Scheduling Medical Residents With Conflicting Requests For Time Off

Brian Lemay, Amy Cohn, Marina Epelman, Stephen Gorga

Abstract

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 football game. Another approach is to assign a weight to each request and maximize the weighted sum 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 scheduling problems with *soft constraints*. **Keywords:** *Scheduling; Healthcare; Residency; Optimization*

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 football game. As an alternative to maximizing the number of satisfied scheduling requests, each request could be weighted according to its importance, but determining weights that accurately represent each residents' and schedulers' preference can be challenging.

To eliminate the challenge of accurately defining an objective function when using integer programming during the scheduling process, we propose to instead identify 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 paper 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 and formulate 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 healthcare has received significant attention from the research community. (Hall, 2012) is a recently published handbook dedicated specifically to scheduling in healthcare systems.

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, as discussed in (Cavanagh & Coffin, 1992; Leveck & Jones, 1996; Gauci Borda & Norman, 1997; Lu et al., 2005; Hayes et al., 2006; Coomber & Louise Barriball, 2007; Jones & Gates, 2007; Leiter & Maslach, 2009; Yildiz et al., 2009; Li & Jones, 2013).

The NSP made one of its first appearances in (Wolfe & Young, 1965). Since then, many different models and solution techniques have been proposed for addressing a variety of specific scheduling rules and objectives. Satisfying the preferences of nurses is a common objective that is considered in (Miller et al., 1976; Warner, 1976; Berrada et al., 1996; Azaiez & Al Sharif, 2005; Chiaramonte & Chiaramonte, 2008; de Grano et al., 2009; Burke et al., 2012). Numerous other models and solution approaches have been proposed in the literature, many of which are reviewed in (Sitompul & Randhawa, 1990; Cheang et al., 2003; Burke et al., 2004).

For physicians, scheduling typically involves assigning each physician blocks of both time and space (e.g., clinic or operating rooms) that can then be filled with individual appointments. (Gunawan & Lau, 2012) defined the Master Physician Scheduling Problem that involves assigning resources and blocks of time to physicians in order for them to complete all of their weekly tasks, including operations. Physician scheduling is also addressed specifically for emergency departments in (Carter & Lapierre, 2001; Beaulieu et al., 2000) and for operating rooms in (Blake & Donald, 2002; Santibáñez et al., 2007).

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 basic, and because their schedules must not only ensure coverage for adequate patient care (similar to nurses and attending physicians) but must also ensure adequate training opportunities, resident scheduling problems can be particularly challenging.

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. Block scheduling for residents is addressed in (Franz & Miller, 1993; Beliën & Demeulemeester, 2006; Javeri, 2011; Smalley & Keskinocak, 2014; Bard et al., 2016; Agarwal, 2016).

Another resident scheduling problem that is more closely related to nurse scheduling is that of assigning residents to shifts, frequently in emergency departments or to cover call schedules. In (Sherali et al., 2002), a mixed integer program and heuristic solution procedures are developed for assigning residents to night shifts while considering staffing needs, skill requirements, and resident preferences. (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. Other multi-objective resident shift scheduling models for emergency medicine residents are presented in (Topaloglu, 2006, 2009; Topaloglu & Ozkarahan, 2011).

In (Ovchinnikov & Milner, 2008), the authors 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 involving chief residents for generating improved schedules for a resident scheduling problem at Boston University School of Medicine. For their approach, the chief residents first provide feedback on solutions generated by the model. Then, the model is modified according to the feedback and a new schedule is generated. This process continues until they are unable to find an improved schedule. As a result of involving the chief residents in this way, the final schedules were accepted without any complaints from the residents.

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 (i.e., it is not possible to grant any additional request). 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. Building on (Chinneck & Dravnieks, 1991) and (Chinneck, 2001), (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; Amaldi & Kann, 1995; Chakravarti, 1994; 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 single 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 and/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). In their work, “unique and maximal maintenance-feasible short connects” for an aircraft maintenance routing problem are identified using optimization. For generating minimally-infeasible sets, we leverage the relationship between maximally-feasible and minimally-infeasible sets presented in (Bailey & Stuckey, 2005), where the authors note that, given the complete set of maximally-feasible sets of constraints for a particular problem, 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 is done (Bailey & Stuckey, 2005), we formulate and solve a mathematical optimization problem.

3 Resident Scheduling Problem

Although our work is generally applicable to scheduling problems 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 (UMHS) and addressing their potentially conflicting personal requests. This problem, like most residency scheduling problems, has a large number of requirements (i.e., hard constraints). In addition to the requirement that each shift must be covered by a resident, each resident must satisfy educational and work-hour related requirements. 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 fulfill their educational requirements and get experience in a variety of areas related to their specialties. Rotations typically last at least one month and are usually no longer than four months. 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 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.

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 paper; 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 the following mathematical optimization problem to find a schedule that satisfies every rule and denies a minimum number of requests (i.e., 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.
- $A = S \times D$ is the set of all shift/day pairs in the planning horizon. For example (1,2) represents shift 1 on day 2.

- $T^{s,d} \subseteq A$ 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, $T^{4,d} := \{(5,d),(6,d),(7,d),(1,d+1),(2,d+1)\}$.
- $C^p \subseteq A$ 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, $B^r \subseteq A$, 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 vacation request $r \in R$ is granted.

Constraints:

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

$$y_{psd} + \sum_{(\bar{s},\bar{d}) \in T^{sd}} y_{p\bar{s}\bar{d}} \leq 1, \qquad \forall\, p \in P,\, s \in S,\, d \in D \tag{2}$$

$$\sum_{p \in P^I} \sum_{s \in S^I} \sum_{d \in D} y_{psd} = 0 \tag{3}$$

$$\sum_{s \in S} \sum_{\bar{d}=d}^{d+\mathrm{D}^{\max}} y_{ps\bar{d}} \leq \mathrm{D}^{\max}, \qquad \forall\, p \in P,\, d \in \{1, 2, \ldots, |D| - \mathrm{D}^{\max}\} \tag{4}$$

$$\sum_{s \in S^N} \sum_{\bar{d}=d}^{d+N^{\max}} y_{ps\bar{d}} \leq \mathrm{N}^{\max}, \qquad \forall\, p \in P,\, d \in \{1, 2, \ldots, |D| - \mathrm{D}^{\max}\} \tag{5}$$

$$\mathrm{MinShifts}^p \leq \sum_{s \in S} \sum_{d \in D} y_{psd} \leq \mathrm{MaxShifts}^p, \qquad \forall\, p \in P \tag{6}$$

$$\mathrm{MinNightShifts}^p \leq \sum_{s \in S^N} \sum_{d \in D} y_{psd} \leq \mathrm{MaxNightShifts}^p, \qquad \forall\, p \in P \tag{7}$$

$$\sum_{(s,d) \in C^p} y_{psd} = 0, \qquad \forall\, p \in P : C^p \neq \emptyset \tag{8}$$

$$y_{p^r sd} \leq (1 - x_r), \qquad \forall\, r \in R,\, (s,d) \in B^r \tag{9}$$

$$y_{psd} \in \{0,1\}\ \forall p \in P,\ s \in S,\ d \in D; \quad x_r \in \{0,1\}\ \forall r \in R \tag{10}$$

Here, (1) ensures that every shift is covered by exactly one resident. (2) guarantees at least 10 hours between consecutive shifts worked for each resident. (3) ensures first-year residents do not work the first (7am) or last (11pm) shifts on any day. (4) limits the number of consecutive days that can be worked. Similarly, (5) limits the number of consecutive nights that can be worked. (6) requires that the total number of shifts assigned to each resident be within a specified range. Likewise, (7) requires that the total number of night shifts assigned to each resident be within a specified range. (8) ensures that a resident who works in a continuity clinic will not work specific shifts before, during, and after working in the clinic. Lastly, (9) links shifts to time-off requests. In particular, 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.

