Essays on Scheduling Models in Service Systems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

FEI LI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DIWAKAR GUPTA

July, 2014

Acknowledgements

I am deeply indebted to my advisor Professor Diwakar Gupta, who brought me great opportunity to dig into several interesting research topics, and has provided me consistent supports and encouragements. Thanks to him, I had the great fortune to pursue my Ph.D. study at the University of Minnesota. Besides knowledge and skills that he passed to me and his other students, I am greatly inspired by Professor Gupta's attitude and passion in research. On many occasions his passion helped us find new paths when the research seemed to be at a dead end, and led to findings that I had not expected.

I would like to express my sincere gratitude to my thesis committee members, Professors John Carlsson, Shuzhong Zhang and Henry Liu, for their careful reading and valuable comments on my dissertation. I have also received great instructions and advices from Professor John Carlsson in my research.

My gratitude extends to my officemates: Hao-Wei Chen, Wen-Ya Wang, Zhi Zhang and Yibin Chen. Elder officemates Hao-Wei and Wen-Ya have given me a lot of support in the early years of my Ph.D study when I had a struggling time. And the support and numerous fruitful discussions with fellows Zhi Zhang and Yibin Chen have stimulated a lot of exiting ideas in work.

I am greatly indebted to my parents, Yijian Li and Yi Luo, who always stand there and give me unlimited love and support.

This dissertation is devoted in part to my previous advisors, Professor Baoding Liu in Tsinghua University, and Professor Bharath Rangarajan who had left University of Minnesota. I could never start my Ph.D. study without their great help.

The material reported in this thesis is based upon work supported in part by an award from the National Science Foundation under grant no. CMMI-1332680 (PI: Diwakar Gupta).

Dedication

This work is dedicated to my parents.

Abstract

Scheduling has been a fundamental area in Operations Research and is receiving increasing attention. Growing scale of operations and increasing availability of data in different industries drive the need for efficient and practical solutions for scheduling resources under customized circumstances. In this thesis, we address three different scheduling problems that come from transit industry and healthcare industry respectively. According to the special features encountered in each industry, we build fixed job scheduling models for the reserve driver scheduling and work assignment problems for transit industry, and resource-constrained bin packing models for the surgery rescheduling problems in healthcare. Three separate but related chapters constitute the main body of the thesis.

Among the three models, two models are deterministic and are proved to be NP-hard. The other model is an online version of the reserve driver work assignment problem. Our target is to provide algorithms that run in polynomial or pseudo-polynomial time and can beat the best-known algorithm in terms of worst-case performance guarantee. For the offline reserve driver scheduling and work assignment problem, we provide an algorithm with approximation ratio between $[1 - 1/e, 19/27]$; and for the online reserve driver work assignment problem, we build a randomized algorithm with $O(\log \Delta)$ competitive ratio, where Δ is the ratio of the longest to the shortest job in duration. For the surgery rescheduling problem the model is not widely studied and we are the first to provide an algorithm and a lower bound with performance guarantee—the worst-case performance guarantee is 3/2 for the approximation algorithm, and 2/3 for the lower bound.

We not only are interested in theoretical results but also care about practical use of our algorithms. All algorithms are experimented with real data and we benchmark with either the current industry performance or a greedy algorithm/policy.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Resource scheduling problems arise in almost all service industries. This thesis focuses on two problems from the transit industry and one from the healthcare industry. The resources being scheduled are reserve drivers in the transit industry setting and operating rooms in the health care setting. The objective of scheduling, broadly defined, is to match demand with availability of resources in a fashion that minimizes operating costs while meeting a variety of constraints. Constraints can arise from problem setting, e.g. off-line or online, from start and end times, e.g. fixed or variable, as well as from scheduler ability to preempt previous assignments. Problem settings can also vary in terms of identical or different resource requirements, and the latter may be either deterministic or random.

There is a huge literature on scheduling topics. This thesis consists of three self-contained chapters on three different scheduling models. Each chapter is written as a separate paper and contains its own motivation as well as literature review. A common feature of all three studies is our focus on providing implementable algorithms with worst-case performance guarantees. Chapters 2 and 3 develop solutions for the reserve drivers' scheduling problem for large transit agencies. It uses data from a collaborating transit agency. Chapter 4 focuses on a model for improving surgery schedules. It utilizes data from three hospitals. We briefly summarize the contents of each chapter in the remainder of this introduction.

Reserve Driver Scheduling and Work Assignment: Day-before

When open work caused by unplanned events such as bus breakdowns, inclement

weather and driver (operator) absenteeism needs to be covered by reserve drivers, an instance of the operational fixed job scheduling problem or interval scheduling problem arises. Jobs may not be preempted once assigned. That is, each work piece, which is referred to as a job, requires one operator who must work continuously between specified start and end times to complete the job. According to work rules, each reserve operator may be assigned up to w hours of work, which may not to be continuous so long as the total work time is within a s-hour time window of that operator's shift start time. Parameters w and s are called allowable work-time and spread-time, respectively.

Our decisions are operators' shift start times and assignments of each piece of work while honoring work-time and spread-time constraints, such that the amount of work covered as part of regular duties is maximized. This problem is solved one day before each day of operations and concerns known pieces of work.

In Chapter 2, we establish the mathematical model for the day-before reserve operator scheduling and assignment problem, and argue that the problem is NP-hard. Next, we present different heuristic approaches for solving the problem, and analyze their worst-case performance ratio. We present numerical experiments using data from a large transit agency, which show that the average performance of the decomposition algorithm is good when applied to real data.

Reserve Driver Scheduling and Work Assignment: Day-of

In Chapter 3, we consider online reserve driver work assignment during the day of operations. During the day, open work pieces open need to be assigned either to reserve drivers or to overtime in an online fashion. That is, assignment decisions must be made sequentially without information about future job requests and the scheduler may need to select a particular driver when multiple drivers can perform a job.

The objective is to maximize the amount of work covered as part of regular duties. Note that in the online problem we are given drivers with fixed shift start and end times. Different from the day-before problem, we do not decide the shift start times and only decide which of the available drivers to assign each piece of work.

We propose a randomized online algorithm that carries a performance guarantee relative to the best offline solution and simultaneously performs better than any deterministic algorithm. We also provide numerical experiments with both real data and randomly generated instance to show the performance of our algorithm.

Surgery Rescheduling: Chapter 4 considers rescheduling of non-urgent surgeries in order to reduce the number of ORs concurrently staffed during the day.

In healthcare industry, doctors, nurses, and other staff work together to assure quality and efficiency of patients' care, hence scheduling is very important and complicated. Operating rooms (ORs) in US hospitals generate about 70% of a hospital's revenues. Surgery schedules are made through a complicated process. Many hospitals allocate blocks of OR time to individual or groups of surgeons as guaranteed allocation, who book surgeries one at a time in their blocks. The booking procedure frequently results in unused time between surgeries. Realizing that this presents an opportunity to improve OR utilization, hospitals manually reschedule surgery times one or two days before each day of surgical operations, in order to decrease OR staffing costs, which are mainly determined by the number of concurrently staffed ORs.

We formulate the rescheduling problem as a variant of the bin packing problem with interrelated items, which are the surgeries performed by the same surgeon. We develop a lower bound (LB) construction algorithm and prove that the LB is at least (2/3) of the optimal staffing cost, for the cases with one or two shift lengths. Our analytical results form the basis of a branch-and-bound algorithm. Besides the theoretical analysis, results from numerical experiments are provided at the end of Chapter 4.

In this thesis, each chapter has a unique notation, which is defined within that chapter. Also, the metrics for worst-case performance are not the same across all chapters. Instead, according to the type of the model, we define the metrics differently in each chapter.

Summary of Contributions: We summarize the key contributions of the following three Chapters here. In the Chapter "Reserve Driver Scheduling and Work Assignment: Day-before" we introduce the best-known polynomial-time algorithm for operational fixed job scheduling problem with spread and work time constraints and fixed number of shift splits, and we show that its approximation ratio is within $[1 - 1/e, 19/27]$. In the Chapter "Reserve Driver Scheduling and Work Assignment: Day-of" we extend the previous best-known result for single-processor online fixed job scheduling problem under the setting of known job duration bounds. The proposed single-processor randomized algorithm has competitive ratio of $O(\log \Delta)$. Moreover, we provide the first approach of multiple-processor online FJS with different processor remaining times and

the algorithm preserves the competitive ratio of $O(\log \Delta)$. In the Chapter "Surgery Rescheduling", we dig deep into the bin packing problem with interrelated items, and provide (2/3) lower bounds with both single bin size and two bin sizes. Based on this result we develop effective B&B algorithm for surgery rescheduling problems.

Roadmap Comments: Before proceeding to the three Chapters that constitute the main body of this thesis, several things need to be kept in mind. First, notation is unique and redefined in each chapter. Second, the metrics of worst-case performance are not uniform. Instead, according to the type of the models and norms adopted in the literature, we define the metrics differently in each chapter.

Chapter 2

Reserve Driver Scheduling and Work Assignment Problem: Day-before

2.1 Introduction

The fixed job scheduling problem (FJS), introduced in [1], concerns the optimal assignment of jobs to operators, where each job has a fixed start time and a fixed end time, and each operator can process at most one job at a time ([2]). Instances of FJS arise in many applications. For example, the problem of scheduling aircraft maintenance jobs that are required to be completed within fixed time windows ([3]), and the problem of scheduling bus operators ([4]) are both instances of the FJS.

There are two broad categories of FJS: *tactical* and *operational*. In the tactical FJS (denoted as TFJS), the objective is to determine the minimum number of operators needed to cover all jobs. There is no a priori limit on the number of available operators. In contrast, in the operational FJS (denoted as OFJS), the number of available operators is fixed and the objective is to maximize a total reward. In this setting, the assignment of a job to an available operator produces a reward (usually proportional to its duration), whereas jobs that remain unassigned do not produce a reward.

Spread-time and work-time constraints are two major types of constraints in FJS

problems. Spread-time is the maximum time span of an operator's workday. That is, given a spread-time limit s, if an operator is scheduled to start work at t_0 on a particular day, then she or he may be assigned work between t_0 and $(t_0 + s)$, but not outside this interval of time. Work-time is the maximum amount of time that an operator may be required to work within the allowed spread time. We denote work-time limit by w. Operators may voluntarily choose to work more than w hours each day, but in such cases they receive overtime pay. We affix letters S and W to TFJS or OFJS to identify problem instances with appropriate constraints. For example, OFJS-S represents the OFJS problem with spread-time constraints only (i.e. $w = s$), and TJFS-W means the TFJS problem with work-time constraints only (i.e. w is finite but s can be arbitrarily large), and OFJS-WS means the OFJS with both types of constraints (i.e. $w \leq s < \infty$). Note that w cannot exceed s if a spread-time limit is specified.

This paper is motivated by an instance of the OFJS-WS that arises in the context of extraboard bus operator (equivalently, reserve bus driver) scheduling and work assignment at a large transit agency. The work rules require that the agency may not assign more than 8 hours of work to operators within a 12-hour spread. Assignments that violate these rules are counted as overtime[1] , which may be accepted by the operator on a voluntary basis. Such rules are common in the transit industry and call for methodologies to solve OFJS-WS because neither OFJS-W nor OFJS-S can provide satisfactory solutions for problems of practical interest.

Extraboard operators are not assigned regular duties in advance, but cover work that arises because of planned and unplanned time off, bus breakdowns, weather, and special events such as a state fair or a major league game. On their duty days, extraboard operators are paid wages for a full shift (typically 8 hours) regardless of how much work is actually assigned to them within their work hours. Open work that is not covered by extraboard operators is assigned to operators who indicate their willingness to work overtime. If neither extraboard nor overtime operators are available to cover a piece of work, then that results in dropped service. [5] show that the size of extraboard workforce including vacation coverage can be as high as 26% of the total workforce size for large transit systems. Because labor costs are a significant portion of the total cost of providing transit services, it is important for transit agencies to utilize the extraboard

[1] The hourly overtime rate is higher than regular hourly wages.

operators efficiently.

The assignment of work to extraboard operators typically occurs in two stages. In the first stage, which we model in this paper, a dispatcher assigns open work to extraboard operators one day before the day on which open work needs to be performed. Extra work also arises during a day, which is assigned dynamically to either available extraboard operators or to overtime operators. Such problems belong to the class of online scheduling problems. We focus on the day-before problem and do not consider the day-of problem in this paper because the two problems require different solution methodologies. The latter is a topic of ongoing research efforts by the authors. Transit agencies set aside a subset of extraboard operators who are used exclusively for day-of assignments (referred to as on-call duty). For this reason, we also do not model the impact of day-before assignments on the transit agency's ability to meet the day-of demand.

Specifically, we are concerned in this paper about the report times of extraboard operators and the assignment of jobs to operators, which occurs a day before each day's start of operations. We use the term report times to mean start of work shifts. Report times of extraboard operators may be different each day and they are finalized one day before. Given a set of open jobs that are known one day before, our objective is to maximize the amount of work assigned to extraboard operators during their regular shifts by choosing shift start times and deciding which pieces of work to assign to which extraboard operators. We assume that pieces of work that are not assigned to extraboard operators are performed by bus drivers in overtime. Because ample availability of overtime was observed in data from a collaborating transit agency, we do not model cases in which service may be dropped.

A different way to understand the scope of the problem we study in this paper is to place it within the hierarchy of extraboard workforce planning and management problems consisting of operational, tactical and strategic levels; see, for example, [6]. Within this hierarchy, we focus on dispatch decisions that belong to the lowest — i.e. the operational level. Examples from the other two levels include extraboard workforce sizing, run cutting methods and the determination of the daily number of operators who would be scheduled to serve as extraboard. [7], [8] and [9] contain additional institutional background on workforce management challenges in the context of transit

operations.

Instances of the OFJS-WS problem arise in many application areas and are of general interest to operations engineers and managers. First, we draw attention to the fact that all types of public transportation operations need extra operators to take care of contingencies and avoid gaps in service. Examples include bus, rail, ferry and passenger airline operations. Work assignment problems similar to what we study in this paper arise in each of these settings. In addition, OFJS-WS problems arise in the context of periodic batch scheduling of jobs on parallel machines. Because jobs and machines could represent different entities in different application areas, there are numerous applications of the OFJS-WS. For example, jobs could be groups of orders that need machining or repair, or deferrable surgeries that need operating room time, or computer programs that need processor time.

OFJS-W, OFJS-S and OFJS-WS are all NP-hard problems, which makes it difficult to compare the difficulty of solving each problem. Consider, for example, the OFJS-S problem. [10] show that this problem is NP-hard by arguing first that one can solve any instance of the TFJS by repeatedly solving the same instance of the OFJS with $1, \cdots, m$ operators, where m is the number of jobs. In the previous sentence, the words "same instance" mean an instance of the problem with the same set of jobs, and w and s, if w and s are specified. The NP-hardness of OFJS-S then follows from [11]. Therefore, to explain the need for focusing attention on OFJS-WS, we present two arguments. First, OFJS-WS is different from OFJS-W because it also considers spread constraints. It is also different from OFJS-S because the work-time constraints limit the amount of work that may be assigned within a spread. Therefore, solutions methods developed in previous studies, discussed below, are not applicable to OFJS-WS. Second, OFJS-S and OFJS-W are both special cases of OFJS-WS and only the latter captures the actual constraints faced by transit agencies.

[2] provide a review of the fixed job scheduling literature, which is also referred to as the interval scheduling problem. The authors divide papers into four groups based on model features and objective. These categories are as follows.

(1) All jobs must be performed and the objective is to minimize the number of machines used.

(2) The number of machines is fixed and the objective is to maximize total weight

of jobs assigned.

(3) Job start times are not fixed, and the objective is either to maximize total weight or number of jobs, or to minimize the number of machines used to cover all jobs.

(4) Jobs are scheduled online (one at a time) or previously scheduled jobs may be preempted, and the objective is either one of the two objectives mentioned in (3).
Our study falls into the second group above. In what follows, we discuss key papers belonging to groups one, two and four. We do not discuss papers belonging to group three because fixed start and end times of jobs is an important feature of OSJF-WS and methods that do not assume fixed start/end times are not relevant in our setting.

Significant contributions in the first group include [11], [12], [13], and [4]. [11] show that the TFJS-S problem is NP-hard. [12] study the TFJS-W, show that it is NP-hard, prove a 2-upper-bounding property of its preemptive version, and provide a branch-and-bound algorithm to solve it. [13] propose several approximation algorithms for solving different versions of the TFJS, including greedy algorithms and preemption-based algorithms. [4] study the bus driver scheduling problem, which can be considered as an instance of the TFJS with work and spread-time constraints, relief point constraints, as well as other work rules, but do not provide an algorithm with guaranteed approximation ratio. Because the tactical version of FJS problem is different from the operational version, the above algorithms do not apply to the extraboard driver scheduling problem we consider. In addition, there have been a variety of applied papers on the topic. For example, [14] list different objective functions that transit agencies try to optimize and summarize heuristics that have been applied to these problems, including the greedy randomized adaptive search procedure ([15]) and genetic algorithms ([16]). However, none of these algorithms provides an approximation ratio for OFJS-WS problem instances, which is the focus of this paper.

Next, we consider the papers in the second group. [17], consider scheduling n jobs with fixed start and end times to k non-identical machines with the goal of maximizing the value of all jobs assigned (value could be duration). When machines are identical, i.e. each job can be processed by any machine, the authors argue that the problem can be solved in $O(n^2 \log n)$ time. When machines are not identical, i.e. each job can be processed by a subset of machines[2] , the authors provide an exact algorithm that runs in

[2] Note, processing ability may be the result of available time of each machine.

$O(n^{k+1})$ time. The main difference is that this formulation assumes machine availability (i.e. shift start and end times in our setting) is known and that machines do not have both work and spread time constraints. We infer the latter implicit assumption from the fact that the authors assume the subset of machines that can process each job is known. This is only possible when $w = s$ in our setting. Spread and work time constraints are two features of our model that are simultaneously important in our setting, which makes our problem formulation different from that in [17].

The second group of papers is also related to the k-track assignment problem, in which k machines (possibly with different spreads) are given and the objective is to schedule the maximum number of jobs with fixed start and end times. [18] give an $O(n^{k-1}k!k^{k+1})$-algorithm to solve the standard k-track problem with n jobs and k identical machines. [19] provide an optimal online algorithm for the k-track assignment problem with identical time windows. [20] provide an online greedy algorithm that guarantees to lose no more than $(k-1)$ jobs relative to the optimal schedule. However, these algorithms do not apply to our setting because the k-track assignment problem maximizes the total number of jobs assigned, not the total weight or duration of assigned jobs. The solution to the k-track assignment problem would be useful in our setting if all jobs had the same weight. That is not the case. Also, we need to consider both spread and shift time constraints of operators, which makes are problem setting different.

Many researchers have proposed algorithms for solving variants of the FJS problem. It is therefore appropriate to ask if these methods can be adapted to solve the OFJS-WS. We argue next that straightforward adaptation will not work for the OFJS-WS problem. Consider, for example, the greedy heuristic included in [13], which is shown to have an approximation ratio of 3. Although a greedy approach is a reasonable approach for solving the TFJS, it can be arbitrarily bad compared with the optimal solution if applied to certain instances of the OFJS. We present arguments to support our claim at a later point in this paper. Similarly, if we were to adapt the branch-and-bound algorithm from [10], we would encounter a combinatorial number of starting nodes, which would limit the suitability of such approaches when the size of the extraboard workforce is large.

As yet another example, consider the branch-and-price approach in [21], which was

used to solve the OFJS-S. This algorithm requires the ability to repeatedly solve the one-operator instance of the OFJS-S. Unfortunately, this approach is not suitable for the OFJS-WS because as we show later in this paper, the OFJS-WS with one-operator is NP-hard. We also introduce the notion of limited shift splits under which the one-operator case can be solved in polynomial time. Even with the limited shift split requirement in place, the approach would be computationally demanding, requiring $O(m^5)$ operations rather than the $O(m^2)$ operations needed to solve the OFJS-S version of the problem ([21]).

More recently, some researchers have focused on online scheduling problems belonging to the fourth group. [22] consider both OFJS-S and TFJS-S problems. For OFJS-S, the authors provide a randomized algorithm with expected reward at least $(1 - 1/e)$ of the optimum, but this performance is not guaranteed in every run of the algorithm. Our decomposition algorithm can be applied to OFJS-S and it is a deterministic algorithm. That is, its performance does not vary for the same problem parameters and it guarantees a performance of at least $(1 - 1/e)$ of the optimum every time it is applied. We believe ours is a more implementable and stronger result for the transit agencies' problem setting.

In this paper we first show that the OFJS-WS is NP-hard. Then we provide three heuristics for solving the OFJS-WS. The first heuristic is a duration-first greedy algorithm, which assigns the longest unassigned job at each step. We show that the greedy algorithm's approximation ratio is zero. We next show that the preemptive and partial credit version of the OFJS-S is solvable in polynomial time. Combining this result and an algorithm provided by [3], we construct a two-stage algorithm. The third algorithm solves the OFJS-WS with limited shift splits. We first establish that the one-operator case of the OFJS-WS with limited shift splits is polynomially solvable, and then prove that a decomposition approach based on maximizing one operator's assignment at a time has an approximation ratio that lies in $[1 - 1/e, 19/27]$. This algorithm and the approximation ratio also apply to the OFJS-S. Thus, our third algorithm improves upon a result reported in [11] about the approximation ratio of an algorithm designed to solve the OFJS-S.

The contribution of this paper is three fold: (i) it presents three algorithms for solving the OFJS-WS, (ii) it establishes approximation ratios for the recommended

decomposition based algorithm, and (iii) it uses real data from a large transit agency to compare the three algorithms. Its methodological novelty lies in developing heuristic methods, analyzing the limited shift splits version of the OFJS-WS that is observed in practice, and establishing a deterministic $(1 - 1/e)$-approximation algorithm for that case.

The organization of the remainder of this paper is as follows. In Section 2.2 we introduce a mathematical formulation of the OFJS-WS problem and establish complexity results. In Section 2.3 we introduce three heuristics for solving the OFJS-WS and investigate whether approximation ratios can be provided in each case. We present numerical experiments that utilize data from a large transit agency in Section 2.4 and conclude the paper in Section 4.8.

2.2 Model Formulation, Complexity and Special Cases

While OFJS-WS is a broad class of problems with many variants, our model formulation and solution methods are motivated by the application domain of extraboard drivers scheduling. For example, we assume that w and s take reasonable finite values and that operators do not take too many (unpaid) splits during a day. That is, our algorithm is allowed to split the working time into only a limited number of pieces within the spread. Additional modeling assumptions are included in this Section. For the version of the OFJS-WS of practical interest, we provide a $(1 - 1/e)$-approximation algorithm that runs in polynomial time.

In this section we present a mathematical formulation of an instance of the OFJS-WS, establish its complexity, and identify special cases that can be solved in polynomial time. We assume that jobs are sorted by start time. The notation used in describing a formal model is presented in Table 4.5, and key assumptions of the model are as follows.

$(0,\, t_{max})$	= start and end time of a day of operation. At the collaborating transit agency, daily operations started at 3:30 AM and ended at 2:00 AM the following day.
$t \in \{0, \cdots, t_{max}\}$	= time index.
n	= number of operators.
$i \in \{1, \cdots, n\}$	= operator index
m	= number of jobs
$j \in J = \{1, \cdots, m\}$	= job index; J = job index set
(s_j, e_j)	= start and end times of job j, with $s_1 \leq s_2 \leq \cdots \leq s_m$
w	= work-time limit
s	= spread-time limit
$d_j = e_j - s_j$	= duration of job j
$I_j(s)$	= set of jobs that cannot be assigned to the same operator who performs job j = $\{k > j :\ s_k < e_j$ or $e_k - s_j > s\}$
x_{ij}	= binary decision variables; $x_{ij} = 1$ if job j is assigned to operator i, and 0 otherwise

Table 2.1: Notation Used in Formulation

Assumption 1:

Time is discrete.

Assumption 2:

Operators are identical in skill. That is, any operator can perform any job.

Assumption 3:

All operators are subject to the same work-time and spread-time limits, denoted by w and s, respectively.

Assumption 4:

Parameter values belong to the following ranges: $1 \leq s \leq t_{max}$ and $d_j \leq w \leq s$, for all $j \in J$. This means that no job takes longer than the regular shift length of an operator.

Assumption 5:

A job that is not covered by available extraboard operators during their regular work time is assigned on an overtime basis. The cost of overtime is proportional to the duration of the job assigned in overtime.

The above assumptions were supported by data from the collaborating transit agency. For example, 1-minute was the smallest unit of time and jobs whose lengths exceeded the work-time limit of 8 hours were assigned first to those operators who were willing to accept overtime. Assignment of such jobs thus occurred independently of the extraboard operator scheduling and work assignment problem addressed in this paper.

In Table 4.5, $I_j(s)$ is the set of jobs that are incompatible with job j, for each $j \in J$. It contains indices of all jobs that would either overlap with job j or violate spread-time

constraints if offered to the same operator.

We are now ready to present a formulation of the OFJS-WS.

$$z = \text{Max}_{\{x_{ij}\}} \sum_{1 \le j \le m} d_j \sum_{1 \le i \le n} x_{ij} \tag{2.1}$$

subject to:

$$\sum_{1 \le i \le n} x_{ij} \le 1, \quad j = 1, \cdots, m \tag{2.2}$$

$$x_{ij} + x_{ik} \le 1, \;\; j = 1, \cdots, m-1, \; i = 1, \cdots, n, \; k \in I_j(s) \tag{2.3}$$

$$\sum_{1 \le j \le m} d_j x_{ij} \le w, \quad i = 1, \cdots, n \tag{2.4}$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, \cdots, n, \; j = 1, \cdots, m \tag{2.5}$$

The objective function (2.1) maximizes total duration of assigned jobs. Recall that the OFJS-WS comprises of dispatch decisions that arise after the number of operators is determined. That is, the wages of extraboard operators are sunk. Therefore, it makes sense to maximize the total amount of work assigned to extraboard operators, which is equivalent to minimizing overtime.

Constraints (2.2) ensure that each job is assigned no more than once. Constraints (2.3) guarantee that jobs assigned to the same operator neither overlap nor violate spread-time constraints. Note that we only need to consider $(m-1)$ jobs with their incompatible sets because the last job's incompatible relations are already included in the incompatible sets of jobs whose labels are smaller than m. Constraints (2.4) are the work-time constraints, and constraints (2.5) specify that x_{ij} variables are binary. In situations where $w = s$, i.e. when the problem is an instance of the OFJS-S, the above formulation remains intact except that constraints (2.4) are no longer needed.

Theorem 1 in [10] shows that the OFJS-S is NP-hard. The proof of this argument is based on the observation that the OFJS-S is NP-hard if the TFJS is NP-hard for problems in which $d_j \le w, j = 1, \cdots, m$. The condition $d_j \le w, j = 1, \cdots, m$ means that we need at most m operators to cover all m jobs. From this observation and the fact that the TFJS-S is NP-hard (see proof in [11]), the authors argue that the OFJS-S

is NP-hard. By using a similar argument, it follows that the OFJS-W is also NP-hard because TFJS-W has been proved to be NP-hard in [12]. Finally, the OFJS-WS is NP-hard because it includes all instances of the OFJS-W as special cases.

2.2.1 One-Operator Cases

We have observed that dispatchers rarely assign work in a manner that results in more than one split (scheduled idle period) within an extraboard operator's work shift. That is, within a spread, operators are usually idled at most once. This is because multiple splits are undesirable from operators' viewpoint. Even in situations where more than one split occurs, the maximum number of such splits is bounded and small. That is, limiting splits is a reasonable assumption in the application domain for our model. Therefore, we also analyze problem instances with one operator and a limit on the number of shift splits, and show that such problems can be solved in polynomial time. We use the fact that one-operator k-split OFJS-WS is polynomially solvable to develop a heuristic that decomposes the n-operator scheduling problem into n one-operator problems. This heuristic is presented in the Section 2.3.3. However, we begin this section with the OFJS-WS (unlimited splits) one operator instance and argue that this version of the problem is NP-hard.

Lemma 2.2.1. *The one-operator case of the OFJS-WS is NP-hard.*

Proof: Consider an instance of the subset sum problem with m items. Let d_j be item values. The subset problem is to find a subset of the m items such that sum of their values is equal to w.

Next, consider the following recognition version of the one-operator case of the OFJS-WS. For each item j, with duration d_j, start times s_j are such that $s_1 = 0$, $e_j = s_j + d_j$, and $s_{j+1} = e_j$, for $j = 1, \cdots, m-1$. Let work time-limit equal w, and spread-time limit equal $s = e_m - s_1$. The recognition version of the problem is to find an assignment to one operator such that the operator's working time is w. Note that if the maximization version of the above problem is polynomially solvable, then so is recognition version. In particular, by solving the maximization version and checking if the optimal value equals w, we solve the recognition version.

In the above instance of the OFJS-WS, there are no overlapping jobs and spread-time limit is large enough that it will not be violated. This means that constraints (2.3) and (2.4) that deal with overlapping jobs and spread-time limit may be removed from the formulation presented in (2.1) – (2.5) without affecting this instance of the OFJS-WS. At this point, it should be clear that the recognition version of one operator case and the subset sum problem are equivalent. Because the subset sum problem is NP-hard ([23]), the one-operator case of the OFJS-WS is also NP-hard. □

In contrast to Lemma 2.2.1, the one-operator case of the OFJS-S is polynomially solvable ([10]). In what follows, we discuss a transformation of the OFJS-S to the shortest-path problem, which is utilized in subsequent analysis.

We construct a directed graph utilizing the following steps; see Figure 2.1 for an illustration. First, we draw a time line and place $2 \times m$ points on the line. These points represent the start and end times of jobs. The length of the line segment between any two points is the difference between the corresponding time epochs. We also connect each job's start and end times by an additional arc. The weight of this arc is set equal to zero. Note that the direction of each arc is from left to right (start time to end time of each job).

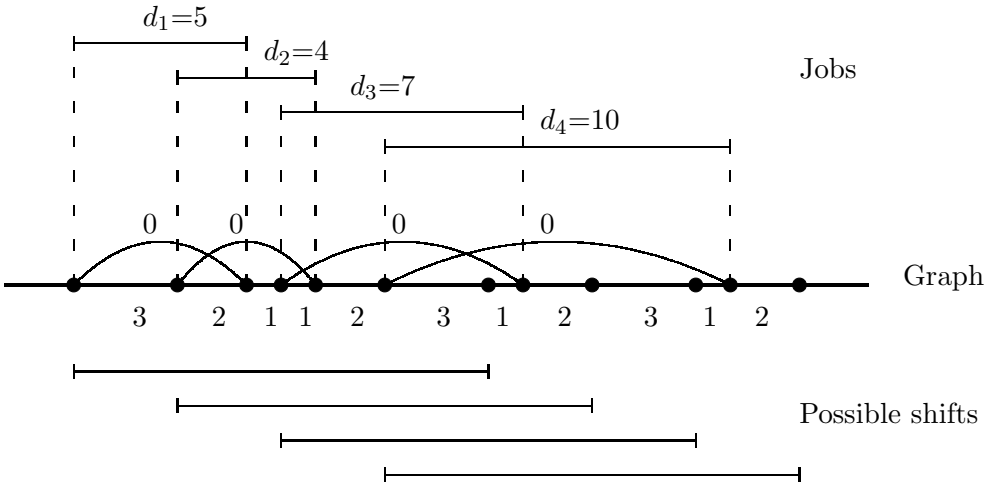


Figure 2.1: A Shortest-Path Transformation Of The Problem With four Jobs

Because there are m jobs, there are at most m choices of shift start times (which are simply the job start times). For each possible shift start time, we cut the graph to match with the shift length and find a shortest path from its leftmost node to its rightmost node. From the shortest path solution, we observe which zero-weight arcs are picked. Those correspond to the jobs that are assigned to the operator. We repeat this procedure for all possible shift start times and the solution with the highest value is the optimal solution of single-operator OFJS-S. Since the complexity of solving shortest path problem is $O(m^2)$ and we solve it $O(m)$ times, the time needed to solve the single-operator OFJS-S is within $O(m^3)$.

Next, we show that the one-operator case of the OFJS-WS is also solvable in polynomial time if the number of shift splits is at most k, although the run time grows exponentially in k.

Lemma 2.2.2. *If there is a limit k on the number of shift splits, then the one-operator case of the OFJS-WS can be solved in $O(m^{2k+3})$ time.*

Proof: It is easy to see that the best choice of a split would be such that the operator's split (idle period) would start at the end time of one job, and end at the start time of another job. So we have periods of continuous working time spaced by idle periods. With at most k splits, there are at most $2k$ jobs that form the bookends of idle periods. There are at most $\binom{m}{2k} = O(m^{2k})$ ways in which this can be done. This is an upper bound of the number of possible shifts and it may include invalid splits. An invalid split occurs, for example, when the end time of the first job is greater than the start time of the second job. Because it is difficult to find the precise number of valid splits, we use an upper bound in our arguments.

With m jobs there are $O(m)$ possible combinations of operator shift start times. For each combination of shift start time and the choice of $2k$ idle-period bookend jobs, we can obtain the best assignment by solving a shortest-path problem, which requires $O(m^2)$ steps. This means that overall, the one-operator case of the OFJS-WS with bounded number of shift splits, can be solved in at most $O(m^{2k+3})$ steps. Hence proved. □

Next, we provide the pseudo-code for an algorithm that can be used to solve the one-operator case of OFJS-WS with at most k shift splits. This algorithm is used as part of Heuristics A_2 and A_3 in Section 2.3.

Algorithm for solving k-split one-operator OFJS-WS

1 **for** $j = 1$ to m

2 consider s_j as the start of the spread and $s_j + s$ as the end of the spread

3 **for** all possible k pairs of jobs

4 each pair of jobs j_1, j_2 determines a possible split: if $e_{j_1} < s_{j_2}$ then it defines a split (e_{j_1}, s_{j_2}); if $e_{j_1} \geq s_{j_2}$ then it does not define a split;

5 if splits overlap, then combine them into one (large) split;

6 if a split exceeds the spread (either starts earlier than s_j or ends later than $s_j + s$), then cut the split such that it lies entirely within the spread;

7 calculate the total duration of splits. If the total duration exceeds $s - w$, then sort the splits by start time, and in this sequence keep as many splits as possible while ensuring that the total duration of splits does not exceed $s - w$. Use d_s to denote the total duration of the selected splits;

8 adjust the end of the spread such that the end equals $s_j + w + d_s$;

9 the spread is cut into at most $(k + 1)$ segments. Consider each segment as a single spread and solve the one operator OFJS-S on each segment. Combine these solutions to obtain the optimal assignment for the current spread with current splits. When a better assignment is found, keep that as the current best solution.

10 when all possible spreads and splits have been evaluated, report the best solution.

2.3 Heuristics

Given that the OFJS-WS is NP-hard, it is natural to spend effort on developing approximate solution techniques. In what follows, we develop three heuristics for solving the OFJS-WS, which are labeled as A_1, A_2 and A_3. In all three cases, we are interested in three characteristics of the algorithms, namely: speed, worst-case performance and average performance. The goodness of heuristics is often measured by their approximation

ratio ([24]). For sake of completeness, we include a formal definition of approximation ratio in the following paragraph. Because algorithms with good approximation ratio do not sometimes have good average performance, we develop multiple heuristics and experiment with real data to understand the speed, and average and worst-case performance trade-offs implied by each algorithm.

Approximation Ratio: Different versions of approximation ratios are used widely. we use the reciprocal of the definition given in [25, p. 400]. That is, an algorithm achieves an approximation ratio ρ for a maximization problem if, for every instance, it produces a solution of value at least $\rho \cdot OPT$, where OPT is the value of the optimal solution.

2.3.1 The Greedy Approach (A_1)

[4] and [13] both introduce algorithms based on the idea of assigning jobs to operators in a greedy fashion. Jobs are sorted according to some rule (e.g. by duration or by start time) and then assigned to available operators in that order.

Our greedy algorithm sorts jobs by duration and then assigns them in this sequence. The first job assignment always activates a new operator. For subsequent jobs, whenever a job cannot be assigned to one of the operators that has been activated before, the algorithm activates a new operator. If all n operators are active, and the job cannot be assigned to any available opeartor, then it is performed using overtime. If a job can be assigned to multiple operators, there are many possible ways to break the resulting tie. Our algorithm breaks the tie by assigning such jobs to operators who were activated the earliest. A pseudo code for the greedy algorithm is shown below.

Pseudo-code for A_1: Let P_u, P_a, and P_f denote, respectively, the set of unassigned operators, active operators, and full operators. Operators whose shifts are fully utilized are called full operators. A greedy algorithm can be constructed as follows.

A_1: The Greedy Heuristic

1 sort jobs according to the chosen criterion;
2 set $j = 1$ and $i = 0$;
3 **while** $j \leq m$ **and** $i < n$
4 search for the set $P \subset P_a$ of active operators that can cover j;
5 **if** $P = \emptyset$ and $P_u \neq \emptyset$, **then** set $i = i + 1$ and assign job j to operator i;

else if $P = \emptyset$ and $P_u = \emptyset$, **then** do not assign job j;

else select an operator in P and assign j to the operator;

6 update P_u, P_a and P_f;

7 set $j = j + 1$;

8 end

Approximation Ratio:

The Example below shows that the greedy algorithm's approximation ratio can be arbitrarily close to zero.

