Lehigh University
Lehigh Preserve

Theses and Dissertations

2011

Stochastic Scheduling in Operating Rooms

Camilo Mancilla
Lehigh University

Follow this and additional works at: http://preserve.lehigh.edu/etd

Recommended Citation
Mancilla, Camilo, "Stochastic Scheduling in Operating Rooms" (2011). *Theses and Dissertations.* Paper 1326.

This Dissertation is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

STOCHASTIC SCHEDULING IN OPERATING ROOMS

by

Camilo Mancilla

Presented to the Graduate and Research Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy

in
Industrial Engineering

Lehigh University

September 2011

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____
Date

_____
Dr. Robert H. Storer
Dissertation Director

_____
Accepted Date

Committee Members:

_____
Dr. Robert H. Storer, Chairman

_____
Dr. Frank Curtis

_____
Dr. Brian Denton

_____
Dr. Jeffrey T. Linderoth

_____
Dr. Aurelie Thiele

Contents

Contents iii

List of Tables vi

List of Figures viii

Abstract 1

1 Introduction 3

1.1 Executive Summary 3

2 A sample average approximation approach to stochastic appointment sequencing and scheduling 7

2.1 Abstract 7

2.2 Introduction 8

2.3 Literature Review 9

2.4 Problem Statement 13

2.4.1 Problem Formulation 14

2.4.2 Properties of the formulation 15

2.4.3 Strengthening the MIP Formulation 16

2.5 Problem Complexity 17

2.6 Proposed Solution Methodology . . . 22
2.6.1 Proposed Algorithm . . . 23
2.6.2 The Master Problem in Benders' Decomposition . . . 25
2.6.3 Simplified Master Problem . . . 26
2.6.4 Restart Rules . . . 28
2.7 Accuracy of the Finite Sample Average Approximation . . . 31
2.8 Computational Experience for the Finite Scenario Case . . . 36
2.8.1 Comparison with Optimal Solutions . . . 37
2.8.2 Comparison with Benchmark Heuristics . . . 38
2.9 Conclusions . . . 44

3 Stochastic Sequencing of Surgeries for a Single Surgeon Operating in Parallel Operating Rooms 45

3.1 Introduction . . . 45
3.2 Literature Review . . . 47
3.3 Mathematical Formulation . . . 48
3.3.1 Problem Complexity . . . 51
3.4 Proposed Solution Approach . . . 52
3.4.1 Computation of the recourse function. . . . 55
3.4.2 Basic Parallel OR Scheduling Algorithm . . . 55
3.4.3 Alternative lower bounds . . . 56
3.4.4 A Heuristic for Computing an Upper Bound. . . . 58
3.4.5 Solving the surgery sequence sub-problems . . . 59
3.5 Computational Experience . . . 61
3.5.1 Comparison of stochastic solution versus common sense heuristic. . . . 63
3.5.2 Zig-Zag is not always optimal . . . 64
3.6 Choosing the number of scenarios . . . 65

3.7 Managerial Insight . . . 65
3.7.1 Estimating the cost Ratio . . . 69
3.8 Conclusions . . . 71

4 Stochastic integer program based algorithms for adaptable open block surgery scheduling 73

4.1 Introduction . . . 73
4.2 Problem definition . . . 76
4.2.1 Motivating Example . . . 77
4.2.2 Multiple Stage Integer Programming Formulation . . . 79
4.2.3 Approximation to the multi-stage stochastic problem . . . 81
4.3 Algorithm Summary . . . 90
4.4 Evaluating the solutions by simulation . . . 92
4.5 Computational Experience . . . 94
4.6 Conclusions . . . 100

5 Conclusions 101

5.1 Conclusions . . . 101

List of Tables

2.1 Optimality Gap Different Cost Case 38
2.2 Optimality Gap Equal Cost Case 38
2.3 Experiment Design 39
2.4 Average percent gap over the best solution found v/s scenarios (equal cost case) 40
2.5 Average percent gap over the best solution found v/s scenarios (different cost case) 41
2.6 Design for experiment 2 42
2.7 For memory algorithm over 1000 iterations (x - y) x: no overtime cost y: overtime cost 43
2.8 For memory algorithm over 1000 iterations (x - y) x: no overtime cost y: overtime cost 43

3.1 Experiment Design 62
3.2 Computational Results (average processing time in seconds) 63
3.3 Optimality gap for the mean value solution with zigzag OR Sequence. . . . 64
3.4 Optimality gap for the perturbed mean value solution with zigzag OR sequence. 64
3.5 Zig-zag OR Sequence comparison with optimal solution. 65

4.1 Experiment Design 87

4.2 Processing time in seconds 87

List of Figures

2.1 2 Scenario Case: ($a_{p,1}$ the first element in partition pt) 18
2.2 Best possible case if we increase $t^* + \delta$ 20
2.3 Optimal sequencing pattern. 21
2.4 Flowchart of the proposed algorithm. 23
2.5 Cumulative Frequency by Order Position 33
2.6 Percent Error Results 34
2.7 Confidence intervals from 50 scenarios instances 35
2.8 Confidence intervals from 500 scenarios instances 36
2.9 Interaction plots for experiment 1 42
2.10 Interaction plots for experiment 2 43

3.1 Illustration of Lower Bound on idle time for P_2 57
3.2 Illustration of Lower Bound on wait and over time for P_2 57
3.3 Tradeoff between number of scenarios and statistical gap. 66
3.4 Comparison between single and parallel ORs, cost ratio 1 68
3.5 Comparison between single and parallel, cost ratio 2 68
3.6 When to use parallel scheduling as a function of cost ratio and B parameter. 69
3.7 Upper and Lower bounds in the cost ratio based on historical decisions. . . 71

4.1 Small example. 78
4.2 Benchmark non emergency presence. 97

4.3 Benchmark non emergency presence, patient waiting cost 0.1. 98
4.4 Benchmark of different initial schedules in presence of emergencies, high patient cost. 98
4.5 Benchmark of different initial schedules in presence of emergencies, low patient cost. 98

Abstract

We investigate methods for sequencing and scheduling patients in health care settings such as operating rooms and clinics. Service times are highly variable in health care settings, thus accounting for this stochastic behavior in planning is crucial. Previous work exists on scheduling patient arrival times assuming a predefined sequence. Little research on sequencing exists due to its computational difficulty.

The first problem addressed involves sequencing and scheduling of surgeries in a single operating room (or more generally scheduling appointments in clinics) with the goal of minimizing (in expectation) patient waiting time, OR idle time, and staff overtime. The approach taken in this research is based on stochastic integer programming and sample average approximation. Algorithms based on Benders' decomposition combined with interior point algorithms for solving the LP scheduling sub-problem have proven to provide excellent results which show that significant additional efficiency is possible through proper sequencing.

The second research problem is sequencing a single surgeon in parallel ORs. While the surgeon is operating in one room, clean up and set up activities are being done in the other. The goal is a sequence and schedule that minimize (in expectation) the surgeon's idle time, OR staff idle time, and staff overtime in each OR. We again take a stochastic integer programming approach with sample average approximation. The solution requires

a specialized algorithm utilizing an additional decomposition step and clever bounds combined with cuts specifically developed for the application. Preliminary testing shows that realistic problems can be solved to optimality in most cases.

While the work above accounts for stochastic system behavior, it does not account for the ability of OR managers to take actions during the day to compensate for this random behavior. When planning for the day's activities what is needed is a "adaptable schedule" that performs well at the end of the day explicitly taking into account both the disruptions and the actions taken by OR managers during the day. We provide methods and also computational testing that shows the advantage of considering this dynamic behavior when constructing the initial schedule.

Chapter 1

Introduction

1.1 Executive Summary

Operating Rooms (OR's) account for about 40% of the total revenue in a hospital, and operate at 69% of their capacity according to HFMA [2005]. This translates into a significant opportunity for OR Management, both from the point of view of utilization, and the coordination of other resources inside the Hospital since the OR schedule drives many if not most Hospital activities.

This dissertation is devoted to investigating new methods and tools for solving realistic OR scheduling problems. The selection of problems proposed for study come from an exhaustive literature review and a partnership with a Local Hospital. This partnership helped ensure that the solutions will be feasible and practical to implement.

The first issue addressed is the creation of a methodology that helps the OR Manager define the sequencing and scheduling of surgeries in a single OR, (or within a surgical block), while explicitly accounting for the stochastic nature of the problem. The duration of surgical procedures is highly variable, thus accounting for randomness is crucial in OR scheduling. Thus we address the problem as a stochastic integer program.

The second problem addressed is the sequencing of one surgeon in parallel ORs and the

third problem is to investigate the "adaptable scheduling" problem. An adaptable schedule means a schedule that, in the presence of disturbances and subsequent rescheduling policies implemented to in response to the disturbances, still performs well. For the purposes of this study, it is assumed that disturbances arise in the surgery processing times and through the occurrence of emergency surgeries. In the following, each problem studied is described in more detail, followed by a discussion of the expected results of the proposed research.

The first problem is "A sample average approximation approach to stochastic appointment sequencing and scheduling". According to information that has been collected through interviews and available literature, OR Managers book a fixed amount of time for a particular surgery based on previous surgery times for the surgeon performing the operation. Typically there is a significant amount of data available on past surgery durations for each surgeon and for each type of procedure he/she performs. In current practice schedulers consider only the (perhaps trimmed) mean duration of the previous K surgeries for similar procedures performed by the same surgeon when building the schedule for the OR. Due to the uncertainty of surgery times, tools such as stochastic programming are indicated, as they allow the OR manager to account for the random nature of surgery times. The objective is to build schedules that minimize (in expectation) patient waiting time due to late surgeries, surgeon and OR staff idle time due to early finishes, and OR staff overtime.

This problem was formulated as an integer stochastic program using a sample average approximation. A heuristic solution approach based on Benders' decomposition was developed and then compared to exact methods and to previously proposed approaches. It was shown with extensive computational testing that the proposed methods produce good results. In addition it was proved that the finite scenario sample average approximation problem is NP-complete.

The second problem results from joint work with our industry partner where it is not

uncommon to book two operating rooms for a single surgeon when procedures are relatively short. While the surgeon is operating in one room, clean up and set up activities are being done in the other. The goal is a sequence and schedule that minimize (in expectation) the surgeon's idle time and staff overtime in each OR. We again take a stochastic integer programming approach with sample average approximation. The solution requires a specialized algorithm utilizing an additional decomposition step and clever bounds combined with cuts developed specifically for the application. Preliminary testing shows that realistic problems can be solved to optimality in most cases. We also note in passing that this problem is related to "open block scheduling" of surgical suites, and that techniques developed here might be useful for this problem as well.

The third problem, adaptable scheduling, is a multi-stage stochastic integer program. Adaptable scheduling means generating schedules that lose little in performance in the presence of disturbances and an associated "rescheduling" or control policy. The application of the proposed methodology was tested in a setting with multiple ORs and multiple surgeons. In this third study we formulated the adaptable scheduling problem as a Multi-Stage stochastic integer program. It is assumed that the control (i.e. rescheduling) actions will be taken by the OR manager as the schedule is executed to compensate for random events as they occur. The main control tool is the ability to shift a surgery to another OR, but each "move" requires a fixed cost. The starting times may not be moved earlier since patient arrival times are established in advance by the initial schedule. The goal of adaptable scheduling is to find schedules that perform well in uncertain environments given both a cost structure and rescheduling policy that specifies control actions.

The research problems that have been introduced are of significant relevance. From an operational and tactical perspective, operating rooms are one of the main drivers of hospital activity and therefore improved efficiencies will have a large, system-wide impact. Furthermore, greater efficiency will increase elective capacity and therefore have a significant impact on costs related to capacity investment and strategic planning. Elective

management of capacity will become even more relevant as the aging population in the U.S. increases pressure on an already stressed system.

From an academic point of view, these problems involve cutting-edge research in Operations Research applied to Health Care Management. The methodologies to be developed may also be applied to similar problems faced in the health care system (e.g. primary care clinics, outpatient procedure centers, etc.) and also other areas like project scheduling, container vessel and terminal operations, airport gate and runway schedules among others thereby extending their impact.

Chapter 2

A sample average approximation approach to stochastic appointment sequencing and scheduling

2.1 Abstract

We develop algorithms for a single resource stochastic appointment sequencing and scheduling problem with waiting time, idle time, and overtime costs. This is a basic stochastic scheduling problem that has been studied in various forms by several previous authors. Applications for this problem cited previously include scheduling of surgeries in an operating room, scheduling of appointments in a clinic, scheduling ships in a port and scheduling exams in an examination facility (see section 2.3). In this chapter the problem is formulated as a stochastic integer program using sample average approximation. A heuristic solution approach based on Benders' decomposition is developed and compared to exact methods

and to previously proposed approaches. Extensive computational testing shows that the proposed methods produce good results compared to previous approaches. In addition we prove that the finite scenario sample average approximation problem is NP-complete.

2.2 Introduction

The problem we address assumes a finite set of jobs with stochastic processing times. It is assumed that the processing time duration of the jobs are random variables with known joint distribution. The marginal distributions of job durations are not assumed identical. The problem requires us to find the sequence in which to perform the jobs, and to assign a starting time to each job. A job may not begin before its scheduled starting time nor may it begin until the previous job is complete. If the i^{th} job in the sequence finishes before the $i+1^{st}$ job is scheduled to start, then there will be idle time on the resource. Conversely, if the i^{th} job in the sequence finishes after the i+1st job is scheduled to start, then job i+1 will incur waiting time. Further, if the last job finishes after a predefined deadline, there will be overtime. The objective is to determine the sequence and scheduled starting times of jobs on the resource that minimize a weighted linear combination of job waiting time, resource idle time, and overtime. We assume a separate cost (per unit time) for each job for both waiting and idle times, and a single overtime cost. Since we explicitly consider the randomness of the job processing times, the objective is to minimize total expected cost where expectation is taken with respect to the joint distribution of processing times. This problem has been called the "appointment scheduling problem" because it is easy to envision by analogy to scheduling appointments in a physician's office. Jobs represent patient appointments, while the doctor represents the resource. Waiting time is the time patients must wait beyond their scheduled appointment time while idle time represents time the doctor is not busy while waiting for the next patient to arrive. The problem can be decomposed into two parts. The first is to determine the sequence in which the

jobs will be performed. Given a sequence one must next determine the amount of time to allocate to each job, or equivalently assign each job a scheduled starting time. This second problem (which we call the scheduling problem) has been studied previously under the name "stochastic appointment scheduling". Previous approaches to this problem include using convolutions to compute starting times Weiss [1990], sample average approximation and the L-Shape Algorithm Denton and Gupta [2003], and heuristics Robinson and Chen. [2003]. We will approach this problem using sample average approximation and linear programming in a similar fashion to Denton et al. [2007].

Given reasonably efficient methods to solve the scheduling problem, we next develop an algorithm for the sequencing problem. According to Gupta [2007] the sequencing problem is still an open question. The main idea of the proposed method is based on a Benders' decomposition scheme. The master problem is used to find sequences and the sub-problems are the scheduling problems (stochastic linear programs) as discussed above. The Benders' master problem becomes extremely hard to solve as cuts are added, thus we turn to heuristics to approximate its solution and generate promising sequences. The remainder of the article is organized as follows. In the next section a brief review of the literature related to the sequencing and scheduling problems and stochastic integer programming is provided. In Section 2.4, the model is described and formulated. In Section 2.5 new complexity results for the sequencing problem are presented. In Section 2.6 algorithms are proposed to solve the sequencing and scheduling problems. In Section 2.7 a method is proposed to choose the number of scenarios. In Section 2.8 computational results are presented. In Section 2.9 conclusions and future research directions are discussed.

