon, which are the arcs the valid inequalities affect.

Table 4.4: Average LP relaxation bound gaps for exact solution methods over 5 instances for each instance type.

Instances	***Opt-MC***	***Opt-IW***	***Opt-VI***
Base	45.1%	45.1%	25.5%
max delay $= \frac{1}{6} d_{max_a}$	45.2%	45.2%	25.2%
max delay $= \frac{2}{3} d_{max_a}$	49.1%	49.1%	29.6%
$\|A\| = 5$	50.0%	50.0%	34.2%
$\|A\| = 20$	43.5%	43.5%	29.0%
$d_{ij} \in [2, 6]$	41.3%	41.3%	38.0%
$d_{ij} \in [8, 18]$	53.0%	53.0%	33.8%
6×5	27.0%	27.0%	19.4%
4×8	60.0%	60.0%	29.0%

4.7 Conclusion

This paper develops methods to model a defender delaying the shortest paths of various attackers, where both the defender and attacker are taking their actions simultaneously. We use a pessimistic approach, assuming the attacker knows the defender's schedule before beginning to traverse their arcs, in order to format this as a bilevel problem. This still becomes a difficult problem to model. We develop IP formulations, and propose valid inequalities to improve the LP relaxation bounds of these models. We also develop an approximation heuristic that finds near optimal solutions. The performance of the approximation heuristic indicates that the defender need not know exactly the attacker arc durations, and scheduling mitigations based on an approximation of attacker timing is still useful. However, each of the models we propose still can be very difficult to solve in a reasonable amount of time. We use relatively small instances in our computational studies, and increasing the instance sizes would make each of these methods unreasonable. For this reason, a future direction for this research would include developing alternate algorithms or heuristics to solve this problem. In particular, modifying the heuristic methods in Chapter 3 could provide faster heuristics for this problem. In addition, an adaptation of the sequential model of Chapter 3 could give a relaxation that could provide improved bounds.

5 CONCLUSION

5.1 Comparison of Models Across Chapters

In this dissertation, we present three different ways to model threats to a system of critical infrastructure and the deployment of mitigations to protect against these threats. In Chapter 4, we assume attackers are searching for a shortest path along an attack graph, while in Chapter 3 we assume attackers must complete all actions in a project network, finally in Chapter 2 we do not consider attackers at all, but rather multiple coverage of vulnerabilities. These each model different situations, but it is worth asking how the solutions of these different models compare. This gives some insight on the importance of accurately modeling the system in consideration. We analyze the solutions for defenders using the models developed in each of Chapters 2 - 4—models (2.1),(3.5),(4.10)—on a small example with synthetic data.

Example Data

We use the same defender for all three chapter's models and a set of two attackers for use in the models from Chapters 3 and 4. We use the attacker data to create vulnerabilities for the model of Chapter 2. We can then see the solution of this model as the solution obtained when using the vulnerability coverage model in Chapter 2 to approximate a network interdiction problem such as those in Chapters 3 and 4.

The defender has 17 jobs available to schedule, $J = \{2, ..., 18\}$. An artificial job 1 is used to precede jobs with no predecessors, as displayed in Figure 5.1 that shows the precedence network for the defender. The defender has a budget of $B = 8$, where each job costs 1, that is, the defender can schedule a total of 8 jobs. The defender has a time horizon of $T = 25$ periods, and four resources $R = \{r_1, r_2, r_3, r_4\}$ to use. Table 5.1 displays the time and resources required for each job for the defender. The attackers' networks are provided in Figure 5.2, where attacker 1 uses nodes of the form 1xx and attacker 2 has nodes of the form 2xx. Both attacker's start and end at nodes 0 and 1, respectively. We choose two attacker's with different network structures, since we use these same attack networks for both the longest and shortest path problems of Chapters 3 and 4.

