Addressing Challenges in Healthcare Provider Scheduling

by

Brian J. Lemay

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Industrial and Operations Engineering)
in the University of Michigan
2017

Doctoral Committee:

Associate Professor Amy E.M. Cohn, Chair
Associate Professor Marina A. Epelman
Professor Anne E. Sales
Professor Pascal Van Hentenryck

© Brian J. Lemay 2017

blemay@umich.edu

ORCID ID: 0000-0002-8705-564X

All Rights Reserved

DEDICATION

Kate, I dedicate this work to you. I can't imagine making it through the last few years without you. You've provided encouragement when I was down, enthusiasm when I was up, and love and support through it all. Thank you for all of the sacrifices you've made to make this all possible. You are an incredible partner in life and I look forward to conquering many future adventures with you. We are an optimal match. I love you.

ACKNOWLEDGMENTS

First and foremost, Amy, thank you for being a wonderful advisor and mentor. I'm immensely grateful for all of the feedback, guidance, encouragement, opportunities, and food you've provided to me. I feel very fortunate to have had the opportunity to be a part of the Center for Healthcare Engineering and Patient Safety.

Marina, thank you for all of the time you spent working with me on optimization theory and technical writing—I also enjoyed our talks about dogs.

To the many graduate students, faculty, and staff that I worked with throughout my time at the University of Michigan, I appreciate all that you did. Andrew, thanks for literally being at my side for those first two years. Jeremy, thanks for always knowing what to say—I thoroughly enjoyed working and traveling with you and will always cherish our "salt and pepper" sessions. Gene, thanks for all of the administrative help and treats.

Mom and Dad, your love and encouragement means the world to me. I feel lucky to have such great parents who provided me with so many opportunities. This accomplishment would not be possible without you. Thank you.

Matt, you're an incredible brother and I deeply admire you. Thank you for being so awesome.

To my extended family and friends, I feel lucky to have you in my life and truly appreciate all of your support—especially if you are reading this. Thank you.

Zoey, I know you are just a dog and will surly never read this, but your wagging support has been incredibly helpful throughout this process. You've been enthusiastic about *all* of my research ideas, made me laugh and smile every day, and provided snuggles when I was down. Thank you.

General Armacost and Doc Lowe, you have inspired me more than you will ever know. You introduced me to operations research and initiated my fascination with optimization and scheduling. I hope to inspire and motivate future students that same way you inspired and motivated me. Thank you.

I thank the U.S. Air Force and the Department of Management at the Air Force Academy for enabling me to pursue my advanced degrees. I'm excited to apply what I've learned during my next assignment and look forward to returning to the faculty.

I also thank the Seth Bonder Foundation for all of its support.

Finally, as an Air Force member, I am required to acknowledge that the views expressed in this dissertation are mine and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

TABLE OF CONTENTS

Dedication . . . . **ii**

Acknowledgments . . . . **iii**

List of Figures . . . . **vi**

List of Tables . . . . **vii**

Abstract . . . . **viii**

Chapter

1 Introduction **1**

2 Building Coordinated Operating Room and Surgical Clinic Schedules Using Integer Programming **6**

2.1 Introduction 6
2.2 Problem Statement 8
2.2.1 Assignment Decisions 8
2.2.2 Requirements 9
2.2.3 Goals 10
2.3 Literature Review 11
2.4 ICORS Formulation 12
2.4.1 Sets 13
2.4.2 Model 1: Basic Assignment Variables 14
2.4.3 Model 2: Daily Assignment Variables 16
2.4.4 Model 3: Weekly Template Variables 19
2.4.5 Objective Function 21
2.4.6 Complete ICORS Formulation 21
2.5 Computational Testing 27
2.5.1 Input Data 27
2.5.2 Computational Analysis 28
2.5.3 Analysis Summary 33
2.6 Summary 34

3 Comparing Modeling Approaches for Solving a Medical Residency Block Scheduling Problem **35**

3.1 Introduction 35

3.2 Literature Review . . . 36
3.3 Block Scheduling Problem Description . . . 39
3.4 Variation One . . . 41
3.4.1 Service Model Formulation . . . 41
3.4.2 Computational Testing . . . 43
3.5 Variation Two . . . 45
3.5.1 Pattern Model Formulation . . . 47
3.5.2 Computational Testing . . . 51
3.6 Variation Three . . . 54
3.6.1 Simple-Activity Model Formulation . . . 55
3.6.2 Complex-Activity Model Formulation . . . 57
3.6.3 Computational Testing . . . 58
3.7 Summary . . . 60

4 Scheduling Medical Residents With Conflicting Requests For Time Off . . . 61

4.1 Introduction . . . 61
4.2 Literature Review . . . 62
4.2.1 Healthcare Personnel Scheduling . . . 62
4.2.2 Generating Maximally-Feasible and Minimally-Infeasible Sets . . . 65
4.3 Resident Scheduling Problem . . . 66
4.3.1 Description of Residency . . . 67
4.3.2 Schedule Requirements . . . 67
4.3.3 Time-Off Requests . . . 68
4.3.4 Resident Scheduling Problem Formulation . . . 69
4.4 RSVC Algorithms . . . 73
4.4.1 Terminology and Notation . . . 73
4.4.2 Sequential RSVC Algorithm . . . 75
4.4.3 Simultaneous RSVC algorithm . . . 83
4.5 Computational Testing . . . 91
4.5.1 Input Data . . . 91
4.5.2 Problem Instance Characteristics . . . 93
4.5.3 Type 1 Problem Instances . . . 94
4.5.4 Type 2 Problem Instances . . . 99
4.5.5 Results Summary . . . 100
4.6 Case Study . . . 101
4.6.1 Case 1 . . . 101
4.6.2 Case 2 . . . 104
4.6.3 Case 3 . . . 109
4.6.4 Case Study Feedback . . . 110
4.7 Conclusion and Future Research . . . 110

5 Conclusion . . . 112

Bibliography . . . 114

LIST OF FIGURES

3.1 Box & Whisker Plot of Solve Times for Variation One 45
3.2 Box & Whisker Plot of Solve Times for Variation Two 53
3.3 Box & Whisker Plot of Solve Times for Variation Three 59
4.1 Sequential Request Selection Via Cuts 82
4.2 Simultaneous Request Selection Via Cuts 89
4.3 Feasibility of Problem Instances 94
4.4 Numbers of Maximally-Feasible and Minimally-Infeasible Sets for Type 1 Problem Instances 96
4.5 Median, Minimum, and Maximum Run-Times for Type 1 Instances 97
4.6 Algorithm Run-Time Comparison for Type 1 Problem Instances 98
4.7 Algorithm Run-Time Comparison for Type 1 Problem Instances (Logarithmic Scaling) 98
4.8 Algorithm Run-time Comparison for Type 2 Problem Instances 100
4.9 Algorithm Run-time Comparison for Type 2 Problem Instances (Zoomed) . . . 100
4.10 Complete Collection of Requests and Maximally-Feasible Request Sets 102
4.11 Requests That Must Be Considered 103
4.12 After Granting All Requests Involving Weddings 103
4.13 After Granting Requests #4 and #131 104
4.14 Complete Collection of Requests and Minimally-Infeasible Request Sets . . . 106
4.15 Requests That Must Be Considered 107
4.16 After Denying Request #87 108

LIST OF TABLES

2.1 Computational Testing Scenarios 29
2.2 Test Set 1 Results 30
2.3 Test Set 2 Results 31
2.4 Test Set 3 Results 33
3.1 Test-5 Inputs 52
4.1 Computational Testing Scenarios 93
4.2 Type 1 Problem Instances 95

ABSTRACT

The goal when solving scheduling problems is to generate a high-quality schedule that satisfies every scheduling requirement. When scheduling healthcare providers, the quality of a schedule is often measured through provider satisfaction, a crucial issue that affects provider morale and patient safety. Manually generating a schedule for healthcare providers, as is often done in practice, can require a significant amount of time and effort. Additionally, since identifying a schedule that satisfies every scheduling requirement is challenging, it may not be practical to also consider all of the additional scheduling preferences that lead to improved provider satisfaction. Using computer-based mathematical programming to solve scheduling problems can dramatically decrease the time required to generate a schedule while also greatly improving the quality of the schedule. However, there are additional challenges associated with solving scheduling problems with computer-aided scheduling methods. This dissertation addresses some of these scheduling challenges in relation to scheduling healthcare providers.

Specifically, we study three healthcare provider scheduling problems in this dissertation and propose methods for overcoming challenges associated with solving them. In the first problem, surgeons must be assigned to both operating and clinical rooms while satisfying many scheduling requirements. For this problem, we elaborate on the challenges we experienced while developing a mathematical scheduling model and show how the use of alternative variable definitions allowed us to overcome those challenges. In doing so, we explore the art of modeling and its impacts on solving a real-world scheduling problem.

In the second scheduling problem we address, medical residents must be scheduled for

their training rotations. For this problem, we expand on the previously discussed concept of using alternative decision variables by showing how different decision variable definitions can be used to simplify complex scheduling rules and improve computational performance.

In both of the first two problems, it is desirable to maximize the number of individual scheduling requests that can be satisfied. Satisfying every scheduling request, however, is typically not possible. For solving the third scheduling problem we address, a resident shift scheduling problem, we develop a novel approach for resolving conflicting scheduling requests. Our approach identifies the exhaustive collection of maximally-feasible and minimally-infeasible request sets which can then be used by the decision maker to determine their preferred schedule.

CHAPTER 1

Introduction

Scheduling problems present significant challenges and are found in almost every industry and application domain, including: manufacturing (Graves, 1981; MacCarthy & Liu, 1993), commercial shipping (Stahlbock & Vo, 2008), passenger railways (Higgins et al., 1996) and airlines (Cohn & Lapp, 2010), sports (Nemhauser & Trick, 1998; Rasmussen & Trick, 2008), and healthcare (Cayirli & Veral, 2003; Cardoen et al., 2010; Hall, 2012). As a result, they have been widely studied for many decades (Pinedo, 2012). Personnel scheduling presents additional challenges above and beyond scheduling machines, due to individuals' unique characteristics and preferences (Van den Bergh et al., 2013). This is certainly true in healthcare (Cheang et al., 2003). Accommodating preferences can empower employees, which often leads to improved task performance and job satisfaction (Sagie & Krausz, 2003).

Personnel scheduling, in its most basic form, is the process of assigning people to work shifts. In some cases, *rostering* (i.e., deciding the size and composition of the workforce) is also part of the scheduling process. Effective personnel scheduling and rostering can reduce operating costs while improving workplace morale, but can be challenging in practice due to the combinatorial nature of the problem. Another major challenge of personnel scheduling is incorporating issues such as workload balance, fairness, and individual requests. Ernst et al. (2004b) reviews scheduling and rostering problems in many different application areas and references literature relating to the models and methods used to solve

these problems. Ernst et al. (2004a) offers an annotated bibliography on over 700 personnel scheduling papers. Van den Bergh et al. (2013) provides a review of more recent personnel scheduling literature.

For healthcare personnel, nurse scheduling has received the most attention after first appearing in the 1960's (Wolfe & Young, 1965). Much of the work has focused on assigning nurses to shifts in order to satisfy work-related rules and service coverage needs while attempting to satisfy hospital and nurse preferences. Burke et al. (2004) describes some of the models and solution approaches that have been used for nurse scheduling problems and provides a review on some of the related papers. Many other models and solution approaches for nurse scheduling problems have been published more recently (Chiaramonte & Chiaramonte, 2008; Aickelin et al., 2009; Burke et al., 2012; Smet et al., 2014; Maass et al., 2015).

For medical residents (licensed physicians who are receiving additional training from more experienced physicians), many scheduling studies focus on rotation/block scheduling (Franz & Miller, 1993; Guo et al., 2014; Bard et al., 2016; Agarwal, 2016; Proano & Agarwal, 2017) or shift scheduling (Sherali et al., 2002; Topaloglu, 2006, 2009; Topaloglu & Ozkarahan, 2011; Perelstein et al., 2016). Rotation scheduling involves assigning residents to medical units for extended periods of time in order to fulfill both individual educational and system coverage requirements. Shift scheduling for medical residents is similar to the nurse scheduling problem, but involves satisfying additional requirements related to the educational needs of the residents. Mathematical models for combining both resident rotation and shift scheduling have also been developed (Smalley & Keskinocak, 2016a).

For physicians who have completed all of their educational requirements, most scheduling work has focused on either assigning physicians to shifts or resources (e.g., operating rooms) for specific blocks of time. Erhard et al. (2016) provides a review of quantitative methods and literature related to physician scheduling. For physician shift scheduling, Carter & Lapierre (2001) present a mathematical model for scheduling emergency room

physicians. Many additional mathematical models and solution approaches for assigning physicians to shifts have also been proposed (Beaulieu et al., 2000; Brunner et al., 2009; Stolletz & Brunner, 2012; Bruni & Detti, 2014; Smalley et al., 2015; Bowers et al., 2016). For assigning resources to physicians, Blake & Donald (2002), Santibáñez et al. (2007), and Zenteno et al. (2016) each develop mathematical programming models that assign operating room block time to specific surgical specialties.

Despite the extensive amount of work that has been done within the field of scheduling healthcare providers, challenges remain. This dissertation contains three main chapters, each of which focuses on a different healthcare provider scheduling problem and addresses challenges associated with solving it.

In Chapter 2, the scheduling problem we focus on involves assigning orthopedic surgeons to both surgical and clinical rooms within a healthcare system. Each surgeon's schedule requires clinical shifts to examine and diagnose patients, as well as operating room (OR) block time to perform procedures. Scheduling both OR blocks and clinic shifts concurrently can improve utilization of both resources and, in turn, improve access to and quality of patient care. In this chapter, we propose an integer programming-based approach to address scheduling challenges faced by the orthopedic surgery team at the University of Colorado Hospital (UCH) along with computational results. We also describe how alternative variable definitions can be used to simplify the modeling of complex scheduling requirements and improve the tractability of the mathematical program used to solve the scheduling problem.

In Chapter 3, we focus on a rotation/block scheduling problem for medical residents and further explore the concept of using alternative decision variable definitions. During residency, medical residents rotate across many different services each year in order to satisfy both educational and service coverage requirements. The duration of each rotation varies by service and the exact services that each resident rotates across are primarily determined by the resident's educational *program* (e.g., pediatrics, internal medicine, surgery). Although

each program has its own unique scheduling characteristics, it is possible to define a single *block scheduling problem* that encompasses a wide array of rules and requirements and thereby covers many programs' scheduling needs. This problem statement can in turn be modeled by a single mixed-integer programming (MIP) formulation. There may be computational advantages, however, to using different formulations depending on the specific instances of each program. We formulate, and compare four different models, contrasting them both analytically and computationally, to assess characteristics of the problem structure that impact the relative success of different formulations. Our work in Chapter 3 is based on our experience in building block schedules for several programs at the University of Michigan Health System.

In Chapter 4, we focus on a shift scheduling problem for medical residents. As is the case in the first two problems we address, for scheduling healthcare providers, the objective is often to maximize provider satisfaction across the space of feasible schedules, relative to the many *hard constraints* that ensure appropriate patient coverage, adequate training opportunities, etc. For residents, a common metric for measuring satisfaction is the number of time-off requests granted. Simply maximizing this total, however, may lead to undesirable schedules since some requests have higher priority than others. For example, it might be better to grant one resident's request for a family member's wedding in place of two residents' requests to attend a rugby game. Another approach is to assign a weight to each request and maximize the total weight of granted requests, but determining weights that accurately represent residents' and schedulers' preferences can be quite challenging. Instead, we propose to identify the exhaustive collection of *maximally-feasible* and *minimally-infeasible* sets of requests which can then be used by schedulers to select their preferred solution. Specifically, we have developed two algorithms, which we call Sequential Request Selection Via Cuts (Sequential RSVC) and Simultaneous Request Selection Via Cuts (Simultaneous RSVC), to identify these sets by solving two sequences of optimization problems. We present these algorithms along with computational results

based on a real-world problem of scheduling residents at the University of Michigan C.S. Mott Pediatric Emergency Department.

Throughout this dissertation, we make a number of contributions. First, we propose a novel method for identifying solutions to scheduling problems in which it is possible to satisfy some, but not all of the scheduling preferences. Although we apply our method to a healthcare scheduling problem, it is applicable to any problem involving preferences. Second, we model real-world problems and demonstrate how alternative variable definitions can be used to simplify the modeling of complex scheduling rules and improve computation performance when solving the problems.

CHAPTER 2

Building Coordinated Operating Room and Surgical Clinic Schedules Using Integer Programming

2.1 Introduction

In order to treat patients, surgeons require blocks of time in both clinic rooms and operating rooms (ORs). Both of these are expensive and limited resources, and using them to full capacity can have many benefits including increasing patient access, reducing costs, and enhancing provider satisfaction.

Scheduling both clinic and OR blocks simultaneously can greatly improve utilization, but when taking into account all requirements and preferences, this can become a complex combinatorial optimization problem. Manual scheduling of just ORs or clinic rooms alone can be a daunting task; scheduling them concurrently without the assistance of decision support tools is virtually impossible.

Our research is motivated by a real-world scheduling problem of this type, which we name the Integrated Clinical and Operating Room Scheduling (ICORS) problem, within the University of Colorado Health (UCH) system. Increases in patient demand, the construction of new ORs, and the hiring of new surgeons were key motivating factors in recognizing the need for decision support tools to help with both incremental changes to the existing

schedule and green field analyses for strategic planning. Working in close collaboration with UCH providers and staff, we have developed a mathematical programming-based approach that captures a wide range of requirements, preferences, and system characteristics that are critical to ensuring an acceptable and implementable solution.

In the course of this work, we also observed a number of interesting modeling issues. In particular, we went through several iterations where, based on increasing levels of knowledge about the system, we formulated successive integer programs that were effective at solving the problem as we currently understood it, but became inadequate when new information and requirements became available. In particular, we observed that bundling progressively larger amounts of information into the definition of an individual variable was key to achieving tractability for the most realistic version of the problem.

Thus, we seek to make contributions of two types through this chapter: 1) We present a new model for solving and analyzing a real-world scheduling problem, ICORS, that allocates time to surgeons in both clinic and operating rooms and 2) We demonstrate how alternative variable definitions were used to simplify the modeling of complex scheduling requirements and improve the tractability of the mathematical program used to solve the scheduling problem. In doing so, we explore the art of modeling and its impacts on solving a real-world scheduling problem.

The remainder of this chapter is organized as follows. Section 2.2 formally defines ICORS and Section 2.3 presents a review of the related literature. Section 2.4 elaborates on the usefulness of alternative variable definitions for modeling complex scheduling requirements and presents the mathematical programs developed to address ICORS. Section 2.5 contains computational testing results. Finally, a conclusion and summary, are included in Section 2.6.

2.2 Problem Statement

This chapter addresses a scheduling problem for which orthopedic surgeons must be assigned to shifts in both operating and clinical rooms within UCH, subject to numerous provider and patient-care requirements and limited availability of resources. In this section, we define the assignment decisions, requirements, and goals of ICORS.

2.2.1 Assignment Decisions

The fundamental decisions in ICORS are to determine the number and type of rooms to assign to each provider during each time slot.

- **Providers:** The Orthopedic Surgery team at UCH consists of 26 providers. Each provider has an *expertise*, of which there are six: sports medicine, adult reconstruction, hand, trauma, oncology, and foot/ankle.

- **Room Types:** Operating and clinic rooms are located at different facilities throughout the health system. Each room is assigned a single *type* based on not only its location but also the equipment that it contains. For example, all operating rooms at the main hospital with spinal surgery equipment are one room type. Additional "room types" are defined to represent other scheduling obligations such as dedicated research time and time off. In total, there are 15 different room types that can be assigned to providers.

- **Numbers of Rooms:** Providers can be assigned to multiple rooms at once, but all rooms assigned must be of the same type. Specifically, providers can be assigned one or two operating rooms (of the same type) at a time. For clinic rooms, providers can be assigned three or four rooms (of the same type) at a time. By being assigned multiple rooms at once, a provider is able to serve more patients in a day since the provider can serve a patient in one room while the other room(s) are being changed

over from one patient to the next.

- **Times:** The scheduling horizon is one month (four weeks) long and involves only weekdays. Each weekday is divided into two "half-days"(morning and afternoon).

2.2.2 Requirements

When deciding how to assign providers to rooms, the following rules must be satisfied:

1. Providers can only be assigned to one room type at a time.
2. There are a limited number of rooms of each room type available during each half-day and providers do not share rooms with other providers during half-days. Room availability varies from day to day over the month since other surgical groups also use them.
3. Providers may only work in specific room types, each of which is associated with a specific location. These restrictions are provider-specific.
4. For each room type, each provider has a minimum number of rooms that they can be assigned per half-day.
5. For each room type, each provider has a minimum and maximum number of half-days for which they must be assigned to that room type per month.
6. Some providers have individual "pre-commitment" half-day scheduling requirements (i.e., some providers must be assigned a specific room type during specific half-days).
7. UCH coverage needs dictate specific requirements for the number of rooms of a specific type assigned to a specific provider expertise during a specific half-day (e.g., at least two ORs with trauma equipment in the main hospital must be assigned to trauma providers for every half-day).

8. UCH coverage needs dictate specific requirements for the number of providers of a specific expertise assigned to specific room types during specific half-days (e.g., at least one trauma provider must be assigned to operating rooms with trauma equipment in the main hospital for every half-day).

9. Providers may only switch room types between morning and afternoon half-days for provider-specific room type combinations. For example, some providers are willing and able to work in a Denver facility in the morning and then drive 30 miles to work in a Boulder facility in the afternoon, but other providers are not.

10. Some providers require being assigned a single type of room for full-days. This requirement is primarily driven by the time required to perform certain types of operations. Additionally, when providers are assigned to a full-day in a single type of room, they must be assigned the same number of rooms of that room type during the morning and afternoon half-days.

2.2.3 Goals

In addition to satisfying the requirements listed in Section 2.2.2, it is desirable to satisfy the following preferences:

1. **Preferred half-days:** Providers each have preferred half-day assignments. Each preference includes a preferred half-day (e.g., the morning of May 26) and room type.

2. **Extra rooms:** Some providers prefer to be assigned an additional room (above their required minimum) during half-days. For example, a provider that requires being assigned three main hospital outpatient clinic rooms at a time may prefer to be assigned a fourth room if capacity is available.

3. **Weekly Continuity:** Providers prefer a consistent schedule from week-to-week.

A consistent schedule is helpful for balancing work loads and makes it easier for providers to remember when and where they work. However, it is not possible to generate a schedule for a single week that is repeated for the entire month since room and provider availability vary on a week-to-week basis. Furthermore, some providers do not require clinic or operating rooms every week.

2.3 Literature Review

Personnel scheduling, generally speaking, is the process of identifying schedules for individuals such that all scheduling rules and requirements are satisfied. The goals of personnel scheduling vary and depend on the situation, but often involve identifying low cost and/or high quality schedules. Schedule quality can be measured many different ways. Typically, schedule quality is based on fairness and/or its ability to satisfy individual scheduling preferences. Thus, effective personnel scheduling has the potential to reduce operating costs while improving workplace morale. However, due to the combinatorial nature of scheduling problems, it can be difficult to identify a schedule that satisfies every scheduling requirement, let alone the cost and quality of the schedule. As such, personnel scheduling is a widely studied field with Ernst et al. (2004a) providing an annotated bibliography on over 700 personnel scheduling papers. Ernst et al. (2004b) reviews the literature on many different scheduling problems and the methods used to solve these problems. Van den Bergh et al. (2013) provides a review of more recent personnel scheduling literature.

For healthcare personnel scheduling, nurse scheduling has received the most attention in the literature. The Nurse Scheduling Problem (NSP) involves assigning nurses to shifts under a variety of individual and system-wide scheduling requirements. The NSP made its first appearance in literature in the 1960's (Wolfe & Young, 1965). Burke et al. (2004) provides a review of much of the nurse scheduling literature and De Causmaecker & Vanden Berghe (2011) presents a categorization for nurse scheduling problems.

For scheduling physicians, much of the literature focuses on either assigning physicians to shifts or resources (e.g, operating rooms). Erhard et al. (2016) reviews much of the literature relating to scheduling healthcare physicians. For physician shift scheduling, Carter & Lapierre (2001) describes a scheduling problem involving emergency room physicians and presents and mathematical model for solving it. Bowers et al. (2016) addresses a specific scheduling problem for neonatal physicians and presents a solution approach that considers each physicians preferences for specific shifts.

For assigning resources to physicians, Blake & Donald (2002), Santibáñez et al. (2007), and Zenteno et al. (2016) each develop mathematical programming models that assign operating room block time to specific surgical specialties, but do not consider the scheduling of individual surgeons. Gunawan & Lau (2012) proposes a model that assigns both resources and blocks of time to individual physicians in order for them to complete all of their tasks for a single week. In their mathematical model, the primary decision variables represent whether or not a specific physician is assigned to a specific duty on a specific day during a specific shift. The problem addressed is unique to those addressed in previous works in that it considers all activities that must be completed in a single week by each individual physician, as opposed to a single type of activity, such as surgeries. In that respect, this problem is most similar to ICORS. However, in ICORS the planning horizon is one month (as opposed to one week) and many relationships exist between daily shifts and across weeks that must be considered when determining a schedule. Thus, a new formulation for addressing the problem is necessary.

2.4 ICORS Formulation

Before presenting the ultimate formulation that we used to solve ICORS, we believe it is of value to review the intermediate formulations that we considered and present the process that we took in reaching the final formulation. In particular, we observe that the

way in which we defined our core variables evolved as new information about the problem definition was presented to us. Thus, we begin by first discussing this evolution and then presenting the ultimate model in Section 2.4.4.

2.4.1 Sets

We use the following notation throughout the remainder of the chapter and provide it here for reference:

- P is the set of all providers.
- R is the set of room types.
- A is the set of all room assignments with $n(a)$ and $r(a)$ representing the quantity and room type of room assignment $a \in A$, respectively.
- B is the set of *work blocks*.
- C is the set of *pre-commitments*.
- W is the set of work weeks in a month (i.e, $\{1, 2, 3, 4\}$).
- T is the set of *templates* where each template is a set of weeks.
- Y is the set of work days-of-the-week (i.e, {Monday, Tuesday, Wednesday, Thursday, Friday}).
- D is the set of work days in a month (i.e, { 1st Monday, 1st Tuesday, ..., 4th Friday}).
- H is the set of work half-days in a month (i.e, { 1st Monday morning, 1st Monday afternoon, ..., 4th Friday afternoon }).
- F is the set of *preferred half-days*.
- E is the set of provider expertises.

- $\hat{E}$ is the set of expertise requirements.
- P^e is the set of all providers with expertise $e \in E$.

2.4.2 Model 1: Basic Assignment Variables

The initial version of the ICORS, as it was presented to us, included only Requirements 1-8 in Section 2.2.2. In the simplest and most direct case, this problem can be modeled by defining integer variables for representing the number of rooms of each type a provider is assigned during each half-day. However, to prevent providers from being assigned to more than one room type at a time, it is necessary to define additional binary variables to represent whether or not each provider is assigned rooms of each room type during each half-day and constraints to link the binary and integer variables together.