Objective:

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

$$\text{Maximize} \sum_{r \in R} x_r \tag{11}$$

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 $R = \{1, 2, \dots, n\}$, 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 grants 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 grants 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 paper:

- $\mathbf{x} \in \{0, 1\}^n$ 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 $r = 1, \dots, n$. 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,

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

  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 A.

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 set of largest cardinality:

$$\begin{aligned}(\text{NewFeas})_0 \quad \text{maximize} \quad & \sum_{r \in R} x_r && (12)\\ \text{subject to} \quad & \mathbf{x} \in \mathbf{X}^H. && (13)\end{aligned}$$

For the residency scheduling problem we consider, $\mathbf{X}^H$ in constraint (13) corresponds to the set of solutions defined by constraints (1)–(10).

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:

$$
\begin{aligned}
(\text{NewFeas})_1 \quad & \text{maximize} && \sum_{r \in R} x_r && (14) \\
& \text{subject to} && \mathbf{x} \in \mathbf{X}^H && (15) \\
& && \sum_{r \in R \setminus R_0^F} x_r \geq 1. && (16)
\end{aligned}
$$

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 with respect to the original scheduling problem, 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 with respect to the original scheduling problem.

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:

$$\begin{aligned}
(\text{NewFeas})_i \quad & \text{maximize} && \sum_{r \in R} x_r && (17)\\
& \text{subject to} && \mathbf{x} \in \mathbf{X}^H && (18)\\
& && \sum_{r \in R \setminus R_k^F} x_r \geq 1 \quad \forall\, k = 0, \ldots, i-1 && (19)
\end{aligned}$$

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.

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 set of all maximally-feasible request sets identified in the first phase of the algorithm:

$$\begin{aligned}
(\text{NewInfeas})_0 \quad & \text{minimize} && \sum_{r \in R} x_r && (20)\\
& \text{subject to} && \sum_{r \in R \setminus F} x_r \geq 1 \quad \forall F \in \mathbb{F}. && (21)
\end{aligned}$$

(If $\mathbb{F} = \{\emptyset\}$, constraint (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, 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 R_0^I is not a subset of any maximally-feasible request set in $\mathbb{F}$ and $\mathbb{F}$ contains every maximally-feasible request set, R_0^I cannot be a feasible request set, and thus it is an infeasible set. Since R_0^I is also minimal in cardinality, R_0^I is a minimally-infeasible set of requests for the original scheduling problem. 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:

$$\begin{aligned} (\text{NewInfeas})_1 \quad & \text{minimize} \quad \sum_{r \in R} x_r && (22) \\ & \text{subject to} \quad \sum_{r \in R \setminus F} x_r \geq 1 \quad \forall F \in \mathbb{F} && (23) \\ & \qquad\qquad \sum_{r \in R_0^I} x_r \leq |R_0^I| - 1. && (24) \end{aligned}$$

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 with respect to the original scheduling problem, 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 with respect to the original scheduling problem.

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 $\sum\limits_{r\in R_1^I} x_r \leq |R_1^I| - 1$ 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 constructing and solving problems of the form:

$$
\begin{aligned}
(\text{NewInfeas})_j \quad & \text{minimize} && \sum_{r\in R} x_r && && (25)\\
& \text{subject to} && \sum_{r\in R\setminus F} x_r \geq 1 && \forall F \in \mathbb{F} && (26)\\
& && \sum_{r\in R_k^I} x_r \leq |R_k^I| - 1 && \forall\ k = 0,\ldots,j-1 && (27)
\end{aligned}
$$

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 (NewInfeas)$_j$ is infeasible, every minimally-infeasible request set will have been identified, by Theorem 4.