2.3 Literature Review

Relevant previous work can be divided in two categories: stochastic appointment scheduling and stochastic integer programming. We begin with previous work on the stochastic

appointment scheduling problem. Weiss [1990] found the optimal sequence when there are only 2 jobs (i.e. convex order) and then showed that this criterion does not guarantee optimality when the number of jobs is greater than 2. Wang [1997] assumed job durations were i.i.d. random variables following a Coxian (phase type) distribution. Since durations were assumed i.i.d., and patient waiting costs were assumed equal across patients, sequencing was irrelevant. He assumed costs for waiting time and total completion time. He developed an efficient numerical procedure to calculate mean job flow times then solved for the optimal scheduled starting times using non-linear programming. For examples with up to 10 jobs, he showed that even though job durations are i.i.d., the optimal starting times are not equally spaced. (Denton, 2007) formulated the sequencing and scheduling problem as a stochastic integer program and then proposed simple heuristics to determine the sequence. Once a sequence is given, the schedule of starting times was found using a sample average approximation (i.e. scenario based) approach. The resulting scheduling problem was shown to be a linear stochastic program which they solved by an L-shaped algorithm described in Denton and Gupta [2003]. To determine a sequence they proposed three methods: sort by variance of duration, sort by mean of duration, and sort by coefficient of variation of duration. These simple heuristics were also compared to a simple interchange heuristic. The application studied in Denton and Gupta [2003] was scheduling surgeries in a single operating room. They reported results with real surgery time data and up to 12 surgeries (jobs). They also assumed equal penalty costs across surgeries for waiting and idle time. They found that sort by variance of duration gave the best results. Kaandorp and Koole [2007] assumed that job durations were exponentially distributed with different means and that patient arrivals can only be scheduled at finite times (every ten minutes). Their objective function included waiting time, idle time, and overtime costs. Given these assumptions, a queuing theory approach was used to calculate the objective function for a given schedule of starting times. They further proved that the objective function was multi-modular with respect to a neighborhood that can move the

start time of jobs one interval earlier or later. This result guaranteed that a local search algorithm in this neighborhood will find the optimal solution.

In Vanden Bosch and Dietz [2000] and Vanden Bosch and Dietz [2001] the authors also assumed discrete scheduled starting times (at 10 minute intervals over 3 hours) and included penalties for waiting time and overtime. They assumed three classes of patients in an outpatient appointment scheduling setting where durations were i.i.d. within class but different between classes. They used Phase Type and Lognormal distributions to model the three duration distributions. Given a sequence, they proposed a gradient based algorithm to find the optimal schedule of starting times based on submodularity properties of the objective function. They proposed an all-pairs swap-based steepest decent local search heuristic to find a sequence. They stopped the search after a fixed number of iterations or when a local minimum is found. They reported testing with simulated data for cases with 4 and 6 jobs and concluded that the heuristics produced good results in terms of iterations and optimality gap when compared with exhaustive enumeration. In Kong et al. [2010] the authors developed a robust optimization approach to the appointment scheduling problem. They assumed that the distributions of the services were unknown, and minimized the worst case expected value over a family of distributions to determine the schedule. The approach to the problem taken in this paper is stochastic integer programming, thus previous approaches to similar problems are relevant. There is a rich literature on stochastic integer programming. In Schultz [2003] a thorough review of methods for solving stochastic programming problems with integer variables was given. For the type of problem we address, there are several methods that might seem to apply. Our problem has both integer variables (for sequencing) and continuous variables (for scheduled starting times) in the first stage, continuous variables in the recourse function (waiting and idle times), and complete recourse. For finite scenario problems, Laporte and Louveaux [1993] proposed the Integer L-Shaped Method, an algorithm that is suitable for solving Stochastic Programs where the first stage variables are binary and the recourse cost

is easily computable. The method uses Benders' Decomposition combined with cuts that differ from traditional Benders' cuts. Another approach based on scenario decomposition was proposed in Caroe and Schultz [1997]. After decomposing the problem by scenarios, they then solved a problem with relaxed non-anticipativity constraints to get a lower bound within a branch and bound scheme. Finally there is the widely used Benders' decomposition approach. In Benders' approach one may decompose by fixing the integer variables or by fixing the set of all first stage decisions. The problem we address is such that if the integer (sequencing) variables are fixed, the continuous (scheduling) variables can be computed with relative ease by solving a linear program using an interior point method. This makes Benders' decomposition particularly attractive for our problem. We also experimented briefly with the Integer L-shaped method but found that it offered no advantage over the Benders' approach in this case. We performed two types of experiments in order to investigate the performance of Integer L-Shaped method:

Type 1: The first type of experiment involved a small number (5) of surgeries. The total number of integer feasible solutions for a problem with 5 surgeries is 120. We observed that straight forward Benders' decomposition needed 105 cuts to find the optimal solution and similar results we found when we applied Integer L-Shaped, thus no benefit over Benders' was observed.

Type 2: The second type of experiment involved a larger number (10) of surgeries. The total number of integer feasible solutions for this problem is 3628800. When we applied Benders decomposition after 1000 iterations the gap still was 70%. The Integer L-Shaped produced similar results. These experiments showed us that we would need a great deal (indeed an unrealistically large) amount of processing time if we were planning to solve this problem using either Benders' or the Integer L-Shaped method.

Scenario decomposition is not appropriate for our problem since solutions to the single scenario problems provide no useful information about the overall solution. This is because in any scenario subproblem, the starting times are simply set equal to the finish time of the

previous job always resulting in zero cost. The current literature also distinguishes between infinite scenario problems and finite scenario problems. For the infinite scenario case, two methodologies are potentially useful. In Kleywegt et al. [2002] the authors proposed an algorithm for solving the sample average approximation (finite scenario problem). They applied this procedure many times until stopping criteria related to statistical bounds are fulfilled. The other method aimed at the infinite scenario case is Stochastic Branch and Bound (Norkin, 1998). This method partitions the integer feasible space and computes statistical upper and lower bounds, then uses these bounds in the same way traditional branch and bound uses upper and lower bounds to find the true optimal value with probability one. Solving our sample average approximation problem turns out to be very time consuming, thus neither of these infinite scenario approaches are practically viable in our case.

2.4 Problem Statement

We assume a finite set of jobs with durations that are random variables. We assume these job durations have a known distribution, and are independent of the position in the sequence to which the job is assigned. Only one job may be performed at a time, and overtime is incurred when job processing extends past a deadline representing the length of the work day. Two sets of decisions must be made, first the job sequence must be determined, then a starting time must be assigned to each job. In application to surgery scheduling, the starting time can be thought of as the time the patient is scheduled to arrive thus a job (surgery) may not begin before its scheduled starting time. The objective function consists of three components, waiting time (the time a patient must wait between his/her scheduled starting time and actual starting time), idle time (the time the O.R. is idle while waiting for the next patient to arrive) and overtime. Given a sequence, starting times for each job, and the duration distributions, the expected waiting time and idle

time before each job and the over time can be estimated by averaging over a sample of scenarios. The objective function is a weighted linear combination of these three expected costs. Note that waiting and idle costs may be different for each job. This problem has been modeled as a two stage stochastic program with binary and continuous variables in the first stage decisions in Denton et al. [2007]. They incorporated the processing time uncertainty into the model using a sample average approximation (i.e. scenario based) approach. Our model is quite similar to Denton et al. [2007], but has two differences. Our binary variables (x_{ij}) define which job (surgery) should be placed in the i^{th} position in the sequence, and we define a scheduled starting time for each job. Denton's model used binary variables that defined precedence and continuous variables that defined the amount of time allocated to each job. The way the binary variables are defined in our formulation will later be useful in developing an approach to solving the master problem in Benders' decomposition. The starting times and the binary variables are all included in the first stage decisions. Our formulation is shown below.

2.4.1 Problem Formulation

$$\text{minimize: } \sum_{k=1}^{K} \frac{1}{K} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} c_j^w w_{i,j}^k + c_j^s s_{i,j}^k + c^l l^k \right) \tag{2.1}$$

$$\text{s.t.: } t_i - t_{i+1} - \sum_{j=1}^{n} w_{i+1,j}^k + \sum_{j=1}^{n} s_{i,j}^k + \sum_{j=1}^{n} w_{i,j}^k = -\sum_{j=1}^{n} \xi_j^k x_{ij} \qquad i < n, k \tag{2.2}$$

$$t_n + \sum_{j=1}^{n} w_{n,j}^k - l^k + g^k = -\sum_{j=1}^{n} \xi_j^k x_{nj} + d \qquad k \tag{2.3}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j \tag{2.4}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i \tag{2.5}$$

$$s_{i,j}^k \leq M_1 x_{ij} \qquad i, j, k \tag{2.6}$$

$$w_{i,j}^k \leq M_2^i x_{ij} \qquad i, j, k \tag{2.7}$$

$$w_{i,j}^k \geq 0 \ , s_{i,j}^k \geq 0 \ \forall (i,j,k) \in (I,J,K) \ l^k \geq 0, \ g^k \geq 0 \ \forall k \in K, \ x_{ij} \in \{0,1\}$$

Indices and Sets

J Jobs to be scheduled j=1,...,n.

I Positions in the sequence i=1,...,n.

K Scenarios to be considered k=1,....,K.

Parameters

c_j^w waiting time penalty for surgery j.

c_j^s idle time penalty for surgery j.

c^l overtime penalty.

d time beyond which overtime is incurred.

M_1, M_2^i are sufficiently large numbers.

ξ_j^k duration of surgery j in scenario k.

Variables

t_i scheduled starting time for the surgery in position i.

w_{ij}^k waiting time in scenario k when surgery j is in position i.

s_{ij}^k idle time in scenario k when surgery j is in position i.

l^k total time to complete all jobs (makespan) in scenario k.

g^k slack variable that measures the earliness with respect to time d.

x_{ij} a binary variable denoting the assignment of surgery j to position i.

Constraints

(2.2),(2.3) define the waiting and idle time for every surgery and scenario.

(2.4),(2.5) assure each surgery is assigned to one position in the sequence.

(2.6),(2.7) are logical constraints that force waiting and idle times to be zero.

2.4.2 Properties of the formulation

Once the sequence is fixed the waiting and idle times and tardiness can be computed for every scenario as a function of the scheduled starting times as shown below (again similar

to Denton et al. [2007]). In the stochastic programming framework the computation of these variables can be seen as the recourse function. Further, any sequence will yield a finite objective function value therefore we have complete recourse. Thus we will only need optimality cuts to solve this problem using Benders' decomposition.

$$w_i^k = \max\left\{w_{i-1}^k + \xi_{i-1}^k - t_i + t_{i-1}, 0\right\}, \quad i = \{2, ..., n\} \tag{2.8}$$

$$s_i^k = \max\left\{-w_i^k - \xi_i^k - t_i + t_{i+1}, 0\right\}, \quad i = \{1, ..., n-1\} \tag{2.9}$$

$$l^k = \max\{w_n^k + \xi_n^k + t_n - d, 0\} \tag{2.10}$$

The waiting time and scheduled starting time of the first job in the sequence are both assumed to be zero. Also waiting time w_i^k is the time the i^{th} surgery has to wait (and thus occurs before the i^{th} surgery). s_i^k is the idle time that occurs after the i^{th} surgery.

2.4.3 Strengthening the MIP Formulation

If we apply traditional branch and bound to the problem formulated in section 2.4, the weakness of the formulation caused by the big M constraints will negatively affect performance. Since we will compare the performance of our algorithms to branch and bound, it is important to strengthen the MIP formulation to the extent possible. Through straightforward but lengthy analysis we found finite values for M_1 and M_i^2 that preserve optimality. Proof of the validity of the results may be found in Mancilla and Storer [2009]. Here we simply state the results. We set M_1 (which appears in the slack time constraints (2.6)) as follows:

$$M_1 = \max_{i \in I}\{max_{k \in K}\{\xi_i^k\} - min_{k \in K}\{\xi_i^k\}\}$$

We set the M_2^i values that appear in the waiting time constraints (2.7) as follows:

$$M_2^i = \sum_{j=1}^{i-1} \delta_j$$

where δ_j corresponds to the j^{th} largest value in terms of $\max_{k \in K} \xi_r^k - \min_{k \in K} \xi_r^k$. With these "big M" values, the formulation can be somewhat tightened.

2.5 Problem Complexity

Previous authors Denton et al. [2007], Gupta [2007] have speculated that the sample average approximation sequencing and scheduling problem (SAA-SSP) is NP-Complete, but to the best of our knowledge the question is still open. In this section we prove SAA-SSP with two scenarios and with equal idle cost but different waiting costs for each job is NP-Complete. The proof uses concepts similar to those in Garey et al. [1976].

Definition of the feasibility version of the SAA-SSP Given a collection of jobs I indexed by i, with durations in scenario k given by ξ_i^k, and a budget B of schedule cost, does there exist a sequence and scheduled starting time for each job in I whose cost does not exceed B ? We construct a polynomial transformation to the 3-Partition problem to show our problem is NP-Complete. The 3-Partition problem as defined below is known to be NP-Complete Garey et al. [1976].

Definition 3-Partition Garey et al. [1976]:

Definition 1. *3-Partition: Given positive integers n, R, and a set of integers* $A = \{a_1, a_2, ..., a_{3n}\}$ *with* $\sum_{i=1}^{3n} a_i = nR$ *and* $\frac{R}{4} < a_i < \frac{R}{2}$ *for* $1 \leq i \leq 3n$*, does there exist a partition* $\langle A_1, A_2, ., ...A_n \rangle$ *of A into 3-elements sets such that, for each i* $\sum_{a \in A_i} a = R$ *?*

Theorem 1. *The Sample Average Approximation Sequencing and Scheduling Problem (SAA-SSP) with two scenarios is NP-Complete when the waiting costs are allowed to differ between jobs.*

Construction

We show that 3-partition polynomially reduces to a particular SAA-SSP with 2 scenarios. We construct an instance of SAA-SSP with 4n jobs and 2 scenarios in which the first 3n jobs (job set "G" indexed by i=1,,3n) have durations $\xi_i^1 = a_i$ and $\xi_i^2 = 0$. The remaining n jobs (job set "V" indexed by i=3n+1, , 4n) have durations $\xi_i^1 = H$, and $\xi_i^2 = H + R$ for $3n + 1 \leq i \leq 4n$. (H is an arbitrary integer) The idle cost penalties are chosen as $c_i^s = 10KRn^2$ for all 4n jobs. The waiting cost for jobs in V are $c_i^w = \frac{5nR}{2}$ for i=3n+1,...,4n. The waiting cost for jobs in set G are $c_i^w = 1$ for i=1,...,3n. The budget of the schedule cost is $B = \frac{5nR}{4}$.

Intuitively, the main idea of the proof can be seen in Figure 1. The schedule consists of n blocks with each block containing one of the partitions in scenario 1. The idle cost is set high enough to guarantee that there will be no idle time in an optimal solution to the SAA-SSP. The scheduled starting time for every job is set equal to its actual starting time in scenario 1. We first show that if a 3-partition exists, the schedule below will meet the budget B. We then show that if the schedule is not the "3-Partition schedule" as shown in Figure 1, or if no partition exists, then the budget to be exceeded.