By using these integer variables in the model, additional variables and constraints must also be added to restrict providers from being assigned to certain quantities of rooms. For example, for most types of clinic rooms, providers may only be assigned to zero, three, or four rooms—they cannot be assigned to only one or two rooms. Modeling these "not equal to" constraints with integer variables requires additional variables and constraints.

As an alternative, we can instead define a room assignment as being a room type and *quantity* of rooms for that type. For example, if it is allowable to simultaneously assign three or four main-campus clinic rooms, we define the two permissible assignments: "3-Main-Campus Clinic Rooms" and "4-Main-Campus Clinic Rooms." We then define A to be the set of all room assignments with $n(a)$ and $r(a)$ representing the quantity and room type of room assignment $a \in A$, respectively. With these definitions, the problem can be modeled by defining binary variables of the type x_{pah} which take value 1 if provider p is assigned exactly $n(a)$ rooms of type $r(a)$ during half-day h for all $p \in P$, $a \in A$, $h \in H$. Notice that variables are not required to represent whether or not providers are assigned zero rooms since doing so is equivalent to setting all associated x_{pah} variables to zero. Thus, by only defining binary variables that represent valid room assignments, it is

implicitly ensured that providers are only assigned to acceptable numbers of rooms.

This basic x_{pah} variable definition is easy to understand and enables capturing Requirements 1-8 listed in Section 2.2.2 with relatively simple equations. For Requirement 1, since room types are defined to represent every possible provider assignment during each half-day (including time off), we can enforce the rule that a provider must be assigned to exactly one room type during each half-day. Specifically, the following set of equations enforces Requirement 1:

$$\sum_{a \in A} x_{pah} = 1 \quad \forall p \in P, h \in H \tag{2.1}$$

Requirement 2 can be enforced by ensuring there is adequate room availability for every room type during every half-day. With α^{rh} representing the quantity of rooms of type r that are available during half-day h, the following set of equations is sufficient:

$$\sum_{p \in P} \sum_{\substack{a \in A: \\ r(a)=r}} x_{pah} \leq \alpha^{rh} \quad \forall r \in R, h \in H \tag{2.2}$$

Requirements 3 and 4 can be modeled using the assignment variables by setting x_{pah} to zero (or by simply not defining x_{pah}) if provider p is not allowed assignment a during half-day h.

To model Requirement 5, let $\underline{\mu}^{pr}$ represent the minimum number of half-days provider p requires of room type r during the month and let $\bar{\mu}^{pr}$ represent the maximum number of half-days provider p is allowed of room type r during the month. Then, the following equations can be used to satisfy these requirements:

$$\underline{\mu}^{pr} \leq \sum_{h \in H} \sum_{\substack{a \in A: \\ r(a)=r}} x_{pah} \leq \bar{\mu}^{pr} \quad \forall p \in P, r \in R \tag{2.3}$$

Requirement 6 can be modeled by ensuring providers are assigned to specific room types

during specific half-days in order to satisfy their pre-commitments:

$$\sum_{\substack{a \in A: \\ r(a)=r}} x_{pah} = 1 \quad \forall (p, r, h) \in C \tag{2.4}$$

Requirement 7 can be modeled by ensuring a sufficient number of rooms are allocated to the providers of a specific expertise during specific half-days. Here, let $\underline{\epsilon}^{erh}$ and $\overline{\epsilon}^{erh}$ be the lower and upper bounds on the number of rooms that must be assigned to providers of expertise e for room type r during half-day h, respectively:

$$\underline{\epsilon}^{erh} \leq \sum_{p \in P^e} \sum_{\substack{a \in A: \\ r(a)=r}} (n(a) * x_{pah}) \leq \overline{\epsilon}^{erh} \quad \forall e \in E, r \in R, h \in H \tag{2.5}$$

Similarly, Requirement 8 can be modeled by ensuring a sufficient number of providers of a specific expertise are assigned to a specific room type during specific half-days. Here, let $\underline{\pi}^{erh}$ and $\overline{\pi}^{erh}$ be the lower and upper bounds on the number of providers of expertise e that must be assigned to rooms of type r during half-day h, respectively:

$$\underline{\pi}^{erh} \leq \sum_{p \in P^e} \sum_{\substack{a \in A: \\ r(a)=r}} x_{pah} \leq \overline{\pi}^{erh} \quad \forall e \in E, r \in R, h \in H \tag{2.6}$$

2.4.3 Model 2: Daily Assignment Variables

The x_{pah} variable definition works well for modeling Requirements 1 through 8 of Section 2.2.2. However, after presenting UCH with schedules generated by our initial model, we were informed of additional requirements that relate a day's morning and afternoon room assignments for each provider (Requirements 9 and 10 in Section 2.2.2).

For Requirement 9, there are certain room pairs that are not located in the same facility; for these pairs, some providers are willing and able to travel between them within a single day and others are not.

It is possible to capture these travel restrictions with the x_{pah} variables, but additional constraints are required. As an example, consider a rule that restricts all providers from traveling between room type r_1 and room type r_2 during a day. With h_1^d representing the first (morning) half-day on day d and h_2^d representing the second (afternoon) half-day on day d, this rule can be formulated using the following sets of constraints:

$$\sum_{\substack{a \in A: \\ r(a)=r_1}} x_{pah_1^d} + \sum_{\substack{a \in A: \\ r(a)=r_2}} x_{pah_2^d} \leq 1 \quad \forall p \in P, d \in D \tag{2.7}$$

$$\sum_{\substack{a \in A: \\ r(a)=r_1}} x_{pah_2^d} + \sum_{\substack{a \in A: \\ r(a)=r_2}} x_{pah_1^d} \leq 1 \quad \forall p \in P, d \in D \tag{2.8}$$

Here, (2.7) says that each provider cannot be assigned to type r_1 rooms in the morning *and* type r_2 rooms in the afternoon on any work day. (2.8) says that each provider cannot be assigned to type r_1 rooms in the afternoon *and* type r_2 rooms in the morning on any work day. Therefore, for every pair of room types this rule applies to, $2 \times |P| \times |D|$ additional constraints are required. For a 20 work-day month and 26 providers, this equates to 1,040 additional constraints for every such restriction. Given the 15 different room types, 210 different combinations of room types can occur within a day so up to 218,400 additional constraints are required to model these types of travel restrictions.

For Requirement 10, some providers perform surgeries that are long enough that they span the morning and afternoon half-days, so these surgeons can only be given full-day, not half-day, room assignments. As an example, consider a requirement for all providers in the set P_1 to be assigned a full-day shift whenever they are assigned rooms of type r_1. To model this requirement with the x_{pah} variables, the following set of constraints can be used:

$$x_{pah_1^d} = x_{pah_2^d} \quad \forall p \in P_1, a \in \{A : r(a) = r_1\}, d \in D \tag{2.9}$$

Here, constraint set (2.9) enforces the requirement that if a provider is assigned type r_1

rooms in the morning (or afternoon) on day d, then they must also be assigned the same number of type r_1 rooms in the afternoon (or morning).

Although (2.9) reduces the solution space of the problem by “fixing” the value of many of the variables, ensuring that all providers who are assigned a full-day within a single room type are assigned the same number of rooms for both the morning and afternoon requires additional constraints that add complexity to the model. To model these requirements, the following set of constraints can be used:

$$x_{pah_1^d} + \sum_{\substack{a' \in A: \\ r(a')=r(a), \\ n(a') \neq n(a)}} x_{pa'h_2^d} \leq 1 \quad \forall p \in P, a \in A, d \in D \tag{2.10}$$

Here, (2.10) prohibits assigning two different quantities of rooms to a provider for the same type of room within a single day.

As an alternative modeling approach that simplifies the modeling of rules that involve a relationship between the morning and afternoon assignments, we can define the decision variables in a slightly different way. First, let S represent the set of sequences, where a *sequence* is defined as the combination of morning and afternoon room assignments. For example, a sequence could be four main-campus clinic rooms in the morning and one main-campus trauma operating room in the afternoon. Using this approach, we then define the binary assignment variables x_{psd} to represent whether or not provider p is assigned sequence s on day d, where $p \in P$, $s \in S$, $d \in D$.

The total number of possible sequences is $(|A|)^2$, which equals 1,296 for UCH, and corresponds to 673,920 x_{psd} variables to model a 20-day schedule for the 26 providers. However, due to Requirements 9 and 10, only 591 *allowable* sequences exist for the UCH problem instance we focus on. Furthermore, due to Requirements 3 and 4, some UCH providers are limited to being assigned as few as 9 sequences while the maximum number of allowable sequences for a single provider is 225. Thus, the maximum number of x_{psd} variables required to create a 20-day schedule for 26 providers reduces to 117,000. More-

over, by only including the allowable sequences in S, we can eliminate constraint sets (2.7), (2.8), (2.9), and (2.10).

In summary, by defining decision variables so that they represent an entire day's worth of work, many constraints can be eliminated from the model. Although this modification has the potential to increase the number of variables in the model since every daily combination of room assignments would need to be represented as a sequence, we find that in practice the number of allowable sequences is manageable.

2.4.4 Model 3: Weekly Template Variables

With the variables definition from Section 2.4.3 we fully captured the set of feasible solutions. However, modeling the different objective criteria presented new challenges.

We were first asked to consider Goals 1 and 2, which can easily be modeled relative to the variable definition from Section 2.4.3. However, after generating potential schedules and presenting them to UCH, we learned that providers prefer to have week-to-week consistency in their schedule (Goal 3 of Section 2.2.3). Modeling Goal 3 with the x_{psd} variables is challenging because it requires relating some weeks to others.

In order to measure weekly continuity in a schedule, it is necessary to determine whether or not a day's assignment for a provider is repeated on the same day-of-week during other weeks. Doing this with the x_{psd} variables requires defining additional variables and constraints that increase the complexity of the model and time required to solve the problem (in some cases making it intractable). For example, to determine if provider p is assigned the same sequence on the same day-of-week for the first and third weeks of the month (this is one type of "bi-weekly" continuity) we can define the binary variables z_{psy} to represent whether or not provider $p \in P$ is assigned sequence $s \in S$ on day-of-week $y \in Y$ during both the first and third weeks of the month. Then, letting $w(d) \in W$ represent the week number of day d, and $y(d) \in Y$ represent the day-of-week of day d, we can define the following constraint set:

$$x_{psd} + x_{psd'} \geq 2 * z_{psy(d)} \tag{2.11}$$

$$\forall p \in P, s \in S, d \in \{D : w(d) = 1\}, d' \in \{D : w(d') = 3, y(d') = y(d)\}$$

By maximizing the sum of z_{psy} variables, (2.11) ensures that z_{psy} equals 1 if provider p is assigned sequence s on the same day-of-week for the first and third weeks of the month. This allows counting (and optimizing) the number of times this type of continuity occurs within a schedule. However, this approach results in increased solve times since it is highly fractional (i.e., if $x_{psd} + x_{psd'} = 1$, (2.11) is satisfied with $z_{psy(d)} = 0.5$, so binary restrictions are required for z_{psy} variables). Additionally, since there are three different types of continuity for UCH (two bi-weekly types and one weekly type) and each provider has up to 225 sequences, modeling this goal for a 20-day month (i.e., four, 5-day weeks) requires up to 87,750 additional constraints and z_{psy} variables.

To eliminate these types of relational constraints required for modeling week-to-week continuity, we first define a *template* as a set of weeks. For example, one template is defined to be the first and third weeks in the month (i.e., $\{$1,3$\}$) while another template is defined to be all four weeks of the month (i.e, $\{$1,2,3,4$\}$). Then, we define the binary assignment variables x_{psyt} to represent whether or not provider p is assigned sequence s on day-of-week y for the weeks included in template t. For example, one variable could represent whether or not Dr. Smith is assigned a full-day with three main-campus clinic rooms on Tuesday for the first and third weeks of the month.

Using the x_{psyt} variables, we eliminate the need for constraint set (11) and the z_{psy} variables described in the previous example for maximizing bi-weekly continuity within a schedule. To achieve the same result with the x_{psyt} variables, we can simply maximize the sum of x_{psyt} variables such that t is a bi-weekly template (see the full formulation in Section 2.4.6). For UCH, there are a total of seven allowable weekly templates: $\{1\}, \{2\}, \{3\}, \{4\}, \{1, 3\}, \{2, 4\}$, and $\{1, 2, 3, 4\}$. Thus, in order to model a 20-day sched-

ule, up to 204,750 x_{psyt} variables may be required. However, since many providers are not allowed to work every sequence or every weekly template, far fewer variables are required in practice.

2.4.5 Objective Function

Based on the goals listed in Section 2.2.3 we defined the following metrics for determining a schedule's quality:

- **Half-day preferences granted:** Each preferred half-day is considered granted if the associated provider is assigned the preferred room type with at least their minimum number of rooms for that room type on the preferred half-day. We count the total number of preferred half-days granted.
- **Extra rooms assigned**: If a provider prefers an extra room for any room types, we count the number of half-days of those room types for which an extra room is assigned to the provider.
- **Bi-weekly continuity score**: We count the number of times providers are assigned a sequence for a bi-weekly template ($\{1, 3\}$ or $\{2, 4\}$).
- **Weekly continuity score**: We count the number of times providers are assigned a sequence for the weekly template ($\{1, 2, 3, 4\}$).

In order to account for differences in the importance of each metric, we define weights for each metric based on user interactions. Then, we maximize the weighted sum of the metrics.

2.4.6 Complete ICORS Formulation

In this section, we provide a formal definition and description of our final scheduling model that incorporates the template variables and metrics described in Sections 2.4.4 and 2.4.5.

As input to the model, each work block $b \in B$ indicates the following:

- Provider Name
- Room type(s) allowable
- Minimum number of full-days required $(\underline{\xi}_b)$
- Maximum number of full-days allowed $(\bar{\xi}_b)$
- Minimum number of half-days required $(\underline{\eta}_b)$ (Note: full-day assignments count towards half-day requirements)
- Maximum number of half-days allowed $(\bar{\eta}_b)$

For each room type $r \in R$, each provider has a minimum number of rooms that they must be assigned to whenever they are assigned to room type r. Additionally, for each room type r, each provider p specifies whether or not they prefer being assigned an extra room (in addition to their minimum number) when they are assigned to room type r.

Each preferred half-day $f \in F$ indicates the following:

- Provider name
- Half-day
- Room type

Each expertise requirement $i \in \hat{E}$ indicates the following:

- Half-day
- Room type
- Expertise

- Minimum number of rooms ($\underline{\chi}_i$)
- Maximum number of rooms ($\bar{\chi}_i$)
- Minimum number of providers ($\underline{\pi}_i$)
- Maximum number of providers ($\bar{\pi}_i$)

Next, we describe the variables, parameters, constraints, and objective function of ICORS.

Decision Variables:

- x_{psyt}: binary variable to represent whether provider $p \in P$ is assigned sequence $s \in S$ on day-of-week $y \in Y$ for the weeks in template $t \in T$
- z: integer variable to represent the total number of half-days that are assigned an extra room (if preferred).
- u_f: binary variable to represent whether preferred half-day $f \in F$ is satisfied.

Parameters:

- ω^{tw}: equals 1 if template t contains week w and equals 0 otherwise.
- ϕ^{ps}: equals 1 if sequence s is prohibited for provider p and equals 0 otherwise.
- ν_c^{psyt}: equals 1 if provider p being assigned sequence s on day-of-week y for template t satisfies pre-commitment $c \in C$ and equals 0 otherwise.
- τ_i^{psyt}: the number of rooms that satisfy expertise requirement $i \in \hat{E}$ when provider p is assigned sequence s on day-of-week y for template t.
- σ_i^{psyt}: equals 1 if assigning provider p sequence s on day-of-week y for template t satisfies expertise requirement $i \in \hat{E}$ and equals 0 otherwise.

- υ_b^{psyt}: the number of full-days that satisfy workblock $b \in B$ when provider p is assigned sequence s on day-of-week y for template t.

- ψ_b^{psyt}: the number of half-days that satisfy workblock $b \in B$ when provider p is assigned sequence s on day-of-week y for template t.

- ζ^{rh}: the number of type r rooms available during half-day h

- ω_{rh}^{psyt}: the number of type r rooms assigned during half-day h by assigning provider p sequence s on day-of-week y for template t.

- ρ_f^{psyt}: equals 1 if assigning provider p sequence s on day-of-week y for template t satisfies preference $f \in F$ with at least the provider's minimum number of rooms and equals 0 otherwise.

- δ^{rpsyt} equals the number of half-days provider p receives a preferred extra room of type r if assigned sequence s on day-of-week y for template t.

- β: the objective function weight for half-day preferences granted.

- λ: the objective function weight for half-days with an extra room assigned.

- θ: the objective function weight for bi-weekly continuity score.

- γ: the objective function weight for weekly continuity score.

Constraints:

One sequence every day: Each provider must be assigned exactly one sequence every day.

$$\sum_{s \in S} \sum_{t \in T} \omega^{tw} x_{psyt} = 1 \quad \forall p \in P, y \in Y, w \in W \tag{2.12}$$

Prohibited sequences: Providers can only be assigned to sequences within their allowable set of sequences.

$$\sum_{t\in T}\sum_{y\in Y}\sum_{s\in S}\phi^{ps}x_{psyt}=0 \quad \forall p\in P \tag{2.13}$$

Pre-commitments: A particular provider must be assigned a particular room type with at least the provider's minimum number of rooms during a particular half-day.

$$\sum_{p\in P}\sum_{s\in S}\sum_{y\in Y}\sum_{t\in T}\nu_c^{psyt}x_{psyt}=1 \quad \forall c\in C \tag{2.14}$$

Bounds on number of rooms of a particular room type assigned to a particular expertise during a particular half-day shift: For each expertise requirement $i\in\hat{E}$, there is a lower bound ($\underline{\chi}_i$) and upper bound ($\bar{\chi}_i$) on the number of rooms of a particular room type that can be assigned to providers of a particular expertise during a particular half-day.

$$\underline{\chi}_i\leq\sum_{p\in P}\sum_{s\in S}\sum_{y\in Y}\sum_{t\in T}\tau_i^{psyt}x_{psyt}\leq\bar{\chi}_i \quad \forall i\in\hat{E} \tag{2.15}$$

Bounds on number of providers of a particular expertise in a particular room type during a particular half-day: For each expertise requirement $i\in\hat{E}$, there is a lower bound ($\underline{\pi}_i$) and upper bound ($\bar{\pi}_i$) on the number of providers of a particular expertise that can be assigned to rooms of a particular type during a particular half-day.

$$\underline{\pi}_i\leq\sum_{p\in P}\sum_{s\in S}\sum_{y\in Y}\sum_{t\in T}\sigma_i^{psyt}x_{psyt}\leq\bar{\pi}_i \quad \forall i\in\hat{E} \tag{2.16}$$

Satisfy requests for minimum number of full-day only shifts: For each workblock $b\in B$, a particular provider must get the minimum number of full days ($\underline{\xi}_b$) and no more than the maximum number of full days ($\bar{\xi}_b$) for room types in the work block that have at least the provider's minimum number of rooms.

$$\underline{\xi}_i\leq\sum_{p\in P}\sum_{s\in S}\sum_{y\in Y}\sum_{t\in T}\upsilon_b^{psyt}x_{psyt}\leq\bar{\xi}_i \quad \forall b\in B \tag{2.17}$$

Satisfy work block requests for total number of half-days: For each workblock $b \in B$, a particular provider must get the minimum number of half-days $(\underline{\eta}_b)$ and no more than the maximum number of half-days $(\bar{\eta}_b)$ for room types in the work block that have at least the provider's minimum number of rooms.

$$\underline{\eta}_b \leq \sum_{p \in P} \sum_{s \in S} \sum_{y \in Y} \sum_{t \in T} \psi_b^{psyt} x_{psyt} \leq \bar{\eta}_b \quad \forall b \in B \tag{2.18}$$

Ensure physical capacity for rooms: There must be enough rooms for the providers assigned to them.

$$\sum_{p \in P} \sum_{s \in S} \sum_{y \in Y} \sum_{t \in T} \omega_{rh}^{psyt} x_{psyt} \leq \zeta^{rh} \quad \forall r \in R, h \in H \tag{2.19}$$

Determine if preferred shifts are granted: The following constraint set assigns the variable u_f the value of 1 if and only if preferred half-day $f \in F$ is granted.

$$\sum_{p \in P} \sum_{s \in S} \sum_{y \in Y} \sum_{t \in T} \rho_f^{psyt} x_{psyt} = u_f \quad \forall f \in F \tag{2.20}$$

Measure the number of half-days for which an extra room is assigned: We count the number of half-days for which providers are assigned an extra room (when preferred).

$$\sum_{r \in R} \sum_{p \in P} \sum_{s \in S} \sum_{y \in Y} \sum_{t \in T} \delta^{rpsyt} x_{psyt} = z \tag{2.21}$$

Objective Function:

We maximize the weighted sum of preferred shifts granted, half-days with a preferred extra room assigned, bi-weekly continuity score, and weekly continuity score. We define $T_2 \subset T$ to be the set of all bi-weekly templates and $T_4 \subset T$ to be the set of all weekly templates.

$$\text{Maximize} \quad \beta \sum_{f \in F} u_f + \lambda z + \theta \sum_{p \in P} \sum_{s \in S} \sum_{y \in Y} \sum_{t \in T_2} x_{psyt} + \gamma \sum_{p \in P} \sum_{s \in S} \sum_{y \in Y} \sum_{t \in T_4} x_{psyt} \tag{2.22}$$

2.5 Computational Testing

In this section, we test the performance of our provider scheduling model formulated in Section 2.4.6 and demonstrate its ability to generate solutions to the scheduling problem presented in Section 2.2. Specifically, we address the following questions:

1. For real-world problem instances, how much time is required to solve the model?
2. How do the quality of schedules generated using our model compare to an existing UCH schedule in terms of performance metrics?

To answer these questions, we use our model to solve problem instances based on data from UCH. To do this, we use an Intel Xeon E3-1230 quad-core running at 3.20 GHz with hyper-threading and 32 GB of RAM. We also use the IBM ILOG Optimization Studio (*CPLEX*) 12.6 C++ API software package.

2.5.1 Input Data

The data we use as a "base case" is from an existing schedule used by UCH. For the base case, 26 providers must be scheduled for a 4-week month (20 workdays). Thus, there are a total of 1,040 half-day assignments for the month. Of these half-day assignments, 132 are "pre-commitments." There are 15 different room types that can be assigned to providers and 591 allowable sequences. Room capacities for each room type are based on actual capacities. Expertise requirements (see Requirements 7 and 8 in Section 2.2) are based on input from UCH. As an example of an expertise requirement, UCH requires at least one inpatient operating room at the main hospital to be assigned to a trauma provider for the morning half-day of every day.

For the 1,040 half-day assignments in the existing schedule, the maximum number of half-days with an extra room that can be granted is 445. In this schedule, 91 weekly templates and 67 bi-weekly templates are assigned. We use these numbers as a point of comparison for assessing the quality of alternative schedules.

2.5.2 Computational Analysis

We conducted three sets of computational experiments. In the first set, we used the same input data as the base case. In the second set, we increased the problem size and schedule flexibility by increasing the number of allowable sequences. In the third set, we reduced the room capacity. For each set of experiments, we evaluated varying trade-offs between the objective criteria of: (a) number of extra rooms assigned; (b) weekly and bi-weekly continuity; and (c) number of preferred half-day assignments granted.

For computational testing, we defined the 908 half-day assignments from the existing schedule that are not pre-commitments as preferred assignments. Thus, it is possible to grant up to 908 preferred assignments. With this definition of preferred assignments this way, we can measure the number of half-days for which the room type is unchanged from that of the existing schedule, recognizing that consistency with the current schedule is also desirable.

For each computational test set, we solved 30 objective function scenarios. For each scenario, the objective function weights were varied as indicated in Table 2.1. For simplicity of exposition, the objective function weight for weekly templates is always five times that of bi-weekly templates.

Table 2.1: Computational Testing Scenarios

Scenario	Objective Function Weights			
	Extra Rooms	Preferred (Unchanged) Assignments	Weekly Templates	Bi-weekly Templates
1	1	1	5	1
2	1	1	25	5
3	1	1	125	25
4	1	5	5	1
5	1	5	25	5
6	1	5	125	25
7	1	25	5	1
8	1	25	25	5
9	1	25	125	25
10	1	125	5	1
11	1	125	25	5
12	1	125	125	25
13	5	1	5	1
14	5	1	25	5
15	5	1	125	25
16	5	5	5	1
17	5	25	5	1
18	5	125	5	1
19	25	1	5	1
20	25	1	25	5
21	25	1	125	25
22	25	5	5	1
23	25	25	5	1
24	25	125	5	1
25	125	1	5	1
26	125	1	25	5
27	125	1	125	25
28	125	5	5	1
29	125	25	5	1
30	125	125	5	1

2.5.2.1 Test Set 1 - Base Case Data

In Table 2.2 we report the solve time and optimal metric values for each objective function scenario tested with the base case data. For comparison purposes, we include the metric values for UCH's existing scheduling in the bottom row.

Table 2.2: Test Set 1 Results

Scenario	Solve Time (sec)	Optimal Solution Metric Values			
		Extra Rooms	Preferred (Unchanged) Assignments	Weekly Templates	Bi-weekly Templates
1	10	630	803	103	47
2	18	625	776	109	36
3	21	617	784	109	36
4	4	449	904	97	58
5	6	421	904	103	48
6	14	435	864	109	36
7	3	437	908	94	64
8	3	397	908	103	48
9	4	397	908	103	48
10	3	437	908	94	64
11	3	397	908	103	48
12	3	397	908	103	48
13	77	644	776	102	47
14	82	642	734	109	36
15	60	642	734	109	36
16	20	624	825	89	68
17	4	445	908	91	67
18	3	445	908	91	67
19	166	644	776	102	47
20	185	644	736	108	36
21	117	642	734	109	36
22	83	644	798	90	63
23	11	629	822	87	60
24	4	445	908	91	67
25	75	644	776	102	47
26	185	644	736	108	36
27	698	644	736	108	36
28	114	644	798	90	63
29	33	644	801	84	66
30	10	625	826	87	60
Existing Schedule	N/A	445	908	91	67

From Table 2.2, we notice that the problem can be solved in under 12 minutes for each of the scenarios, with some scenarios solving in as few as 3 seconds. For scenarios in which schedule continuity is prioritized over preferred assignments (i.e., the objective weight of weekly continuity is greater than that of preferred assignments), the average solve time is 114 seconds—5.7 times greater than the average solve time of the other scenarios. We also notice that when granting preferred assignments is a relatively low priority, as in Scenarios 14 and 15, it is possible to increase the number of half-days granted with an extra room (from 445 to 642) while simultaneously assigning more weekly templates (109 instead of 91) than in the existing schedule. In other words, by allowing changes to the existing schedule, it is possible to assign more extra rooms *and* improve the weekly continuity of

the schedule.