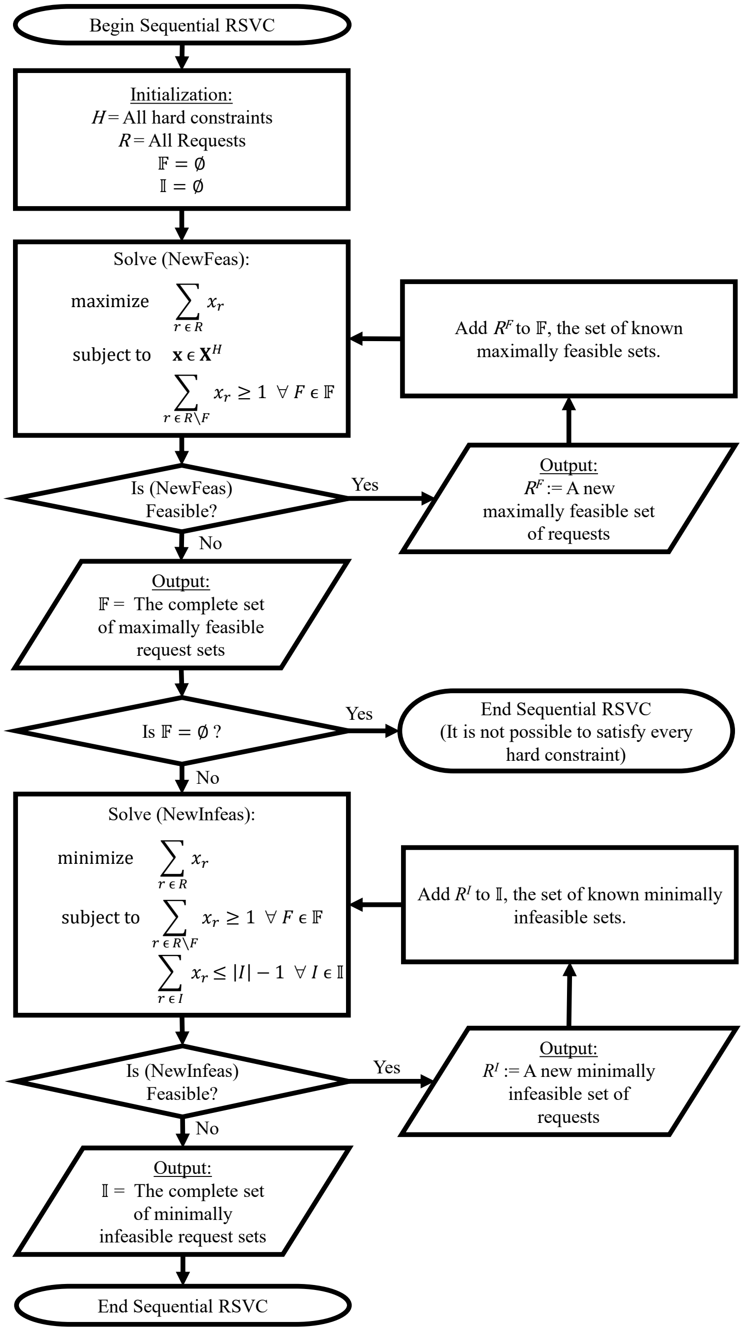

Begin Sequential RSVC
Initialization:
H = All hard constraints
R = All Requests
$\mathbb{F} = \emptyset$
$\mathbb{I} = \emptyset$
Solve (NewFeas):
maximize $\sum_{r \in R} x_r$
subject to $\mathbf{x} \in \mathbf{X}^H$
$\sum_{r \in R \setminus F} x_r \geq 1 \ \forall F \in \mathbb{F}$
Add R^F to $\mathbb{F}$, the set of known maximally feasible sets.
Is (NewFeas) Feasible?
Yes
No
Output: R^F := A new maximally feasible set of requests
Output: $\mathbb{F}$ = The complete set of maximally feasible request sets
Is $\mathbb{F} = \emptyset$?
Yes
No
End Sequential RSVC (It is not possible to satisfy every hard constraint)
Solve (NewInfeas):
minimize $\sum_{r \in R} x_r$
subject to $\sum_{r \in R \setminus F} x_r \geq 1 \ \forall F \in \mathbb{F}$
$\sum_{r \in I} x_r \leq |I| - 1 \ \forall I \in \mathbb{I}$
Add R^I to $\mathbb{I}$, the set of known minimally infeasible sets.
Is (NewInfeas) Feasible?
Yes
No
Output: R^I := A new minimally infeasible set of requests
Output: $\mathbb{I}$ = The complete set of minimally infeasible request sets
End Sequential RSVC

Figure 1: Sequential Request Selection Via Cuts

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 B.

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:

$$\begin{aligned} \text{(FirstFeas)} \quad \text{maximize} \quad & \sum_{r \in R} x_r & \text{(28)} \\ \text{subject to} \quad & \mathbf{x} \in \mathbf{X}^H. & \text{(29)} \end{aligned}$$

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:

$$\begin{aligned} \text{(CandidateSet)} \quad \text{minimize} \quad & \sum_{r \in R} x_r & & \text{(30)} \\ \text{subject to} \quad & \sum_{r \in R \setminus F} x_r \geq 1 & \forall\, F \in \mathbb{F} \quad & \text{(31)} \\ & \sum_{r \in I} x_r \leq |I| - 1 & \forall\, I \in \mathbb{I}. \quad & \text{(32)} \end{aligned}$$

Here, (31) ensures that the candidate set is not a subset of any known maximally-feasible set and (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:

$$\begin{aligned} \text{(FeasTest)} \quad & \text{maximize} \quad && \sum_{r \in R} x_r && (33) \\ & \text{subject to} \quad && \mathbf{x} \in \mathbf{X}^H && (34) \\ & && x_r = 1 \quad \forall\, r \in R^\star. && (35) \end{aligned}$$

Here, (34) ensures feasibility of the schedule and (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 Threorem 7 this will happen after a finite number of iterations, and the algorithm will terminate.

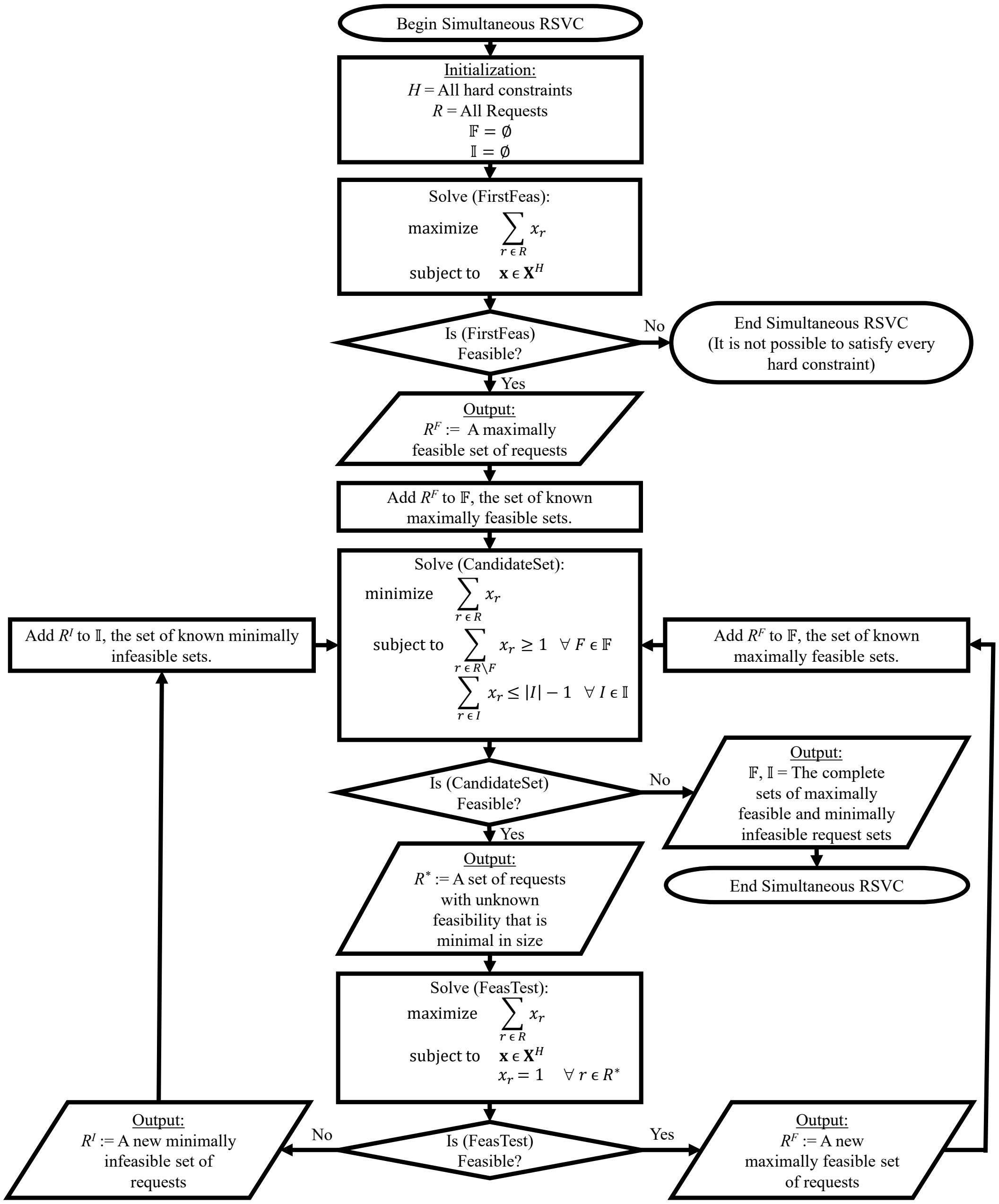

Begin Simultaneous RSVC
Initialization:
H = All hard constraints
R = All Requests
$\mathbb{F} = \emptyset$
$\mathbb{I} = \emptyset$
Solve (FirstFeas):
maximize $\sum_{r \in R} x_r$
subject to $\mathbf{x} \in \mathbf{X}^H$
Is (FirstFeas) Feasible?
No
End Simultaneous RSVC (It is not possible to satisfy every hard constraint)
Yes
Output:
$R^F :=$ A maximally feasible set of requests
Add R^F to $\mathbb{F}$, the set of known maximally feasible sets.
Solve (CandidateSet):
minimize $\sum_{r \in R} x_r$
subject to $\sum_{r \in R \setminus F} x_r \geq 1 \quad \forall F \in \mathbb{F}$
$\sum_{r \in I} x_r \leq |I| - 1 \quad \forall I \in \mathbb{I}$
Add R^I to $\mathbb{I}$, the set of known minimally infeasible sets.
Add R^F to $\mathbb{F}$, the set of known maximally feasible sets.
Is (CandidateSet) Feasible?
No
Output:
$\mathbb{F}, \mathbb{I}$ = The complete sets of maximally feasible and minimally infeasible request sets
End Simultaneous RSVC
Yes
Output:
$R^* :=$ A set of requests with unknown feasibility that is minimal in size
Solve (FeasTest):
maximize $\sum_{r \in R} x_r$
subject to $\mathbf{x} \in \mathbf{X}^H$
$x_r = 1 \quad \forall r \in R^*$
Output:
$R^I :=$ A new minimally infeasible set of requests
No
Is (FeasTest) Feasible?
Yes
Output:
$R^F :=$ A new maximally feasible set of requests

Figure 2: Simultaneous Request Selection Via Cuts

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.

