POMS PRODUCTION AND OPERATIONS MANAGEMENT

PRODUCTION AND OPERATIONS MANAGEMENT

Vol. 26, No. 9, September 2017, pp. 1778–1793
ISSN 1059-1478 | EISSN 1937-5956 | 17 | 2609 | 1778

DOI 10.1111/poms.12728
© 2017 Production and Operations Management Society

New Methods for Resolving Conflicting Requests with Examples from Medical Residency Scheduling

Brian Lemay, Amy Cohn, Marina Epelman
Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan 48109, USA, blemay@umich.edu, amycohn@umich.edu, mepelman@umich.edu

Stephen Gorga
C.S. Mott Children's Hospital, University of Michigan, Ann Arbor, Michigan 48109, USA, smgorga@med.umich.edu

In scheduling medical residents, the objective is often to maximize resident satisfaction across the space of feasible schedules, relative to the many *hard constraints* that ensure appropriate patient coverage, adequate training opportunities, etc. A common metric of resident satisfaction is the number of time-off requests that are 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. Although we focus our exposition on the problem of resident scheduling, our approach is applicable to a broad class of problems with *soft constraints*.

Key words: Scheduling; healthcare; medical residency; optimization
History: Received: October 2016; Accepted: April 2017 by Nitin Joglekar, after 1 revision.

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 the study is organized as follows. In section 2, we review existing literature on healthcare personnel scheduling and finding maximally feasible and minimally infeasible sets. In section 3, we describe the specific resident scheduling problem that we are considering. In section 4, we present the two Request Selection via Cuts (RSVC) algorithms and provide computational results in section 5. In section 6, we present our findings from a scheduling case-study conducted at Mott Children's Hospital. We conclude in section 7 by summarizing our findings and providing suggestions for future work.

2. Literature Review

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 health care 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 and Jones 2013).

The NSP first appeared in the literature in the 1960s (Wolfe and 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 and Keskinocak 2016; 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 schedules. 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 and Ozkarahan (2011) and Topaloglu (2006, 2009) discuss other multi-objective resident shift scheduling models for emergency medicine residents.

Ovchinnikov and 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 makers 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.

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 and 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 and 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 and 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 and Stuckey (2005) do, we formulate and solve a mathematical optimization problem.

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.

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 fulfil 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.

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 7 am, 9 am, 12 pm, 4 pm, 5 pm, 8 pm, and 11 pm. The shifts starting at 8 pm and 11 pm 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.

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, 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 in this study; 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 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 Appendix S1.

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.

4.1. 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 the study:

- $\mathbf{x} \in \{0, 1\}^n$ is a "request vector,"that is, 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.

4.2. 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 1 and a formal description in Appendix S2.

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:

$$(\text{NewFeas})_0 \quad \text{maximize} \sum_{r \in R} x_r, \tag{1}$$

$$\text{subject to} \quad \mathbf{x} \in \mathbf{X}^H. \tag{2}$$

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 (25)–(34) in Appendix S1), or, more precisely, the projection of the feasible region onto the space of x variables. Thus, solving the problem represented by Equations (1) and (2) is equivalent to solving the problem described in section 3.

If $(\text{NewFeas})_0$ 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 $(\text{NewFeas})_0$ 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:

$$(\text{NewFeas})_1 \quad \text{maximize} \sum_{r \in R} x_r, \tag{3}$$

$$\text{subject to} \quad \mathbf{x} \in \mathbf{X}^H, \tag{4}$$

$$\sum_{r \in R \setminus R_0^F} x_r \geq 1. \tag{5}$$

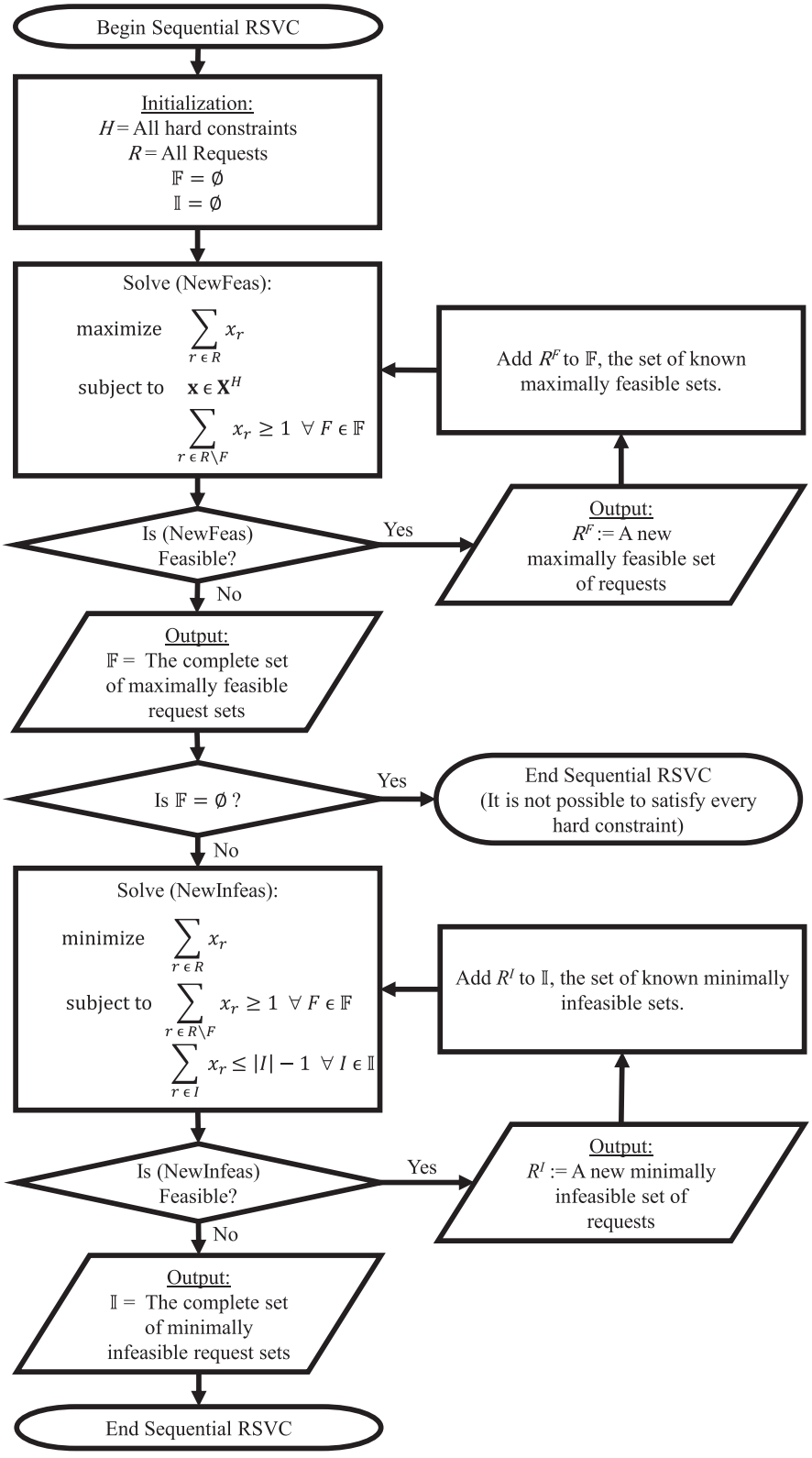
The cut $\sum_{r \in R \setminus R_0^F} x_r \geq 1$ eliminates exactly those solutions that only satisfy the requests in R_0^F or a proper subset of the requests in R_0^F. (If $R_0^F = \emptyset$, this constraint is interpreted as $\sum_{r \in R} x_r \geq 1$, and if $R_0^F = R$ — as "$0 \geq 1$.") Since 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:

$$(\text{NewFeas})_i \quad \text{maximize} \sum_{r \in R} x_r, \tag{6}$$

Figure 1 Sequential Request Selection Via Cuts



$$\text{subject to} \quad \mathbf{x} \in \mathbf{X}^H, \tag{7}$$

$$\sum_{r \in R \setminus R_k^F} x_r \geq 1 \quad \forall\, k = 0, \ldots, i-1, \tag{8}$$

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 in Appendix S4. At the first iteration when a problem $(\text{NewFeas})_i$ is infeasible, every maximally feasible request set will have been identified, by Theorem 2.