2.5.2.2 Test Set 2 - Additional Allowable Sequences

For Test Set 2, we increased the number of allowable sequences by making every combination of room groups an allowable sequence, for a total of 841 sequences—250 more than the base case. This change increases the number of decision variables in the problem which can lead to increased solve times. Since the additional variables provide more options for an individual provider's daily schedule, it may be possible to improve the optimal objective values over those of the base case. In Table 2.3 we report the results from testing each objective function scenario for Test Set 2.

Table 2.3: Test Set 2 Results

Scenario	Solve Time (sec)	Optimal Solution Metric Values			
		Extra Rooms	Preferred (Unchanged) Assignments	Weekly Templates	Bi-weekly Templates
1	22	606	828	104	46
2	68	603	806	109	36
3	50	603	806	109	36
4	6	467	904	96	60
5	7	435	904	103	48
6	12	491	860	109	36
7	4	451	908	94	64
8	4	411	908	103	48
9	4	411	908	103	48
10	4	451	908	94	64
11	4	411	908	103	48
12	4	411	908	103	48
13	84	644	780	103	45
14	135	642	742	109	36
15	1944	642	742	109	36
16	20	624	825	95	61
17	5	463	908	89	71
18	4	463	908	89	71
19	78	644	780	103	45
20	612	644	744	108	36
21	613	642	742	109	36
22	73	644	802	91	60
23	21	629	826	88	60
24	5	463	908	89	71
25	74	644	780	103	45
26	388	644	744	108	36
27	2847	644	744	108	36
28	69	644	801	92	60
29	28	644	805	85	66
30	20	628	827	88	60

By comparing Table 2.3 to Table 2.2, we notice that the solve time remained similar

or increased for nearly every scenario. For the 30 scenarios tested, the maximum solve time was just over 47 minutes and the median solve time was 21 seconds. Although the solve times in Test Set 2 increased, the schedules generated are of better quality for every scenario (as measured by the optimal objective function value). Given the increase in the number of allowable sequences, these findings are not surprising.

2.5.2.3 Test Set 3 - Reduced Room Capacity

For Test Set 3, we modified the base case data by reducing the capacity of every room type by one room during every half-day of the planning horizon. Making this change reduces the number of half-days for which it is possible to assign providers an extra room for their preferred assignments. In Table 2.4 we report the results for each objective function scenario for Test Set 3.

Table 2.4: Test Set 3 Results

Scenario	Solve Time (sec)	Optimal Solution Metric Values			
		Extra Rooms	Preferred (Unchanged) Assignments	Weekly Templates	Bi-weekly Templates
1	18	462	812	102	50
2	43	470	768	109	36
3	50	458	780	109	36
4	8	354	862	96	57
5	8	331	858	103	48
6	15	335	828	109	36
7	6	330	866	94	61
8	5	291	866	102	49
9	8	300	864	103	47
10	5	330	866	94	61
11	4	291	866	102	49
12	5	291	66	102	49
13	365	491	763	101	47
14	415	488	718	109	36
15	69	488	718	109	36
16	22	468	820	92	65
17	9	355	864	88	70
18	6	341	866	89	68
19	255	491	763	101	47
20	719	491	712	108	37
21	356	489	712	109	36
22	345	491	780	88	67
23	13	471	818	92	60
24	8	356	864	85	70
25	269	491	760	102	45
26	491	491	712	108	37
27	903	491	712	108	37
28	371	491	780	88	67
29	55	491	781	86	68
30	10	469	820	92	60

When comparing the results in Table 2.4 to those of Test Set 1 in Table 2.2 we notice that the solve time increased for every scenario, but most scenarios were solved relatively quickly and all scenarios were solved to optimality in roughly 15 minutes or less.

2.5.3 Analysis Summary

In Section 2.5, we tested our model by solving multiple problem instances based on real-world data for a variety of objective function weight scenarios. Although the time required to solve each problem is affected by specifics of the problem instance (e.g., number of allowable sequences) and objective function weights, 56 of the 90 instances tested were solved in less than one minute and no instances required more than 48 minutes to find an optimal solution. For future work, we hope to explore the problem characteristics that lead

to longer solve times.

By testing numerous objective function weight scenarios, it is possible to assess the trade-offs that exist amongst the performance metrics and propose alternative schedules to UCH. For example, if UCH's highest priority is assigning half-days with an extra room, it is possible for them to assign up to an additional 199 half-days with an extra room.

2.6 Summary

In this chapter we address an important healthcare provider scheduling problem faced by the University of Colorado Health System. In this problem, providers must be assigned both clinic and operating rooms in order to satisfy numerous work-related requirements and restrictions.

In formulating a new integer programming model to solve the scheduling problem, we describe how alternative variable definitions can be used to simplify the modeling of complex scheduling requirements and improve the tractability of the mathematical program. Our approach is most useful for problem instances in which the number of allowable decision variables (as determined by the problem's constraints and input data) is relatively small compared to the number of possible decision variables. Through computational testing, we find that our model is able to quickly solve realistic problem instances to generate improved schedules.

CHAPTER 3

Comparing Modeling Approaches for Solving a Medical Residency Block Scheduling Problem

3.1 Introduction

Following medical school, physicians normally complete an additional three to five years of medical training as *residents*—licensed physicians who work under the supervision of attending physicians. During residency, medical residents rotate across many different services in order to satisfy both educational and service coverage requirements. The exact services that each resident rotates across are primarily determined by the resident's educational program (e.g., pediatrics, internal medicine, surgery). The duration and potential start times of each service varies by service. For example, some services are a full-month in duration and can start at the beginning of a calendar month while others are a half-month in duration and can start at the beginning and/or middle of a calender month. Schedules for each educational program are generated on an annual basis and are often called *block schedules* since the residents are assigned to services for specific blocks of time during the year.

Although it is possible to formulate a generalized mathematical model that is capable of solving each educational program's block scheduling problem, due to differences in service durations and allowable start dates across programs, there may be computational benefits to using specialized models for each program. Specifically, it may be possible to improve

computational performance by basing the definitions of decision variables on the specifics of the block scheduling problem being solved. In this chapter, we present a generalized block scheduling problem and then demonstrate how alternative variable definitions can improve computational performance in specific situations. Our research is based on our experiences scheduling residents at the University of Michigan Hospital in Ann Arbor, Michigan.

The remainder of this chapter is organized as follows: In Section 2, we review the relevant literature. In Section 3, we describe the decisions, rules, and objective of the generalized block scheduling problem that serves as the core of the specific scheduling scenarios we address in the remainder of the chapter. In Section 4, we discuss the specifics of a rather basic block scheduling problem variation, propose a mathematical model for solving it, and present computational results from solving a variety of scheduling instances with the model. In Section 5, we explain specific scheduling rules that require modifying our initial model and propose an alternative model to account for the additional rules. Through computational testing, we demonstrate the improved performance of this alternative model over that of our modified initial model. In Section 6, we describe another scheduling problem variation involving an additional rule that necessitates formulating a new scheduling model. To address the scheduling scenario we describe, we propose two model possibilities that build on our two initial models and then we compare the performance of each model through computational testing. We conclude in Section 7 with a summary of our findings and suggestions for future work.

3.2 Literature Review

Personnel scheduling problems are especially challenging due to the unique characteristics and preferences of individuals. In their most basic form, personnel scheduling problems require determining a schedule of work for individuals under a set of scheduling

requirements. Unlike machine scheduling problems, personnel scheduling problems often consider factors such as schedule fairness and/or the preferences of individuals. Accommodating the preferences of individuals can raise the morale of personnel and may lead to improved task performance (Sagie & Krausz, 2003). Van den Bergh et al. (2013) and Ernst et al. (2004b) review much of the literature related to personnel scheduling. The extent to which personnel scheduling has been studied is evident through the annotated bibliography provided by Ernst et al. (2004a) which includes more than 700 scheduling papers.

Within the application area of healthcare, much of the work on personnel scheduling has focused on scheduling nurses. The Nurse Scheduling Problem (NSP), which involves assigning nurses to shifts, has received the most attention. Nurse rostering, the process of determining the size and composition of the workforce, is another commonly addressed scheduling problem. Burke et al. (2004) explains some of the models and methods that have been used to solve nurse scheduling problems and reviews many of the relevant papers. Maass et al. (2015) is a recent paper on nurse rostering that presents a stochastic programming formulation to account for variability in patient census and nurse absenteeism.

For scheduling medical residents, much of the research has focused on: a) block/rotation scheduling and/or b) shift scheduling. Although similar to nurse scheduling problems, resident scheduling problems include additional rules relating to each resident's educational requirements, which can make them especially difficult to solve (Guo et al., 2014). Using automated shift-scheduling tools for solving resident scheduling problems can improve the quality of schedules while reducing the time required to generate them (Perelstein et al., 2016).

Block/rotation scheduling involves assigning residents to services (i.e., rotations) for blocks of time. Smalley & Keskinocak (2016a) formulates a mathematical program for solving a multi-objective block scheduling problem and then proposes a separate model for generating daily shift schedules. Agarwal (2016) also proposes separate models for generating weekly rotation schedules and daily shift schedules, using a goal programming

approach for satisfying the multiple objectives. Bard et al. (2016) proposes a model for adjusting annual rotation schedules in way that more evenly distributes the number of residents that are available to work in the clinic each half-day.

During certain rotations, residents must also be scheduled for shift work. Sherali et al. (2002) addresses a specific resident shift scheduling problem that involves assigning resident to night-shifts. Cohn et al. (2009) presents their findings from developing shift schedules for residents at Boston University School of Medicine—they propose an iterative approach in which chief residents provide feedback on solutions generated by the model until an improved schedule cannot be found. Topaloglu & Ozkarahan (2011) also addresses a multi-objective resident shift scheduling problem and proposes a column generation model the uses constraint programming to identify feasible schedules.

Many of the mathematical models proposed for solving resident scheduling problems define decision variables to indicate whether or not each resident is assigned to a specific service/shift during each block of time. For some models, such as the the weekly rotation scheduling model proposed in Agarwal (2016), additional decision variables are defined to indicate whether or not a resident starts a specific rotation during each block. These types of additional variables are useful when modeling multi-block rotations that are flexible in when they can start.

Defining decision variables to represent basic assignments (i.e., the assignment of a single rotation/shift during a specific block of time) enables modeling many problem variations. However, in some cases there can be computational advantages to defining decision variables that represent multiple decisions (Armacost et al., 2002). Although Armacost et al. (2002) focuses on using *composite variables* to model and solve a package shipping problem, the concept of using more complex decision variables to improve computational performance is applicable to a wide variety of problems. In this chapter, we explore the potential computational advantages of using alternative variable definitions for solving a resident block scheduling problem.

We start by formulating a block scheduling problem using basic decision variables to represent whether or not each resident is assigned to a specific service rotation during each block of time. We then introduce alternative models with more complex decision variables and conduct computational testing to analyze the performance of each model.

3.3 Block Scheduling Problem Description

For the general block scheduling problem we consider, residents must be assigned to services for blocks of time during the year in order to satisfy both educational and service related requirements. Thus, for each block of time within the one year scheduling horizon, we must decide which service is assigned to each resident. The duration of the blocks can vary based on specifics of the scheduling problem being solved. For example, for an educational program in which all services are one month in duration and can only start at the beginning of a calendar month, it is reasonable to define blocks to be one month in duration. However, if some services are a half-month in duration, it is necessary to define smaller blocks of time. Each resident being scheduled is a specific *level* (e.g., Level-1, Level-2, etc.) based on their experience. For assigning residents to services in each of the educational programs we consider, the following scheduling requirements must be satisfied:

1. **One Service Per Block:** Each resident must be assigned exactly one service during each block.

2. **Educational Requirements:** For each service, each resident has a lower and upper bound on the number blocks for which he or she is assigned to that service.

3. **Service Coverage Requirements:** For each block, each service has a lower and upper bound on the total number of residents that can be assigned to it. Additionally, for each block, each service may have lower and upper bounds on the number of

residents that can be assigned to it for each resident level.

4. **Prerequisite Requirements:** Each prerequisite requirement specifies a set of residents and two services, with one of the services being a prerequisite for the other. Each resident in the specified set must be assigned to the prerequisite service before they are assigned to the other service (e.g., Level-1 residents must complete Service X before they start Service Y).

5. **Spacing Requirements:** For certain sets of services, it is necessary to spread the services out across a resident's schedule (e.g., after a resident completes Service X, there must be at least four blocks of time before the resident starts Service Y). To capture these types of requirements more broadly, each spacing requirement specifies a set of services, a duration of time, and a maximum number of services. For the specified set of services, each resident is limited to being assigned to no more than the maximum number of them within the duration of time.

In addition to satisfying these requirements, it is desirable to satisfy specific scheduling requests. Each scheduling request includes a resident, a block of time, and a service. If the resident is assigned to the service during the specific block of time, the request is considered to be satisfied. In this work, we seek to maximize the total number of requests satisfied.

The following notation will be used throughout the remainder of the article:

Sets:

- R is the set of all residents.
- S is the set of all services.
- B is the set of all blocks of time in the scheduling horizon.
- L is the set of all resident levels.

- R_l is the set of all level l residents for $l \in L$.
- Q is the set of all resident requests. Each request in Q includes the triplet (r, s, b) that represents a resident $r \in R$, service $s \in S$, and block $b \in B$.
- E is the set of all prerequisite requirements. Each requirement $e \in E$ includes a pair of services (s_1^e, s_2^e) and a set of residents R_e which indicates that for each resident $r \in R_e$, s_2^e cannot occur unless s_1^e has already occurred.
- C is the set of all spacing requirements. Each requirement c in C includes a set of services S_c, a duration of time, δ_c (in blocks of time), and a maximum number of services, σ_c, which indicates that no more than σ_c of the services in S_c can be assigned to a resident within any δ_c consecutive blocks.

3.4 Variation One

We start with a relatively basic variation of the block scheduling problem in which all services are one month in duration and each service must start at the beginning of a calendar month. Therefore, we define blocks of time to correspond to the calendar months. To model this specific scheduling scenario, we define binary decision variables of type x_{rsb} that represent if resident r is assigned to service s for block (i.e., month) b. Next, we present our full mathematical formulation for modeling this scenario, which we call the *Service Model*.

3.4.1 Service Model Formulation

Parameters:

- $\underline{\omega}_{rs}$ and $\bar{\omega}_{rs}$ are the lower and upper bounds on the total number of blocks that resident $r \in R$ can be assigned to service $s \in S$, respectively.

- $\underline{\rho}_{sb}$ and $\bar{\rho}_{sb}$ are the lower and upper bounds on the total number of residents assigned to service $s \in S$ during block $b \in B$.

- $\underline{\lambda}_{lsb}$ and $\bar{\lambda}_{lsb}$ are the lower and upper bounds on the total number of level l residents assigned to service $s \in S$ during block $b \in B$.

Decision Variables:

- $x_{rsb} \in \{0, 1\}$ indicates if resident $r \in R$ is assigned service $s \in S$ for block $b \in B$.

Constraints:

$$\sum_{s \in S} x_{rsb} = 1 \qquad \forall\, r \in R, b \in B \tag{3.1}$$

$$\underline{\omega}_{rs} \leq \sum_{b \in B} x_{rsb} \leq \bar{\omega}_{rs} \qquad \forall\, r \in R, s \in S \tag{3.2}$$

$$\underline{\rho}_{sb} \leq \sum_{r \in R} x_{rsb} \leq \bar{\rho}_{sb} \qquad \forall\, s \in S, b \in B \tag{3.3}$$

$$\underline{\lambda}_{lsb} \leq \sum_{r \in R_l} x_{rsb} \leq \bar{\lambda}_{lsb} \qquad \forall\, l \in L, s \in S, b \in B \tag{3.4}$$

$$x_{rs_2^e b} = 0 \qquad \forall\, e \in E, r \in R_e, b \in \{1\} \tag{3.5}$$

$$\sum_{i=1}^{b-1} x_{rs_1^e i} \geq x_{rs_2^e b} \qquad \forall\, e \in E, r \in R_e, b \in \{2, \ldots, |B|\} \tag{3.6}$$

$$\sum_{s \in S_c} \sum_{i=b}^{b+\delta_c-1} x_{rsi} \leq \sigma_c \qquad \forall\, r \in R, c \in C, b \in \{1, \ldots, |B| - \delta_c + 1\} \tag{3.7}$$

Here, (3.1) ensures that each resident is assigned exactly one service during each block. Constraint set (3.2) enforces the educational requirements for each resident. (3.3) sets bounds on the total number of residents assigned to each service during each block. Similarly, (3.4) sets bounds on the number of residents of each level assigned to each service during each block. Together, (3.5) and (3.6) ensure that each prerequisite requirement e in

E is satisfied. Specifically, (3.5) ensures that s_2^e does not occur during the first block and (3.6) ensures that if s_2^e occurs, s_1^e occurs at least once in a prior block. Finally, constraint set (3.7) enforces each spacing requirement c in C by limiting the number of services in S_c that can be assigned during any consecutive δ_c blocks to σ_c.

Objective Function:

The objective is to maximize the number of resident requests satisfied:

$$\max \sum_{(r,s,b)\in Q} x_{rsb} \tag{3.8}$$

3.4.2 Computational Testing

To demonstrate the time required to solve fairly simple variations of the block scheduling problems using our Service Model and assess how the size of the problem instance affects solve times, we consider four different sets of data instances for Variation One. Across the sets, we vary the number of residents and number of services. There are three levels of residents in each data set: one-fourth are Level-1, one-fourth are Level-2, and one-half are Level-3. For all of the sets, three of the services are considered "hard" services. For every three consecutive blocks of a resident's schedule, only two of the hard services can be assigned. Level-1 residents have a prerequisite requirement that requires them to be assigned the service "PreReq1" before they are assigned the service "PreReq2." For the services "Vacation" and "Elective," each resident requests when (i.e., which block) they would like each service to occur.

For the first data set, labeled "Test-1," 100 residents must be scheduled. In Test-1, there are a total of 12 services and each resident must be assigned to each service exactly once. For the ten services other than "Vacation" and "Elective," eight to ten residents must be assigned to each service during each block—at least one resident must be Level-3 and no more than three can be Level-1.

For Test-2, we double the number of residents being scheduled to 200. Similarly, we double the coverage requirements. Specifically, sixteen to twenty residents must be assigned to each service other than “Vacation” and “Elective” during each block—at least two residents must be Level-3 and no more than six can be Level-1.

For Test-3, 100 residents must be scheduled, but we double the number of services to 24. In this case, eight of the services (including “Vacation” and “Elective”) are required for all residents, and each resident must be assigned four of the sixteen “optional” services. Eight to ten residents must be assigned to each of the mandatory services during each block and two to four residents must be assigned to each optional service during each block. For all services other than “Vacation” and “Elective,” at least one Level-3 resident must be assigned to it during each block and no more than three Level-1 residents can be assigned to it during each block.

For Test-4, 200 residents must be scheduled and there are 24 services. Like Test-3, eight of the services are mandatory for all residents and each resident must be assigned to four of the sixteen “optional” services. We double the coverage requirements of Test-3 which means that sixteen to twenty residents must be assigned to each mandatory service during each block and four to eight residents must be assigned to each optional service during each block. For all services other than “Vacation” and “Elective,” at least two Level-3 residents must be assigned to it during each block and no more than six Level-1 residents can be assigned to it during each block.

For the purposes of testing, we create ten problem instances for each case by randomly generating a set of requests for each instance. For Test-3, we use that same sets of requests as Test-1. For Test-4, we use the same sets of requests as Test-2. For all computational testing throughout the remainder of this chapter, we use an Intel Xeon E3-1230 quad-core running at 3.20 GHz with hyper-threading and 32 GB of RAM. We use the IBM ILOG Optimization Studio (*CPLEX*) 12.6 C++ API software package. In Figure 3.1, we report the solve times for each Variation One Test.

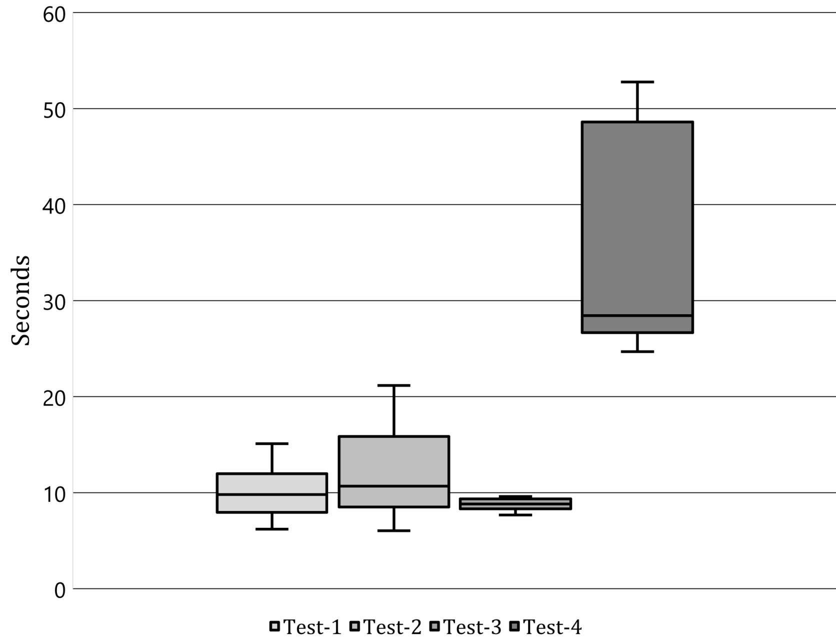


Figure 3.1: Box & Whisker Plot of Solve Times for Variation One

From Figure 3.1, we notice that an optimal solution was found for all instances in less than one minute. For the majority of instances in the first three tests, an optimal solution was found in approximately ten seconds. For Test-4, which included the most residents and most services of the cases tested, the median solve time was 28 seconds. We recognize that our sets of test data are rather simplistic and do expect longer solve times for larger, more complex data sets. However, based on our results and the fact that it is only necessary to solve these types of problems annually, we conclude that our Service Model is adequate for solving simple block scheduling problems similar to those presented in this section.

3.5 Variation Two

The mathematical model presented in Section 3.4 works well for solving rather basic block scheduling problem variations. However, for more complex variations there can be computational advantages (or even necessities) to using to using an alternative model. In this section we describe a more complex variation of the block scheduling problem that we have

encountered in our experience with the University of Michigan Health System (UMHS). We then propose an alternative model for solving it and compare the performance of the two models.

Specifically, we have observed that it is not always the case that service assignments are made in one-month blocks. Some assignments might only be for a half month, some assignments might be for a full month, and in some cases the educational requirement is for a full month, but it is acceptable to split the assignment into two separate half-month blocks. This is the case in the UMHS programs in pediatrics and in internal medicine where, for example, it may be necessary to spend a month on an emergency medicine service, but this can be done in two half-month blocks.

A further complication stems from the fact that for those activities which can be split into half-month blocks, not all combinations of services are valid. For example, a resident can spend a half-month on emergency medicine followed by a half month on vacation, or a half month on emergency medicine followed by a half month on elective, but cannot combine elective and night-team in the same month.

To model these types of *compatibility restrictions* using the Service Model, we can define the blocks to be a half-month in duration and then define additional constraints to restrict which combinations of services can occur within a single month for each resident. Specifically, there are two types of combinations that require additional constraints: 1) full-month services (i.e., services that must be assigned to both blocks of a month) and 2) prohibited combinations of half-month services.

Enforcing full-month service restrictions can be done by defining constraints that set decision variables for the second half of a month equal to decision variables for the first half of that month. For example, letting S_1 be the set of all one-month services, B_1 be the set of all blocks that occur during the first half of a month, and $b^\star$ be the block corresponding to the second half of the month for block $b \in B_1$, we can model full-month service

relationships using the x_{rsb} decision variables with the following set of constraints:

$$x_{rsb} = x_{rsb^\star} \quad \forall r \in R, s \in S_1, b \in B_1 \tag{3.9}$$

Although modeling the one-month services in this manner requires additional constraints, constraint set (3.9) actually fixes the values of many decision variables, which in actuality reduces the overall size of the problem instance.

To model prohibited combinations of half-month services however, we must ensure that specific combinations of services are not assigned to any residents within any month. To do this, for each prohibited combination of half-month services, we must define constraints that prevent both services from being assigned within a single month. For example, if the half-month services s_1 and s_2 are not allowed to occur within the same month, we can enforce this restriction using the following sets of constraints:

$$x_{rs_1b} + x_{rs_2b^\star} \leq 1 \quad \forall r \in R, b \in B_1 \tag{3.10}$$

$$x_{rs_2b} + x_{rs_1b^\star} \leq 1 \quad \forall r \in R, b \in B_1 \tag{3.11}$$

Together, (3.10) and (3.11) ensure that s_1 and s_2 are not assigned to a resident within the same month. Unlike constraint set (3.9), constraint sets (3.10) and (3.11) are highly fractional and can lead to increased solve times. Instead of defining additional constraint sets (3.9), (3.10), and (3.11) to account for half-month service relationships, we propose an alternative model in which we define the decision variables differently.

3.5.1 Pattern Model Formulation

As an alternative to defining variables of type x_{rsb} as is done for the Service Model described in Section 3.4, we define variables of type y_{rpt} to represent if resident r is assigned *service-pattern* p during *time interval* t. Here, a service-pattern is an allowable sequence

of services that can occur within a multi-block time interval. For example, in the situation described above, the blocks would be half months, the intervals would be full months (i.e. two blocks), and we would define patterns of the form (emergency medicine, vacation) and (vacation, emergency medicine) as well as (S1, S1) for any service that must be a full month, but would not define prohibited patterns (S1, emergency medicine), (S1, Vacation), etc.

By using variables of type y_{rpt}, constraints for enforcing relationships between services within intervals of time (i.e., (3.9), (3.10), and (3.11)) are not required. Instead, we only define decision variables to represent each allowable pattern. Thus, as more compatibility restrictions are added to the scheduling problem, the Service Model grows through the addition of constraints, but the Pattern Model shrinks through the elimination of decision variables. Consequently, the computational advantages of the Pattern Model are likely to increase as more compatibility restrictions are added. Although there may be many possible patterns, we find that in practice the number of allowable patterns is manageable.

In this example we have focused on assigning pairs of services for each month, but we define a generalized version of the mathematical model, which we call the *Pattern Model*. Next, we define the sets (in addition to those defined in Section 3.3), parameters, and mathematical formulation for our Pattern Model.

Additional Sets:

- $T \in \{1, 2, \ldots, |T|\}$ is the set of all time intervals in the scheduling horizon. Every time interval is equal in length and is divided into n equally-long blocks.

- P is the set of all service patterns that can occur within a single time interval $t \in T$. Each pattern $p \in P$ is represented as $(s_1, s_2, \ldots, s_n)$ to indicate which services occurs during each block of the pattern.

- $P^{\star}$ is the set all patterns for prerequisite requirement $e \in E$ in which: a) service s_2^e

occurs before service s_1^e, or b) service s_2^e occurs, but s_1^e does not.