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 from 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).
- **Vacation 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 D 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" is used to describe 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.

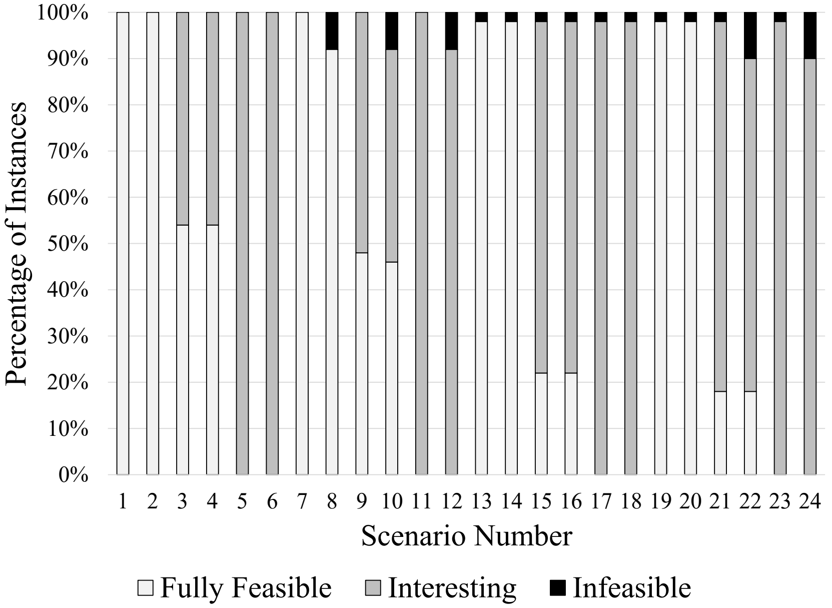


Figure 3: Feasibility of Problem Instances

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 includes all instances 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 includes all remaining "interesting" problem instances. Type 2 instances have greater than 1,000 maximally-feasible sets or greater than 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.

5.3 Type 1 Problem Instances

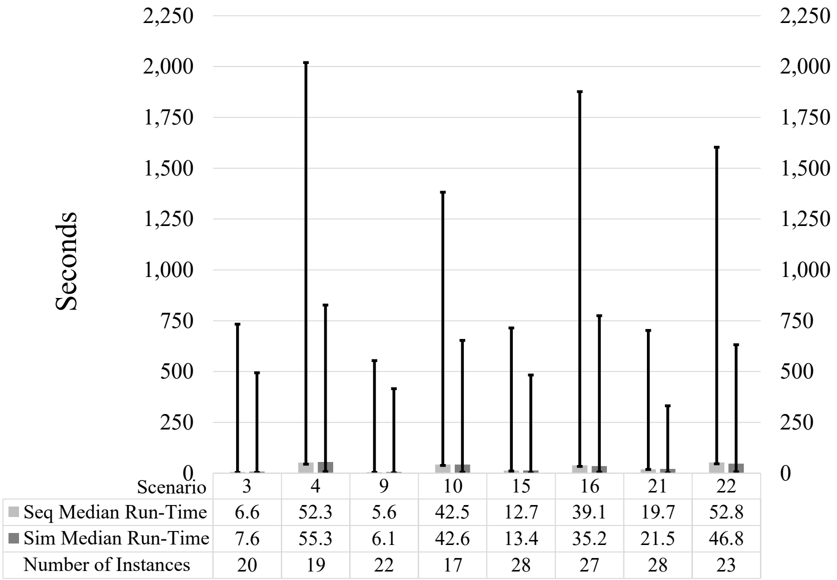
In Table 1, we list the number of Type 1 and number of "interesting" instances (out of 50) for each of the relevant scenarios. We also list the median, minimum, and maximum numbers of maximally-feasible sets (MFSs) and minimally-infeasible sets (MISs) for Type 1 instances.

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

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 vacation 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 in Figure 12 of Appendix D.

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 plot the median, 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.



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: 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 13 and 14 in Appendix D are plots of the solve times for Type 1 instances). In some cases, the Simultaneous method was up to 20 minutes faster than the Sequential method. At first glance, this might seem surprising — the Simultaneous approach solves two optimization problems to yield each new (feasible or infeasible) request set, whereas the Sequential approach solves only one. 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 RSVC algorithm.

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. In this section, we compare the quantity 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 identify 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 (as much as 2.6 hours), and in the remaining cases is typically comparable. We plot the run-times for each Type 2 instance in Figures 15 and 16 of Appendix D.

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 the Simultaneous RSVC algorithm identifies the exhaustive collection of request sets up to 20 minutes faster than 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 Real-World 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 paper) who is responsible for scheduling pediatric residents. For the case study, we solved multiple problem instances 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.

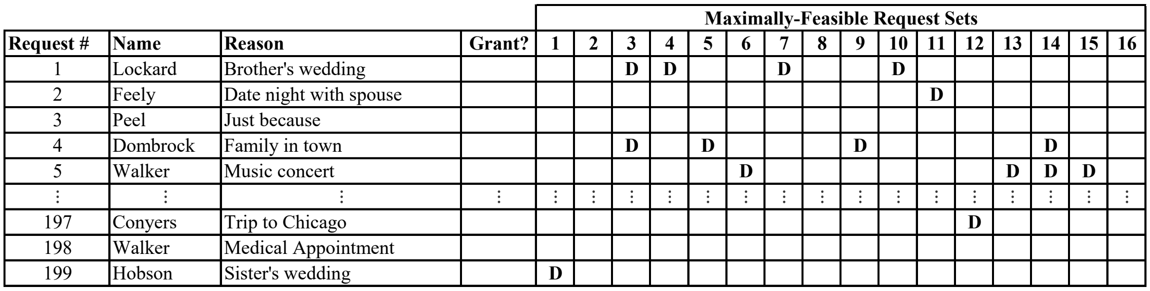
In the second approach, we use our RSVC algorithms to generate the maximally-feasible and minimally-infeasible request sets for the problem instance. The Chief then uses this information to select a schedule.

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

6.1 Case 1

For Case 1, we solved a problem instance 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 additional 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, 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.