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:

$$(\text{NewInfeas})_0 \quad \text{minimize} \sum_{r \in R} x_r, \tag{9}$$

$$\text{subject to} \quad \sum_{r \in R \setminus F} x_r \geq 1 \quad \forall F \in \mathbb{F}. \tag{10}$$

(If $\mathbb{F} = \{\emptyset\}$, constraint (10) 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 Equation (10) 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 $(\text{NewInfeas})_0$ obtained by the solver. Since Equation (10) is satisfied for any optimal solution and $\mathbb{F}$ in Equation (10) 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 $(\text{NewInfeas})_0$ to get:

$$(\text{NewInfeas})_1 \quad \text{minimize} \sum_{r \in R} x_r, \tag{11}$$

$$\text{subject to} \sum_{r \in R \setminus F} x_r \geq 1 \quad \forall F \in \mathbb{F}, \tag{12}$$

$$\sum_{r \in R_0^I} x_r \leq |R_0^I| - 1. \tag{13}$$

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 $(\text{NewInfeas})_1$ 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 $(\text{NewInfeas})_1$ 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 $(\text{NewInfeas})_1$ to get $(\text{NewInfeas})_2$. $(\text{NewInfeas})_2$ can then be solved to find the next smallest minimally infeasible request set. Continuing in this manner of iteratively constructing and solving problems of the form:

$$(\text{NewInfeas})_j \quad \text{minimize} \sum_{r \in R} x_r, \tag{14}$$

$$\text{subject to} \sum_{r \in R \setminus F} x_r \geq 1 \qquad \forall F \in \mathbb{F}, \tag{15}$$

$$\sum_{r \in R_k^I} x_r \leq |R_k^I| - 1 \quad \forall\, k = 0, \ldots, j-1, \tag{16}$$

will result in identifying one new minimally infeasible request set in each iteration, identified in nondecreasing 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. In Figure 1, Equation (16) 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.

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 2 and a formal description in Appendix S3.

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:

$$\text{(FirstFeas)} \quad \text{maximize} \sum_{r\in R} x_r, \tag{17}$$

$$\text{subject to} \quad \mathbf{x} \in \mathbf{X}^H. \tag{18}$$

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, that is, 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:

$$\text{(CandidateSet)} \quad \text{minimize} \sum_{r\in R} x_r, \tag{19}$$

$$\text{subject to} \sum_{r\in R\setminus F} x_r \geq 1 \qquad \forall\, F \in \mathbb{F}, \tag{20}$$

$$\sum_{r\in I} x_r \leq |I| - 1 \quad \forall\, I \in \mathbb{I}. \tag{21}$$

Here, Equation (20) ensures that the candidate set is not a subset of any known maximally feasible set and Equation (21) 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:

$$\text{(FeasTest)} \ \text{maximize} \sum_{r\in R} x_r, \tag{22}$$

$$\text{subject to} \quad \mathbf{x} \in \mathbf{X}^H, \tag{23}$$

$$x_r = 1 \quad \forall\, r \in R^\star. \tag{24}$$

Here, Equation (23) ensures feasibility of the schedule and Equation (24) ensures that the solution grants every request in the candidate set $R^\star$. If (FeasTest) is infeasible, $R^{\star\cdot}$ 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.