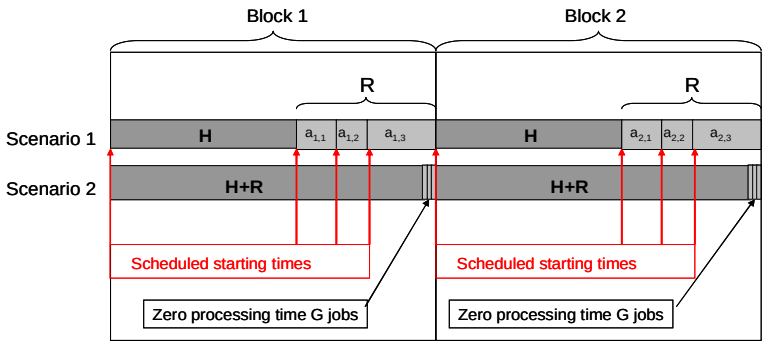

Figure 2.1: 2 Scenario Case: ($a_{p,1}$ the first element in partition pt)

Proof. We start by showing that if a solution to the 3-Partition problem exists, then it leads to a feasible solution of SAA-SSP that meets the budget B. Assume there is a 3-partition A1, A2, ...,An of A that fulfills the condition $A_p = \{a_{p,1}, a_{p,2}, a_{p,3}\}$ and , $\sum_{\alpha=1}^{3} a_{p,\alpha} = R$,$1 \leq p \leq 3$. From this partition we construct the sequence and schedule as shown in Figure 1. Recall that in scenario 2 the G jobs all have zero duration. The cost Z^* for the schedule in Figure 1 can be shown to be bounded above by the budget B $=\frac{5nR}{4}$, as follows:

$$
\begin{aligned}
Z^* &= \tfrac{1}{2}\left(\sum_{i=1}^{3n} c_i^w w_i^1 + c_i^s s_i^1 + \sum_{i=3n+1}^{4n} c_i^w w_i^1 + c_i^s s_i^1\right) + \tfrac{1}{2}\left(\sum_{i=1}^{3n} c_i^w w_i^2 + c_i^s s_i^2 + \sum_{i=3n+1}^{4n} c_i^w w_i^2 + c_i^s s_i^2\right) \\
&= \tfrac{1}{2}\left(\sum_{i=1}^{3n} c_i^w w_i^1 + \sum_{i=3n+1}^{4n} c_i^w w_i^1\right) + \tfrac{1}{2}\left(\sum_{i=1}^{3n} c_i^w w_i^2 + + \sum_{i=3n+1}^{4n} c_i^w w_i^2\right) \text{ all idle times} = 0 \\
&= \tfrac{1}{2}\left(\sum_{i=1}^{3n} c_i^w w_i^2 + \sum_{i=3n+1}^{4n} c_i^w w_i^2\right) \text{ all wait times} = 0 \text{ in scenario 1} \\
&= \tfrac{1}{2}\left(\sum_{i=1}^{3n} c_i^w w_i^2 + \sum_{i=3n+1}^{4n} \tfrac{5nR}{4}\right) \text{ all G wait costs} = 1, \text{ V wait times} = 0 \\
&= \tfrac{1}{2}\left(\sum_{p=1}^{n} (a_{p,1} + a_{p,2} + a_{p,3}) + (a_{p,2} + a_{p,3}) + a_{p,3}\right) \text{ wait times of G jobs in scenario 2} \\
&= \tfrac{1}{2}\left(nR + \sum_{p=1}^{n} (a_{p,2} + a_{p,3}) + a_{p,3}\right) \text{ schedule constructed from perfect partition} \\
&\leq \tfrac{1}{2}\left(nR + \sum_{p=1}^{n} (2\tfrac{R}{2} + \tfrac{R}{2})\right) \text{ upper bound on } a_{p,\alpha}
\end{aligned}
$$

Lemma 2.5.1. *For any sequence of jobs, the optimal scheduled starting times for the set of 4n jobs with idle cost penalties* $c_i^s = 10n^2KR$ *(K is the number of scenarios), wait cost penalties* c_i^w*=1* $\forall i \in \{1, .., 3n\}$ *and* $c_i^w = \frac{5nR}{2}$ *for* $i = 3n+1, ..4n$ *is given by:* $t_i^* = \min_{k \in K}\{t_{i-1}^* + w_{i-1}^k + \xi_{i-1}^k\}$.

This lemma basically says that if the idle cost is set high enough, the optimal schedule will be the one that contains no idle time. This is equivalent to setting the scheduled start time of the i^th job in the sequence equal to the earliest finish time in any scenario of the $i-1^{st}$ job in the sequence.

Proof. We will prove this lemma by contradiction. Case 1: Assume that ∃ a job i such that $t^* < \min_{k\in K}\{t^*_{i-1} + w^k_{i-1} + \xi^k_{i-1}\}$ in the optimal schedule. This means that job i has positive waiting time in every scenario (i.e. $w^k_i > 0 \forall k \in K$). This contradicts the assumed fact that this is an optimal solution since we can increase t^* until some w^k_i becomes zero without affecting the waiting time of the other jobs. This implies that we were in a suboptimal solution.

Case 2: Assume that ∃ a job i such that $t^* > \min_{k\in K}\{t^*_{i-1} + w^k_{i-1} + \xi^k_{i-1}\} + \delta$ in the optimal schedule and Z* is the optimal objective function value. This means that we have idle time of length δ for at least one scenario in job i (see Figure 2.2). This increases the objective function by δc^s_i . The best possible case is that the waiting time of all future jobs in all other scenarios is reduced by δ (see Figure 2.2) thus reducing the objective function by $(K-1)\delta \sum_{j=i}^{4} nc^w_j$. We have:

$$(K-1)\delta \sum_{j=i}^{4} nc^w_j < (K-1)\delta \sum_{j=1}^{4} nc^w_j < (4n-1)(K-1)\delta \max_i c^w_i = (4n-1)(K-1)\delta \frac{5nR}{2}$$

Thus if we set $c^s_i = (4n-1)(K-1)\frac{5nR}{2}$ the optimal schedule will contain no idle time.

□

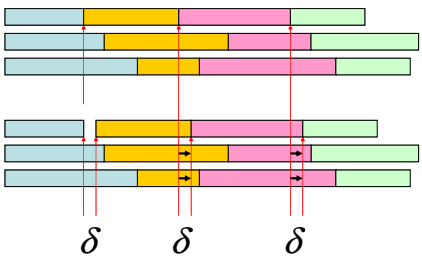

Figure 2.2: Best possible case if we increase $t^* + \delta$

Lemma 2.5.2. *For any given sequence of problem SAA-SSP, the optimal schedule leads to a solution that has an integer valued objective function.*

Proof. By lemma 1 we know that the scheduled starting times follow a recursive formula that depends on waiting time and durations of jobs earlier in the sequence: $t_1^* = 0$, $t_i^* = \min_{k \in K}\{t_{i-1}^* + w_{i-1}^k + \xi_{i-1}^k\}$. Further we know that we can express waiting time of the ith job in the sequence using: $w_1^k = 0, w_i^k = \sum_{j=1}^{i-1} \xi_j^k - t_i$. Since the durations are integers, and the integers are closed under addition and subtraction, we can conclude that all waiting times are integer (and all idle times are zero). Since waiting costs are also integer, the objective function value of the optimal solution to the scheduling problem also be an integer.

□

Lemma 2.5.3. *The actual starting times of jobs in set V must be the same in each scenario otherwise the budget will be exceeded.*

Proof. Since no idle time can exist (Lemma 2.5.1), the schedule for scenario 2 must appear as below. If the V jobs in scenario 1 do not start at the same time as in scenario 2, there will be non-zero waiting time for at least 1 V job. The waiting time must be at least 1 (from the integer lemma 2.5.2), and since waiting cost for jobs in V is 5nR/2, the budget is exceeded Thus the schedule for V jobs must look like this

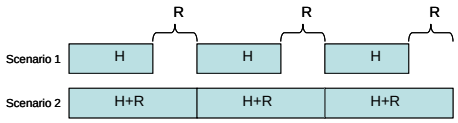

Figure 2.3: Optimal sequencing pattern.

□

Lemma 2.5.4. *There must be subsets of G jobs that fit perfectly into the first n-1 open slots (of width R) in the schedule above if there exists a perfect partition. These subsets*

consist of 3 G jobs and correspond to the perfect partition.

Proof. If these subsets of G jobs do not exist there will be idle time in the schedule, and the budget will be exceeded (by Lemma 2.5.1). Further, these subsets must consist of 3 G jobs due to the bounds $\frac{R}{4} < a_i < \frac{R}{2}$, otherwise the subsets will add to more or less than R. The only possible solution for these subsets is the perfect partition and by definition of the partition problem, the remaining 3 jobs must fit perfectly into the last open slot. Lemma 4 completes the proof that the SAA-SSP with two scenarios is NP-complete when the waiting costs are allowed to differ between jobs. □

□

2.6 Proposed Solution Methodology

The approach we propose for the SAA-SSP uses a heuristic method to find good solutions in a reasonable amount of time. The Master problem in our Benders' Decomposition is an integer program, and thus is difficult to solve and even more difficult as we add more cuts in every iteration. Relaxing the side constraints (optimality Benders' cuts) results in an easy to solve assignment problem (please see section 2.6). We use this property to construct a heuristic to generate good feasible solutions to the master problem. The idea of solving the master problem heuristically has also been proposed by several authors including Cote and Laughton [1984] and Aardal and Larsson [1990].

2.6.1 Proposed Algorithm

The basic outline of our algorithm is as follows; start with an arbitrary sequence. Set the Upper Bound (UB)=∞ and Lower Bound (LB)=-∞.

1. Start with an arbitrary sequence (x_o).
2. Set the Upper Bound (UB)=∞ and Lower Bound (LB)=-∞.
3. Solve the LP scheduling subproblem for the current sequence yielding z^*, x^*, p^*.
4. Update the UB if z^* <UB.
5. Generate a Benders' cut from the optimal extreme point in the dual (p^*).
6. Use our simplified master problem heuristic to try to find a feasible solution to MP.
7. If the new sequence is the same as the previous sequence, restart the heuristic.
8. If (number of iterations < max iterations) go to 3, else stop.

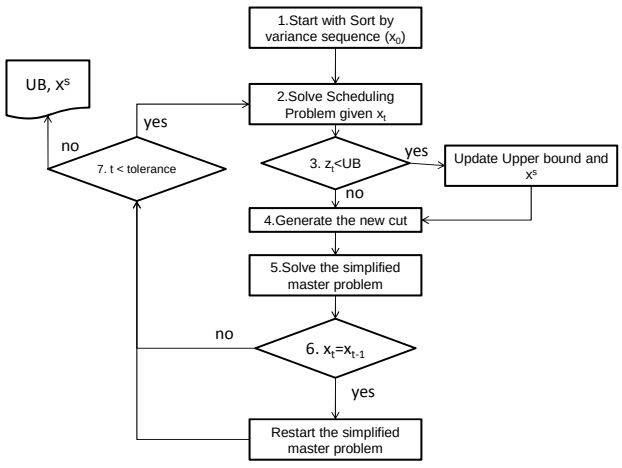

Figure 2.4: Flowchart of the proposed algorithm.

The LP Scheduling Subproblem

Given a sequence, finding the optimal scheduled starting times is known to be a linear program (Denton and Gupta [2003],Denton et al. [2007]). This is the LP subproblem in step one above. The size of this LP grows rapidly with the number of scenarios. Denton and Gupta [2003] proposed solving this problem using the L-Shaped method. The dual of the problem is very close to a network flow problem. We were able to create an algorithm to solve this problem that was similar to a network simplex algorithm. The main idea of this algorithm was to find feasible points by creating a residual network where we find the maximum number of units by which we could modify the current flow without making the solution infeasible. We performed thorough testing that showed us the advantage of interior point method. We generated different instances by changing the number of scenarios (100 to 2700 every 100 scenarios) and cost coefficients. However, further testing revealed that interior point methods were at least one order of magnitude faster than any of the other methods on problems with up to 10,000 scenarios. Thus, we use the Cplex barrier algorithm to solve the subproblems within our algorithm. The LP subproblem which, for a given sequence, finds scheduled starting times that minimize total cost averaged over the scenarios is given below.

$$\min : \sum_{k=1}^{K} \frac{1}{K} \left(\sum_{i=1}^{n} c_i^w w_i^k + c_i^s s_i^k + c^l l^k \right)$$

$$\text{s.t.: } t_i - t_{i+1} - w_{i+1}^k + s_i^k + w_i^k = -\xi_i^k \quad \forall i \in I \ \forall k \in K \quad (2.11)$$

$$t_n + w_n^k - l^k + g^k = d - \xi_n^k \quad \forall k \in K \quad (2.12)$$

$$w_i^k, s_i^k \geq 0 \ \forall (i,k) \in (I,K), \ l^k, g^k \geq 0 \ \forall k \in K, \ t_1 \geq 0$$

This is the dual of the scheduling problem,

$$\max : -\sum_{k=1}^{K}\sum_{i=1}^{N} f_i^k \xi_i^k - \sum_{k=1}^{K} f_n^k (d - \xi_n^k)$$

$$\begin{aligned}
\text{s.t.} \quad & \textstyle\sum_{k\in K} f_1^k \leq 0 \\
& \textstyle\sum_{k\in K} f_i^k - \sum_{k\in K} f_{i+1}^k = 0 && \forall i \in I \\
& f_{i+1}^k - f_i^k \leq \tfrac{1}{K} c_i^w && \forall k \in K \\
& f_i^k \leq \tfrac{1}{K} c_i^s && \forall i \in \{1, .., n-1\} \\
& -\tfrac{1}{K} c^l \leq f_n^k \leq 0 && \forall k \in K.
\end{aligned}$$

2.6.2 The Master Problem in Benders' Decomposition

We use a straightforward application of Benders' Decomposition on the mixed integer stochastic program defined in section 3. The master problem contains the integer sequencing variables (x_{ij}). The master problem formulation at iteration T appears below.

$$\begin{aligned}
& \text{minimize: } \theta_M \\
\text{s.t.:} \quad & \textstyle\sum_{i=1}^{n} x_{ij} = 1 && \forall j \in \{1..n\} \\
& \textstyle\sum_{j=1}^{n} x_{ij} = 1 && \forall i \in \{1..n\} \\
& \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n} u_{ij}^t x_{ij} + d\sum_{k=1}^{K} f_n^k < \theta_M && t \in \{1, .., T\} \quad (2.13) \\
& \theta_M < UB_T \\
& x_{ij} \in \{1, 0\}
\end{aligned}$$

Where

$$x_{ij} = \begin{cases} 1 & \text{if we assign the surgery j to position i in the sequence} \\ 0 & \text{Otherwise} \end{cases}$$

$$u_{ij}^t = \begin{cases} -\sum_{k=1}^{K} \xi_j^k f_i^k & \text{if } x_{ij}=1 \\ -\sum_{k=1}^{K} \xi_j^k f_i^k - M_1 \sum_{k=1}^{K} p_{ij}^k - \sum_{k=1}^{K} M_2^i q_{ij}^k & \text{Otherwise} \end{cases}$$

The constraints (2.13) are the Benders' optimality cuts. The coefficients of these constraints come from the inner product between the dual variables (f_i^k , p and q variables corresponding to the dual variables of the constraints (2.11) and (2.12)) and the job durations and big M's. These cuts contain information found from the solutions of the LP subproblems and allow us to implicitly eliminate sequences that will not improve the objective function.