Example: Consider an instance of the OFJS-WS with one operator, $w = (m-1)d$, and $s = md$, where m is the number of jobs and d is the duration of jobs $2, 3, \cdots, m$. Job 1's duration is $d+\epsilon$, and the job start times are as follows: $s_1 = 0,\ e_1 = d+\epsilon,\ s_2 = md,\ e_j = s_j + d$, and $s_{j+1} = s_j + d$, for $j = 2, 3, \cdots, m$. A graphical representation of this problem (with $m = 5$) is shown in Figure 2.2.

The optimal assignment to one operator would consist of jobs $2, 3, \cdots, m$. But if we sort jobs by duration we would assign job 1 to the operator and cannot thereafter assign any other jobs because all other jobs violate spread-time constraint. Hence the greedy algorithm assigns $d+\epsilon$ units of work whereas the optimal value is $(m-1)d$. Then, the approximation ratio is $(d+\epsilon)/[(m-1)d]$, which goes to 0 as $m \to \infty$ and $\epsilon \to 0$. The same argument would also apply if we were to sort jobs by start times, because job 1 had the earliest start time.

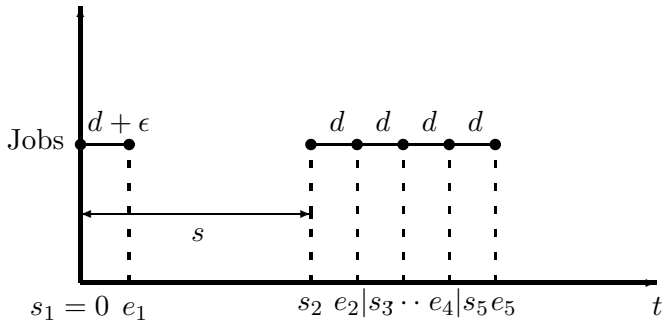


Figure 2.2: A Graphical Representation Of Job Start And End Times In Example 1.

2.3.2 The Two-Stage Approach (A_2)

As the name suggests, the two-stage heuristic solves the OFJS-WS in two stages. The first stage determines the report times of operators, whereas the second stage consists of an iterative upper and lower bounding approach that assigns jobs to operators whose report times have been fixed. We use the best lower bound as the heuristic solution. The successive bounding approach serves to improve the quality of the solution.

The first stage uses the solution of a polynomially-solvable relaxation of the OFJS-S. This relaxation is called the preemptive and partial-credit version of the OFJS-S, which we denote in this paper as the PP-OJFS-S. *Preemption* means that jobs may be divided into several parts and each part may be covered by a different operator. *Partial credit* indicates the setting in which credit is applied even when a job is only partially covered. For example, if partial credit were granted, we would be able to divide a 2-hour job into two pieces of 1-hour each, cover a 1-hour piece without covering the remainder, and still count that as finishing 1 hour of productive work. We use PP-OFJS-S, rather than the PP-OFJS-WS to obtain report times because the former can be solved in polynomial time, In contrast, we are not able to show that the PP-OFJS-WS can be solved in polynomial time even with limited shift splits.

Report times are assumed to be known in the second stage in which we assign jobs to operators. Note that the second stage problem is still NP-hard ([26]). In this stage, we develop an iterative approach for solving the OFJS-WS with fixed report times and limited shift splits. This approach is based on the well-known subgradient optimization procedure (see e.g. [27] and [3]). In what follows, we describe the two stages in detail.
The first stage: Because the PP-OFJS-S is central to the first stage, we begin by formulating the PP-OFJS-S and then show that it is solvable in polynomial time. For any instance of the OFJS-S, we first define δ_t to be the stacking of pieces of work at each $t \in \{0, \cdots, t_{max}\}$. In particular, $\delta_t = |\{j : \ s_j \leq t \leq e_j\}|$ is the total number of jobs that need to be covered at time t. Then we define $\mathcal{S} := \{s_j : \ j \in J\}$ as the set of job start times and $\mathcal{T} := \mathcal{S} \cup (\mathcal{S} + s) \cup \{e_j : \ j \in J\}$ as the set of critical time points. Clearly, $\mathcal{S}$ is the set of all possible operator report times, and $\mathcal{T}$ contains all job start and end times and all possible start and end times of operator shifts.

Figure 2.3 is a depiction of how the critical times are obtained. In this diagram, t_1 through t_4 are jobs' start and end times, so they are automatically critical times.

In addition, $t_5 = t_1 + s$ and $t_6 = t_2 + s$ are two additional critical times. We assume that the time points in $\mathcal{T}$ are sorted chronologically. Let $p = |\mathcal{T}|$. Then we can write $\mathcal{T} = \{t_k\}_{k=1}^{p}$ where $t_k < t_{k+1}$. Moreover, for any $t \in \mathcal{T}$ we define $c_{t_k} := t_{k+1} - t_k$ ($c_{t_p} := 0$) as the weight of time interval $[t_k, t_{k+1})$.

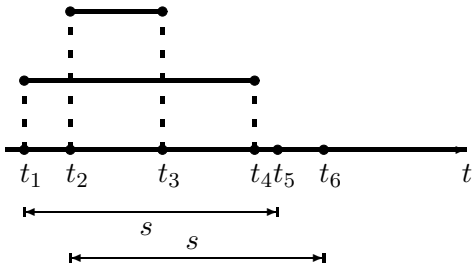


Figure 2.3: Critical Times. Note That $t_5 = t_1 + s$, $t_6 = t_2 + s$.

Let y_t be the number of operators who begin their shifts at time t, for $t \in \mathcal{S}$. Because there are n operators to schedule, we must have $\sum_{t\in\mathcal{S}} y_t \le n$. Let n_t denote the number of operators who are on-duty during $[t, t+1]$. Then, n_t should count all the operators whose report times are within $[t-s+1, t]$, or $\tau:\ \tau \le t \le \tau+s-1$, which can be written as $n_t = \sum_{\tau\in\mathcal{S}:\ \tau\le t\le\tau+s-1} y_\tau$. Next, we define x_t as the total number of jobs that can be covered at each $t \in \mathcal{T}$. With preemptive assignment and partial credit, x_t and y_t are related as follows: $x_t = \min\{n_t,\ \delta_t\}$, and the PP-OFJS-S can be formulated as shown in (2.6) – (2.11).

$$\text{Max}_{\{x_t, y_t\}} \sum_{t\in\mathcal{T}} c_t x_t \tag{2.6}$$

subject to:

$$x_t - \sum_{\tau\in\mathcal{S}:\ \tau\le t\le\tau+s-1} y_\tau \le 0 \tag{2.7}$$

$$x_t \le \delta_t, \quad t \in \mathcal{T} \tag{2.8}$$

$$\sum_{t\in\mathcal{S}} y_t \le n \tag{2.9}$$

$$x_t \ge 0,\ \text{integer}, \quad t \in \mathcal{T} \tag{2.10}$$

$$y_t \ge 0,\ \text{integer}, \quad t \in \mathcal{S} \tag{2.11}$$

Lemma 2.3.1. *Preemptive and partial credit version of the OFJS-S is polynomially solvable.*

Proof: We prove Lemma 2.3.1 by arguing that the coefficient matrix of constraints in the formulation shown in (2.6)-(2.11) is totally unimodular. Let $\mathbf{x}$ denote a vector of x_ts and $\mathbf{y}$ denote a vector of y_ts. Then, the constraints in (2.7)-(2.11) can be written as follows.

$$A \begin{pmatrix} x \\ y \end{pmatrix} \leq b$$
$$x \in \mathbb{Z}_+^{|\mathcal{T}|}$$
$$y \in \mathbb{Z}_+^{|\mathcal{S}|}$$

The coefficient matrix A and the column vector b have the following structures:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix}, \text{ and } b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix},$$

where

$$A_{11} = A_{21} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & \ddots & & & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}_{|\mathcal{T}| \times |\mathcal{T}|},$$

$$A_{12} = \begin{pmatrix} -1 & & & \\ -1 & -1 & & \\ \vdots & \ddots & \ddots & \\ -1 & \cdots & \cdots & -1 \\ & \ddots & \ddots & \vdots \\ & & \ddots & \vdots \\ & & & -1 \end{pmatrix}_{|\mathcal{T}| \times |\mathcal{S}|},$$

$A_{22} = (0)_{|\mathcal{T}| \times |\mathcal{S}|}$, $A_{31} = \begin{pmatrix} 0 & \cdots & 0 \end{pmatrix}_{1 \times |\mathcal{T}|}$, $A_{32} = \begin{pmatrix} 1 & \cdots & 1 \end{pmatrix}_{1 \times |\mathcal{S}|}$,
$b_1 = [(0, \cdots, 0)_{1 \times |\mathcal{T}|}]^T$, $b_2 = [(\delta_1, \cdots, \delta_{t_{max}})_{1 \times |\mathcal{T}|}]^T$, and $b_3 = (n)$, where $[\cdot]^T$ indicates transpose of matrix.

It is easy to see that A^T satisfies the condition in Theorem 5.23 in [28, p.103][3] for being a totally unimodular matrix. We include the theorem below and show how A^T satisfies the key condition.

Theorem 2.3.2. *(from [28, p.103]) A matrix $A = (a_{ij} \in Z^{m \times n})$ is totally unimodular if and only if for every $R \subseteq \{1, \ldots, m\}$ there is a partition $R = R_1 \cup R_2$ such that $\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij} \in \{-1, 0, 1\}$ for all $j = 1, \ldots, n$.*

We write the transpose of A as follows:

$$A^T = \begin{pmatrix} A_{11}^T & A_{21}^T & A_{31}^T \\ A_{12}^T & A_{22}^T & A_{32}^T \end{pmatrix}.$$

The lower part of A^T is called the "interval matrix". That is, each column contains either all zeros or one consecutive block of either all ones or all minus ones. Then, within a subset R of A, the rows that belong to the lower part of A^T contain consecutive blocks of ones or minus ones in each column. Therefore, we can partition rows in R that belong to the lower part of A^T such that the odd rows belong to R_1 and the even rows belong to R_2, to satisfy the condition $\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij} \in \{-1, 0, 1\}$ for every j. Moreover, because A_{31}^T and A_{22}^T are zero matrices and A_{11}^T is an identity matrix, the rows in R that belong to the upper part of A^T can be assigned to R_1 and R_2 such that $\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij} \in \{-1, 0, 1\}$ remains intact for every j.

We have shown that A^T, hence also A, is totally unimodular. As a result, a linear relaxation of the PP-OFJS-S has an integer solution and the problem is solvable in polynomial time by solving its LP-relaxation. This completes the proof. □

At the end of the first stage, we obtain operator report times, which are used in the second stage. The job assignments obtained by solving the PP-OFJS-S are ignored.

The second stage: Let $r = (r_1, \cdots, r_n)$ be the report times obtained from Stage 1 of the two-stage algorithm. Next, we present both upper and lower bounding approaches that can be iteratively improved. The upper bound is obtained by relaxing constraints (2.2) within the OFJS-WS formulation presented in (2.1)-(2.5). We employ a set of non-negative multipliers $u = \{u_1, \cdots, u_m\}$, one for each constraint in (2.2), resulting in the addition of the penalty term $\sum_{1 \le j \le m} u_j \left(\sum_{1 \le i \le n} x_{ij} - 1 \right)$ to the objective function.

[3] [28] stated that this theorem was originally proved in [29].

Let $\overline{z(u,r)}$ denote the optimal value of the objective function given penalty terms u and report time vector r. Then, a formulation of the Lagrangian relaxation of the OFJS-WS, which we denote by LR-OFJS-WS, is as follows:

$$\overline{z(u,r)} = \text{Max}_{\{x_{ij}\}} \sum_{1\le j\le m} (d_j - u_j) \sum_{1\le i\le n} x_{ij} + \sum_{1\le j\le m} u_j \tag{2.12}$$

subject to:

$$\begin{aligned}
x_{ij} + x_{ik} &\le 1, \quad j = 1,\cdots,m-1,\ i = 1,\cdots,n,\ k \in I_j && (2.13)\\
\sum_{1\le j\le m} d_j x_{ij} &\le w, \quad i = 1,\cdots,n && (2.14)\\
x_{ij} &= 0,\ j \in R_i(s) \quad i = 1,\cdots,n && (2.15)\\
x_{ij} &\in \{0,1\}, \quad i = 1,\cdots,n,\ j = 1,\cdots,m && (2.16)
\end{aligned}$$

The terms $R_i(s)$ in constraints (2.15) denote the sets of jobs that cannot be assigned to operator i given that operator's report time selected in Stage 1 and spread s. Specifically, given report time r_i from the first stage, $R_i(s) = \{j :\ s_j < r_i \text{ or } e_j > r_i + s\}$. Note that the set I_j in (2.13) is slightly different from the set $I_j(s)$ in (2.3) because only the indices of overlapping jobs are included in I_j. Spread-time violations are avoided via constraints (2.15). For this reason, we no longer show s as an argument of I_j in (2.13).

Upon examining the LR-OFJS-WS, we observe that it can be decomposed into n independent assignment problems, one for each operator i. Therefore, the LR-OFJS-WS can be solved by repeatedly solving one-operator problems, each of which is polynomially solvable when the number of permissible shift splits is finite (see Lemma 2.2.2). Before describing additional details of this approach, we provide an intuitive explanation behind the decomposition that results from the LR-OFJS-WS formulation.

Constraints (2.2) guarantee that a job would not be assigned to more than one operator. By relaxing constraints (2.2), it becomes feasible in the LR-OFJS-WS to make such assignments. In other words, after some jobs are assigned to one or more operators, we are still able to choose jobs from the original set of jobs for the remaining operators. This immediately means that optimal assignment to each operator is independent of other operators' assignments and the problem decomposes into independent problems for each operator. Each assignment of job j improves the value of the objective function by $(d_j - u_j)$, which can be affected by changing u_j. In particular, a higher value of u_j

makes it less desirable to multiply assign job j to several operators. This observation is the basis of a procedure for updating u_j that can iteratively improve the upper bound obtained via the LR-OFJS-WS.

Recall from (1) that the optimal value of the OFJS-WS is denoted by z (see 2.1). It is straightforward to argue that $\overline{z(u,r)} \geq z$ for any $u \geq 0$ ([27]). This means that solving the LR-OFJS-WS by decomposition is a polynomial-time upper-bounding procedure for the OFJS-WS when the number of allowable shift splits is bounded. Next, we present a lower bounding formulation, which is used for updating u iteratively in an attempt to improve our solution.

The lower-bounding algorithm assigns jobs to each operator one at a time such that jobs assigned in an earlier step of the algorithm are deleted from the set of available jobs for each new operator. The process continues until all operators are considered one by one. Job weights $(d_j - u_j)$ are used to assign jobs, but actual weights d_j are used to calculate the overall value of the objective function once assignments are made. At the end of each iteration, the lower bound obtained is compared to the previous best lower bound (largest value of lower bounds obtained in previous iterations) and the new value is kept if it is higher, or else the previous best value is retained. We use the notation $\underline{z(u,r)}$ to denote the lower bound obtained at an arbitrary iteration.

Next, we describe the two-stage algorithm below. We use k as iteration count and attach superscript $^{(k)}$ to each term to denote iteration number. The maximum number of iterations is denoted by N. The algorithm stops when $[\overline{z(u,r)} - \underline{z(u,r)}]/\underline{z(u,r)}$ either reaches or drops below a predetermined threshold denoted by δ, or the maximum number of iterations N are exhausted.

A_2: The Two-Stage Heuristic

1 (First stage) solve the PP-OJFS-S with spread limit s; obtain a set of operator report times;

2 (Second stage) set the weight of each job equal to duration; set iteration count $k = 0$ and set $u_j^{(k)} = 0$;

3 **while** $[\overline{z(u,r)^{(k)}} - \underline{z(u,r)^{(k)}}]/\underline{z(u,r)^{(k)}} > \delta$ and $k \leq N$

4 execute the upper-bounding procedure and record the total assigned value $\overline{z(u,r)^{(k)}}$ and assignments $x_{ij}^{(k)}$;

execute the lower-bounding procedure and record the highest lower bound at iteration

k denoted by $\underline{z(u,r)^{(k)}}$;

5 execute the multiplier updating procedure (see next paragraph);

6 set $k = k + 1$;

7 end

8 report x_{ij}s that correspond to the best lower bound.

Multiplier updating procedure: The key to the second-stage approximation algorithm is the choice of the Lagrangian multipliers u. Although it has not been proved that there exist u such that $\overline{z(u,r)} = z$, the Lagrangian relaxation method is widely used and performs well for many problem categories ([27]). Because $\overline{z(u,r)}$ is a piece-wise linear function of u, and one of its subgradients is known to be the vector $(\sum_i x_{i1} - 1, \cdots, \sum_i x_{im} - 1)$, we are able to utilize a commonly-used approach for the multiplier updating, which is described next.

Let $u_j = \max(0, u_j + \lambda(\sum_i x_{ij} - 1))$, where x_{ij} is a solution of the upper-bounding procedure, and $\lambda = \frac{\mu(\overline{z(u,r)} - \underline{z(u,r)})}{\sum_j (\sum_i x_{ij} - 1)^2}$. In the previous expression, μ is referred to as the step-size parameter. When implementing a two-stage algorithm, μ is set equal to an initial value (we used 1) and its value is halved each time when the gap between $\overline{z(u,r)}$ and $\underline{z(u,r)}$ does not decrease after carrying out a certain number of iterations (we used 5).

Approximation Ratio: We were unable to establish an approximation ratio for the two-stage heuristic. However, the successive bounding approach is expected (although not guaranteed) to improve the solution at each iteration, resulting in good overall performance. In the upper bounding procedure, each job is allowed to be assigned multiple times. If this happens, the value $(\sum_i x_{ij} - 1)$ will be positive and the multiplier updating procedure will increase u_j, which will cause a decrease in the value $(d_j - u_j)$. This makes job j less attractive in future iterations of the algorithm in both upper-bounding and lower-bounding procedures. Conversely, if job j is not assigned, then its value $(d_j - u_j)$ will increase and the job will become a more attractive candidate for assignment in the next iteration. The updating step presents an opportunity to reduce the value of the upper bound and to increase the value of the lower bound at each iteration. For these reasons, the two-stage algorithm performed well in numerical experiments with real data (see Section 2.4).

2.3.3 The Decomposition-Based Approach (A_3)

The third approach we propose uses a decomposition to solve the OFJS-WS. The idea is to decompose the n-operator assignment problem into n separate one-operator cases. We assume a limited split setting. Recall that such problems can be solved in polynomial time, as shown in Section 2.2.1. At each iteration we introduce a new operator and obtain the best assignment with the remaining jobs. In the following description of A_3 we use $\tilde{J}$ to denote the set of unassigned jobs and i to denote the current operator being scheduled.

A_3: The Decomposition Heuristic

1 set $\tilde{J} = J$ and $i = 0$;

2 **while** $\tilde{J} \neq \emptyset$ and $i < n$

3 introduce a new operator;

4 maximize assignment to the operator by using limited split one-operator algorithm (see Section 2.2.1);

5 update $\tilde{J}$ and set $i \leftarrow i + 1$;

6 **end**

Algorithm Complexity: Because the limited split one-operator algorithm runs in $O(m^{2k+3})$ time (see Lemma 2.2.2), it is clear that Algorithm A_3 also runs in $O(nm^{2k+3})$ time. It is polynomial in m and n, but its run time increases exponentially in k.

Approximation Ratio: Finding the exact approximation ratio of a decomposition algorithm for solving the FJS class of problems is difficult. For example, [13] attempted to do so for the TFJS-S, but did not find a precise approximation ratio. Instead, [13] presented an instance of the TFJS-S and used its solution to argue that the approximation ratio could not be larger than 3/2. Recall that the TFJS-S is a minimization problem and therefore the approximation ratio is never smaller than 1, and a higher approximation ratio implies worse performance in this case. Similar to these earlier papers, we are also unable to establish a precise approximation ratio for the OFJS-WS. Instead, we define $f(p) = 1 - [(p-1)/p]^p$ and show that the approximation ratio lies in $[f(\infty), f(3)]$ with $f(\infty) = 1 - 1/e$ and $f(3) = 19/27$.

To establish the main result in this section, we begin by arguing that the OFJS-WS can be viewed as a special case of the *weighted maximum coverage problem (W-MCP)* ([30]). An arbitrary instance of the W-MCP has a finite universe set Ω, a positive weight

w_j for each element in Ω, a collection $\mathcal{A} = \{S_1, S_2, \cdots, S_\ell\} \subset 2^\Omega$ of ℓ subsets of Ω, and a designated number n. The objective of the W-MCP is to find a subcollection A' with cardinality at most n, such that the number of elements covered by A' is maximized. Below, we present a greedy algorithm for solving W-MCP due to [31], which is known to have an approximation ratio of at least $(1 - 1/e)$.

The correspondence between the W-MCP and the OFJS-WS is illustrated in Table 2.2.

Table 2.2: The Correspondence Between the W-MCP and the OFJS-WS

	W-MCP	OFJS-WS
Candidate Set	$\mathcal{A}$ (given)	$\mathcal{J}$ (not given)
Candidate Set Size ℓ	Given	Determined by (s_j, e_j), $\forall\ j \in J$, w, s, and number of shift splits
Search for the best Candidate	Exhaust all Elements in the remainder of $\mathcal{A}$	Solve One-Operator case with the remaining job set

Greedy Algorithm for W-MCP ([31])

1 Repeat n times:

2 find one element S' in $\mathcal{A}$ that yields the largest increment of the current objective function if added to the solution.

3 add S' to the solution, and delete S' from $\mathcal{A}$.

In the following proposition we establish the correspondence between the OFJS-WS and the W-MCP and argue that A_3 is a version of the greedy algorithm for solving the W-MCP. From [31], it then follows that A_3 has an approximation ratio of at least $(1 - 1/e)$.

Proposition 2.3.3. $\rho(A_3) \geq 1 - 1/e$.

Proof: We perform the following transformation from the OFJS-WS to the W-MCP. Let the set of jobs be the universal set Ω and d_j be the weight of the j-th element, for $j = 1, \cdots, m$. Define the candidate set $\mathcal{J} = \{S_1, S_2, \cdots, S_\ell\}$, in which S_k is a set of jobs that can be assigned to an operator, and ℓ is the total number of possible sets. Let

the number of operators n be the number of subsets we can choose. Given this setup, the correspondence between the W-MCP and the OFJS-WS is straightforward.

Note that the set of all possible one-operator assignments $\mathcal{J}$ is determined by jobs' start and end times, w, s, and the number of splits allowed. Therefore it is difficult to calculate ℓ without obtaining all feasible assignments to a single-operator problem. Fortunately, it is not necessary to know all elements of $\mathcal{J}$ to implement a greedy approach in our setting. If we were to select subsets S_i in a greedy way in the W-MCP, then that would be equivalent to finding the best single-operator assignments from remaining jobs in J at each assignment step. This can be accomplished in polynomial time when the number of splits is finite (see Lemma 2.2.2). This means that A_3 is equivalent to the greedy algorithm in [31]. Therefore, from [31], we immediately have that $\rho(A_3) \geq 1 - 1/e$. □

In order to establish a good upper bound, we need to construct examples whose approximation ratio is as close as possible to the lower bound established in Proposition 2.3.3. Note that the lower bound can be viewed as the limiting case of a sequence $f(p) = 1 - [(p-1)/p]^p$, $p \geq 1$, in the limit as $p \to \infty$. If we find an example with performance $f(\hat{p})$ for some $\hat{p}$, then that establishes a good upper bound for all $p \leq \hat{p}$ because $f(p)$ is a decreasing sequence. If we find such examples for every p, then the lower bound is tight.

In what follows, we describe an example with approximation ratio $f(p)$ for $p = 3$, i.e. the approximation ratio is $f(3) = 19/27$. Note that p is not a parameter of this example. In particular, it does not relate to either the number of jobs or the number of operators. Unfortunately, the approach used to construct this class of examples does not extend to cases with $p > 3$. Therefore, we were not able to find even sharper bounds on the approximation ratio of the decomposition algorithm proposed here.

Proposition 2.3.4. $\rho(A_3) \leq 19/27$

Proof of Proposition 2.3.4: We construct an example with 3 operators, $s = w = 9$, and 9 jobs. The 9 jobs are divided into 3 groups each including 3 jobs. The start and end times of each job in group 1 are respectively: $(0, 3-\epsilon)$, $(3-\epsilon, 5)$ and $(5+4\epsilon, 9)$ with durations $3-\epsilon$, $2+\epsilon$ and $4-4\epsilon$. Jobs in group 2 and 3 have the same sequence of durations, except that the start times are shifted by $3-\epsilon$ for group 2 and $2(3-\epsilon)$ for

group 3.

The jobs are shown in Figure 2.4 in which each row represents one group of jobs. Given this layout, it is easy to see that an optimal solution is to have each operator cover a group of jobs.

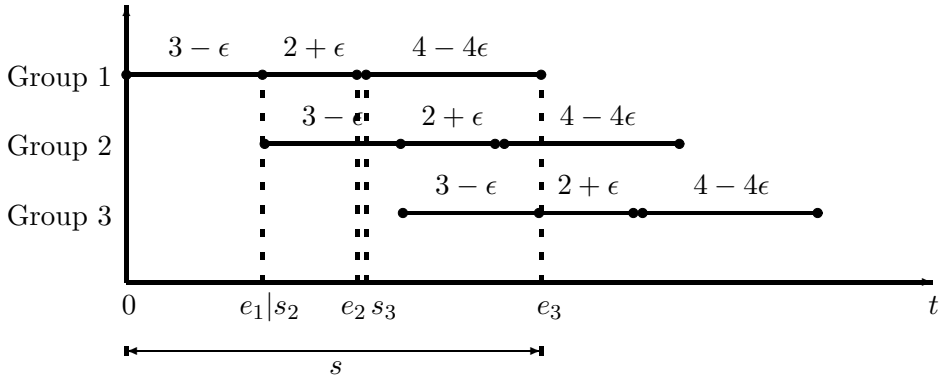


Figure 2.4: Step 1 of 3-Operator Example.

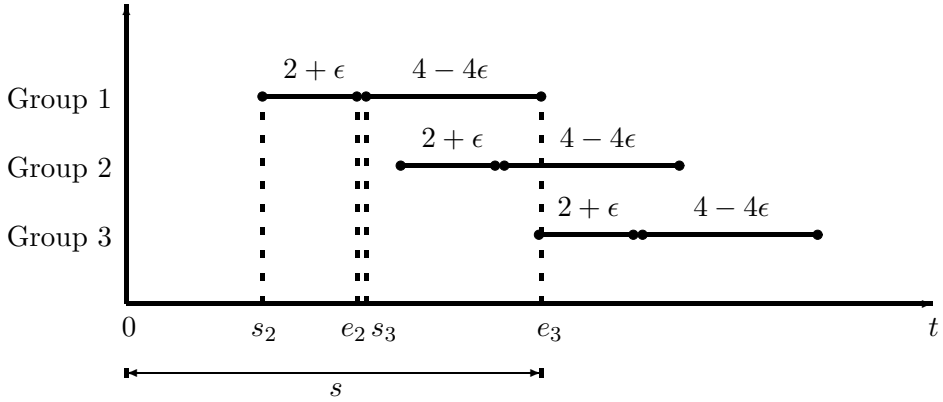


Figure 2.5: Step 2 of 3-Operator Example.

Next, consider what would happen if the decomposition algorithm is used to obtain the best solution for the first operator. With that approach, the optimal one-operator solution will be to assign all three jobs with duration $3-\epsilon$ to the first operator. The remaining jobs available for assignment to the second operator appear as shown in Figure 2.5. Note that two jobs of duration $2+\epsilon$ cannot be combined with a job with duration $4-4\epsilon$ because of the spread-time constraint. Therefore, at this step, the best one-operator assignment is to assign the three jobs with duration $2+\epsilon$ to operator 2. Finally, at the last step only the longest jobs remain. Since any two of the longest

jobs cannot be assigned to one operator because of overlap or spread-time violation, the third operator only covers a single job of duration $4 - 4\epsilon$.

Combining the results from the above steps, we see that A_3 returns an objective value of $19 - 8\epsilon$, whereas the optimal value is $27 - 12\epsilon$. That is, the approximation ratio can be made arbitrarily close to $1 - (2/3)^3$ via the choice of ϵ. This completes the proof of the Proposition. □

2.4 Numerical Experiments

We received extraboard operations data for five randomly-picked months from a large transit agency that served as a research partner for this study. The agency had 5 garages and each month's data came from a different garage. The data included all jobs that were assigned to either extraboard operators or on an overtime basis. The agency rarely dropped service. In fact, there were no examples of dropped service in our data set, which meant that the jobs in our data set represented the entire demand for extraboard services. From the data, we identified the subset of jobs that were known to be open a day before each day of operations and calculated the number of extraboard operators available to serve those jobs. Recall that a certain number of operators are placed on call duty each day – i.e. they are only assigned jobs that arise during the course of the day. We did not include those operators in our study.

Because extraboard operators' wages are sunk, transit agency attempts to assign as much work as possible in their regular work time. Our algorithms mimic this objective. The transit agency also considers overtime availability constraints on certain days of the year, e.g. the Christmas day when overtime availability is limited. Our algorithms ignore overtime availability constrains. Thus, they are close approximation of the problem faced by the dispatcher on most days of the year.

For each given day, the data sometimes contained one or more 8-hour long jobs. We excluded such jobs from the data and reduced the number of available operators on that day by the same amount. This was done because assigning each 8-hour long job to a single operator is trivially an optimal strategy, irrespective of other assignments. The set of jobs we worked with were the jobs that were left after this process of elimination. On many weekends and some weekdays, the number of individual 8-hour long jobs exceeded

the number of extraboard operators available to perform such duties. We excluded such days from our experiments altogether, which left 100 instances of the problem from the 5 months of data. In table 4.3 we summarize the data. Empirical distributions of job start times and job durations, job numbers, and operator numbers, are provided in Tables 2.3 – 2.7.

Table 2.3: Distribution of Job Durations (Percent)

Duration (Minutes)	Garage 1	Garage 2	Garage 3	Garage 4	Garage 5
≤ 30	1.1	1.8	3.1	7.0	1.5
31-60	0.8	5.0	1.5	4.4	0.8
61-90	0.4	5.8	9.4	7.5	1.1
91-120	2.4	10.5	16.4	10.6	6.8
121-150	4.0	6.8	4.9	4.6	4.6
151-180	7.2	7.5	7.1	9.8	5.3
181-210	10.3	6.8	13.0	9.0	7.7
211-240	9.5	6.8	2.2	5.1	9.0
241-270	4.8	8.1	2.0	3.0	4.5
271-300	6.0	4.2	3.2	1.4	10.8
301-330	7.3	3.2	1.2	3.0	6.0
331-360	6.6	3.0	0.5	4.8	6.0
361-390	3.7	7.3	6.1	2.8	4.7
391-420	0.9	2.9	4.0	0.4	3.6
421-450	1.5	4.4	3.5	2.9	2.1
451-480	1.5	5.1	2.4	1.8	2.4
≥ 481	10.3	5.3	10.1	13.7	8.2
each column sums to 100					

Table 4.3 shows that there was a great deal of variation in open jobs from one day to another. The average job duration was different for each garage and lay between 3 and 4 hours. The job durations were quite variable – the coefficient of variation was more than 0.5 for 3 out of 5 garages. The number of available extraboard operators and the number of jobs also varied a great deal. Together this suggests an environment in which there is no particular pattern of open jobs and each day's problem requires a tailor-made solution.

We used CPLEX (version 10.0.0) to solve the model presented in (2.1) – (2.5). This solution served as a benchmark for comparison with solutions obtained from the three

Table 2.4: Distribution of Job Numbers (Percent)

Job Number	Garage 1	Garage 2	Garage 3	Garage 4	Garage 5
≤ 50	0	0	80.9 (21-30)	25 (21-30)	0
51-60	4.8	0	14.3 (31-40)	60 (31-40)	0
61-70	9.5	0	4.8 (41-50)	10 (41-50)	9.1
71-80	19.0	0	0	5 (51-60)	54.5
81-90	19.0	0	0	0	36.4
91-100	14.3	0	0	0	0
101-110	9.5	0	0	0	0
111-120	4.8	0	0	0	0
121-130	0	18.2	0	0	0
131-140	9.5	9.1	0	0	0
141-150	4.7	4.5	0	0	0
151-160	4.7	18.2	0	0	0
161-170	0	18.2	0	0	0
171-180	0	13.6	0	0	0
181-190	0	4.5	0	0	0
191-200	0	9.1	0	0	0
≥ 201	0	4.5	0	0	0
each column sums to 100					

Table 2.5: Distribution of Job Start Times (Percent)

Start Time	Garage 1	Garage 2	Garage 3	Garage 4	Garage 5
3:30-3:59 AM	1.7	0.4	4.6	1.4	0.2
4:00-4:29 AM	2.7	2.4	0.2	0.8	0.8
4:30-4:59 AM	3.3	5.6	5.6	2.5	7.7
5:00-5:29 AM	5.6	2.6	7.8	3.3	11.3
5:30-5:59 AM	5.4	11.0	2.9	5.7	7.9
6:00-6:29 AM	9.2	5.0	9.1	5.0	6.4
6:30-6:59 AM	3.3	4.4	5.1	7.7	4.8
7:00-7:29 AM	2.5	1.2	2.2	0.7	0.7
7:30-7:59 AM	1.3	0.2	0.7	1.9	0.1
8:00-8:29 AM	3.5	1.1	5.9	8.6	2.0
8:30-8:59 AM	4.0	1.3	6.1	1.9	1.6
9:00-9:29 AM	2.2	1.1	1.2	2.3	2.3
9:30-9:59 AM	1.4	2.3	4.7	1.2	0.4
10:00-10:29 AM	4.6	2.2	2.0	1.8	1.9
10:30-10:59 AM	2.5	1.5	1.3	2.8	0.8
11:00-11:29 AM	2.8	2.9	0.7	1.9	2.5
11:30-11:59 AM	1.1	2.1	6.4	1.2	4.3
12:00-12:29 PM	3.6	3.4	0.7	3.3	2.1
12:30-12:59 PM	0.8	2.4	1.0	2.3	2.0
1:00-1:29 PM	3.0	1.6	0.8	3.7	2.7
1:30-1:59 PM	6.8	1.7	0.5	4.4	5.1
2:00-2:29 PM	7.6	4.7	2.0	4.4	9.2
2:30-2:59 PM	4.7	2.8	6.6	1.7	5.8
3:00-3:29 PM	5.6	5.4	6.2	6.6	4.9
3:30-3:59 PM	3.1	9.4	5.7	6.6	3.3
4:00-4:29 PM	2.0	7.7	5.7	8.0	1.2
4:30-4:59 PM	0.3	2.2	0.8	2.1	0
5:00-5:29 PM	0.0	0.3	5.1	0.3	0.1
5:30-5:59 PM	0.4	0.9	0.2	1.2	0
6:00-6:29 PM	0.1	0.2	0.3	0.8	1.2
6:30-6:59 PM	0.9	0.4	0.3	2.3	0.8
7:00-7:29 PM	0.4	1.2	1.2	1.0	1.5
7:30-7:59 PM	1.5	1.1	0.3	0.1	1.1
8:00-8:29 PM	1.8	0.8	0	0.1	2.8
8:30-8:59 PM	0	2.0	0.2	0	0.1
9:00-9:29 PM	0	0.2	0.3	0	0
9:30-9:59 PM	0	0.6	0.2	0	0
10:00-10:29 PM	0	0.3	0	0	0
10:30-10:59 PM	0	0.8	0	0	0
11:00-11:29 PM	0	0	0	0	0
11:30-11:59 PM	0	0.8	0	0	0
each column sums to 100					

Table 2.6: Distribution of Job Numbers (Percent)

Job Number	Garage 1	Garage 2	Garage 3	Garage 4	Garage 5
≤ 30	0	0	80.9	25	0
31-40	0	0	14.3	60	0
41-50	0	0	4.8	10	0
51-60	4.8	0	0	5	0
61-70	9.5	0	0	0	9.1
71-80	19.0	0	0	0	54.5
81-90	19.0	0	0	0	36.4
91-100	14.3	0	0	0	0
101-110	9.5	0	0	0	0
111-120	4.8	0	0	0	0
121-130	0	18.2	0	0	0
131-140	9.5	9.1	0	0	0
141-150	4.7	4.5	0	0	0
151-160	4.7	18.2	0	0	0
161-170	0	18.2	0	0	0
171-180	0	13.6	0	0	0
181-190	0	4.5	0	0	0
191-200	0	9.1	0	0	0
≥ 201	0	4.5	0	0	0
each column sums to 100					

Table 2.7: Distribution of Operator Numbers (Percent)

Operator Number	Garage 1	Garage 2	Garage 3	Garage 4	Garage 5
1-2	0	0	0	0	0
3-4	0	0	4.8	0	0
5-6	0	0	19.0	0	0
7-8	0	0	42.8	5	0
9-10	4.8	0	23.8	20	4.5
11-12	14.3	0	9.5	20	13.6
13-14	47.6	0	0	35	18.2
15-16	9.5	4.5	0	15	22.7
17-18	23.8	4.5	0	5	31.8
19-20	0	9.1	0	0	9.1
21-22	0	27.3	0	0	0
23-24	0	13.6	0	0	0
25-26	0	22.7	0	0	0
27-28	0	9.1	0	0	0
29-30	0	9.1	0	0	0
≥ 31	0	4.5	0	0	0
each column sums to 100					

Table 2.8: Job Statistics by Garage (C.V. = Coefficient of Variation)

		Min	Max	Avg	Std. Dev	C.V.
Garage 1 (21 days)	number of daily jobs	24	50	37	6.5	0.18
	job duration (min)	21	480	226	87.3	0.38
	number of operators	16	24	19	2.4	0.13
Garage 2 (20 days)	number of daily jobs	51	81	70	7.9	0.11
	job duration (min)	15	480	205	116.4	0.57
	number of operators	19	34	26	4.3	0.17
Garage 3 (21 days)	number of daily jobs	27	49	37	6.2	0.17
	job duration (min)	16	480	191	111.0	0.58
	number of operators	8	17	13	2.1	0.16
Garage 4 (19 days)	number of daily jobs	26	64	38	9.3	0.24
	job duration (min)	20	480	177	109.8	0.62
	number of operators	13	20	16	2.3	0.14
Garage 5 (19 days)	number of daily jobs	38	57	48	5.2	0.11
	job duration (min)	20	480	239	96.1	0.40
	number of operators	15	25	19	2.6	0.14

heuristic algorithms introduced in Section 2.3. All experiments were performed on a PC with Intel Core 2 CPU 6600 2.40 GHz processor and 4 GB of RAM. CPLEX failed to converge to an optimum solution in 33 out of the 100 problem instances after running for 30 minutes, which was set as a criterion for stopping CPLEX. In the 33 instances, we used the best feasible solution at the time of stopping as benchmark and recorded the upper and lower bounds to calculate percent gap. A summary of the percent gap between bounds in the 33 cases is shown in Table 2.9 below.