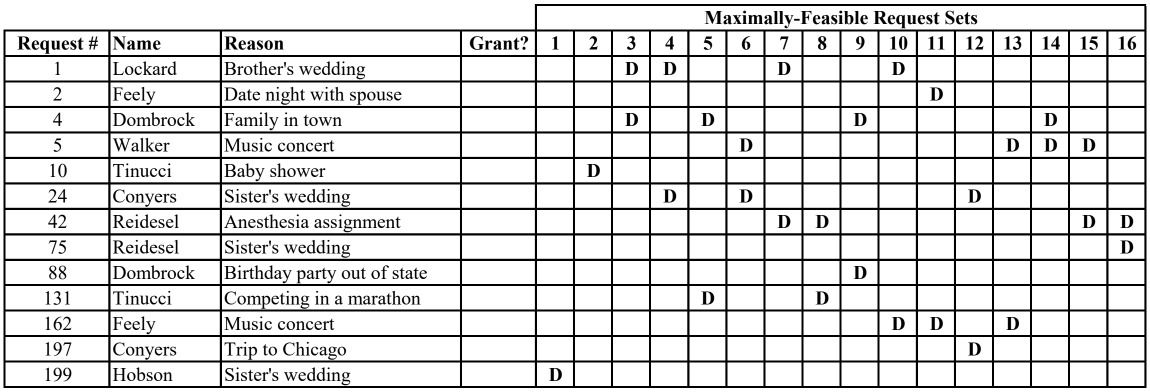
Next, as part of our proposed, alternative scheduling approach, we generated the exhaustive collections of maximally-feasible and minimally-infeasible request sets. In Figure 5 we visually represent every maximally-feasible request set (16 total) and a subset of the 199 requests for this problem instance 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 5

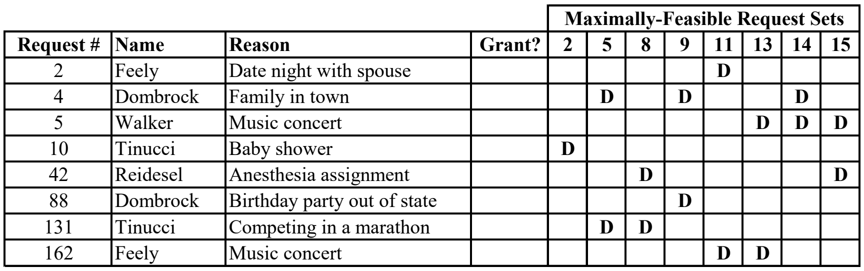
Although this problem includes 199 requests, many of the requests (such as Request #3) are granted in every maximally-feasible set and therefore do not need to be considered when deciding which requests to grant. By hiding all requests that are always possible to grant, as is done in Figure 6, it is easier for schedulers to compare the sets. In Figure 6, the three solutions considered during the traditional scheduling approach are represented by 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 6

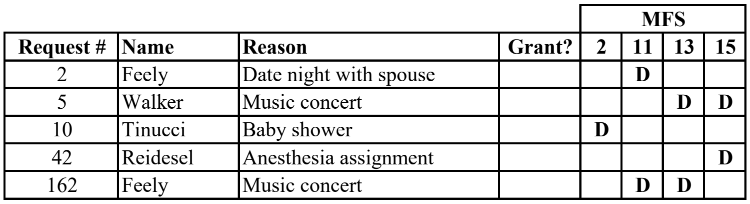
When presented with Figure 6, the Chief first indicated that he wanted to ensure that the four requests involving weddings were granted. By inputting this information into the “Grant?” column, the tool identifies which maximally-feasible sets are no longer an option and hides them from view. Then, each 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 7, 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 7

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 8, 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 8

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 providing the Chief with every maximally-feasible set, he 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. In Case 2, we consider a different problem instance that included many maximally-feasible sets and discuss how the collection of minimally-infeasible sets can be used when selecting a schedule.

6.2 Case 2

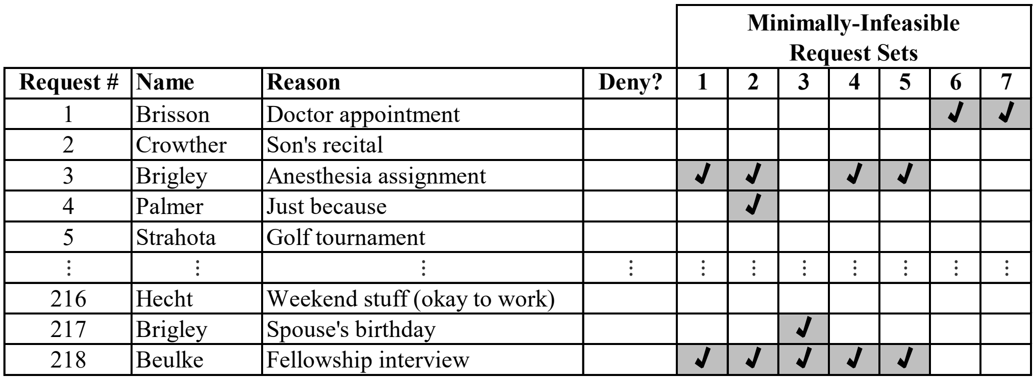
For Case 2, the problem instance included 218 total requests. We started by solving the instance using the traditional approach of generating a solution that grants the maximum number of requests and then adding additional requirements to the problem based on feedback from the Chief. After five iterations 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 quantities, 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 again created a simple visualization tool that helps the scheduler work with the minimally-infeasible sets.

Figure 9 is a picture 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 9, 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 many other columns as well). For example, from Figure 9 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 satisfy 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.