To assure convergence of the iterative Benders' solution approach, all that is needed is a feasible integer solution to the master problem at each iteration. However, since this problem is computationally hard to solve, we construct a simplified version that is easy to solve, but does not guarantee a feasible solution to the master problem.

2.6.3 Simplified Master Problem

We remove the side constraints (Benders' cuts) and create a new objective function based on the coefficients of these constraints. The resulting simplified master problem (SMP) is an assignment problem and thus can be easily solved (but is not guaranteed to be a feasible solution to the original master problem).

$$\text{minimize: } \theta_{SMP} = \sum_{i=1}^{n} \sum_{j=1}^{n} \overline{ba}_{ij} x_{ij}$$

$$\begin{aligned} \text{s.t.:} \quad & \textstyle\sum_{i=1}^{n} x_{ij} = 1 \ \forall j \in \{1..n\} \\ & \textstyle\sum_{j=1}^{n} x_{ij} = 1 \ \forall i \in \{1..n\} \\ & 0 \leq x_{ij} \leq 1 \end{aligned}$$

The idea is to capture the information in the cut constraints in such a way that we can find a feasible solution to the master problem with reasonable reliability. We experimented

with several methods for constructing these objective function coefficients $\overline{ba}_{ij}$. Based on this experimentation the best method was found to be $\overline{ba}_{ij} = \max_{t\in\{1..T\}}\{ba^t_{ij}\}$ where the ba^t_{ij} come from a modified Bender's optimality cut coefficients. Specifically, we construct ba^t_{ij} based on Benders' cut but with a slightly modified dual vector. We set to zero the p_{ij} and q_{ij} dual variables. These dual variables correspond to constraints (2.11) and (2.12) in the original formulation. Setting the p_{ij} and q_{ij} dual variables equal to zero is valid when the costs are equal, but does not guarantee valid cuts in the unequal cost case. However, retaining these dual variables adds unnecessary noise to the information contained in the dual in the important constraints (2.2) and (2.3). We found that this method produces better results, and since the proposed algorithm is heuristic, we adopt it. For a particular iteration t the ba vector is generated as follows:

$$ba_{ij} = \begin{cases} -\sum_{k=1}^{K} \xi_j^k f_i^k & \text{i<n} \\ -\sum_{k=1}^{K} (\xi_j^k - d) f_i^k & \text{i=n} \end{cases}$$

The desirability of using the max operator to aggregate the constraints is supported by the following proposition.

Proposition 2.6.1. *The optimal solution to the SMP is an upper bound of the master problem in Benders' Decomposition. Furthermore, if the optimal objective function value of the SMP is less than the current upper bound, the optimal solution is a feasible solution to the master problem.*

Proof 1. *In the first iteration (i.e. with one side constraint), it is clear that the simplified problem finds a feasible solution to the master problem. In subsequent iterations with more than one side constraint the master problem can be defined as follows,*

$$\min_{\rho\in S} \max_{t\in T} \{F_t(\rho)\}$$

Where S is the set of sequences and F_t is the left hand side in constraint t for sequence

ρ in the Master Problem. Given a sequence ρ, exactly n x_{ij}'s will be equal to 1. Let $ba^t_{l_\rho}$ l_ρ=1,..,n be the coefficients in constraint t for the x_{ij}'s that are equal to 1 under sequence ρ.

$$F_t : S \mapsto \mathbb{R}$$
$$F_t(\rho) = \sum_{l_\rho=1}^{n} ba^t_{l_\rho}$$

Thus the Master Problem can be written as the summation of coefficients, and in this form we can see that

$$\min_{\rho \in S} \max_{t \in T} \{ \sum_{l_\rho=1}^{n} ba^t_{l_\rho} \} \Rightarrow \min_{\rho \in S} \max_{t \in T} \{ \sum_{l_\rho=1}^{n} ba^t_{l_\rho} \} \leq \min_{\rho \in S} \{ \sum_{l_\rho=1}^{n} \max_{t \in T} \{ ba^t_{l_\rho} \} \}$$

Thus the solution of the SMP is an upper bound to the master problem.

If this upper bound is less than the current bound, we have a feasible solution to the Master Problem. Of course there is no guarantee that the upper bound provided by the SMP improves the overall upper bound. When $\theta_{SMP} \geq \theta_M$ two things can happen, (1) we get a new sequence in which case we continue iterating or, (2) we get the same sequence in which case the algorithm will produce this same sequence on subsequent iterations (i.e. we are stuck). When the algorithm gets stuck we restart the algorithm from a new sequence using one of the "restart rules" described in the next section.

2.6.4 Restart Rules

Once the algorithm returns the same sequence for two consecutive iterations, it will continue to do so ad infinitum, therefore, we need to restart from a new sequence. Given the new restart sequence, we simply remove all previous Benders' cuts and start again. We tried three restart rules as discussed below.

Worst Case

This anti-cycling rule is based on finding a sequence "far way" from sequences visited since the last restart. To accomplish this we simple replace "minimize" with "maximize" in the SMP objective function.

Perturbation

This anti-cycling rule tries to slightly perturb the sequence in order to escape the cycle.

$$\text{minimize: } \theta_{A_r} = \sum_{i=1}^{n}\sum_{j=1}^{n}(\overline{ba_{ij}} + ba_{min}U(0,1))x_{ij}$$

$$\begin{aligned} \text{s.t.:} \quad & \textstyle\sum_{i=1}^{n} x_{ij} = 1 \ \forall j \in \{1..n\} \\ & \textstyle\sum_{j=1}^{n} x_{ij} = 1 \ \forall i \in \{1..n\} \\ & 0 \leq x_{ij} \leq 1 \end{aligned}$$

where $\overline{ba_{ij}} = \max_{t\in\{1..T\}}\{ba_{ij}^t\}$ and $ba_{min} = min_{i,j}\overline{ba_{ij}}$ and U(0,1) represents pseudo-randomly generated Uniform(0,1) deviates. The value of ba_{min} was chosen so that the sequence will not change much. Unfortunately, this rule does not guarantee a new restart sequence difference from all previous restart sequences. Thus we may repeat a previous iteration. To avoid this we developed the next method.

Memory Random Restart

This anti-cycling rule is based on finding a sequence different from all previous restart sequences. We store all restart sequences $R_1, .., R_q$ where q is the number of times that we have restarted the algorithm. The idea of this restart rule is to guarantee that we are not going to cycle between restarting points and therefore the algorithm will visit at least one new sequence each iteration. We formulated the following feasibility IP (M_q) which will produce a new starting sequence when solved. Let $\phi(R_r) = (i, j)$ such that $x_{ij} = 1$ in solution R_r

$$
\begin{aligned}
\text{s.t.:} \quad & \textstyle\sum_{i=1}^{n} x_{ij} = 1 && \forall j \in \{1..n\} \\
& \textstyle\sum_{j=1}^{n} x_{ij} = 1 && \forall i \in \{1..n\} \\
& \textstyle\sum_{\gamma \in \phi(R_r)} x_{\gamma} - \sum_{\gamma \notin \phi(R_r)} x_{\gamma} \leq \psi && \forall r \in \{1,..,q\}, \gamma \in (I,J) \\
& x_{ij} \in \{1,0\}
\end{aligned}
$$

where ψ is random number generated from IntUniform(0,n-1). Basically this IP will find a new starting solution that is different than all previous restart solutions. $R_1, , R_q$ are the previous restart solutions from the first q iterations. The parameter ψ can be used to specify "how different" the new starting solution should be compared to the previous q starting solutions. We use Cplex to solve this problem. Cplex finds a feasible solution extremely quickly as it turns out, so that the impact on the overall algorithm's execution time is minimal. The third set of constraints that contain the previous restart points guarantee that we will not restart the next iteration from any of these previous sequences. Further, the parameter ψ serves as a kind of distance from the set of restarting points where when ψ =n-1 we might obtain a feasible sequence for M_q that differs from restart sequences R_r in at most 2 elements. When ψ =0 the new restart sequence will differ from every previous restart point in all the elements.

2.7 Accuracy of the Finite Sample Average Approximation

The ultimate goal is to solve the infinite scenario stochastic programming problem. The method proposed in this research attempts to solve a finite scenario problem. Further, the method is heuristic and thus does not guarantee an optimal solution to this finite scenario problem. In this section we will try to evaluate how our algorithm will perform on the infinite scenario problem. Linderoth et al. [2006] developed a way to compute statistical upper and lower bounds on the optimal solution to the infinite scenario case based on external sampling techniques. Unfortunately, this method requires solving a finite scenario problem many many times and thus is computational prohibitive in our case, even for a small number of scenarios. We therefore designed a simpler experiment to quantify the performance of our algorithm on the infinite scenario case. There are two main issues. The first issue is sampling error, that is: "How well is the infinite scenario objective function approximated by the finite scenario (sample average) objective function?" The second issue is: "How does the run time of the algorithm affect performance with respect to the infinite scenario problem?" There is a basic tradeoff we need to evaluate. For a fixed computation time allowance, how many scenarios should we use? If we use too many scenarios, we will only have time to generate a few candidate sequences thus limiting our ability to "optimize". On the other hand with too few scenarios, we can generate many sequences, and perhaps even solve the finite scenario problem to optimality. However, the optimal solution to the finite scenario problem may be a poor solution to the infinite scenario problem when the number of scenarios is small. To quantify this tradeoff we constructed the following experiment. We generated 50 test problems, each with 10 jobs. For each of these 50 problems we generated finite scenario instances with 50, 100, 250, and 500 scenarios. For each of these 200 instances we then ran the algorithm for 20,40,60,80,100,120,140 seconds (the experiments were conducted under the same computational conditions detailed in section 7). For each run of the algorithm we saved the best ten sequences where "best" is with respect to the number of scenarios used in the current

run. This resulted in the generation of 7x4x10=280 (not necessarily all different) total sequences for each of the 50 test problems. For each of these 280 sequences, we solved the LP scheduling sub-problem with 10,000 scenarios and reported the best (of the 280) sequence S^*_{10000} and objective function z^*_{10000} found. This solution serves as our approximation to an overall best solution to the infinite scenario problem for each of the 50 test cases.

Our first set of results is aimed at understanding the sampling error. The basic question we asked is: "How often does the sequence our algorithm thinks is best turn out to be the best sequence in the 10,000 scenario case?" For each run of 140 seconds, we stored the ten best solutions (with respect to the finite scenario objective function) found. Thus the 10 solutions are those that the finite scenario objective function "thinks are best". These ten sequences were then evaluated with 10,000 scenarios. Figure 2.5 is a plot of the percentage of the time that the solution judged to be the ith best with finite scenarios (for 50, 100, 250, and 500 scenario cases) was actually the best for the 10,000 scenario case. For example, for 50 and 100 scenarios, each of the top ten solutions has roughly equal probability of being the best for the 10,000 problem. In other words, 50 or 100 scenarios is not enough to distinguish which of the 10 proposed solutions is actually best for 10,000 scenarios. On the other hand Figure 5 shows that for 500 scenarios, the solution judged to be the best of the top ten by the algorithm (with 500 scenarios) was indeed the best (as determined by 10,000 scenarios) 40% of the time. By examining the plot we see that 50 and 100 scenarios incur significant sampling error since each of the top ten solutions has essentially the same probability of being best for the 10,000 scenario case. With 500 scenarios, we see that solutions that are good for the finite scenario problem also tend to be good for the 10,000 scenario problem. We can conclude that 100 scenarios do not provide a sufficient approximation to the 10,000 scenario case. 500 scenarios seems to provide a reasonable approximation while 250 is borderline.

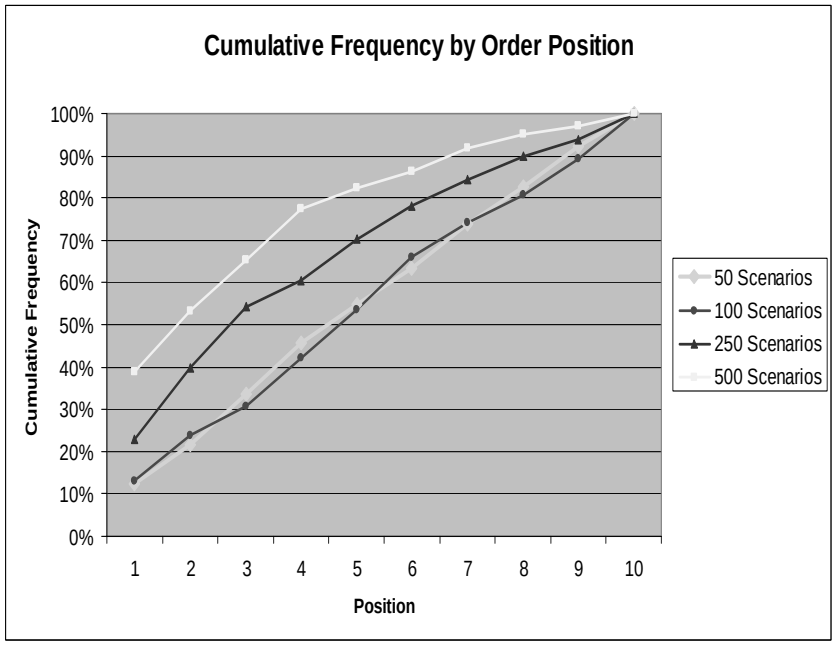

Figure 2.5: Cumulative Frequency by Order Position

The second set of results demonstrates the tradeoff between the number of scenarios and the number of sequences generated. For each number of scenarios and run times we take the best sequence (according to the finite scenario algorithm), evaluate it with 10,000 scenarios using the LP scheduling sub-problem, then compute the percent error from z^*_{10000}. The percent error is averaged over the 50 test instances and shown in the plot below. We observe that in the case of 50 and 100 scenarios, the algorithm makes little or no progress in finding improved solutions with respect to the 10,000 scenario objective. This is presumably because 50 and 100 scenarios are not enough to adequately approximate the infinite scenario case. In the case of 250 and 500 scenarios, it appears that the approximation of the infinite scenario case is good enough since the algorithm does make steady progress in improving the solution with respect to 10,000 scenarios. The 250 scenario algorithm makes faster initial progress presumably because it executes more

algorithm iterations in a fixed amount of run time than the 500 scenario algorithm. After 140 seconds, the 500 scenario case has "caught up with" the 250 scenario case. If one were to run the algorithm for more than 140 seconds, 500 scenarios are conjectured to be the better choice. We base this conjecture on the observation that as the solutions get better and better, and thus closer in objective function value, it becomes harder for the finite scenario approximation to discriminate between them.