Table 2.9: Percent Gap Between Upper and Lower Bounds

Min (%)	Max (%)	Avg (%)	Std.Dev (%)
0.1	4.3	1.4	1.3

To evaluate algorithm A_2 we first compare the report times obtained by solving the PP-OFJS-S problem with the report times provided by CPLEX in Table 2.10. We see that the two results do not match often – in fact the number of times the two report times match is generally well below 50%. Still A_2 performs quite well in terms of total amount of work assigned to extraboard operators in regular time because it finds near-optimal work assignments for each set of report times in the second stage. This suggests that the overall good solutions exist for multiple selections of report times.

Table 2.10: Percent of Report Time Matches: PP-OFJS-S versus CPLEX

Garage (# of days)	Min (%)	Max (%)	Avg (%)	Std.Dev (%)
Garage 1 (21)	17.6	62.5	40.1	11.6
Garage 2 (20)	22.2	73.7	45.0	14.9
Garage 3 (21)	16.7	85.7	51.3	16.2
Garage 4 (19)	18.5	66.7	44.8	15.5
Garage 5 (19)	16.7	62.5	33.9	12.2

The purpose of numerical experiments was to identify an algorithm that performed well relative to the CPLEX solution and that was also fast. We calculated three performance metrics for this purpose: two of these measured the relative quality of the solution produced and one measured speed. The metrics were: (1) Algorithm solution/CPLEX

solution (in percent), (2) Percent of times that the algorithm solution is $\geq 99\%$ of the CPLEX solution, and (3) Computation time (in seconds). The results are reported in Table 2.11. A quick look at this table reveals that A_1 runs quite fast, but produces the worst average performance among the three algorithms. A_2 has much better average performance than A_1, but runs much slower. Algorithm A_3 produces good average performance and runs fast at the same time. Moreover, A_3 is the only algorithm that has a proven non-zero approximation ratio. Therefore, the experiments support the claim that algorithm A_3 is the best among the three algorithms evaluated.

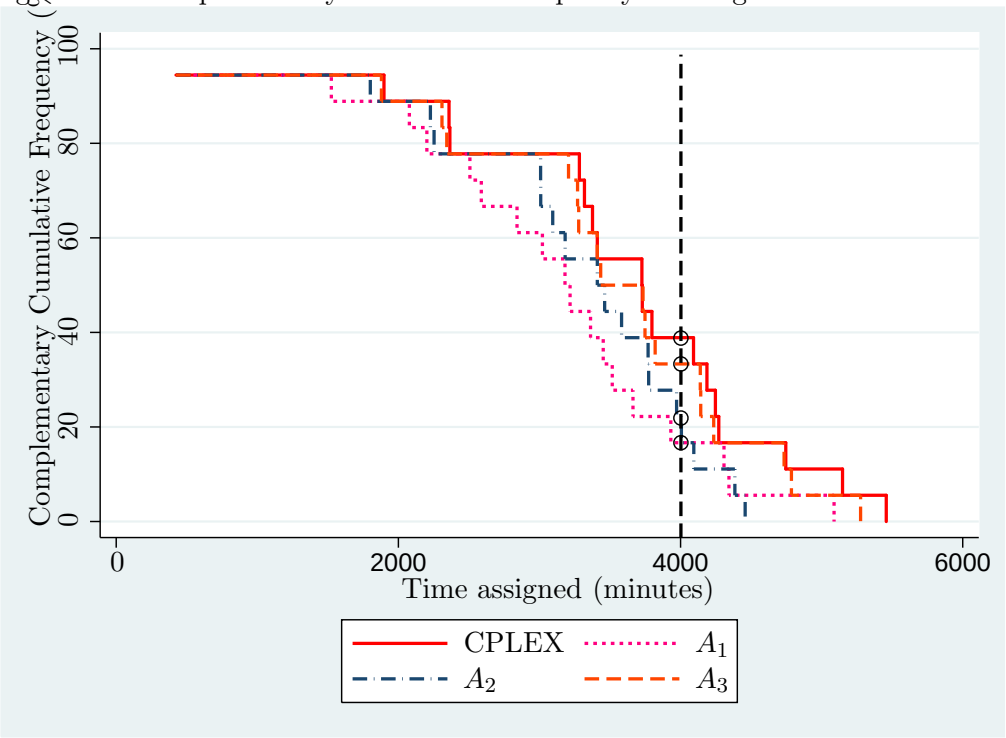
Table 2.11: Comparison of Algorithms' Performances

Metric	Garage	A_1	A_2	A_3
$\frac{\text{Solution from algorithm}}{\text{Solution from CPLEX}}$ (percent)	1	88	92	98
	2	96	95	98
	3	94	92	96
	4	96	97	98
	5	92	97	99
Frequency that algorithm returns $\geq 99\%$ of CPLEX solution (percent)	1	6	11	44
	2	5	23	41
	3	5	32	16
	4	6	39	44
	5	0	45	86
Computation time (Sec)	1	2.1	240	3.4
	2	2.4	320	4.6
	3	1.7	120	2.9
	4	2.0	170	3.3
	5	2.7	290	5.0

To further compare the three algorithms, we developed complementary cumulative frequency plots of the number of minutes assigned by each method – see Figure 2.6. Complementary cumulative frequency is 100% minus the cumulative frequency of assigned time for each algorithm. Roughly speaking, if an algorithm's performance is to the right, then that implies a superior performance. We observe in Figure 2.6 that A_3 is to the right of A_1 and A_2 for nearly all values of assigned work. For example, we draw a vertical line in Figure 2.6 at 4000 minutes to draw attention to the fact that whereas A_1 and A_2 assign more than 4000 minutes of work in about 20% of all problem instances,

A_3 does so in nearly 34% of instances. Therefore, the performance of A_3 dominates the performance of the other two heuristics in the usual stochastic order[4] . This algorithm runs fast and does not require the transit agency to invest resources in purchasing a commercial optimization software such as CPLEX. The data also shows that the use of A_3 could save somewhere between 1.2 to 6.5 hours of overtime on weekdays. The average saving per day per garage is 3.6 hours. Using average overtime wage rate of \$42, which we obtained from our research partner, this implies approximate annual savings of $\$196,560$ (which is obtained by calculating $5 \times 52 \times 5 \times 3.6 \times 42$).

Figure 2.6: Complementary Cumulative Frequency of Assigned Work in Minutes



[4] A random variable X is stochastically smaller than another random variable Y in the usual order, written $X \leq_{st} Y$, if $E[\phi(X)] \leq E[\phi(Y)]$ for all non-decreasing functions ϕ for which the expectations exist (see [32] and [33] for further details).

2.5 Concluding Remarks

The paper is motivated by extraboard operator scheduling and work assignment problems that are faced by transit agencies on a daily basis. We present a model and three algorithms for solving the operational fixed job scheduling problem with work-time and spread-time constraints (OFJS-WS). We show that the OFJS-WS is NP-hard. We prove that A_3, a decomposition-based approach, has an approximation ratio that lies in the range $[1 - 1/e, 19/27]$. We perform numerical experiments using data from the collaborating transit agency and show that our algorithm provides close-to-optimal solutions and has the potential to improve extraboard work assignments. Ongoing efforts by the authors are focused on solving the day-of scheduling problems, and improving understanding of the relationship between the day-before scheduling and the day-of scheduling problems.

Chapter 3

Reserve Driver Work Assignment Problem: Day-of

3.1 Introduction

Transit agencies (bus, light rail, subway, ferry) use *reserve drivers* to cover work that arises from planned and unplanned time off, equipment breakdowns, weather, and special events. On their duty days, some reserve drivers cover another driver's full shift (typically 8 hours) or some combination of open pieces of work (which we also refer to as *jobs*) that are known in advance, while the rest are placed on *call* duty. Moreover, if a reserve driver covers some pieces of work that are known in advance but has open time in his or her shift, then for those periods of time, he or she is considered to be on call duty. A dispatcher assigns open work as it arises either to available on-call drivers, or to drivers that indicate their willingness to take overtime work assignments, giving rise to an *online interval scheduling* problem ([2] and [34]). This problem, also known as the *fixed job scheduling* problem (FJS) ([1]), is the focus of our paper. In particular, we develop an approach for solving the reserve driver scheduling problem that takes into account transit agency objectives.

Given that the wages of all reserve drivers are already committed, performance is measured by the amount of work covered by on-call drivers. Transit agencies would prefer a methodology that maximizes the worst case performance and at the same time competes well with straw man approaches in terms of average performance. In order to

realize good performance, the agency may strategically assign some work to overtime drivers even when the same piece of work could be assigned to an on-call driver because that may reduce opportunity cost (i.e. allow the assignment of a longer future job to that driver). This gives rise to the key tradeoff considered in this paper – the extent to which the proposed algorithm acts either myopically (i.e. assigns all feasible jobs) or strategically (i.e. assigns jobs that minimize opportunity cost). Before describing our solution strategy for this problem, we explain the problem scenario in detail in the next several paragraphs.

Open work due to planned absences (training, union meetings, and vacations), and special events such as a major league game are known in advance. However, open work due to equipment breakdowns, weather-related delays, accidents, drivers calling in sick just before the start of their shifts, and unexpectedly high volume of riders on some routes are not known in advance. Our focus in this paper is on jobs whose specifications are revealed just before their start time and the scheduler must make an instantaneous decision whether to assign them to an on-call driver or to an overtime driver without information about future job requests. All jobs have fixed start and end times, drivers work in shifts, and each driver can process at most one job at a time. Also, previously assigned jobs may not be preempted because of the effort involved in driver and equipment mobilization. Hourly overtime wages are higher than hourly regular wages and many drivers have part-time appointments. As a consequence, if agencies choose an appropriate number of reserve drivers, they typically have ample supply of drivers willing to perform occasional extra work in overtime. For example, in the data provided to us by Metro Transit, the agency responsible for the bulk of transit operations in the Twin Cities of Minneapolis and Saint Paul, there were no instances of dropped service on account of unavailability of overtime. We assume ample availability of overtime in this paper.

Examples of online interval scheduling problems arise in the context of scheduling jobs on parallel machines in a whole host of make-to-order or on-demand-processing environments. Jobs and machines represent different entities in different application areas. For example, jobs could be orders that need machining or repair, or deferrable surgeries that need operating room time, or computer programs that need processor time. The instances of such problems that we study are economically important. To

underscore this point, we provide some statistics from transit industry. According to [5], approximately 26% (on average) of the total workforce size of large transit agencies consists of reserve drivers. Metro Transit shared bus driver data with the authors for the period March–August, 2010. In this period, Metro Transit operated three large and two small garages and employed approximately 1500 bus drivers, of which approximately 30% were reserve drivers. The average utilization of on-call drivers was between 50% and 60%, depending on the garage (see [35] for details). Still, the daily overtime usage during weekdays was well over 100 hours in each of its three large garages. At approximately $42 per hour, this added tens of thousands of dollars in overtime cost daily. Our algorithm has the potential to reduce overtime costs relative to the myopic approach and benefit transit agencies across the United States.

Because we are motivated by providing an implementable solution to the reserve driver scheduling problem, our strategy for solving this problem is different from that in the literature. We provide a review of the literature at a later point in this section. But first, we describe our solution strategy.

Nearly all previous papers dealing with online interval scheduling problem focus on proving the worst-case performance guarantee. In contrast, taking cue from practitioners, our objective is to develop an algorithm that carries a worst-case performance guarantee and at the same time performs well in terms of average performance. We use real data and demonstrate the practical value of the proposed algorithm for reserve-driver scheduling. Second, we consider features that are relevant in transit agency application domain. In particular, we model the possibility that multiple drivers may be able to perform a particular job, and the fact that the minimum and the maximum job duration are known. These considerations result in several innovations, which we describe next.

Because the minimum and maximum job durations are known in the reserve driver scheduling problem context, we are able to find *optimal* coin-flip probabilities that determine whether a job will be assigned to a reserve driver or an overtime driver. In contrast, previous papers focus on finding a sequence of probabilities whose sum converges to 1 and that lead to a provable approximation ratio bound. In technical terms, we use the *marriage problem* framework introduced in [36] to propose a *finite-step marriage problem*, whose solution serves as a building block of our approach for solving the

single-driver problem scenario.

Another innovation in this paper is that our algorithm considers multiple drivers with different remaining shift lengths at the time a job arrives. We found that algorithms that ignored this fact led to poor average performance. To overcome this shortcoming, we introduce two parameters in our algorithm. These parameters are denoted as d_T and α. The first parameter, d_T, serves to strike a balance between the myopic and strategic approaches. Essentially, the algorithm behaves myopically if a job duration is d_T or longer. The second parameter α specifies the criterion for considering the next job following a rejection. If the next job is not at least α times the previously rejected job, then it is rejected outright. Otherwise, it is considered for assignment according to prescribed coin-flip probabilities. The introduction of these two parameters allows our algorithm to realize good average performance, in addition to having a guaranteed worst-case performance.

We decompose the online FJS problem into two subproblems. The first subproblem considers which driver should be assigned to each arriving job. The second subproblem decides whether to accept or reject a job for each driver. This is essentially the single-driver case. When presenting our approach in the sequel, we present the second subproblem first because it solves the simpler single-driver problem. We review the relevant literature next.

There are many papers that deal with the online FJS and interval scheduling problems. In the literature, many papers use the term processor to refer to the server, i.e. the reserve driver in our setting. Therefore, we also use processor when describing the literature.

[37] provided an algorithm to deal with the single-processor cases with preemption. Several later papers that address this class of problems deal with special cases, which allow the authors to obtain sharper performance bounds. For example, if the set of jobs is such that when job j_1 arrives before j_2 it also ends before j_2, then a 3-competitive algorithm is provided in [38]. Rather than describe all such papers in detail, we list the features of key papers in Table 3.1. This is not an exhaustive list. The papers contain either proofs of competitive ratios (CRs) of proposed algorithms, or lower bounds of competitive ratios (CBs) for all possible algorithms, or both.

Paper	Main Features				Result
	Machine Environment	Weight and duration	Preemption	Others	
[37]	(1)	(3) (4)	(8)		CR=4
[39]	(2)	(3)	(8)		
[40]	(1)	(3)	(8)		CR= $(2+\sqrt{3})$
[38]	(1)	(3)	(8)	(10)	CR=3
[41]	(1)	(3)	(5) (7)		CB=$4(1+\rho)$*
[42]	(2)	(4) (9)	(8)		CR=3.618, CB=4/3
[43]	(2)	(3)	(6)	(11)	CR=4, CB=4
[44]	(2)	(4)	(8)		CR=$O(\log k)$ ($k = w_{\max}/w_{\min}$)
[45, 46]	(1)	(3)	(7)		CR=$O(\Delta)$

(1) single-processor, (2) multiple-processor, (3) weight=duration, (4) weight≠duration,
(5) preemption-restart, (6) preemption-resume, (7) preemption-penalty, (8) no penalty
(9) identical duration, (10) non-decreasing job sequence, (11) jobs not fixed but have deadline.
$d_{\min}$, $d_{\max}$ are the minimum and maximum job durations;
$w_{\max}$, $w_{\min}$ are the maximum and minimum importance factors;
ρ is the penalty factor.

Table 3.1: Notation Used in Formulation

Papers on online FJS problem with preemption mainly study when and how the preemption action needs to be performed, which is not relevant in the non-preemptive cases. Instead, it is important in settings such as ours to decide whether or not to accept a job when the driver is available. Papers that consider preemption with penalty may obtain solutions that contain no preemption if the penalty is large, but the best known algorithm in such cases is $O(\Delta)$-competitive ([45]), where Δ is used to denote $d_{\max}/d_{\min}$, the ratio of maximum and minimum job durations.

Papers that deal with realtime scheduling in which job start time is not fixed but each job has a fixed duration and deadline ([43]) are also related to the online FJS problems. These papers focus on (1) either single ([40]) or multiple processors ([42]), (2) either deterministic ([37]) or random algorithms ([41]), and (3) either establishing the competitive ratio ([37]), or providing a bound on the competitive ratio ([41]) of a particular algorithm. Key model features include (1) whether job weight is proportional to duration ([37]), and (2) types of preemption—(a) preemption-restart ([41]), (b) preemption-resume ([43]), and (c) preemption with penalty ([41], [45], [46]). For models in which job weights are not proportional to durations, [44] introduce a parameter called the importance factor of a job. Each job is then defined by three parameters: (w, d, e), where d is the duration, e the deadline, and w the importance. The importance ratio is defined as $k := w_{max}/w_{min}$. [47] provide guaranteed offline algorithms for different settings of real-time scheduling problems.

In this paper, we build on the work of [36], who provide an algorithm that can be used to solve single driver online FJS without preemption and that has a performance ratio of $O((\log \Delta)^{1+\epsilon})$. As mentioned earlier, the authors identify a converging infinite sequence of coin-flip probabilities that leads to the above mentioned performance ratio bound. [48] extend the result of [36] in two ways. First, the authors show that multiple different acceptance probability sequences $\{f(1), f(2), \cdots\}$ can be used to form different randomized algorithms if $\sum 1/f(n)$ converges, and that a competitive ratio bound of $1/O(\log(f(n))$ can be achieved. Second, the paper extends the result into multiple identical processor case. There are two key differences between these papers and our approach. First, we find the optimal coin-flip probabilities. Second, we consider non-identical processors and introduce two additional parameters to realize a good average performance.

[49] considers the problem of online scheduling of continuous media streams, in which each job requires a fixed processing time and a portion of bandwidth. This can be considered as the model with multiple identical processors and jobs that require different numbers of parallel machines to process. [49] prove a competitive ratio lower bound of $O((\log \Delta)/(1-r))$, where r is the maximum fraction of the server's bandwidth that a job can demand.

[49] first show that the greedy polity is $O(\Delta)$ competitive. Then the authors show that the $O(\log \Delta)$ is a lower bound of all possible random or deterministic algorithms. Finally the authors provide algorithms with $O((\log \Delta)/(1-r))$ competitive ratio with known minimum and maximum lengths of jobs. The idea of the algorithm is as follows. Divide the available bandwidth into $\lceil \log \Delta \rceil$ partitions evenly. Then, suppose the min/max lengths of jobs are $d_{\min}$ and $d_{\max}$ respectively: partition the interval $[d_{\min}, d_{\max}]$ into $\lceil \log \Delta \rceil$ parts, with ℓ-th interval being $[2^{\ell-1} d_{\min}, 2^{\ell} d_{\min})$. That is, divide durations into exponential levels. Then, for an upcoming job j, if the duration j lands in the i-th interval, then it can use i parts of the pre-divided bandwidth, or $i/O((\log \Delta)/(1-r))$ share of bandwidth. So if job j does not require more than that, accept it; otherwise, reject it.

We pursue alternate approaches in this paper. To our knowledge, no previous paper has examined the problem with multiple processors who might have different shift lengths. The referred papers do not consider the case where processors have different

available time windows.

The rest of this paper is organized as follows. We introduce common notation and assumptions, the myopic algorithm and the finite-step marriage game in Section 3.2. Then, in Section 3.3, we study the first subproblem. The second subproblem, i.e. the job-to-processor allocation problem, is analyzed in Section 3.4. These two parts together complete our recommended algorithm. Numerical experiments are presented in Section 3.5 and we conclude the paper in Section 3.6.

3.2 Preliminaries

In the remainder of this paper, we use the terms processor and driver interchangeably. An arbitrary instance I of the online FJS with multiple processors is characterized by a finite set of jobs and processor shifts. Each processor's shift is a time window denoted by (a_i, b_i) where i is the processor index and processors are sorted such that $a_1 \leq \cdots \leq a_n$. When a driver's shift is cut up because of previous assignments, each available period of the driver can be treated as a separate driver with start and end times of the shift equal to the start and end times of available time window. Processor i cannot accept any job that starts before a_i or ends after b_i. Parameters s_j and e_j denote the start and end times of job j, job indices are sorted by their start times (i.e. $s_j \leq s_{j+1}$ for all j). The duration of each job $d_j = e_j - s_j$ is its weight or reward. For the purpose of performance evaluation, we only consider instances I that contain jobs that can be performed by at least one processor during his or her shift. The number of processors and their shift start and end times are known, but not the number of jobs in I, although that is limited.

At each job's arrival epoch, an online algorithm allocates the job to one of the processors, and then decides whether to accept or reject this job based only on the current state of the processor and features of the job. In contrast, an offline algorithm knows the entire sequence of job requests in advance of making assignment decisions. For an arbitrary randomized algorithm A, let $W(A, I)$ denote the random total reward (i.e. the sum of durations of jobs that are processed) realized upon completion of an arbitrary run of A on I. Because $W(A, I)$ may vary each time A is applied to the same problem instance I, we use $E[W(A, I)]$ to benchmark the performance of A. Let

$z(I)$ be the value obtained from applying an optimal offline algorithm, which assumes complete information about all jobs in I before scheduling any job. Algorithm A is called β-competitive if $\beta \cdot E[W(A,I)] \geq z(I)$ for any instance I. The smallest value of β such that the above inequality holds for every I is the exact competitive ratio of A (see, e.g. [36]).

Because the offline version of our problem is NP hard ([50]), we replace $z(I)$ by an upper bound $U(I)$ and $E[W(A,I)]$ by a lower bound $F(A,I)$. That is, for an algorithm A, rather than calculate $\mathrm{CR}(A) = \max_I\{z(I)/E[W(A,I)]\}$, we estimate the upper bound $\mathrm{CB}(A) = \max_I\{U(I)/F(A,I)\} \geq \mathrm{CR}(A)$. Our efforts therefore focus on finding $U(I)/F(A,I)$ and on designing algorithm A that results in provably smallest possible value of $\mathrm{CB}(A)$.

We summarize all notation used in this paper in Table 3.2 and thereafter list assumptions underlying our models. These assumptions are justified by the intended application of our models to reserve-driver scheduling problem. For example, the dispatcher knows that no job would be shorter than about half an hour ($d_{\min}$), or longer than the typical shift length of 8 hours ($d_{\max}$). When two or more jobs start at the same time, we assume that either the sequence in which they need to be assigned to drivers is known, or generated randomly. In constructing proofs of various claims throughout this paper, we remove all "empty spaces" to simplify our analysis. That is, when there is a period with no demand between two jobs, we remove such time periods between the two jobs and also remove the corresponding time period from every processor's shift. Similarly, if a shift start time a_i is earlier than $\underline{s} = \min_j\{s_j\}$, the start time of the first job in I, or a shift end time b_i is later than $\overline{e} = \max_j\{e_j\}$, the finish time of the latest ending job in I, then portions of shifts $(a_i, \underline{s})$ and $(\overline{e}, b_i)$ are also removed. Note that such "empty spaces" can be recognized one by one as successive jobs are revealed. Because no jobs can be assigned during empty spaces caused by either the pattern of arrivals in I or driver shifts, removal of such spaces does not affect the total work assigned by any algorithm in both online and offline versions.

Assumption 1: Each job's arrival epoch coincides with its start time.

Assumption 2: Jobs in progress may not be preempted by a new arrival.

Assumption 3: All rejected jobs are covered by overtime drivers.

Assumption 4: A job's weight is equal to its duration.

Assumption 5: The minimum and the maximum job durations are known.

3.2.1 The Myopic Algorithm for Single-Processor Cases

The myopic (greedy) algorithm accepts every job when it arrives so long as the processor is capable of processing that job. It has a finite competitive ratio as shown below.

Lemma 3.2.1. *The competitive ratio of the myopic algorithm is* $(\Delta + 1)$.

Proof: In this proof we assume without loss of generality that $d_{\min} = 1$ and therefore $\Delta = d_{\max}/d_{\min} = d_{\max}$. The statement of the lemma is proved by establishing two claims. The first claim is that the competitive ratio of the myopic algorithm cannot be better (smaller) than $(\Delta + 1)$. The second claim is that for any $\epsilon > 0$, there exists an instance in which the myopic algorithm assigns no more than $(1 + \epsilon)/(\Delta + 1)$ of the optimal assignment.

Each algorithm produces periods in which the driver is busy and periods in which he or she is idle. After empty spaces are removed, the first job arrives at $t = 0$. If the dispatcher uses a myopic algorithm, then the arrival of the first job must start a busy period because the myopic algorithm will accept that job. Observe that no busy period can be shorter than $d_{\min} = 1$. Moreover, because $d_{\max} = \Delta$ and idle periods under a myopic algorithm result from job overlap, all idle periods must be shorter than Δ. If we take any two connected busy and idle periods, we have that at least $1/(\Delta + 1)$ of the sum of their durations must be busy. This proves the first claim.

For the second claim, we draw attention to an instance of the problem illustrated in Figure 3.1, where $\epsilon > 0$ is arbitrary. Job 1: $(0, 1 + \epsilon)$, Job 2: $(\epsilon/2, 1 + \epsilon/2)$, and Job 3: $(1 + \epsilon/2, 1 + \Delta + \epsilon/2)$. In this instance, the myopic algorithm results in the driver being busy for the duration $(1 + \epsilon)$.
Jobs j_2 and j_3 overlap j_1 and cannot be taken after j_1 is taken. But the optimal solution is to take jobs j_2 and j_3 with total duration $(\Delta + 1)$. This proves the second claim. □

The arguments presented in the proof of Lemma 3.2.1 can be used to establish that $(\Delta + 1)$ is the best (smallest) competitive ratio that *any* deterministic algorithm can achieve. We believe such arguments are well known, but we include them here for sake of completeness. Every deterministic algorithm can be placed into one of two classes when a job with duration $(d_{\min} + \epsilon)$ arrives, and the processor is idle – Class 1 contains

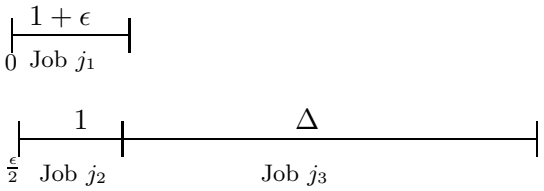


Figure 3.1: A $(\Delta + 1)$-Competitive Case

algorithms that accept that job and Class 2 contains algorithms that reject it. Next, we compare their performance in two instances: (1) the example shown in Figure 3.1, and (2) the example in which the job with duration $(d_{\min} + \epsilon)$ is the only job. We see that algorithms in Class 1 by accepting the first job have a competitive ratio of $(\Delta + 1)$, whereas Class-2 algorithms by rejecting the first job have a competitive ratio of ∞, establishing that the best competitive ratio must be $(\Delta + 1)$.

Next we present a variant of the [36]'s marriage problem that utilizes known duration bounds and obtains a better CR.

3.2.2 The Finite-Step Marriage Problem

In the marriage problem presented in [36], a host selects a number N, which the player doesn't know. At step k, the host offers the player a reward of 2^k for $k = 1, \cdots, N$. At any step, the player can choose to accept the offer which terminates the game. If the player doesn't accept any offer in all N steps, the game terminates and the player gets zero. [36] proposed a policy, according to which the player would accept at step k with probability $1/[k^{1+\epsilon}\zeta(1+\epsilon)]$, for $k = 1, \cdots, \infty$, where $\zeta(1+\epsilon) = \sum_{i=1}^{\infty} 1/i^{1+\epsilon}$. Observe that $\sum_{i=1}^{\infty} 1/i^{1+\epsilon}$ converges for any positive ϵ, so the policy is well-defined. Because the proposed policy accepts the offer at step N with probability $1/[N^{1+\epsilon}\zeta(1+\epsilon)]$, it is at most $O(N^{1+\epsilon})$-competitive, which is $O((\log \Delta)^{1+\epsilon})$-competitive because $\Delta = 2^{N-1}$.

The finite-step marriage problem is similar to the classical marriage problem introduced in [36] with the difference that there is a maximum-possible number of turns $K \geq 2$ and this fact is known to both the player and the host. Another difference is that each successive offer is not necessarily twice as large as the previous one. Rather

it can be any multiple $\alpha > 1$ of the previous offer. Starting from turn 1, the host offers the player α^k at turn k. The host also picks the number of turns N, $N \leq K$. The player does not know N but knows that N cannot be greater than K. The player can accept only once. If the player does not accept in N turns, then the game ends and the player receives zero.

Suppose the player decides to take the offer at turn $\hat{N}$. If $\hat{N} > 1$, then all those cases in which $N < \hat{N}$ yield zero reward, which has an infinite competitive ratio. If $\hat{N} = 1$, then the ratio of the performance relative to the best outcome is $1/\alpha^{N-1}$. That is, the only deterministic strategy that yields a finite competitive ratio is $\hat{N} = 1$. Choosing to accept at the first turn results in a competitive ratio of α^{N-1}, which is bounded by α^{K-1}. Next, we investigate whether the player can earn a greater expected reward by adopting a randomized strategy.

Suppose the player flips a coin at each turn to decide whether to accept or reject the host's offer at that turn. Then, a randomized strategy consists of conditional probabilities p_k, $k = 1, \cdots, K$, of accepting the offer at step k if the game proceeds to step k. Lemma 3.2.2 identifies an optimal randomized strategy and we also establish its performance bound.

Lemma 3.2.2. *The optimal randomized strategy for solving the finite-step marriage problem consists of coin-flip probabilities* $p_1 = \alpha/[K(\alpha-1)+1]$ *and* $p_k = 1/(K-k+1)$, $k = 2, \cdots, K$. *This strategy is* $\frac{K(\alpha-1)+1}{\alpha}$*-competitive, i.e.* $O(\log \Delta)$ *competitive because* $K = \log \Delta + 1$.

Proof: Let c_k denote the probability that the game terminates at step k. Clearly, p_k and c_k are related as follows: $p_k = c_k/(\sum_{i=k}^{K} c_i)$, $k = 1, \cdots, K$. The problem of finding p_k's is equivalent to that of finding a set of probabilities $\{c_1, \cdots, c_K\}$ satisfying $\sum_k c_k = 1$. The performance ratio if $N = k$ is $R_k := \sum_{i=1}^{k} \alpha^{i-k} c_i$ such that $R_1 = c_1$ and $R_k = R_{k-1}/\alpha + c_k$, for $k = 2, \cdots, K$. The player would choose c_k's to maximize the worst-case performance, i.e. solve $\max[\min\{R_1, \cdots, R_K\}]$, subject to $\sum_k c_k = 1$, and $c_k \geq 0$. This is achieved by setting all R_k's equal, i.e. $R_k = \alpha/[K(\alpha-1)+1]$ for every k, which gives $c_1 = \alpha/[K(\alpha-1)+1]$ and $c_k = (\alpha-1)/[K(\alpha-1)+1]$ for $i = 2, \cdots, K$. Such a strategy leads to a competitive ratio of $[K(\alpha-1)+1]/\alpha$. Given c_k's, it is easy to calculate the coin-flip probabilities shown in the statement of the lemma. Hence proved. □

3.3 A Randomized Algorithm for Single-Processor Online FJS

The proposed algorithm, labeled $A(\alpha, d_T)$, has two discretionary parameters α and d_T. Parameter $\alpha > 1$ determines which jobs are considered by the processor and $d_{\min} < d_T < d_{\max}$ is a threshold duration such that for jobs with duration d_T or greater, algorithm $A(\alpha, d_T)$ uses a greedy approach (accept if feasible), and for jobs with duration less than d_T, $A(\alpha, d_T)$ decides based on an approach similar to that in the finite-step marriage game. Thus, d_T determines the extent to which $A(\alpha, d_T)$ is conservative (close to myopic) or risk taking (close to marriage problem). It should be clear that $A(\alpha, d_{\min})$ is identical to the myopic approach. We label a job as type-1 if its duration is smaller than d_T, and as type-2 if its duration is at least d_T.

When job j arrives at time t in an arbitrary instance of single-processor online FJS, it may find the processor in one of three states: *clear*, *virtually busy*, and *busy*. Busy indicates that job j overlaps a job that is being processed at t. Virtually busy means that job j overlaps a job that was considered but not accepted because it was not long enough and there was an unfavorable coin flip, and clear means neither of the above. A job that causes the driver to be virtually busy is referred to as a *virtually taken* job.

Algorithm $A(\alpha, d_T)$ upon finding the processor in clear state will accept the arriving job if its duration is at least d_T. Otherwise, this arrival starts the equivalent of a finite-step marriage game. Each such epoch resets a counter that we call the *D-level*. The value of D-level counter equals $\delta := \lceil \log_\alpha(d_{\max}/d_{j_1}) \rceil$, where d_{j_1} is the duration of the job that starts the marriage game. Next, suppose a job j arrives at time t, the processor is virtually busy, and the current D-level is δ. Let v denote the index of the virtually taken job at t. Define parameter $\ell := \lfloor \log_\alpha(d_v/d_{j_1}) \rfloor$ as the degree of the current virtually taken job. To ascertain whether d_j is sufficiently large relative to d_v, we compare $r := \lfloor \log_\alpha(d_j/d_{j_1}) \rfloor$ and ℓ. If $r > \ell$, then d_j is deemed sufficiently large and the algorithm would consider job j as a candidate job. This means, it will flip a coin to decide whether to accept or reject j. Otherwise, it will reject j. The coin-flip probabilities are determined as follows.

If following j_1, each overlapping job is exactly α times as long as its predecessor, then from the marriage problem analogy j_1 should be taken with probability $p_1^\delta :=$

$\alpha/[\delta(\alpha-1)+1]$ and each subsequent job with probability $(\alpha-1)/[\delta(\alpha-1)+1]$. (It should be clear that δ is analogous to K in the finite-step marriage problem.) But jobs do not necessarily arrive in such a pattern. For example, the job following virtually taken j_1 may be α^3 times the duration of j_1. In such cases, we award the second job the sum of probabilities that correspond to the duration αd_{j_1}, $\alpha^2 d_{j_1}$, and $\alpha^3 d_{j_1}$. That is, we calculate the r value to determine which equivalent turn index the new job j belongs to and compare it with the turn index of job v. Note that the turn index in the equivalent finite-step marriage problem is calculated relative to j_1 and that the turn index associated with v is labeled ℓ. If $r > \ell$, we calculate the corresponding coin-flip probability. The coin-flip probability is the ratio of the number of the slots from ℓ to r divided by the total number of slots from ℓ to δ, i.e. $(r-\ell)/(\delta-\ell)$. These arguments lead to Algorithm $A(\alpha, d_T)$, below, which can be implemented in real time.

Pseudo-code for Algorithm $A(\alpha, d_T)$

```
0   when job j arrives at time t:
1     if processor is processing a job, then reject j
2     end
3     if processor is clear or virtually busy and j is type-2, then accept j.
4     end
5     if processor is clear and j is type-1, then calculate the D-level δ and flip a coin: accept j with
      probability p_1^δ = α/[δ(α−1)+1] or virtually take j with probability 1 − p_1^δ.
      Set j_1 = j_v = j and ℓ = 0.
6     end
7     if processor is virtually busy and d_j < d_T, then calculate r = ⌊log_α(d_j/d_{j_1})⌋:
8       if r ≤ ℓ, then reject j
9       else flip a coin for job j: accept j with probability (r−ℓ)/(δ−ℓ) and virtually take j with
        probability 1 − (r−ℓ)/(δ−ℓ). Set j_v = j and ℓ = r.
10      end
11    end
```

3.3.1 The Competitive Ratio Bound

The derivation of the competitive ratio bound hinges upon finding $U(I) \geq z(I)$ and $F(A, I) \leq E[W(A, I)]$. To accomplish this task, we construct disjoint subsets $(S_1, S_2, \cdots)$

of jobs in I that have the following properties. First, the construction allows us to estimate an upper bound on the time period between each S_k and S_{k+1}, which we denote by y_k such that $\sum_k |S_k| + y_k \geq \overline{e} - \underline{s}$, the relevant span of I (see the discussion just before Section 3.2.1). We use $|S_k|$ to denote the time period covered by the set S_k. Second, by construction, sets S_k have the property that the minimum expected amount of work that $A(\alpha, d_T)$ assigns to the processor within each S_k is independent and identical across all subsets. Therefore, the fraction of work that $A(\alpha, d_T)$ assigns within each $(|S_k| + y_k)$ is no less than $U(I)/F(A, I)$. The former is then our estimate of CB(A). We begin with the rules for constructing sets S_k.