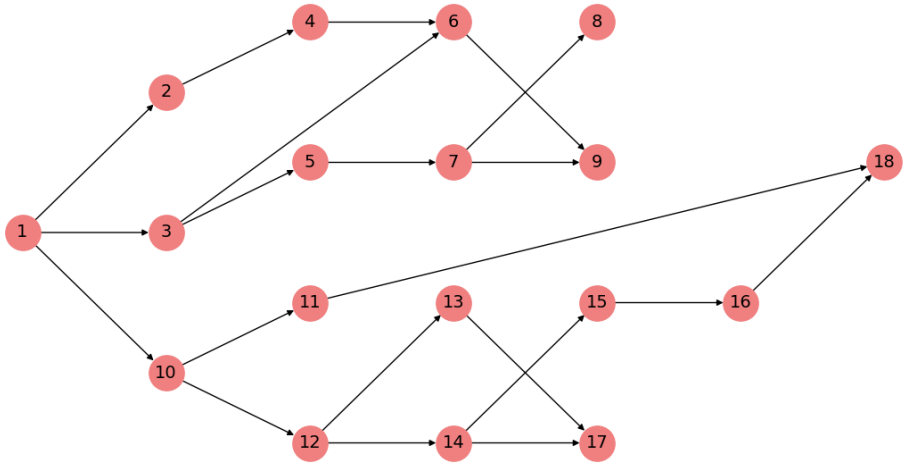


Figure 5.1: Defender's precedence network for example problem

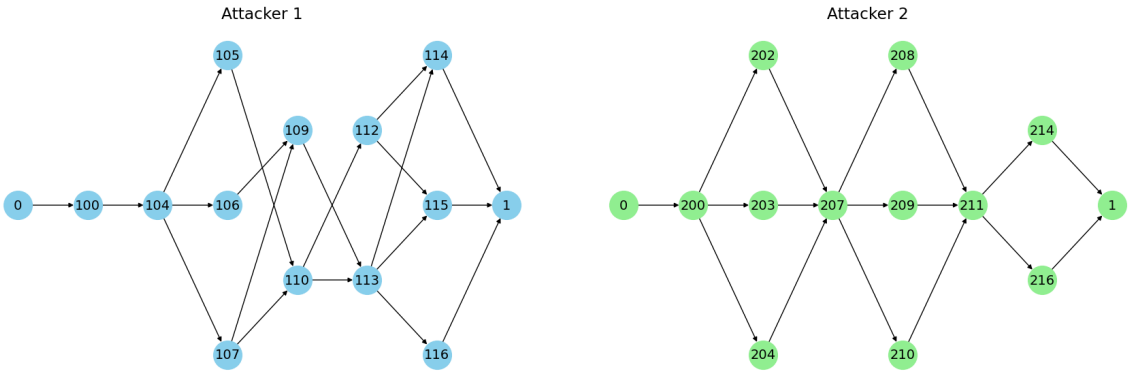


Figure 5.2: Attackers' networks for example problem

Tables 5.2 and 5.3 display the amount of delay each job provides each edge, for attackers 1 and 2, respectively. Additionally, the first two columns of these tables provide the initial duration d_{ij} and maximum delay $\bar{\delta}_{ij}$ for each arc (i, j). Only jobs that provide delay for any arcs are included in this table. We simplify the attacker data from these networks to provide data for the model of Chapter 2. To do so, we let the set of vulnerabilities be the interdictable edges. Then we define piecewise linear functions $f_{(i,j)}(z_{(i,j),t}) = \min(z_{(i,j),t}, \bar{\delta}_{ij})$ for $(i, j) \in \mathcal{A}$. We set the time weighting for this model with an exponential base of 0.95.

Table 5.1: Job durations τ_m and per period resource demands for each job $m \in J$, as well as per period resource budget.

$m \in J$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	b_r
τ_j	4	3	3	5	2	4	4	5	4	5	3	6	4	2	1	5	4	
r_1	10	10	0	0	0	6	0	8	2	3	0	5	0	0	8	7	10	15
r_2	6	0	6	4	2	10	0	0	0	10	0	3	3	0	3	0	0	12
r_3	0	0	0	2	0	8	0	0	0	0	3	0	4	4	1	0	0	9
r_4	0	8	0	9	0	0	0	0	0	0	3	0	0	9	0	0	9	15

Table 5.2: Arc durations d_{ij}, maximum delay $\bar{\delta}_{ij}$, and delays given by each job $m \in J$ for attacker 1.

			$m \in J$												
$(i,j) \in \mathcal{A}_1$	d_{ij}	$\bar{\delta}_{ij}$	4	5	6	7	8	9	10	13	14	15	16	17	18
(0, 100)	1														
(100, 104)	9														
(104, 105)	9														
(104, 106)	5														
(104, 107)	4	4	4												
(105, 110)	10	4												4	
(107, 110)	5	4	3						3						
(106, 109)	6	6				6									
(107, 109)	4	9		3					3			4			
(110, 112)	6	6			6										6
(109, 113)	7	14		6				6				6			
(110, 113)	6	4								4					
(112, 114)	8														
(113, 114)	8	4	4												
(112, 115)	7	9											6	4	
(113, 115)	10	2	2												
(113, 116)	4														

Example Solutions

We solve the models from each chapter for this example instance to get solutions for the defender. We denote the model from Chapter 2 as *Cover*, the model from Chapter 3 as *LPP*, and the model from Chapter 4 as *SPP*. Table 5.4 gives the time period in which jobs are completed for each model. We see that each model provides a different solution for the defender that best protects the system given the model of threats. Some jobs (2,3,4,5,10) are scheduled by all defenders. These may be jobs that precede most other jobs and thus are useful to schedule in any case, jobs that are more impactful and generally worth doing regardless of the model, or jobs that are easy to complete and thus easy to include in any schedule.

We analyze how robust the solutions are to model choice by evaluating the objective

Table 5.3: Arc durations d_{ij}, maximum delay $\bar{\delta}_{ij}$, and delays given by each job $m \in J$ for attacker 2.

			$m \in J$												
$(i,j) \in \mathcal{A}_2$	d_{ij}	$\bar{\delta}_{ij}$	4	5	6	7	8	9	10	13	14	15	16	17	18
(0, 200)	1														
(200, 202)	7														
(200, 203)	6														
(200, 204)	6														
(202, 207)	6	4									4				
(203, 207)	6	7							4					5	
(204, 207)	5	4	4												
(207, 208)	6														
(207, 209)	7	9		3					3			4			
(207, 210)	10	4	3						3						
(208, 211)	7														
(209, 211)	8	5					4								4
(210, 211)	4	6				6									
(211, 214)	5														
(211, 216)	8	5		3				4							

value and attacker solution for each model given each defender solution presented in Table 5.4. We first provide a table of objective values, Table 5.5. Each column reports which model objective values were calculated for, and each row determines which defender solution was used to calculate the objective. The diagonal of the table shows the optimal objective values for each model since those entries use the defender solution for the matching attacker model. We see the solution for the *SPP* model tends to be more tuned to the *SPP* model in particular. We note that the *SPP* defender solution schedules fewer jobs than the other models' solutions. The shortest path attackers can be more difficult to defend against since the attackers traverse the graph more quickly. A job useful against a longest path attacker completing a project, may seem less useful to a defender considering shortest path attackers, as the attacker is more likely to have traversed the arcs that job delays. Thus a defender against shortest path attackers may choose less impactful jobs that can be done quicker, resulting in poor performance in models that allow more time. In addition, a shortest path attacker only uses arcs on a single path, so that delaying an arc that already has a long length may be less useful as the attacker may never use that arc. This is contrasting from the *LPP* defender, as an arc that is unlikely to be on the shortest path is more likely to be on the longest path, and thus potentially more useful to delay. Finally, we see that the *Cover* defender performs well on each of the other models. This coverage model may be a good general purpose model that can be used when specific attacker information is less clear.