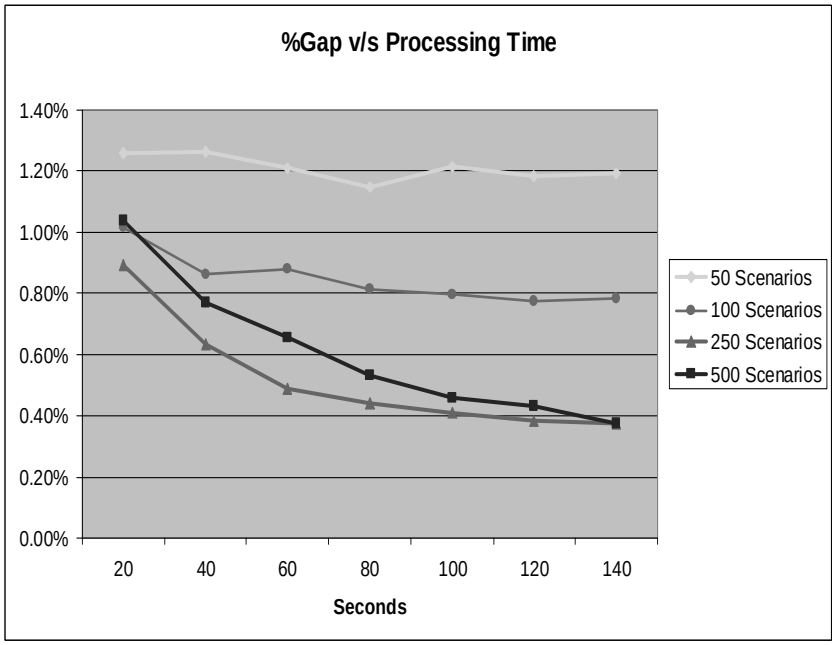

Figure 2.6: Percent Error Results

In order to further understand the impact of the number of scenarios, we created two additional plots, Figure 2.7 for N=50 scenarios and Figure 2.8 for N=500 scenarios. Recall that for each N we saved the best 10 solutions. For each solution we calculated the average and sample standard deviation (over the N scenarios) and over 10,000 scenarios. On each plot we show approximate 95% confidence intervals based on N and on 10,000 scenarios for each of the ten solutions. In the plot based on 50 scenarios we observe that the confidence intervals are typically disjoint. From this one can infer that 50 scenarios is not enough to accurately estimate the infinite scenario objective function value. For the plot based on 500 scenarios, the N=500 confidence interval always contains the 10,000 scenario confidence interval indicating that 500 scenarios provides a reasonable approximation to the infinite scenario objective function.

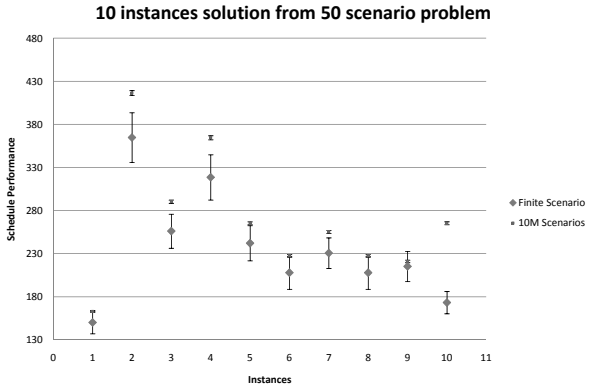

Figure 2.7: Confidence intervals from 50 scenarios instances

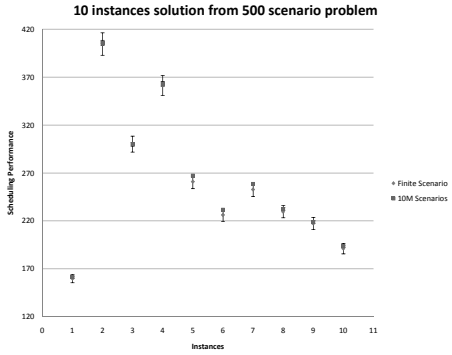

Figure 2.8: Confidence intervals from 500 scenarios instances

2.8 Computational Experience for the Finite Scenario Case

Our computational experiments are based on joint work with a local hospital. This hospital follows a "block-booking" procedure to schedule surgeries in their operating rooms. In block booking, a surgeon or surgical practice is assigned a specific time block in an operating room on a recurring basis (e.g. Tuesday-Thursday 7-4). A surgical patient will typically meet with the surgeon during an office visit to determine if surgery is needed and if to pick a date for the surgery. As time progresses, the blocks will eventually be filled with surgeries. However, the final sequence and schedule for a block is not determined until a couple of days before hand. At this time the final daily surgical schedule is constructed, and surgical patients are notified when to report to the hospital. It is at this point that the proposed methodology can be applied.

To quantify the performance of our algorithms for solving the finite scenario problem we performed two sets of experiments. In the first we compared solutions from the algorithms with the optimal solution on smaller problems. In the second we compared algorithm performance with two simple benchmark heuristics over a broader set of test problems. The experiments were conducted on a Pentium Xeon 3.0 GHz (x2) with a 4000 (MB) server in the Coral Lab at Lehigh University.

2.8.1 Comparison with Optimal Solutions

In this section, we compared the solutions found by our memory restart algorithm to the optimal solution found by Cplex using branch and bound. We generated two categories of test problems, those with equal costs (i.e. costs are equal across surgeries) and those with different costs. It is worth noting that the cases with equal cost were significantly easier for Cplex to solve to optimality than were the cases where the costs were different. We generated 10 instances with 10 scenarios and 10 instances with 100 scenarios in each category. In the next section we discuss in detail how we generated a large suite of test problems by varying a variety of factors. In this section we selected instances so as to span a wide range of these factors.

To find optimal solutions we used Cplex 10.2 to solve the strengthened IP formulation discussed in subsection 3.2. Within Cplex the "MIP emphasis" parameter was set to "optimality". We ran our memory restart algorithm for 1000 and 4000 iterations for each test problem. Tables 2.1 shows the results of this experiment. The Gap is calculated by

$$\frac{\text{Hueristic objective function value - Optimal objective function value}}{\text{Optimal objective function value}}$$

ABend.(s) gives the run time of the heuristic in seconds. Cplex(h) gives the run time for Cplex in hours.

The run time results and gaps are the averages over the 10 problem instances. Note that in some cases Cplex took several days to find the optimal solution while our proposed method took at most 4 minutes to find solutions with average optimality gap of at most 5.38%. It is interesting to note that the optimality gap seems to be smaller for the problems with more scenarios. One possible explanation for this behavior is that as the number of scenarios increases, the number of iterations between restarts of our algorithm also increases.

Table 2.1: Optimality Gap Different Cost Case

#Inst.	#Surgeries	Scenarios	Iter.	Opt. Gap (%)	ABenders(sec)	Cplex (hours)
10	10	10	1000	2.20	7.3	0.88
10	10	10	4000	0.84	47.3	0.88
10	10	100	1000	0.90	44	126.73
10	10	100	4000	0.48	183.2	126.73

Table 2.2: Optimality Gap Equal Cost Case

#Inst.	#Surgeries	Scenarios	Iter.	Opt. Gap (%)	ABenders(sec)	Cplex (hours)
10	10	10	1000	5.38	7.3	0.078
10	10	10	4000	3.65	47.3	0.078
10	10	100	1000	1.00	44	13.67
10	10	100	4000	0.80	183.2	13.67

2.8.2 Comparison with Benchmark Heuristics

Benchmark Heuristics

We implemented two simple heuristics for comparison purposes. The first is the "sort by variance" heuristic proposed by Denton et al. [2007]. This simply sequences jobs from smallest to largest variance. This heuristic can be expected to work fairly well in the equal costs case, but there is no reason to expect it to work well in the unequal costs case. The second benchmark heuristic was a simple perturbation heuristic based on sort by variance. A randomly generated perturbation was added to each job's variance, then jobs were sequenced from smallest to largest perturbed variance. The perturbations were generated from $2\mathrm{U}(-\theta,\theta)(\theta=\frac{\sigma_{max}-\sigma_{min}}{n}$, where n is the total number of jobs and σ_{max} is the maximum standard deviation of the jobs) .The idea is to generate a number of sequences that are "close" to the sort by variance sequence. We generated the same number of sequences as were generated by the Benders' based heuristics and reported the best one found. Since computation time is dominated by the time required to solve the LP subproblems, the run times were roughly equivalent.

Test Problems

We created a set of test problems based on factors that might affect algorithm performance. The factors (shown in Table 2.3) chosen were: (1) number of surgeries, (2) presence of overtime cost, (3) wait and idle cost structure, (4) number of scenarios, and (5) surgery time distribution. The number of iterations was fixed at 1000 (we study the effects of number of iterations in the second experiment to follow). Please, note that the indices

Factor	Possible Value
Number of Surgeries	10, 15, 20
Overtime Cost	Yes , No
Cost Structure	equal waiting and equal idle costs, different costs
Number of Scenarios	10, 50, 100, 250, 500
Surgery Distribution	$(\mu, \sigma_i)_1, (\mu, \sigma)_2, (\mu_i, \sigma)_3, (\mu_i, \sigma_i)_4$

Table 2.3: Experiment Design

1,2,3,4 in the possible values of the factor Surgery Distribution are used as the labels of Data in Figure 2.9.

The means and variances of the surgery duration distributions were based on real data from a local hospital. We had data on surgery durations by surgeon and procedure. There were 44 surgeon-procedure pairs with at least 30 observations. We used these 44 data sets as the basis for generating scenarios in our problems. We then generated simulated surgery times from normal distributions (truncated at zero) with parameters reflected in the real data.

In table 2.3 under surgery distribution the symbol μ,σ means all surgery durations were generated using the same mean and standard deviation (186 , 66). The symbol μ_i,σ means that σ was set at 66 and μ_i was set based on the coefficient of variation generated from a uniform(0.21,1.05)distribution.

The symbol μ,σ_i means that μ was set at 186 then σ_i was set based on the coefficient of variation generated from a uniform(0.21,1.05) distribution.

The symbol μ_i,σ_i means that μ_i was first generated from uniform(90,300) distribution

then σ_i was set based on a coefficient of variation generated from a uniform(0.21,1.05) distribution.

In the equal cost case we generated a single waiting cost and idle cost each from a uniform (20,150) distribution. These two costs were then applied to every surgery. In the unequal cost case individual idle and waiting costs were generated for each surgery, again from a uniform (20,150) distribution. When overtime is included, the over time cost is set to 1.5 times the average of the waiting costs. The deadline was set equal to the sum (over surgeries) of the average (over scenarios) duration plus one standard deviation (over scenarios) of this sum. We created a full factorial experimental design and performed 5 replicates for each combination of factor levels. This resulted in 1200 instances for each of the five algorithms tested: Approximate Benders' Decomposition (with the three different restart rules), sort by variance, and perturbed sort by variance with 1000 iterations. We generated 1000 (not necessarily unique) sequences for each algorithm (except sort be variance), solved the LP sub-problem for each to get the objective function, and report the best solution found. Thus for each problem instance we have five solutions, one for each heuristic. We take the best solution of these five, then compute the percent gap from this best solution for each heuristic for each problem instance. The overall results appear in tables 2.4 and 2.5.

Algorithm v/s Scenarios	10	50	100	250	500	% Gap
Approximate Benders' (RS: Perturbation)	4.7	2.5	1.5	1.0	0.7	2.1
Approximate Benders' (RS: Worst Case)	3.5	1.8	0.7	0.5	0.4	1.4
Approximate Benders' (RS: Memory Random)	1.2	1.2	0.3	0.2	0.3	0.7
Sort by Variance	32.1	17.0	7.3	4.9	2.9	12.9
Perturbed Sort by Variance	17.0	10.3	3.6	3.2	1.7	7.2

Table 2.4: Average percent gap over the best solution found v/s scenarios (equal cost case)

The results show that all three Benders' based algorithms significantly out perform the simpler heuristics. We performed an analysis of variance and graphed interaction plots to see how algorithm performance was affected by the experimental factors. The

Algorithm v/s Scenarios	10	50	100	250	500	% Gap
Approximate Benders' (RS: Perturbation)	5.9	2.9	2.0	1.6	1.4	2.8
Approximate Benders' (RS: Worst Case)	3.9	1.9	1.0	0.7	0.7	1.6
Approximate Benders' (RS: Memory Random)	1.2	1.0	0.5	0.5	0.5	0.7
Sort by Variance	28.5	15.1	12.7	11.8	10.3	15.7
Perturbed Sort by Variance	13.7	7.5	5.5	4.9	3.6	7.1

Table 2.5: Average percent gap over the best solution found v/s scenarios (different cost case)

interaction plots which appear in figure 2.9 show that the performance of each algorithm was reasonably uniform over the different factors.

Tables 2.4 and 2.5 show the algorithm performance as the number of scenarios varies. It is interesting to note that as the number of scenarios increases, the difference in performance decreases. In particular for the equal costs case, the simple sort by variance heuristic performs quite well compared with the other methods. Since the ultimate goal is to solve the infinite scenario problem, it would seem that sort by variance is an effective heuristic in the case of equal costs. When costs are not equal, significant improvement over sort by variance is possible.

The previous experiment fixed the number of iterations at 1000. We conducted a second experiment which examined the performance of the three algorithms over a varying number of iterations. In this experiment the factors chosen were: number of surgeries, number of scenarios and number of iterations (1000, 2000, 3000,4000). We further assume no overtime cost, the distributions of the job durations are from the case where σ_i and μ_i have no restrictions, and the cost coefficient are not equal. The results of the analysis of variance show that all main effects (algorithm, iterations, scenarios, and number of surgeries) are statistically significant. In particular the "memory" restart heuristic was (statistically) significantly better than the other two. Interestingly, no two factor or higher interactions were significant implying that algorithm performance behaved in a very uniform manner across the other factors. From this we conclude that the memory heuristic appears to be the best in a statistical sense in all cases, although the actual difference in

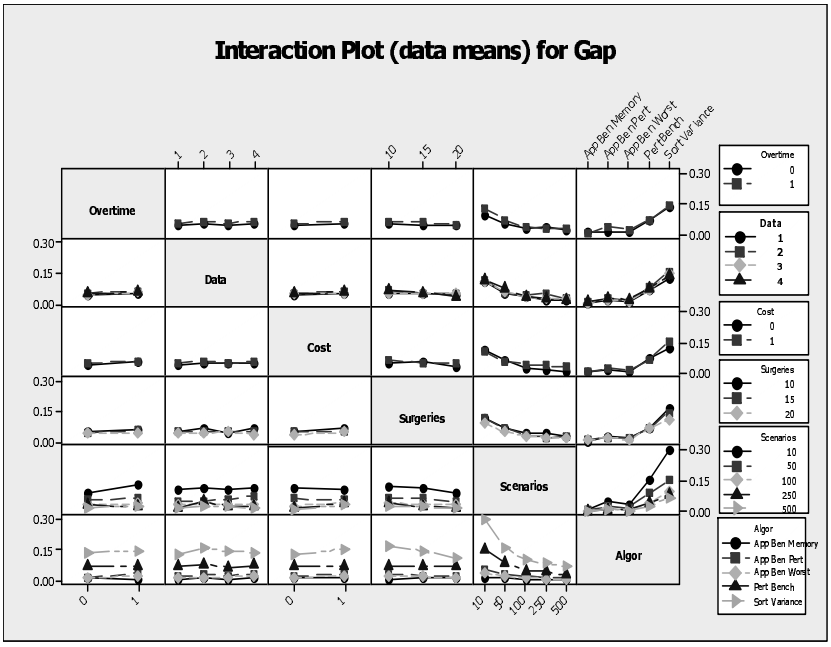

Figure 2.9: Interaction plots for experiment 1

performance is fairly small. The interaction plots in figure 2.10 illustrate these results. We also report the average run times for the memory algorithm and the average number of restarts in Table 2.7.