- C_p is the set of all pattern-specific spacing requirements. Here, we assume that block-specific spacing requirements are translated into pattern-specific spacing requirements before solving the problem. Each requirement c in C_p includes a set of patterns P_c, a duration of time, δ_c (in time intervals), and a maximum number of patterns, σ_c, which indicates that no more than σ_c of the patterns in S_c can be assigned to a resident within any δ_c consecutive time intervals.

Parameters:

- $\underline{\rho}_{st}^j$ and $\bar{\rho}_{st}^j$ for $j \in \{1, \ldots, n\}$ are the lower and upper bounds on the total number of residents assigned to service $s \in S$ in block j of interval $t \in T$.

- $\underline{\lambda}_{lst}^j$ and $\bar{\lambda}_{lst}^j$ for $j \in \{1, \ldots, n\}$ are the lower and upper bounds on the total number of level $l \in L$ residents assigned to service $s \in S$ in block j of interval $t \in T$.

- $\tau_{sp} \in \{0, 1, \ldots, n\}$ is the total number of blocks of service $s \in S$ included in pattern $p \in P$.

- $\alpha_{sp} \in \{0, 1\}$ indicates whether or not service $s \in S$ occurs in pattern $p \in P$.

- $\beta_{sp}^j \in \{0, 1\}$ for $j \in \{1, \ldots, n\}$ indicates whether or not service $s \in S$ occurs in block j of pattern $p \in P$.

- $\pi_{rptq} \in \{0, 1\}$ equals 1 if assigning resident $r \in R$ pattern $p \in P$ during interval $t \in T$ satisfies request $q \in Q$, and equals 0 otherwise.

Decision Variables:

- $y_{rpt} \in \{0, 1\}$ indicates if resident $r \in R$ is assigned service pattern $p \in P$ for interval $t \in T$.

Constraints:

$$\sum_{p\in P} y_{rpt} = 1 \qquad \forall\, r \in R, t \in T \tag{3.12}$$

$$\underline{\omega}_{rs} \leq \sum_{p\in P}\sum_{t\in T} \tau_{sp} y_{rpt} \leq \bar{\omega}_{rs} \qquad \forall\, r \in R, s \in S \tag{3.13}$$

$$\underline{\rho}^j_{st} \leq \sum_{p\in P}\sum_{r\in R} \beta^j_{sp} y_{rpt} \leq \bar{\rho}^j_{st} \qquad \forall\, s \in S, j \in \{1,\ldots,n\}, t \in T \tag{3.14}$$

$$\underline{\lambda}^j_{lst} \leq \sum_{p\in P}\sum_{r\in R_l} \beta^j_{sp} y_{rpt} \leq \bar{\lambda}^j_{lst} \qquad \forall\, l \in L, s \in S, t \in T \tag{3.15}$$

$$\sum_{p\in P^\star} y_{rpt} = 0 \qquad \forall\, e \in E, r \in R_e, t \in \{1\} \tag{3.16}$$

$$\sum_{i=1}^{t-1}\sum_{p\in P} \alpha_{s^e_1 p} y_{rpi} \geq \sum_{p\in P^\star} y_{rpt} \qquad \forall\, e \in E, r \in R_e, t \in \{2,\ldots,|T|\} \tag{3.17}$$

$$\sum_{p\in P_c}\sum_{i=t}^{t+\delta_c-1} y_{rpi} \leq \sigma_c \qquad \forall\, r \in R, c \in C_p, t \in \{1,\ldots,|T|-\delta_c+1\} \tag{3.18}$$

Here, (3.12) ensures that each resident is assigned exactly one pattern during each time interval. Constraint set (3.13) enforces the educational requirements for each resident. (3.14) sets bounds on the total number of residents assigned to each service during each block. Similarly, (3.15) sets bounds on the number of residents of each level assigned to each service during each block. Together, (3.16) and (3.17) ensure that each prerequisite requirement e in E is satisfied. Specifically, (3.16) ensures that s^e_2 does not occur before s^e_1 during the first time interval and (3.17) ensures that s^e_1 occurs in a block prior to when s^e_2 occurs. Finally, constraint set (3.18) enforces each spacing requirement.

Objective Function:

The objective is to maximize the number of resident requests satisfied:

$$\max \sum_{r \in R} \sum_{p \in P} \sum_{t \in T} \sum_{q \in Q} \pi_{rptq} y_{rpt} \tag{3.19}$$

3.5.2 Computational Testing

The specific data sets we use in this section to compare the computational performance of the Pattern Model to the Service Model are approximately based on those we have encountered when scheduling pediatric and internal medicine residents. For each test, all services are either a full or half month in duration and full-month services correspond to the calendar months. For solving each problem instance, we define blocks to be a half month in duration and time intervals that correspond with the calendar months.

For *each* service in the data sets, we define the following inputs (see Table 3.1 for example input values):

- **Name:** Specifies the name of the service.
- **Duration:** Indicates whether the service must be assigned as a full month or can be assigned in half month blocks.
- **Min Blocks:** The minimum number of half-month blocks required by each resident.
- **Max Blocks:** The maximum number of half-month blocks that can be assigned to each resident.
- **Compatibility:** A list of each half-month service that can occur during the same month.
- **Min Coverage:** The minimum number of total residents that must be assigned to the service during each block.
- **Max Coverage:** The maximum number of total residents that can be assigned to the service during each block.

- **Max Level-1:** The maximum number of Level-1 residents that can be assigned to the service during each block.

- **Min Level-3:** The minimum number of Level-3 residents that must be assigned to the service during each block

- **Level-1 PreReq:** Indicates which other service(s), if any, are a prerequisite for Level-1 residents.

- **isHard**: Indicates whether or not the service is defined to be a "hard" service. Hard services are subject to spacing requirements (as described in Section 3.3) in the problem instances we test.

- **isPreference**: Indicates whether or not residents can request when each block of the service is assigned to them.

In Test-5, 100 residents must be scheduled and there are nine total services. For simplicity, we name the services S1, . . . , S9. The Test-5 inputs for each service are included in Table 3.1.

Table 3.1: Test-5 Inputs

Name	Duration	Min Blocks	Max Blocks	Compatibility	Min Coverage	Max Coverage	Max Level-1	Min Level-3	Level-1 PreReq	isHard	isPreference
S1	full	4	4	S1	16	18	5	3	-	TRUE	-
S2	full	2	2	S2	7	9	5	3	-	TRUE	-
S3	full	2	2	S3	7	9	5	3	-	-	-
S4	full	2	2	S4	7	9	5	3	S3	-	-
S5	half	2	2	S5,S6,S7,S8	0	100	100	0	-	-	TRUE
S6	half	2	2	S5,S6	7	9	5	3	-	-	-
S7	half	2	2	S5,S7	7	9	5	3	-	-	-
S8	half	7	7	S5,S8,S9	28	30	8	3	-	-	-
S9	half	1	1	S8, S9	3	5	5	2	-	-	-

For Test-5, the three full months of "hard" services can not be assigned to a resident in three consecutive months. With nine services, there are $9 * 9 = 81$ *possible* patterns. However, since only five of the nine services in Test-5 can be split into half-month blocks, there are a total of $(5 * 5) + 4 = 29$ possible monthly patterns. Furthermore, based on the compatibility of half-month services with one another, only 16 patterns are allowed.

To test the effect of compatibility restrictions on the two models, for Test-6 we remove all compatibility restrictions from Test-5. Thus, every half-month service is allowed to occur in the same month as any other half-month service. As a result, we are able to remove constraints from the Service Model, but must add decision variables to the Pattern Model to account for the additional allowable patterns.

We compare the performance of the Pattern Model to the Service Model by randomly generating ten instances of preferences for each case and then solving each instance with both models. We report the solve times for each case and model in Figure 3.2.

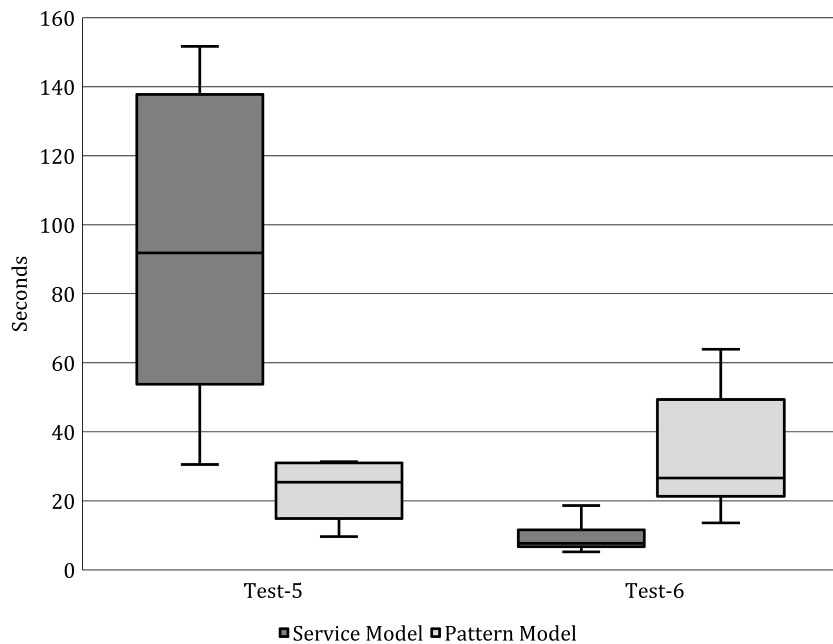


Figure 3.2: Box & Whisker Plot of Solve Times for Variation Two

For Test-5, the median solve time of the Service Model was 66 seconds (nearly 4x) greater than that of the Pattern Model. For Test-6, with the compatibility restrictions removed, we observe that there is no longer a computational advantage to using the Pattern Model over the Service Model. By comparing the solve times of the models instance by instance, the Pattern Model out-performs the Service Model for every instance in Test-5, but under-performs the Service Model for every instance in Test-6. In summary, we observed that as the number of compatibility restrictions increases for a particular scheduling prob-

lem, the computational advantage of the Pattern Model over the Service Model increased.

3.6 Variation Three

In Section 3.5 we considered multi-block services (i.e., services that occurred for consecutive blocks of time) that were only allowed to start during specific blocks. Specifically, full-month services corresponded with the calender months and therefore were only allowed to start at the beginning of a calendar month. However, in some resident programs at UMHS, multi-block services are less restricted in terms of when they are allowed to start. For example, in some programs residents are assigned to half-month, full-month, and two-month services, and the two-month services are allowed to start at the beginning of any calendar month. Thus, it is possible for one resident to be assigned to a two-month service for the first two months of the year while another resident is assigned to the same two month service for the second and third months of the year. It is not possible to model this problem variation using the Service Model without defining additional variables to represent when services start. To model this variation using the Pattern Model, it is necessary to define every allowable full-year pattern (i.e, every possible year-long schedule). Even for rather small scheduling problems, doing so is not practical and typically leads to an intractable model. In this Section, we propose two alternative models that build on our previously proposed models and then compare their performance through computational testing.

For the models we propose in this section, we define a new set of variables which we call *activity variables*. Specifically, we define variables of type z_{rab} to indicate if resident r starts activity a at the beginning of block b. For the first model we propose in this section, which is an expanded version of the Service Model, an activity is defined to be a single service for a specific duration of time (in blocks). For example, one activity could represent an assignment to general medicine for six consecutive blocks while another activity could represent an assignment to vacation for one block. We call activities that represent a single

service *simple activities*.

To model the problem variation described in this section, we link the values of the x_{rsb} variables in the Service Model to the values of the new z_{rab} variables with constraints. Consequently, deciding when each activity starts for each resident (i.e., the value of the z_{rab} variables) determines which service is assigned to each resident for each block (i.e., the value of the x_{rsb} variables). Next, we provide the full mathematical formulation of our alternative model with activity variables, which we call the *Simple-Activity Model*.

3.6.1 Simple-Activity Model Formulation

Additional Sets:

- A is the set of all activities.
- B_a is the set of blocks in which activity $a \in A$ is prohibited from starting.

Parameters:

- $s(a)$ is the service represented by activity $a \in A$.
- $d(a)$ is the duration (in blocks) of activity $a \in A$.
- $\underline{\omega}_{rs}$ and $\bar{\omega}_{rs}$ are the lower and upper bounds on the total number of blocks that resident $r \in R$ can be assigned to service $s \in S$, respectively.
- $\underline{\rho}_{sb}$ and $\bar{\rho}_{sb}$ are the lower and upper bounds on the total number of residents assigned to service $s \in S$ during block $b \in B$.
- $\underline{\lambda}_{lsb}$ and $\bar{\lambda}_{lsb}$ are the lower and upper bounds on the total number of level l residents assigned to service $s \in S$ during block $b \in B$.

Decision Variables:

- $z_{rab} \in \{0,1\}$ indicates if resident $r \in R$ starts activity $a \in A$ during block $b \in B$.
- $x_{rsb} \in \{0,1\}$ indicates if resident $r \in R$ is assigned service $s \in S$ for block $b \in B$.

Constraints:

$$x_{rsb} = \sum_{i=\max(1,b-d(a)+1)}^{b} \sum_{\substack{a \in A: \\ s(a)=s}} z_{rai} \qquad \forall\, r \in R, s \in S, b \in B \tag{3.20}$$

$$\sum_{r \in R} \sum_{b \in B_a} z_{rab} = 0 \qquad \forall a \in A \tag{3.21}$$

$$\sum_{s \in S} x_{rsb} = 1 \qquad \forall\, r \in R, b \in B \tag{3.22}$$

$$\underline{\omega}_{rs} \leq \sum_{b \in B} x_{rsb} \leq \bar{\omega}_{rs} \qquad \forall\, r \in R, s \in S \tag{3.23}$$

$$\underline{\rho}_{sb} \leq \sum_{r \in R} x_{rsb} \leq \bar{\rho}_{sb} \qquad \forall\, s \in S, b \in B \tag{3.24}$$

$$\underline{\lambda}_{lsb} \leq \sum_{r \in R_l} x_{rsb} \leq \bar{\lambda}_{lsb} \qquad \forall\, l \in L, s \in S, b \in B \tag{3.25}$$

$$x_{rs_2^e b} = 0 \qquad \forall\, e \in E, r \in R_e, b \in \{1\} \tag{3.26}$$

$$\sum_{i=1}^{b-1} x_{rs_1^e i} \geq x_{rs_2^e b} \qquad \forall\, e \in E, r \in R_e, b \in \{2, \dots, |B|\} \tag{3.27}$$

$$\sum_{s \in S_c} \sum_{i=b}^{b+\delta_c-1} x_{rsi} \leq \sigma_c \qquad \forall\, r \in R, c \in C, b \in \{1, \dots, |B| - \delta_c + 1\} \tag{3.28}$$

Here, the only differences from the Service Model are the addition of constraint sets (3.20) and (3.21). Constraint set (3.20) links x_{rsb} variables to the z_{rab} variables and (3.21) restricts when each activity is allowed to start. The remainder of constraints are the same as the Service Model and are explained in Section 3.4.1.

Although our Simple-Activity Model can be used to solve a wide-variety of block scheduling problems, for problem variations similar to those presented in Section 3.5, there

can be computational advantages to defining activities to represent service patterns, such as those used in the Pattern Model. We elaborate on how this can be done in the following section.

3.6.2 Complex-Activity Model Formulation

In Section 3.5.2 we showed that for problem variations in which relationships exist across services within intervals of time (e.g., calendar months), there can be advantages to defining patterns of services as we did for the Pattern Model. Based on our findings in Section 3.5.2, we can extend the idea from 3.6.1 trivially to recognize that each block of an activity need not be the same service, giving us greater flexibility.

For the last model we propose in this chapter, we define *complex activities*. Each complex activity represents an allowable service pattern (possibly a single service) and a duration of time (in blocks). By only defining allowable complex patterns, compatibility restrictions such as those explained in Section 3.5 can be accounted for without needing additional constraints. For example, consider a specific problem variation involving half-month, full-month, and two-month services for which many compatibility restrictions limit the combinations of services occurring within a calendar month. To model this problem variation, instead of defining activities to represent the half-month, full-month, and two-month services, activities could be defined to represent the allowable full-month patterns and the two-month services.

To formulate a new model using complex activities, we simply modify constraint set (3.20) in the Simple-Activity Model to account for the fact that activities may represent more than one service. Specifically, we first define the parameter $\mu_n^{sa} \in \{0, 1\}$ to equal 1 if service s occurs during the nth block of activity $a \in A$, and equals 0 otherwise. Then, we

replace (3.20) in the Simple-Activity Model with the following constraint set:

$$x_{rsb} = \sum_{a \in A} \sum_{i=\max(1, b-d(a)+1)}^{b} \mu^{sa}_{(b-i+1)} z_{rai} \qquad \forall\, r \in R, s \in S, b \in B \tag{3.29}$$

Here, constraint set (3.29) links x_{rsb} variables to the z_{rab} variables. We call this new model the *Complex-Activity Model*.

3.6.3 Computational Testing

To compare the performance of the Simple-Activity Model to the Complex-Activity Model, we make slight modifications to the Test-5 and Test-6 data sets of Section 3.5.2 to create Test-7 and Test-8. Specifically, we modify the attributes of service S1 in Test-5 and Test-6 by changing it from a full-month service to a two-month service while still allowing it to start at the beginning of any calendar month. Thus, for Test-7 and Test-8, S1 can start during the first block of any month and must be assigned to each resident for four consecutive half-month blocks. As discussed previously, modeling this problem variation with the Service Model is not possible without defining additional variables. Modeling it with the Pattern Model requires defining every allowable full-year pattern. For Test-7 and Test-8, this means defining 11,543,176 and 483,736,625 patterns, respectively. Defining this many patterns leads to an intractable model.

For computational testing, we solve a set of ten problem instances for Test-7 and Test-8 with each model. In the Simple-Service Model, each activity represents a single service. Thus, there are a total of nine activities in the Simple-Activity Model for Test-7 and Test-8. In the Complex-Activity Model, activities represent allowable full-month patterns or two-month services. Therefore, in the Complex-Activity Model, there are 16 allowable activities for Test-7 and 29 allowable activities for Test-8. In Figure 3.3, we present the solve times from using the Simple-Activity Model and Complex-Activity Model to solve Test-7 and Test-8 problem instances.

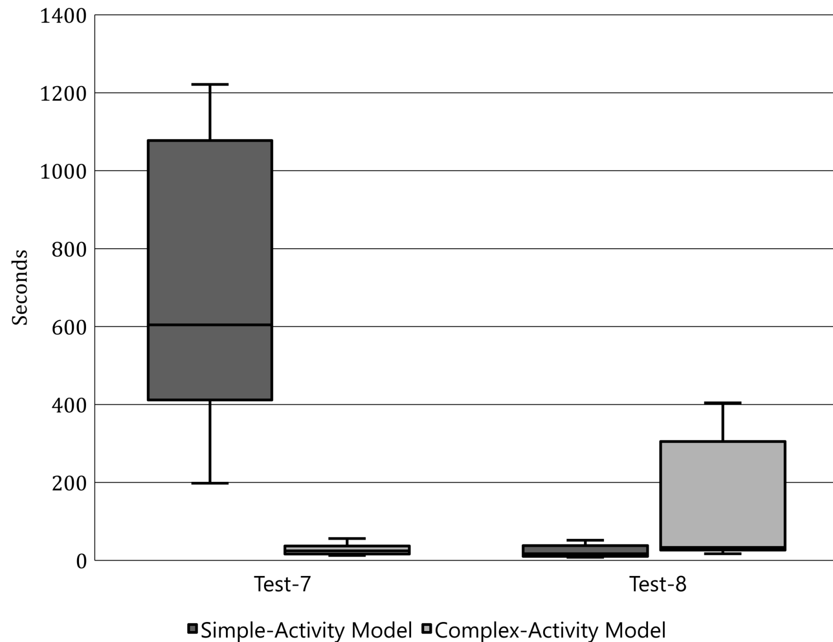


Figure 3.3: Box & Whisker Plot of Solve Times for Variation Three

For Test-7 we notice there is a significant difference in solve times between the two models—the median solve time of the Simple-Activity Model is approximately ten minutes, while the median solve time of the Complex-Activity Model is 25 seconds. This is much less than the 67 seconds difference between the median solve times of the Service Model and Pattern Model in Test-5. Thus, for the data sets tested, adding activity variables to the models had much less of an effect when activities were linked to patterns instead of individual services. We expect that the computational effects of the service compatibility restrictions were compounded by the addition of activity variables. We acknowledge that Test-7 only included one service lasting longer than one month and expect solve times to increase for both models as more multi-block services such as S1 are included.

For Test-8, in which the compatibility restrictions are removed from Test-7, the Simple-Activity Model performed better than the Complex-Activity Model. This result is similar to that of Test-6—without service compatibility restrictions, there is no advantage to defining patterns of services.

In summary, the use of activity variables enables solving more complex problem varia-

tions. Depending on the specific compatibility restrictions in place for a particular problem, there can be computational advantages of using complex activities over simple activities.

3.7 Summary

In this chapter, we presented a generalized block scheduling problem and then demonstrated how alternative variable definitions can be used to enable or improve the computational performance of mathematical models in specific situations. We explained three different scheduling problem variations, proposed mathematical models for solving each problem, and assessed the performance of each model through computational testing.

First, we introduced a rather basic block scheduling problem in which all services were identical in duration. To solve it, we proposed a model with decision variables representing a single service for a single block of time. In the second problem variation, compatibility restrictions limited the combinations of services that could occur within a particular time interval. For solving this variation, we proposed defining decision variables such that each variable represented the assignment of a service-pattern for a single interval of time. In the third problem variation, we introduced multi-block services with less restricted start times. For solving it, we first proposed defining "simple activities" to represent the assignment of a single service for some number of consecutive blocks. Since there can be advantages to using service-patterns for solving certain problem variations, we formulated a fourth model by defining variables to represent "complex activities"—a combination of service patterns for some number of consecutive time intervals.

Through computational testing, we demonstrated that there can be computational advantages to using complex decision variables for solving specific problem variations However, we also showed that no single model was best for every problem variation. Our findings reinforce the computational benefits of formulating the "right" model for the specific problem being solved.

CHAPTER 4

Scheduling Medical Residents With Conflicting Requests For Time Off

4.1 Introduction

Scheduling medical residents involves satisfying many unique and complex scheduling requirements. These *hard constraints* include Accreditation Council for Graduate Medical Education (ACGME) work-hour restrictions along with hospital- and program-specific work and educational requirements. Simply creating a schedule that satisfies all of these hard constraints can be both challenging and time-consuming. Therefore, when manually creating a schedule, as is often done by chief residents, the primary focus is on finding a feasible schedule. The resulting schedule often fails to also satisfy many of the scheduling preferences, or *soft constraints*, such as requests for time off.

Computerized decision support tools, based on underlying approaches such as integer programming, not only greatly reduce the time needed to build a schedule, but may dramatically improve the quality of the schedule as well. However, defining an objective function that precisely represents the preferences of the scheduler can be difficult. When scheduling residents, it is desirable to satisfy personal requests, but simply maximizing the number of satisfied requests may not be appropriate. For example, it might be better to grant one resident's request for their family member's wedding in place of two residents' requests to attend a rugby game. As an alternative to maximizing the number of satisfied scheduling

requests, each request could be weighted according to its importance a priori, but determining weights that accurately represent the schedulers' preferences and result in the most preferred schedule can be challenging.

To eliminate the challenge of accurately defining an objective function when using integer programming, we propose identifying the complete collection of *maximally-feasible* and *minimally-infeasible* sets of time-off requests. Here, a set is maximally feasible if it is possible to grant all requests in the set but adding any additional request to the set will make the resulting set infeasible (i.e., it is not possible to grant any additional requests). Similarly, a set is minimally infeasible if it is not possible to simultaneously grant all requests in the set, but removing any one request from the set will make the remaining set feasible (i.e., it is possible to grant all requests in any proper subset of the set). The collection of maximally-feasible and minimally-infeasible sets of requests can then be used by the scheduler to make trade-offs in deciding which resident requests to grant.

The remainder of this chapter is organized as follows. In Section 3.2 we review existing literature on healthcare personnel scheduling and finding maximally-feasible and minimally-infeasible sets. In Section 4.3 we describe the specific resident scheduling problem that we are considering. In Section 4.4 we present the two Request Selection Via Cuts (RSVC) algorithms and provide computational results in Section 4.5. In Section 4.6, we present our findings from a scheduling case-study conducted at Mott Children's Hospital. We conclude in Section 4.7 by summarizing our findings and providing suggestions for future work.

4.2 Literature Review

4.2.1 Healthcare Personnel Scheduling

Given the prevalence and complexity of scheduling problems in healthcare, the potential cost savings of efficient scheduling, and the ability to improve provider morale and patient

safety with high-quality schedules, scheduling in healthcare has received significant attention from the research community including a recently published handbook (Hall, 2012).

The majority of research in healthcare personnel scheduling focuses on nurse scheduling. The nurse scheduling problem (NSP) involves assigning nurses to shifts and work days under various hard and soft constraints such as government regulations, hospital-specific rules, and individual nurse preferences such as their vacation requests. Satisfying preferences improves nurse satisfaction and is especially important because it affects retention, a critical issue faced by many hospitals (Hayes et al., 2006; Li & Jones, 2013).

The NSP first appeared in the literature in the 1960's (Wolfe & Young, 1965). Since then, many different models and solution techniques have been proposed for addressing a variety of specific scheduling rules and objectives. A common objective of nurse scheduling problems is to satisfy nurse scheduling preferences (Warner, 1976; de Grano et al., 2009). Numerous other models and solution approaches have also been proposed in the literature (Cheang et al., 2003; Burke et al., 2004).

Medical residents are licensed physicians who are still receiving additional hands-on training under the supervision of more experienced providers. Because residents rotate between many different medical services, often as frequently as on a monthly basis, and because their schedules must not only ensure coverage for adequate patient care (similar to nurses) but must also ensure adequate training opportunities, resident scheduling problems can be particularly challenging (Guo et al., 2014).

One important resident scheduling problem is block/rotation scheduling (i.e., scheduling residents to different services for each month of the year). Block schedules must satisfy coverage needs of the system in addition to individual training requirements in order to fulfill each resident's educational needs (Smalley & Keskinocak, 2016b; Bard et al., 2016; Agarwal, 2016).

Another resident scheduling problem that is closely related to nurse scheduling is that of assigning residents to shifts, frequently in emergency departments or to cover call sched-

ules. Sherali et al. (2002) develops a mixed integer program and heuristic solution procedures for assigning residents to night shifts while considering staffing needs, skill requirements, and resident preferences. Bard et al. (2013) presents an integer goal program and three-phase solution approach for creating monthly schedules that minimize violations of a prioritized set of goals. Güler et al. (2013) uses a goal programming model with a weighted objective function in order to assign the residents to shifts in an anesthesia and reanimation department. Topaloglu & Ozkarahan (2011) and Topaloglu (2006, 2009) discuss other multi-objective resident shift scheduling models for emergency medicine residents.