Rules for adding a job to S_k

1.0 Suppose j is the last-added job in S_k. Then add $\hat{j} > j$ to S_k if:

1.1 $\hat{j}$ overlaps j and

1.2 either $d_{\hat{j}} \geq \alpha d_j$ or $d_{\hat{j}} \geq d_T$

Jobs are added to S_k following Rule 1 until there are no qualifying jobs left.

Rules for initiating S_k (finding the first job in S_k)

2.0 If $k = 1$, then j_1 (the first job in I) begins S_1 by definition.

Suppose $k \geq 2$, and let j be the last-added job in S_{k-1}.

Find the lowest-indexed job $\hat{j} > j$ to begin S_k such that

2.1 either: $d_{\hat{j}} \geq d_T$,

or: $(d_{\hat{j}} < d_T)$ and for every job ℓ indexed between j and $\hat{j}$ that overlaps $\hat{j}$,

2.2.1 either ℓ does not overlap any job in S_{k-1},

2.2.2 or $d_{\hat{j}} \geq \alpha d_\ell$.

We illustrate the application of these rules with the help of several examples. In Figure 3.2, we present an example in which I results in a single chain S_1. For this example, we have $\alpha = 2$, $d_T = 4.5$, and 5 jobs with the following durations: $d_{j_1} = 1$, $d_{j_2} = 2$, $d_{j_3} = 2$, $d_{j_4} = 4$, and $d_{j_5} = 5$. Job j_1, being the first job is automatically included in S_1. Next, job j_2 is included because $d_{j_2} \geq \alpha d_{j_1}$. Continuing to apply the rules for composing chains, we find that j_3 is not included because $d_{j_3} < \alpha d_{j_2}$, j_4 is included because $d_{j_4} \geq \alpha d_{j_2}$, and finally, j_5 is included because $d_{j_5} > d_T$. There are no other jobs and this terminates the process of forming chains. In this example, job j_3 does not belong to a chain.

Next, we present another example in Figures 3.3 that explains the construction of S_k after finalizing S_{k-1}. The same logic applies to the construction of an arbitrary S_{k+1}

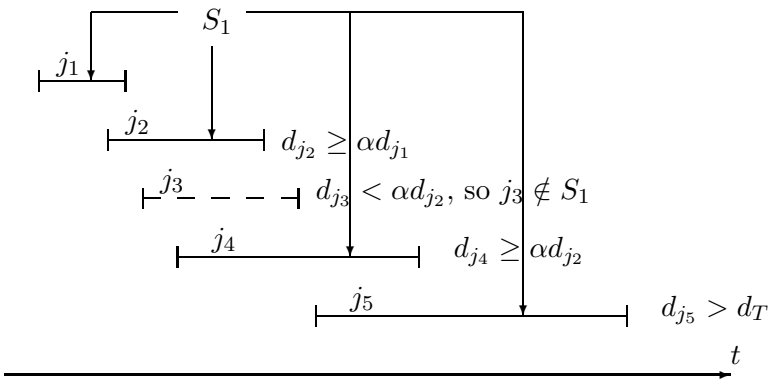


Figure 3.2: Construction of S_1, $S_1 = \{j_1, j_2, j_4, j_5\}$

following S_k. In such cases, the tricky part is finding the "starting job" of S_k because the formation of the chain is identical to that in the earlier example. In Figure 3.3, we see that j_3 does not qualify as the starting job of S_k. Job j_3 fails because $d_{j_3} < \alpha d_{j_2}$ and j_3 overlaps S_{k-1}. Job j_4 does not overlap j_3 and any other job in S_{k-1}. It starts S_k. Note that job j_3 does not belong to any chain.

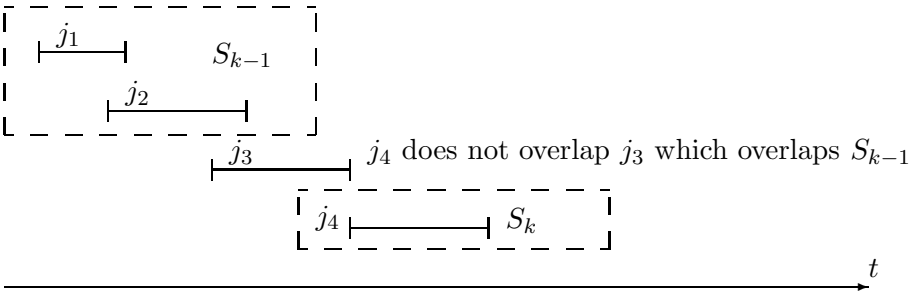


Figure 3.3: The first job of S_k is j_4 following Rule 2.2.1

Similarly, Figure 3.4 shows another example in which job j_5 starts S_k. In this example, job j_3 does not belong to S_k because it overlaps S_{k-1} (Rule 2.2.1). Job j_4 overlaps j_3, and $d_{j_4} < \alpha d_{j_3}$. So j_4 also cannot begin S_k (Rule 2.2.2). However, $d_{j_5} \geq \alpha d_{j_3}$. Therefore, j_5 satisfies Rule 2.2.2 and starts S_k. Note that jobs j_3 and j_4 do not belong to any subset.

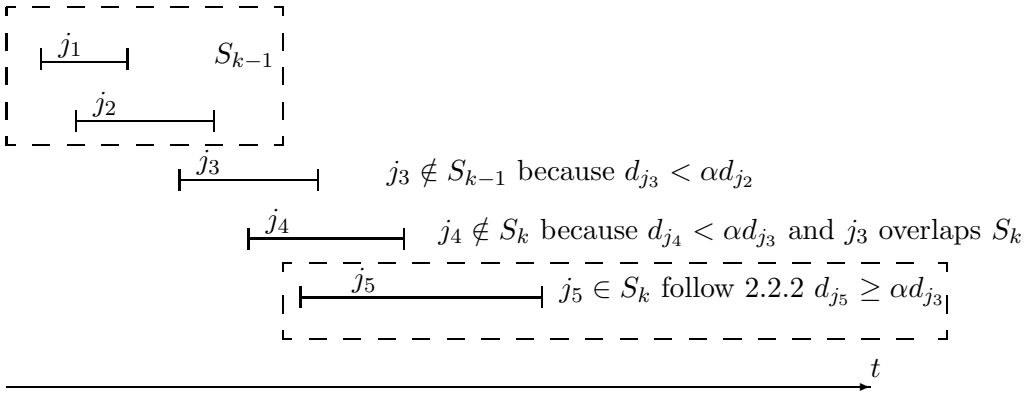


Figure 3.4: The first job of S_k is j_5 following Rule 2.2.2

As seen in examples above, the first job in S_k must be such that during the execution of $A(\alpha, d_T)$, it cannot be rejected on account of its relationship with jobs in S_k. Two cases arise. If the last job in S_k has duration less than d_T, then the first job j of S_{k+1} must meet the following criteria: either its duration is at least d_T, or if its duration is less than d_T, then any j' that overlaps j either does not overlap a job in S_k or $\alpha d_{j'} < d_j$. If the duration of the last job in S_k is at least d_T, then the first job of S_{k+1} is the first job that does not overlap S_k. This method of construction avoids interaction among jobs in S_k and we can analyze the amount of work assigned in each S_k independently.

Lemma 3.3.1. *The time periods y_k between consecutive pairs S_k and S_{k+1} can be calculated as follows.*

Case (1) *If all jobs in S_k are type-1, then y_k cannot be greater than $(\alpha^2 + \alpha)|S_k|$.*

Case (2) *If there is at least one type-2 job in S_k, then y_k cannot be greater than $|S_k|$.*

Proof: Our proof for Case (1) follows the logic presented in [36] but generalizes it for an arbitrary $\alpha > 1$ and known duration bounds ([36] considers only the $\alpha = 2$ case). No job that overlaps with S_k can be longer than $\alpha|S_k|$ because otherwise it should belong to S_k. Therefore, any job starting outside S_k with duration at least $\alpha^2|S_k|$ qualifies to be the first job of S_{k+1}. In the extreme case, a job with duration $\alpha|S_k| - \epsilon$ overlaps S_k, and another job with duration $\alpha^2|S_k| - \epsilon$ overlaps the previous job, giving rise to the maximum y_k of $(\alpha + \alpha^2)|S_k|$ – an example is shown in Figure 3.5. A type-2 job also

qualifies to be the starting job of S_{k+1}, but in that case, y_k is smaller leaving intact the worst-case argument presented above.

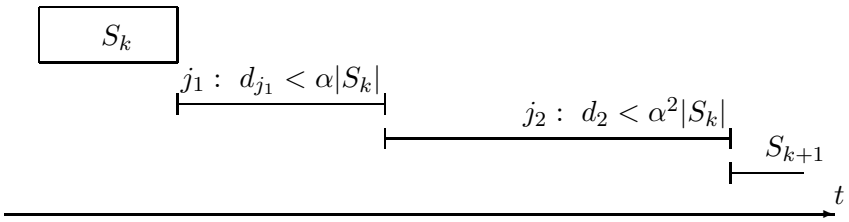


Figure 3.5: Largest y_k (when all jobs in S_k are type-1)

In Case (2), S_k contains at least one type-2 job and we select the first job that does not overlap any job that overlaps S_k as the first job of S_{k+1}. The worst case in this situation is one in which a job with duration $(d_T - \epsilon)$ overlaps both S_k and S_{k+1}. In this case y_k is at most $(|S_k| - \epsilon)$, which happens when S_k contains only one job with duration d_T. □

Note that the statement in Lemma 3.3.1 remains intact for the last set, indexed by κ. In that case, y_κ is defined as the time period between the end of S_κ and $\overline{e}$, the end of I.

Lemma 3.3.2. *The minimum expected amount of work assigned in each S_k can be calculated as follows.*

Case (1) *If all jobs in S_k are type-1, then the expected amount of work $A(\alpha, d_T)$ will assign on the region covered by S_k is at least $(\alpha-1)^2|S_k|/\{\alpha^2[(\alpha-1)\log_\alpha \Delta_1+1]\}$.*

Case (2) *If there are type-2 jobs in S_k, then the expected amount of work $A(\alpha, d_T)$ will assign on the region covered by S_k is at least $(\alpha-1)|S_k|/\{(\Delta_2 + \frac{\alpha}{\alpha-1})[(\alpha-1)\log_\alpha \Delta_1 + 1]\}$.*

Proof: The proof of Case (1) uses arguments similar to those presented in [36], which considers the $\alpha = 2$ case. We provide a proof with arbitrary α for sake of completeness.

Let j_m be the last job of an arbitrary subset S_k and let σ be a feasible schedule of work assignments. Because in this case, all jobs are of type 1, it follows that $d_{j_m} < d_T$.

For an arbitrary run of the algorithm $A(\alpha, d_T)$, when presented with j_m, one of the following cases may occur.

(a) processor is clear;

(b) processor is busy processing a job labeled j_ℓ, and either

(b.1) $d_{j_m} < \alpha d_{j_\ell}$, or

(b.2) $d_{j_m} \geq \alpha d_{j_\ell}$;

(c) processor is virtually processing a job labeled j_ℓ, and either

(c.1) $d_{j_m} \geq \alpha d_{j_\ell}$, or

(c.2) $d_{j_m} < \alpha d_{j_\ell}$.

In Case (a), $A(\alpha, d_T)$ will flip a coin for j_m with success probability $p_1^\delta = \alpha/[(\alpha - 1)\delta + 1]$. That is, j_m is assigned with probability at least $\alpha/[(\alpha - 1)\delta + 1]$.

In Case (b.1), j_ℓ whose duration is at least $1/\alpha$ times the duration of j_m, would have been taken with probability at least $(\alpha - 1)/[\delta(\alpha - 1) + 1]$.

To evaluate Case (b.2), we consider all jobs that arrive within (s_{j_ℓ}, s_{j_m}), i.e. the interval between the start times of jobs j_ℓ and j_m, and use $\bar{S}$ to denote this set. If there exist some jobs in $\bar{S}$ with duration greater than $(1/\alpha)d_{j_m}$, we denote the set of such jobs by $\bar{S}'$. If $\bar{S}'$ is not empty, then there exists at least one job j in the set $\bar{S}' \cup \{j_m\}$ with the property: $\lfloor \log_\alpha(d_j/d_{j_1}) \rfloor > \lfloor \log_\alpha(d_{j_\ell}/d_{j_1}) \rfloor$, where j_1 is the index of the job that started the finite-step marriage problem sequence associated with j_ℓ. Pick the first job satisfying this property in the set $\bar{S}' \cup \{j_m\}$ and refer to it as j'_ℓ. Then, j'_ℓ, with duration at least $(1/\alpha)d_{j_m}$, would be taken with probability at least $(\alpha-1)/[\delta(\alpha-1)+1]$. Alternatively, $\bar{S}'$ may be empty because all jobs in S_ℓ have duration less than $(1/\alpha)d_{j_m}$. Then j_m would be taken with probability at least $(\alpha - 1)/[\delta(\alpha - 1) + 1]$. These two cases are shown in Figures 3.6 and 3.7.

In Case (c.1), a coin flip would decide if j_m is accepted. That is, j_m would be taken with probability at least $(\alpha - 1)/[\delta(\alpha - 1) + 1]$. In Case (c.2), a coin was flipped (unsuccessfully) at an earlier job arrival epoch resulting in j_ℓ being virtually taken. Job j_ℓ could have been taken with probability at least $(\alpha - 1)/[\delta(\alpha - 1) + 1]$. Note that d_{j_ℓ} is at least $(1/\alpha)d_{j_m}$.

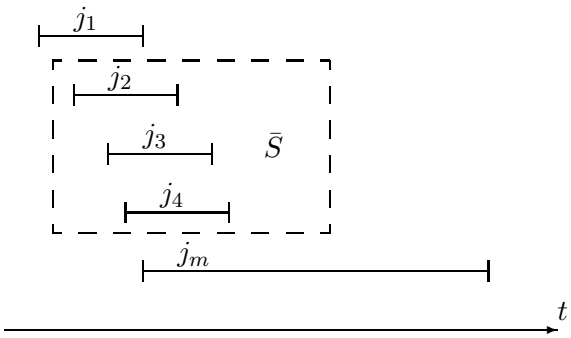


Figure 3.6: When no job in $\bar{S}$ has duration $(1/\alpha)d_{j_m}$ or larger.

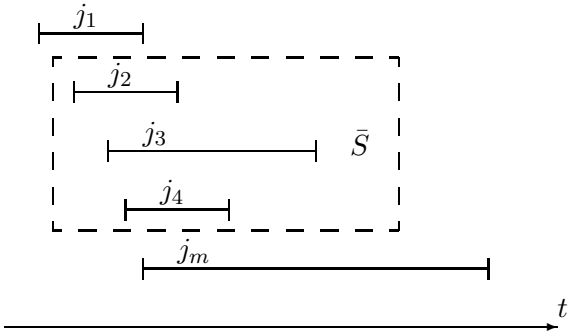


Figure 3.7: When at least one job in $\bar{S}$ has duration $(1/\alpha)d_{j_m}$ or larger (j_3 in this case).

Now we consider the fact that $|S_k| \leq d_{j_m}(1+\alpha^{-1}+\alpha^{-2}+\cdots) < (\frac{\alpha}{\alpha-1})d_{j_m}$, which means $d_{j_m} > (\frac{\alpha-1}{\alpha})|S_k|$. Putting together the above arguments, we have shown that a job of duration at least $(1/\alpha)(\frac{\alpha-1}{\alpha})|S_k|$ would be accepted with probability $(\alpha-1)/[\delta(\alpha-1)+1]$. That is, a lower bound of expected work performed is $(\alpha-1)^2|S_k|/\{\alpha^2[\delta(\alpha-1)+1]\}$.

For Case (2), let j denote the first type-2 job in S_k. Then, j divides S_k into two parts: every job before j is of type-1 and every job starting with j is type-2. Two cases now arise:

(2.1) If all type-2 jobs overlap a type-1 job, we have $|S_k| < (\Delta_2 + \frac{\alpha}{\alpha-1})d_j$. This is the worst case in which $d_j = d_T$, job j overlaps the longest-possible job within the set of type-1 jobs, and another job with $d_{\max} = \Delta_2 d_j$ also overlaps job j. So the part of S_k that contains type-1 jobs cannot be longer than $\frac{\alpha}{\alpha-1}(d_T - \epsilon)$, and the part that contains type-2 jobs cannot be longer than $\Delta_2 d_j$.

The probability that j is accepted equals the probability that all type-1 jobs are virtually taken. This probability is at least $(\alpha-1)/[(\alpha-1)\log_\alpha \Delta_1 + 1]$. Therefore, the expected assigned time in this case is at least $(\alpha-1)|S_k|/\{(\Delta_2 + \frac{\alpha}{\alpha-1})[(\alpha-1)\log_\alpha \Delta_1 + 1]\}$.

(2.2) If not all type-2 jobs overlap a type-1 job, then the myopic approach kicks in as soon as $A(\alpha, d_T)$ encounters its first type-2 job that can be assigned. Whether the type-1 job that overlaps a type-2 job is assigned depends on the random realization of $A(\alpha, d_T)$, but we know for sure that either the first type-2 job that overlaps type-1 job will assigned, or the first type-2 job that does not overlap a type-1 job will be assigned. After it accepts the first type-2 job, the algorithm will accept every non-overlapping job that is of duration at least d_T in a myopic fashion. This implies that it yields a $\frac{1}{\Delta_2+1}$ worst-case performance on part of S_k covered by type-2 jobs. On the part covered by type-1 jobs, the algorithm still assigns at least $(\alpha-1)^2/\{\alpha^2[(\alpha-1)\log_\alpha \Delta_1 + 1]\}$ of the covered time. It is easy to see that both $\frac{1}{\Delta_2+1}$ and $(\alpha-1)^2/\{\alpha^2[(\alpha-1)\log_\alpha \Delta_1 + 1]\}$ are greater than $(\alpha-1)/\{(\Delta_2 + \frac{\alpha}{\alpha-1})[(\alpha-1)\log_\alpha \Delta_1 + 1]\}$. Hence proved. □

From Lemma 3.3.1 and 3.3.2 we find that in Case (1), Algorithm $A(\alpha, d_T)$ assigns at least $(\alpha-1)^2|S_k|/\{\alpha^2[(\alpha-1)\log_\alpha \Delta_1 + 1]\}$ and the distance between S_k and S_{k+1}

is at most $(\alpha^2+\alpha)|S_k|$. Similarly, in Case (2), Algorithm $A(\alpha, d_T)$ assigns at least $(\alpha-1)|S_k|/\{(\Delta_2+\frac{\alpha}{\alpha-1})[(\alpha-1)\log_\alpha \Delta_1+1]\}$ and the distance between S_k and S_{k+1} is at most $|S_k|$. Upon combining these arguments we obtain our main result in this paper, presented in Theorem 3.3.3 below.

Theorem 3.3.3. *The following is an upper bound of the competitive ratio of* $A(\alpha, d_T)$

$$\kappa \doteq \max\{\frac{(\alpha^3-1)\alpha^2[(\alpha-1)\log_\alpha \Delta_1+1]}{(\alpha-1)^3}, \frac{2(\Delta_2+\frac{\alpha}{\alpha-1})[(\alpha-1)\log_\alpha \Delta_1+1]}{\alpha-1}\}.$$

Clearly, Algorithm $A(\alpha, d_T)$ is $O(\Delta_2 \log \Delta_1)$-competitive for the single-processor cases. We investigate how one would choose α and d_T in Section 3.5 after developing performance bounds for the multiple-processor cases.

Before closing this section, we point out that there may be other algorithms that achieve a competitive ratio bound of $O(\log(\Delta))$. For example, one can divide the duration range in intervals $(2^i d_{\min}, 2^{i+1} d_{\min}]$ and set the acceptance probability for each interval to be $(1/\log_2 \Delta)$. Our algorithm is better for two reasons. First, we select acceptance probabilities based on optimizing the finite-step marriage problem. Second, our algorithm is more flexible with parameters (α, d_T), which can be chosen to maximize performance. We use an example next to underscore the importance of using optimal coin-flip probabilities. Assume that job durations occur in lengths of 2^k, $k = 1, \cdots, K$, all jobs in I overlap, and shorter job always arrive first. Using the framework of a finite-step marriage problem with at most K steps, Lemma 3.2.2 provides acceptance probabilities that have a competitive ratio of $(\frac{K+1}{2})$ upon setting $\alpha = 2$. In contrast, if we were to use uniform acceptance probability of $(1/\log_2 K)$, then the competitive ration would have been K. That is, in this example, our approach could provide twice as good worst-case performance.

3.4 Multiple-Processor Online FJS

Our recommended algorithm for dealing with problem instances involving multiple processors is denoted by $A_n(\alpha, d_T)$, where n is the number of processors. As mentioned earlier, the problem is decomposed into two subproblems. We discussed in Section 3.3 how $A(\alpha, d_T)$ decides whether the processor that receives the job will accept it or not. In this section we focus on how to assign arriving jobs to different processors.

Let $F(j) = \{i : \ a_i \leq s_j < e_j \leq b_i\}$ be the set of processors that can perform job j. For each processor, we keep track of the set of *allocated* jobs. This set is denoted by C_i. In addition, we define τ_i as the end time of the last-ending job in C_i, i.e. $\tau_i = \max_{j \in C_i}\{e_j\}$. Here allocated means that a job is routed to processor i but it may or may not be accepted. The decision to accept the job is made after the allocation decision. The act of allocating a job j to processor i may make it necessary to update τ_i, because j may now be the last-ending job in C_i.

$A_n(\alpha, d_T)$ selects a processor in $F(j)$ that has the smallest τ value among processors in $F(j)$. We refer to this part of $A_n(\alpha, d_T)$ as the allocation Procedure G (as shown below). Every job is allocated by G because every job can be processed by at least one processor and G does not consider potential overlap with previously allocated jobs when allocating a job to a processor.

Allocation Procedure G

Step 0 when job j arrives at time t, identify $F(j) = \{i : \ s_j \geq a_i, \ e_j \leq b_i\}$,
Step 1 find $i = \min\{k : \ \tau_k \leq \tau_{k'}$ for every $k, \ k' \in F(j), \ k \neq k'\}$,
allocate job j to processor i and update τ_i.

We show next that G guarantees the utilization of at least half of the optimal number of processors that could be utilized at each time epoch t. To do so, we establish the connection between our allocation problem and the bipartite matching problem and argue that the optimal value of the bipartite matching problem is an upper bound of the number of processors that can be allocated. Then, we arrive at the desired conclusion by showing that the allocation procedure always allocates at least half of the upper bound.

For an arbitrary time $t \in [\underline{s}, \overline{e}]$, we define $J(t) = \{j : \ s_j \leq t < e_j\}$ as the set of jobs that need processing at t, and $P(t) = \{i : \ a_i \leq t < b_i\}$ as the set of processors that are available at t. We associate a node with each job in $J(t)$ and with each processor in $P(t)$. Then we connect two nodes, one associated with a job j and the other with a processor i, with an undirected edge if and only if job j can fit in processor i's shift, i.e. if $a_i \leq s_j < e_j \leq b_i$. This exercise gives rise to a bipartite graph. The relevant matching

problem is to find the maximum number of unconnected edges[1] . The maximum matching problem thus defined is related to the job-to-processor allocation problem as shown in Lemma 3.4.1.

Lemma 3.4.1. *For each t, let $w(t)$ be the number of matched pairs in an optimal solution to the maximum matching problem. Then, in any job-to-processor allocation procedure, the number of processors that are allocated at least one job at time t cannot be greater than $w(t)$.*

Proof: Because we focus on time t, the relevant sets of jobs and processors are those that belong to $J(t)$ and $P(t)$, respectively. For a given job-to-processor allocation, assume $m'(t)$ processors are allocated at least one job. Then, by choosing one allocated job from each of the $m'(t)$ allocated processors, we can find $m'(t)$ job-to-processor pairs, and this corresponds to an $m'(t)$-matching. Because $w(t)$ is the optimal value of the bipartite matching problem, it immediately follows that $m(t)' \leq w(t)$. □

Next we establish the competitive ratio of the allocation procedure. For this purpose, we note that if for some processor i, $\tau_i \geq t$, then that processor is already matched in the equivalent bipartite matching problem. That is, allocating a job to processor i does not increase the number of matched processors. In contrast, $\tau_i < t$ indicates that processor i is not matched at time t and allocating a job to that processor does increase the number of matched processors.

Lemma 3.4.2. *The number of processors allocated by Procedure G at time t is at least $w(t)/2$.*

Proof: Let $m'(t)$ denote the number of processors that are allocated at least one job at t. Procedure G chooses a feasible processor with the smallest τ when a new job arrives. This implies that this procedure always selects an unmatched processor that connects to the current job whenever an unmatched processor exists. Then, the statement of the Lemma follows from the well-known result that a greedy algorithm that picks a new edge (excluding those already selected), achieves a performance bound of at least 1/2

[1] Note that a connected edge means either that a job is assigned to more than 1 processors or that a processor handles more than 1 jobs at the same time t. Connected edges therefore represent infeasible assignments in our problem setting, which must be avoided.

([51]). If there are no unmatched processors at t, then $m'(t) = w(t)$, completing the proof. □

Next, we provide an example to show that the performance bound in Lemma 3.4.2 is tight.

Theorem 3.4.3. *Procedure G is 2-competitive and no allocation procedure can be better than 2-competitive.*

Proof: We showed in Lemma 3.4.2 that G cannot be worse than 2-competitive. In this proof, we provide an example to establish that the bound is tight and that no allocation procedure can be better than 2-competitive.

Suppose there are two processors, labeled P_1 and P_2, with shift start and end times $(0, 10)$ and $(4, 14)$, and three jobs, labeled j_i, $i \in \{1, 2, 3\}$, with start and end times $(1, 6)$, $(4, 9)$, and $(7, 12)$. Note that job j_1 can be allocated only to processor P_1 and job j_3 only to processor P_3. It should be clear that upon executing Procedure G, j_1 will be allocated to P_1, and j_2 and j_3 to P_2. Next, we observe that at $t = 8$, job j_2 can be matched with P_1 and j_3 with P_3. That is, $w(8)$ is 2 but Procedure G allocates only P_2 at $t = 8$. This proves that 2 is the exact competitive ratio of Procedure G.

Next, consider time points 5 and 8. It is easy to see that $w(5) = w(8) = 2$. Given that j_1 can be allocated only to P_1 and j_3 only to P_3, if a procedure allocates j_2 to P_1, then $m'(5) = 1$, whereas if it allocates j_2 to P_2, then $m'(8) = 1$. That is, no allocation procedure can achieve more than half of $w(t)$ simultaneously for both time points 5 and 8. Put differently, no procedure can be better than 2-competitive. □

Corollary 3.4.4. *Procedure G is an optimal procedure when either processor shifts are identical or shift lengths are unlimited.*

Proof: Recall that no job in I arrives earlier than the shift start time of at least one processor and ends later than the shift end time of at least one processor. When processor shifts are either identical or unlimited, it means that every job can be allocated to every processor. In the bipartite graph, this means that each j in $J(t)$ is connected to every i in $P(t)$ by an edge. In this instance, $w(t) = \min\{|P(t)|, |J(t)|\}$, and Procedure G will allocate $w(t)$ processors. The claim is based on the argument that Procedure G will allocate each arriving job to a processor indexed i that has $\tau_i < t$, if such a processor

exists. Note that if all processors are already matched, then $w(t) = |P(t)|$ proving the claim above. By picking a processor that has not been matched before, Procedure G increases the number of matches each time until that number reaches $\min\{|P(t)|, |J(t)|\}$. □

With a 2-competitive allocation procedure in hand, we can establish the overall competitive ratio of our approach in Theorem 3.4.5.

Theorem 3.4.5. *An upper bound of the competitive ratio of $A_n(\alpha, d_T)$ is*

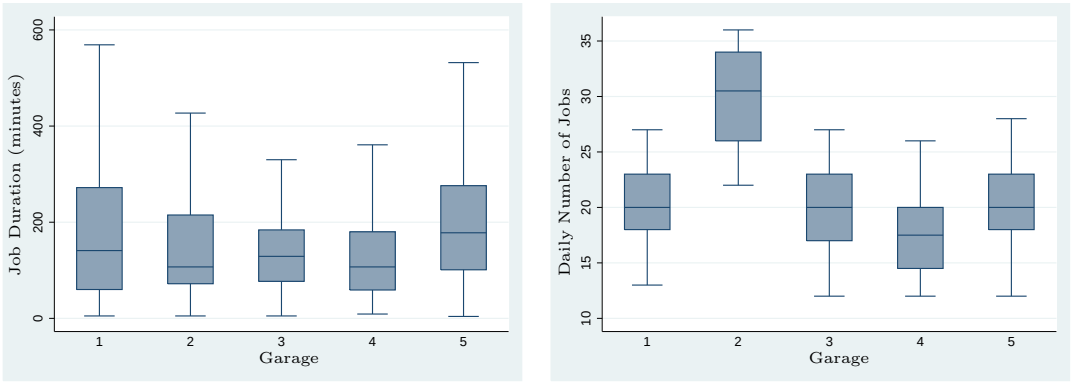
$$2 \times \max\{\frac{(\alpha^3-1)\alpha^2[(\alpha-1)\log_\alpha \Delta_1 + 1]}{(\alpha-1)^3}, \frac{2(\Delta_2 + \frac{\alpha}{\alpha-1})[(\alpha-1)\log_\alpha \Delta_1 + 1]}{\alpha-1}\}.$$

3.5 Numerical Experiments

We obtained data concerning call driver operations for five randomly-picked months from Metro Transit. The agency had 5 garages and each month's data came from a different garage. The data included all jobs that were assigned to either call operators or on an overtime basis. The agency did not drop service in the period for which we obtained the data, which meant that the jobs in our data set represented all open work. We considered each day as an independent instance, and excluded weekends from our experiments because weekends have a special service schedule. This left 100 problem instances, one for each weekday of operations (5 garages × (≈) 20 days per garage).

The shortest job was approximately half an hour long, and the longest job possible was 8 hours long, with a few exceptions. These exceptions include cases when absentee drivers worked a non-standard shift (either 9- or 10-hour shift) or when there are very short pieces of work. Because the exceptions are rare, we set $\Delta = 16$ in our experiments. Figure 3.8(a) in shows a box plot of job durations by garage. We see that Garage 1 and 5 have jobs with more variable durations and that Garage 3's job durations do not vary as much. The available time of each driver is a continuous 8-hour period, but drivers start at different time of day, in order to cover the work day of about 20 hours. The shift start and end times are read from data.

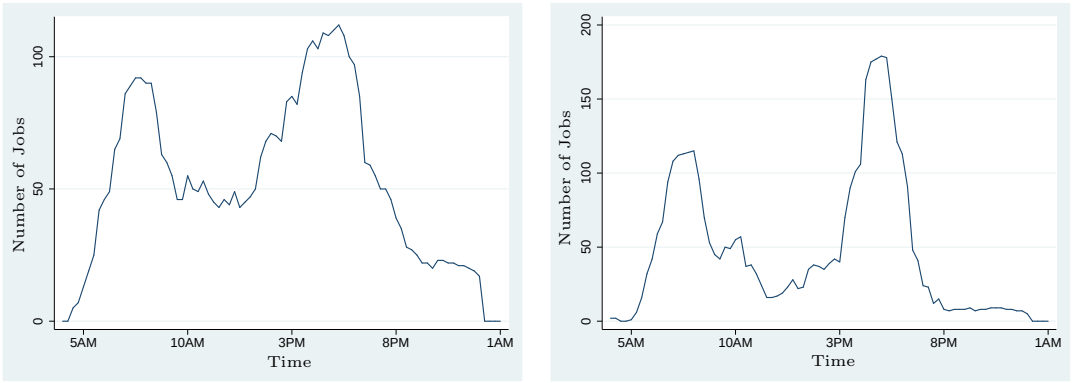
Box plots in Figures 3.8(b) and 3.10 show the variation in problem size by garage. The daily number of on-call reserve drivers ranged from 7 to 18 and the daily number of jobs ranged from 8 to 36. The job-to-driver ratio also varied significantly across garages.



(a) Job Durations by Garage (b) Daily Number of Jobs by Garage

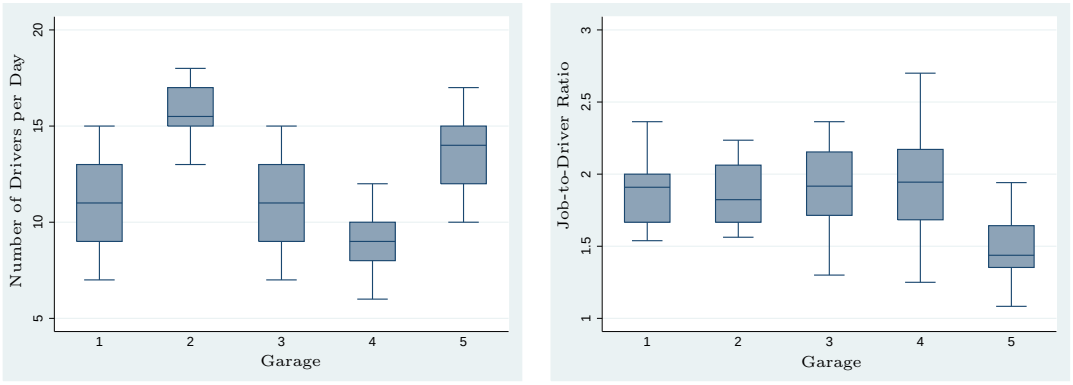
Figure 3.8: Job Durations and Daily Frequency

We graphed the demand profile by time of day for Garages 1 and 3 in Figure 3.9 and found differences in the pattern of job arrival times. Figure 3.9 was obtained by dividing time into 15-minute intervals and counting the number of jobs that were active in each interval across all weekdays for each garage. We observed that every garage experienced two peaks each day, but Garage 3's peaks were sharper. Pattern differences observed in Figures 3.8-3.9 influence the performance of our algorithm, which we explain later in this section.



(a) Garage 1 (b) Garage 3

Figure 3.9: Demand Profile by Time of Day



(a) Driver Numbers by Garage

(b) Job-to-Driver Ratio by Garage

Figure 3.10: Daily Driver Availability and Job-to-Driver Ratio

For comparing performance of our algorithm, we chose myopic approach as the straw-man policy. There are several reasons for doing so. First, the myopic approach is the most commonly used approach in practice. Second, in addition to ensuring that work assignment algorithms have bounded worst case performance, transit agency managers are also interested in achieving good average performance. Myopic or greedy algorithms generally produce good average performance and in some cases, it can be proved that their worst-case performance is also good, e.g. in the maximum coverage problem ([52]), the greedy algorithm achieves theoretically best worst-case performance ([53]). Finally, the myopic approach is the most conservative policy that accepts jobs in their arrival sequence so long as a feasible assignment is possible. Our algorithm contains a parameter d_T that provides a hedge between myopic and strategic approaches.

For each garage, we searched for the best parameters by comparing the results for different combinations of (α, d_T), where $\alpha \in [1.1, 2.4]$ (in increments of 0.1) and $d_T \in [90, 240]$ (in increments of 10 minutes). We used two different search criteria. In one case, α and d_T were chosen to maximize worst-case performance of the randomized algorithm, and in the other case to maximize its average performance. Note that selected parameters vary by garage, which is caused by differences in patterns of job durations, frequency, and driver availability as explained in Figures 3.8-3.9. Also, the performance of optimal values (α^* and d_T^*) was generally not significantly different from other values of α and d_T that were close to the optimal values. Both sets of results are reported in

Table 3.3. For each day, we simulated 10 runs of $A_n(\alpha, d_T)$ to obtain an estimate of its expected performance. We report the percentage of total job time that each algorithm assigns averaged over 20 days of data for each garage. Columns 2–4 show comparisons when parameters of the randomized algorithm were selected to maximize worst-case performance and columns 5–7 show similar results for average performance. Finally, columns 8–9 show the average advantage from the randomized algorithm. We observe that the use of the randomized algorithm could save a total of 1091 minutes, or 18.2 hours, of overtime daily. At approximately \$42 per hour in overtime wages, this could save approximately \$190,925 annually (assuming 250 week days in a year).

In addition to experiments based on actual service days, we also simulated problem instances by sampling from the set of all jobs and driver shifts in our data set. There were three reasons for performing such experiments. First, we could in this way generate many more than 100 problem instances. Second, we could design experiments with different problem sizes: i.e. numbers of jobs and drivers. Third, we could test the performance of the algorithm in a realistic setting by selecting parameters α and d_T based on a training data set and testing the realized performance over randomly generated test data.