Factor	Possible Value
Number of Surgeries	10, 15, 20
Number of Scenarios	10, 50, 100, 250, 500
Number of Iterations	1000, 2000, 3000, 4000

Table 2.6: Design for experiment 2

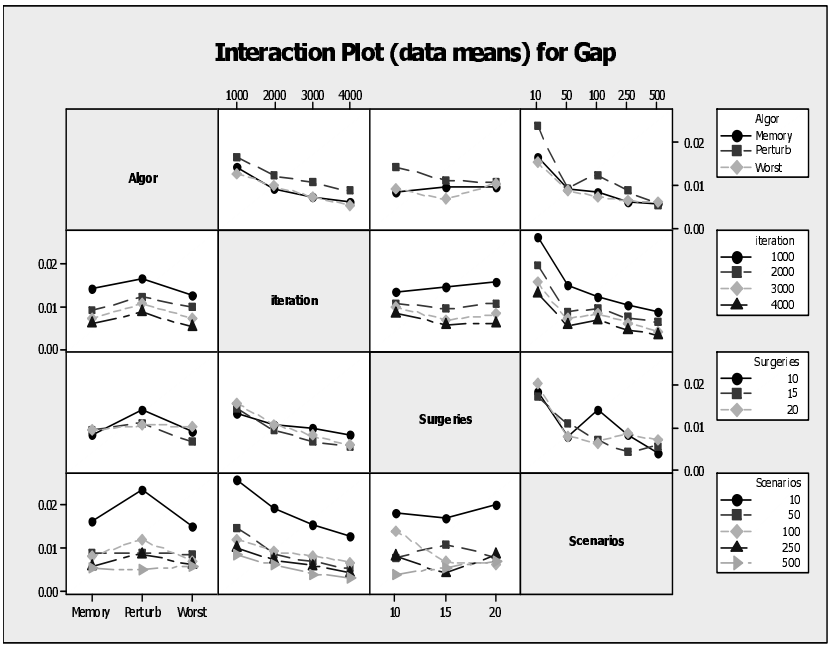

Figure 2.10: Interaction plots for experiment 2

	Average processing time (secs.)				
$\|J\|$ v/s $\|K\|$	10	50	100	250	500
10	5.8 - 7.3	18.4 - 18.9	34.7 - 39.0	97.4 - 112.6	200.2 - 234.9
15	8.0 - 9.4	28.0 - 29.9	69.7 - 67.3	247.0 - 204.3	382.3 - 435.0
20	10.5 - 12.9	40.9 - 56.9	99.5 - 104.2	282.5 - 315.9	694.7 - 771.8

Table 2.7: For memory algorithm over 1000 iterations (x - y) x: no overtime cost y: overtime cost

	Average restarts				
$\|J\|$ v/s $\|K\|$	10	50	100	250	500
10	42.6 - 89.7	31.5 - 49.5	33.3 - 50.2	32.6 - 49.4	33.1 - 47.8
15	26.1 - 64.2	17.1 - 33.5	16.5 - 32.0	16.6 - 31.3	16.7 - 32.0
20	18.7 - 60.8	11.1 - 24.5	10.3 - 24.6	10.0 - 22.7	10.1 - 16.4

Table 2.8: For memory algorithm over 1000 iterations (x - y) x: no overtime cost y: overtime cost

2.9 Conclusions

In this paper we developed new sequencing algorithms for the stochastic appointment scheduling problem. Few papers have addressed this problem, the lone exception being Denton et al. [2007]. Denton tested some simple heuristics and showed that "sort by variance" was the best of those tested. In this paper we developed new algorithms based on Benders' decomposition which perform significantly better than sort by variance and a perturbed sort by variance heuristic, especially in the unequal cost case. To be fair, the proposed algorithms utilize much more computing time (a couple of minutes) than sort by variance, but roughly the same time as perturbed sort by variance. The algorithm run times are more than sufficient for implementation on real problems. Of the three Benders' based algorithms, the "memory restart" method provides the best results (statistically), but the difference between the three methods is fairly small from a practical standpoint. The results further showed that the relative performance of the three algorithms is uniform across the several factors used to create problem instances with different characteristics in testing. It is worth noting that the general approach used to create our heuristics may work well for other problems where the master problem without Benders' cuts is easy to solve, but the problem with cuts is hard. This approach seems particularly amenable to stochastic sequencing problems where the master problem (before Benders' cuts) is an assignment problem. A final contribution of this paper is a formal proof that the sequencing problem is NP-Complete. While this has been alluded to by several authors, the question, to the best of our knowledge, was previously open.

Chapter 3

Stochastic Sequencing of Surgeries for a Single Surgeon Operating in Parallel Operating Rooms

3.1 Introduction

Operating Rooms (OR's) account for more than one third of the total revenue in a hospital according to the Health Care Financial Management Association HFMA [2005]. Further, many if not most hospital activities are driven by the surgery schedule, thus improvement in utilization and efficiency can have a significant cost impact. In this paper a method is developed to help the OR Manager schedule a set of surgeries for a single surgeon operating in two parallel OR's. Specifically the problem is to assign each surgical procedure to one of the two OR's and determine the sequence of surgeries in each room. It is assumed that the surgical procedures can be divided into three stages; setup, surgery, and cleanup, where surgery refers to that part of the procedure requiring the presence of the surgeon. Since the setup and cleanup stages do not require the surgeon, he or she can move to the other OR to perform surgery. If all surgeries were performed in a single OR, the surgeon would

experience waiting time during the setup and cleanup stages of each procedure (excluding the setup before the first surgery and the cleanup after the last surgery). With a single OR, there will be no OR staff idle time since there is always an activity to perform. If parallel OR's are used there is an opportunity to significantly reduce the surgeon's waiting time. However, the possibility for OR staff idle time now exists when surgery in one room extends past the time setup is complete in the other room. When surgery times are long relative to setup and cleanup, using parallel ORs is likely to be unrealistic due to the large amounts of OR staff idle time that would necessarily result. If on the other hand surgery time is comparable to setup and cleanup, the possibility exists to greatly reduce surgeon waiting time without incurring significant OR staff idle time. In this paper the setup, surgery, and cleanup times are modeled as random variables with known joint distribution. The marginal distributions of setup, surgery and cleanup durations are not assumed identical for each surgical procedure but rather differ based on the surgeon, type of surgical procedure, patient attributes, etc. The objective of our optimization problem is to minimize a weighted linear combination of expected costs that include surgeon waiting time, OR idle time and staff overtime. In order to make operational decisions on a daily basis, the method must produce solutions in a timely manner.

We formulate a two-stage stochastic integer programming model then propose a decomposition method based on the first stage variables and the mean value problem to solve it. We perform computational testing to investigate algorithm behavior, and also provide managerial insight regarding when single-surgeon, parallel O.R. scheduling is viable.

The remainder of the article is organized as follows. In the next section a brief review of the literature related to the two-machine single-server scheduling problem and OR scheduling is provided. In Section 3.3, the model is described and formulated. In Section 3.4 algorithms are proposed to solve the sequencing and assignment problems. In Section 3.5 computational results are presented. In Section 3.6 we present computational results with respect to the number of scenarios needed to solve the problem. In Section 3.7

we provided insight as to when parallel operating rooms is cost effective. In Section 3.8 conclusions and future research directions are discussed.

3.2 Literature Review

Two branches of previous research are relevant to this paper, work on surgical scheduling, and work on two-machine, single-server sequencing. Batun et al. [2011] consider a model where surgical procedures are also composed of random setup, surgery and cleanup times with known distributions. They consider a problem with M operating rooms and N surgeons. Each surgeon has a known set of surgeries and a pre-specified sequence in which they will be performed. The authors find the optimal assignment of these surgeries to operating rooms and the sequence of surgeries in each room. They show computational results for problems with 4 to 11 total surgeries, 2 to 3 surgeons and 3 to 6 ORs. They used the L-Shaped method and also derived cuts based on the mean value problem to speed up the algorithm. In our work, we do not assume that the sequence of surgeries is known in advance, and have only a single surgeon and two OR's.

In the machine scheduling literature a deterministic version of the problem we address has been studied by Koulamas [1996], Abdekhodaee and Wirth [2002], Abdekhodaee et al. [2006]. This problem involves scheduling a set of jobs on 2 machines where the setup is performed by a single server (e.g. a robot). Once the setup for a job is complete, the job is processed on one of the two machines. Problems of this type are found in manufacturing and network computing according to Glass et al. [2009]. All of these studies assume deterministic setup and processing times. The correspondence between the machine scheduling problem and our parallel OR scheduling problem can be seen if we set our setup times to zero, let our surgery times equate to their machine setup times (which require a single resource, the surgeon or robot), and their machine processing times equate to our cleanup time (which can be done in parallel). Glass et al. [2009] created an approximation algorithm with a performance guarantee and also provided a description

of polynomially solvable cases. Abdekhodaee and Wirth [2002] considered problems in which all setup times were greater than all processing times. Abdekhodaee et al. [2006] extended the work of Abdekhodaee and Wirth [2002] to the case in which the setup and processing times do not have any restrictions. They developed different heuristics that they then compared against meta-heuristic approaches. None of these studies considered stochastic processing times.

3.3 Mathematical Formulation

In this section we formulate a stochastic integer programming model for the problem described above (the Sample Average Parallel Operating Room Scheduling (SAPOS) Problem). In order to account for the randomness of the setup, surgery, and cleanup times we use sample average approximation. The model we develop has pure binary first stage decision variables x_{ij} representing the sequence of surgeries, and m_{ir} representing the assignment of surgery i to one of the two ORs. The model also includes the variable q_{ij} representing the immediate predecessor surgeries in the overall surgery sequence). Note that the q variables can be calculated directly from $\vec{x}$. The second stage variables include, for each scenario, the start and finish time of each stage of each procedure, the surgeon waiting time, OR idle time, and overtime in each OR. Since the stage two variables are easily computed given values for the stage one binary variables $\vec{x}$ and $\vec{m}$, we have "simple recourse". Finally we can compute the average (over scenarios) cost. The problem formulation appears below.

$$\text{minimize: } \sum_{k=1}^{K} \frac{1}{K} \left(\sum_{i \neq j} c_w W_{ij}^k + \sum_{r \leq R} c_i I_r^k + \sum_{r \leq R} c_o O_r^k \right) \tag{3.1}$$

s.t.:

$$x_{ij} + x_{ji} = 1 \qquad \forall i, j \tag{3.2}$$

$$x_{ij} + x_{je} + x_{ei} \leq 2 \qquad \forall 1 \leq i \neq j \neq e \leq n \tag{3.3}$$

$$a_{ijr} \leq x_{ij} \qquad \forall i, j, r \tag{3.4}$$

$$\sum_{r \leq R} m_{ir} = 1 \qquad \forall i \tag{3.5}$$

$$a_{ijr} + a_{jir} \leq m_{ir} \qquad \forall i, j > i, r \tag{3.6}$$

$$a_{ijr} + a_{jir} \leq m_{jr} \qquad \forall i, j > i, r \tag{3.7}$$

$$a_{ijr} + a_{jir} \leq m_{ir} + m_{jr} - 1 \qquad \forall i, j > i, r \tag{3.8}$$

$$\sum_{t \neq i} x_{it} + 1 = \sum_{u=1}^{n} u q_{iu}^1 \qquad \forall i \tag{3.9}$$

$$\sum_{u=1}^{n} q_{iu}^1 = 1 \qquad \forall i \tag{3.10}$$

$$q_{ij} = \sum_{u=1}^{n-1} q_{iju}^2 \qquad \forall i, j \neq i \tag{3.11}$$

$$q_{iju}^2 \leq q_{iu+1}^1 \qquad \forall i, j \neq i, u < n \tag{3.12}$$

$$q_{iju}^2 \leq q_{ju}^1 \qquad \forall i, j \neq i, u < n \tag{3.13}$$

$$q_{iju}^2 \geq q_{ju}^1 + q_{iu+1}^1 - 1 \qquad \forall i, j \neq i, u < n \tag{3.14}$$

$$C_i^k \leq M m_{ir} \qquad \forall i, r \tag{3.15}$$

$$W_{ij}^k \geq \sum_{r \leq R} C_{jr}^k - \sum_{r \leq R} C_{ir}^k + post_i^k - sur_j^k - post_j^k - M(1 - q_{ij}) \qquad \forall i, j \neq i, k \tag{3.16}$$

$$W_{ij}^k \leq \sum_{r \leq R} C_{jr}^k - \sum_{r \leq R} C_{ir}^k + post_i^k - sur_j^k - post_j^k + M(1 - q_{ij}) \qquad \forall i, j \neq i, k \tag{3.17}$$

$$\sum_{r \leq R} C_{jr}^k \geq \sum_{r \leq R} C_{ir}^k + pre_j^k + sur_j^k + post_j^k - M(1 - a_{ijr}) \qquad \forall i, j \neq i, k \tag{3.18}$$

$$C_{ir}^k \geq pre_i^k + sur_i^k + post_i^k - M(1 - m_{ir}) \qquad \forall i, r, k \tag{3.19}$$

$$C_{ir}^k \leq l_r^k \qquad \forall i, r, k \tag{3.20}$$

$$I_r^k = l_r^k - \sum_{j=1}^{n} m_{ir}(sur_j^k + pre_j^k + post_j^k) \qquad \forall r, k \tag{3.21}$$

$$l_r^k - d_r = O_r^k - g_r^k \qquad \forall r, k \tag{3.22}$$

$$C_{ir}^k, I_r^k, l_r^k, O_r^k, g_r^k, W_{ij}^k \geq 0 \qquad \forall i, j, k, r$$

$$x_{ij},\ m_{ir},\ a_{ijr},\ q_{iu}^1,\ q_{ij},\ uq_{iju} \in \{0, 1\} \qquad \forall i, j, u, r$$

Indices and Sets

J Jobs to be scheduled j=1,...,n.

R Operation Rooms to be considered.

K scenarios to be considered k=1,....,K.

Parameters

c^w Doctor waiting time penalty.

c^o overtime penalty.

d_r time beyond which overtime is incurred.

M is sufficiently large "big M" number. .

pre_j^k duration of setup for procedure j in scenario k.

sur_j^k duration of surgery for procedure j in scenario k.

$post_j^k$ duration of cleanup for procedure j in scenario k.

Variables

W_{ij}^k Surgeon waiting time between procedure i and j in scenario k.

I_r^k Total OR idle time in scenario k for OR r.

C_{ir}^k Completion time of procedure i in OR r in scenario k.

O_r^k Overtime time in OR r in scenario k.

g_r^k a slack variable that measures the earliness with respect to time d_r in scenario k in O.R. r.

x_{ij} a binary variable denoting the assignment of surgery i as a predecessor of surgery j in the overall surgery sequence.

m_{ir} a binary variable denoting the assignment of surgery i to OR r.

a_{ijr} a binary variable denoting the assignment of surgery i as a predecessor of surgery j in OR r.

q_{ij} a binary variable denoting the assignment of surgery i as the immediate predecessor of

surgery j in the overall surgery sequence.

q^1_{ij},uq_{ij} utility binary variables that help to compute q_{ij}.

Constraints

(3.2), (3.3), (3.4) define the possible sequences for the surgeon.

(3.5) every surgery must be assigned in one OR.

(3.6), (3.7), (3.8) define the sequence inside each OR.

(3.9), (3.10), (3.11), (3.12), (3.13), (3.14) compute the immediate predecessor of surgeries.

(3.15) set to zero the value of completion time if surgery not assigned to this room.

(3.16), (3.17) define the surgeon waiting time.

(3.18), (3.19) define the completion time for each surgery for a given assignment.

(3.20) calculates the completion time for every OR.

(3.21) calculates the OR idle time.

(3.22) calculates the OR overtime.