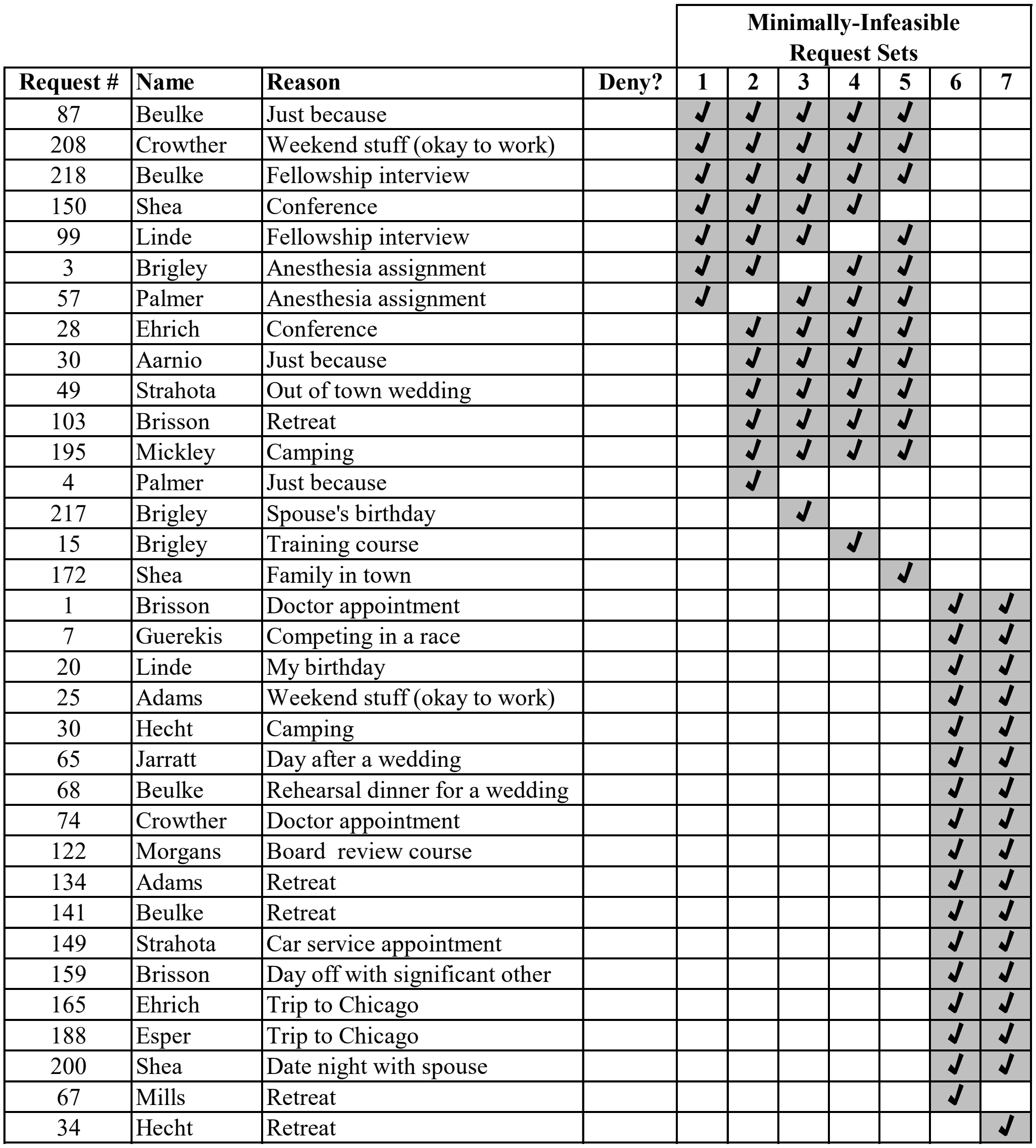


				Minimally-Infeasible Request Sets						
Request #	**Name**	**Reason**	**Deny?**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	Brisson	Doctor appointment							✓	✓
2	Crowther	Son's recital								
3	Brigley	Anesthesia assignment		✓	✓		✓	✓		
4	Palmer	Just because			✓					
5	Strahota	Golf tournament					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

Figure 9

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 10, we can see that only 34 requests need consideration. The other 184 requests can always be granted.

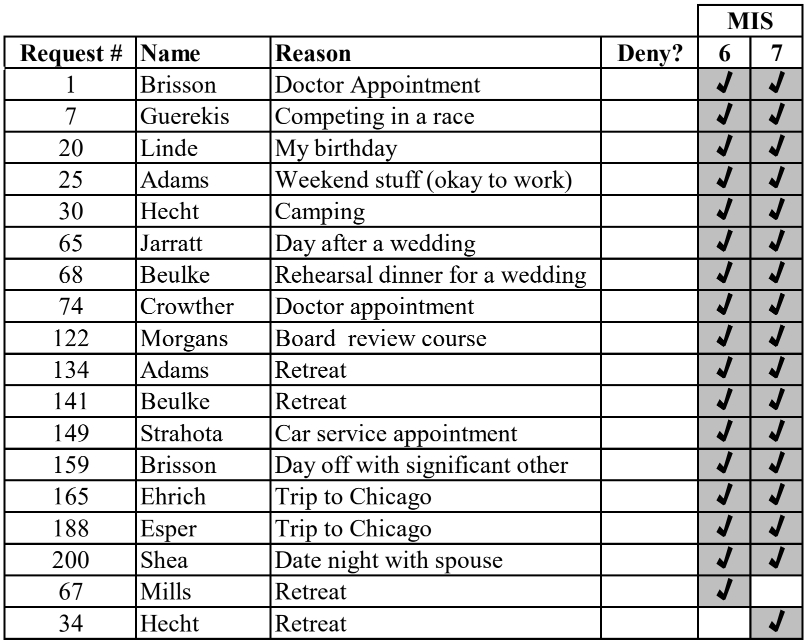
One way to work through the requests and sets represented in Figure 10 is to first look at Minimally-Infeasible Set #1. From it, we can see that at least one of the top seven requests must be denied. When presented with this decision, the Chief indicated that he was willing to deny the first request 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.



				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 10

By updating the tool with this information, it hides each repaired set and each request that is not a part of any remaining minimally-infeasible set, as can be seen in Figure 11. In Figure 11, we can see that the Chief must decide if he prefers to deny any single request from the first 16 requests, or to deny both of the final two requests from Dr. Mills (#67) and Dr. Hecht (#34).



				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			✓

Figure 11

When presented with this decision, the Chief indicated that he preferred denying Dr. Strahota's request for a "Car Service Appointment" since he knew that this could be easily rescheduled. At this point, 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 10. 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 return to the maximally-feasible set visualization tool and select their most preferred set from those that only deny a subset of the requests selected by the schedulers.

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.

6.3 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 relatively few maximally-feasible sets and a process was discussed for using them to select a solution. For Case 2, 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. 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 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. This can be done by using the process explained in Section 6.1. 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 6.2.

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 the 242 requests, and furthermore that it was not possible to grant any additional requests. Like 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.

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.”

7 Conclusion and Future Research

In this paper, we address an important problem that is regularly encountered when scheduling medical residents. Specifically, for resident scheduling problems in which it is not possible 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 chose 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 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 sets to schedulers is that the schedulers can be certain that no better solutions exist.

Although our new method for resident scheduling has numerous benefits over more traditional methods, there are many opportunities for further research and improvements.

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 when 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. Lastly, although the focus of this paper is on requests for time-off in resident scheduling problems, our methods are applicable to any problem involving soft constraints.

Bibliography

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

Amaldi, E. & Kann, V. (1995). The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147(1), 181–210.

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.

Azaiez, M. N. & Al Sharif, S. S. (2005). A 0-1 goal programming model for nurse scheduling. *Computers and Operations Research*, 32(3), 491–507.

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., 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.

Beliën, J. & Demeulemeester, E. (2006). Scheduling trainees at a hospital department using a branch-and-price approach. *European Journal of Operational Research*, 175(1), 258–278.

Berrada, I., Ferland, J. A., & Michelon, P. (1996). A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences*, 30(3), 183–193.

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

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.

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

Cavanagh, S. J. & Coffin, D. A. (1992). Staff tumover among hospital nurses. *Journal of Advanced Nursing*, 17, 1369–1376.

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. (2001). Fast heuristics for the maximum feasible subsystem problem. *INFORMS Journal on Computing*, 13(3), 210–223.

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

Chinneck, J. W. & Dravnieks, E. W. (1991). Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2).

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.

Coomber, B. & Louise Barriball, K. (2007). Impact of job satisfaction components on intent to leave and turnover for hospital-based nurses: A review of the research literature. *International Journal of Nursing Studies*, 44(2), 297–314.

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.

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.

Gauci Borda, R. & Norman, I. J. (1997). Factors influencing turnover and absence of nurses: a research review. *International Journal of Nursing Studies*, 34(6), 385–394.

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.

Hall, R. (2012). *Handbook of Healthcare System Scheduling*. Springer.

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.

Javeri, M. A. (2011). Rotation planning and scheduling for medical resident education. Master's thesis, Purdue University.

Jones, C. & Gates, M. (2007). The costs and benefits of nurse turnover: A business case for nurse retention. *The Online Journal of Issues in Nursing*, 12(3).

Leiter, M. P. & Maslach, C. (2009). Nurse turnover: The mediating role of burnout. *Journal of Nursing Management*, 17(3), 331–339.