Table 5.4: Periods in which jobs are completed in the solutions for each model.

Job	*Cover*	*LPP*	*SPP*
2	7	7	4
3	3	3	7
4	10	10	7
5	8	8	12
6	12		
7	14		
8			
9	19		
10	4	4	4
11			
12		7	7
13			13
14		11	
15		13	
16			
17			
18			

Table 5.5: Objective values for each model, using each of the defender solutions presented in Table 5.4.

Solution	**Objective Model**		
Model	*Cover*	*LPP*	*SPP*
Cover	23.9	57.5	34.5
LPP	22.2	58	34.5
SPP	18.6	52	36

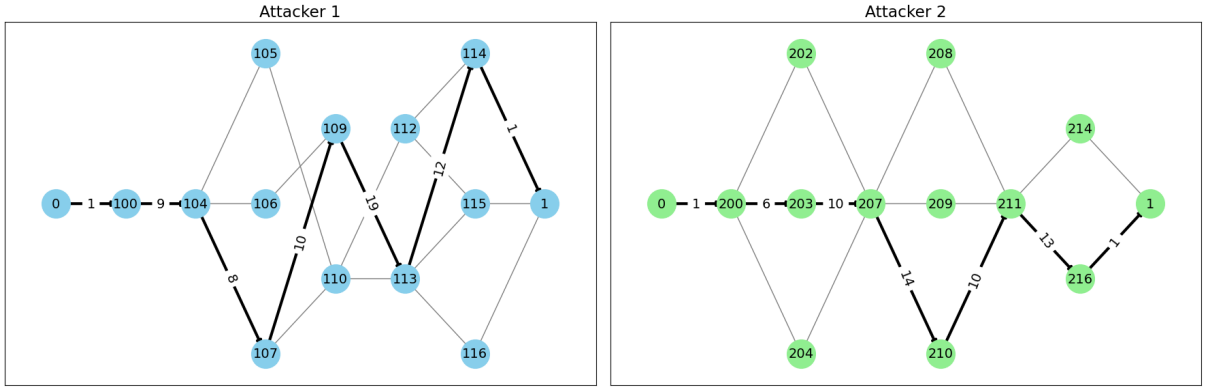
We lastly compare the attackers' solutions given each defender solution, for both longest and shortest path attackers. In Figure 5.3 we show the longest paths in each attacker's graph, with the lengths of those edges shown as determined by each defender solution. Looking at attacker 1, we see that there was a specific path that was the longest in all 3 cases, however, the *LPP* defender did a better job at delaying arcs along this path. Arcs $(107, 109)$ and $(109, 113)$ were mitigated less by the other defenders, particularly *SPP*, resulting in a shorter longest path. Looking at attacker 2, we see that when delaying longest paths, interdicting arcs not originally on the longest path can still be useful, as is seen with the

Cover defender. Here, the *Cover* defender achieves the same length between nodes 207 and 211 as the *LPP* defender, by delaying arcs going through node 210 instead of 209. On the other hand, as previously mentioned, the *SPP* defender prioritizes arcs on the shortest attacker path, and thus does not significantly impact the attacker's longest path.

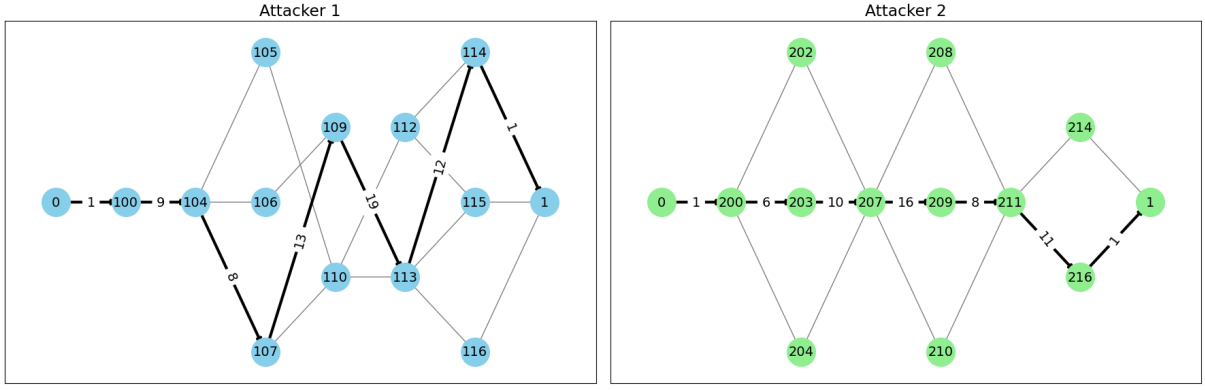
In Figure 5.4 we switch to attackers using the shortest path in the network. For both attackers we see that the shortest path against both *LPP* and *Cover* defenders differs from the shortest path against the *SPP* defender. Since the *SPP* defender is trying to lengthen the shortest path, it successfully switches the attackers' paths to new longer paths than those used by the attackers using the *LPP* and *Cover* solutions. While longest path attackers can still be affected by delaying edges not on the original longest path, even if none of the original longest path edges are delayed, this is not the case with shortest path attackers. If sufficient delays to the initial shortest path are not imposed by the defender, any other delays they make to that attacker have no impact.