Problem instances in our experiments have different jobs-to-driver ratios but preserve the ranges we observed in the data. Specifically, we observed that the daily minimum, median and maximum numbers of drivers in the 100 problem instances were 8, 13, and 17. Similarly, the daily minimum, median and maximum job-to-driver ratios were 1.0, 1.6, and 2.3. Therefore our simulated instances have 9 different combinations. These are: 8 processors with 8, 13, or 18 jobs; 13 processors with 13, 21, or 30 jobs, and 17 processors with 17, 27, or 39 jobs (see Tables 3.6 and 3.7).

Tables 3.4 and 3.5 show a summary of simulated data. We consider each day as one sample that gives rise to that day's average job duration and average shift start time. Then, treating each of these averages as a single observation, we calculated the summary statistics in Tables 3.4 and 3.5. To explain these statistics further, we discuss two example. In Table 3.4, the minimum average duration of 66 means that among 300 days of simulated data with 8 jobs, the minimum daily average duration was 66 minutes. Similarly, in Table 3.5, among 300 simulated days with 8 drivers, the minimum daily average shift start time was 5:01 AM.

We generated 300 random instances for each set of parameters independently as training and test data. Parameters α and d_T were selected to maximize randomized algorithm's worst-case performance on the training data. The same values, denoted by α^* and d_T^*, were then applied to the test data. Similar to Table 3.3, this involved searching for best α and d_T over a range of possible values ($\alpha \in [1.1, 2.4]$ in increments of 0.1 and $d_T \in [90, 240]$ in increments of 10). The selected parameters provided the best worst-case performance (which is shown in parentheses in Tables 3.6 and 3.7), and the average performance was based on the same parameters. For each problem instance, we simulated $A_n(\alpha, d_T)$ ten times to estimate the expected performance. The number of repetitions were limited because of computational burden and because expected performance was quite stable over ten simulations.

Table 3.7 shows that $A_n(\alpha^*, d_T^*)$ performs better on both worst-case and average performance in all problem instances relative to the myopic approach. The relative advantage of our approach, which we call Daily Advantage in Table 3.7, was measured as average percentage of extra work assigned and as additional average minutes of work assigned on each day of operations with each set of parameters. For transit agencies with large garages (i.e. with many jobs and drivers), the randomized algorithm has the potential to save significant overtime costs. The performance of both algorithms in experiments involving the simulated data is not as good as with real data because in the simulated data, drivers' shift schedules were determined by a random pick. In contrast, dispatchers assigned start times one day before each day of operations and they were able to better match shift starts to the time-of-day pattern of job arrivals in their respective garages.

Table 3.7 suggests that the job-to-driver ratio affects relative performance. As expected, the average percentage of work assigned decreases as the job-to-driver ratio increases. However, generally more work is assigned when there are more drivers and concomitantly more jobs. This is not surprising because with more jobs and more drivers, there are more opportunities to benefit from the use of randomized algorithm.

3.6 Concluding Remarks

Transit agencies use reserve drivers to cover work that arises from planned and unplanned time off, equipment breakdowns, weather, and special events. In this paper, we developed a randomized algorithm that can be used to improve utilization of reserve drivers who take care of unanticipated work. In the highly random environment of reserve driver scheduling, a decision algorithm needs to tradeoff the reward that would be realized if the current job is accepted against a potentially higher reward from a future job, which may be rejected on account of the earlier decision. Because both the timing and durations of such jobs vary significantly from one day to the next, a reasonable objective for transit agencies is to try to achieve the best worst-case performance. Our randomized algorithm guarantees performance no worse than a certain threshold of the best possible performance, where the latter is realized if all pieces of work are known before making work assignments and such assignments are made optimally. The algorithm strategically assigns some work to overtime drivers to improve overall utilization of reserve drivers. The randomized algorithm is easy to implement and could help transit agencies reduce personnel costs.

Although the focus of this paper on algorithm development, it offers some generalizable insights. First, by considering only those jobs whose lengths are increasing in a geometric sequence, the randomized algorithm leads to worst-case performance that is of order $\log \Delta$. This implies that the performance of our algorithm does not degrade substantially when job durations cover a wide range. In contrast, the best deterministic algorithm has performance of order Δ, which does degrade linearly. This establishes that the desirability of using randomized algorithms increases with Δ. Second, Δ affects the likelihood of finding a value of α that makes the virtual algorithm more competitive than the myopic approach. The larger the value of Δ, the larger this likelihood. However, with a relatively large value of α, a randomized algorithm takes more risk (rejects more jobs) and could perform poorly on average. This risk can be counterbalanced by introducing a parameter such as d_T. That is, randomized algorithms, which choose optimal α and d_T, are likely to result in desirable outcomes for other highly random scheduling environments.

Table 3.2: Notation

$n =$	number of processors
$i =$	processor index; $i \in \{1, \cdots, n\}$
$(a_i, b_i) =$	start and end time of the shift of processor i, where $a_i < b_i$
$j =$	job index
$(s_j, e_j) =$	start and end times of job j, where $s_j < e_j$
$d_j =$	duration (weight) of job j; $d_j = e_j - s_j$
$d_{\min} =$	minimum job duration
$d_{\max} =$	maximum job duration
$\Delta =$	$d_{\max}/d_{\min}$
$\alpha =$	criterion for candidacy of a job when processor is virtually taken
$d_T =$	threshold duration
$\Delta_1 =$	$d_T/d_{\min}$
$\Delta_2 =$	$d_{\max}/d_T$
$j_1 =$	index of a job that starts a marriage problem sequence
$v =$	current virtually taken job index
$\ell =$	$\lfloor \log_\alpha d_v/d_{j_1} \rfloor$; degree of current virtually taken job ($\lfloor \cdot \rfloor$ returns the integer floor)
$r =$	$\lfloor \log_\alpha d_j/d_{j_1} \rfloor$; degree of current job
$\delta =$	$\lceil \log_\alpha d_{\max}/d_{j_1} \rceil$ is called the D-level corresponding to j_1 (only if $d_{j_1} < d_T$, $\lceil \cdot \rceil$ returns the integer ceiling)
$y_k =$	distance between S_k and S_{k+1}
$C_i =$	considered set of processor i
$F(j) =$	$\{i : \ s_j \geq a_i, \ e_j \leq b_i\}$: the set of processors that can perform j
$\tau_i =$	$\max_{j \in C_i}\{e_j\}$
$t =$	an arbitrary time epoch
$J(t) =$	$\{j : \ s_j \leq t < e_j\}$: the set of jobs that cross t
$P(t) =$	$\{i : \ a_i \leq t < b_i\}$: the set of processors whose shifts cross t
$w(t) =$	the optimal value of bipartite matching problem defined at t

Table 3.3: Performance Comparison – Myopic versus $A_n(\alpha, d_T)$

Garage	Worst Case			Average			Daily Advantage	
Number	Myopic	(α^*, d_T^*)	$A_n(\alpha^*, d_T^*)$	Myopic	(α^*, d_T^*)	$A_n(\alpha^*, d_T^*)$	Percent	Minutes
1	27.1	$(1.7, 210)$	35.0	50.5	$(1.5, 200)$	53.6	6.1	285
2	51.8	$(1.8, 140)$	57.5	68.5	$(1.9, 200)$	70.6	3.1	167
3	48.8	$(1.4, 110)$	53.8	69.4	$(1.6, 210)$	71.8	3.5	110
4	38.1	$(1.3, 120)$	48.0	65.0	$(1.5, 170)$	69.7	7.2	206
5	42.8	$(1.4, 210)$	46.0	61.3	$(1.9, 240)$	65.0	6.0	323

Performance = percentage of total work assigned.

Table 3.4: Job Duration Summary (minutes), SD= Standard Deviation

# of Jobs	Min	Max	Mean	SD
8	66	340	175	55
13	80	307	175	39
17	90	290	173	34
18	98	314	172	34
21	91	293	173	35
27	84	247	171	29
30	98	286	173	36
39	88	307	172	34

Table 3.5: Driver Shift Start Time Summary

# of Drivers	Min	Max	Mean
8	5:01 AM	12:34 PM	8:52 AM
13	5:10 AM	12:41 PM	8:56 AM
17	6:08 AM	12:18 PM	9:00 AM

Table 3.6: Parameter Selection Using Training Data Set

Jobs, Drivers	Performance Myopic	(α^*, d_T^*)	Performance $A_n(\alpha^*, d_T^*)$
8, 8	56.1 (16.8)	(1.5,160)	59.8 (19.6)
13, 8	45.6 (17.5)	(1.4,230)	54.2 (22.4)
18, 8	36.8 (13.8)	(1.6,230)	50.2 (18.2)
13, 13	60.8 (24.1)	(1.4,230)	63.6 (24.6)
21, 13	47.4 (23.3)	(1.1,140)	61.7 (36.0)
30, 13	36.2 (13.9)	(1.2,210)	53.9 (27.8)
17, 17	60.5 (27.8)	(1.1,190)	66.8 (31.8)
27, 17	48.5 (19.5)	(1.9,230)	63.2 (32.1)
39, 17	37.6 (18.5)	(1.9,120)	59.0 (32.9)

Performance = percentage of total work assigned.

Table 3.7: Performance Comparisons Using Test Data Set

Jobs, Drivers	Performance Myopic	(α^*, d_T^*)	Performance $A_n(\alpha^*, d_T^*)$	Daily Advantage	
				Percent	Minutes
8, 8	55.8 (10.5)	(1.5,160)	58.8 (17.1)	5.4	42
13, 8	45.0 (12.2)	(1.4,230)	53.6 (15.6)	19.1	193
18, 8	36.4 (9.5)	(1.6,230)	49.8 (10.9)	36.8	417
13, 13	59.1 (13.2)	(1.4,230)	62.8 (22.5)	6.3	83
21, 13	46.4 (16.7)	(1.1,140)	60.4 (25.3)	30.2	508
30, 13	36.9 (17.8)	(1.2,210)	55.9 (27.8)	51.5	986
17, 17	61.0 (23.2)	(1.1,190)	67.8 (29.6)	11.1	200
27, 17	48.8 (15.8)	(1.9,230)	62.3 (24.2)	27.7	630
39, 17	40.5 (19.5)	(1.9,120)	59.1 (32.2)	45.9	1255

Performance = percentage of total work assigned.

Chapter 4

Improving Operating Room Schedules

4.1 Introduction

Operating rooms (ORs) in US hospitals generate about 70% of revenues and 20-40% of operating costs while operating at a staffed capacity utilization of 60-70% ([54]). ORs are also responsible for a significant proportion of hospital admissions ([55]) and a recent estimate puts the cost of a staffed OR at approximately $15-20 per minute ([56]). Therefore, hospitals spend considerable administrative resources to ensure that OR time is used efficiently. A typical scenario in many hospitals is that each day the OR management team looks at the two-day-ahead surgical schedule, and tries to manually revise case start times to reduce the number of operating rooms that would need to run concurrently. This reduces staffing costs. At this point in time, there is already a surgical schedule in place with planned start times and planned surgical case lengths. The latter are provided by scheduling software used by hospitals, with some adjustments based on discussions with the surgeons at the time of booking procedures. The management team treats the surgical case lengths as fixed, changing only the case start times. The purpose of the model we develop is to aid in this daily schedule revision process.

Although regular staff salaries are already committed, staff can be asked to either change their work schedules, or work in a different area of the hospital. Because hospitals

regularly use extra and overtime shifts to meet staffing needs, reducing OR staffing requirements reduces overall staffing costs. Alternatively, freed-up OR time can be used to increase case volumes and revenue.

The OR rescheduling problem mentioned above is a variant of the bin-packing problem with bins being the staffed ORs and items or jobs being the surgeries. The goal is to minimize the weighted sum of bins used (i.e. cost of staffed ORs), where the weight of a bin is proportional to its size. Two features of the OR rescheduling problem that make it different from problem formulations studied in the literature are (1) surgeries performed by the same surgeon must not overlap, and (2) hospitals may employ staff with different shift lengths. The bin-packing and therefore the OR rescheduling problems are NP hard. Therefore, we establish a lower bound on the cost of staffing ORs that is guaranteed to be at least (2/3) of the optimal staffing cost for any subset of surgeries. The lower bound is used in a branch-and-bound algorithm developed to solve the rescheduling problem. Upon testing our approach on data obtained from three hospitals, we identify significant opportunities for reducing OR staffing costs. We also analyze resulting OR schedules to study how rescheduling would affect surgeons' work days, delays in surgery start times, and overtime usage.

OR management practices vary from one hospital to another. We present the ensuing institutional background as broadly representative of common practices at US hospitals that allocate periodically occurring (e.g. weekly, biweekly or monthly) blocks of OR time to individual or groups of surgeons as guaranteed allocation. Surgeons holding blocks may book surgeries in their blocks up until the auto-release date. On the auto-release date, which may occur between 0 to 14 days in advance of the day of surgery, any unused block time reverts back to the hospital. This OR time may be used either by surgeons who do not have assigned blocks or by those whose demand exceeds their block times, or for urgent and emergent cases. Non-block surgeons' cases are typically booked on a first-come-first-served basis. Different hospitals may follow different approaches to deal with urgent cases. Some schedule urgent blocks with zero auto-release dates, whereas some others reserve dedicated ORs for urgent and emergent cases. All hospitals also try to "fit" urgent cases into available open times between scheduled non-urgent cases. Finally, any remaining urgent cases are scheduled as add-on cases at the end of shifts, incurring overtime charges.

Because staffed OR utilization is low even after hospitals' manual attempts to reorganize surgery schedules (see Section 4.3 for details), we focus in this paper on developing an algorithm that would allow hospitals to accommodate the same number of surgeries with fewer staffed ORs by reworking the case start times. Such rearrangements are often feasible because blocks typically have 2 to 5 day auto-release dates and surgeons are willing, within reason, to accept some changes to the start times of their cases. Moreover, patients are typically asked to arrive several hours before the start of their surgeries to prevent delays due to peri-operative activities, which facilitates rescheduling.

Our algorithm for improving OR schedules significantly reduces the number of open times between scheduled non-urgent cases. Therefore, upon implementing this approach, hospitals would need to schedule dedicated ORs for handling urgent cases. The management team would need to commit to having these ORs staffed before knowing the true urgent and emergent demand. We evaluate the performance of our algorithm both with and without accounting for urgent cases. In each case, our approach results in significant total cost savings, including overtime charges. There is emerging evidence in biomedical literature that dedicated urgent/emergent ORs also help improve health outcomes – see, e.g. [57].

We develop a specialized algorithm for the OR rescheduling problem, rather than use a general-purpose optimization software, because the latter neither provide insights into the nature of a surgeon's cases that trigger the use of a staffed OR, nor solve all instances of problems encountered at typical hospitals. Moreover, a recent survey of hospital executives found that top among information technology solutions that hospital executives believed might help improve OR operations was "scheduling: better/accurate scheduling" ([58]), and recent healthcare innovations – e.g. accountable care organizations (ACO) and bundled payment care improvement initiative (BPCI) – allow hospitals and physicians to find efficiency enhancing strategies and share rewards that come from reduced costs ([59], [60] and documents posted on the BPCI web site at `http://innovation.cms.gov/initiatives/bundled-payments/`). Our approach may be viewed as an attempt to quantify the benefit of cooperation. The ability to quantify such gains is a needed first step before developing models for gain sharing. Another feature of our approach is that the developed algorithm can be integrated with a hospital's OR scheduling software via an Application Programming Interface.

A key contribution in this paper lies in establishing a performance guarantee for a lower bound on the cost of staffing ORs to accommodate a given set of surgeries in the presence of overlap-avoidance constraints. We present an example next to highlight the difficulty involved in doing so. Suppose 6 surgeons are scheduled to operate on a particular day and the hospital uses a single shift of length T. Suppose the first five surgeons need to perform three surgeries, each with duration $(4/15)T$, for a total duration of $(4/5)T$. The sixth surgeon needs to perform five surgeries with duration $(1/5)T$ each, for a total duration of T. If we ignore the constraint that surgeries performed by the same doctor must not overlap, we will find that a lower bound and optimal solution is to use 5 rooms, one for each of the first five surgeons with $(1/5)T$ in each room assigned to the sixth doctor. However, this solution is not feasible because at least one of the six doctors' surgeries cannot be assigned in a way that they do not overlap. In fact, a feasible solution for scheduling ORs is to use six rooms. This happens because each of the six doctors must be working at $(1/2)T$ regardless of how his or her surgeries are arranged.

We develop a framework to overcome the problem illustrated in the above example. This framework involves three steps. In step one, we classify connected sequences of surgeries that we call "chains." We also classify doctors into different categories based on the properties of chains formed by their surgeries. Surgeon classification is used in step two to assign surgeries to ORs in a particular sequence, which not only produces a lower bound but also helps us in step three to recover a feasible solution that is no more than (3/2) of the lower bound.

Because our algorithm may increase surgeons idle or unused time, we test the quality of our solution by applying our algorithm to data from three hospitals. This leads to several insights. First, rescheduling works as expected by flattening the peak number of concurrently staffed ORs and scheduling surgeries uniformly throughout the day. We also find that efficiency is greater when a hospital has the flexibility to schedule some long shifts because that leads to more efficient packing of surgical cases. Second, efficiency gains come at the expense of increased staff overtime, surgeon idle time, and the total amount of time that surgeons spend at the hospital. We quantify the impact of rescheduling on surgeons and staff and find that savings from efficiency gains are high, suggesting that hospitals may be able to obtain surgeons' cooperation through an

appropriate gain sharing plan.

literature in Section 4.2, summarize data from three hospitals in Section 4.3, formulate the model in Section 4.4, and present key analytical results in Sections 4.5 and 4.6. Numerical experiments that utilize real data are presented in Section 4.7 and we conclude the paper in Section 4.8.

4.2 Literature Review

There are three bodies of literatures that are related to our work. These are OR/surgery scheduling, bin packing, and resource-constrained scheduling. We position our work next in relation to each of these literatures.

Surveys of OR scheduling literature are provided in several recent papers; see, for example, [61], [62], [63], [64], and [65]. OR capacity planning problems fall into six broad categories: (i) determining the number of ORs and the equipment/capability of each OR, (ii) determining staffing needs and corresponding shift lengths, (iii) assigning blocks of OR time to surgeon groups or individual surgeons, (iv) putting in place booking rules for the use of OR time and the release of exclusive blocks, (v) rescheduling, and (vi) coping with day-of-surgery variations. Planning problems in each category arise with different frequency and therefore relevant models need to consider different levels of granularity and time scales. For instance, the problem of determining the number of ORs and equipment may be revisited once every few years and relevant models may consider aggregate demand over a quarter or a year. It may be appropriate for such models to assume that surgeries are packed in a fluid fashion. In contrast, when choosing planned start times of surgeries, resulting schedules must fit surgeries into available shift lengths.

Surgery scheduling problems, i.e. problems mentioned in items (iv) and (v) above, can be categorized in several different ways. For example, by the assumed booking protocol (*online* or *offline*), by procedure lengths (*constant* or *random*), and by urgency status (*emergent/urgent* or *non-urgent*) ([65]). Online scheduling occurs when surgeries are booked one at a time. Offline means all requests for surgeries that need to be performed on a particular day are known before determining scheduled procedure lengths and the sequence in which surgeries will be performed. In many US hospitals, surgeries

are booked in an online fashion, booking clerks assume that estimated case lengths are constant, and non-urgent cases are booked first, followed by urgent and emergent cases. Case length estimates may depend on a whole host of factors including the surgery types, patients' characteristics, and track records of surgeons performing the surgeries. When surgeons holding blocks book non-urgent cases, they determine the sequence in which surgeries will be performed. In contrast to what is common in practice, the Operations Management literature focuses primarily on the problem of determining the scheduled duration and the sequence of surgeries assuming surgeries are booked offline. Most papers in the scheduling literature consider only one urgency type, i.e. they focus either entirely on non-urgent cases, or entirely on urgent/emergent cases. Articles that consider both types, e.g. [66], do not model discrete surgery durations. That is, they assume that surgeries can be scheduled in a fluid fashion.

Uncertain actual surgery duration is an important consideration in surgery planning and scheduling. Consequently, many papers focus on the problem of estimating time allowances for different surgical procedures. Two variants of such models exist. All models assume an offline scheduling environment. In the first case, surgery durations are random but their distributions are assumed known or it is assumed that actual durations can be sampled from an existing database of surgery durations. In the second case, surgery durations are unknown. Examples of models of the former type can be found in [67], [68, 69], and [70], whereas an example of the latter can be found in [71]. In these models, overlap-avoidance constraints are absent, which is a key feature of the analysis presented in this paper. The absence of overlap-avoidance constraints is justified by considering either only one OR, or assuming that surgeons work in the same OR on a single day. In contrast, our data show that surgeons routinely operate in several rooms on a given day. Also, our objective is not to determine optimal time allowances for different surgical procedures. Instead, we focus on creating surgery schedules that require fewer staff shifts upon assuming that hospitals' estimates of case lengths are not affected by sequencing of cases. This assumption is commonly made by practitioners. We present supporting evidence in Section 4.3.

The above-mentioned problem types, the importance of considering uncertainty,

scheduling constraints, and possible concern for smoothing downstream resources (e.g. hospital beds) give rise to many variants of the surgical scheduling problem. Because structured reviews exist that discuss each problem class, we do not describe these problem instances in detail, except to point out that none of the existing models addresses the problem of rescheduling surgeries to minimize staffing costs. There are a few papers, however, that consider the difficulty of scheduling surgeries when surgeons perform multiple surgeries on the same day and overlap avoidance is an important consideration. We discuss these papers below.

[72] model the constraint that scheduled surgeries performed by the same surgeon must not overlap. The authors formulate the daily surgery scheduling problem as a two-stage hybrid flow-shop problem with the objective of minimizing the cost induced both by the ORs and the recovery rooms. A hybrid genetic algorithm is proposed to solve this model. The paper does not provide either bounds or performance guarantees, which are key elements of our approach. [73] consider the surgery allocation problem in an ambulatory surgical center. The authors formulate the sequencing step as a variant of the two-stage no-wait flow shop scheduling problem. A tabu search based heuristic is used to find a near-optimal solution. Once again, neither bounds nor performance guarantees are established.

Turning next to the bin-packing and machine scheduling literatures, we find several problem formulations that have elements in common with the problem studied in this paper. Our problem is closer to bin-packing as opposed to machine-scheduling (see [74] for a survey of machine-scheduling literature) because our goal is to find the minimum number of ORs (bins) needed to fit all procedures. In contrast, in machine scheduling problems, the number of machines is known and the goal is to minimize the makespan, i.e. to complete all work at the earliest possible time. However, our problem has some features of machine scheduling because we do have the constraint that procedures done by the same surgeon cannot overlap.

[75] provide a review of the online and offline approximation algorithms for bin packing. Our setting is offline and our problem is significantly different from the standard bin-packing problem because we incorporate overlap-avoidance constraints. Upon placing such constraints, existing offline bin-packing algorithms may not even find a feasible solution if applied to our problem because same-surgeon jobs may overlap. Overlap

avoidance provides a natural segue into a discussion of papers on bin packing with conflicts ([76]). Given a set of items, the goal in such problem formulations is to find a partition of items such that items that are predefined to be in conflict cannot be placed in the same bin. Conflicts in this setting are "horizontal" – i.e. they need to be avoided when certain items are placed in the same bin. In contrast, in our problem, the conflicts are "vertical" – i.e. overlap in time must be avoided across all bins for jobs performed by the same surgeon.

Many papers in the bin-packing literature present online algorithms, i.e., heuristically pack items one at a time (see [75], and [77]). The online bin-packing literature also includes problems with variable-sized bins (see [78], and [79]), which is relevant because we consider different shift lengths. In this literature, it is not common for papers to focus on developing lower bounds on the number of bins needed. This is the case in part because online algorithms do not rely on a branch-and-bound type approach and the need to develop lower bounds does not arise. In contrast, we provide lower bounds for the relevant problem formulation and show that it is no less than (2/3) times the cost of a feasible solution.

Key papers on resource-constrained scheduling are [80], [81], and [82], who define the problem as that of minimizing the makespan of a set of unit-length independent jobs that cannot be scheduled before their start times on identical processors. Each job needs a certain amount of each resource from a set of available resources. All resources are available throughout the planning horizon, but the available quantity of each resource is bounded. [82] provide a $(1+\epsilon)$-approximation algorithm for such problems. Even with the unit job-length assumption, the approximation algorithm studied in these works does not apply to our setting because of two reasons. The first reason is that these models assume that the number of machines (ORs in our model) is fixed, and minimize the finish time of jobs (i.e. makespan). A guaranteed bound of the scheduling problem does not result in a guaranteed bound to our problem of minimizing the number of ORs. Second, we consider a model with two shift lengths, which makes our problem significantly different from the resource-constrained scheduling problem.

Another related stream of work concerns resource-constrained project scheduling in which each activity has a potentially different processing time and the goal is to minimize

the makespan; see the survey in [83]. Papers in this literature usually develop branch-and-bound algorithms. The lower bounds on the makespan are calculated by solving a relaxed problem, e.g. by relaxing the resource constraints ([84]), or the precedence constraints and allowing preemption ([85]). The key difference relative to our approach is that the lower bound in these papers is for makespan, which does not translate into a lower bound for the number of bins needed.

4.3 Data

We obtained surgical scheduling data from three hospitals, which included scheduled surgery start times, scheduled procedure lengths, surgeon codes, names of surgical services and surgical groups, dates and times when surgeries were booked, OR numbers, actual surgery start times and durations, and assigned staff codes. Note that a particular surgical service, which is also sometimes called a surgical department, could have multiple surgical groups with block assignments. Our data did not contain patient, surgeon or staff identifying information. Only one hospital kept records of cancelations and only for those cases that were canceled on the day of surgery. We obtained block schedules and auto release time information separately because these data are not stored in computerized scheduling records. Table 4.1 summarizes these data. Non-urgent refers to deferrable surgeries that are booked at least two days in advance of the day when they are performed. Non-urgent cases are booked primarily on non-holiday weekdays. All cases include non-urgent cases, cases scheduled on weekends, and those scheduled within 2 days of each surgery day, i.e. urgent and emergent cases. We use a 2-day threshold because the auto-release date is at least 2 days for the vast majority of surgical groups in all hospitals.

Hospital 1 had the largest number of ORs, whereas Hospital 2 had the smallest. Hospital 3 had the most number of surgical services, followed by Hospital 1, and then Hospital 2. Before the auto-release date, the blocked OR time was 60% of OR capacity in Hospital 1, 100% in Hospital 2 and 84% in Hospital 3. Hospital 1 used two shift lengths – 8 hours and 12 hours, whereas the other two hospitals used a single shift length. However, the shift lengths were different in Hospitals 2 and 3. The presence of two shift lengths provides greater flexibility in scheduling cases, but it also complicates

Table 4.1: Basic Data Summary

	Hospital 1		Hospital 2		Hospital 3	
	All	Non-urgent	All	Non-urgent	All	Non-urgent
Working Days	364	257	252	213	530	379
No. of ORs	18		10		14	
Percent of OR time blocked	60%		100%		84%	
Shift lengths	8 or 12 hr		8 hr		10 hr	
Scheduled Surgeries	10,191	7,483	10,866	9,446	12,394	8,875
Cancelations	222	167	N/A		N/A	
Surgical Services	14		3		17	
Surgeons	209	187	106	102	82	82

N/A means data were not available.

Cancellations for Hospital 1 refer to those cases that were cancelled on the surgery day.

the corresponding optimization problem. The three hospitals differed a great deal in the mix of surgeries performed and the volume of each major surgery type (see what follows for details[1]).

Figure 4.1 shows volume and complexity encountered in the three hospitals. We show service volume (as percent of the number of procedures done) and procedure time (mean, and 95% confidence intervals).

The operating characteristics are quite different across the three hospitals. As can be seen in Figure A-1, Hospital 2 performed a limited set of procedure types, specifically ENT, ophthalmology, and orthopedics. Their caseload tended to be high volume, low complexity procedures with short mean procedure times and small variability around the mean. Because of this surgical case mix, Hospital 2 performed more non-urgent surgeries per day (44.35) compared to the other hospitals (29.12 for Hospital 1 and 23.42 for Hospital 3), but still had lower room utilization (56.8%) compared to Hospital

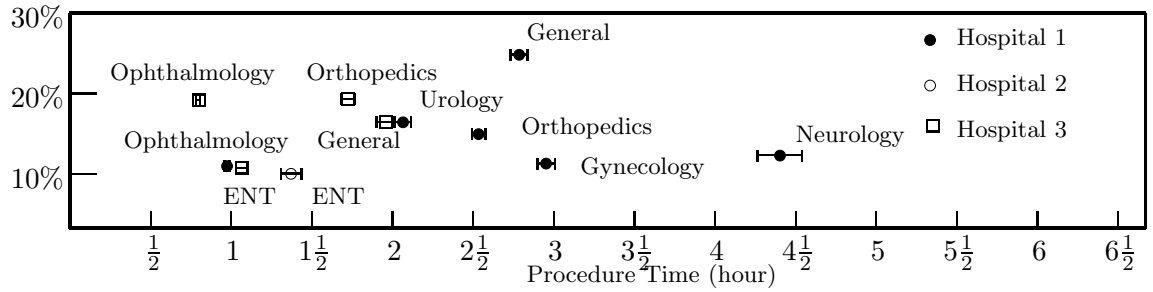
[1] In order to demonstrate this visually, we plot the range of procedure durations (complexity) and volume (percent of total number of procedures) in Figure 4.1. This figure is divided into three parts because two of the three hospitals support surgical services with very small volumes but highly variable surgical procedure durations. The horizontal axis shows mean procedure duration and the 95% confidence interval of scheduled procedure durations of each surgery type. The vertical axis shows the volume of each surgery type in terms of percent of all procedures performed. Not surprisingly, complexity decreases as volume increases.

1. Hospital 2 also had a larger percentage of its surgeons performing multiple surgeries per day (76%), with each of these surgeons performing almost 5 surgeries per day on these days.

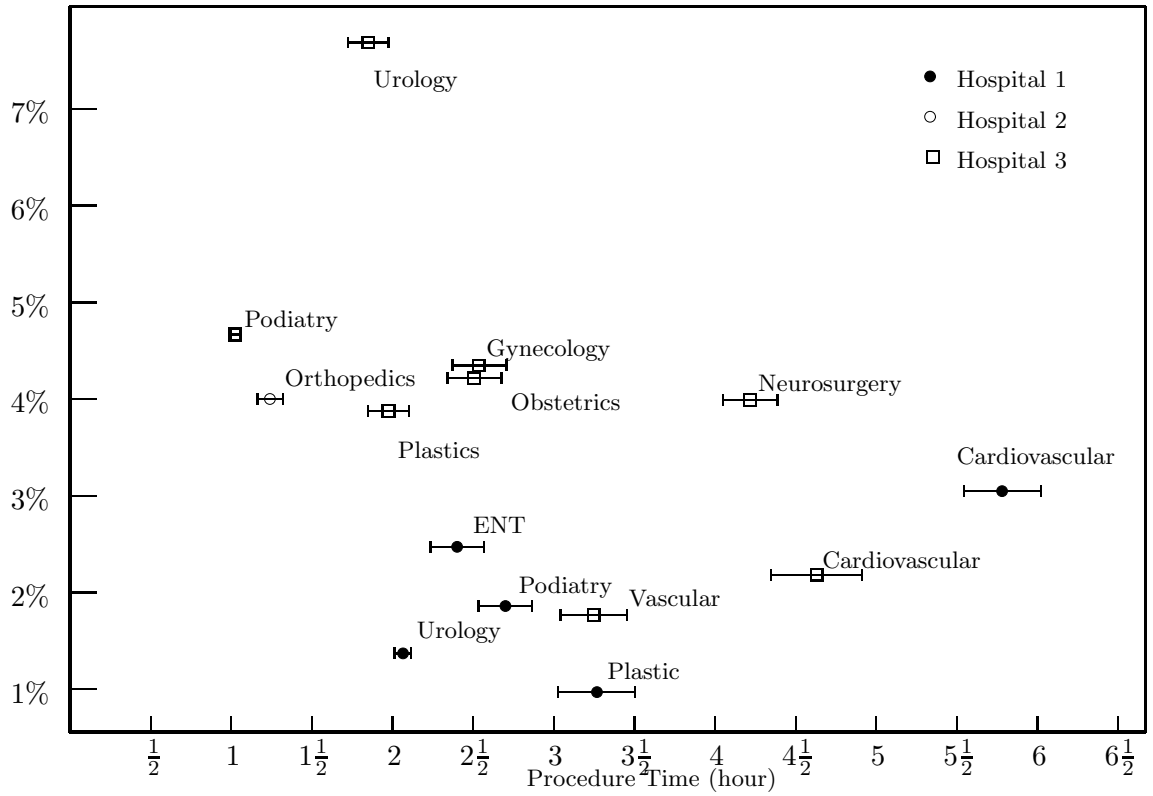
Hospitals 1 and 3 performed a more varied set of procedure types compared to Hospital 2. Hospital 1 provided some surgical procedures types that, although low in volume, have relatively long average procedure times with large variability. Also, for the same surgical procedures types (e.g. cardiology), Hospital 1's average surgical times were longer, suggesting that they may be doing more complex surgical cases relative to Hospital 3. In spite of this likely higher case complexity in Hospital 1, its operating room utilization of 65.7% is higher than that of Hospital 3, at 51.4%.

However, in Hospital 3, 63% of its surgeons performed more than one procedure on the days they operated, compared with 34% in Hospital 1. In addition, for surgeons who performed more than one case per day, the average number of daily cases performed by these surgeons was almost one case more in Hospital 3 compared to Hospital 1 (3.72 cases compared to 2.75 cases on average). Thus, Hospital 3 may face more scheduling challenges trying to accommodate these surgical practice patterns.

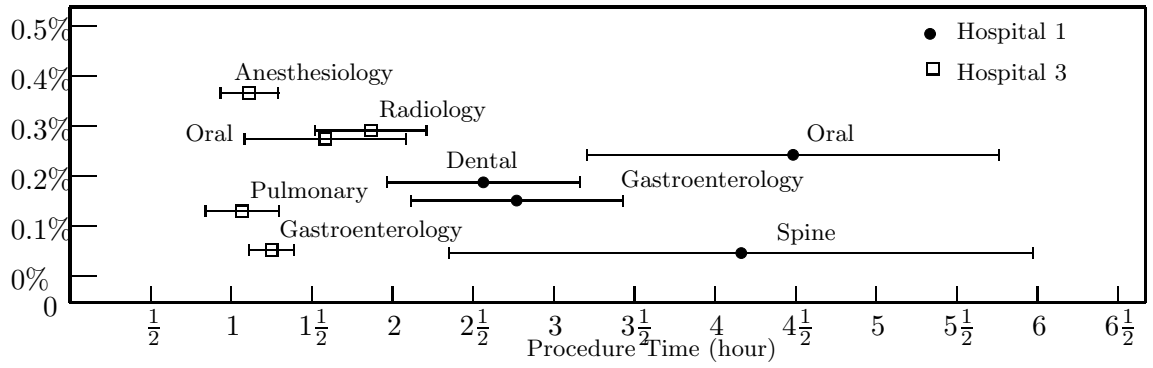
In Table 4.2, we report results of the mixed effects generalized linear regression model described in the last paragraph of Section 4.3. This analysis was performed by using the `xtmixed` procedure in STATA 12.1. Table 4.2 shows that surgeon effect is signifiant in Hospitals 2 and 3. Also, in those hospitals, the actual duration is not significantly affected by the surgery sequence number, after controlling for the surgeon effect. However, in Hospital 1, surgeon effect is not significant and both surgery sequence number and the difference between actual and planned start times are significant. Note, however, that the effect of these factors is not high in practical terms. For example, if actual start time is 60 minutes later than planned, then that will increase actual duration, relative to planned, by approximately 2.94 minutes. Similarly, when performing surgeries that occur later in the sequence, surgeons tend to speed up relative to the planned start time. However, because the number of surgeries performed in each OR on any particular day is quite small, the practical effect of surgery sequence number is quite small. To be more specific, in Hospital 1, the maximum sequence number was 8, with a mean sequence number of 2. Only about 3% of surgeries had a sequence number equal to or greater than 5. For a surgery with sequence number 5, actual duration will be shortened by



(a) Service Volume Between 10 and 20%



(b) Service Volume Between 1 and 10%



(c) Service Volume Between 0 and 1%.

Figure 4.1: Volume versus Complexity

about 9 minutes, relative to the planned duration, on account of the order in which surgeries are performed.