Leveck, M. L. & Jones, C. B. (1996). The nursing practice environment, staff retention, and quality of care. *Research in Nursing & Health*, 19, 331–343.

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

Lu, H., While, A. E., & Louise Barriball, K. (2005). Job satisfaction among nurses: A literature review. *International Journal of Nursing Studies*, 42(2), 211–227.

Miller, H. E., Pierskalla, W. P., & Rath, G. J. (1976). Nurse scheduling using mathematical programming. *Operations Research*, 24(5), 857–870.

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.

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.

Sitompul, D. & Randhawa, S. (1990). Nurse scheduling models: a state-of-the-art review. *Journal of the Society of Health Systems*, 2, 62–72.

Smalley, H. K. & Keskinocak, P. (2014). Automated medical resident rotation and shift scheduling to ensure quality resident education and patient care. *Health Care Management Science*.

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. *European Journal of Operational Research*, 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 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.

Yildiz, Z., Ayhan, S., & Erdogmus, S. (2009). The impact of nurses' motivation to work, job satisfaction, and sociodemographic characteristics on intention to quit their current job: An empirical study in Turkey. *Applied Nursing Research*, 22(2), 113–118.

A Sequential RSVC Algorithm

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 $\sum_{r \in R} x_r$ 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: maximize $\sum_{r \in R} x_r$ 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 $\sum_{r \in R \setminus R^F} x_r \geq 1$ to $C_F$
17:     Goto Step 12
18: end if
19: for all $F \in \mathbb{F}$ do
20:     Add the cut $\sum_{r \in R \setminus F} x_r \geq 1$ to $C_I$
21: end for
22: Solve the problem: minimize $\sum_{r \in R} x_r$ 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 $\sum_{r \in R^I} x_r \leq |R^I| - 1$ to $C_I$
27:     Goto Step 22
28: else
29:     End algorithm.
30: end if
```

B Simultaneous RSVC Algorithm

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
```

C Proofs

C.1 Analysis of Phase I of Sequential RSVC

Theorem 1. *Let R_K^F be the set of requests that are satisfied in an optimal solution of* $(\text{NewFeas})_K$ *after iteratively solving the sequence of problems* $(\text{NewFeas})_k$ *for $k = 0, \ldots, K$ 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 $k = 0, \ldots, K-1$. 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 $(\text{NewFeas})_0$. Then, by construction, R_0^F is a feasible set and, because it corresponds to an optimal solution of $(\text{NewFeas})_0$, 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 $(\text{NewFeas})_{K+1}$ 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 $\sum\limits_{r \in R \setminus R_k^F} x_r \geq 1$ 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 with respect to the original scheduling problem, 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 $(\text{NewFeas})_{K+1}$, 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 $k = 0, \ldots, K$. Moreover, since problem $(\text{NewFeas})_{K+1}$ is constructed by adding a cut to problem $(\text{NewFeas})_K$, 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, $k = 0, \ldots, \bar{K}-1$ is the exhaustive list of maximally-feasible request sets for the resident scheduling problem.*

Proof By Theorem 1, the solution to each problem $(\text{NewFeas})_k$ 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 $(\text{NewFeas})_{\bar{K}}$ 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 $\sum\limits_{r \in R \setminus R_k^F} x_r \geq 1$ 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. ∎

C.2 Analysis of Phase II of Sequential RSVC

Theorem 3. *Let R^I_K 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^I_K is a minimally-infeasible request set such that $R^I_K \neq R^I_k$ for $k = 0, \ldots, K-1$. Moreover, $|R^I_K| \geq |R^I_{K-1}|$, 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^I_0 be the set of requests contained in an optimal solution of $(\text{NewInfeas})_0$. By construction, $R^I_0 \not\subseteq F \; \forall F \in \mathbb{F}$, and since $\mathbb{F}$ contains every maximally-feasible set of requests for the scheduling problem, R^I_0 is an infeasible request set. Since R^I_0 is also minimal in the number of requests it contains, R^I_0 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^I_{K+1} 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^I_{K+1} is an infeasible request set for the scheduling problem. Additionally, each cut of the form $\sum\limits_{r \in R^I_k} x_r \leq |R^I_k| - 1$ for $k = 0, \ldots, K$ eliminates exactly those solutions that only contain the requests in R^I_k or a proper superset of the requests in R^F_k. Since any solution that only contains a proper superset of the requests in R^I_k is not minimally infeasible with respect to the scheduling problem, R^I_k is the only minimally-infeasible request set that is eliminated from the feasible solution space by each cut. Therefore, R^I_{K+1} is a minimally-infeasible request set for the scheduling problem such that $R^I_{K+1} \neq R^I_k$ for $k = 0, \ldots, K$. Moreover, since problem $(\text{NewInfeas})_{K+1}$ is constructed by adding a cut to problem $(\text{NewInfeas})_K$, its optimal value cannot be better (i.e., smaller), we can conclude that $|R^F_K| \leq |R^F_{K+1}|$. ■

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

Proof If problem $(\text{NewInfeas})_0$ 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 $(\text{NewInfeas})_0$ is feasible.

By Theorem 3, the solution to each problem $(\text{NewInfeas})_k$ 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 $(\text{NewInfeas})_{\bar{K}}$ 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 $\sum\limits_{r \in R^I_k} x_r \leq |R^I_k| - 1$ eliminate exactly those solutions that only contain the requests in R^I_k or a superset of the requests in R^I_k, 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. ∎

C.3 Analysis of Simultaneous RSVC

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 (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 (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 (31) or (32), but will not violate constraints of both types. Recall that if $\tilde{x}$ violates a constraint of type (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 (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 (32), i.e., it violates a constraint of the form $\sum\limits_{r \in I} x_r \leq |I| - 1$ 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 $\sum\limits_{r \in I} x_r \leq |I| - 1$, 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 (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 (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 (31) removes precisely those solutions that correspond to feasible sets that are contained in some $F \in \mathbb{F}$, and every constraint of type (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. ■

D Additional Tables and Figures

Table 2: 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 Vacation 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