It can be seen that computing the idle time, overtime time and surgeon waiting time is a simple calculation once we have the sequence of surgeries (x variables) and the respective sequence inside each OR (m variables).

3.3.1 Problem Complexity

The SAPOS problem can be shown to be NP-Hard through a simple extension of results from Koulamas [1996]. Koulamas [1996] proved that the deterministic version of the following problem is NP-Hard. This definition is extracted as it appeared in Koulamas' paper.

"Consider a deterministic n job, two-machine one-server scheduling problem with the following assumptions.

- There are two parallel identical semiautomatic machines. The machines run unattended during job processing; however the presence of a server (robot) is required during set-ups.
- There is a set of jobs awaiting processing at time zero. Each job i has a processing time P_i and a sequence independent set-up time s_i. All s_i and P_i are deterministic and known in advance.
- There is one server (robot) serving both machines. Overlapping requests for the server result in machine idle time (interference).
- Each machine processes one job at a time and job pre-emption is not allowed."

The problem described in Koulamas [1996] is a special case of our problem where our setup times are zero, their setup times equate to our surgery times, and their processing times equate to our cleanup times. Therefore, the parallel OR scheduling problem is NP-Hard even in the single scenario (deterministic) case with zero setup time.

3.4 Proposed Solution Approach

Our first attempt at solving the problem was to simply use Cplex to solve the extended formulation. Unfortunately, Cplex was not able to achieve a reasonable relative gap (100%) in two hours with only 1 scenario and ten surgeries. Our second attempt was to use the Integer L-Shaped method, with the binary (x and m) first stage decision variables. The Integer L-Shaped method failed to close the relative gap with results similar to Cplex. During experimentation we discovered that if we fixed surgeon-OR sequence, the resulting MIP sub-problem (which we call the "surgery-sequencing sub-problem") was solvable in

manageable processing time. To clarify, we define vector y as the "surgeon-OR sequence". For instance, if y is (0,1,0,0,1,1) then the surgeon will perform the first surgery in room 1, then room 2, then 2 surgeries in a row in room 1, then 2 in a row in room 2, etc. The search space of the vector y is large (there are 2n possible surgeon-OR sequences) but we can take advantage of other characteristics of the problem to make this search easier. The following is the extended problem formulation assuming we have assigned the surgeries in a "zig-zag" fashion ($\vec{y}$=(0,1,0,1,0,1....))

$$\min \sum_{k=1}^{K} \frac{1}{K} \left(\sum_{i=1}^{n-1} c_w W_i^k + \sum_{i=1}^{n-2} c_i I_i^k + c_i f s^k + \sum_{r \leq R} c_o O_r^k \right) \tag{3.23}$$

$$\text{s.t. } : C_1^k = \sum_{j=1}^{n} (pre_j^k + post_j^k + sur_j^k) v_{1j} \qquad k \tag{3.24}$$

$$W_i^k = C_{i+1}^k - C_i^k + \sum_{j=1}^{n} post_j^k v_{ij} - \sum_{j=1}^{n} (sur_j^k + post_j^k) v_{i+1,j} \qquad i < n, k \tag{3.25}$$

$$I_i^k = C_{i+2}^k - C_i^k - \sum_{j=1}^{n} (pre_j^k + sur_j^k + post_j^k) v_{i+2,j} \qquad i < n-1, k \tag{3.26}$$

$$f s^k = C_2^k - \sum_{j=1}^{n} (pre_j^k + sur_j^k + post_j^k) v_{2j} \qquad k \tag{3.27}$$

$$C_n^k - d_1 = O_1^k - g_1^k \qquad k \tag{3.28}$$

$$C_{n-1}^k - d_2 = O_2^k - g_2^k \qquad k \tag{3.29}$$

$$\sum_{j=1}^{n} v_{ij} = 1 \qquad i \tag{3.30}$$

$$\sum_{i=1}^{n} v_{ij} = 1 \qquad j \tag{3.31}$$

$$C_i^k, I_i^k, f s^k, O^k, g_1^k, g_2^k, W_i^k \geq 0 \qquad \forall i, k$$

$$v_{ij} \in \{0, 1\} \qquad \forall i, j$$

Indices and Sets

n procedures to be scheduled indexed by j

K scenarios to be considered indexed by k=1,....,K.

i indexes positions in the surgery sequence i=1,...,n.

Variables

The variable names are the same as in the extended formulation previously described. with the following exceptions:

C_i^k Completion time of procedure in position i in scenario k.

I_i^k Idle time between procedure in position i and position $i+2$ in scenario k.

W_i^k Surgeon waiting time for procedure in position $i+1$ in scenario k.

v_{ij} is a binary variable denoting the assignment of surgery j to position i in the sequence of surgeries.

fs^k is the idle time caused by the second surgery in scenario k.

Constraints

(3.24) Define the completion time for the first surgery.

(3.25),(3.26),(3.27) calculation of surgeon waiting time, OR idle time for the second surgery and OR idle time between surgeries for every scenario.

(3.28),(3.29) calculation of overtime for each OR and every scenario.

(3.30),(3.31) assignment constraints.

The surgery-sequencing sub-problem has symmetry if we consider equal overtime cost penalties in each OR, but this symmetry is easily broken by always assigning the first surgery to OR 1. We will search in the surgeon-OR sequence space (y) that has a cardinality of 2^{n-1}. Each possible surgeon-OR sequence (y vector) results in a surgery sequencing sub-problem in which the sequence of procedures must be determined.

3.4.1 Computation of the recourse function.

Once the first stage variables are fixed, the computation of the completion time for every surgery is easily found using the following formula executed from i=1 to n.

$$C^k_{jr} = \begin{cases} C^k_{ip(j),r} + pre^k_j + sur^k_j + post^k_j & \text{if } r(ip(j)) = r(j) \\ \max\{C^k_{ip(j),r} - post^k_{ip(j)} + sur^k_j + post^k_j, C^k_{rp(j),r} + pre^k_j + sur^k_j + post^k_j\} & \text{if r(i-1)} \neq \text{r(i)} \end{cases}$$

Where ip(j) is the immediate predecessor of surgery j, rp(j) is the immediate predecessor in the same OR of surgery j, and r(j) is the room assigned to surgery j. Given the completion times for every procedure we can compute the surgeon waiting time using the following formula

$$W^k_i = C^k_{i+1} - C^k_i + post^k_{ps(i)} - sur^k_{ps(i+1)} - post^k_{ps(i+1)}$$

3.4.2 Basic Parallel OR Scheduling Algorithm

The following decomposition algorithm uses bounds provided by the mean value problem.

Set the upper bound (UB) to ∞.

1. Initialize S as the finite set of all surgeon-OR sequences (with cardinality 2n-1).
   Initialize si as the zig-zag surgeon-OR sequence and remove from S.
   Solve the corresponding SAA MIP and initialize the UB.

2. Remove a new surgeon-OR sequence (s_i) from S if S is not empty, otherwise end.

3. Solve the LP relaxation of the mean value problem for (s_i). If the solution is less than the UB go to 3. Otherwise go to 2.

4. Solve the surgery sequencing sub-problem SAA for s_i. If the solution is less than the current UB update it, otherwise go to 2.

The idea behind this algorithm is that often the zig-zag surgery sequencing sub-problem will be optimal, or near optimal, while the vast majority of the 2^{n-1} surgeon-OR sequences will be terrible. Since the LP relaxation of the mean value problem provides an easily computable lower bound, it can be used to quickly eliminate all but a few surgeon-OR sequences. In the following sections we discuss enhancements to speed up the basic algorithm.

3.4.3 Alternative lower bounds

We next define an alternative lower bound that can eliminate entire subsets of surgeon-OR sequence sub-problems in the algorithm. First, define a partition of the set S as follows: let P_i be the set of all possible surgeon-OR sequences in which there is a maximum of i surgeries consecutive in the same OR. For example (0,0,0,1,1,0,1,0,1,0,1,1,1) belongs to P_3 since the maximum consecutive procedures in one OR is 3. This clearly partitions S since the union of P_i's give us S and the intersection of any P_is is empty. Next we define a lower bound based on these partitions. First, we define a lower bound for each cost component.

Idle time bound: We use order statistic notation to simplify the reading of the bounds. For instance, $\overline{post_{[j]}}$ means the jth smallest post value in the mean-value problem. L_i^I is the idle time lower bound for partition i. The bound is illustrated in figure 3.1.(In L_i^I the function $ind_{>2}$ takes the value 1 if i>2 and takes the value 0 otherwise)

$$L_i^I = (\overline{sur} + \overline{post})_{[1]} + (\overline{sur} + \overline{pre})_{[1]} - \overline{post_{[n]}} - \overline{pre_{[n]}} + ind_{>2} \sum_{j=1}^{i-2} (\overline{pre} + \overline{sur} + \overline{post})_{[j]}$$

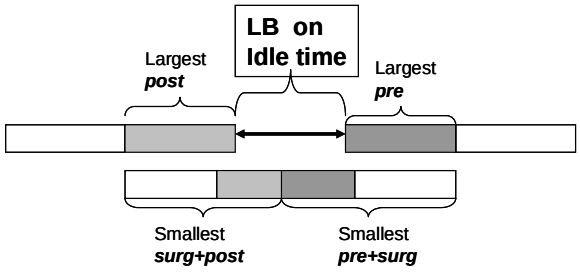

Figure 3.1: Illustration of Lower Bound on idle time for P_2

Waiting time bound: L_i^W is the wait time lower bound for partition i.

$$L_i^W = \sum_{j=1}^{i-1} \overline{pre}_{[j]} + \overline{post_{[j]}}$$

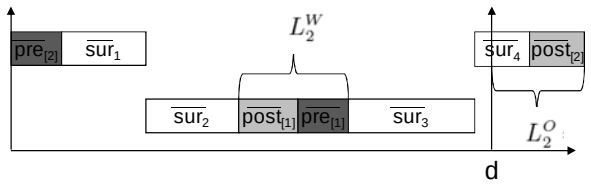

Figure 3.2: Illustration of Lower Bound on wait and over time for P_2

Overtime time bound: L_i^O is the overtime lower bound for partition i.

$$L_i^O = \max\left\{\sum_{i\in J}\overline{sur_j} + \sum_{j=1}^{i-1}\overline{pre}_{[j]} + \overline{post_{[j]}} - d, 0\right\}$$

$$L_i = c_i L_i^I + c_w L_i^W + c_o L_i^O$$

The idea is to find a lower bound for the mean value problem by finding the best case scenario in terms of idle time, waiting time and overtime. The mean value problem is a lower bound for the expected value problem, thus the new bound is as well. Thus we can use this easily computable bound to eliminate whole sets of OR-sequences. Indeed, when the lower bound L_i exceeds the current upper bound we can eliminate all surgeon-OR sequences belonging to P_i, P_{i+1}, up to P_n.

3.4.4 A Heuristic for Computing an Upper Bound.

In order to speed up the convergence of the algorithm we created a simple heuristic that seeks to find an improved initial upper bound (that is better than the solution to the zigzag problem) by spending additional computation time. This heuristic is based on Problem Search Space (Storer et al. [1992]). We perturb (using bootstrap sampling) the mean processing times in the mean value problem, solve the perturbed mean value problem to generate a new solution (sequence of surgeries and OR assignment), then evaluate the solution using the full sample average objective function. We generate 100 perturbed mean value problems and solutions, then keep the best upper bound found. Details of the perturbation algorithm are as follows:

PSS perturbed mean value heuristic

1. Generate K random numbers from IntUnif(1,K) and store the result in index i(h) (i(h)= IntUnif(1,K) for h=1,..,K).

2. Compute the bootstrap sample average setup, cleanup and surgery times for each surgery from the bootstrap sample using the following formula

$$\mu_t = \frac{1}{K}\sum_{h=1}^{K} t_{i(h)} \quad \text{t=\{setup, cleanup, surgery\}}$$

3. Construct the "perturbed mean value" problem using the bootstrap averages.

4. Solve the perturbed mean value problem and then evaluate the solution using the objective function with the original set of scenarios.

We apply this procedure 100 times and then we pick the best value found in step 4. In our computational experiments (section 3.5) we found that the gap between the optimal solution and the mean value solution (evaluated using the zigzag OR-sequence) was 70% on average. The gap between the optimal solution and the heuristic solution was 36%. Thus the heuristic ultimately saved significant computation time.

3.4.5 Solving the surgery sequence sub-problems

Two straightforward approaches for solving the surgery sequence sub-problems are branch and bound and Benders' decomposition. We conducted a brief experiment to find which factors dictate the choice of algorithms in terms of CPU time. For the Benders' decomposition approach we implemented the followings strategies:

Benders' Cuts

The method used to compute the Benders' cuts may have an impact on the processing time of the algorithm, so we developed ways to speed up their computation using (1) the complementary slackness theorem and (2) the structure of the basis matrix. The basis matrix is lower-triangular in the primal therefore it is upper-triangular in the dual. We can thus use simple recursive formulas to compute the dual variables.

Recycling Benders' Cuts

The sequencing decision in every subproblem has the same feasible region therefore we can store the solutions that we find in one subproblem, and use these solutions in other subproblems to generate cuts. By doing this we save time by not having to solve the Master problem, but we still have to find the dual in the new subproblem to find the cut. Of course the cuts generated from recycled solutions are not guaranteed to improve the solution. Because they are inexpensive to generate, it seemed worth trying. In the end, these cuts did not seem to improve the performance of the algorithm.

Generalized upper bound cuts

Since we have assignment constraints in the master problem we have set Cplex "generalized upper bound cuts" parameter to "intensive" in order to speed up the solution of the master problem.

Cut strengthening

We observed significant problems with degeneracy in certain instances. We therefore used methodology from Magnanti and Wong [1981] designed for such cases where they strengthen cuts using the concept of dominance. Assume that x* is the optimal solution of an integer program. We will say that (π, π_0) dominates (π^1, π_0^1) if

$$\pi x^* + \pi_0 \geq \pi^1 x^* + \pi_0^1$$

The idea behind these cuts is to find the dual variables that most improve the lower bound. Since, we have multiple solutions in the dual problem; we solve another linear problem that approximates the search for the dual variables that most improve the lower bound in the master problem. Since this strategy is costly, we only apply it every five iterations.

In applying this method, We observed degeneracy whenever we had idle time between surgeries (which occurred in virtually every case). When idle time is present, surgery starting times can be anywhere in a range defined by the beginning and ending of the idle period. The Magnanti and Wong method was developed specifically for cases like this where there are multiple choices for the dual variables.

Generalized upper bound (GUB) branching

The idea behind this branching rule is to take advantage of the assignment constraints. Instead of branching by variables we defined weights for variables that we will use to branch on with the fractional solution that results from the LP relaxation. This will lead to improved solution times according to Nemhauser and Wolsey [1999]. We implemented GUB using the SOS1 feature in Cplex 12.1. We use this feature for the master problem in Benders' and also in straightforward branch and bound.