Table 4.2: Result of the Mixed-Effects Multiple Linear Regression Analysis

	Hospital 1			Hospital 2			Hospital 3		
Random Effect:	Standard Deviation			Standard Deviation			Standard Deviation		
Surgeon (Intercept)	2.22			10.96			34.47		
Residual	42.67			31.88			44.60		
Prob $> \bar{\chi}^2$	0.110			$<0.001^*$			$<0.001^*$		
Fixed Effects:	Est.	S.E.	P.V.	Est.	S.E.	P.V.	Est.	S.E.	P.V.
(Intercept)	1.86	1.68	0.268	13.64	1.52	$<0.001^*$	34.38	4.76	$<0.001^*$
Planned Duration	0.957	0.006	$<0.001^*$	0.799	0.006	$<0.001^*$	0.744	0.008	$<0.001^*$
Surgery Sequence in Room	-1.866	0.538	0.001^*	-0.239	0.128	0.062	-0.227	0.308	0.462
Start Time Difference	0.049	0.018	0.006^*	0.010	0.008	0.205	-0.006	0.010	0.561
Delay Reason Indicator	-1.838	1.585	0.246	-0.349	0.774	0.652	1.776	1.069	0.097
Prob $>$ Wald $\bar{\chi}^2$	$<0.001^*$			$<0.001^*$			$<0.001^*$		

Standard Deviation: Standard deviation of the random intercepts.
Est.: Estimate, S.E.: Standard Error of the Estimate, P.V.: p-value of Wald statistic.
$*$: Significant at 0.05 level.

Next, in Table 4.3 we report the current performance statistics of the three hospitals. Note that all three hospitals use some form of manual rescheduling and the results shown in Table 4.3 are obtained after such efforts. Data show that historical utilization was highest in Hospital 1 (60-65% range) and lowest in Hospital 3 (48-52% range). We explain how we calculate utilization as follows.

For Hospitals 2 and 3, we calculate the percent of time that is scheduled surgery time (including clean-up and change over time) within the 8-hour or 10-hour shifts. The daily utilization is the ratio of the total scheduled surgery time within shift to the total available time of shifts used. Hospital 1 has two shifts (8-hour and 12-hour), which makes the calculation of utilization more complicated. Because staff often work overtime and we do not know from the data which OR is assigned 12-hour shift and which OR is assigned 8-hour shift, we employ a heuristic calculation: a shift is treated as a 12-hour shift if the scheduled surgery time during the four extra hours (in excess

of 8 hours) is greater than 2 hours; otherwise it is treated as an 8-hour shift. After determining the shift lengths this way, we calculate the daily utilization as the ratio of scheduled surgery time within shifts and the total available time according to the shift lengths.

On any given day, at least a third of the surgeons perform multiple surgeries. If we count surgeon-days (each surgeon performing cases on a particular day counts as a single surgeon-day), then in the vast majority of surgeon days, surgeons perform multiple cases (87-99% in the third to last row of Table 4.3). Recall that multiple cases give rise to the key difficulty in rescheduling because cases that belong to the same doctor must not overlap[2] . Although many surgeons perform multiple surgeries on their OR day, the number of surgeries performed by a single surgeon are often small. The average number of cases per MD per day lies between 2.76 and 3.76 for non-urgent cases. We utilize this fact in constructing our approach for efficiently solving the rescheduling problem. Data also reveal that the number of surgeons who operate in multiple rooms on any given day ranges from 1 to 8 among the three hospitals.

All three hospitals in our data seem to choose planned surgery times that are good estimates of actual surgery times. Between 68 and 70 percent of surgeries at the three hospitals finish within the allotted time. Among surgeries that take longer than scheduled, the amount of extra time needed is on average between 22 and 41 minutes. We found that hospitals schedule slightly more time on average than the actual duration. This is not surprising because the cost of delays, which result in patient inconvenience, surgeon idleness, and staff overtime, is high.

Because rescheduling changes the sequence in which surgeries are performed, it is important to test whether it will be appropriate to continue to use original planned durations. In order to do so, we fitted each hospital's data to separate generalized linear mixed models. The dependent variable in each model was the actual surgery duration. The independent variables belonged to two groups. The surgeon ID was the random effect (intercept). The fixed effects were the the planned duration, the sequence number of a surgery in its assigned OR, the difference between actual and planned surgery start times, and an indicator that was set to 1 if the difference in

[2] In practice, if a surgeon has a helper, he or she may overlap procedures scheduled in different ORs. However, usually this creates a short overlap (in our data, usually 10 minutes or less), which we ignore in our rescheduling procedure.

Table 4.3: Performance Statistics

	Hospital 1		Hospital 2		Hospital 3	
	All	Non-urgt	All	Non-urgt	All	Non-urgt
AVG Number of ORs used	10.7, 3.2*	11.9, 1.6*	9.0	8.6	8.6	7.9
SD ORs used	2.5 1.5*	1.9 1.1*	1.5	1.3	1.5	1.2
Utilization AVG (%)	61.56	65.24	58.76	54.06	47.69	49.34
Utilization SD (%)	17.54	8.65	16.55	9.14	40.61	29.57
MDs/day AVG	17.29	18.23	10.73	11.04	8.64	8.65
MDs/day SD	9.44	4.10	4.89	2.92	4.53	2.23
MDs With > 1 Case/day AVG	6.06	6.19	7.90	8.35	5.33	5.43
MDs With > 1 Case/day SD	4.04	2.39	4.06	2.52	3.27	1.98
MD-days With > 1 Case (%)	88	98	87	99	93	99
AVG Cases/ MD/day†	2.75	2.76	5.10	4.98	3.76	3.72
SD Cases/MD/day† MD/day†	1.38	1.39	4.15	4.11	2.21	2.26

AVG=Average, SD=Standard Deviation, MD = Surgeon.

* Shifts marked with an asterisk are long shifts, † Only MDs with > 1 case/day were counted.

start times could be attributed to the surgeon (e.g. when the surgeon arrived late) and 0 otherwise. Results of this analysis are shown in Table 4.2. In Hospitals 2 and 3, we found that the surgeon random effect was strong and that after controlling for the surgeon effect, only planned duration had a significant fixed effect. In particular, this implies that actual surgery durations in Hospitals 2 and 3 are not affected by surgery sequence number and early or late start relative to planned start time. Hospital 1 was different in the sense that the surgeon effect was not strong, and planned surgery times,

surgery sequence number, and difference between actual and planned start times were all significant. However, the coefficients of surgery sequence number and difference in start time were small, indicating that their practical impact on actual durations was small. We use this analysis to support our assumption in Section 4.4 that original planned durations will continue to be good estimates of the actual durations after rescheduling.

4.4 Notation and Model Formulation

We model a hospital with multiple ORs staffed by anesthesiologists, nurses and health technicians for either αT (called *short shift*) or T minutes (called *long shift*), where $\alpha \leq 1$. Note that $\alpha = 1$ means that there is only one shift type. In this section, we formulate the rescheduling problem for a particular day, which we call the tagged day. The number of surgeries (jobs) to be scheduled on the tagged day is known. A job j is characterized by its physician index $\mu(j)$, duration d_j and originally scheduled start time s_j^0. For each surgeon indexed i, $J(i)$ denotes the set of jobs that are performed by that surgeon, and $\mathcal{J}$ is the set of all jobs. We make the following assumptions to develop a parsimonious model.

Assumption 1: Time is discrete and 1 minute is a unit of time.

Assumption 2: All ORs are interchangeable and there are no equipment constraints.

Assumption 3: There are enough staff for both long and short shifts and enough ORs to accommodate all procedures.

Assumption 4: The relationship between scheduled and actual surgery durations remains unchanged when surgeries are rescheduled.

Assumption 1 is justified by the fact that scheduled surgery durations are measured in whole minutes. Assumptions 2 – 4 are made for mathematical tractability. We discuss extensions of our model that allow us to relax Assumption 2 in Section 4.8. The availability of ORs in Assumption 3 is typically not an issue in problems of practical interest because there is a feasible solution that schedules all surgeries on the tagged day into available rooms. Assumption 3 is therefore equivalent to the assumption that the OR manager can choose any number of long shifts when rescheduling. We consider practical constraints on the availability of long shifts when describing our branch-and-bound algorithm as follows. Finally, Assumption 4 is consistent with practice and

justified by the analysis presented at the end of Section 4.3.

The Branch & Bound (B&B) Algorithm

We modify the standard branch-and-bound algorithm to account for two shift types. We first obtain an upper bound on the number of long shifts that can be used. A theoretical bound is the number of surgeons who operate on any given day. However, because we already have a feasible solution, a much better practical bound is provided by either (1) the number of shifts that the existing schedule uses, or (2) the maximum number of long shifts available in a particular hospital. For a fixed upper limit, $\bar{n}$, we run the branch-and-bound procedure at most $(\bar{n}+1)$ times: once for each iteration index i, where i goes from 1 to $(\bar{n}+1)$, and in the i-th iteration, we fix the first $(i-1)$ shifts to be long shifts. Furthermore, during execution of the algorithm, if we find that in the k-th run there is a feasible solution in which the first $(k-1)$ long shifts accommodate all work, then we do not need to consider additional iterations. Note that which shift indices are assigned to long shifts and which are assigned to short shifts is not relevant because the B&B algorithm exhausts all possible assignments with $(i-1)$ long shifts in iteration i.

Each iteration finds an optimal assignment with a fixed number of long shifts. Within each iteration, we use backtracking to undo a recently assigned job and place it at the end of the queue of available jobs. Backtracking is performed when (1) there is no feasible assignment of the current job either in existing rooms or in a new room because of same-surgeon overlap, or (2) the sum of the current partial assignment's cost and the lower bound on the cost of assigning the remaining jobs is not strictly smaller than the current best feasible solution. Note that we calculate a lower bound cost for the remaining jobs at each job assignment epoch. In this way, backtracking searches all possible assignments for a fixed number of long shifts within each iteration.

The algorithm terminates at each iteration if either (1) the global lower bound calculated at the beginning of the iteration is achieved by a feasible solution, or (2) all possible assignments are exhausted, or (3) the maximum number of steps is reached. We set the maximum number of steps equal to 100,000. Our implementation of the algorithm required average, standard deviation and maximum run times of (158, 231, 686), (23, 111, 865), and (4, 27, 243) seconds for the three hospitals data when run on a PC with 2.40 GHz processor and 4 GB of RAM. We present the algorithm in a pseudo

code below.

The Branch & Bound (B&B) Algorithm: Pseudo Code

We introduce several additional notation for clarity, then present the algorithm in a pseudo code below.

Table 4.4: Additional Notation

J_A	=	set of currently assigned job
J_U	=	$\mathcal{J} \setminus \mathcal{J}_\mathcal{A}$; currently unassigned job set
$L(J)$	=	lower bound cost for job set J
L_0	=	$L(\mathcal{J})$; global lower bound
$L(J_U)$	=	current partial lower bound
C_c	=	cost of current partial solution
$\tilde{L} = C_c + L(J_U)$	=	current lower bound

For each problem instance, we take all jobs, arrange them in an arbitrary sequence, and then index them starting with 1. Unassigned job set J_U is repeatedly updated during the execution of the algorithm. These updates cause jobs to be shuffled as explained below. When we decide to assign a job, we always take the first job in J_U. When a job is unassigned, it is always placed at the end of J_U, i.e. in the last position.

1 **while** iteration limit is not reached

2 calculate $L_0 = L(\mathcal{J})$. **if** a feasible solution is found such that $C_c = L_0$, terminate.

3 **if** job indexed 1 starts in empty shift indexed 1 for the second time in an iteration, terminate. (all possible solutions are considered)

4 **if** $J_A = \mathcal{J}$ (all jobs are scheduled), update the stored best solution (including job assignments). Let C^* denote the cost associated with the current best solution.

5 calculate partial lower bound $L(J_U)$ and add to the current cost C to acquire $\tilde{L} = C_c + L(J_U)$.

6 **if** $[\tilde{L} > C^*]$, then first **Backtrack** and then go to 3.

7 else Branch and then go to 3.

10 end

Branch: add the first job in J_U to J_A, and construct the compact feasible solution with J_A. We explain what we mean by a compact solution in the sequel.

Backtrack: Un-schedule the last-scheduled job. Place the job at the end of j_U. If this results in emptying the current room, then remove that room from the current solution.

Compactness means that we use the smallest number of rooms to assign all jobs. Specifically, if job i is sequenced in front of job j, then in the assignment, either i and j are in different rooms such that the room index of i is smaller, or, if i and j are in the same room, then i is earlier than j. During the execution of our algorithm, each feasible solution is compact. The algorithm exhausts all possible sequences if no backtracking occurs. We explain this with the help of an example below.

Example: suppose we have 5 jobs, $J_U = \{1, 2, 3, 4, 5\}$ at the beginning, and the first sequence is $1 - 2 - 3 - 4 - 5$. Then, the algorithm would backtrack leading to $J_A = (1 - 2 - 3 - 4)$ and $J_U = (5)$. Next, if it branches, it will obtain the previous solution. Therefore, it will backtrack once more resulting in $J_A = (1 - 2 - 3)$ and $J_U = (5, 4)$. At this point, it is possible to branch and generate a different solution. Specifically, it is possible to get $J_A = (1 - 2 - 3 - 5 - 4)$, $J_U = \emptyset$. Note that these steps exhaust all permutations that follow 1-2-3.

The algorithm maintains this routine and similarly exhausts all permutations that follow 1-2 next, and then all those that follow 1. At this point, the initial sequence would change such that every job could be the first job in the the sequence. This way, all permutations will be exhausted, unless some other stopping criterion is triggered first.

Our goal in rescheduling is to choose a set of new start times, denoted s_j, that reduce the staffing cost. Before rescheduling, we remove all jobs that have $d_j > T$ because it is trivially optimal to assign those surgeries to single long shifts. [3] . That is, in the

[3] Note that d_j is the scheduled surgery duration, which may not equal the realized duration. We use realized duration only to calculate the impact of rescheduling on different performance metrics, but not for rescheduling purposes.

rescheduling problem $d_j < T$ for each $j \in \mathcal{J}$. A key decision variable in our formulation is $y_{j,t}$, which is 1 if job j is rescheduled to start at time t, and 0 otherwise. In particular, if $y_{j,t} = 1$, then $s_j = t$ is the new start time of surgery j. To prevent overlap among surgeries performed by the same surgeon, we introduce binary variables p_{jk}, which equal 1 if jobs j and k are performed by the same surgeon and j is scheduled before k, and 0 otherwise. When $\mu(j) = \mu(k)$, $p_{jk} + p_{kj} = 1$ must hold because either job j is performed before job k, or its opposite occurs. Because each job that is active (being performed) at time t must be scheduled in a separate OR, the minimum number of staffed ORs required at time t equals $h_t = \sum_j \sum_{\tau:\tau \le t \le \tau + d_j} y_{j,\tau}$, the number of active jobs, where the inner sum identifies if a job j is active at time t and the outer sum counts all active jobs. An arbitrary job j is active at time t if it started at time τ and t occurs no later than d_j after τ. Problem parameters and decision variables are summarized in Table 4.5 for convenience.

4.4.1 Model Formulation

With the above notation in hand, we formulate the OR rescheduling problem as the following integer program.

$$z^* = \min \ \alpha n_1 + n_2 \tag{4.1}$$

Subject to:

$$\sum_t t y_{j,t} + d_j p_{jk} - T p_{kj} \ \le \ \sum_t t y_{k,t}, \quad \forall j,k \text{ such that } \mu(j) = \mu(k) \tag{4.2}$$

$$p_{jk} + p_{kj} \ = \ 1, \ \ \forall j,k \text{ such that } \mu(j) = \mu(k) \tag{4.3}$$

$$n_1 + n_2 \ \ge \ h_t, \quad t = 1, \cdots, \alpha T \tag{4.4}$$

$$n_2 \ \ge \ h_t, \quad t = \alpha T + 1, \cdots, T \tag{4.5}$$

$$h_t \ \ge \ \sum_j \sum_{\tau:\tau \le t \le \tau + d_j} y_{j,\tau}, \ t = 1, \ \cdots, T \tag{4.6}$$

$$\sum_t y_{j,t} \ = \ 1, \ \forall j \tag{4.7}$$

$$p_{jk} \ \in \ \{0,1\}, \ \forall j,k \text{ such that } \mu(j) = \mu(k) \tag{4.8}$$

$$y_{j,t} \ \in \ \{0,1\}, \ \forall j,t \tag{4.9}$$

The objective (4.1) minimizes staffing cost, i.e. the total number of staffed ORs after

Table 4.5: Notation Used in Model Formulation

Parameters		
$\alpha T, T$	=	shift lengths, $\alpha \leq 1$
t	=	time index, $t \in \{1, \cdots, T\}$
m	=	number of jobs (surgeries) scheduled on the tagged day
$\mathcal{J}$	=	job index set, $\mathcal{J} = \{1, \cdots, m\}$
d_j	=	scheduled duration of job j
s_j^0	=	originally scheduled start time of job j
$\mu(j)$	=	index of the surgeon who performs job j
$J(i)$	=	set of jobs that are performed by surgeon i
$d_\Sigma(i)$	=	$\sum_{j \in J(i)} d_j$ = the sum of job durations of surgeon i
Decision variables		
n_i	=	number of type-i shifts used after rescheduling, $i = 1, 2$
$y_{j,t}$	=	1 if job j starts at time t, 0 otherwise
s_j	=	new start time of job j
p_{jk}	=	1 if jobs j and k are performed by the same doctor and j is scheduled before k, 0 otherwise
h_t	=	the minimum number of ORs that need to be staffed at time t
$z^*(\mathcal{J})$	=	minimum cost of serving jobs in set $\mathcal{J}$
Other notation		
L	=	lower bound
z	=	cost associated with a feasible solution, $L \leq z^* \leq z$

weighting the shorter staff lengths by a factor α. Constraints (4.2) can be explained as follows. Suppose jobs j and k belong to the same surgeon. Then, either p_{jk} or p_{kj} must equal 1 (from Constraint 4.3). Suppose $p_{jk} = 1$. Then, Constraint (4.2) ensures that $s_j + d_j \leq s_k$ because $s_j = \sum_t t y_{j,t}$ and $s_k = \sum_t t y_{k,t}$. Conversely, if $p_{kj} = 1$, then Constraint (4.2) reduces to $s_j - T \leq s_k$, which is trivially true because $s_j \leq T$ and start times are non-negative. Constraints (4.2) thus enforce a non-overlapping ordering of job start times if they belong to the same surgeon. Constraints (4.6) count the number of active jobs at each time t and Constraints (4.4) and (4.5) ensure that the number of ORs needed is at least equal to the maximum of h_t across all t. Constraints (4.7) are needed to ensure that each job is assigned a start time. Finally, Constraints (4.8) and (4.9) require that p_{jk} and $y_{j,t}$ must be binary variables.

The OR rescheduling problem (4.1) – (4.9) is NP hard because upon ignoring constraints (4.2) and (4.3) and setting $\alpha = 1$, we obtain the well known bin-packing problem. Therefore, we focus in this paper on developing a lower bound with a performance guarantee, which is utilized in a branch-and-bound algorithm.

Lemma 4.4.1. *The problem of rescheduling ORs, as shown in (4.1) – (4.9), is NP hard.*

Sketch of Proof: The proof is straightforward. The statement of the Lemma follows from arguments that reduce our problem to the bin-packing problem, which is known to be NP hard ([86]). This reduction requires that we ignore constraints (4.2) and (4.3) and set $\alpha = 1$. □

The practical difficulty of solving the rescheduling problem (4.1) – (4.9) with a general-purpose software such as CPLEX would depend on the unit of time. A common unit of time used by hospitals is 1 minute, but it would be possible to consider 5 and 10-minute intervals as units of time. In order to gain an understanding of the complexity of the problem formulated above, we solved instances of the OR schedule-improvement problem using CPLEX when time was incremented in units of 1, 5 and 10 minutes.

With 1-minute time increments, the problem formulation had approximately 15 to 20 thousand integer variables (depending on the Hospital), and CPLEX did not solve any instance of the problem for Hospital 1, approximately 60% for Hospital 2, and 100% for Hospital 3 after running overnight. With 5-minute increments, the number of

variables were approximately 3 to 4 thousand, and CPLEX solved all problem instances for Hospitals 2 and 3, and 80% of instances for Hospital 1. With 10-minute increments, the number of variables were approximately 1,500 to 2,000, and CPLEX solved all instances of problems encountered at Hospitals 2 and 3, and 95% of instances at Hospital 1.

Note that for Hospital 1, CPLEX does not solve all instances of the problem even after running overnight with 10-minute increments. For that hospital, the optimality gap (difference between the best solution and a bound) with 1-min-increment instances was 2.5%, and with 5-min-increment instances was 1.5%. Similar statistics for Hospital 2 with 1-min-increment instances was 2.3%. The hospitals in our sample would be considered small to medium-sized hospitals in terms of number of beds and ORs. Therefore, we conclude that general-purpose optimization software are not a reliable means of solving typical OR rescheduling problem.

4.5 One Shift Type

Our approach consists of three steps. In the first step, we develop a classification of surgeon types. We do not differentiate between those surgeons who have block assignments and those who do not. Taking advantage of the surgeon classification, we construct a staffing cost lower bound L in the second step. Finally, in the third step, we develop a procedure for recovering a feasible schedule z from the lower bound construction such that $z \leq (3/2)L$, which immediately leads to the conclusion that the constructed lower bound is at least (2/3) of the optimal solution. Put differently, we use the argument that $\frac{L}{z^*} \geq \frac{L}{z} \geq 2/3$.

4.5.1 Step 1: Surgeon Types

Suppose Y is a set of $q \geq 1$ jobs with indices $\{j_1, \cdots, j_q\}$, then a *chain* of jobs in Y satisfies the property that $s_{j_k} + d_{j_k} = s_{j_{k+1}}$, where s_{j_1} is arbitrary. In other words, any arbitrary connected sequence of jobs is called a chain. Note that Y could be either all jobs of a particular surgeon, or a subset of his or her jobs, and that $s_{j_q} + d_{j_q} < T$.

Definition 4.5.1. *A chain of jobs in Y is called an* O-chain *with respect to shift length T if upon splitting the chain in the middle, i.e. at a point $t = (s_{j_q} + d_{j_q} + s_{j_1})/2$, one*

of the following two properties holds (1) either no job is cut into two pieces, or (2) if a job is cut, then upon taking the job that is cut and assigning it to either the first or the second piece of the chain, both sides of the chain are no longer than $(T/2)$ *in at least one of the two assignments.*

From the above definition, it should be clear that if an O-chain is split at a point that is not the midpoint of the chain, and if the job that is cut (if any) is combined with either one of the two pieces of the chain, then at least one of these two pieces (after combining the cut job) must be no more than $(T/2)$. This is an important property of O-chains that we use later in this paper.

Definition 4.5.2. $P_2||C_{\max}(Y)$ *refers to a two-machine minimum makespan problem ([87]) for job set* Y. *In the minimum makespan problem formulation, there are no overlap avoidance constraints, such as constraints (4.2) and (4.3) in the OR rescheduling problem. The optimization problem can be written as* $C_2(Y) = \min C_{\max}$, *subject to* $\sum_j x_{ij} d_j \leq C_{\max},\ i = 1, 2,\ \sum_i x_{ij} = 1,\ \forall j \in Y,\ x_{ij} \in \{0, 1\},\ \forall i, j$. *The decision variable* x_{ij} *equals 1 if job* j *is assigned to machine* i, *and 0 otherwise.*

We use $P_2||C_{\max}(Y)$ to identify those surgeon types whose jobs can be arranged in an O-chain. Note that $P_2||C_{\max}(Y)$ is also NP hard ([86]). However, in the OR rescheduling context, we find that surgeons who perform multiple surgeries on a particular day perform a relatively small number of surgeries (typically in single digits, see Table 4.3 in Section 4.3) and pseudo-polynomial algorithms exist for solving such problems (for example, via dynamic programming algorithm for an equivalent knapsack problem with size $(d_\Sigma/2)$ – see [88]). Therefore, in the intended application of our approach, the $P_2||C_{\max}$ problem that arises is easy to solve.

Definition 4.5.3. *Consider an arbitrary surgeon indexed* i *with job set* $J(i)$. *This surgeon is referred to as an A-type if and only if* $C_2(J(i)) > T/2$. *Similarly, a surgeon is O-type if and only if* $C_2(J(i)) \leq T/2$.

Clearly, a surgeon may be either A-type or O-type, but not both. The importance of this surgeon classification is that if a surgeon is A-type, then there does not exist an O-chain of his or her jobs. Conversely if a surgeon is O-type, then there must exist at least one O-chain of his or her jobs. We prove this preliminary result in Lemma 4.5.4,

but before doing so, we summarize the additional notation used in this Section in Table 4.6. In this table, we introduce notation S_k to denote the index set of k-type surgeons and n_k to denote the number of k-type surgeons, where $k \in \{A, O\}$. A proof of Lemma 4.5.4 is provided as follows.

Proof of Lemma 4.5.4: We prove each statement separately. Suppose the surgeon is A-type. This means $C_2(J(i)) > T/2$. If we find an O-chain of $J(i)$, then that implies we can divide jobs into two parts such that both parts are at most $(T/2)$ in length. This is a contradiction because then $C_2(J(i)) > T/2$ cannot be true. That is, when $C_2(J(i)) > T/2$, it is not possible to find an O-chain of surgeon i's jobs.

Next, suppose the surgeon is O-type. Solve $P_2||C_{\max}(J(i))$ to obtain minimum makespan assignments to two machines such that each assignment is no more than $(T/2)$. Take the jobs assigned to each machine and organize them into an arbitrary connected sequence (i.e. a chain). Because each chain is obtained from the solution to the $P_2||C_{\max}(J(i))$, either the two chains are of equal length or differ by at most the duration of one job. Therefore, if we combine the two chains to form a chain of all jobs in $J(i)$ and then split it in the middle, either no job will be split (which satisfies Property 1 of O chains), or if a job is split, it will belong to the chain of either machine 1 or machine 2. Then, by keeping the split job in the chain to which it was originally assigned by $P_2||C_{\max}(J(i))$, we satisfy Property 2 of O-chains. This completes the proof. □

Table 4.6: Additional Notation

Chain	=	a connected sequence jobs
$P_2\|\|C_{\max}(J)$	=	the two-machine makespan-minimization problem with job set J
$C_2(J)$	=	the optimal value of $P_2\|\|C_{\max}(J)$
A-type surgeon	=	surgeon i whose jobs satisfy the property: $C_2(J(i)) > T/2$
O-type surgeon	=	surgeon i whose jobs satisfy the property: $C_2(J(i)) \leq T/2$
S_k	=	the index set of k-type surgeons, where $k \in \{A, O\}$
n_k	=	number of k-type surgeons, where $k \in \{A, O\}$

Lemma 4.5.4. *Given a surgeon i with job index set $J(i)$, the following statements are true.*

1. *If surgeon i is A-type, then there does not exist an O-chain of jobs in $J(i)$.*

2. *If surgeon i is O-type, then there exists at least one O-chain of jobs in $J(i)$.*

4.5.2 Step 2: Lower Bound Construction

A key step in the construction of lower bound involves arranging surgeons' jobs in a chain and filling them in available empty spaces of previously activated operating rooms in a fluid fashion. We refer to this step as *fluid filling*. Essentially, this means that we use all open time in a staffed room before choosing to staff more rooms and do not worry about the fact that this procedure may cause a particular surgeon's chain to be *split*, i.e. placed in more than one room. Splitting may cause a conflict, which means at least one job is placed in multiple rooms and/or some same-surgeon jobs overlap. We focus in Section 4.5.3 on eliminating all splits, and thus eliminating all conflicts. In the LB construction algorithm, shown in a graphical form in Figure 4.2, we ignore splits.

LB Algorithm

Step 1: Arrange A-type surgeons' jobs in arbitrary chains and place them in separate rooms, using n_A rooms. Each room may have some unused time. The remaining surgeons are all O-type surgeons.

Step 2: Arrange O-type surgeons' jobs in arbitrary O-chains and assign these chains one at a time to available rooms in a fluid fashion. Use extra rooms as needed if spaces left in n_A rooms are not enough to fit jobs of all O-type surgeons.

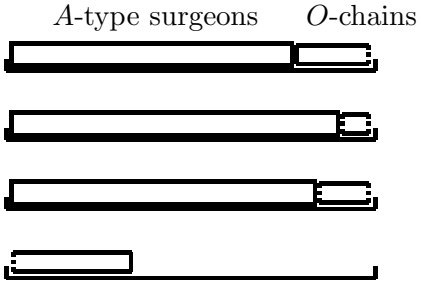


Figure 4.2: LB Construction Example (splits are shown by dotted lines)

A count of the number of shifts needed is $L := n_A + \left\lceil \frac{\sum_{j \notin S_A} d_j - (n_A T - \sum_{j \in S_A} d_j)}{T} \right\rceil^+$, where the notation $\lceil \cdot \rceil$ denotes the integer ceiling of its argument. In order to prove that L is a valid lower bound, we first prove that at least one job of an A-type surgeon must cross $(T/2)$ in any feasible assignment of his or her jobs. This is a crucial step because it immediately implies that each A-type surgeon requires at least one room.

Lemma 4.5.5. *If a surgeon is A-type, then in any feasible solution, one of the surgeon's jobs crosses* $(T/2)$*. That is, there must exist one job* j_k *such that* $s_{j_k} < T/2 < s_{j_k} + d_{j_k}$.

Proof: We prove the result by contradiction. Suppose there is no s_{j_k} such that $s_{j_k} < T/2 < s_{j_k} + d_{j_k}$. Then, $(T/2)$ divides the surgeon's jobs into two non-overlapping parts. Each of these parts need not be scheduled in a single room. In one part, each job starts and ends before $(T/2)$ and in the other part, each job starts and ends after $(T/2)$. This implies that $C_2(J(i)) \leq T/2$ and contradicts the definition of A-type surgeons. Hence proved. □

L is a valid lower bound because at least n_A rooms are needed for A-type surgeons and O-type surgeons' jobs are assigned in a fluid manner. Lemma 4.5.6 presents this result.

Lemma 4.5.6. $L = n_A + \left\lceil \frac{\sum_{j \notin S_A} d_j - (n_A T - \sum_{j \in S_A} d_j)}{T} \right\rceil^+$ *is a valid lower bound.*

Proof: Lemma 4.5.5 shows that the number of rooms used cannot be smaller than n_A. Therefore, the total residual capacity after accommodating A-type surgeons cannot be smaller than $(n_A T - \sum_{j \in S_A} d_j)$. Then, in the best case the capacity $(n_A T - \sum_{j \in S_A} d_j)$ will be completely filled by O-type surgeons' jobs, and the additional rooms needed cannot be greater than $\left\lceil \frac{\sum_{j \notin S_A} d_j - (n_A T - \sum_{j \in S_A} d_j)}{T} \right\rceil^+$. □

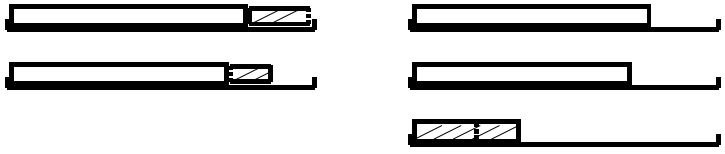
4.5.3 Step 3: Feasible Solution Construction

Next, we obtain a feasible solution from L that uses no more than $(1/2)L$ more ORs. Our main result is presented in Theorem 4.5.7 below.

Theorem 4.5.7. *L is a $(2/3)$-lower bound. Specifically, there exists a feasible solution z such that $L \geq (2/3)z \geq (2/3)z^*$ for every instance of the OR rescheduling problem.*

Proof: The LB algorithm causes at most $(L-1)$ O-type surgeons' chains to be split (see Figure 4.2 for an example). We show next that splits can be removed by considering the following three cases. In these arguments, μ denotes an arbitrary O-type surgeon whose O-chain is split by the fluid-filling routine.

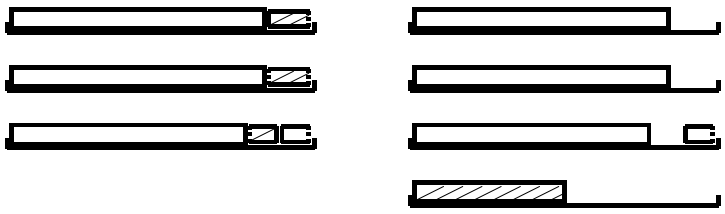
1. $|J(\mu)| = 1$, i.e. surgeon μ has only one job, labeled k. Because $C_2(J(\mu)) \leq T/2$, it follows that job k can be scheduled in a new room and it will occupy no more than $(T/2)$ of that room's time. That is, we can eliminate the assignment conflict of one room by adding at most $(1/2)$ more room. This is shown graphically in Figure 4.3.



(a) LB Assignment (b) Feasible Solution

Figure 4.3: Feasible Solution Construction when $|J(\mu)| = 1$

2. $|J(\mu)| > 1$ and surgeon μ's O-chain is split at least twice (i.e. occupies time in at least three different ORs). In this case, we take all jobs in $J(\mu)$ and schedule them in a new room. This resolves potential scheduling conflict of at least two rooms, each of which would have contained pieces of O-chain of the same surgeon. Thus, for each room whose assignment conflict is resolved, this step adds at most $(1/2)$ extra room. An example showing the LB and feasible solution construction when a surgeon's O-chain is split twice is shown in Figure 4.4.

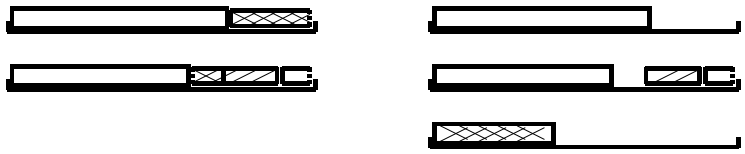


(a) LB Assignment (b) Feasible Solution

Figure 4.4: Feasible Solution Construction When Surgeon μ's Chain is Split Twice

3. $|J(\mu)| > 1$ and surgeon μ's O-chain is split only once. If the split does not cause a job to be cut, then conflict may arise because surgeon μ's jobs may overlap. Such conflict can be avoided relatively easily by scheduling the two pieces of surgeon μ's jobs at opposite ends of the two rooms. Next, we consider the case in which splitting causes a job to be cut.

   Using Definition 4.5.1 and the discussion that follows this definition, we can argue that upon taking the split job and combining it with one of the two pieces of the O-chain, at least one piece must be no more than $(T/2)$. We remove the piece that is less than $(T/2)$ and assign it to a new room, utilizing at most $(1/2)$ extra room to resolve the conflict – see example in Figure 4.5. Moreover, we schedule this surgeon's jobs at the two ends of the ORs that contain his or her jobs to avoid overlap.



(a) LB Assignment: Double hatched part is $\leq T/2$

(b) Feasible Solution Uses $\leq 1/2$ Extra Room

Figure 4.5: When Surgeon μ's Chain is Split Exactly Once

In all cases discussed above, the task of turning the surgery schedule of a room into a feasible schedule adds at most half extra room. That is, we require at most $\lceil \frac{1}{2}(L-1) \rceil$ additional room to obtain a feasible assignment, which establishes our claim. □

The above procedure gives us a (3/2)-approximation algorithm.We use this approach to generate the initial feasible solution in our implementation of the branch-and-bound algorithm.

4.6 Two Shift Types

The proof of the lower bound's performance guarantee requires three cases to be considered separately: (1) $\alpha \leq 1/2$, (2) $1/2 < \alpha < 2/3$, (3) $2/3 \leq \alpha < 1$. In each case, we define surgeon types, then develop a LB construction algorithm, and finally an approach to convert the LB into a feasible solution that is at most 3/2 of the lower bound.

4.6.1 Step 1: Surgeon Types When $1/2 < \alpha < 2/3$

We start by defining surgeon types in Definition 4.6.1. A summary of the defining characteristics of surgeon types is presented in Table 4.7.

Definition 4.6.1. **C-type:** *A surgeon i is called C-type if and only if $C_2(J(i)) > \alpha T$. Clearly, for C-type surgeons, $d_{\sum}(i) > \alpha T$.*

B_1-type: *A surgeon i is called B_1-type if and only if (1) $d_{\sum}(i) > \alpha T$ and (2) $\frac{1}{2}T < C_2(J(i)) \leq \alpha T$. Furthermore, a B_1-type surgeon is said to belong to Group-1 if $d_{\sum}(i) - C_2(J(i)) > (\frac{1-\alpha}{2})T$, and to Group 2 otherwise. That is, for B_1-type surgeons who are in Group 2, $d_{\sum}(i) - C_2(J(i)) \leq (\frac{1-\alpha}{2})T$;*

B_2-type: *A surgeon i is called B_2-type if and only if (1) $d_{\sum}(i) \leq \alpha T$ and (2) $\frac{1}{2}T < C_2(J(i)) \leq \alpha T$.*

A_1-type: *A surgeon i is called A_1-type if and only if (1) $d_{\sum}(i) > \alpha T$ and (2) $\frac{1}{2}\alpha T < C_2(J(i)) \leq \frac{1}{2}T$.*

A_2-type: *A surgeon i is called A_2-type if and only if (1) $d_{\sum}(i) \leq \alpha T$ and (2) $\frac{1}{2}\alpha T < C_2(J(i)) \leq \frac{1}{2}T$.*

$O(\alpha)$-type: *A surgeon i is called $O(\alpha)$-type if and only if $C_2(J(i)) \leq \frac{1}{2}\alpha T$. In this case, $d_{\sum}(i) \leq \alpha T$ must be true as well.*

Table 4.7: Surgeon Types

Duration Sum ($d_{\sum}$)	Makespan ($C_2(J(i))$)			
	$(0, \frac{1}{2}\alpha T]$	$(\frac{1}{2}\alpha T, \frac{1}{2}T]$	$(\frac{1}{2}T, \alpha T]$	$(\alpha T, T]$
$(0, \alpha T]$	$O(\alpha)$	A_2	B_2	N/A
$(\alpha T, T]$	N/A	A_1	B_1	C

The above classification is a *partition*, i.e. each surgeon must belong to exactly one type and the types are exhaustive. Recall from Table 4.6 that S_x and n_x denote, respectively, the subset and number of x-type surgeons, where now $x \in \{A_1, A_2, B_1, B_2, C\}$. We also use g_i to denote the number of B_1-type surgeons that belong to Group-i.