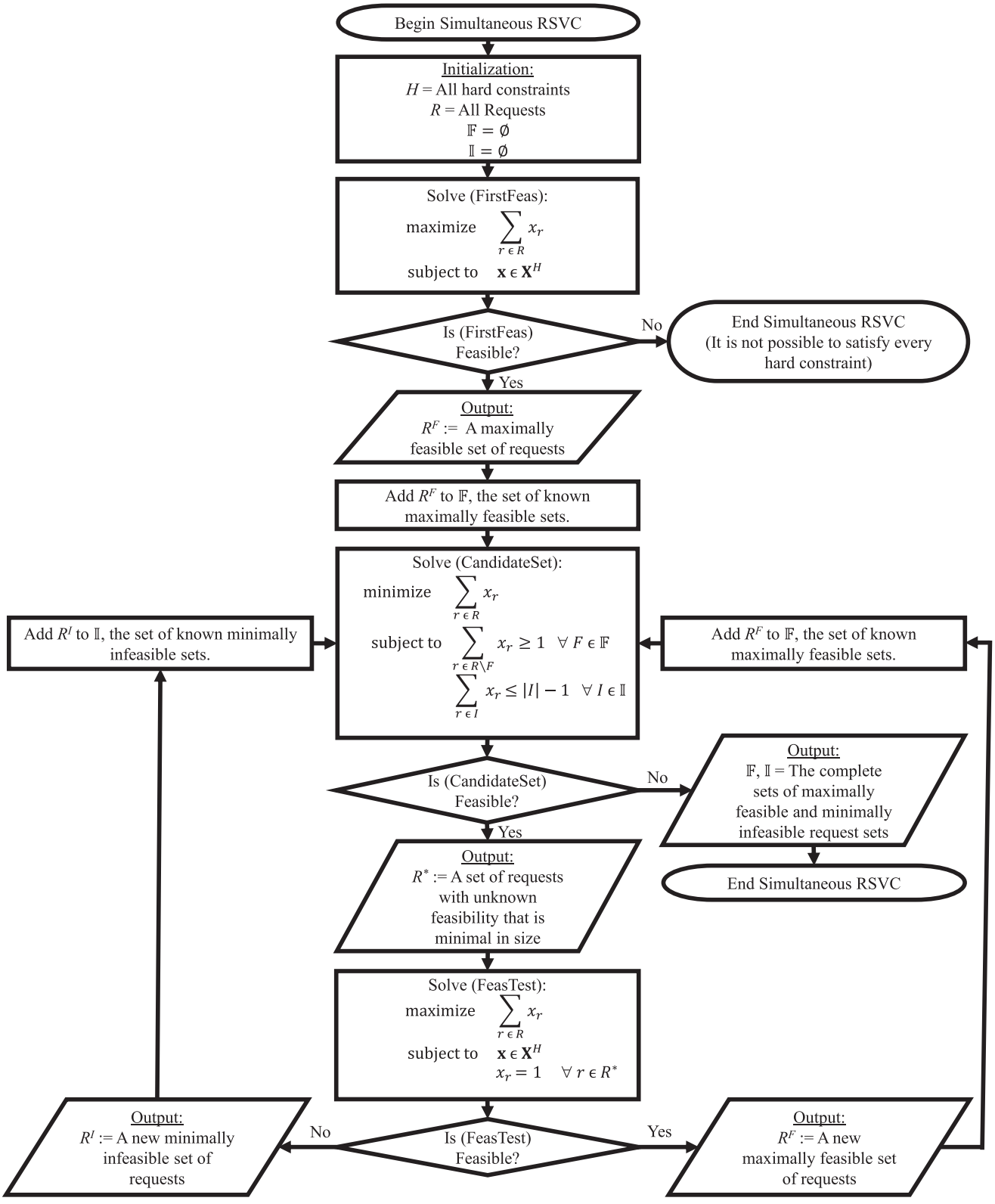
By Theorem 6, (CandidateSet) is infeasible precisely when $\mathbb{F}$ and $\mathbb{I}$ contain every maximally feasible and minimally infeasible request set, respectively. By Theorem 7, this will happen after a finite number of iterations, and the algorithm will terminate.