Since the *Cover* model has no attackers, we do not provide figures for these solutions. However, we can look back on Table 5.5 and our other analysis to consider how the *LPP* and *SPP* defender solutions would do in this model. Here, timing prioritization between jobs becomes less intricate, and gaining the most coverage as quickly as possible is what is important. Both the *LPP* and *SPP* defenders desire to gain "coverage" in a timely matter, so they find decent solutions to this problem. However, by specifically tailoring their mitigation preferences to specific paths of attacker networks they sacrifice some overall coverage in the *Cover* model. This is seen more prominently for the *SPP* defender, as can be seen by the objective values. This makes sense, since, as we have noted, the *SPP* defender has more specific priorities in the arcs they delay.

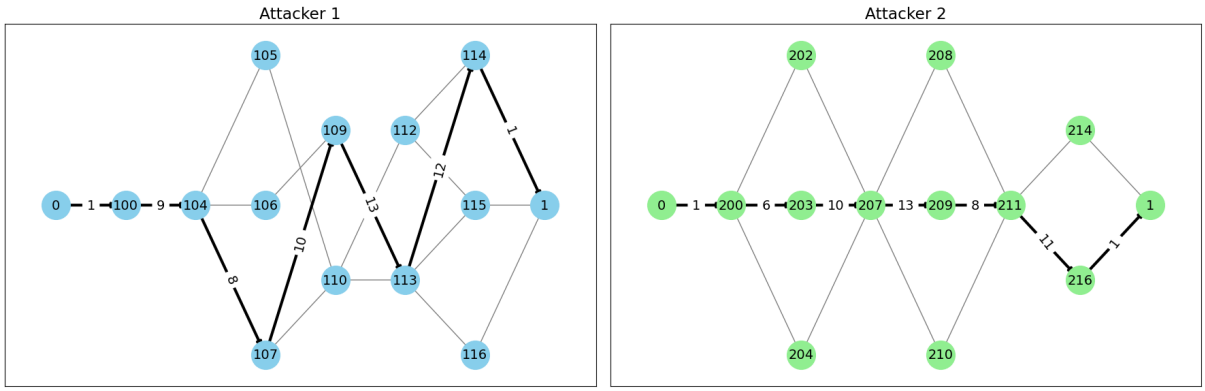
Overall, we see the importance of choosing an appropriate model for the defender. Using the more general model from Chapter 2 can be most useful in a generic situation with a variety of threats to the system. However, identifying sophisticated attacker actions can allow a defender to more specifically tailor their schedule of mitigations to best defend the system. We see that defending against attackers completing projects and attackers completing shortest paths in an attack graph require delaying different arcs and thus require different defender schedules. Nonetheless, some mitigations can be useful to start on right away if they are easy to implement and/or required in order to complete most more advanced mitigations.