Ovchinnikov & Milner (2008) acknowledge some of the challenges of using a multi-objective function and instead set targets for each of the schedule's metrics and attempt to find a feasible schedule that satisfies their targets. However, there are two downsides to this approach: 1) a feasible solution may not exist (in this case, the targets will need to be adjusted); 2) solutions may not be Pareto-optimal (i.e., it may be possible to improve a metric without negatively affecting any other metrics).

As another alternative to using a weighted objective function for a multi-objective problem, Cohn et al. (2009) proposes an iterative approach in which chief residents provide feedback on solutions generated by the model until an improved schedule cannot be found.

Like much of the referenced work, we address a multi-objective resident shift scheduling problem that includes many scheduling rules and requirements. However, our approach for solving this problem is unlike previous work that generates a single feasible schedule by either optimizing a weighted objective function or satisfying a set of targets for each metric. Instead, for a set of time-off requests (i.e., soft constraints), we present an algorithm that identifies every maximally-feasible set of time-off requests.

Maximally-feasible sets are useful since they indicate combinations of requests that can be granted simultaneously and are maximal in size. With this information, decision makers can simply decide which maximally-feasible combination of requests they prefer most. Since some problems have many such sets, making it challenging for decision mak-

ers to pick their most preferred, we extend our algorithm to also identify every minimally-infeasible set of time off requests (i.e., sets of requests that are incompatible with one another and are minimal in size). By identifying every minimally-infeasible sets of requests, each set can be "repaired" by removing any one of its requests from the scheduling problem in order to generate a feasible schedule.

4.2.2 Generating Maximally-Feasible and Minimally-Infeasible Sets

Although the generation of maximally-feasible and/or minimally-infeasible sets of constraints has been studied for other purposes, much of the previous work has focused on identifying a *single* maximally-feasible or minimally-infeasible set of constraints. The motivation for this comes from the desire to determine the cause of infeasiblilty in systems of constraints, such as those used in mathematical programs. Chinneck (2007) covers a wide variety of methods related to analyzing infeasible systems and references many of the works that have made contributions to the area, including Van Loon (1981), Amaldi et al. (1999), Chakravarti (1994), and Guieu & Chinneck (1999). Currently, the commercial solver software IBM ILOG CPLEX Optimization Studio and Gurobi Optimizer both have built-in functionality for identifying a single minimally-infeasible set of constraints, also referred to as an irreducible inconsistent set (IIS).

For a given minimally-infeasible set of constraints, it is possible to repair the set by removing one of the constraints (in our case, this is equivalent to choosing a time-off request to deny). If the revised problem were then evaluated again, a new minimally-infeasible set could be found and the process repeated until the overall problem was feasible. However, by repairing minimally-infeasible sets one at a time, it is possible to unnecessarily remove some constraints from the problem, for example, if one fails to notice that some constraints appear in multiple minimally-infeasible sets. For resident scheduling, this could mean denying requests that do not need to be denied. Therefore, it is beneficial to identify many (or all) minimally-infeasible request sets before choosing to deny any individual requests.

Unlike the previously proposed methods for generating maximally-feasible or minimally-infeasible sets, our method identifies *every* maximally-feasible and minimally-infeasible set for a set of constraints. For identifying maximally-feasible sets, our method is most similar to that of Cohn & Barnhart (2003) who use optimization to identify "unique and maximal maintenance-feasible short connects" for an aircraft maintenance routing problem. For generating minimally-infeasible sets, we leverage the relationship between maximally-feasible and minimally-infeasible sets presented by Bailey & Stuckey (2005): given the complete set of maximally-feasible sets of constraints, any set of constraints that is not a subset of any maximally-feasible set is an infeasible set. Therefore, the smallest-cardinality set of constraints that is not a subset of any maximally-feasible set is a minimally-infeasible set. Instead of using a heuristic to identify such minimal sets as Bailey & Stuckey (2005) do, we formulate and solve a mathematical optimization problem.

4.3 Resident Scheduling Problem

Although our work is generally applicable to any problem with soft constraints, the motivation for our research is assigning residents to shifts to cover the Pediatric Emergency Department at C.S. Mott Children's Hospital in the University of Michigan's Health System and addressing their potentially conflicting personal requests. This problem, like most residency scheduling problems, has a large number of requirements (i.e., hard constraints). Many of the work-hour related rules are governed by the Accreditation Council for Graduate Medical Education (ACGME). In addition to these rules, there are scheduling requirements that are particular to the hospital and the specific resident program. For example, at Mott Children's Hospital, first-year residents are not allowed to work the first or last shift of each day. For the sake of exposition, we will focus on a simplified version of the real-world problem in which we incorporate the primary hard constraints.

4.3.1 Description of Residency

Following medical school, doctors typically spend three to five years as residents — licensed, practicing physicians who work under the supervision of attending physicians. During residency, physicians rotate through various programs in order to fulfill their educational requirements and get experience in a variety of areas related to their specialties. During each rotation, residents are assigned to work shifts in the hospital according to the requirements of their current program. In addition to working shifts, residents are often required to hold clinic hours each week. Residents may also have additional time commitments related to their particular program, such as mandatory seminars.

4.3.2 Schedule Requirements

For the problem being considered here, residents who have been assigned to spend the current month staffing the pediatric emergency department must be assigned to specific shifts. Every day includes seven shifts, each of which lasts for nine hours. Shifts start at 7am, 9am, 12pm, 4pm, 5pm, 8pm, and 11pm. The shifts starting at 8pm and 11pm are considered "night" shifts. The following rules must be satisfied by a schedule:

- Each shift must be worked by exactly one resident.
- First-year residents are not allowed to work the 7am or 11pm shift on any day.
- The number of shifts worked by each resident during each month must be within a specified range.
- The number of night shifts worked by each resident during each month must be within a specified range.
- Each resident is restricted to working no more than five consecutive days in a row. A day is counted as being worked if a shift starts on that day. For example, if a resident works the 11pm shift starting on day 2 and no shifts starting on day 3, this corresponds to working day 2, but not day 3.

- Each resident is restricted to working no more than four consecutive nights in a row. Working consecutive night shifts is defined as starting night shifts on consecutive days.
- Each resident is required to have at least ten hours of rest between two consecutive work shifts.
- In addition to working shifts in the emergency room, some residents are required to work in the *continuity clinic* one day per week, from 8am to 12pm. The specific day of week (if any) that each resident needs to hold clinic hours remains constant throughout his or her residency and is determined for each resident before shift schedules are created. When a resident works in the continuity clinic, this resident cannot work any shifts that start after the 4pm shift on the previous day or before the 8pm shift on the day of the clinic.

4.3.3 Time-Off Requests

Before each month begins, residents submit requests for days off. It is desirable to grant every request for time-off, but it is often not possible do so. We begin by describing how to find a schedule that satisfies every scheduling rule and grants the maximum number of requests. Then in Section 4.4, we show how this process can be used as the kernel for generating maximally-feasible and minimally-infeasible request sets.

For the problem we consider, each request is for a single day-off and there is no limit on the number of requests each resident can submit. We acknowledge that residents may also make requests for multiple, consecutive days off in practice, but for simplicity of exposition, we only consider single-day requests; the approach can easily be extended to accommodate multi-day requests.

If a resident is granted one day-off request, that resident will not be assigned to work any shift starting after the 12pm shift on the day before the request or before the 7am shift on the day following the request. This means that this resident will finish working by 9pm

the day before the requested day-off and will not start working until 7am, at the earliest, on the day following the requested day-off.

Given a set of requests for time-off and the scheduling rules described in Section 4.3.2, we formulate and solve a mathematical optimization problem to find a schedule that satisfies every rule and grants a maximum number of requests. We include the full formulation of the problem in Section 4.3.4.

4.3.4 Resident Scheduling Problem Formulation

In this Section, we present the mathematical formulation for the residency scheduling problem previously described that we use for the computational testing in Section 4.5 and case study in Section 4.6. We solve this mathematical optimization problem to find a feasible schedule that grants a maximum number of requests.

Sets:

- P is the set of all residents (physicians).
- $P^I \subseteq P$ is the set first-year ("intern") residents. First-year residents have special work restrictions.
- S is the set of all shifts. For convenience, shifts are numbered 1 through 7, with the 7am shift being shift 1.
- $S^N \subseteq S$ is the set of night shifts.
- $S^I \subseteq S$ is the set of shifts that cannot be worked by first-year residents.
- D is the set of days in the planning horizon.
- $B = S \times D$ is the set of all shift/day pairs in the planning horizon. For example (1,2) represents shift 1 on day 2.
- $T^{s,d} \subseteq B$ is the set of shift/day pairs that start within ten hours of the end of shift $s \in S$ on day $d \in D$. For example, $T^{4,d} := \{(5,d),(6,d),(7,d),(1,d+1),(2,d+1)\}$.

- $C^p \subseteq B$ is the set of shift/day pairs that cannot be worked by resident p due to his or her continuity clinic day. For residents that do not work in the continuity clinic, $C^p = \emptyset$.
- R is the set of time-off requests. Specification of each request $r \in R$ consists of the associated resident, $p^r \subseteq P$, and a set, $B^r \subseteq B$, that contains the shift/day pairs that are requested off. For example, if a resident requests day 2 off, the shift/day pairs for this request are:
  $B^r = \{(4,1),(5,1),(6,1),(7,1),(1,2),(2,2),(3,2),(4,2),(5,2),(6,2),(7,2)\}$.

Parameters:

- $\mathrm{D}^{\max}$ is the maximum number of consecutive days that can be worked by a resident.
- $\mathrm{N}^{\max}$ is the maximum number of consecutive night shifts that can be worked by a resident.
- $\mathrm{MinShifts}^p$ and $\mathrm{MaxShifts}^p$ are the minimum and maximum number of total shifts that can be worked by resident $p \in P$, respectively.
- $\mathrm{MinNightShifts}^p$ and $\mathrm{MaxNightShifts}^p$ are the minimum and maximum number of total night shifts that can be worked by resident $p \in P$, respectively.

Decision Variables:

- $y_{psd} \in \{0,1\}$ is a binary variable that specifies whether resident $p \in P$ is assigned shift $s \in S$ on day $d \in D$
- $x_r \in \{0,1\}$ is a binary variable that specifies whether time-off request $r \in R$ is granted.

Constraints:

- **Shift Coverage:** Every shift must be covered by exactly one resident.

$$\sum_{p \in P} y_{psd} = 1, \qquad \forall\, s \in S, d \in D \tag{4.1}$$

- **10-hour Rest:** There must be at least 10 hours between consecutive shifts for each resident.

$$y_{psd} + \sum_{(\bar{s},\bar{d}) \in T^{sd}} y_{p\bar{s}\bar{d}} \leq 1, \qquad \forall\, p \in P, s \in S, d \in D \tag{4.2}$$

- **First-Year Resident Limitations:** First-year residents are not allowed to work the first (7am) or last (11pm) shifts on any day.

$$\sum_{p \in P^I} \sum_{s \in S^I} \sum_{d \in D} y_{psd} = 0 \tag{4.3}$$

- **Consecutive Working Days:** There is a maximum number of consecutive days that can be worked in a row (any shift).

$$\sum_{s \in S} \sum_{\bar{d}=d}^{d+\mathrm{D}^{\max}} y_{ps\bar{d}} \leq \mathrm{D}^{\max}, \qquad \forall\, p \in P, d \in \{1, 2, \ldots, |D| - \mathrm{D}^{\max}\} \tag{4.4}$$

- **Consecutive Working Nights:** There is a maximum number of consecutive nights that can be worked in a row.

$$\sum_{s \in S^N} \sum_{\bar{d}=d}^{d+N^{\max}} y_{ps\bar{d}} \leq \mathrm{N}^{\max}, \qquad \forall\, p \in P, d \in \{1, 2, \ldots, |D| - \mathrm{N}^{\max}\} \tag{4.5}$$

- **Total Shifts:** There is a minimum and maximum number of total shifts that can be

assigned to each resident.

$$\text{MinShifts}^p \leq \sum_{s \in S} \sum_{d \in D} y_{psd} \leq \text{MaxShifts}^p, \qquad \forall\, p \in P \tag{4.6}$$

- **Total Nights:** There is a minimum and maximum number of total nights that can be assigned to each resident.

$$\text{MinNightShifts}^p \leq \sum_{s \in S^N} \sum_{d \in D} y_{psd} \leq \text{MaxNightShifts}^p, \qquad \forall\, p \in P \tag{4.7}$$

- **Continuity Clinic:** A resident that works in a continuity clinic cannot work specific shifts before, during, or after the clinic.

$$\sum_{(s,d) \in C^p} y_{psd} = 0, \qquad \forall\, p \in P \tag{4.8}$$

- **Linking Shifts to Vacation Requests:** A resident who is granted a vacation request cannot work any shifts included in that vacation request. In other words, if x_r is 1 (the resident is granted the request) then all variables associated with shifts in the set B^r must be 0 for that resident.

$$y_{p^r sd} \leq (1 - x_r), \qquad \forall\, r \in R, (s,d) \in B^r \tag{4.9}$$

- **Variable Restrictions:** All y_{psd} and x_r variables may only take on values of 0 or 1.

$$y_{psd} \in \{0,1\}\ \forall p \in P,\ s \in S,\ d \in D;\ \ x_r \in \{0,1\}\ \forall r \in R \tag{4.10}$$

Objective:

The objective of this problem is to maximize the number of time-off requests granted.

$$\text{Maximize} \sum_{r \in R} x_r \tag{4.11}$$

4.4 RSVC Algorithms

As an alternative to finding just a *single* solution which maximizes the number of requests granted (without considering the relative importance of each request), we propose to instead identify *all* maximally-feasible and *all* minimally-infeasible request sets.

To generate these sets, we first present a two-stage sequential algorithm that we have entitled Sequential Request Selection Via Cuts (Sequential RSVC) which first finds all maximally-feasible request sets and then uses this information as input to find all minimally-infeasible request sets.

The ideas developed in Sequential RSVC are then used to motivate the more complex but more effective Simultaneous RSVC algorithm which alternates between maximization and minimization problems to ultimately find complete collections of both types of request sets.

4.4.1 Terminology and Notation

We will use the following terminology to describe sets of requests in an instance of the resident scheduling problem:

- **Request Set:** For a problem containing n requests, we denote the complete set of requests by $R = \{1, 2, \ldots, n\}$, where each number in the set represents a specific request.
- **Feasible Request Set:** A subset of requests $A \subseteq R$ is *feasible* if it is possible to

create a schedule that satisfies every *hard constraint* in the scheduling problem and grant every request in A.

- **Maximally-Feasible Set:** A feasible request set $A \subseteq R$ is *maximally feasible* if there exists no $r \in R \setminus A$ such that the set $A \cup \{r\}$ is feasible.
- **Infeasible Request Set:** A subset of requests $A \subseteq R$ is *infeasible* if it is not possible to create a schedule that satisfies every hard constraint in the scheduling problem and grant every request in A.
- **Minimally-Infeasible Set:** An infeasible request set $A \subseteq R$ is *minimally infeasible* if for any $r \in A$ the set $A \setminus \{r\}$ is feasible.

The following notation will be used throughout the rest of this chapter:

- $\mathbf{x} \in \{0, 1\}^n$ is a "request vector," i.e., an indicator vector of a request set such that $x_r = 1$ indicates that request r is included in the set, and $x_r = 0$ — that request r is not included in the set, for $r = 1, \ldots, n$. For example, the request set $A = \{1, 3, 4\}$ in a problem with six requests corresponds to $\mathbf{x} = \{1, 0, 1, 1, 0, 0\}$. We refer to a set of requests and the corresponding request vector interchangeably.
- If C is a set of constraints on a schedule,

$$\mathbf{X}^C = \{\mathbf{x} : \text{ there exists a schedule that grants every request in } \mathbf{x} \\ \text{and satisfies every constraint in } C\}.$$

  In other words, $\mathbf{X}^C$ is the set of all request vectors that are feasible under C.
- H is the set of hard constraints in a scheduling problem; $\mathbf{X}^H$ is the set of all request vectors that are feasible under H.

4.4.2 Sequential RSVC Algorithm

Given a set of hard constraints H, Sequential RSVC proceeds in two phases: first it finds all maximally-feasible sets and then — all minimally-infeasible sets. We include a visual representation of the algorithm in Figure 4.1 and a formal description in Section 4.4.2.3.

4.4.2.1 Phase I of Sequential RSVC: Identifying Maximally-Feasible Request Sets

Sequential RSVC begins by solving the following problem to find a maximally-feasible request set of largest cardinality:

$$(\text{NewFeas})_0 \quad \text{maximize} \quad \sum_{r \in R} x_r \tag{4.12}$$

$$\text{subject to} \quad \mathbf{x} \in \mathbf{X}^H. \tag{4.13}$$

For the residency scheduling problem we consider, $\mathbf{x} \in \mathbf{X}^H$ indicates that $\mathbf{x}$ is part of the feasible region defined by the problem's hard constraints ((4.1)–(4.10)), or, more precisely, the projection of the feasible region onto the space of x variables. Thus, solving the problem represented by (4.12) and (4.13) is equivalent to solving the problem described in Section 4.3.

If (NewFeas)$_0$ is infeasible, then it is not possible to generate a schedule that satisfies all of the hard constraints and the algorithm terminates. Otherwise, let us denote by R_0^F the set of requests satisfied by the optimal solution of (NewFeas)$_0$ returned by the solver. (Note that it may be possible to satisfy all the hard constraints, but not to grant any requests, in which case $R_0^F = \emptyset$.) The set R_0^F is maximally feasible (otherwise, a larger feasible request set would exist, yielding a larger objective value and thus contradicting optimality of the solution).

To find the next-largest maximally-feasible set (which might have the same cardinality

as R_0^F), we add the cut $\sum_{r\in R\setminus R_0^F} x_r \geq 1$ to $(\text{NewFeas})_0$ to get:

$$
\begin{aligned}
(\text{NewFeas})_1 \quad & \text{maximize} \quad \sum_{r\in R} x_r && (4.14)\\
& \text{subject to} \quad \mathbf{x} \in \mathbf{X}^H && (4.15)\\
& \sum_{r\in R\setminus R_0^F} x_r \geq 1. && (4.16)
\end{aligned}
$$

The cut $\sum_{r\in R\setminus R_0^F} x_r \geq 1$ eliminates exactly those solutions that only satisfy the requests in R_0^F or a proper subset of the requests in R_0^F. (If $R_0^F = \emptyset$, this constraint is interpreted as $\sum_{r\in R} x_r \geq 1$, and if $R_0^F = R$ — as "$0 \geq 1$.") Since any solution that only satisfies a proper subset of the requests in R_0^F is not maximally feasible, the only maximally-feasible request set that is eliminated from the feasible solution space is R_0^F. Therefore, if problem $(\text{NewFeas})_1$ is feasible, the set of requests satisfied by any of its optimal solutions is different than R_0^F and is maximally feasible.

If $(\text{NewFeas})_1$ is infeasible, the first phase of Sequential RSVC algorithm terminates. Otherwise, let R_1^F denote the set of requests satisfied by the optimal solution of $(\text{NewFeas})_1$ returned by the solver. We can add a new cut $\sum_{r\in R\setminus R_1^F} x_r \geq 1$ to $(\text{NewFeas})_1$ to get $(\text{NewFeas})_2$. $(\text{NewFeas})_2$ can then be solved to find the next-largest maximally-feasible request set.

Continuing in this manner of iteratively constructing and solving problems of the form:

$$
\begin{aligned}
(\text{NewFeas})_i \quad & \text{maximize} \quad \sum_{r\in R} x_r && (4.17)\\
& \text{subject to} \quad \mathbf{x} \in \mathbf{X}^H && (4.18)\\
& \sum_{r\in R\setminus R_k^F} x_r \geq 1 \quad \forall\, k = 0, \ldots, i-1 && (4.19)
\end{aligned}
$$

will result in identifying one new maximally-feasible set of requests R_i^F for each iteration, identified in non-increasing order of cardinality, by Theorem 1. At the first iteration when a problem $(\text{NewFeas})_i$ is infeasible, every maximally-feasible request set will have been

identified, by Theorem 2.

Theorem 1. *Let R_K^F be the set of requests that are satisfied in an optimal solution of* (NewFeas)$_K$ *after iteratively solving the sequence of problems* (NewFeas)$_k$ *for $k = 0, \dots, K$ in Phase I of Sequential RSVC. R_K^F is a maximally-feasible request set such that $R_K^F \neq R_k^F$ for any $k = 0, \dots, K-1$. Moreover, $|R_K^F| \leq |R_{K-1}^F|$, i.e., maximally-feasible requests sets are identified in the order of non-increasing cardinality.*

Proof. We use induction on K:

Induction base: Suppose $K = 0$ and let R_0^F be the set of requests satisfied in an optimal solution of (NewFeas)$_0$. Then, by construction, R_0^F is a feasible set and, because it corresponds to an optimal solution of (NewFeas)$_0$, R_0^F is maximal in size. Therefore, R_0^F is a maximally-feasible request set.

Induction step: Suppose the statement is true for some $K \geq 0$ and consider $K+1$. Every optimal solution of (NewFeas)$_{K+1}$ satisfies every hard constraint in the scheduling problem and each of the $K+1$ cuts that were created from the $K+1$ previously identified maximally-feasible request sets. Each cut of the form $\sum_{r \in R \setminus R_k^F} x_r \geq 1$ eliminates exactly those solutions that only satisfy the requests in R_k^F or a proper subset of the requests in R_k^F. Since any solution that only satisfies a proper subset of the requests in R_k^F is not maximally feasible, R_k^F is the only maximally-feasible request set that is eliminated from the feasible solution space by each cut. Therefore, if R_{K+1}^F is the set of requests satisfied in an optimal solution of (NewFeas)$_{K+1}$, R_{K+1}^F is a maximally-feasible request set for the scheduling problem such that $R_{K+1}^F \neq R_k^F$ for $k = 0, \dots, K$. Moreover, since problem (NewFeas)$_{K+1}$ is constructed by adding a cut to problem (NewFeas)$_K$, its optimal objective value cannot be better (i.e., larger), and we conclude that $|R_K^F| \geq |R_{K+1}^F|$. ∎

Theorem 2. *Suppose it is possible to satisfy the hard constraints of the scheduling problem. Then Phase I of Sequential RSVC algorithm terminates after a finite number of iterations, say, $\bar{K}$. $R_k^F,\ k = 0, \dots, \bar{K}-1$ is the exhaustive list of maximally-feasible request sets for*

the resident scheduling problem.

Proof. By Theorem 1, the solution to each problem (NewFeas)$_k$ found by the solver corresponds to a new maximally-feasible set of requests. Since there is a finite number of maximally-feasible sets, there exists $\bar{K}$ such that (NewFeas)$_{\bar{K}}$ is the first infeasible problem encountered by Phase I of the algorithm, and thus Phase I will terminate at iteration $\bar{K}$. Since cuts of the form $\sum_{r \in R \setminus R_k^F} x_r \geq 1$ eliminate exactly those solutions that only satisfy the requests in R_k^F or a subset of the requests in R_k^F, any maximally-feasible set is only eliminated from the feasible regions of subsequent problems by a cut after it has been identified. Thus, every maximally-feasible set will eventually be identified. ∎

4.4.2.2 Phase II of Sequential RSVC: Identifying Minimally-Infeasible Request Sets

Once every maximally-feasible set has been found, the Sequential RSVC algorithm uses these maximally-feasible sets to identify every minimally-infeasible set. An infeasible set is, by definition, not a subset of any feasible set and therefore not a subset of any maximally-feasible set. Thus, any infeasible set must include at least one request from the complement of each maximally-feasible set. To find the infeasible request set of the smallest cardinality we can therefore solve the following minimization problem, in which $\mathbb{F}$ is the exhaustive collection of maximally-feasible request sets identified in the first phase of the algorithm:

$$\text{(NewInfeas)}_0 \quad \text{minimize} \quad \sum_{r \in R} x_r \tag{4.20}$$

$$\text{subject to} \quad \sum_{r \in R \setminus F} x_r \geq 1 \quad \forall F \in \mathbb{F}. \tag{4.21}$$

(If $\mathbb{F} = \{\emptyset\}$, constraint (4.21) is interpreted as $\sum_{r \in R} x_r \geq 1$, and if $\mathbb{F} = \{R\}$ — as "$0 \geq 1$.") Note that we no longer consider the feasible region defined by the constraints H since $\mathbb{F}$ in (4.21) contains every maximally-feasible set and, by definition, any set of requests that is not a subset of any maximally-feasible request set is infeasible.

Let R_0^I be the set of requests corresponding to the optimal solution of (NewInfeas)$_0$

obtained by the solver. Since (4.21) is satisfied for any optimal solution and $\mathbb{F}$ in (4.21) contains every maximally-feasible request set, R_0^I is not a subset of any maximally-feasible request set, and thus it is an infeasible request set. Since R_0^I is also minimal in cardinality, R_0^I is a minimally-infeasible set of requests. Indeed, if this were not the case, a smaller infeasible request set would exist, yielding a smaller objective value and thus contradicting optimality of the solution.

The next-smallest minimally-infeasible set can be found by adding the cut $\sum_{r \in R_0^I} x_r \leq |R_0^I| - 1$ to (NewInfeas)$_0$ to get:

$$\text{(NewInfeas)}_1 \quad \text{minimize} \quad \sum_{r \in R} x_r \tag{4.22}$$

$$\text{subject to} \quad \sum_{r \in R \setminus F} x_r \geq 1 \quad \forall F \in \mathbb{F} \tag{4.23}$$

$$\sum_{r \in R_0^I} x_r \leq |R_0^I| - 1. \tag{4.24}$$

The cut $\sum_{r \in R_0^I} x_r \leq |R_0^I| - 1$ eliminates exactly those infeasible sets that contain every request in R_0^I. Since any proper superset of R_0^I is not minimally infeasible, the only minimally-infeasible request set that is eliminated from the feasible region by the cut is R_0^I. Therefore the set of requests corresponding to any optimal solution of (NewInfeas)$_1$ is different than R_0^I and is minimally infeasible.

Similarly, letting R_1^I be the set of requests corresponding to the optimal solution of (NewInfeas)$_1$ obtained by the solver, we can add a new cut of the form $\sum_{r \in R_1^I} x_r \leq |R_1^I| - 1$ to (NewInfeas)$_1$ to get (NewInfeas)$_2$. (NewInfeas)$_2$ can then be solved to find the next-smallest minimally-infeasible request set. Continuing in this manner of iteratively con-

structing and solving problems of the form:

$$
\begin{aligned}
(\text{NewInfeas})_j \quad \text{minimize} \quad & \sum_{r \in R} x_r && && (4.25)\\
\text{subject to} \quad & \sum_{r \in R \setminus F} x_r \geq 1 && \forall F \in \mathbb{F} && (4.26)\\
& \sum_{r \in R_k^I} x_r \leq |R_k^I| - 1 && \forall\, k = 0, \ldots, j-1 && (4.27)
\end{aligned}
$$

will result in identifying one new minimally-infeasible request set in each iteration, identified in non-decreasing order of cardinality, by Theorem 3. At the first iteration when a problem $(\text{NewInfeas})_j$ is infeasible, every minimally-infeasible request set will have been identified, by Theorem 4.