Analogous to Lemma 4.5.4, we list properties of each surgeon type in Lemma 4.6.2.

Proof of Lemma 4.6.2: We prove the three statements in the Lemma one by one.

1. Suppose there exists a feasible solution in which surgeon i is either C, or B_1, or B_2-type, and none of his or her jobs cross $t_1 = \frac{1}{2}T$. Then $t_1 = \frac{1}{2}T$ divides surgeon-i's jobs into two parts – the first part consists of jobs assigned before t_1 and the second part of jobs after t_1. Neither part is more than $\frac{1}{2}T$ and surgeon-i's jobs can be arranged in an O-chain. But this indicates that we have a feasible solution to $P_2||C_{\max}(J(i))$ with makespan no more than $\frac{1}{2}T$, which contradicts the definition of a C, or B_1, or B_2-type surgeon.

Similarly, if surgeon-i's jobs do not cross $t_2 = \alpha T$, then t_2 also divides his or her jobs into two parts such that neither part is more than αT. This also contradicts the definition of a C, or B_1, or B_2-type surgeon.

If there exists an O-chain or an $O(\alpha)$-chain of surgeon-i's jobs, then that means we can divide that surgeon's jobs into two parts and each part is smaller than either $\frac{1}{2}T$ or $\frac{1}{2}\alpha T$. This indicates that we have a feasible solution to $P_2||C_{\max}(J(i))$ with a makespan no more than either $\frac{1}{2}T$ or $\frac{1}{2}\alpha T$, which is once again a contradiction.

2. The argument in this case is identical to the argument we presented for O-type surgeons in Lemma 4.5.4. We do not repeat the argument here for sake of brevity.

3. The argument for the existence of an $O(\alpha)$-chain can be obtained by replacing T by αT in the proof of Lemma 4.5.4. This completes the proof. □

Lemma 4.6.2. *Given surgeon i with job index set $J(i)$, the following statements are true.*

1. *If the surgeon is either C-type, or B_1-type, or B_2-type, then in any feasible assignment, jobs of this surgeon must cross $t_1 = \frac{1}{2}T$ and $t_2 = \alpha T$. There does not exist an O or $O(\alpha)$-chain of $J(i)$.*

2. *If the surgeon is either A_1 or A_2-type, then there exists an O-chain of $J(i)$.*

3. *If the surgeon is $O(\alpha)$-type, then there exists an $O(\alpha)$ and also an O-chain of $J(i)$.*

Immediate consequences of Lemma 4.6.2 are as follows: (1) in any feasible assignment of jobs to rooms, the number of long shifts used cannot be smaller than n_C; and (2) in addition to n_C, every feasible assignment must use at least $(n_{B_1} + n_{B_2})$ short shifts.

We use these properties in Section 4.6.3 to prove asymptotic performance of our lower bound. But first, we show how to construct the lower bound in Section 4.6.2.

4.6.2 Step 2: Lower Bound Construction When $1/2 < \alpha < 2/3$

The construction of the lower bound in this case is more complicated than in Section 4.5.2 because we can not argue that we need long shifts to accommodate B_1-type surgeons. This is best illustrated with a simple example. Suppose all B_1 surgeons' portfolios consist of two jobs: one of duration αT, and the other of duration $\epsilon = 1$ minute. Then, for all problems of practical interest (specifically, when $n_{B_1} \ll T$), we can accommodate B_1 surgeons in n_{B_1} rooms with short shifts and one room with a long shift. We may not need to introduce new long rooms because the ϵ-duration jobs may fit into the leftover spaces in n_C shifts. Therefore, we can only argue that we need at least n_{B_1} rooms with short shifts. We describe our LB construction procedure next. A graphical representation of this algorithm can be found in Figure 4.6.

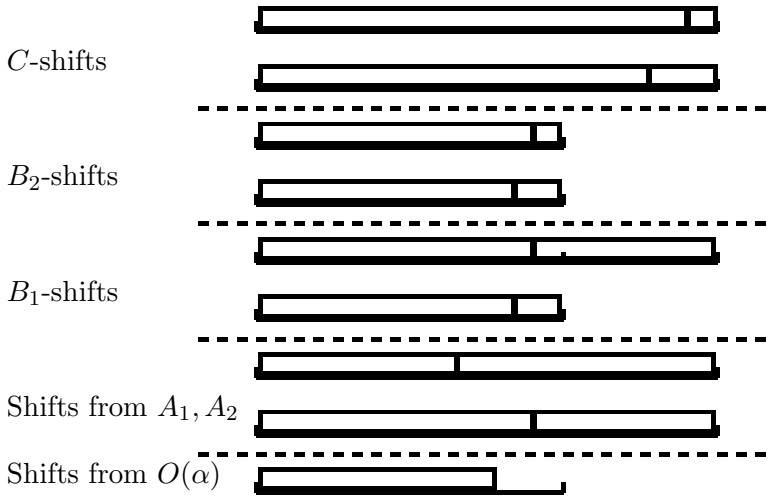


Figure 4.6: LB Construction Algorithm

LB Algorithm

Step 1: Arrange each C-type surgeon's jobs in an arbitrary chain and place the chain in a separate room with long shift. Each of the n_C rooms with shift length T may have unused time.

Step 2: Arrange each B_2-type surgeon's jobs in arbitrary chains and place the chain in a separate room with short shift. Each of the n_{B_2} rooms with shift length αT may have unused time.

Step 3: Arrange B_1-type surgeons such that those in Group 1 are assigned first. Upon solving $P_2||C_{\max}(J(i))$ for the ith B_1-type surgeon, the optimal solution splits $J(i)$ into two chains consisting of whole jobs such that the longer chain is at least $(1/2)T$, but no more than αT, and the shorter chain is $[d_{\sum}(i) - C_2(J(i))]$ in length. Place the longer chains into separate short shifts, starting with Group-1 surgeons first.

Step 4: From the first unfilled shift in the above steps, use fluid filling to place the short chains (also called pieces) of B_1-type surgeons, starting with Group-1 surgeons first. If the shifts introduced in Steps 1, 2 and 3 are all filled up, expand the shifts introduced in Step 3 from short to long shifts to fill the rest of the second pieces. Note that we do not need to introduce new shifts because at most n_{B_1} long shifts are needed to accommodate all B_1-type surgeons.

Step 5: From the first unfilled shift in the above steps, use fluid filling to place the A_1, A_2 and $O(\alpha)$-type surgeons in this sequence. Arrange each A_1 and A_2-type surgeon's jobs into an O-chain and each $O(\alpha)$-type surgeon's jobs into an $O(\alpha)$-chain before fluid filling. Do not alter the length of a previously activated shift. If the unused time of existing shifts is not enough, introduce new long shifts until all A_1 and A_2-type surgeons are placed. Thereafter introduce new short shifts until all $O(\alpha)$-type surgeons are placed.

Step 6: If no new shift is introduced in Step 5, then

$$n'_{B_1} = \lceil \frac{\sum_{i \in S_C \cup S_{B_2} \cup S_{B_1}} d_{\sum}(i) - (n_C + \alpha n_{B_2} + \alpha n_{B_1})T}{(1-\alpha)T} \rceil$$

B_1-type shifts are extended. In this case, report

$$L = n_C + \alpha n_{B_2} + \alpha n_{B_1} + n'_{B_1}(1-\alpha) \tag{4.10}$$

as the lower bound. Otherwise, report

$$L = \sum d_{\sum}(i)/T \tag{4.11}$$

as the lower bound.

We need two separate constructs in Step 6 because if Step 5 does not introduce new shifts, then in addition to $n_C + \alpha n_{B_2} + \alpha n_{B_1}$, the lower bound must extend n'_{B_1} short shifts, giving rise to Equation (4.10). In contrast, if we introduce new shifts in Step 5, then the weighted number of shifts used is not necessarily a lower bound because it may be possible to use fewer shifts by choosing a different combination of short and long shifts. The latter depends on the value of α. Because we need a bound that works for all $\alpha \in (1/2, 2/3)$, we use (4.11), the number of rooms needed when all jobs are filled in a fluid fashion as a lower bound. These arguments also help establish that L is a valid lower bound.

Lemma 4.6.3. *The amount L obtained from the LB Algorithm is a valid lower bound for any instance of the OR rescheduling problem.*

4.6.3 Step 3: Feasible Solution Construction When $1/2 < \alpha < 2/3$

In this section, we describe a method for constructing a feasible solution and show that the ratio of the lower bound and the feasible solution costs is asymptotically at least (2/3) in Theorem 4.6.6. For brevity, we refer to x-type surgeons as x surgeons and to shifts that were introduced in LB Algorithm to accommodate x surgeons as x-shifts, where $x \in \{C, B_1, B_2, A_1, A_2, O(\alpha)\}$. The most complicated part in our algorithm is the treatment of B_1 surgeons. Therefore, we present a preliminary result first to facilitate the proof of Theorem 4.6.6.

Definition 4.6.4. *In fluid filling procedure of B_1 surgeons' second pieces, let $r(i)$ denote the index of the first unextended B_1-shift in which the second piece of i-th surgeon begins to fill. We define $r(i) = 0$ if the second piece of the i-th B_1 surgeon begins to fill in either C-shift or B_2-shift, and $r(i) = \infty$ if the second piece begins to fill in the extended part of a B_1-shift.*

Lemma 4.6.5. *If there exists a B_1 surgeon (suppose the $(\hat{i}+1)$-th B_1 surgeon; $\hat{i}$ can be 0) such that $r(\hat{i}+1) \geq \hat{i}+1$, then for any $i > \hat{i}$, we must also have $r(i) \geq i$. Furthermore,*

the number of extended B_1 shifts is at least

$$\left\lceil \frac{\sum_{i=n_c+n_{B_2}+\hat{i}+1}^{n_c+n_{B_2}+n_{B_1}} d_{\sum}(i) - (n_{B_1} - \hat{i})\alpha T}{(1-\alpha)T} \right\rceil .$$

Proof: To avoid dealing with trivial cases, we focus on situations in which $r(i)$ is finite. We prove the first claim in Lemma 4.6.5 by contradiction. Suppose there exists $i > \hat{i}$ such that $r(i) < i$. Then, from the $(\hat{i}+1)$-th to the i-th surgeon, the second pieces of those surgeons are all filled in the left-over spaces of the unextended B_1-shifts $r(\hat{i}+1)$ to $r(i)$. Let the number of surgeons from the $(\hat{i}+1)$-th to the i-th be n'. That means we fill all of these n' B_1 surgeons' jobs into at most n' short shifts. This is impossible because $d_{\sum}(i) > \alpha T$ for each B_1 surgeon.

Second pieces of surgeons whose indices range from $(n_c + n_{B_2} + \hat{i} + 1)$ to $(n_c + n_{B_2} + n_{B_1})$ are filled into unextended B_1 shift starting from shift indexed $r(\hat{i}+1)$. The total amount of work of these surgeons is $\sum_{i=n_c+n_{B_2}+\hat{i}+1}^{n_c+n_{B_2}+n_{B_1}} d_{\sum}(i)$. The empty space in remaining unextended B_1 shifts is no more than $(n_{B_1} - \hat{i})\alpha T$ because $r(\hat{i}+1) \geq \hat{i}+1$ and each unextended shift is αT in length. Therefore, the amount of work that needs to be placed in extended B_1 shifts is at least $(\sum_{i=n_c+n_{B_2}+\hat{i}+1}^{n_c+n_{B_2}+n_{B_1}} d_{\sum}(i) - (n_{B_1} - \hat{i})\alpha T)$. Finally, because each extended B_1 shift adds $(1-\alpha)T$ capacity, these arguments help establish the second claim. □

We are now ready to prove our main result of this Section, presented in Theorem 4.6.6.

Theorem 4.6.6. *L is an asymptotic $\frac{2}{3}$ lower bound.*

Proof: The LB algorithm places B_2 and C surgeons' jobs in separate rooms. These assignments are feasible. However, the assignment of remaining surgeons' jobs may result in conflicts. In what follows we consider different surgeon types in the order in which their jobs are assigned to rooms by the LB algorithm. In each case, we find the ratio of the costs incurred in the LB solution and the feasible solution. But before doing so, we summarize our results in Table 4.8. Similar to Theorem 4.5.7, our approach consists of eliminating assignments that result in splits, because splits may lead to conflicts.

Table 4.8: Lower Bound (LB) and Feasible Solution (F) Costs

Surgeon Type	Shifts Involved (Full or Partial)	LB & Feasible Solution Cost	Relationship
C & B_2	Shifts that do not split any surgeon	LB_0 & F_0	$LB_0 = F_0 \geq (2/3)F_0$
Group-1 B_1	(1) C & B_2-shifts, (2) unextended first g_1 B_1-shifts, (3) extended parts of B_1-shifts	LB_{g_1} & F_{g_1}	$LB_{g_1} \geq (2/3)F_{g_1}$
Group-2 B_1	(1) C & B_2-shifts, (2) unextended last g_2 B_1-shifts	LB_{g_2} & F_{g_2}	$LB_{g_2} \geq (2/3)(F_{g_2} - 1)$
A_1	(1) C & B_2-shifts, (2) B_1-shifts & A_1-shifts	LB_{A_1} & F_{A_1}	$LB_{A_1} \geq (2/3)(F_{A_1} - 2)$
A_2	(1) C & B_2-shifts, (2) B_1-shifts, A_1, & A_2-shifts	LB_{A_2} & F_{A_2}	$LB_{A_2} \geq (2/3)F_{A_2}$
$O(\alpha)$	(1) C & B_2-shifts, (2) B_1-shifts, A_1, A_2, & $O(\alpha)$-shifts	$LB_{O(\alpha)}$ & $F_{O(\alpha)}$	$LB_{O(\alpha)} \geq (2/3)F_{O(\alpha)}$
All	All Shifts	LB & F	$LB \geq (2/3)(F - 4)$*

*a shared -1 is included for the case when the last extended B_1 shift splits an A_1, A_2, or $O(\alpha)$ surgeon.

[B_1 surgeons:] Using the properties of Group-1 and Group-2 surgeons, we propose the following procedure for recovering a feasible solution for B_1 surgeons whenever their second pieces are split by the LB algorithm.

For each Group-1 surgeon, extend the shift introduced by his or her first piece (if it is not extended already) and place both the first and the second pieces in this shift. This means that for every Group-1 surgeon, the extra cost is at most $(1-\alpha)T$.

For every two Group-2 surgeons, extend one of their short shifts (if neither is extended already) and place the second pieces of two surgeons in the extended part. Because for Group-2 surgeons, $d_{\sum}(i) - C_2(J(i)) \leq (\frac{1-\alpha}{2})T$, we increase cost by at most $(1-\alpha)T$ for every two such surgeons.

First, we consider all possibilities regarding where the second pieces of B_1 surgeons may be placed. According to the LB algorithm, they may be filled sequentially in C shifts, B_2 shifts, unextended B_1 shifts and extended B_1 shifts. It is clear that if all B_1 shifts are extended, the space would be enough for all B_1 surgeons' work, so they do not need additional shifts. By Definition 4.6.4, we know that the second pieces of the first to the $\hat{i}$-th B_1 surgeons fill in C, B_2 and first to the $\hat{i}$-th B_1 shifts. We consider two cases. Case (1): $\hat{i} \leq g_1$, and Case (2): $\hat{i} > g_1$. Recall that g_i denote the number of Group-i B_1 surgeons, $i = 1, 2$.

Case (1): $\hat{i} \leq g_1$. Divide g_1 into four groups: k_0 surgeons whose second pieces are filled in C shifts but not split; k_1 surgeons whose second pieces are split by C-shifts; the next $k_2 = \hat{i} - k_1 - k_0$ surgeons whose second pieces are filled in either B_2 or unextended B_1-shifts; and the remaining $k_3 = (g_1 - \hat{i})$ surgeons.

The assignment of jobs of the k_0 surgeons in the LB algorithm is feasible: their first pieces are placed between 0 and αT, and their second pieces are placed between αT and T with no split. Therefore, we do not change the assignment in the feasible solution construction.

Next, consider k_1 surgeons whose second pieces are split by a C-shift. Because each C-shift can only split a surgeon once, at least k_1 C-shifts are considered here. In the feasible solution construction, we place each such surgeon into his or her own shift, extending it into long shift if it has not been extended. So the additional capacity

needed is at most $(1-\alpha)T$. The ratio of the lower bound to the feasible solution cost is therefore no less than

$$\frac{k_1 T}{(k_1 + k_1(1-\alpha))T} = \frac{1}{2-\alpha} \geq \frac{2}{3},\ \forall\ \alpha \in (\frac{1}{2}, \frac{2}{3}). \tag{4.12}$$

Focusing next on the k_2 surgeons whose second pieces are filled in either B_2 or B_1-shifts., we find that each such second piece is at least $((1-\alpha)/2)T$ in duration and the amount of open space in B_2 or B_1-shifts is at most $(\alpha - 1/2)T$. Therefore, for k_2 surgeons, we need at least $\lceil \frac{k_2(1-\alpha)}{2(\alpha-1/2)} \rceil$ shifts. Note that here all the B_2 shifts and the unextended B_1 shifts indexed from 1 to $\hat{i}$ are included. If this is not true, then the $(\hat{i}+1)$-th B_1 surgeon's second piece will fill in a shift indexed before $(\hat{i}+1)$-th B_1 shift, violating the definition of $\hat{i}$. In the LB, the cost of placing these surgeons' jobs is at least $(\lceil \frac{k_2(1-\alpha)}{2(\alpha-1/2)} \rceil \alpha T$. Upon converting to a feasible solution, the total cost increases by at most $(1-\alpha)k_2 T$. So, the ratio of LB to feasible solution costs is

$$\begin{aligned}
&\frac{\lceil \frac{k_2(1-\alpha)}{2(\alpha-1/2)} \rceil \alpha T}{\lceil \frac{k_2(1-\alpha)}{2(\alpha-1/2)} \rceil \alpha T + k_2(1-\alpha)T} \geq \frac{\frac{k_2(1-\alpha)}{2(\alpha-1/2)} \alpha T}{\frac{k_2(1-\alpha)}{2(\alpha-1/2)} \alpha T + k_2(1-\alpha)T} \\
&= \frac{(1-\alpha)\alpha}{(1-\alpha)\alpha + (1-\alpha)(2\alpha-1)} = \frac{\alpha}{3\alpha - 1} \geq \frac{2}{3},\quad \forall\ \alpha \in (\frac{1}{2}, \frac{2}{3})
\end{aligned} \tag{4.13}$$

At this point, k_3 Group-1 surgeons remain. According to Lemma 4.6.5, the presence of these surgeons requires that we extend at least $\lceil \frac{\sum_{i=n_c+n_{B_2}+\hat{i}+1}^{n_c+n_{B_2}+g_1} d_{\sum}(i) - k_3 \alpha T}{(1-\alpha)T} \rceil$ B_1-shifts. Note that we do not count those jobs of Group-2 surgeons that need to be placed in extended shifts when calculating the lower bound. That is, in the lower bound, the cost associated with k_3 Group-1 surgeons is at least

$$\lceil \frac{\sum_{i=n_c+n_{B_2}+\hat{i}+1}^{n_c+n_{B_2}+g_1} d_{\sum}(i) - k_3 \alpha T}{(1-\alpha)T} \rceil (1-\alpha)T + k_3 \alpha T \geq \sum_{i=n_c+n_{B_2}+\hat{i}+1}^{n_c+n_{B_2}+g_1} d_{\sum}(i) \geq (2/3) k_3 T. \tag{4.14}$$

The first inequality comes from removing the integer ceiling and canceling $k_3 \alpha T$. In the last inequality, we have used the fact that $d_{\sum}(i) > (2/3)T$ for each Group-1 surgeon. In the feasible solution, the cost is $k_3 T$ because each Group-1 surgeon is assigned to a separate room with long shift. Therefore, (4.12), (4.13) and (4.14) together lead to $LB_{g_1} \geq (2/3) F_{g_1}$.

For Group-2 surgeons, the lower bound incurs a cost of at least $g_2\alpha T$ because these surgeons require at least one short shift each. As mentioned earlier, we do not count the amount of Group-2 surgeons' work that is placed in extended B_1 shifts. A feasible solution is obtained by extending at most $(g_2/2)$ shifts into long shifts because at least two Group-2 surgeons' second pieces can be fitted in $(1-\alpha)T$. Therefore, the cost for Group-2 surgeons in a feasible solution is $(g_2\alpha T + \lceil \frac{1}{2}g_2 \rceil (1-\alpha)T)$.

When g_2 is even, the ratio of the lower bound to the feasible turns out to be

$$\frac{LB_{g_2}}{F_{g_2}} = \frac{g_2\alpha T}{g_2\alpha T + \lceil \frac{1}{2}g_2 \rceil (1-\alpha)T} = \frac{2\alpha}{1+\alpha} \geq \frac{2}{3},\ \forall\ \alpha \in (\frac{1}{2}, \frac{2}{3}). \tag{4.15}$$

Similarly, when g_2 is odd, we have

$$\frac{LB_{g_2}}{F_{g_2}} = \frac{g_2\alpha T}{g_2\alpha T + \lceil \frac{1}{2}g_2 \rceil (1-\alpha)T} = \frac{g_2\alpha T}{g_2\alpha T + \frac{1}{2}(g_2+1)(1-\alpha)T} = \frac{2g_2\alpha}{g_2(1+\alpha)+1-\alpha}$$
$$= \frac{2\alpha}{(1+\alpha)+\frac{1}{g_2}(1-\alpha)}. \tag{4.16}$$

This means $LB_{g_2} \geq (2/3)(F_{g_2}-1)$ when g_2 is odd. So (4.15) and (4.16) together lead to $LB_{g_2} \geq (2/3)(F_{g_2}-1)$.

Case (2): $\hat{i} > g_1$. The arguments we presented above work when there are no k_3 Group-1 surgeons, which is the consequence of having $\hat{i} > g_1$. We omit details in the interest of brevity.

[A_1 surgeons:] Jobs belonging to A_1 surgeons can be placed into four types of shifts: (1) long shifts consisting of C-shifts, (2) extended B_1-shifts (3) B_2 or B_1-type short shifts, and (4) long shifts introduced for A_1 surgeons. Because $C_2(J(i)) \leq (1/2)T$ when i is the index of an A_1 surgeon, we are able to organize their jobs into O-chains before fluid filling into long shifts. Then, following the proof given in Theorem 4.5.7, we can recover a feasible solution in scenarios (1), (2) and (4) that is at most (3/2) of the lower bound, with one exception that we discuss below.

In case (2), the last extended B_1 may split an A_1 surgeon's chain. When that happens, we arrange a new long shift in the feasible solution. And this issue may happen for A_2 or $O(\alpha)$ surgeon also, but only one surgeon can be split. So we include a shared "-1" in the feasible solution cost in our calculation of the ratio of LB to F. See the footnote in Table 4.8.

In addition to the problem identified above, it may happen that an A_1 surgeon's chain is split at the end of a long shift as well as a short shift, e.g. when filling in C and B_2-type shifts. When we obtain a feasible solution by placing all of the surgeon's work in a long shift, this results in a lower-bound to feasible solution ratio of $(1+\alpha)$ to $(2+\alpha)$, which is smaller than $(2/3)$. However, the above situation may occur at most twice since the long-to-short transition can only happen between C and B_2 shifts and extended and unextended B_1 shifts. Therefore, the performance guarantee remains $(2/3)$ in an asymptotic sense.

Next, we analyze Case (3), i.e. when A_1 surgeons' jobs are filled in short shifts. Recall that the amount of empty space in each B_2-shift or short B_1-shift is smaller than $(\alpha-(1/2))T$ because $C_2(J(i)) > (1/2)T$ for B_2 and B_1 surgeons. Also, we have $d_{\sum} > \alpha T$ for any A_1 surgeon. Let v denote the number of A_1 surgeons whose jobs are placed into these short shifts. Therefore, we would have needed at least $\lceil v\alpha/(\alpha-(1/2))\rceil$ such short shifts for each A_1 surgeon. For each A_1 surgeon split, we can find a feasible solution by placing all of his or her work into a separate long shift. Then, the ratio of the LB to the feasible solution cost is

$$\frac{\lceil v(\frac{\alpha}{\alpha-\frac{1}{2}})\rceil\alpha}{\lceil v(\frac{\alpha}{\alpha-\frac{1}{2}})\rceil\alpha+v} \geq \frac{v(\frac{\alpha}{\alpha-\frac{1}{2}})\alpha}{v(\frac{\alpha}{\alpha-\frac{1}{2}})\alpha+v} \geq \frac{2}{3},\ \forall\ \alpha\in(\frac{1}{2},\ \frac{2}{3}). \tag{4.17}$$

Together with the fact that the situation in which the $(2/3)$ ratio is violated can cause at most two fewer shifts in the LB, we have $LB_{A_1} \geq (2/3)(F_{A_1}-2)$.

[A_2 surgeons:] Jobs belonging to A_2 surgeons can be placed into three types of shifts: (1) long shifts consisting of C-shifts and extended B_1-shifts, (2) B_2 or B_1-type short shifts, and (3) long shifts introduced for either A_1 or A_2 surgeons. From arguments similar to those presented above, it suffices to focus on Case (2). Because B_2 surgeons' jobs consume at least $(1/2)T$ within B_2 and short B_1 shifts, when A_2 surgeons' jobs are assigned to them, each surgeon's jobs must take at least two shifts (because $\frac{\frac{1}{2}\alpha T}{(\alpha-\frac{1}{2})T} > 2$). We recover a feasible solution by placing each A_2 surgeon's jobs into a separate short shift. This means the ratio of LB to feasible cost is at least $\frac{2\alpha T}{3\alpha T} = \frac{2}{3}$. This gives us $LB_{A_2} \geq (2/3)F_{A_2}$.

[$O(\alpha)$ surgeons:] The $O(\alpha)$ surgeons have the property that any two splits can be recovered with a single short shift. This argument is identical to what we presented in Theorem 4.5.7. We omit details in the interest of brevity. Here we have $LB_{O(\alpha)} \geq (2/3)F_{O(\alpha)}$.

By taking sum of all parts, we have $LB \geq (2/3)(F-4)$, as shown in Table 4.8. That is, the ratio of the sum of lower bound and the sum of feasible solution costs is asymptotically (2/3). Hence proved. □

In the above we have completed the lower bound construction for $\frac{1}{2} < \alpha < \frac{2}{3}$. The idea for the other two cases are similar but the constructions also need special care. We explain the procedures in the following two subsections.

4.6.4 Two Shift Types with $0 < \alpha \leq \frac{1}{2}$

When $0 < \alpha \leq \frac{1}{2}$, the surgeon types, the LB algorithm, and the construction of a feasible solution are very similar to what we presented in Section 4.5. We provide the details below.

Step 1: Surgeon Types We utilize the definitions of A- and O-type surgeons from Section 4.5 (see Definition 4.5.3). No new surgeon types are needed in this case.

Step 2: Lower bound Construction The LB algorithm involves the following steps.

LB Algorithm

Step 1: Arrange A-type surgeons' jobs in arbitrary chains and place them in separate long rooms (one room per surgeon), such that each shift starts at the same time. This step uses n_A rooms and each room may have unused time. The remaining surgeons are all O-type surgeons.

Step 2: Arrange O-type surgeons' jobs in arbitrary O-chains and assign these chains one at a time using the fluid filling routine.

Step 3: If only n_A shifts are used, then return $L = n_A$ as the lower bound. If $\hat{L} > n_A$ shifts are used, then only the last shift can have unused time. Return $L = \hat{L} - 1$ as the lower bound if the last shift has unused time, otherwise return $L = \hat{L}$.

Lemma 4.6.7. *L is a valid lower bound.*

Proof: Using arguments similar to those in Lemmas 4.5.5 and 4.5.6, we argue that the number of long shifts used cannot be smaller than n_A. Therefore, if in Step 3, the first case occurs, then $L = n_A$ is trivially a valid lower bound. If the second case occurs, then we remove the last shift from $\hat{L}$ with the result that L is no more than the total duration of all jobs. Once again, it is a valid lower bound. □

Step 3: Feasible Solution Construction

Theorem 4.6.8. $\frac{L}{z^*} \geq \frac{\hat{L}}{z^*} - \frac{1}{z^*} \geq \frac{2}{3} - \frac{1}{z^*}$. *That is, L is an asymptotic* 2/3*-lower bound.*

Proof: It suffices to prove that $\frac{\hat{L}}{z^*} \geq \frac{2}{3}$. The LB Algorithm results in at most $(\hat{L}-1)$ O-type surgeons' jobs to be cut. From arguments similar to those presented in Theorem 4.5.7, we can recover a feasible assignment of O-type surgeons' jobs by introducing at most $\lceil \frac{1}{2}(\hat{L}-1) \rceil$ long shifts. This guarantees that $\frac{\hat{L}}{z^*} \geq \frac{2}{3}$. □

G.5 Two Shift Types with $2/3 \leq \alpha < 1$

For $\alpha \in [2/3, 1)$, we use the same surgeon classification and lower bound construction methods that were introduced in Sections 4.6.1 and 4.6.2. Therefore, we do not repeat them here. Note that the validity of the lower bound, proved in 4.6.2, does not depend on the value of α. Therefore, it only remains to show how to construct a feasible solution and that its associated cost is not more than (3/2) of the lower bound. We proceed to do that next.

Feasible Solution Construction

Theorem 4.6.9. *L is an asymptotic* $\frac{2}{3}$*-lower bound.*

Proof: Recall that when constructing the lower bound, we may use empty spaces in C- and B_2-shifts to accommodate other types of surgeons in a fluid manner, causing some of the assignments to be infeasible. We focus on how to recover a feasible solution from such assignments. Similar to Theorem 4.6.6, we also present the sketch of the proof in Table 4.9.

[B_1-type surgeons:] In LB construction, we use at least n_{B_1} short shifts to accommodate the longer pieces of chains of B_1 surgeons' jobs. Next, the second (shorter) pieces may be placed into the four classes of shifts in the following sequence: C-shifts, B_2-shifts, B_1-shifts, and the extended portions of B_1-shifts. These assignments are made such that Group 1 surgeons' jobs are assigned first.

A straightforward way to construct a feasible assignment of B_1-type surgeons's jobs (which we refer to as recovering a feasible solution) is to extend the n_{B_1} short shifts and place these surgeons' jobs in separate long shifts. Then, the LB cost is at least $n_{B_1}\alpha T$ and the feasible solution cost is at most $n_{B_1}T$. Their ratio is no less than α, which is no less than (2/3).

Table 4.9: Lower Bound (LB) and Feasible Solution (F) Costs

Surgeon Type	Shifts Involved (Full or Partial)	LB & Feasible Solution Cost	Relationship
C & B_2	Shifts that do not split any surgeon	LB_0 & F_0	$LB_0 = F_0 \geq (2/3)F_0$
B_1	(1) C & B_2-shifts, (2) unextended first g_1 B_1-shifts, (3) extended parts of B_1-shifts	LB_{B_1} & F_{B_1}	$LB_{B_1} \geq (2/3)F_{B_1}$
A_1	(1) C & B_2-shifts, (2) B_1-shifts & A_1-shifts	LB_{A_1} & F_{A_1}	$LB_{A_1} \geq (2/3)(F_{A_1} - 2)$
A_2	(1) C & B_2-shifts, (2) B_1-shifts, A_1, & A_2-shifts	LB_{A_2} & F_{A_2}	$LB_{A_2} \geq (2/3)F_{A_2}$
$O(\alpha)$	(1) C & B_2-shifts, (2) B_1-shifts, A_1, A_2, & $O(\alpha)$-shifts	$LB_{O(\alpha)}$ & $F_{O(\alpha)}$	$LB_{O(\alpha)} \geq (2/3)F_{O(\alpha)}$
All	All Shifts	LB & F	$LB \geq (2/3)(F-3)^*$

*a shared -1 is included for the case when the last extended B_1 shift splits an A_1, A_2, or $O(\alpha)$ surgeon.

[A_1-type surgeons:] The analysis for A_1-type surgeons is identical to what we presented in Theorem 4.6.6. We do not repeat these arguments for sake of brevity.

Recall that we also addressed the case when the last extended B_1 split one A_1, A_2 or $O(\alpha)$ surgeon. Similar to Theorem 4.6.6, we include a shared "-1" in the feasible solution cost in our calculation of the ratio of LB to F.

[A_2-type surgeons:] Because each A_2 surgeon has property $(1/2)\alpha T < C_2 < (1/2)T$, we treat them as O surgeons if split at long shifts. For brevity we do not repeat the argument. Next we only focus on A_2 surgeons whose chains are split by short shifts. By the LB Algorithm, these short shifts can only be B_2 or unextended B_1-shifts, since all A_1 and A_2-shifts are long and these surgeons cannot be split by an $O(\alpha)$-shift. We combine every two A_2 surgeons whose chains are split by short shifts and consider each pair of surgeons in the following way.

Because each A_2 surgeon has the property $(1/2)\alpha T < C_2 < (1/2)T$, we take an

optimal solution of $P_2||C_{\max}$ of each A_2 surgeon's jobs and call the longer part the first piece and the shorter part the second piece. Note that the second piece may be empty. For every two A_2 surgeons, we combine their first pieces and call the combination an E-type pseudo surgeon. Similarly, we also combine their second pieces and call the combination an F-type pseudo surgeon. The job duration of each E-type surgeon cannot exceed T because $C_2 \leq T/2$. E surgeons have the property that $d_{\sum} > \alpha T$, which means that these surgeons are similar to A_1-type. F surgeons have the property that $d_{\sum} < \alpha T$ and $C_2 < \frac{1}{2}\alpha T$, which makes them analogous to $O(\alpha)$-type surgeons.

When filling A_2-type surgeons' jobs in a fluid manner, we can also assume that all the first pieces of A_2-type surgeons are assigned before all the second pieces, since that sequence does not influence the lower bound. Now, to construct a feasible solution, we treat E pseudo surgeons the same as A_1 surgeons and F pseudo surgeons the same as $O(\alpha)$ surgeons. The feasible solution construction for both types is described in the proof of Theorem 4.5.7. Hence, we do not include the proof of (2/3) performance of the lower bound. However, because each E and F pseudo surgeon has jobs belonging to two surgeons, an extra operation is needed, in order to ensure that each A_2-type surgeon's jobs do not overlap. We perform the following operations without changing the cost to ensure that each A_2 surgeon does not have overlap in the feasible solution.

1. Ensure that each E-type pseudo surgeon's two pieces lie on different sides of $(1/2)T$. As for A_1 surgeons, when E-type pseudo surgeon is split in short shifts, we construct feasible solution by placing the surgeon's entire work into a new long shift. This guarantees (2/3) lower bound performance and ensures that we can place the two pieces as described. Because E-type surgeons have property $d_{\sum} > \alpha T$, each E pseudo surgeon that is placed in short shifts must be split.

2. Some F pseudo surgeon may not be split. If an F surgeon is not split in B_2 or unextended B_1 shift, that shift includes at least three pieces: the B_2 surgeon or the B_1 surgeon's first piece; the first piece of the F pseudo surgeon, and the second piece of the F pseudo surgeon. These three pieces are not split. Now the B_2 surgeon or the B_1 surgeon's first piece is placed on the left. In feasible solution construction, we do not incur extra cost, but do the following to avoid overlapping of the involved A_2 surgeons: we switch position such that the B_2 surgeon or the B_1

surgeon's first piece is in the middle, and the two pieces of the F pseudo surgeon are on left and right side. This way, we ensure that the two pieces lie on different sides of $(1/2)T$ since the B_2 surgeon or the B_1 surgeon's first piece is longer than $(1/2)T$.

3. Since F pseudo surgeons' work can be very short, there can be cases in which more than one F pseudo surgeons are placed in the same B_2 or unextended B_1 shift without being split. In such cases we can still arrange the position such that the B_2 surgeon or the B_1 surgeon's first piece is in the middle, and each F pseudo surgeon's two pieces lie in both sides.

4. Some F pseudo surgeons may be split. If an F pseudo surgeon is split twice or more times, we introduce a new short shift and place the F surgeon entirely in the new shift. Let d_1 and d_2 denote the durations of the two pieces. We can ensure that the two pieces are placed from the two ends of the shift. That is, the first piece is placed from 0 to d_1, and the second place is placed from $(\alpha T - d_2)$ to αT. Here, although one piece of the F pseudo surgeon may cross $(1/2)T$, we can avoid overlapping as by switching sides. Switching ensures that if an A_2 surgeon's first piece is in the upper left side, then his or her second piece is in the lower right side. If they overlap, then that means the duration sum of the surgeon is greater than αT, which violates the definition of A_2 surgeon.

[$O(\alpha)$ surgeons:] The argument for $O(\alpha)$-type surgeons is identical to what we presented in Theorem 4.5.7. We omit details in the interest of brevity.