(a) *LPP* attacker solution against *Cover* defender

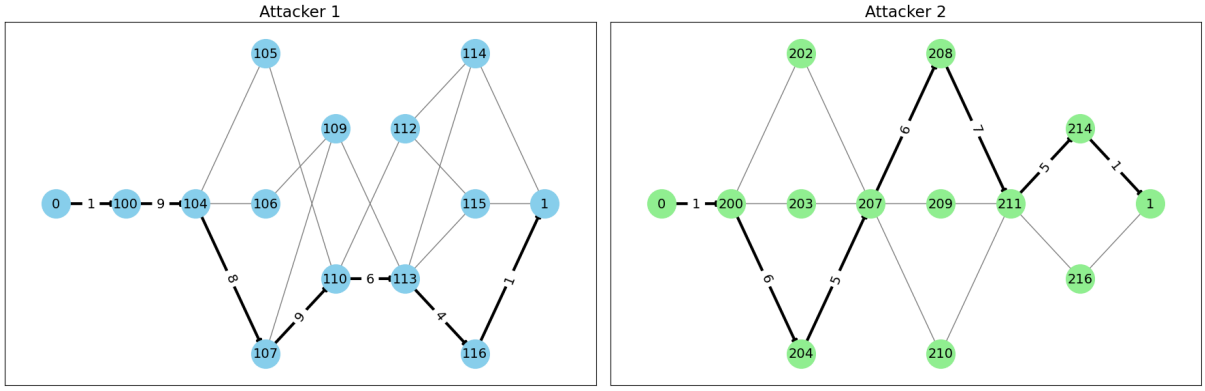


(b) *LPP* attacker solution against *LPP* defender

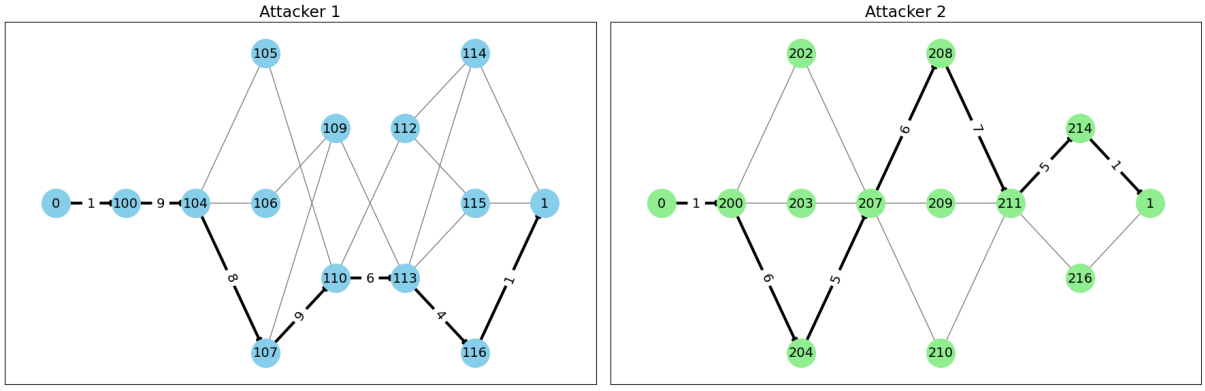


(c) *LPP* attacker solution against *SPP* defender

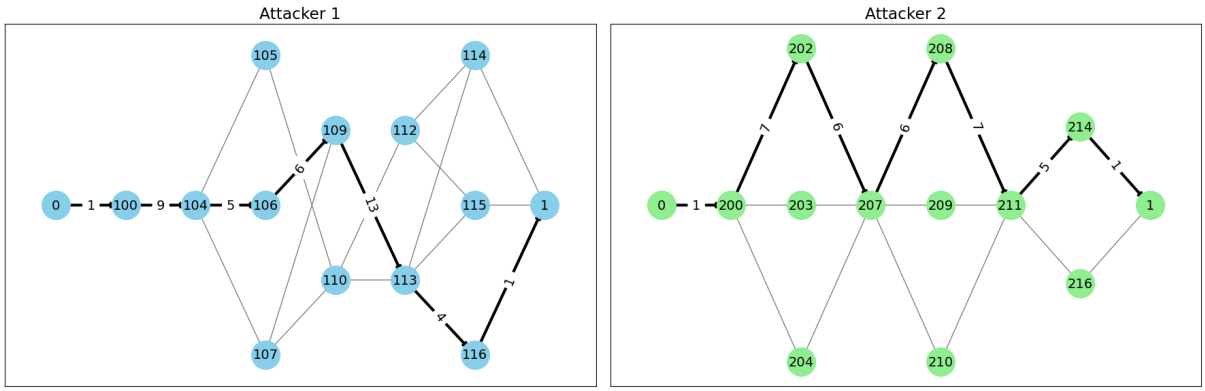
Figure 5.3: Attacker longest path solutions given each defender solution.



(a) *SPP* attacker solution against *Cover* defender



(b) *SPP* attacker solution against *LPP* defender



(c) *SPP* attacker solution against *SPP* defender

Figure 5.4: Attacker shortest path solutions given each defender solution.

5.2 Future Directions

In this dissertation we have advanced our understanding of how to deploy mitigations for critical infrastructure systems in resource constrained settings over time. In Chapters

2 and 3 we consider adaptations of preexisting methods to this problem as a metric of how well a system could be protected using existing models when addressing the fact that implementing mitigations takes time and resources. We find that the models we present in this dissertation improve upon solutions found using existing methods. We thus conclude that there is generally a benefit to considering the time and resources required to implement mitigations, in the models used to decide which mitigations to implement to best protect the system.

While the models we propose gain the benefit of combining selection and scheduling, they do have the downside of often being more difficult to solve than existing models that only select mitigations. In this work we thus propose a few heuristic methods to help find good solutions to our models more quickly. However, this is an area that could be studied further. In particular, due to working with both an attacker and defender network the models of Chapters 3 and 4 are particularly difficult to solve. Developing alternate methods to solve these problems or find heuristic solutions would allow us to gain the benefit of considering both selecting and scheduling for larger instances where the models we present may be too large to solve. Second, literature considering the deployment of mitigations for critical infrastructure over time is sparse. While we incorporate this into three different ways of modeling threats to a system, other models exist that could potentially benefit from this consideration. The models and heuristics provided in this dissertation can provide a beginning framework for other problems in this area.

Finally, the models presented in this dissertation are all deterministic models and assume the defender has full knowledge of the vulnerabilities to their system. In reality this may not be well known, particularly in working with sophisticated attackers. In Chapters 3 and 4 we allow the defender to defend against multiple different attackers. By using a large number of attackers, these can be treated like the scenarios of a stochastic programming problem. In this way, we give some consideration to uncertainty in the parameters of the problems we study. However, we see that using a large number of attackers can greatly increase the size of the problem and thus the models presented in Chapters 3 and 4 may not be as reasonable to solve. Additionally, Chapter 2 gives less consideration to uncertainty in the model, and we give no consideration to uncertainty in the defender's parameters. Thus, integrating uncertainty into these models, and developing methods to these more complex problems is an avenue for future research.

A APPENDIX

A.1 Existing Methods

The model of Chapter 2, (2.1), chooses mitigations to implement based on a multiple coverage setting while simultaneously scheduling them, which allows us to optimally choose a set of mitigations given the constraints and time effects of scheduling. To investigate the potential benefit of incorporating these two aspects of mitigation planning, we consider how solutions to this integrated model could be obtained by solving models that consider these aspects separately. To do so we either simplify mitigation effects to approximate coverage or separate these two steps. We consider first a method where we approximately model the problem as an RCPSP. Then, we consider a second method where we select jobs while considering multiple coverage in a first phase, and schedule the chosen jobs in a second phase.

RCPSP with Approximated Objective Function

One heuristic approach to solve the problem is to formulate it as an RCPSP that ignores the vulnerability node coverage aspects and instead approximates them with an objective function that depends only on the completion time of jobs.