Theorem 3. *Let R_K^I be the set of requests corresponding to an optimal solution of* $(\text{NewInfeas})_K$ *after iteratively solving the sequence of problems* $(\text{NewInfeas})_k$ *for $k = 0 \ldots K$. R_K^I is a minimally-infeasible request set such that $R_K^I \neq R_k^I$ for $k = 0, \ldots, K-1$. Moreover, $|R_K^I| \geq |R_{K-1}^I|$, i.e., minimally-infeasible request sets are identified in the order of non-decreasing cardinality.*

Proof. We use induction on K:

Induction base: Suppose $K = 0$ and let R_0^I be the set of requests contained in an optimal solution of $(\text{NewInfeas})_0$. By construction, $R_0^I \nsubseteq F \; \forall F \in \mathbb{F}$, and since $\mathbb{F}$ contains every maximally-feasible set of requests for the scheduling problem, R_0^I is an infeasible request set. Since R_0^I is also minimal in the number of requests it contains, R_0^I is a minimally-infeasible request set for the scheduling problem.

Induction step: Suppose the statement is true for some $K \geq 0$ and consider $K+1$. Let R_{K+1}^I be the set of requests contained in the optimal solution to $(\text{NewInfeas})_{K+1}$ identified by the solver. Following the argument above, we conclude that R_{K+1}^I is an infeasible request set for the scheduling problem. Additionally, each cut of the form $\sum_{r \in R_k^I} x_r \leq |R_k^I| - 1$ for $k = 0, \ldots, K$ eliminates exactly those solutions that only contain the requests in R_k^I or

a proper superset of the requests in R_k^F. Since any solution that only contains a proper superset of the requests in R_k^I is not minimally infeasible, R_k^I is the only minimally-infeasible request set that is eliminated from the feasible solution space by each cut. Therefore, R_{K+1}^I is a minimally-infeasible request set for the scheduling problem such that $R_{K+1}^I \neq R_k^I$ for $k = 0, \dots, K$. Moreover, since problem (NewInfeas)$_{K+1}$ is constructed by adding a cut to problem (NewInfeas)$_K$, its optimal value cannot be better (i.e., smaller), we can conclude that $|R_K^F| \leq |R_{K+1}^F|$. ∎

Theorem 4. *Phase II of Sequential RSVC algorithm terminates after a finite number of iterations, say, $\bar{K}$. R_k^I for $k = 0, \dots, \bar{K} - 1$ is the exhaustive list of minimally-infeasible requests sets for the scheduling problem.*

Proof. If problem (NewInfeas)$_0$ is infeasible (which happens if all requests in R can be granted simultaneously), there are no infeasible sets and the conclusion holds trivially. In the remainder of the proof we consider the case when (NewInfeas)$_0$ is feasible.

By Theorem 3, the solution to each problem (NewInfeas)$_k$ found by the solver corresponds to a new minimally-infeasible set of requests. Since there is a finite number of minimally-infeasible sets, there exists $\bar{K}$ such that (NewInfeas)$_{\bar{K}}$ is the first infeasible problem encountered by Phase II of the algorithm, and thus Phase II will terminate at iteration $\bar{K}$. Since cuts of the form $\sum_{r \in R_k^I} x_r \leq |R_k^I| - 1$ eliminate exactly those solutions that only contain the requests in R_k^I or a superset of the requests in R_k^I, any minimally-infeasible set is only eliminated from the feasible region of subsequent problems by a cut after it has been identified. Thus, every minimally-infeasible set will be identified. ∎

In Figure 4.1, (4.27) is represented as $\sum_{r \in I} x_r \leq |I| - 1 \; \forall \; I \in \mathbb{I}$ where I is a single minimally-infeasible request set and $\mathbb{I}$ is the set of minimally-infeasible requests sets identified so far.

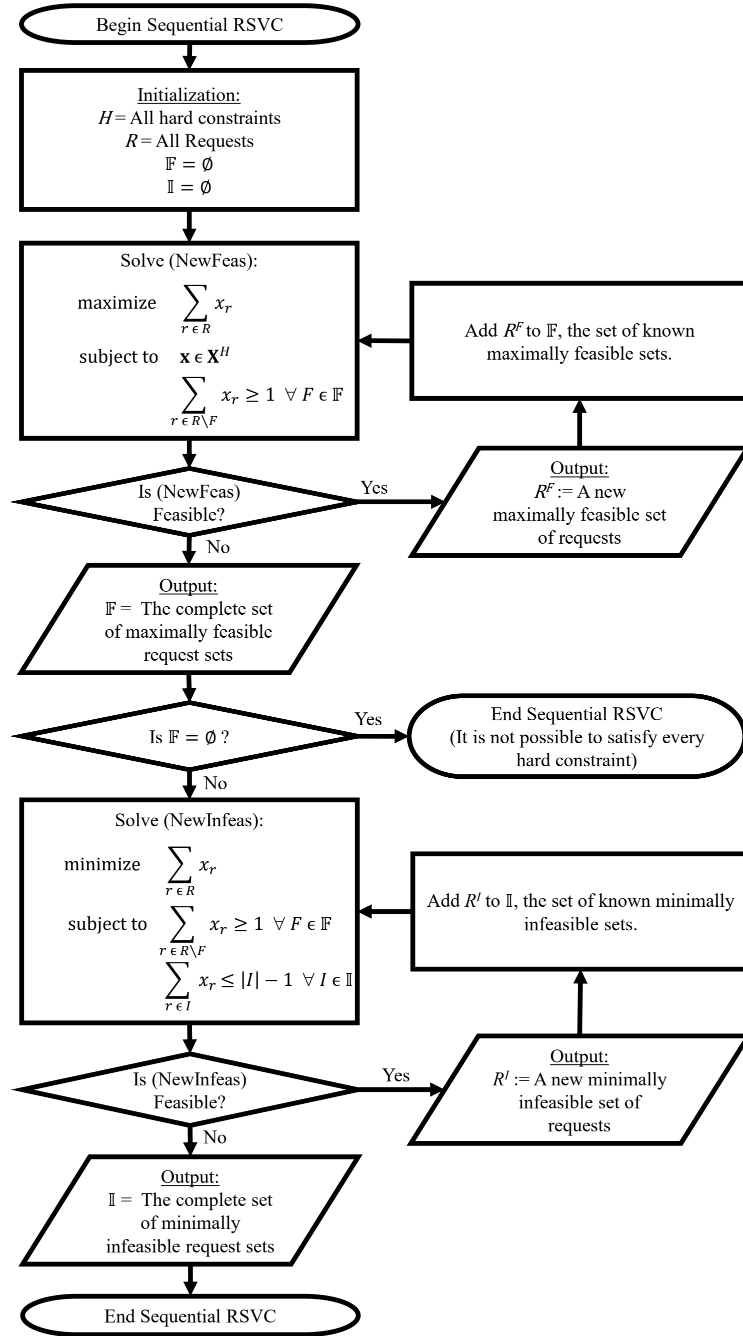


Figure 4.1: Sequential Request Selection Via Cuts

4.4.2.3 Formal Description of Sequential RSVC

Algorithm 1 Sequential RSVC Algorithm

1: Begin Initialization:
2: Set $\mathbb{F} = \emptyset$ and $\mathbb{I} = \emptyset$, where the sets $\mathbb{F}$ and $\mathbb{I}$ are used to store the identified maximally-feasible and minimally-infeasible request-sets, respectively. Let H be the set of all hard constraints for the scheduling problem and let R be the set of all requests for the problem. Set $C_I = \emptyset$ and $C_F = \emptyset$, where C_I and C_F are sets of constraints.
3: Generate an initial maximally-feasible request set by solving the problem: maximize $\sum\limits_{r \in R} x_r$ subject to $\mathbf{x} \in \mathbf{X}^H$.
4: **if** problem is infeasible **then**
5: QUIT. It is not possible to satisfy the hard constraints.
6: **else** problem is feasible **then**
7: Let R^F be the set of requests granted in the optimal solution found.
8: Add R^F to $\mathbb{F}$ (it is a maximally-feasible request set).
9: Add the cut $\sum\limits_{r \in R \setminus F} x_r \geq 1$ to C_F.
10: **end if**
11: End Initialization.
12: Solve the problem: maximize $\sum\limits_{r \in R} x_r$ subject to $\mathbf{x} \in \mathbf{X}^H$ and $\mathbf{x} \in \mathbf{X}^{C_F}$.
13: **if** a feasible solution exists **then**
14: Let R^F be the set of requests that are satisfied in the optimal solution found.
15: Add R^F to $\mathbb{F}$ (it is a maximally-feasible set).
16: Add the cut $\sum\limits_{r \in R \setminus R^F} x_r \geq 1$ to C_F
17: Goto Step 12
18: **end if**
19: **for all** $F \in \mathbb{F}$ **do**
20: Add the cut $\sum\limits_{r \in R \setminus F} x_r \geq 1$ to C_I
21: **end for**
22: Solve the problem: minimize $\sum\limits_{r \in R} x_r$ subject to $\mathbf{x} \in \mathbf{X}^{C_I}$
23: **if** a feasible solution exists **then**
24: Let R^I be the set of all requests r such that $x_r = 1$ in the optimal solution found.
25: Add R^I to $\mathbb{I}$ (it is a minimally-infeasible set).
26: Add the cut $\sum\limits_{r \in R^I} x_r \leq |R^I| - 1$ to C_I
27: Goto Step 22
28: **else**
29: End algorithm.
30: **end if**

4.4.3 Simultaneous RSVC algorithm

The Sequential RSVC algorithm first identifies the exhaustive collection of maximally-feasible sets and then uses that information to identify the exhaustive collection of minimally-infeasible sets. However, since in some problem instances the number of maximally-

feasible sets can be quite large, it may not be practical to generate every maximally-feasible set. Without the exhaustive collection of maximally-feasible sets, the Sequential RSVC algorithm is unable to identify any minimally-infeasible sets.

On the other hand, given even a small number of minimally-infeasible sets, it may be possible in some cases to find a high-quality solution by eliminating one request from each set. This motivates our development of an alternative method, which we call the Simultaneous RSVC algorithm. In this section we present the Simultaneous RSVC algorithm which has the ability to identify some (possibly all) minimally-infeasible sets without first having to identify the exhaustive collection of maximally-feasible sets. We include a visual representation of the algorithm in Figure 4.2 and a formal description in Section 4.4.3.1.

The key idea behind Simultaneous RSVC is as follows: Given (non-exhaustive) collections of known maximally-feasible and minimally-infeasible sets, we can find a new candidate request set, $R^\star$, which is neither a subset of any of the known maximally-feasible sets nor a superset of any of the known minimally-infeasible sets. Then we can "convert" $R^\star$ into either a new maximally-feasible set or a new minimally-infeasible set, depending on its feasibility status.

The algorithm maintains $\mathbb{F}$ and $\mathbb{I}$ — sets of request sets containing all maximally-feasible and minimally-infeasible sets found so far, respectively (both are initialized with an empty set). The algorithm begins by solving the now-familiar problem:

$$\text{(FirstFeas)} \quad \text{maximize} \quad \sum_{r \in R} x_r \tag{4.28}$$

$$\text{subject to} \quad \mathbf{x} \in \mathbf{X}^H. \tag{4.29}$$

If (FirstFeas) is infeasible, it is not possible to satisfy the problem's hard constraints, so the algorithm terminates. Otherwise, let R^F be the set of requests granted in the optimal solution found. R^F is a maximally-feasible request set, by Theorem 1, so we add it to $\mathbb{F}$.

At the beginning of a typical iteration of Simultaneous RSVC, $\mathbb{F}$ contains at least one

maximally-feasible set, and $\mathbb{I}$ may be empty or contain some minimally-infeasible sets. We first find a candidate set of requests, i.e., a set that is not a subset of any known feasible set and not a superset of any known infeasible set, by solving the problem:

$$\begin{aligned}
\text{(CandidateSet)} \quad \text{minimize} \quad & \sum_{r \in R} x_r && (4.30)\\
\text{subject to} \quad & \sum_{r \in R \setminus F} x_r \geq 1 \qquad \forall\, F \in \mathbb{F} && (4.31)\\
& \sum_{r \in I} x_r \leq |I| - 1 \quad \forall\, I \in \mathbb{I}. && (4.32)
\end{aligned}$$

Here, (4.31) ensures that the candidate set is not a subset of any known maximally-feasible set and (4.32) ensures that the candidate set is not a superset of any known minimally-infeasible set. Suppose (CandidateSet) is feasible, and let $R^{\star}$ be the set of requests that corresponds to the optimal solution of (CandidateSet) found by the solver. Note that feasibility status of $R^{\star}$ is unknown; thus, we next check if there exists a schedule that grants every request in $R^{\star}$ by solving the following problem:

$$\begin{aligned}
\text{(FeasTest)} \quad \text{maximize} \quad & \sum_{r \in R} x_r && (4.33)\\
\text{subject to} \quad & \mathbf{x} \in \mathbf{X}^H && (4.34)\\
& x_r = 1 \quad \forall\, r \in R^{\star}. && (4.35)
\end{aligned}$$

Here, (4.34) ensures feasibility of the schedule and (4.35) ensures that the solution grants every request in the candidate set $R^{\star}$. If (FeasTest) is infeasible, $R^{\star}$ is a new minimally-infeasible set, by Theorem 5 part (a), so we add it to $\mathbb{I}$. If (FeasTest) is feasible, let R^F be the set of requests granted in the optimal solution found. R^F is a new maximally-feasible set, by Theorem 5 part (b), so we add it to $\mathbb{F}$. Then, we add the appropriate cut to (CandidateSet) and re-solve it to identify a new candidate set.

By Theorem 6, (CandidateSet) is infeasible precisely when $\mathbb{F}$ and $\mathbb{I}$ contain every maximally-feasible and minimally-infeasible request set, respectively. By Theorem 7 this

will happen after a finite number of iterations, and the algorithm will terminate.

Theorem 5. *Suppose an instance of (CandidateSet) solved in the Simultaneous RSVC algorithm is feasible. Let $x^\star$ be the optimal solution found and let $R^\star$ be the corresponding set of requests. Furthermore, suppose the problem (FeasTest) is solved using $R^\star$ in constraint* (4.35).

(a) If (FeasTest) is infeasible, then $R^\star$ is a minimally-infeasible request set such that $R^\star \notin \mathbb{I}$.

(b) If (FeasTest) is feasible and R^F is the set of requests granted in an optimal solution of (FeasTest), then R^F is a maximally-feasible request set such that $R^F \notin \mathbb{F}$.

Proof: First, consider the case when (FeasTest) is infeasible. Then $R^\star$ is an infeasible request set, by definition. By construction, $\mathbf{x}^\star$ does not violate any constraints of the type (4.32) in (CandidateSet); therefore $R^\star \notin \mathbb{I}$

We now prove that $R^\star$ is minimally infeasible by showing that every proper subset of $R^\star$ is a feasible request set. Let $\tilde{R}$ be a set of requests such that $\tilde{R} \subset R^\star$, and let $\tilde{\mathbf{x}}$ be the corresponding request set. Since $|\tilde{R}| < |R^\star|$, $\tilde{\mathbf{x}}$ is not feasible to (CandidateSet). Indeed, if $\tilde{\mathbf{x}}$ were feasible to (CandidateSet), $\mathbf{x}^\star$ would not be an optimal solution. Therefore, $\tilde{\mathbf{x}}$ must violate at least one constraint of (CandidateSet).

Notice that $\tilde{\mathbf{x}}$ may violate constraints of type (4.31) or (4.32), but will not violate constraints of both types. Recall that if $\tilde{x}$ violates a constraint of type (4.31), this indicates that $\tilde{R}$ is a subset of some known maximally-feasible set, i.e., $\tilde{R}$ is a feasible set. Similarly, if $\tilde{x}$ violates a constraint of the type (4.32), this indicates that $\tilde{R}$ is a superset of some known minimally-infeasible set, i.e., $\tilde{R}$ is an infeasible set. Thus, there are two cases to consider:

Case 1: $\tilde{\mathbf{x}}$ violates at least one constraint of type (4.32), i.e., it violates a constraint of the form $\sum_{r \in I} x_r \leq |I| - 1$ for some minimally-infeasible set $I \in \mathbb{I}$. Therefore, $\tilde{R} \supseteq I$. However, $R^\star \supset \tilde{R} \supseteq I$. Thus, $\mathbf{x}^\star$ does not satisfy $\sum_{r \in I} x_r \leq |I| - 1$, a contradiction to the fact that $\mathbf{x}^\star$ is feasible to (CandidateSet).

Case 2: $\tilde{\mathbf{x}}$ violates at least one constraint of type (4.31). Therefore, $\tilde{R} \subseteq F$ for some $F \in \mathbb{F}$, which means that $\tilde{R}$ is a feasible request set.

We conclude that every proper subset of $R^{\star}$ is a feasible request set. Therefore, $R^{\star}$ is a minimally-infeasible request set such that $R^{\star} \notin \mathbb{I}$, establishing part (a) of the theorem.

Next, suppose (FeasTest) is feasible, and let R^F be as defined in part (b) of the theorem. Due to the structure of (FeasTest), R^F is a feasible request set, and, since it is not possible to grant any additional requests, it is a maximally-feasible request set.

If $R^F \in \mathbb{F}$, then, since $R^{\star} \subseteq R^F$, $\mathbf{x}^{\star}$ violates some constraint of the type (4.31) in (CandidateSet), which is a contradiction. Thus, R^F is a newly-identified maximally-feasible request set, establishing part (b) of the theorem. ∎

Theorem 6. *The problem (CandidateSet) is infeasible if and only if* $\mathbb{F}$ *and* $\mathbb{I}$ *contain every maximally-feasible and minimally-infeasible request set, respectively.*

Proof: Recall that every constraint of (CandidateSet) of type (4.31) removes precisely those solutions that correspond to feasible sets that are contained in some $F \in \mathbb{F}$, and every constraint of type (4.32) excludes precisely those solutions that correspond to infeasible sets that contain some $I \in \mathbb{I}$.

If $\mathbb{F}$ does not contain maximally-feasible set $\bar{F}$, in light of the above observation, request vector $\bar{x}$ that corresponds to $\bar{F}$ is a feasible solution of (CandidateSet). If $\mathbb{I}$ does not contain minimally-infeasible set $\tilde{I}$, in light of the above observation, request vector $\tilde{x}$ that corresponds to $\tilde{I}$ is a feasible solution of (CandidateSet). Thus (CandidateSet) is feasible if $\mathbb{F}$ does not include all maximally-feasible sets, or $\mathbb{I}$ does not include all minimally-infeasible sets.

Conversely, every feasible set is contained in some maximally-feasible set, and every infeasible set contains some minimally-infeasible set. Thus, if $\mathbb{F}$ and $\mathbb{I}$ contain every maximally-feasible and minimally-infeasible request set, respectively, every request vector violates some constraint of (CandidateSet). ∎

Theorem 7. *The Simultaneous RSVC algorithm terminates after a finite number of iterations. At termination,* $\mathbb{F}$ *and* $\mathbb{I}$ *contain every maximally-feasible and minimally-infeasible request set, respectively.*

Proof: If it is not possible to satisfy the hard constraints of the scheduling problem, the algorithm will terminate in its initialization phase. Otherwise, the algorithm will terminate if it encounters an infeasible instance of (CandidateSet).

By Theorem 5, each time (CandidateSet) is solved, a new maximally-feasible set is added to $\mathbb{F}$, or a new minimally-infeasible set is added to $\mathbb{I}$. Since there is a finite number of maximally-feasible and minimally-infeasible requests sets, every such set will be found after a finite number of iterations. At that point, (CandidateSet) will become infeasible, by Theorem 6, and the algorithm will terminate. ■

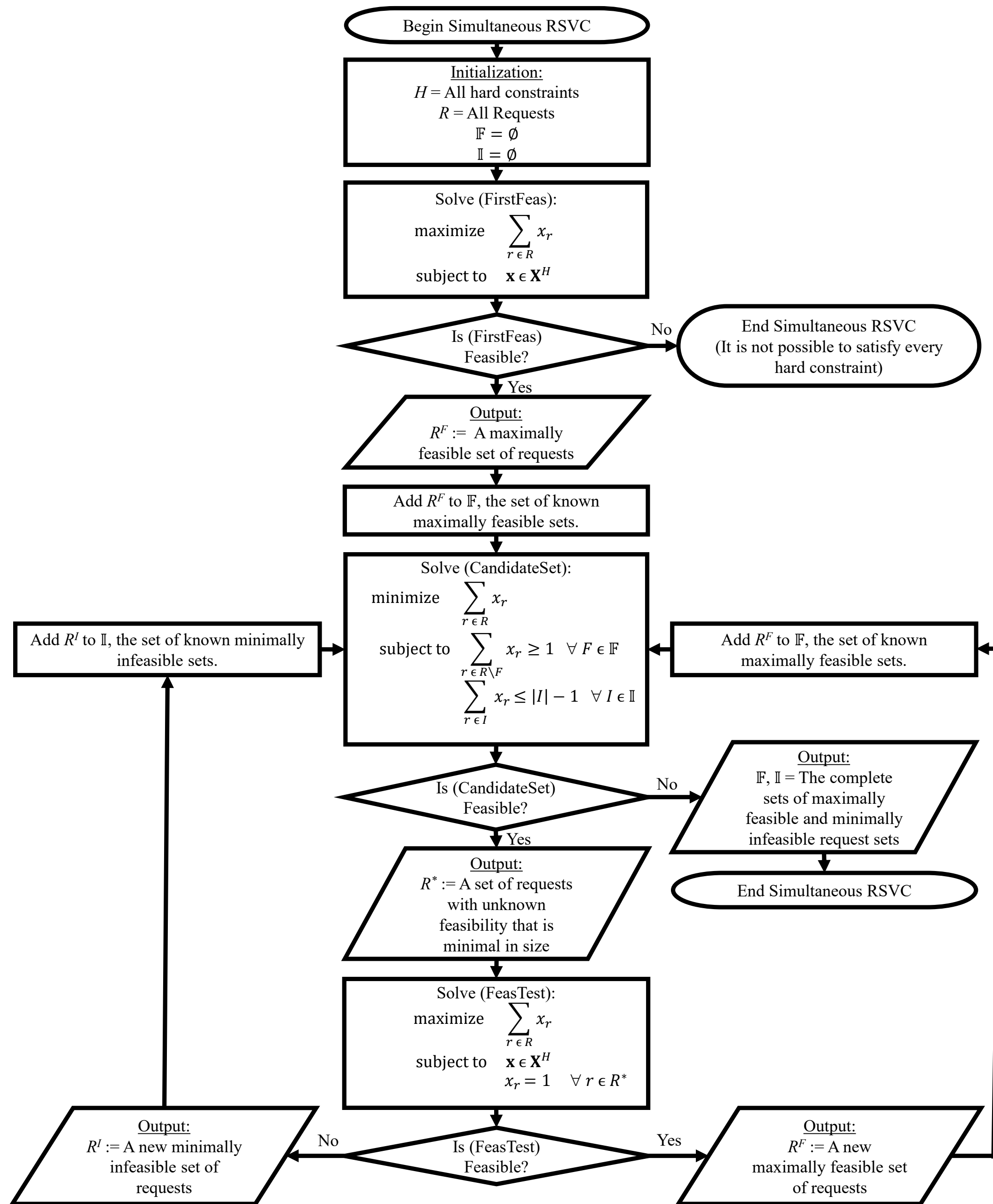


Figure 4.2: Simultaneous Request Selection Via Cuts

4.4.3.1 Formal Description of Simultaneous RSVC

Algorithm 2 Simultaneous RSVC Algorithm

1: Begin Initialization:
2: Set $\mathbb{F} = \emptyset$ and $\mathbb{I} = \emptyset$ where the sets $\mathbb{F}$ and $\mathbb{I}$ are used to store the identified maximally-feasible and minimally-infeasible request-sets, respectively. Let H be the set of all hard constraints for the scheduling problem and let R be the set of all requests for the problem. Set $C_I = \emptyset$ and $C_F = \emptyset$ where C_I and C_F are sets of constraints.
3: Generate an initial maximally-feasible request set by solving the problem: maximize $\sum_{r \in R} x_r$ subject to $\mathbf{x} \in \mathbf{X}^H$.
4: **if** problem is infeasible **then**
5: QUIT. It is not possible to satisfy the hard constraints.
6: **else** problem is feasible **then**
7: Let R^F be the set of requests granted in the optimal solution found.
8: Add R^F to $\mathbb{F}$ (it is a maximally-feasible request set).
9: Add the cut $\sum_{r \in R \setminus F} x_r \geq 1$ to C_F.
10: **end if**
11: End Initialization.
12: Solve the problem (CandidateSet):

$$\text{minimize} \sum_{r \in R} x_r \text{ subject to } \mathbf{x} \in \mathbf{X}^{C_F} \text{ and } \mathbf{x} \in \mathbf{X}^{C_I}.$$

13: **if** (CandidateSet) is infeasible **then**
14: QUIT. $\mathbb{F}$ and $\mathbb{I}$ contain every maximally-feasible and minimally-infeasible request set, respectively.
15: **else** (CandidateSet) is feasible **then**
16: Let $\mathbf{x}^\star$ be the optimal solution found and let $R^\star$ bet the set of requests in $\mathbf{x}^\star$.
17: Solve the problem (FeasTest):

$$\text{maximize} \sum_{r \in R} x_r \text{ subject to } \mathbf{x} \in \mathbf{X}^H \text{ and } x_r = 1 \,\forall\, r \in R^\star.$$

18: **if** (FeasTest) is infeasible **then**
19: Add $R^\star$ to $\mathbb{I}$ (it is a minimally-infeasible request set).
20: Add the cut $\sum_{r \in R^\star} x_r \leq |R^\star| - 1$ to C_I.
21: **else** (FeasTest) is feasible **then**
22: Let R^F be the set of requests satisfied in the optimal solution found for (MAX SET).
23: Add R^F to $\mathbb{F}$ (it is a maximally-feasible request set).
24: Add the cut $\sum_{r \in R \setminus R^F} x_r \geq 1$ to C_F.
25: **end if**
26: **end if**
27: Goto Step 12

4.5 Computational Testing

In this section we present computational experiments to address the following questions:

- **Practicality**: For real-world residency scheduling problems, how many maximally-feasible and minimally-infeasible sets exist, typically? For cases in which it is not practical to identify and evaluate every maximally-feasible request set, does the Simultaneous RSVC algorithm identify useful information for the decision maker?
- **Performance**: How long does it take to run the algorithms? Are they tractable for real-world use? How do the Sequential RSVC and Simultaneous RSVC algorithms compare in terms of run time?

To answer these questions, we apply the two algorithms to the resident scheduling problem described in Section 3. We use an Intel Xeon E3-1230 quad-core running at 3.20 GHz with hyper-threading and 32 GB of RAM. We use the IBM ILOG Optimization Studio (*CPLEX*) 12.6 C++ API software package.