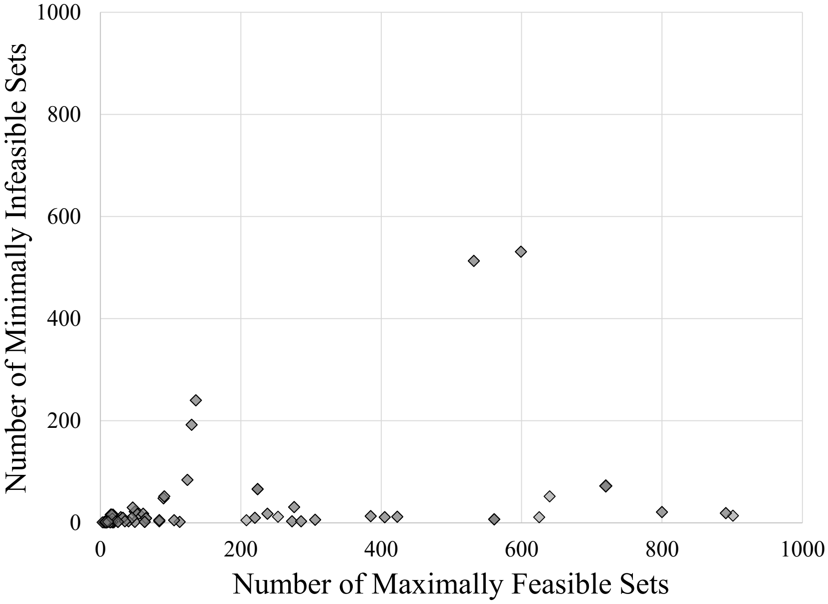


Figure 12: Numbers of Maximally-Feasible and Minimally-Infeasible Sets

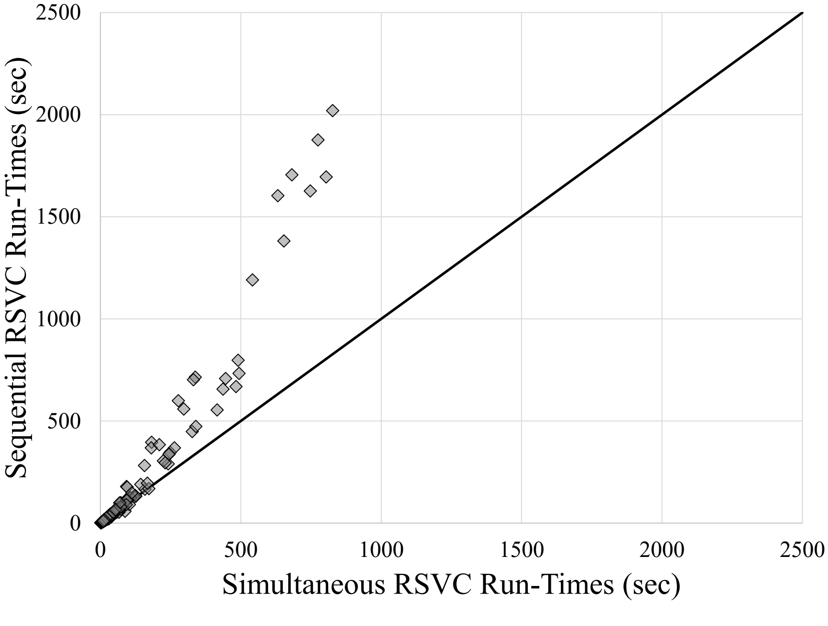


Figure 13: Algorithm Run-Time Comparison for Type 1 Problem Instances

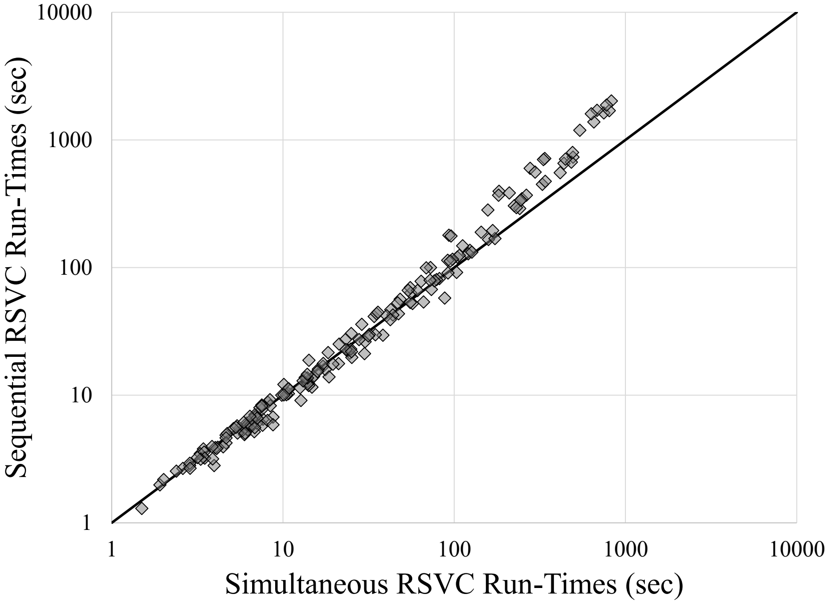


Figure 14: Algorithm Run-Time Comparison for Type 1 Problem Instances (Logarithmic Scaling)

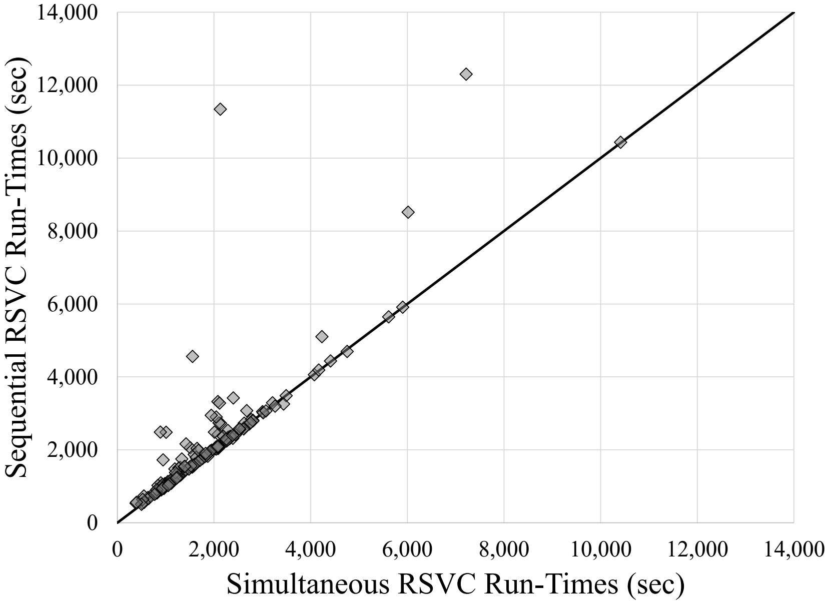


Figure 15: Algorithm Run-time Comparison for Type 2 Problem Instances

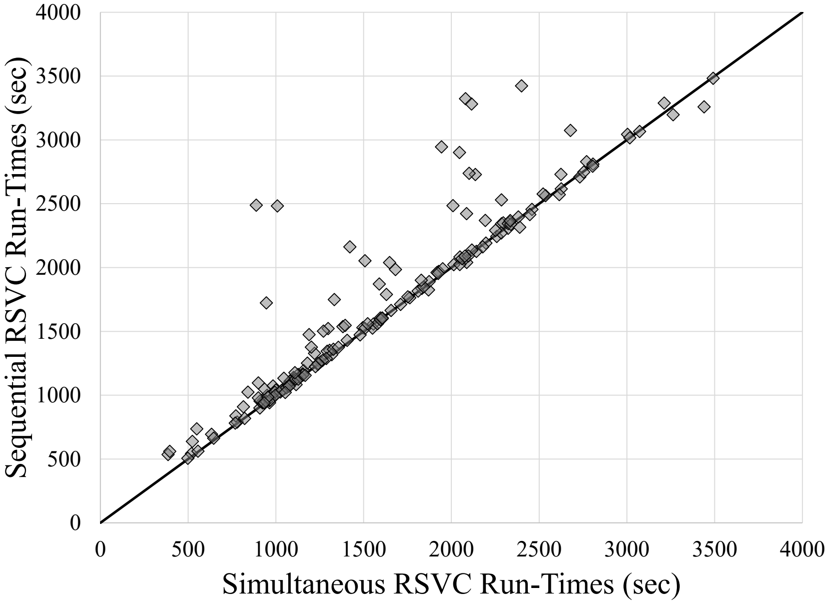


Figure 16: Algorithm Run-time Comparison for Type 2 Problem Instances (Zoomed)