At this point in time, we have considered all possible cases and shown how to recover a feasible solution such that in each case, the ratio of the lower bound to the cost of feasible solution is asymptotically no less than (2/3). Hence proved. □

4.7 Numerical Experiments and Insights

We implemented our approach on data from the three hospitals and tabulated two types of impacts: (1) on staffing costs, and (2) on surgeons. These experiments reveal the essential tradeoffs for hospitals considering gainsharing with physician groups to

realize staffing cost reductions. We begin with the results related to efficiency (hospital perspective), which are presented in Table 4.10.

In Table 4.10, we first calculate efficiency gains from non-urgent cases only. Later, we consider the combined effect of both urgent and non-urgent cases. Because the data contained instances in which urgent cases were "fitted" in open time between non-urgent cases, and this were not possible after rescheduling (which created a more tightly packed schedule), we included the cost of staffing dedicated rooms for urgent and emergent cases in the second part of our analysis. In Table 4.10, "before" refers to statistics based on data obtained from the hospitals and "after" refers to similar statistics obtained after applying our rescheduling algorithm. Note that all three hospitals exhibit substantial decrease in staffed OR requirements for non-urgent cases and concomitant gains in utilization. Planned utilization gains range from 23 to 34 percent with Hospitals 2 and 3 showing above 30 percent gains. The realized utilization is calculated using actual case lengths as opposed to planned case lengths. In these calculations, we included delays that were caused in the original schedule by the surgeon arriving late. However, delays that were caused by the sequence of surgeries were recalculated based on the new sequence. We also kept day-of-surgery cancelations intact when calculating the effect of rescheduling.

The realized utilization gains are smaller. The difference comes from the fact that the new schedule uses significantly fewer rooms. Therefore, idleness introduced in the revised schedule by late surgeon-arrival, sequence-related delays, and surgeries completing earlier than planned, occupy a much greater percent of the total staffed time. The number of staffed ORs are calculated using the planned case lengths and we count the amount of overtime that would be needed to accommodate non-urgent cases in the original and revised schedules. Savings are counted only when an OR is not staffed for the entire day and overtime costs are subtracted from such savings. The numbers we report are average daily savings.

The last five rows of Table 4.10 show the impact of considering urgent and emergent cases. Using a simple local search, we find the fixed number of dedicated ORs that would minimize the cost of scheduling urgent and emergent cases for each hospital. Hospitals would commit to staffing these rooms and their staffing costs would be incurred regardless of realized urgent/emergent demand. The optimal number of dedicated rooms

Table 4.10: OR Efficiency Metrics

	Hospital 1		Hospital 2		Hospital 3	
Non-Urgent Only	Planned	Realized	Planned	Realized	Planned	Realized
Utilization (before)	65.24	60.41	54.06	50.68	49.34	45.66
Utilization (after)	88.22	80.83	86.78	75.61	83.55	69.15
# of staffed ORs (before)	11.9, 1.6* (14.3)		8.3		7.7	
# of staffed ORs (after)	4.0, 4.6* (10.9)		5.7		4.5	
Daily Overtime (before)	281		305		36	
Daily Overtime (after)	133		321		280	
$ Savings/day	29,970		18,360		23,310	
With Urgent Cases						
# of staffed ORs (before)	10.7, 3.2* (15.5)		9.0		8.7	
# of staffed ORs (after)	6.0, 4.6* (12.9)		6.7		5.5	
Daily Overtime (before)	$281 + 170 = 451$		$305 + 60 = 365$		$36 + 70 = 106$	
Daily Overtime (after)	$133 + 380 = 513$		$321 + 90 = 411$		$280 + 200 = 480$	
$ Savings/day	17,325		15,525		20,385	

$ Savings/day are based on $15/minute of regular OR time and $22.5/minute of overtime
*Entries marked with an asterisk show ORs with long shifts.
Numbers in parentheses show equivalent number of 8-hour shift.

were 2 (8-hour shifts) for Hospital, 1 (8-hour shift) for Hospital 2 and 1 (10-hour shift) for Hospital 3. Urgent cases are scheduled as compactly as possible in the order of arrival. Cases that cannot be accommodated in dedicated rooms are scheduled as add-on cases at the end of shift and incur overtime charges. Note that projected savings decline, but remain substantial nonetheless. Notwithstanding potential cost savings, dedicated ORs for urgent/emergent cases may also improve health outcomes because urgent cases no longer have to wait until a suitable opening in the existing schedule of non-urgent cases (see [57]).

We notice in Table 4.10 that the effect on the use of overtime is quite different in the three hospitals. Overtime use decreases in Hospital 1, remains about the same in Hospital 2, and increases in Hospital 3. This can be explained based on structural differences among these hospitals. Hospital 1 has the ability to use long shifts. Therefore, by planning to staff more ORs with long shifts, as our algorithm recommends, it can reduce the use of overtime while at the same time reducing the requirement to staff a large number of concurrent rooms. Hospital 2 does not have this flexibility and its use of overtime increases as one would expect. Hospital 3 has long open times between

surgeries in the original schedule. This results in an unusually low overtime usage in the original schedule. Such open times are eliminated by our algorithm, resulting in overtime use that is similar to that in other hospitals.

One of the key structural differences among the three hospitals is the relative size of open intervals between scheduled cases in the data. We find that Hospital 3 tends to leave larger intervals open. This difference explains, to some extent, the differences in the realized performance of the rescheduling algorithm. We illustrate the differences by plotting the proportion of total idle time (in the planned schedule) that is accounted for by a certain count of open intervals, after these intervals are arranged from the longest to the shortest – see Figure 4.7, which shows the distribution of open intervals. We find that Hospital 1 requires a stochastically larger number of intervals to achieve the same proportion of idle time within its schedule. Put differently, it tends to leave open small intervals of unused time between procedures. In contrast, Hospital 3 leaves larger chunks of open time and Hospital 2 lies somewhere between these two.

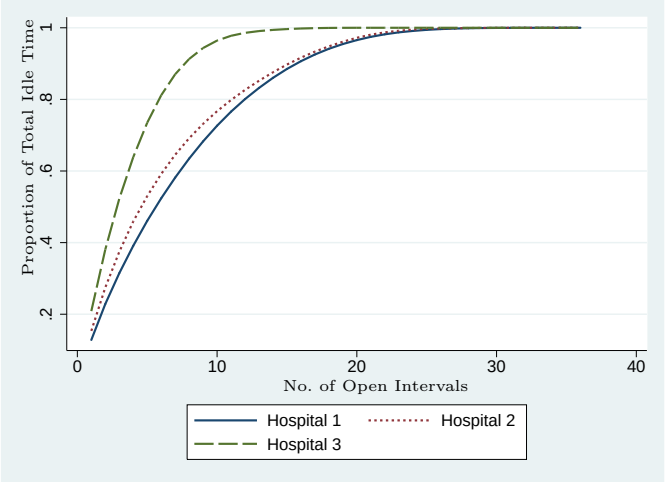
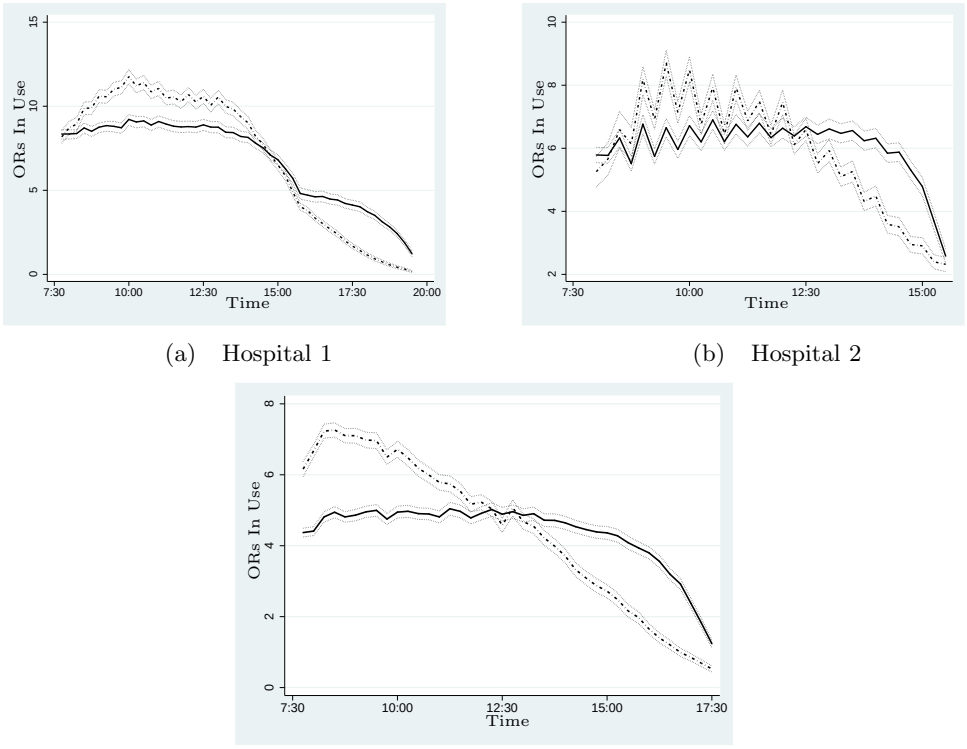


Figure 4.7: Distribution of Open Intervals in Planned Schedule

How are the number of staffed ORs affected by the rescheduling procedure? Table 4.10 suggests that surgeries are packed more efficiently and that Hospital 1 will need more long shifts. In order to provide greater insight into how greater efficiency is realized, we plot the profile of number of ORs in use in each 15-minute interval of the day from 7:30 AM till 7:30 PM in Figure 4.8. What we show here are the average

number of ORs in use based on original and rescheduled start times and actual surgery durations. We also show 95% confidence intervals because the actual usage of ORs changes from one day to the next.



(a) Hospital 1

(b) Hospital 2

(c) Hospital 3

Figure 4.8: Number of ORs In Use by Time of Day

In Figure 4.8, dash-dotted lines show original schedule and solid lines show revised schedules.

A common trend across all hospitals is the flattening of OR-use requirements. Rescheduling creates more uniform utilization of ORs throughout the day, which allows the hospital to staff fewer ORs concurrently and achieve greater utilization. We also see that Hospital 1 benefits from planning to open more long rooms than it currently does. In cases such as these, the hospital administration may need to work with nursing coordinators to identify staff who are willing to work long shifts.

Practitioners are also interested in knowing how rescheduling affects the work day

of surgeons. In order to present this information in a succinct manner, we developed a number of metrics to compare before and after rescheduling results. These results are summarized in Table 4.11. OR managers may be concerned that a denser packing of surgical procedures may lead to greater surgeon delays. We calculate three types of delays. Type 1 delays occur when the affected doctor is delayed by late finish of a preceding procedure performed by a different surgeon, Type 2 delays occur when the affected doctor is delayed but he or she performed the previous case in a different OR, and Type 3 delays occur when the affected doctor is running late for an earlier procedure performed by the same surgeon in the same room. Clearly, Type 1 delays are more serious from doctors' perspectives than Type 2 or Type 3 delays. Calculations reported in Table 4.11 show that doctors operating in Hospital 1 and 2 can expect greater frequency of Type 1 delays, but the average number of minutes delayed will be smaller. This happens because our algorithm schedules procedures performed by doctors with multiple cases and long durations in the same room. For reasons explained via Figure 4.7, Hospital 3 is different. Doctors in that hospital will experience relatively more Type-1 delays because in the original schedule, they experience very small delays on account of having large chunks of unused times between scheduled procedures. The differences between mean delays are statistically significant (p-values are close to zero in all three cases).

The effect on Type 2 and Type 3 delays are quite different. Generally, both the frequency and mean delays increase upon rescheduling. Similarly, the total time that a doctor spends performing surgeries (from the start time of their first case to the end time of their last case), which we call spread, increases as a result of rescheduling. Upon performing statistical tests, we found the mean differences to be statistically significant between before and after mean spreads across all hospitals. We also find that rescheduling will cause a significant proportion of doctors to either report earlier or later than the first case in their original schedule. Also, rescheduling will cause surgeons to have more idle time in between surgical procedures. We found that the average daily increase in surgeons' idleness across all surgeons amounted to 249 minutes in Hospital 1 (with SD = 577 minutes), 272 minutes in Hospital 2 (SD = 570 minutes) and 376 minutes in Hospital 3 (SD = 343 minutes).

Doctors often decide the sequence in which they prefer to perform surgeries on their

Table 4.11: Impact on Surgeons

	Hospital 1		Hospital 2		Hospital 3	
	Before	After	Before	After	Before	After
Type-1 Delay						
Count (%)	557 (9%)	1352 (21%)	300 (4%)	3023 (38%)	85 (1%)	1470 (17%)
Avg (min)	95	80	88	78	53	124
Type-2 Delay						
Count (%)	931 (15%)	709 (11%)	395 (5%)	2216 (28%)	178 (2%)	616 (7%)
Avg (min)	45	220	62	193	46	244
Type-3 Delay						
Count (%)	676 (11%)	442 (7%)	2411 (30%)	731 (9%)	1586(19%)	1716 (20%)
Avg (min)	36	32	51	84	79	97
Spread						
Avg (min)	248	280	250	324	279	337
SD	149	173	151	198	157	232
Early Report Time (planned)						
Count (%)	1465 (37%)		1386 (60%)		765(24%)	
Avg (min)	174		114		165	
Late Report Time (planned)						
Count (%)	1688 (46%)		753 (33%)		1316(41%)	
Avg (min)	207		117		198	

Avg = average, SD = standard deviation

Type-1: Same room different MD; Type-2: Same MD different room; Type-3: Same room same MD

OR day. Rescheduling may produce an undesirable sequence. However, if a doctor's cases are scheduled consecutively, it will be possible for that doctor to rearrange the sequence of his or her surgeries without affecting the overall schedule. When a doctor has multiple long cases, our algorithm favors placing that doctors' jobs in the same room. In order to calculate the flexibility that a hospital will have to re-sequence surgeries according to a doctor's wishes after running our algorithm, we calculated the percent of total surgery durations that occur in connected sequences. Connected means that the procedures are done by the same doctor and are placed consecutively in the same room. We found that in Hospital 1 and 2, 53.3 and 81.2 percent of surgery durations occurred in connected sequences in the original schedule. In contrast, after running our algorithm, these percentages were 59.9 and 86.9, respectively. Therefore, for these two hospitals, there will be flexibility to re-sequence surgeries if desired. Hospital 3 is once again different. In that hospital, 84.4 percent of surgery durations were in connected sequences in the data, whereas our algorithm produces a schedule in which 65.4 percent of the surgery durations occur in connected sequences. We believe that these differences relate to the way in which Hospital 3 scheduled cases and the case-mix of doctors who perform surgeries at that hospital.

The impact on doctors has to be weighed against the potential savings. Across the 3 hospitals, daily savings are sufficiently high that we believe it is not inconceivable that doctors will find it attractive to allow more flexible scheduling of their cases. The three hospitals may have available up to \$119, \$68, and \$62 per physician idle minute, respectively, to share with physicians or alternatively incur as additional cost of having salaried physicians idle. The exact details of gain-sharing plan need to be worked out separately in each situation because hospitals are likely to have a mix of independent and employed physicians. It is also possible to place additional constraints on the degree to which case start times may be changed. That will reduce the extent of savings, but may lead to greater doctor participation. We discuss extensions of our work in the next section.

In an effort to gain further managerial insights, we analyzed whether combining a surgeon's jobs into one single job would result in similar savings. The resulting formulation is simpler because overlap-avoidance constraints can be eliminated. We used CPLEX to solve that model with 5-minute time increments, and compared the

resulting staffing cost with that from optimal solutions without combining cases – see Table 4.12. The reason why we used 5-minute increments is that CPLEX is not able to solve quite a few instances of the problem for Hospital 1 with 1-minute increments when we do not consolidate a surgeon's cases. That is, the benchmark scenario against which we compare the effect of consolidation requires us to consider 5-minute increments. Savings are calculated based on \$15 per minute. Thus, not combining same-doctor cases into a single long case will save about 1 staffed OR every 2 weeks in Hospitals 1 and 2, and about 2 staffed ORs every 2 weeks in Hospital 3.

Table 4.12: Effect of Consolidating Same-doctor Cases

	Cases not combined	Cases combined	% Difference	\$ Difference
H 1 (equivalent 8-hr shifts)	10.5	10.6	1%	\$720/day
H 2 (8-hr shifts)	5.2	5.3	2%	\$720/day
H 3 (10-hr shifts)	4.2	4.4	5%	\$1800/day

Next, we considered what would happen if we were to allow a small overlap between end time of one procedure and the start time of the next procedure, so long as both procedures are performed by the same surgeon. Such overlap is sometimes possible when a surgeon is assisted by another. However, upon solving cases with and without permissible 10-minute overlap, we found no difference in the optimal number of ORs needed. The key reason behind this finding is that cost is determined by the number of concurrent ORs used, which was not affected upon allowing a 10-minute overlap.

Finally, we analyzed the impact of using two shift types in Hospitals 2 and 3, which currently use a single shift type. For concreteness, the shift lengths chosen were 8 and 12 hours. Results are shown in Table 4.13. In Column 3 of Table 4.13, we provide the number of 8 and 12 hour shifts that would be needed. The quantity in brackets shows the equivalent number of original shifts. We show percentage and dollar savings in Columns 4 and 5.

The potential savings from rescheduling need to be weighed against the impact on surgeons. Across the three hospitals, daily savings are sufficiently high that we believe it is not inconceivable that doctors will find it attractive to allow more flexible scheduling

Table 4.13: Impact of Using Two Shifts

	Original shift structure	Two shift structure (Equivalent original shifts)	% Difference	$ Difference
H 2	5.22×8hr	3.1×8hr, 1.1×12hr, (4.78×8hr)	8%	$3200/day
H 3	4.23×10hr	3.0×8hr, 1.4×12hr, (4.06×10hr)	4%	$1500/day

of their cases. The three hospitals may have available up to $70, $57, and $54 per physician idle minute, respectively, to share with physicians or alternatively incur as additional cost of having salaried physicians idle. The exact details of gain-sharing plan need to be worked out separately in each situation because hospitals are likely to have a mix of independent and employed physicians. It is also possible to place additional constraints on the degree to which case start times may be changed. That will reduce the extent of savings, but may lead to greater doctor participation. We discuss extensions of our work in the next section.

4.8 Extensions and Concluding Remarks

Practitioner considerations may lead to alternate formulations and further extensions of our work. Surgeons may wish to have all of their cases scheduled within a short time window, i.e. without too many breaks in between so they can utilize their time more effectively. A surgeon may also wish to have all of his or her cases scheduled either in the AM or the PM block if the total duration is no more than 4 hours. We refer to such constraints as spread constraints. Our branch-and-bound algorithm can deal with such constraints, and our lower bounds will be valid, but its worst-case performance will be reduced to (1/2) from (2/3). The key to obtaining a bound with guaranteed performance is that each time a chain is split, we place the chain (which includes all of a surgeon's jobs) into a new empty room. This way, we recover feasibility by using at most twice as many rooms as in the lower bound. Investigation of better ways of constructing lower bounds and feasible solutions are topics for future research.

Some hospitals have specialized equipment in some rooms, but not all rooms, which gives rise to a constraint that certain cases can be scheduled only in some rooms. If

the rooms with specialized equipment are not used for routine cases, then the problem of rescheduling cases can be divided into two separate problems and solved using our methodology. However, when rooms with specialized equipment are also routinely used for cases that do not require such equipment, the problem of rescheduling cases remains a challenge. Similarly, some hospitals have limited copies of movable equipment that they wheel from one OR to another. In this case, it would be necessary to make sure that the number of concurrently scheduled cases that require a particular piece of equipment do not exceed the number of available pieces of that equipment, creating an an additional non-overlapping constraint. Such constraints are also difficult to deal with. In both scenarios, our branch-and-bound algorithm and lower bounds will remain valid, but the worst-case performance guarantee will not apply. We believe such problem settings provide important areas for future work.

Consistent with common practice, we assume that at the time when surgeries are rescheduled, the hospital does not consider using strategic overtime. In some instances, it may be more economical to use a small amount of overtime and avoid staffing a room for the entire shift length. Rescheduling with the use of strategic overtime is a hard problem, which requires a great deal of information about work rules and availability of scheduled overtime. One of the primary reason why hospitals do not consider strategic overtime is that rescheduling is done at least two days before the surgery date. Many more surgeries will be booked after the rescheduling is done, which may use open time in staffed rooms and also lead to the use of overtime anyway. That is, there is a potential that the empty space in an OR that is not well utilized will be required for other surgeries that are scheduled late. If we allow strategic overtime, our theoretical bounds may not remain intact. We believe considering extensions of our model with strategic overtime is another area for future research.

The analysis presented in this paper leads to several managerial insights. First, it shows that significant improvements in OR utilization are possible. Hospitals that are able to obtain cooperation from their surgeons can increase case volumes with the same number of ORs and lower staffing costs, or open up block time for additional surgeons. Second, our analysis identifies patterns of surgical case durations that should be placed in a single OR and those that may be spread across multiple rooms. These patterns can be explained to OR schedulers and may lead to better initial schedules. Third, the

analysis shows that the use of an appropriate number of long shifts is beneficial. In particular, Hospital 1 in our data sample used two shift lengths. Upon rescheduling, we found that Hospital 1 realized the greatest efficiency gains, which is likely due to the fact that our algorithm selected an optimal number of long shifts. A take away for hospital executives is to determine the optimal mix of short and long shifts, and to incentivize staff to work long shifts.

Chapter 5

Conclusions

Each Chapter contains a conclusion section. In what follows, we briefly summarize the similarities and differences across the three Chapters and discuss future work to conclude this thesis.

Motivated by improving reserve driver performance, offline operational fixed-job scheduling models are studied for the day-before reserve driver scheduling and work assignment problems in Chapter 2. With knowledge of all jobs that need to be performed, we consider different heuristics, and we are able to show that one algorithm has approximation ratio between $[1 - 1/e, 19/27]$. In the day-of reserve driver work assignment problem, we do not have information of future jobs so we consider the online model. With this difference, we cannot apply most of the approaches from the model for the day-before problem. As a result, future job durations are categorized into intervals with exponentially increasing sizes, and the worst-performance we could achieve is worse than the day-before model which is an offline model.

In our surgery rescheduling model, unlike the two models for the driver scheduling problems, only the job durations are fixed but we can decide the job start times. Also, each job is attached to a surgeon. Furthermore, we cannot reject any job, (in reserve driver scheduling, reject indicates assigning to overtime) and our objective is to minimize the number of ORs used. The main idea of our approach comes from bin packing literature, and to adapt the same-doctor constraints, we introduce a classification of surgeons which can be done by solving a parallel machine scheduling problem or a knapsack problem. The main contribution is finding performance-guaranteed lower

bound, and a performance-guaranteed upper bound or approximation algorithm and a branch-and-bound algorithm as byproducts.

In conclusion, two fixed-job scheduling models and a bin-packing model with resource constraints are studied. Among the three models, two models are deterministic and one is online. The majority of our study and the main contributions are algorithms with guaranteed worst-case performance. To our knowledge, all theoretical results are currently the best guarantees for such problems. Experiments on real data are also a common feature of the three Chapters. Our work is not only theoretically interesting but also practically useful, because all models are motivated by real-world problems. Moreover, our models are theoretically more general than the scope of their motivating problems, so our algorithms can potentially adapt to scheduling and work assignment problems in many other industries with appropriate adjustments.

Each of the three problems is theoretically difficult and therefore not solved to optimality in polynomial time. Instead, we present algorithms with provable approximation ratios and in what follows we discuss future work that remains for each of our models.

For the day-before reserve driver scheduling and assignment problem, we have yet to find the exact approximation ratio of our algorithm. The difficulty of preemptive but non partial credit version of OFJS-S is still open problem in the future. For the day-of reserve driver work assignment, how to schedule the shifts for the day-of reserve drivers is one of the directions that may be pursued in the future.

For surgery rescheduling, more challenges remain open. First, more types of spread constraints may arise. Some surgeons may require that all of their cases scheduled within a shorter time window, i.e. without too many breaks in between so they can utilize their time more effectively. A surgeon may also wish to have all of his or her cases scheduled either in the AM or the PM block if the total duration is no more than 4 hours. Second, equipment constraints can also arise. If there are not enough copies of the equipment, it may be necessary to make sure that the number of concurrently scheduled cases that require a particular equipment do not exceed the number available. Also, strategic overtime use may need to be addressed. Although these constraints can be potentially dealt with in the branch-and-bound algorithm, our current bounds may not be valid or the worst-case performance guarantee may be compromised.

References

[1] I. Gertsbakh and H. I. Stern. Minimal resources for fixed and variable job schedules. *Operations Research*, 26(1):68–85, 1978.

[2] A.W.J. Kolen, J.K. Lenstra, C.H. Papadimitriou, and F.C.R. Spieksma. Interval scheduling: A survey. *Naval Research Logistics*, 54(5):530–543, 2007.

[3] L. G. Kroon, M. Salomon, and L. N. Van Wassenhove. Exact and approximation algorithms for the operational fixed interval scheduling problem. *European Journal of Operational Research*, 82:190–205, 1995.

[4] S. Martello and P. Toth. A heuristic approach to the bus driver scheduling problem. *European Journal of Operational Research*, 24:106–117, 1986.

[5] C. P. DeAnnuntis and W. P. Morris. Transit extraboard management - optimum sizing and strategies final report. 2007. URL: `http://www.nctr.usf.edu/pdf/77707.pdf`.

[6] H. N. Koutsopoulos. Scheduling of extraboard operators in transit systems. *Transportation Sciences*, 24(2):87–105, 1990.

[7] L. C. MacDorman and J. C. MacDorman. The transit extraboard: some opportunities for cost savings. *APTA Annual Meeting*, 1982.

[8] J. L. Perry and L. Long. Extraboard scheduling, workers compensation, and operators stress in public transit: research results and managerial implications. *Transportation Research Record*, 1002:21–28, 1984.

[9] L. C. MacDorman. Extraboard management: procedures and tools. *National Cooperative Transit Research and Development Program, Synthesis of Transit Practice Report No. 5*, 1985.

[10] D. T. Eliiyi and M. Azizoglu. Spread time considerations in operational fixed job scheduling. *International Journal of Production Research*, 44(20):4343–4365, 2006.

[11] M. Fischetti, S. Martello, and P. Toth. The fixed job schedule problem with spread-time constraints. *Operations Research*, 35(6):849–858, 1987.

[12] M. Fischetti, S. Martello, and P. Toth. The fixed job schedule problem with working-time constraints. *Operations Research*, 37:395–403, 1989.

[13] M. Fischetti, S. Martello, and P. Toth. Approximation algorithms for fixed job schedule problems. *Operations Research*, 40(S):96–108, 1992.

[14] H. R. Lourenco, J. Paixao, and R. Portugal. Multiobjective metaheuristics for the bus driver scheduling problem. *Tansportation Science*, 35(3):331–343, 2001.

[15] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search heuristic. *Journal of Global Optimization*, 6:109–133, 1995.

[16] F. Sivrikaya-Serifoglu and G. Ulusoy. Parallel machine scheduling with earliness and tardiness penalties. *Computers and Operations Research*, 26:773–787, 1999.

[17] E. M. Arkin and B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1–8, 1987.

[18] P. Brucker and L. Nordmann. The k-track assignment problem. *Computing*, 52:97–122, 1994.

[19] U. Faigle and W. M. Nawijn. Note on scheduling intervals on-line. *Discrete Applied Mathematics*, 58:13–17, 1995.

[20] U. Faigle, W. Kern, and W. M. Nawijn. A greedy on-line algorithm for the k-track assignment problem. *Journal of Algorithms*, 31:196–210, 1999.

[21] O. Solyali and O. Ozpeynirci. Operational fixed job scheduling problem under spread time constraints: a branch-and-price algorithm. *International Journal of Production Research*, 47(7):1877–1893, 2009.

[22] R. Bhatia, J. Chuzhoy, A. Freund, and J. Naor. Algorithmic aspects of bandwidth trading. *ACM Trans. Algorithms*, 3(1):1–19, February 2007.

[23] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theorey of NP-completeness*. W. H. Freeman, 1979.

[24] D. S. Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Company, 20 Park Plaza, Boston, MA, 02116, 1997.

[25] S. Arora and C. Lund. *Hardness of approximations*. Chapter 10 in Approximation algorithms for NP-hard problems, Editor: Hochbaum, D. S., PWS Publishing Company, 20 Park Plaza, Boston, MA, 02116, 1997.

[26] A. W. J. Kolen and L. G. Kroon. On the computational complexity of (maximum) shift class scheduling. *European Journal of Operational Research*, 64:138–151, 1993.

[27] M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[28] B. Korte and J. Vygen. *Combinatorial optimization: theory and algorithms*. Springer, 2006.

[29] A. Ghouila-Houri. Caractérisation des matrices totalement unimodulaires. *Comptes rendus hebdomadaires des Séances de l'Académie des Sciences*, 254:1192–1194, 1962.

[30] S. Khuller, A. Moss, and S. J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70:39–45, 1999.

[31] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: an analytic study exact and approximate algorithms. *Management Science*, 23:789–810, 1977.

[32] M. Shaked and J.G. Shanthikumar. *Stochastic Orders and Their Applications*. Academic Press, New York, 1994.

[33] A. Müller and D. Stoyan. *Comparison Methods for Stochatic Models and Risks*. John Wiley & Sons, Chichester, 2002.

[34] M.Y. Kovalyov, C.T. Ng, and T.C.E. Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research*, 178(2):331–342, 2007.

[35] D. Gupta, F. Li, and N. Wilson. Extraboard workforce planning for bus transit operations. *CURA Reporter*, 41(3-4):11–18, 2011.

[36] R. J. Lipton and A. Tomkins. Online interval scheduling. *Proceedings of the fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, 1994.

[37] G. J. Woeginger. Online scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130(1):5–16, 1994.

[38] H. Miyazawa and T. Erlebach. An improved randomized online algorithm for a weighted interval selection problem. *Journal of Scheduling*, 7(4):293–311, 2004.

[39] Ulrich Faigle and Willem M Nawijn. Note on scheduling intervals on-line. *Discrete Applied Mathematics*, 58(1):13–17, 1995.

[40] S. Seiden. Randomized online interval scheduling. *Operations Research Letters*, 22:171–177, 1998.

[41] S. P. Y. Fung. Lower bounds on online deadline scheduling with preemption penalties. *Information Processing Letters*, 108(4):214–218, 2008.

[42] S. P. Y. Fung, C. K. Poon, and F. Zheng. Online interval scheduling: randomized and multiprocessor cases. *Journal of Combinatorial Optimization*, 16(3):248–262, 2008.

[43] S. Baruah, G. Koren, B. Mao, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On the competitivenes of on-line real-time task scheduling. *The Journal of Real-time Systems*, 4(2):125–144, 1992.

[44] G. Koren and D. Shasha. An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24:318–339, 1995.

[45] F. Zheng, W. Dai, P. Xiao, and Y. Zhao. Competitive strategies for on-line production order disposal problem. *Proc. of 1st International Conference on Algorithmic Applications in Management*, pages 46–54, 2005.

[46] F. Zheng, Y. Xu, and E. Zhang. On-line production order scheduling with preemption penalties. *Journal of Combinatorial Optimization*, 13(2):189–204, 2007.

[47] Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing*, 31(2):331–352, 2001.

[48] Ulrich Faigle, R Garbe, and Walter Kern. Randomized online algorithms for maximizing busy time interval scheduling. *Computing*, 56(2):95–104, 1996.

[49] Minos Garofalakis, Yannis Ioannidis, Banu Özden, and Avi Silberschatz. Competitive on-line scheduling of continuous-media streams. *Journal of Computer and System Sciences*, 64(2):219–248, 2002.

[50] F. Li and D. Gupta. The extraboard operator scheduling and work assignment problem. *IIE Transactions*, 2013, http://www.tandfonline.com/doi/pdf/10.1080/0740817X.2014.882036.

[51] B. Korte and D. Hausmann. An analysis of the greedy algorithm for independence systems. *Annals of Discrete Mathematics*, 2:65–74, 1978.

[52] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[53] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: an analytic study exact and approximate algorithms. *Management Science*, 23:789–810, 1977.

[54] R. L. Jackson. The business of surgery. *Health management Technology*, pages 20–22, 2002.

[55] C.J. DeFrances and M.J. Hall. 2005 National hospital discharge survey. *Adv Data*, 385:1–19, 2007.

[56] A. Macario. What does one minute of operating room cost? *Journal of Clinical Anesthesia*, 22:233–236, 2010.

[57] T. Bhattacharyya, M. S. Vrahas, S. M. Morrison, E. Kim, R. A. Wiklund, R. M. Smith, and H. E. Rubash. The value of the dedicated orthopaedic trauma operating room. *J Trauma*, 60(6):1336–40, 2006.

[58] Anonymous. Survey: hospital executives nationwide facing operating room case volume increases, seeking greater efficiency through healthcare information technology (HIT), 2012. available at `sisfirst.com/news-and-resources/pressreleases/survey`, Downloaded Oct 25, 2013.

[59] Anonymous. Affordable care act to improve quality of care for people with medicare, 2011. U.S. Department of Health and Human Services News Room, For Immediate Release March 31, 2011, available at `http://www.hhs.gov/news/press/2011pres/03/20110331a.html`, Downloaded Oct 21, 2013.

[60] S.M. Shortell. Bending the cost curve: A critical component of health care reform. *JAMA*, 302(11):1223–1224, 2009.

[61] D. Gupta. Surgical suites' operations management. *Production and Operations Management*, 16(6):689–700, NOV-DEC 2007.

[62] D.N. Pham and A. Klinkert. Surgical case scheduling as a generalized job shop scheduling problem. *European Journal of Operational Research*, 185:1011–1025, 2008.

[63] P.T. Vanberkel, R.J. Boucherie, E.W. Hans, J.L. Hurink, and N. Litvak. A survey of health care models that encompass multiple departments. *International Journal of Health Management and Information*, 1(1):37–69, 2010.

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

[65] F. Guerriero and R. Guido. Operational research in the management of the operating theatre: a survey. *Health care management science*, 14(1):89–114, 2011.

[66] Y. Gerchak, D. Gupta, and M. Henig. Reservation planning for elective surgery under uncertain demand for emergency surgery. *Management Science*, 42(3):321–334, 1996.

[67] B. Denton and D. Gupta. A sequential bounding approach for optimal appointment scheduling. *IIE Transactions*, 35(11):1003–1016, 2003.

[68] E. Hans, G. Wullink, M. van Houdenhoven, and G. Kazemier. Robust surgery loading. *European Journal of Operational Research*, 185(3):1038–1050, 2008.

[69] M.A. Begen and M. Queyranne. Appointment scheduling with discrete random durations. *Mathematics of Operations Research*, 41(2):845–854, 2009.

[70] M.A. Begen, R. Levi, and M. Queyranne. Technical note: A sampling-based approach to appointment scheduling. *Operations Research*, 60(3):675–681, 2012.

[71] Q. Kong, C.Y. Lee, C.P. Teo, and Z Zheng. Scheduling arrivals to a stochastic service delivery system using copositive cones. *Operations Research*, 61(3):711–726, 2013.

[72] H. Fei, N. Meskens, and C. Chu. A planning and scheduling problem for an operating theatre using an open scheduling strategy. *Computers & Industrial Engineering*, 58(2):221–230, 2010.

[73] V.N. Hsu, R. de Matta, and C.Y. Lee. Scheduling patients in an ambulatory surgical center. *Naval Research Logistics (NRL)*, 50(3):218–238, 2003.

[74] J. Błażewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European journal of operational research*, 93(1):1–33, 1996.

[75] E.G. Jr. Coffman, M.R. Garey, and D.S. Johnson. *Approximation Algorithms for Bin Packing: A Survey.* Chapter 2 in Approximation Algorithms for NP-Hard Problems, Editor: Hochbaum, D., PWS Publishing, 20 Park Plaza, Boston, MA, 02116, 1999.

[76] L. Epstein and A. Levin. On bin packing with conflicts. In *Approximation and Online Algorithms*, pages 160–173. Springer, 2007.

[77] G. Galambos and G.J. Woeginger. On-line bin packing: a restricted survey. *Zeitschrift für Operations Research*, 42(1):25–45, 1995.

[78] C. Chu and R. La. Variable-sized bin packing: tight absolute worst-case performance ratios for four approximation algorithms. *SIAM Journal on Computing*, 30:2069–2083, 2001.

[79] S. Seiden, R. Van Stee, and Epstein L. New bounds for variable-sized online bin packing. *SIAM Journal on Computing*, 32(2):455–469, 2003.

[80] M.R. Garey, R.L. Graham, D.S. Johnson, and C. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory (A)*, 21(3):257–298, 1976.

[81] J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[82] A. Srivastav and P. Stangier. Tight approximations for resource constrained scheduling and bin packing. *Discrete applied mathematics*, 79(1):223–245, 1997.

[83] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[84] J.P. Stinson, E.W. Davis, and B.M. Khumawala. Multiple resource–constrained scheduling using branch and bound. *AIIE Transactions*, 10(3):252–259, 1978.

[85] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44(5):714–729, 1998.

[86] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-Completeness*. Freeman, San Fransisco, C.A., 1979.

[87] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[88] D. Bertsimas, J.N. Tsitsiklis, and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.