4.5.1 Input Data

In order to test how variations in problem data affect the performance and output of the RSVC algorithms, we consider 24 different scheduling scenarios of varying levels of flexibility based on real-world scheduling instances within Pediatric Emergency Medicine at Mott Children's Hospital. Using these scenarios as a foundation, we randomly generate 50 problem instances for each scenario.

In every scenario, 20 residents must be scheduled for a 30-day month that starts on a Saturday. Each resident is allowed to work a maximum of five days in a row and a maximum of four nights in a row. Across the 24 scenarios, we vary the following inputs:

- **Number of total shifts and night shifts (2 variations)** There is a minimum number and a maximum number of total shifts and night shifts that each resident can work

during the month. Depending on the scenario, residents are required to work a total of 10 to 11 shifts with 3 to 4 of these as night shifts per month (i.e. a tightly-constrained schedule) or 5 to 15 total shifts with 0 to 10 as night shifts (much more loosely constrained).

- **First-year residents (2 variations)** For the resident scheduling problem we consider, first-year residents are not allowed to work the first or last shifts of the day. The first-year status of each resident is assigned randomly, with a probability of either 40% (tightly constrained) or 10% (loosely constrained) of being a first-year resident.

- **Continuity clinic days (2 variations)** Each resident has a weekly continuity clinic (or no continuity clinic at all). In the first variation, each resident has probability 1/3 each of being assigned to clinic on Mondays, Wednesdays, or Fridays (tightly constrained). In the second variation, the probability is 1/8 for each day of the week, and 1/8 that they do not get assigned to continuity clinic at all (loosely constrained).

- **Time-off requests (3 variations)** For each of the 30 days in the month, each resident has a 10% (loosely constrained), 35%, or 50% (tightly constrained) chance of requesting that specific day off, depending on the scenario. In scenarios where there is a 10% chance of requesting any particular day off, each resident will request, on average, a total of three days off during the month.

Using every combination of input variations results in 24 scenarios. As an example of a scenario, consider Scenario 1. For Scenario 1 problem instances, each resident must work five to fifteen total shifts and zero to ten night shifts. There is a 10% chance that each resident is a first-year resident, a 10% chance that each resident requests each day off, and residents may work in the clinic on any day of the week or not at all. Based on these characteristics, we then create 50 problem instances associated with Scenario 1. Table 4.1 summarizes all 24 scenarios that we use to generate problem instances for computational testing.

Table 4.1: Computational Testing Scenarios

Scenario Name	Minimum Total Shifts	Maximum Total Shifts	Minimum Total Night Shifts	Maximum Total Night Shifts	Probability of First-Year	Probability of Time-off Request	Clinic Day (equally likely)
1	5	15	0	10	10%	10%	Any/None
2	10	11	3	4	10%	10%	Any/None
3	5	15	0	10	10%	35%	Any/None
4	10	11	3	4	10%	35%	Any/None
5	5	15	0	10	10%	50%	Any/None
6	10	11	3	4	10%	50%	Any/None
7	5	15	0	10	40%	10%	Any/None
8	10	11	3	4	40%	10%	Any/None
9	5	15	0	10	40%	35%	Any/None
10	10	11	3	4	40%	35%	Any/None
11	5	15	0	10	40%	50%	Any/None
12	10	11	3	4	40%	50%	Any/None
13	5	15	0	10	10%	10%	Mon/Wed/Fri
14	10	11	3	4	10%	10%	Mon/Wed/Fri
15	5	15	0	10	10%	35%	Mon/Wed/Fri
16	10	11	3	4	10%	35%	Mon/Wed/Fri
17	5	15	0	10	10%	50%	Mon/Wed/Fri
18	10	11	3	4	10%	50%	Mon/Wed/Fri
19	5	15	0	10	40%	10%	Mon/Wed/Fri
20	10	11	3	4	40%	10%	Mon/Wed/Fri
21	5	15	0	10	40%	35%	Mon/Wed/Fri
22	10	11	3	4	40%	35%	Mon/Wed/Fri
23	5	15	0	10	40%	50%	Mon/Wed/Fri
24	10	11	3	4	40%	50%	Mon/Wed/Fri

4.5.2 Problem Instance Characteristics

For computational testing, a set of 50 random problem instances for each testing scenario in Table 4.1 was solved using both algorithms. In Figure 4.3, we report the percentage of those 50 instances in which it was possible to grant every request ("fully feasible"), the percentage of instances in which no feasible solutions existed ("infeasible"), and the percentage of "interesting" instances for each scenario. Here, "interesting" describes instances in which it is possible to satisfy all of the scheduling rules, but not possible to satisfy all of the time-off requests. These are the instances for which the RSVC algorithms are rele-

vant. For the remainder of our computational experiments, we focus on these interesting instances.

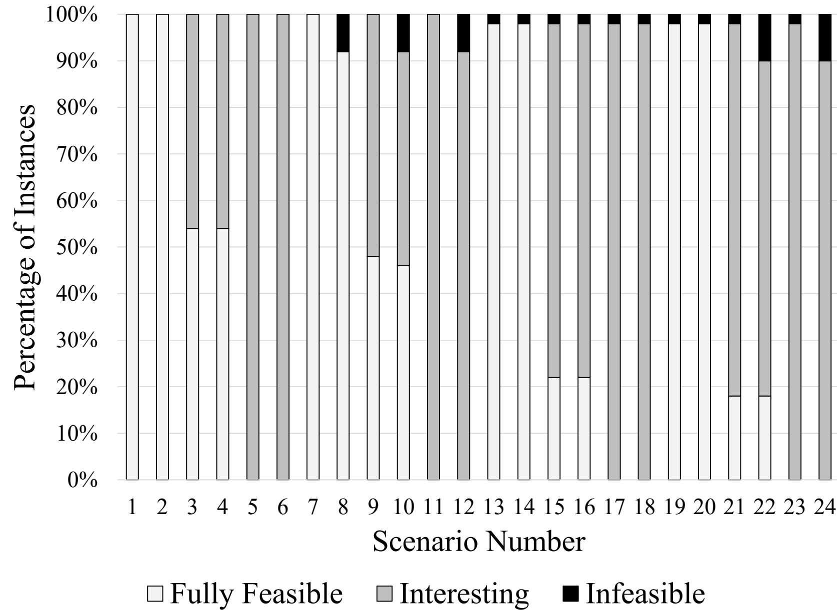


Figure 4.3: Feasibility of Problem Instances

Since a given instance may have a large number of maximally-feasible and/or minimally-infeasible request sets, we categorize every "interesting" problem instance as either Type 1 or Type 2. Type 1 instances are those with 1,000 or fewer maximally-feasible sets *and* 1,000 or fewer minimally-infeasible sets. For Type 1 instances, each algorithm is allowed to run until it identifies every maximally-feasible and every minimally-infeasible set.

Type 2 instances are the remaining "interesting" problem instances. Type 2 instances have more than 1,000 maximally-feasible sets or more than 1,000 minimally-infeasible sets.

4.5.3 Type 1 Problem Instances

In Table 4.2, we list the number of Type 1 and number of "interesting" instances (out of 50) for each of the relevant scenarios. We also list the median, minimum, and maximum numbers of maximally-feasible sets (MFSs) and minimally-infeasible sets (MISs) for Type

1 instances.

Table 4.2: Type 1 Problem Instances

Scenario Name	# Type 1/ # Interesting	# MFSs (Median)	# MFSs (Minimum)	# MFSs (Maximum)	# MISs (Median)	# MISs (Minimum)	# MISs (Maximum)
3	20/23	17	9	800	8	1	531
4	19/23	49	9	901	11	1	531
9	22/26	16.5	6	532	5	1	513
10	17/23	49	6	532	6	1	513
15	28/28	35.5	5	891	5	1	84
16	27/38	40	5	891	5	1	84
21	28/40	63	3	720	5	1	72
22	23/37	63	7	720	4	1	72

For Type 1 instances, there are generally far fewer minimally-infeasible sets than maximally-feasible sets. In these cases, it is typically most efficient for schedulers to identify their preferred schedule by working with the minimally-infeasible sets and selecting one request from each to deny. Each scenario also includes at least one instance in which there is a single minimally-infeasible set. When there is only one minimally-infeasible set, we know that only one total time-off request must be denied. Of the 184 Type 1 instances, only eight had fewer maximally-feasible sets than minimally-infeasible sets. We plot the number of maximally-feasible sets against the number of minimally-infeasible sets for Type 1 problem instances in Figure 4.4.

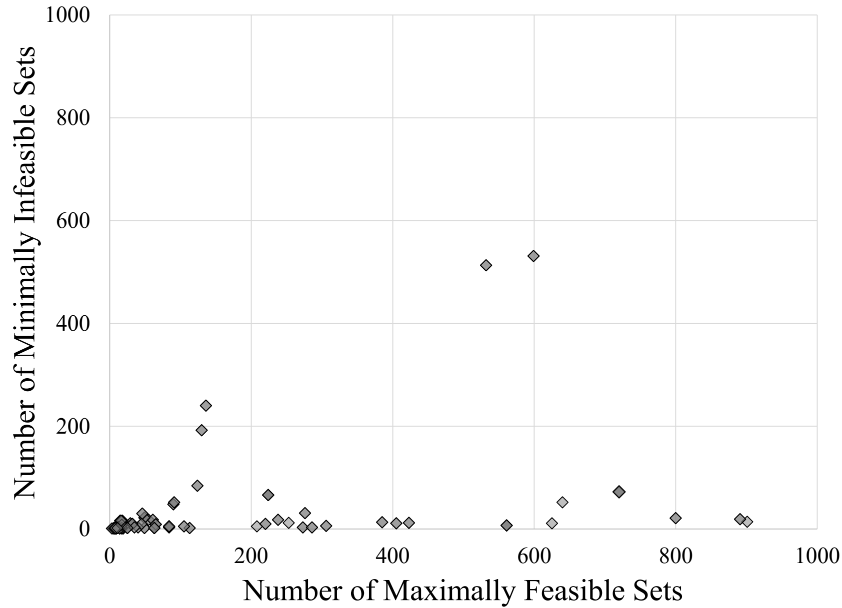
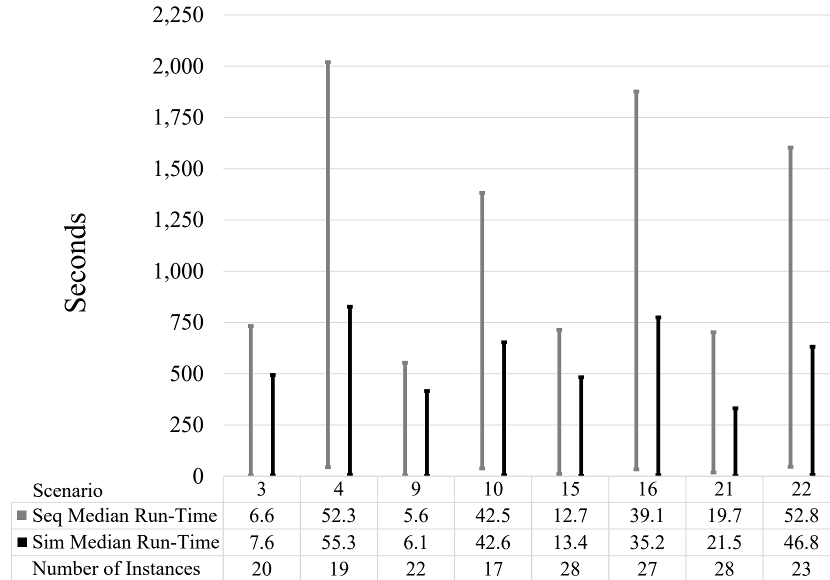


Figure 4.4: Numbers of Maximally-Feasible and Minimally-Infeasible Sets for Type 1 Problem Instances

For Type 1 problem instances, the Sequential RSVC and Simultaneous RSVC algorithms both generate the same solutions, so we can compare their run-times directly. To compare the run-times for both algorithms, we report the median and plot the minimum and maximum run-times for Type 1 problem instances in Figure 4.5. From Figure 4.5, we notice that although the median run-times of the two algorithms are similar, the maximum run-time for the Sequential RSVC algorithm is larger for each of the scenarios. Not surprisingly, we also see that the scenarios with less flexibility had longer run-times. Specifically, when residents have tighter restrictions on the number of shifts and night-shifts that must be worked, as is the case in Scenarios 4, 10, 16, and 22, it takes more time for the algorithms to run since the optimization problems take longer to solve, on average.



Scenario	3	4	9	10	15	16	21	22
Seq Median Run-Time	6.6	52.3	5.6	42.5	12.7	39.1	19.7	52.8
Sim Median Run-Time	7.6	55.3	6.1	42.6	13.4	35.2	21.5	46.8
Number of Instances	20	19	22	17	28	27	28	23

Figure 4.5: Median, Minimum, and Maximum Run-Times for Type 1 Instances

When comparing the run-times of the two algorithms across all Type 1 instances, the Simultaneous RSVC algorithm rarely takes more time than the Sequential RSVC algorithm to run and often runs significantly faster, especially for instances that take both algorithms longer than 100 seconds to solve (Figures 4.6 and 4.7 are plots of the run-times for Type 1 instances). In some cases, the Simultaneous method was up to 20 minutes faster (2x faster) than the Sequential method. Given that both methods solve similar problems and the Simultaneous approach solves two optimization problems to yield each new (feasible or infeasible) request set, whereas the Sequential approach solves only one, this might seem surprising. However, the first problem in the simultaneous approach, (CandidateSet), is a small problem that can typically be solved in a fraction of a second and the results from (CandidateSet) fix many of the decision variables in the second problem, (FeasTest). Consequently, (FeasTest) is much easier to solve than the similar maximization problem, (NewFeas), that is solved during the Sequential method.

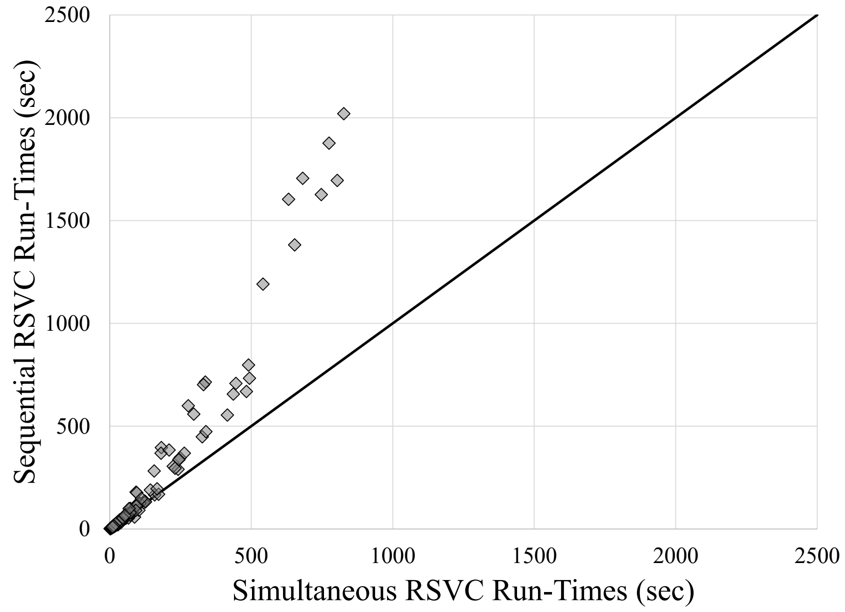


Figure 4.6: Algorithm Run-Time Comparison for Type 1 Problem Instances

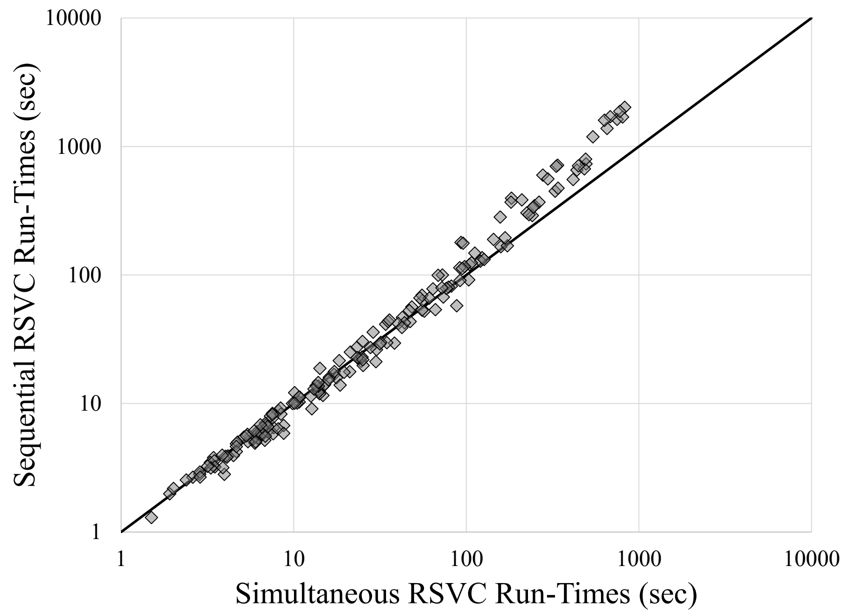


Figure 4.7: Algorithm Run-Time Comparison for Type 1 Problem Instances (Logarithmic Scaling)

4.5.4 Type 2 Problem Instances

All Type 2 problem instances include at least 1,000 maximally-feasible or at least 1,000 minimally-infeasible sets. For testing Type 2 problem instances, each algorithm is run on every instance until it finds a total of 1,000 sets, which may be maximally-feasible, minimally-infeasible, or some of each. In this section, we compare the number of maximally-feasible and minimally-infeasible sets identified by each algorithm and how long it takes each algorithm to identify the first 1,000 sets.

Of the 452 Type 2 problem instances, only two had fewer than 1,000 maximally-feasible sets and instead had more than 1,000 minimally-infeasible sets. For the other 450 instances the Sequential algorithm did not identify any minimally-infeasible sets. An advantage of the Simultaneous algorithm is that it can potentially identify some minimally-infeasible sets before identifying the exhaustive collection of maximally-feasible sets. For nearly 65% of the Type 2 problem instances, the Simultaneous algorithm identified at least one minimally-infeasible set. Identifying minimally-infeasible sets for schedulers can be useful since they indicate sets of requests that are incompatible with one another and therefore require making decisions about which requests to deny. We elaborate on a process for using minimally-infeasible sets with schedulers in Section 4.6.

When comparing the run-times of the two algorithms for Type 2 instances, we find that for many cases Simultaneous RSVC is much faster (up to 5x (2.6 hours) faster), and in the remaining cases is typically comparable. We plot the run-times for each Type 2 instance in Figures 4.8 and 4.9.

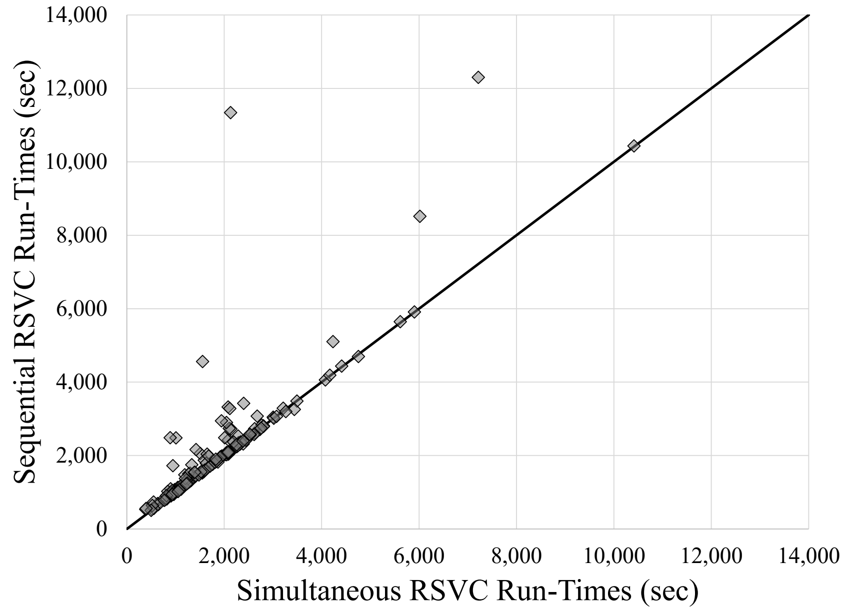


Figure 4.8: Algorithm Run-time Comparison for Type 2 Problem Instances

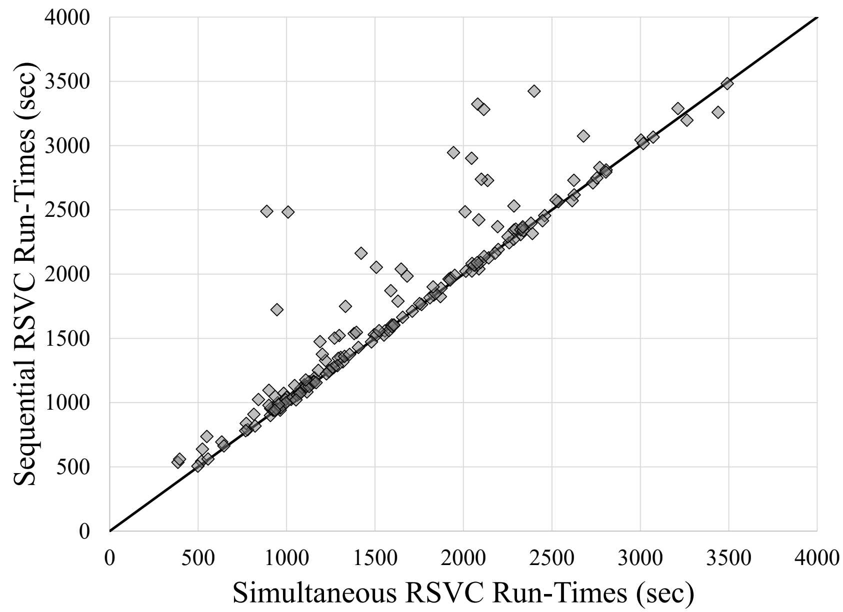


Figure 4.9: Algorithm Run-time Comparison for Type 2 Problem Instances (Zoomed)

4.5.5 Results Summary

From our testing of Type 1 instances we discovered that there are generally far fewer minimally-infeasible sets than maximally-feasible sets and that for some instances the Si-

multaneous RSVC algorithm identifies the exhaustive collection of request sets twice as fast as the Sequential RSVC algorithm. We find that the Simultaneous algorithm is also faster for Type 2 instances and unlike the Sequential algorithm, is often able to identify minimally-infeasible sets.

4.6 Case Study

To assess the usability and value of the information provided to schedulers by the RSVC algorithms, we conducted a case study at Mott Children's Hospital with a Chief Resident who is responsible for scheduling pediatric residents. For the case study, we considered several "interesting" problem instances from Section 4.5 with the Chief using two different scheduling approaches.

In the first approach, similar to what is done in practice, we first maximize the number of granted requests. Then, the Chief reviews the list of denied requests and if he feels something on that list is important to grant, a requirement is added to the scheduling problem to ensure the request is granted. Next, a solution that maximizes the number of granted requests subject to these additional requirements is generated and presented to the Chief. This process continues until the Chief is satisfied with the solution.

In the second approach, we use our RSVC algorithms to generate the maximally-feasible and minimally-infeasible request sets for the problem instance. The Chief then uses this information to select a schedule.

In the remainder of the section, we describe the Chief's experiences using each scheduling approach for a number of problem instances (cases), and discuss his feedback.

4.6.1 Case 1

For Case 1, we solved a problem instance from Scenario 8 involving 199 requests. Using the traditional approach of maximizing the number of granted requests, we discovered that

it was possible to grant all but one of the requests. However, the request that was denied was for a resident's sister's wedding. Unsatisfied with this solution, the Chief asked for an alternative solution in which the wedding request was granted. After adding this requirement to the problem and maximizing the number of granted requests, another solution that granted all but one of the requests was generated. In this solution, the request that was denied was for "a baby shower." After adding a requirement to ensure granting this request as well, the next solution required denying two requests, one for a wedding and one for "family in town." At this point, the Chief decided he was okay with the solution that only denied the baby shower request, and no additional solutions were generated using the traditional approach.

Next, as part of our proposed, alternative scheduling approach, we generated the exhaustive collections of maximally-feasible and minimally-infeasible request sets. In Figure 4.10 we visually represent every maximally-feasible request set (16 total) and a subset of the 199 requests for this problem instance using a spreadsheet tool that we created. Here, the rows represent specific requests by individuals (including the request reason) and the numbered columns represent the maximally-feasible request sets. For each maximally-feasible request set, a "D" is used to indicate a request that is denied in that set. For example, Maximally-Feasible Set #5 involves denying Request #4 from Dr. Dombrock.

				Maximally-Feasible Request Sets															
Request #	**Name**	**Reason**	**Grant?**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	Lockard	Brother's wedding				**D**	**D**			**D**			**D**						
2	Feely	Date night with spouse												**D**					
3	Peel	Just because																	
4	Dombrock	Family in town				**D**		**D**				**D**					**D**		
5	Walker	Music concert							**D**							**D**	**D**	**D**	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
197	Conyers	Trip to Chicago													**D**				
198	Walker	Medical Appointment																	
199	Hobson	Sister's wedding		**D**															

Figure 4.10: Complete Collection of Requests and Maximally-Feasible Request Sets

Although this problem includes 199 requests, many of the requests (such as Request #3) are granted in every maximally-feasible set and therefore do not need to be considered when deciding which requests to grant. By hiding all requests that are always possible to

grant, as is done in Figure 4.11, it is easier for schedulers to compare the sets. In Figure 4.11, the three solutions considered during the traditional scheduling approach correspond to Sets 1, 2, and 3.

				Maximally-Feasible Request Sets															
Request #	**Name**	**Reason**	**Grant?**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	Lockard	Brother's wedding				**D**	**D**			**D**			**D**						
2	Feely	Date night with spouse												**D**					
4	Dombrock	Family in town				**D**		**D**				**D**					**D**		
5	Walker	Music concert							**D**							**D**	**D**	**D**	
10	Tinucci	Baby shower			**D**														
24	Conyers	Sister's wedding					**D**		**D**						**D**				
42	Reidesel	Anesthesia assignment								**D**	**D**							**D**	**D**
75	Reidesel	Sister's wedding																	**D**
88	Dombrock	Birthday party out of state										**D**							
131	Tinucci	Competing in a marathon						**D**			**D**								
162	Feely	Music concert											**D**	**D**		**D**			
197	Conyers	Trip to Chicago													**D**				
199	Hobson	Sister's wedding		**D**															

Figure 4.11: Requests That Must Be Considered

When presented with Figure 4.11, the Chief first indicated that he wanted to ensure that the four requests involving weddings were granted. By inputting this information into the "Grant?" column, the tool identifies which maximally-feasible sets are no longer an option and hides them from view. Then, each of the request rows that do not include a "D" in any of the remaining columns are hidden from view, as is done in Figure 4.12, since they no longer need to be considered.