The key question with this approach is how to assign objective function weights for each job, given that each job may cover multiple nodes and nodes may vary in importance. We determine the objective function weight of a mitigation as a weighted sum of coverage for all the nodes the mitigation covers, weighting by node importance. This yields the objective:

$$\max \sum_{j \in J} \sum_{t \in \mathcal{T}} a_t h_j x_{jt}, \tag{A.1}$$

where a_t is the same objective function weight as given in (2.3) and $h_j = \sum_{n \in \mathcal{N}} v_n w_{jn}$ gives the reward for completing job j, with v_n giving the chosen weight for node n. We choose to set v_n to be the slope of f_n at $z_{n0} = 0$. Note that we use a_t instead of α_t since the variable x_{jt} equals to 1 in at most one period, indicating the period in which job j completes, so it is appropriate to use the a_t coefficients. The remainder of the formulation matches our main model with anything involving node coverage removed from the model. That is, we consider an integer program with objective function (A.1) subject to constraints (2.1b) -(2.1f) .

Select-then-Schedule

The next heuristic we consider is a two-phase method in which, as a first phase, we select jobs while considering multiple coverage of vulnerabilities by making use of the model from Zheng et al. (2019). Then in the second phase, we solve a scheduling problem that attempts to schedule the selected jobs while considering the resource limitations. We refer to this heuristic as 'select-then-schedule.' This method can be viewed as an extension of the first method by adding a preceding job selection step. The motivation for this heuristic is that by scheduling only a subset of jobs that provide good coverage, the RCPSP problem should be easier to solve and will ideally provide better solutions given the multiple coverage objective.

For the first phase, we use a simplified version of the budgeted maximal multiple coverage model from Zheng et al. (2019) that only has one vulnerability node per attack path. The overall budget constraint (2.1c) fits the budget constraint of this model, and we can utilize the piecewise linear functions f_n for $n \in \mathcal{N}$ for the objective.

Since in the first phase we do not consider how to schedule the jobs, we can then add in relaxations of our time-based constraints to promote feasibility. First we add a budget constraint for every resource. To avoid choosing too many jobs that use the same resource, we relax the time-based resource budget constraints by summing over time periods. Each job $j \in J$ uses $\tau_j c_{jr}$ units of resource r, and we have a total budget of $B_r = \sum_{t=1}^{T} b_{rt}$ units of resource $r \in R$. Second, we add a non-time-based precedence constraint, that is, for $(i, j) \in P$ we add a constraint that ensures if we choose job j, we also choose job i.

We maintain variable notation for the select-then-schedule model to represent job completion and node coverage, however, these variables differ from those in the full model (2.1) since we are disregarding time in this phase. Let z_n give the amount of coverage for node $n \in \mathcal{N}$, and let x_j be a binary variable that equals 1 if we choose job $j \in J$ and 0 otherwise.

Then the formulation for the first phase is as follows:

$$\max \sum_{n \in \mathcal{N}} f_n(z_n) \tag{A.2a}$$

$$\text{s.t.} \sum_{j \in J} (\tau_j c_{jr}) x_j \leqslant B_r \qquad \forall r \in R \tag{A.2b}$$

$$\sum_{j \in J} C_m x_j \leqslant B \tag{A.2c}$$

$$x_j \leqslant x_i \qquad \forall (i,j) \in P \tag{A.2d}$$

$$z_n \leqslant \sum_{n \in \mathcal{N}} w_{jn} x_j \qquad \forall j \in J \tag{A.2e}$$

$$z_n \geqslant 0 \qquad \forall n \in \mathcal{N} \tag{A.2f}$$

$$x_j \in \{0,1\} \qquad \forall j \in J. \tag{A.2g}$$

After solving problem (A.2), assume we obtain an optimal solution $\bar{x}$. We then take the jobs chosen in this solution, that is $\{j \in J : \bar{x}_j = 1\}$, and use these as the set of jobs for the RCPSP model presented in Section A.1. We can then solve the RCPSP model to find a schedule.

A.2 Data Generation

We generate a dataset for this problem by following the methods used by Kolisch and Sprecher (1997) to create a base RCPSP structure, and modify this structure to fit the design of this model. Matching the format of the project scheduling literature, we utilize multiple 'projects', which we interpret as separate sets of jobs that may not relate to each other. Each project has various precedences between its corresponding jobs, where each project has some number of "start" jobs that have no predecessors and "finish" jobs that have no successors. While not all jobs in a project provide coverage, we ensure that each of the finish jobs cover at least one node, since they would otherwise have no incentive to be completed. We follow the methods presented by Kolisch and Sprecher (1997) in creating the set of projects and jobs as well as the precedences between jobs, with the one exception of disregarding the case of creating redundant arcs when adding a new precedence. This

may simply cause overestimation of complexity of the precedence network.

Next we generate the resource data. Kolisch and Sprecher (1997) define three types of resources: renewable, nonrenewable, and doubly constrained (both). We take guidance from their renewable resources for our resource constraints. We use the same procedure to generate job resource demands, using a slightly modified parameter for per period resource availability. We find the minimum per period availability for each resource $r \in R$ as $\max_{j \in J} c_{jr}$, and then use a parameter RS to give the per period resource budget of $RS \max_{j \in J} c_{jr}$. Scaling up this parameter allows jobs to be scheduled more concurrently and makes the precedence relations, rather than resource capacity, a more important factor in when jobs can be completed. Since we only consider one nonrenewable resource, budget, we use a simplified resource generation process in this case. We assign it as a resource used by every job, and assign costs per job as with the renewable resources, then we define an overall budget as some percentage Bud of total cost.

We next generate data for the coverage of nodes. Since we assume that not every job covers nodes, we create a subset of jobs, call them coverage jobs, which each cover at least one vulnerability node. Note that all finish jobs are coverage jobs. We ensure that every non-coverage job precedes at least one coverage job, so there are no illogical jobs in this sense. In addition to the finish jobs, we assume other jobs may gain intermediate coverage, and thus classify these as coverage jobs as well. To determine which non-finish jobs are coverage jobs, we randomly assign jobs to be coverage jobs with some probability ML so that we obtain an expected proportion of ML non-finish jobs as coverage jobs.