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?

Figure 2 Simultaneous Request Selection Via Cuts



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.

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 10–11 shifts with 3–4 of these as night shifts per month (i.e. a tightly-constrained schedule) or 5–15 total shifts with 0–10 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 2 in Appendix S5 summarizes all 24 scenarios that we use to generate problem instances for computational testing.

5.2. Problem Instance Characteristics

For computational testing, a set of 50 random problem instances for each testing scenario in Table 2 was solved using both algorithms. In Figure 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 relevant. For the remainder of our computational experiments, we focus on these interesting 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 1000 or fewer maximally feasible sets *and* 1000 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

Figure 3 Feasibility of Problem Instances

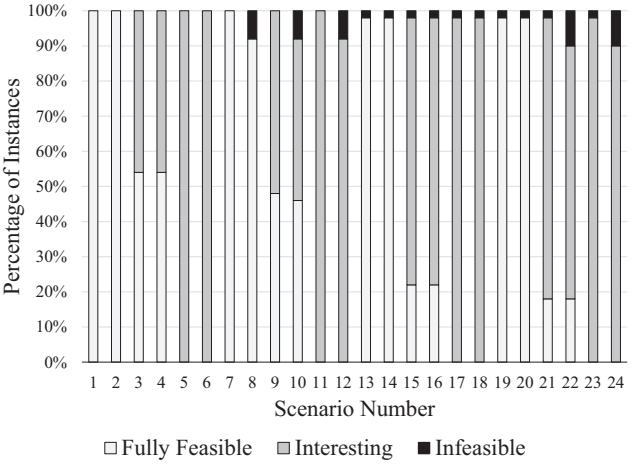
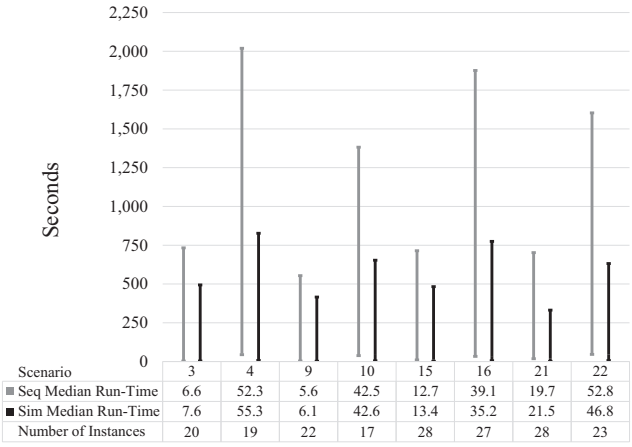


Table 1 Type 1 Problem Instances

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

Figure 4 Median, Minimum, and Maximum Run-Times for Type 1 Instances



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

1000 maximally feasible sets or more than 1000 minimally infeasible sets.

5.3. Type 1 Problem Instances

In Table 1, 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.

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 8 of Appendix S5.

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. From Figure 4, 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.

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 9 and 10 in Appendix S5 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.

5.4. Type 2 Problem Instances

All Type 2 problem instances include at least 1000 maximally feasible or at least 1000 minimally

infeasible sets. For testing Type 2 problem instances, each algorithm is run on every instance until it finds a total of 1000 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 1000 sets.

Of the 452 Type 2 problem instances, only two had fewer than 1000 maximally feasible sets and instead had more than 1000 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 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 5 x (2.6 hours) faster), and in the remaining cases is typically comparable. We plot the run-times for each Type 2 instance in Figures 11 and 12 of Appendix S5.

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 Simultaneous 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.

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 a co-author of this study) who is responsible for scheduling pediatric residents. For the case study, we considered several "interesting" problem instances from section 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 following section, we describe the Chief's experiences using each scheduling approach for a specific problem instance (case) involving many maximally feasible and few minimally infeasible request sets. In Appendix S6 we discuss the Chief's experiences for two other cases: one involving few maximally feasible request sets and one in which the exhaustive collection of maximally feasible and minimally infeasible request sets is unknown.