				Maximally-Feasible Request Sets							
Request #	**Name**	**Reason**	**Grant?**	**2**	**5**	**8**	**9**	**11**	**13**	**14**	**15**
2	Feely	Date night with spouse						**D**			
4	Dombrock	Family in town			**D**		**D**			**D**	
5	Walker	Music concert							**D**	**D**	**D**
10	Tinucci	Baby shower		**D**							
42	Reidesel	Anesthesia assignment				**D**					**D**
88	Dombrock	Birthday party out of state					**D**				
131	Tinucci	Competing in a marathon			**D**	**D**					
162	Feely	Music concert						**D**	**D**		

Figure 4.12: After Granting All Requests Involving Weddings

Following the first round of decisions, the Chief indicated that of the remaining requests he wanted to grant Request #4 and Request #131. By again removing the maximally-feasible sets that are no longer an option, as is done in Figure 4.13, four sets remained.

From these sets, the Chief selected MFS #13 (which denies Requests #5 and #162) for implementation.

				MFS			
Request #	**Name**	**Reason**	**Grant?**	**2**	**11**	**13**	**15**
2	Feely	Date night with spouse			**D**		
5	Walker	Music concert				**D**	**D**
10	Tinucci	Baby shower		**D**			
42	Reidesel	Anesthesia assignment					**D**
162	Feely	Music concert			**D**	**D**	

Figure 4.13: After Granting Requests #4 and #131

Using the traditional scheduling approach for this problem, the Chief settled for a solution that only denied Dr. Tinucci's request for a "Baby shower" (Request $\#10$). However, after analyzing each of the maximally-feasible sets, the Chief selected a *different solution* that he was *more satisfied* with. Thus, by having access to every maximally-feasible set, the Chief was able to select a better solution despite the fact that the solution requires denying more requests than the solution selected using the traditional scheduling process. In addition to finding a solution that the Chief was more satisfied with, since the maximally-feasible sets are identified in advance, the Chief was not required to wait for new solutions to be generated after each round of feedback, as is required by the traditional scheduling approach.

Given the relatively small number of maximally-feasible request sets for this problem instance, it was easy for the Chief to quickly analyze the exhaustive collection of them. As a result, it was not necessary to also consider the minimally-infeasible sets.

4.6.2 Case 2

For Case 2, we solved a different problem instance from Scenario 8 involving 218 total requests. We started by solving the instance using the traditional approach of generating a solution that grants the maximum number of requests and then adding additional requirements to the problem based on feedback from the Chief. After five iterations of generating

solutions and getting feedback, the Chief settled for a solution that denied two separate requests for "Weekend Stuff."

Next, we generated the exhaustive collections of maximally-feasible and minimally-infeasible request sets. In total, this problem included 516 maximally-feasible and 7 minimally-infeasible sets. Given these numbers, we decided to work with the minimally-infeasible sets.

When using minimally-infeasible sets for the scheduling process, for each minimally-infeasible set of requests it is necessary to deny at least one request that is included in the set in order to *repair* the minimally-infeasible set. To simplify this process, we created a simple visualization tool that helps the scheduler work with the minimally-infeasible sets.

Figure 4.14 is a snapshot of the tool being used to represent all seven minimally-infeasible request sets and a portion of the 218 requests included in the problem instance. In Figure 4.14, each column represents a minimally-infeasible set and each row represents an individual request. Each check mark indicates that the request is a member of the minimally-infeasible set in the corresponding column. To obtain a feasible solution, the scheduler must repair every minimally-infeasible set by choosing at least one check mark in each column and denying the associated request (note that denying a request to repair one column may repair some other columns as well). For example, from Figure 4.14 we can see that denying Request #1 repairs Sets #6 and #7; denying Request #3 repairs Sets #1, #2, #4, and #5; finally, either Request #217 or #218 can be denied to repair Set #3 (note that denying Request #218 would eliminate the need to deny Request #3). Requests that are not part of any minimally-infeasible set, such as Request #2, can always be granted without denying any requests and therefore do not need to be considered when deciding which requests to deny.

				Minimally-Infeasible Request Sets						
Request #	**Name**	**Reason**	**Deny?**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	Brisson	Doctor appointment							✓	✓
2	Crowther	Son's recital								
3	Brigley	Anesthesia assignment		✓	✓		✓	✓		
4	Palmer	Just because			✓					
5	Strahota	Golf tournament								
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
216	Hecht	Weekend stuff (okay to work)								
217	Brigley	Spouse's birthday				✓				
218	Beulke	Fellowship interview		✓	✓	✓	✓	✓		

Figure 4.14: Complete Collection of Requests and Minimally-Infeasible Request Sets

By hiding all requests that are not a part of any minimally-infeasible request set (and therefore never need to be denied) and arranging the remaining requests in lexicographical order, as is done in Figure 4.15, we can see that only 34 requests need consideration. The other 184 requests can always be granted.

One way to work through the requests and sets represented in Figure 4.15 is to first look at Minimally-Infeasible Set #1. From it, we can see that at least one of the top seven requests must be denied. When presented with this decision, the Chief indicated that he was willing to deny the first request from Dr. Beulke (Request #87) since it appeared to be the least important of the requests while also repairing five minimally-infeasible sets.

				Minimally-Infeasible Request Sets						
Request #	**Name**	**Reason**	**Deny?**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
87	Beulke	Just because		✓	✓	✓	✓	✓		
208	Crowther	Weekend stuff (okay to work)		✓	✓	✓	✓	✓		
218	Beulke	Fellowship interview		✓	✓	✓	✓	✓		
150	Shea	Conference		✓	✓	✓	✓			
99	Linde	Fellowship interview		✓	✓	✓		✓		
3	Brigley	Anesthesia assignment		✓	✓		✓	✓		
57	Palmer	Anesthesia assignment		✓		✓	✓	✓		
28	Ehrich	Conference			✓	✓	✓	✓		
30	Aarnio	Just because			✓	✓	✓	✓		
49	Strahota	Out of town wedding			✓	✓	✓	✓		
103	Brisson	Retreat			✓	✓	✓	✓		
195	Mickley	Camping			✓	✓	✓	✓		
4	Palmer	Just because			✓					
217	Brigley	Spouse's birthday				✓				
15	Brigley	Training course					✓			
172	Shea	Family in town						✓		
1	Brisson	Doctor appointment							✓	✓
7	Guerekis	Competing in a race							✓	✓
20	Linde	My birthday							✓	✓
25	Adams	Weekend stuff (okay to work)							✓	✓
30	Hecht	Camping							✓	✓
65	Jarratt	Day after a wedding							✓	✓
68	Beulke	Rehearsal dinner for a wedding							✓	✓
74	Crowther	Doctor appointment							✓	✓
122	Morgans	Board review course							✓	✓
134	Adams	Retreat							✓	✓
141	Beulke	Retreat							✓	✓
149	Strahota	Car service appointment							✓	✓
159	Brisson	Day off with significant other							✓	✓
165	Ehrich	Trip to Chicago							✓	✓
188	Esper	Trip to Chicago							✓	✓
200	Shea	Date night with spouse							✓	✓
67	Mills	Retreat							✓	
34	Hecht	Retreat								✓

Figure 4.15: Requests That Must Be Considered

Updating the tool with this information, we hide each repaired set and each request that is not a part of any remaining minimally-infeasible set, as can be seen in Figure 4.16. In Figure 4.16, we can see that the Chief must decide to deny a single request from the first 16 requests, or to deny the final two requests from Dr. Mills (#67) and Dr. Hecht (#34).

Request #	Name	Reason	Deny?	MIS 6	MIS 7
1	Brisson	Doctor Appointment		✓	✓
7	Guerekis	Competing in a race		✓	✓
20	Linde	My birthday		✓	✓
25	Adams	Weekend stuff (okay to work)		✓	✓
30	Hecht	Camping		✓	✓
65	Jarratt	Day after a wedding		✓	✓
68	Beulke	Rehearsal dinner for a wedding		✓	✓
74	Crowther	Doctor appointment		✓	✓
122	Morgans	Board review course		✓	✓
134	Adams	Retreat		✓	✓
141	Beulke	Retreat		✓	✓
149	Strahota	Car service appointment		✓	✓
159	Brisson	Day off with significant other		✓	✓
165	Ehrich	Trip to Chicago		✓	✓
188	Esper	Trip to Chicago		✓	✓
200	Shea	Date night with spouse		✓	✓
67	Mills	Retreat		✓	
34	Hecht	Retreat			✓

Figure 4.16: After Denying Request #87

When presented with this decision, the Chief indicated that he preferred denying Dr. Strahota's request for a "Car Service Appointment." Consequently, no additional requests must be denied and it is possible to implement a schedule that only denies the two requests selected by the Chief.

When working with minimally-infeasible sets in this manner, it is possible for schedulers to unnecessarily deny individual requests. For example, consider Figure 4.15. If the schedulers had first chosen to deny Request #150, they might then choose to deny Request #87 in order to repair Minimally-Infeasible Request Set #5. However, since Request #87 is in every minimally-infeasible set that Request #150 is in, if #87 is denied, it is not necessary to deny #150. To avoid unnecessarily denying requests, once the schedulers select a set of requests to deny, they can use the visualization tool to view all maximally-feasible sets that only deny a subset of the requests selected and then choose their most preferred set.

With our proposed scheduling approach, the Chief used minimally-infeasible sets to select a different solution than the one he selected using the traditional scheduling approach.

Although both solutions denied a total of two requests, the Chief preferred the solution selected through using the minimally-infeasible sets.

4.6.3 Case 3

In Case 1 and Case 2 we considered problem instances for which the exhaustive collection of sets is known (i.e., Type 1 problem instances). For Case 1, there were many maximally-feasible sets, but a small number of minimally-infeasible sets, and a process was presented for using these minimally-infeasible sets to select a solution. For Case 2, there were relatively few maximally-feasible sets and a process was discussed for using them to select a solution. In this section we consider a problem instance for which the exhaustive collection of sets is not known (i.e., a Type 2 problem instance) and present some process options for selecting a solution.

The problem instance in this case was from Scenario 8 and included 245 total requests. Using the traditional approach, an initial solution was generated that granted 242 requests. However, the Chief was not satisfied with the three specific requests that were denied in the solution, so we added additional requirements to the problem and generated a different solution. After four iterations of this process, the Chief settled for a solution that granted a different set of 242 requests.

Using our proposed scheduling process on this problem instance, we generated maximally-feasible and minimally-infeasible sets until a total of 1,000 sets were generated. In total, 987 maximally-feasible and 13 minimally-infeasible sets were identified. With these sets, one option for determining a solution is to choose the preferred maximally-feasible request from those that were identified, similar to Case 2. For this problem instance, since a relatively small number of minimally-infeasible sets had been identified, we decided to work with them using the process explained in Section 4.6.1.

By doing this, the Chief was able to quickly identify a set of three requests that he was willing to deny in order to repair all 13 of the known minimally-infeasible sets. To

check the feasibility of granting the remaining 242 requests and ensure no requests were unnecessarily denied, we added requirements to the problem to ensure that all of the 242 requests were granted before maximizing the number of additional requests granted. From this, we confirmed that it was possible to grant these 242 requests, and furthermore that it was not possible to grant any additional requests. Like in the previous two cases, the Chief was more satisfied with the solution selected using our proposed scheduling approach than the solution selected using the traditional process. Thus, even though we did not generate the exhaustive collections of sets in this case, the Chief was able to quickly select a better solution using our proposed scheduling approach.

We recognize that when the exhaustive collection of sets is unknown, repairing the minimally-infeasible sets that are known does not guarantee a feasible schedule. We plan to explore this situation through future research.

4.6.4 Case Study Feedback

By working with the Chief through multiple problem instances, we learned that his personal preference is for working with minimally-infeasible sets since each set requires choosing a single request to deny and it is easier for him to think about "fixing all of the problems." When asked if he prefers the traditional scheduling approach or our new scheduling approach using maximally-feasible and minimally-infeasible sets, the chief commented, "without question, I like the new approach. With it, it is easy to see what problems need to be fixed and what solutions are possible."

4.7 Conclusion and Future Research

In this chapter, we address an important problem that is regularly encountered when scheduling medical residents. Specifically, for resident scheduling problems in which it is impossible to grant every time-off request, we develop a method that identifies the exhaustive

collection of maximally-feasible and minimally-infeasible request sets which can then be used by schedulers to choose their preferred solution. To do this, we create two algorithms that each identify the exhaustive collection of sets and develop visualization tools for presenting the sets to schedulers in a way that allows them to quickly select their preferred solution.

Through computational testing on our Sequential and Simultaneous RSVC algorithms, we conclude that Simultaneous RSVC is superior to Sequential RSVC based on run-times and the fact that Simultaneous RSVC is able to identify some minimally-infeasible sets without necessarily having to generate the exhaustive collection of maximally-feasible sets.

We directly compare a scheduler's experience using our proposed scheduling method to that of the current scheduling process. We find that by presenting a scheduler with every maximally-feasible and minimally-infeasible set, the scheduler was able to quickly identify a high-quality solution. An additional benefit of using maximally-feasible and minimally-infeasible is that the schedulers can be certain that no better solutions exist.

Our new method for resident scheduling has numerous benefits over more traditional methods, but many opportunities for further research and improvements remain. Specifically, incorporating additional scheduling metrics of interest other than time-off requests, such as the number of weekend shifts that each resident is assigned to work, would be useful. For problem instances where it is not practical to generate every maximally-feasible set, we are currently exploring methods for determining the complete collection of minimally-infeasible sets. Additionally, we are working to improve the scheduling process through increased automation and by improving our visualization tool to make it more interactive.

Although the focus of this chapter is on a specific resident shift scheduling problem, our proposed methods are applicable to any problem involving soft constraints. Specifically, our methods make it easier for decision makers to see solution possibilities for problems in which it is feasible to satisfy some, but not all preferences.

CHAPTER 5

Conclusion

In this dissertation, we study three healthcare provider scheduling problems and address challenges associated with solving each of them in practice.

In Chapter 2, we develop a mathematical model for allocating both operating and clinical rooms to surgeons within a multi-location health system. In doing so, we describe how alternative variable definitions can be used to simplify the modeling of complex scheduling requirements and improve the tractability of the mathematical program used to solve the scheduling problem.

In Chapter 3, we focus on a block scheduling problem for medical residents and further explore the concept of using alternative decision variable definitions to improve the computational performance of the mathematical model used to solve the problem. We begin by describing a general block scheduling problem and then formulate a model for solving a rather simple problem variation. We then introduce a more complex problem variation and propose an alternative model for solving it. We use computational testing to compare the performance of the proposed models. Next, we explain an additional problem variation that necessitates an alternative modeling approach. For addressing this variation, we define additional decision variables to indicate whether or not each resident starts a specific rotation during each block of time. Using these variables, we formulate two new models that are extensions of the first two models. We demonstrate the performance of each model by using it to solve a set of test problems. In our work, we show that for specific scheduling

problem variations, there can be computational advantages to defining composite decision variables that each represent multiple decisions.

For scheduling healthcare providers, a typical objective is to maximize provider satisfaction. A common approach for accomplishing this objective is to maximize the number of individual scheduling requests satisfied—as was the case for the scheduling problems addressed in Chapters 3 and 4. However, since requests have differing levels of importance, simply maximizing the total number of requests satisfied may not result in a preferred schedule. Instead of generating a single solution that maximizes the total number of satisfied requests, in Chapter 4, we develop a method that identifies the exhaustive collection of maximally-feasible and minimally-infeasible request sets which can then be used by schedulers to choose their preferred solution. We apply our methods to a resident shift scheduling problem and directly compare a scheduler's experience using our proposed scheduling method to that of the current scheduling process. We find that by presenting a scheduler with every maximally-feasible and minimally-infeasible set, the scheduler is able to quickly identify a high-quality solution. Although we focus on scheduling medical residents in Chapter 4, our proposed method is applicable to any problem involving soft constraints.

In this thesis, we make a number of contributions. Specifically, we propose a novel method for identifying solutions to scheduling problems in which it is possible to satisfy some, but not all of the individual scheduling requests. Although we apply our method to a healthcare provider scheduling problem, it is applicable to any problem involving preferences. Additionally, we model real-world problems and demonstrate how alternative variable definitions can be used to simplify the modeling of complex scheduling rules and improve computational performance when solving the problems. The work presented in this dissertation is intended to help healthcare provider schedulers with some of the challenges they may experience.

BIBLIOGRAPHY

Agarwal, A. (2016). *Balancing Medical Resident Education and Workload while Ensuring Quality Patient Care*. PhD thesis, Rochester Institute of Technology.

Aickelin, U., Burke, E. K., & Li, J. (2009). An evolutionary squeaky wheel optimization approach to personnel scheduling. *IEEE Transactions on Evolutionary Computation*, 13(2), 433–443.

Amaldi, E., Pfetsch, M. E., & Trotter Jr, L. E. (1999). Some structural and algorithmic properties of the maximum feasible subsystem problem. *In Proceedings of the 10th Integer Programming and Combinatorial Optimization Conference. Lecture Notes in Computer Science*, 1610, 45–59.

Armacost, A. P., Barnhart, C., & Ware, K. A. (2002). Composite variable formulations for express shipment service network design. *Transportation science*, 36(1), 1–20.

Bailey, J. & Stuckey, P. J. (2005). Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. *In Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages*, 3350, 174–186.

Bard, J. F., Shu, Z., & Leykum, L. (2013). Monthly clinic assignments for internal medicine housestaff. *IIE Transactions on Healthcare Systems Engineering*, 3(4), 207–239.

Bard, J. F., Shu, Z., Morrice, D. J., Leykum, L. K., & Poursani, R. (2016). Annual block scheduling for family medicine residency programs with continuity clinic considerations. *IIE Transactions*, 8830(April), 1–15.

Beaulieu, H., Ferland, J. A., Gendron, B., & Michelon, P. (2000). A mathematical programming approach for scheduling physicians in the emergency room. *Health care management science*, 3(3), 193–200.

Blake, J. T. & Donald, J. (2002). Mount sinai hospital uses integer programming to allocate operating room time. *Interfaces*, 32(2), 63–73.

Bowers, M. R., Noon, C. E., Wu, W., & Bass, J. K. (2016). Neonatal physician scheduling at the University of Tennessee medical center. *Interfaces*, 46(2), 168–182.

Bruni, R. & Detti, P. (2014). A flexible discrete optimization approach to the physician scheduling problem. *Operations Research for Health Care*, 3(4), 191–199.

Brunner, J. O., Bard, J. F., & Kolisch, R. (2009). Flexible shift scheduling of physicians. *Health Care Management Science*, 12(3), 285–305.

Burke, E. K., De Causmaecker, P., Berghe, G. V., & Van Landeghem, H. (2004). The state of the art of nurse rostering. *Journal of scheduling*, 7(6), 441–499.

Burke, E. K., Li, J., & Qu, R. (2012). A pareto-based search methodology for multi-objective nurse scheduling. *Annals of Operations Research*, 196(1), 91–109.

Cardoen, B., Demeulemeester, E., & Beliën, J. (2010). Operating room planning and scheduling: a literature review. *European Journal of Operational Research*, 201(3), 921–932.

Carter, M. W. & Lapierre, S. D. (2001). Scheduling emergency room physicians. *Health Care Management Science*, 4(4), 347–360.

Cayirli, T. & Veral, E. (2003). Outpatient scheduling in health care: a review of literature. *Production and Operations Management*, 12(4), 519–549.

Chakravarti, N. (1994). Some results concerning post-infeasibility analysis. *European Journal of Operational Research*, 73(1), 139–143.

Cheang, B., Li, H., Lim, A., & Rodrigues, B. (2003). Nurse rostering problems - a bibliographic survey. *European Journal of Operational Research*, 151(3), 447–460.

Chiaramonte, M. V. & Chiaramonte, L. M. (2008). An agent-based nurse rostering system under minimal staffing conditions. *International Journal of Production Economics*, 114(2), 697–713.

Chinneck, J. W. (2007). *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, volume 118. Springer Science & Business Media.

Cohn, A. & Lapp, M. (2010). Airline resource scheduling. *Wiley Encyclopedia of Operations Research and Management Science*, (October).

Cohn, A. M. & Barnhart, C. (2003). Improving crew scheduling by incorporating key maintenance routing decisions. *Operations Research*, 51(3), 387–396.

Cohn, A. M., Root, S., Kymissis, C., Esses, J., & Westmoreland, N. (2009). Scheduling medical residents at boston university school of medicine. *Interfaces*, 39(3), 186–195.

De Causmaecker, P. & Vanden Berghe, G. (2011). A categorisation of nurse rostering problems. *Journal of Scheduling*, 14(1), 3–16.

de Grano, M. L., Medeiros, D. J., & Eitel, D. (2009). Accommodating individual preferences in nurse scheduling via auctions and optimization. *Health Care Management Science*, 12(3), 228–242.

Erhard, M., Schoenfelder, J., Fgener, A., & Brunner, J. O. (2016). State of the art in physician scheduling. Available at SSRN: https://ssrn.com/abstract=2813360.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., & Sier, D. (2004a). An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127(1-4), 21–144.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004b). Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1), 3–27.

Franz, L. S. & Miller, J. L. (1993). Scheduling medical residents to rotations: solving the large-scale multiperiod staff assignment problem. *Operations Research*, 41(2), 269–279.

Graves, S. C. (1981). A review of production scheduling. *Operations Research*, 29(4), 646–675.

Guieu, O. & Chinneck, J. W. (1999). Analyzing infeasible mixed-integer and integer linear programs. *INFORMS Journal on Computing*, 11(1), 63–77.

Güler, M. G., Idin, K., & Yilmaz Güler, E. (2013). A goal programming model for scheduling residents in an anesthesia and reanimation department. *Expert Systems with Applications*, 40(6), 2117–2126.

Gunawan, A. & Lau, H. C. (2012). Master physician scheduling problem. *Journal of the Operational Research Society*, 64(3), 410–425.

Guo, J., Morrison, D. R., Jacobson, S. H., & Jokela, J. A. (2014). Complexity results for the basic residency scheduling problem. *Journal of Scheduling*, 17(3), 211–223.

Hall, R. (2012). *Handbook of Healthcare System Scheduling*. New York, NY: Springer Science & Business Media.

Hayes, L. J., Brien-pallas, L. O., Duffield, C., Shamian, J., Buchan, J., Hughes, F., Spence, H. K., North, N., Stone, P. W., O'Brien-Pallas, L., Duffield, C., Shamian, J., Buchan, J., Hughes, F., Spence Laschinger, H. K., North, N., & Stone, P. W. (2006). Nurse turnover: a literature review. *International Journal of Nursing Studies*, 43(2), 237–63.

Higgins, A., Kozan, E., & Ferreira, L. (1996). Optimal scheduling of trains on a single line track. *Transportation Research Part B: Methodological*, 30(2), 147–158.

Li, Y. & Jones, C. B. (2013). A literature review of nursing turnover costs. *Journal of Nursing Management*, 21(3), 405–418.

Maass, K. L., Liu, B., Daskin, M. S., Duck, M., Wang, Z., Mwenesi, R., & Schapiro, H. (2015). Incorporating nurse absenteeism into staffing with demand uncertainty. *Health Care Management Science*.

MacCarthy, B. & Liu, J. (1993). Addressing the gap in scheduling research - a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1), 59–79.

Nemhauser, G. L. & Trick, M. A. (1998). Scheduling a major college basketball conference. *Operations Research*, 46(1), 1–8.

Ovchinnikov, A. & Milner, J. (2008). Spreadsheet model helps to assign medical residents at the University of Vermont's College of Medicine. *Interfaces*, 38(4), 311–323.

Perelstein, E., Rose, A., Hong, Y.-c., Cohn, A., & Long, M. T. (2016). Automation improves schedule quality and increases scheduling efficiency for residents. *Journal of Graduate Medical Education*, 8(1), 45–49.

Pinedo, M. L. (2012). *Scheduling: Theory, Algorithms, and Systems*. New York, NY: Springer Science and Business Media, 4th edition.

Proano, R. & Agarwal, A. (2017). Scheduling internal medicine resident rotations to ensure fairness and facilitate continuity of care. Rochester Institute of Technology, New York.

Rasmussen, R. V. & Trick, M. A. (2008). Round robin scheduling - a survey. *European Journal of Operational Research*, 188(3), 617–636.

Sagie, A. & Krausz, M. (2003). What aspects of the job have most effect on nurses? *Human Resource Management Journal*, 13(1), 46–62.

Santibáñez, P., Begen, M., & Atkins, D. (2007). Surgical block scheduling in a system of hospitals: an application to resource and wait list management in a british columbia health authority. *Health care management science*, 10(3), 269–282.

Sherali, H. D., Ramahi, M. H., & Saifee, Q. J. (2002). Hospital resident scheduling problem. *Production Planning & Control*, 13(2), 220–233.

Smalley, H. K. & Keskinocak, P. (2016a). Automated medical resident rotation and shift scheduling to ensure quality resident education and patient care. *Health Care Management Science*, 19(1), 66–88.

Smalley, H. K. & Keskinocak, P. (2016b). Automated medical resident rotation and shift scheduling to ensure quality resident education and patient care. *Health care management science*, 19(1), 66–88.

Smalley, H. K., Keskinocak, P., & Vats, A. (2015). Physician scheduling for continuity: an application in pediatric intensive care. *Interfaces*, 45(2), 133–148.

Smet, P., Bilgin, B., De Causmaecker, P., & Vanden Berghe, G. (2014). Modeling and evaluation issues in nurse rostering. *Annals of Operations Research*, 218(1), 303–326.

Stahlbock, R. & Vo, S. (2008). Operations research at container terminals: a literature update. *OR Spectrum*, 30(1), 1–52.

Stolletz, R. & Brunner, J. O. (2012). Fair optimization of fortnightly physician schedules with flexible shifts. *European Journal of Operational Research*, 219, 622–629.

Topaloglu, S. (2006). A multi-objective programming model for scheduling emergency medicine residents. *Computers & Industrial Engineering*, 51(3), 375–388.

Topaloglu, S. (2009). A shift scheduling model for employees with different seniority levels and an application in healthcare. 198(3), 943–957.

Topaloglu, S. & Ozkarahan, I. (2011). A constraint programming-based solution approach for medical resident scheduling problems. *Computers and Operations Research*, 38(1), 246–255.

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., & De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3), 367–385.

Van Loon, J. (1981). Irreducibly inconsistent systems of linear inequalities. *European Journal of Operational Research*, 8(3), 283–288.

Warner, D. M. (1976). Scheduling nursing personnel according to nursing preference: A mathematical programming approach. *Operations Research*, 24(5), 842–856.

Wolfe, H. & Young, J. P. (1965). Staffing the nursing unit: Part i. controlled variable staffing. *Nursing Research*, 14(3), 236–242.

Zenteno, A. C., Carnes, T., Levi, R., Daily, B. J., & Dunn, P. F. (2016). Systematic OR block allocation at a large academic medical center: comprehensive review on a data-driven surgical scheduling strategy. *Annals of Surgery*, 264(6), 973–981.