Once we have our set of coverage jobs, we employ the same technique as in the resource demand generation to determine which nodes each coverage job covers. We use a *coverage factor* CF, and randomly assign nodes to each coverage job within some specified range for the amount of nodes a coverage job can cover.

To determine the amount of coverage each coverage job achieves, we take into consideration the amount of work necessary to complete the job. We do so by considering the jobs that precede the coverage job. We provide a parameter SF that defines how much importance is given to this. A possible range for the amount of coverage for a job is specified, and the point in the range a coverage job's coverage for a node lies is proportional to $(1-SF)$ times a random $U[0,1]$ value plus SF times a proportion of the total amount of resources needed to complete the job (including precedences) out of the largest such amount of all

jobs. The final amount of coverage the job provides the node is then given by adjusting this number based on CF and the number of nodes the job covers.

Lastly, we describe how we generate the objective functions f_n for each node $n \in \mathcal{N}$. Recall these functions are concave and piecewise linear. We assume that in general, each time a node is covered, the amount of coverage gained by a subsequent job is decreased. Thus, we define the breakpoints of our piecewise linear functions at the expected amount of coverage a job gives each node. We then determine how many times we would expect to cover each node if we covered them all equally, by computing $\mathsf{CF}(\mathsf{Bud})|\{j \in J : \max_{n \in \mathcal{N}}(w_{jn}) > 0\}|$. Taking a number slightly larger than this, by multiplying by 1.3 and rounding to the nearest integer, we then use this as our average number of breakpoints for each f_n, randomly varying between one more and one less than this value for each node. In this way we avoid trivial solutions where all nodes can easily be covered up to the last interval of their corresponding function. We then generate the slopes between each breakpoint based on a parameter ES. Our first slope is determined randomly, and the last slope is given by multiplying the first slope by ES. We then randomly decrease the slope at each breakpoint, such that on average we expect to reach the desired final slope at the final breakpoint. If we happen to surpass the desired final slope early, we set that and all subsequent slopes equal to the final slope.

REFERENCES

Albert, Laura A. 2023. A cybersecurity planning framework for evaluating, selecting, and deploying security controls. In *Proceedings of the Institute for Industrial and Systems Engineers Annual meeting*. New Orleans, LA.

Artigues, Christian, Peter Brucker, Sigrid Knust, Oumar Koné, Pierre Lopez, and Marcel Mongeau. 2013. A note on "event-based MILP models for resource-constrained project scheduling problems". *Computers & Operations Research* 40(4):1060–1063.

Berthold, Timo, Stefan Heinz, Marco E Lübbecke, Rolf H Möhring, and Jens Schulz. 2010. A constraint integer programming approach for resource-constrained project scheduling. In *Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings 7*, 313–317. Springer.

Borrero, Juan S, Oleg A Prokopyev, and Denis Sauré. 2016. Sequential shortest path interdiction with incomplete information. *Decision Analysis* 13(1):68–98.

Brown, Gerald G, W Matthew Carlyle, Robert C Harney, Eric M Skroch, and R Kevin Wood. 2009. Interdicting a nuclear-weapons project. *Operations Research* 57(4):866–877.

Brown, Gerald G, W Matthew Carlyle, Johannes O Royset, and R Kevin Wood. 2005. On the complexity of delaying an adversary's project. In *The Next Wave in Computing, Optimization, and Decision Technologies*, 3–17. Springer.

Carrasco, Rodrigo A, Diego Fuentes, and Eduardo Moreno. 2022. Approximation algorithm for resource-constrained project scheduling problems with net present value objective. *arXiv preprint arXiv:2209.02029*.

Contardo, Claudio, and Jorge A Sefair. 2022. A progressive approximation approach for the exact solution of sparse large-scale binary interdiction games. *INFORMS Journal on Computing* 34(2):890–908.

Dimitrov, Nedialko B, and David P Morton. 2012. Interdiction models and applications. In *Handbook of Operations Research for Homeland Security*, 73–103. Springer.

DuBois, Eric, Ashley Peper, and Laura A Albert. 2023. Interdicting attack plans with boundedly rational players and multiple attackers: An adversarial risk analysis approach. *Decision Analysis* 20(3):202–219.

Enayaty-Ahangar, Forough, Laura A Albert, and Eric DuBois. 2020. A survey of optimization models and methods for cyberinfrastructure security. *IISE Transactions* 53(2): 182–198.

Fila, Barbara, and Wojciech Wideł. 2020. Exploiting attack–defense trees to find an optimal set of countermeasures. In *2020 IEEE 33rd computer security foundations symposium (CSF)*, 395–410. IEEE.

Gutin, Eli, Daniel Kuhn, and Wolfram Wiesemann. 2015. Interdiction games on markovian pert networks. *Management Science* 61(5):999–1017.

Hall, Leslie A, Andreas S Schulz, David B Shmoys, and Joel Wein. 1997. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of operations research* 22(3):513–544.

Hartmann, Sönke, and Dirk Briskorn. 2022. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* 297(1):1–14.

Hill, Alessandro, Andrea J Brickey, Italo Cipriano, Marcos Goycoolea, and Alexandra Newman. 2022. Optimization strategies for resource-constrained project scheduling problems in underground mining. *INFORMS Journal on Computing* 34(6):3042–3058.