6.1. Case 1

For Case 1, we solved a problem instance from Scenario 8 (see Appendix S5 for details) 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 5 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 5, each column represents a minimally infeasible set and each row represents an individual request. Each check mark indicates that the

Figure 5 Complete Collection of Requests and Minimally Infeasible Request Sets

				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		✓	✓	✓	✓	✓		

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 5, 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.

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 6, 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 6 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 of the minimally infeasible sets.

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 7. In Figure 7, 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).

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 6. 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.

6.2. 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" (see Appendix S6 for discussion of the Chief's

Figure 6 Requests That Must Be Considered

				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								✓

experiences working with maximally feasible sets). 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."

7. Conclusion and Future Research

In this study, 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

Figure 7 After Denying Request #87

				MIS	
Request #	**Name**	**Reason**	**Deny?**	**6**	**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			✓

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 study 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.

Acknowledgments

We gratefully acknowledge the financial support of the Seth Bonder Foundation, the University of Michigan Center for Healthcare Engineering and Patient Safety, and the University of Michigan Department of Pediatrics.

Disclaimer

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

References

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

Amaldi, E., M. E. Pfetsch, L. E. Trotter, Jr. 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.

Bailey, J., P. J. Stuckey. 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., Z. Shu, L. Leykum. 2013. Monthly clinic assignments for internal medicine housestaff. *IIE Trans. Healthcare Syst. Eng.* **3**(4): 207–239.

Bard, J. F., Z. Shu, D. J. Morrice, L. K. Leykum, R. Poursani. 2016. Annual block scheduling for family medicine residency programs with continuity clinic considerations. *IIE Trans.* **48**(9): 797–811.

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

Chakravarti, N. 1994. Some results concerning post-infeasibility analysis. *Eur. J. Oper. Res.* **73**(1): 139–143.

Cheang, B., H. Li, A. Lim, B. Rodrigues. 2003. Nurse rostering problems - A bibliographic survey. *Eur. J. Oper. Res.* **151**(3): 447–460.

Chinneck, J. W. 2007. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*, Vol. **118**. Springer Science & Business Media, New York, NY.

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

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

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

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

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

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

Hall, R. 2012. *Handbook of Healthcare System Scheduling*, Vol. **168**. Springer, New York.

Hayes, L. J., L. O. Brien-pallas, C. Duffield, J. Shamian, J. Buchan, F. Hughes, H. K. Spence, N. North, P. W. Stone, L. O'Brien-Pallas, C. Duffield, J. Shamian, J. Buchan, F. Hughes, H. K. Spence Laschinger, N. North, P. W. Stone. 2006. Nurse turnover: A literature review. *Int. J. Nurs. Stud.* **43**(2): 237–63.

Li, Y., C. B. Jones. 2013. A literature review of nursing turnover costs. *J. Nurs. Manag.* **21**(3): 405–418.

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

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

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

Topaloglu, S. 2006. A multi-objective programming model for scheduling emergency medicine residents. *Comput. Ind. Eng.* **51**(3): 375–388.

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

Topaloglu, S., I. Ozkarahan. 2011. A constraint programming-based solution approach for medical resident scheduling problems. *Comput. Oper. Res.* **38**(1): 246–255.

Van Loon, J. 1981. Irreducibly inconsistent systems of linear inequalities. *Eur. J. Oper. Res.* **8**(3): 283–288.

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

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

Supporting Information

Additional supporting information may be found online in the supporting information tab for this article:

Appendix S1: Resident Scheduling Problem Formulation.
Appendix S2: Sequential RSVC Algorithm.
Appendix S3: Simultaneous RSVC Algorithm.
Appendix S4: Proofs.
Appendix S5: Additional Tables and Figures.
Appendix S6: Additional Case Studies.