Holzmann, Tim, and J Cole Smith. 2021. The shortest path interdiction problem with randomized interdiction strategies: Complexity and algorithms. *Operations Research* 69(1): 82–99.

Israeli, Eitan, and R Kevin Wood. 2002. Shortest-path network interdiction. *Networks: An International Journal* 40(2):97–111.

Khouzani, MHR, Zhengliang Liu, and Pasquale Malacaria. 2019. Scalable min-max multi-objective cyber-security optimisation over probabilistic attack graphs. *European Journal of Operational Research* 278(3):894–903.

Khuller, Samir, Anna Moss, and Joseph Seffi Naor. 1999. The budgeted maximum coverage problem. *Information processing letters* 70(1):39–45.

Kolisch, Rainer, and Sönke Hartmann. 1999. *Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis*. Springer.

———. 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research* 174(1):23–37.

Kolisch, Rainer, and Arno Sprecher. 1997. PSPLIB - a project scheduling problem library: Or software - orsep operations research software exchange program. *European Journal of Operational Research* 96(1):205–216.

Kordy, Barbara, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. 2014. Dag-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review* 13-14:1–38.

Liess, Olivier, and Philippe Michelon. 2008. A constraint programming approach for the resource-constrained project scheduling problem. *Annals of Operations Research* 157(1): 25–36.

Lunday, Brian J, and Hanif D Sherali. 2010. A dynamic network interdiction problem. *Informatica* 21(4):553–574.

Malaviya, Ajay, Chase Rainwater, and Thomas Sharkey. 2012. Multi-period network interdiction problems with applications to city-level drug enforcement. *IIE Transactions* 44(5):368–380.

Mingozzi, Aristide, Vittorio Maniezzo, Salvatore Ricciardelli, and Lucio Bianco. 1998. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management science* 44(5):714–729.

Nagurney, Anna, and Shivani Shukla. 2017. Multifirm models of cybersecurity investment competition vs. cooperation and network vulnerability. *European Journal of Operational Research* 260(2):588–600.

Nguyen, Di H, Yongjia Song, and J Cole Smith. 2023. A two-stage network interdiction-monitoring game. *Networks* 81(3):334–358.

Peper, Ashley. 2024. Mitigation Scheduling Data. `https://github.com/apeper2/MitigationSchedulingData`.

Pritsker, A Alan B, Lawrence J Waiters, and Philip M Wolfe. 1969. Multiproject scheduling with limited resources: A zero-one programming approach. *Management science* 16(1): 93–108.

Schilling, Andreas, and Brigitte Werners. 2016. Optimal selection of it security safeguards from an existing knowledge base. *European Journal of Operational Research* 248(1):318–327.

Schmidt, Adam, Laura A Albert, and Kaiyue Zheng. 2021. Risk management for cyber-infrastructure protection: A bi-objective integer programming approach. *Reliability Engineering & System Safety* 205:107093.

Schutt, Andreas, Geoffrey Chu, Peter J Stuckey, and Mark G Wallace. 2012. Maximising the net present value for resource-constrained project scheduling. In *International conference on integration of artificial intelligence (AI) and Operations research (OR) techniques in constraint programming*, 362–378. Springer.

Sefair, Jorge A, and J Cole Smith. 2016. Dynamic shortest-path interdiction. *Networks* 68(4):315–330.

Smith, J Cole, Mike Prince, and Joseph Geunes. 2013. Modern network interdiction problems and algorithms. In *Handbook of combinatorial optimization*, 1949–1987. Springer.

Song, Yongjia, and Siqian Shen. 2016. Risk-averse shortest path interdiction. *INFORMS Journal on Computing* 28(3):527–539.

Talbot, F Brian, and James H Patterson. 1978. An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. *Management Science* 24(11):1163–1174.

Tanınmış, Kübra, and Markus Sinnl. 2022. A branch-and-cut algorithm for submodular interdiction games. *INFORMS Journal on Computing* 34(5):2634–2657.

Vanhoucke, Mario, Erik Demeulemeester, and Willy Herroelen. 2001. On maximizing the net present value of a project under renewable resource constraints. *Management Science* 1113–1121.

Vorobeychik, Yevgeniy, and Joshua Letchford. 2015. Securing interdependent assets. *Autonomous Agents and Multi-Agent Systems* 29(2):305–333.

Wei, Ningji, and Jose L Walteros. 2022. Integer programming methods for solving binary interdiction games. *European Journal of Operational Research* 302(2):456–469.

———. 2024. On supervalid inequalities for binary interdiction games. *Mathematical Programming* 1–42.

Weninger, Noah, and Ricardo Fukasawa. 2025. A fast combinatorial algorithm for the bilevel knapsack problem with interdiction constraints. *Mathematical Programming* 210(1): 847–879.

Yang, Kum Khiong, F.Brian Talbot, and James H. Patterson. 1993. Scheduling a project to maximize its net present value: An integer programming approach. *European Journal of Operational Research* 64(2):188–198.

Zheng, Jiefu, and David A Castañón. 2012. Dynamic network interdiction games with imperfect information and deception. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, 7758–7763. IEEE.

Zheng, Kaiyue, and Laura A Albert. 2019. Interdiction models for delaying adversarial attacks against critical information technology infrastructure. *Naval Research Logistics (NRL)* 66(5):411–429.

Zheng, Kaiyue, Laura A Albert, James R Luedtke, and Eli Towle. 2019. A budgeted maximum multiple coverage model for cybersecurity planning and management. *IISE Transactions* 51(12):1303–1317.

Zhuo, Yueran, and Senay Solak. 2014. Measuring and optimizing cybersecurity investments: A quantitative portfolio approach. In *IIE Annual Conference. Proceedings*, 1620.