3.5 Computational Experience

The proposed algorithm based on decomposing by surgeon-OR sequences will solve some problems very quickly, while others take longer. In general, when the zig-zag sequence (or a sequence close to zig-zag) is optimal, the algorithm solves quickly (we also note that we expect problems to behave this way in reality when parallel ORs are an attractive option). The experiment was designed to discover which factors affect the run time of the algorithm. The factors investigated are shown in table 3.1. The experiment had 5 replicates (and thus a total of 60 instances in the experiment). The stopping criteria for the algorithm was a 1% optimality gap. In order to clarify the effect of the distribution of set up and cleanup time with respect to surgery time we created the "setup to surgery time ratio" B. This factor has two levels; "high" means that the summation of the means of setup and cleanup times is greater than 1.5 the summation of the surgery times, while "low" means the opposite.

$$B = \frac{\sum_{j=1}^{n}\sum_{k=1}^{K} sur_j^k}{\sum_{j=1}^{n}\sum_{k=1}^{K} pre_j^k + post_j^k}$$

We also defined "cost ratio" as waiting cost divided by idle cost. The cost ratio is set to low (value=1) and high (value=2) similar to Batun et al. [2011].

Table 3.1: Experiment Design

Factor	Possible Value
Number of Scenarios	10, 50, 100
B Parameter	High , Low
Cost ratio	High , Low

In order to construct instances that reflect the complexity of real problems we gathered data from our industry partner on surgeons that performed their surgeries in parallel ORs (most commonly orthopedic and eye surgeries). Next we constructed empirical distributions for setup, surgery and cleanup times based on the type of surgery and the surgeon. Unfortunately we were not able to get sufficient data at the level of individual surgeons and procedures to directly generate (from data) sufficient scenarios to run our experiments. We next took data for all surgeons and surgeries provided by the hospital and fit a regression model between the mean and the standard deviation for setup, surgery and cleanup times. We utilized log-normal distributions to model the surgery time as suggested in (May, 2000). We also used log-normal distributions to model setup and cleanup times because we observed long-tails in the histograms constructed from real data. To create Log-Normal distribution models for each random variable, we took the average duration from the empirical data, and then used the regression model to get a standard deviation. Scenarios were then generated from these Log-normal distributions.

When the setup to surgery time ratio is high, we found that the parallel OR scheduling problem becomes much more difficult to solve. We also looked carefully at the performance of our Benders' decomposition based algorithm for solving the surgery sequence

sub-problem.

Interestingly, despite the enhancements discussed in section 4, the Benders' algorithm was not faster than Cplex branch and cut. The Cplex settings we used were: (1) generalizing upper bound branching, (2) mipemphasis = emphasize moving best bound and (3) the initial starting algorithm was "barrier". The precise reasons for the disappointing results of the Benders' algorithm are not entirely clear. The behavior we observed was that as the algorithm progresses, each cut will eliminate the current solution but no others thus leading to slow progress. We further note that Schultz [2003] discusses that classical stochastic programming methods seem to have great difficulty on stochastic sequencing and scheduling problems, and cites this as an area of future research.

Table 3.2: Computational Results (average processing time in seconds)

Factors	Scenarios		
B Parameter, Cost Ratio	10	50	100
High , Low	278	2007	6146
High , High	240	1599	4844
Low, Low	188	1724	5887
Low, High	146	1554	4403

We see in table 3.2 that experiments with low cost and high B parameter take more time to solve.

3.5.1 Comparison of stochastic solution versus common sense heuristic.

Next, we present the percentage optimality gap for the solution obtained using the mean value problem in zig zag OR Sequence. We can observe that the savings in term of objective function might justify the time employed solving the stochastic version of the problem. We also can notice that the heuristic proposed improved in average 18% the solution when we have 100 scenarios.

Table 3.3: Optimality gap for the mean value solution with zigzag OR Sequence.

Factors	Scenarios		
B Parameter, Cost Ratio	10	50	100
High , Low	104%	44%	33%
High , High	109%	55%	44%
Low , Low	124%	55%	45%
Low , High	115%	56%	50%

Table 3.4: Optimality gap for the perturbed mean value solution with zigzag OR sequence.

Factors	Scenarios		
B Parameter, Cost Ratio	10	50	100
High , Low	52%	28%	23%
High , High	56%	31%	29%
Low, Low	56%	28%	23%
Low, High	56%	29%	25%

3.5.2 Zig-Zag is not always optimal

In the scheduling literature it is pointed out that the zig-zag sequence is not always optimal Abdekhodaee et al. [2006]. To illustrate this, we perfumed a new set of experiments in order to see if similar observations can be made when we have random processing times. We generated test instances in the same fashion as Abdekhodaee et al. [2006]. Their problem involves one server and two machines and two processing step, setup and processing. By analogy to our OR scheduling problem, their setup time is our surgery time and their processing time is our cleanup time. The mean surgery times where generated from U(0,30) and clean up times from U(0,60). In order to generate the scenarios we use the regression equations that we found previously to generate standard deviations and we used log-normal distribution. We generated 5 problem instances with 100 scenarios and we tried 2 cost ratios. The average percentage improvement in table 5 below represents the savings gained by using the proposed algorithm with respect to zig-zag OR with optimized sequence.

In table 3.5 we observe that the improvement over zig-zag are 14% and 30% for the two cost ratios.

Table 3.5: Zig-zag OR Sequence comparison with optimal solution.

Cost ratio	Average processing time (sec)	Average percentage improvement
0.1666	2546	30%
1	3556.8	14%

3.6 Choosing the number of scenarios

In order to determine an appropriate number of scenarios, we applied the method proposed by Linderoth et al. [2006]. The main idea is to generate several samples of scenarios that allow us to compute statistical lower and upper bounds on the true (infinite scenario) objective function value. The confidence interval on these estimates is strictly related to the number of samples used. Once we have computed these bounds for each set of scenarios, we can compute a statistical gap. The basic trade off is that as we increase the number of scenarios we will decrease the gap. The following plot shows how the gap decreases as we increase the number of scenarios. We worked with 2 instances both having the same number of surgeries and the same set of scenarios, but with different cost ratios (2 and 8). We generated 10 replicates of sets of scenarios with the same surgery, cleanup and setup duration distributions and two different cost ratios to see if the cost ratio had an effect on the gap. In figure 10 we observe that the cost ratio did not have a large effect with 50 or more scenarios. Further we see, that for this instance, 100 scenarios is enough to achieve a 5% statistical gap (Linderoth et al. [2006]).

3.7 Managerial Insight

In this section we conduct experiments that provide insight about parallel OR scheduling and when it is a viable alternative. Our interest is in further exploring the tradeoff between surgeon waiting time and OR idle time when we schedule in parallel ORs (rather than a single OR). This tradeoff will depend on the relative costs of surgeon waiting time and OR idle time and on the setup to surgery time ratio B. Based on data collected from our

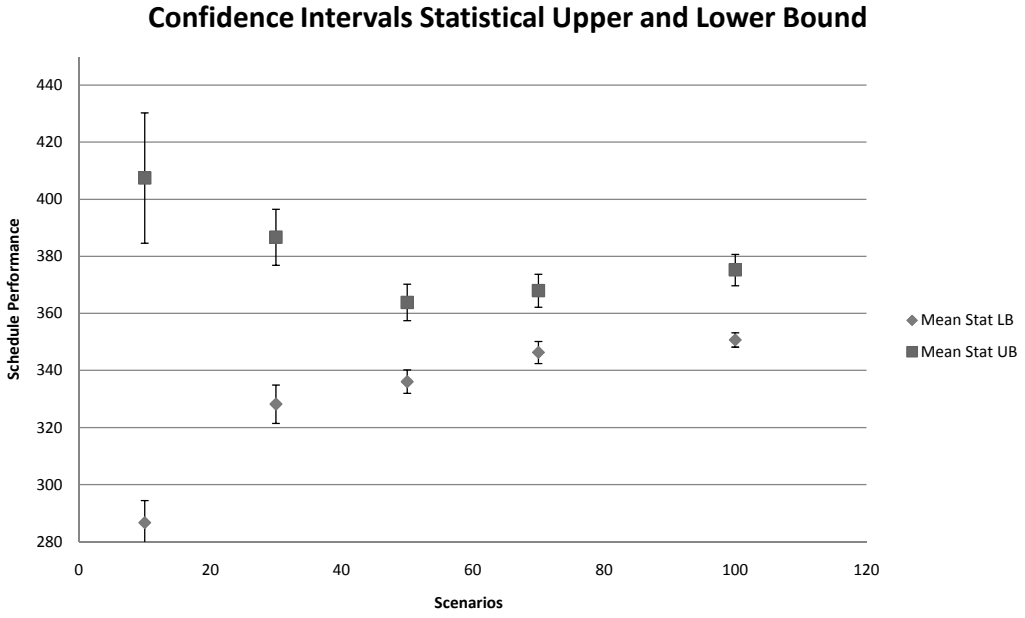

Figure 3.3: Tradeoff between number of scenarios and statistical gap.

industry partner the B factor typically ranges from 0.5 to 1.5 in cases where parallel ORs were used. We generated problem instances with varying B factor to investigate its affects. We started with a base problem with n=10 procedures and K=100 scenarios. The means and variances were generated as described in section 5. Let , be the setup, surgery and cleanup times for procedure j in scenario k. In order to generate problems with varying B we altered these as follows:

Let pre_j^k , sur_j^k and $post_j^k$ be the original setup, surgery, and cleanup times for a particular surgery j in scenario k.

Let D be the total duration $(pre_j^k + sur_j^k + post_j^k)$

Let C = $\frac{pre_j^k}{post_j^k}$

Let B be the desired ratio.

Let G, H, P be the new setup, surgery , and cleanup times that have the desired B

We have three equations:

G+H+P=D (total duration must remain the same)

$\frac{G}{P}$ = C (the pre to post ratio must remain the same)

$\frac{H}{G+P}$ = B (to get the desired B ratio)

These equations can be easily solved to yield:

$$G = \frac{CD}{BC + C + B + 1}$$

$$H = \frac{BD(C + 1)}{BC + C + B + 1}$$

$$P = \frac{D}{BC + C + B + 1}$$

Next we solved the parallel OR scheduling problem for values of B from 0.5 to 3.32.

We assumed that any idle time in either OR throughout an 8 hour work day is penalized. The reason for this assumption is that we wanted to penalize the total idle time in both ORs. This assumption was necessary for a somewhat esoteric reason. Our first thought was to stop charging for idle time once surgeries are complete in an (either) OR. With this scheme, we frequently obtained solutions where only the second surgery was performed in OR 2 while all other surgeries were performed in OR 1. This type of solution occurred in cases where using a single OR was clearly more efficient. In these cases, scheduling the second surgery in OR 2 was cheaper than scheduling all surgeries in OR 1 since we did not charge for idle time after surgeries in OR 2 are complete. To avoid this problem, we charged for idle time up to 8 hours which is roughly when surgeries should be completed if both OR's are fully utilized throughout the schedule.

In the first experiment we assume that surgeon waiting cost is equal to OR idle cost, and recorded the idle time and waiting time for each value of B. For purposes of comparison, we also computed surgeon waiting time in the case where all procedures are scheduled in a single OR. Note that with a single OR, the idle time is always zero. The results appear in Figure 3.4.

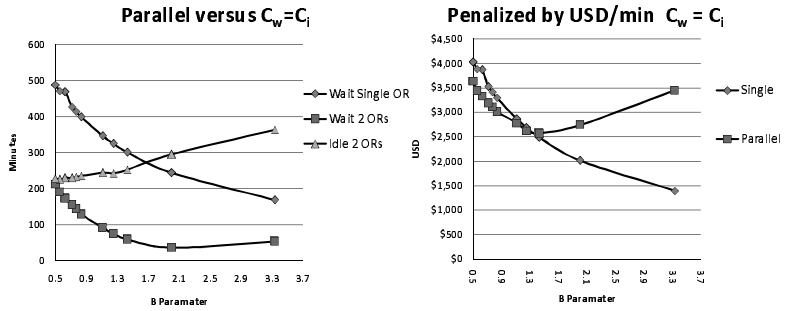

Figure 3.4: Comparison between single and parallel ORs, cost ratio 1

Next we repeated the experiment, with the assumption that surgeon waiting time is twice as expensive as OR idle time (cost ratio=2). The results appear in Figure 3.5.

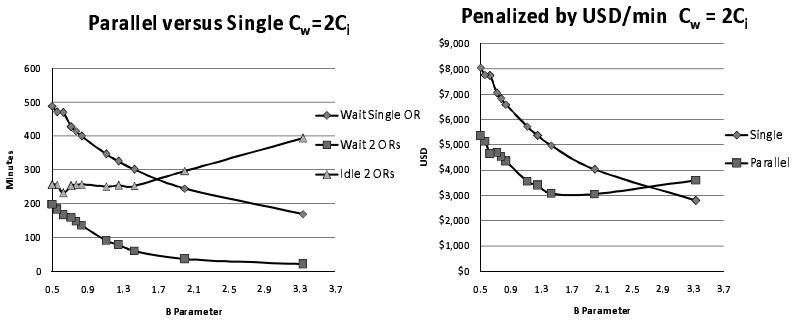

Figure 3.5: Comparison between single and parallel, cost ratio 2

In figure 3.4 we observe a cutoff value for the B parameter after which single ORs

becomes cost effective (B= 1.43). This value changes with changing cost ratio. In figure 3.5 we can observe that in this case B is 2.7 .

Clearly the viability of parallel OR scheduling depends on the ratio B for a given set of surgeries and on the cost ratio c_w/c_i. In the next experiment we attempted to quantify when parallel OR scheduling is viable. We tested a problem with 100 scenarios and 10 surgeries used in section 5. We next fixed the cost ratio c_w/c_i to a constant, then varied B until we found the value of B for which the cost of a single OR and parallel ORs was equal. This gave us one point on one of the curve in Figure 3.6 below. By varying the cost ratio c_w/c_i and repeating, we generated one full curve in Figure 3.6. Given the cost ratio for a particular hospital, one can use the graph below to find values of B for which parallel OR scheduling appears attractive.

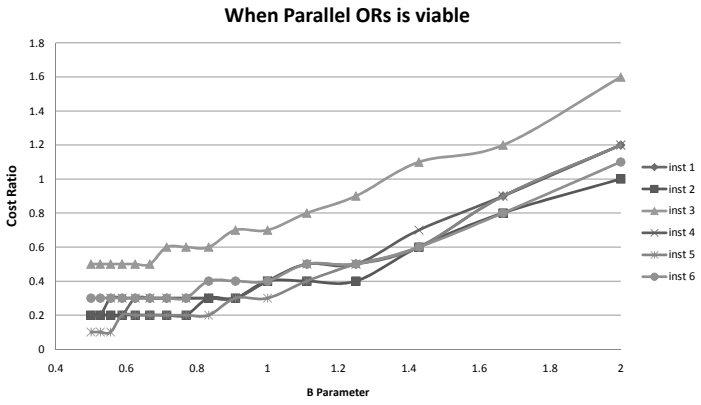

Figure 3.6: When to use parallel scheduling as a function of cost ratio and B parameter.

3.7.1 Estimating the cost Ratio

Finding a realistic value for the ratio of surgeon waiting time cost to OR idle time cost seems to be problematic. Hospitals seem not to have this information available, and various authors have suggested different methods for estimating it. Batun et al. [2011]

estimated the cost ratio based on the relative cost of opening a new OR in terms of staff cost, they assumed two cases: 50 minutes of surgeon waiting time is equivalent to opening another OR and the other case was that 250 minutes of surgeon with these two cases they can compute a penalty in USD for every minute of surgeon waiting. Here we propose a new method to estimate this ratio based on an analysis of historical data. We examined actual outcomes of surgical blocks in both single OR and parallel OR cases. Assuming the decision to assign the block to either a single or parallel ORs to be rational, we can use this data to compute bounds on the underlying cost ratio.

When the OR scheduler decides to perform surgeries in a single OR, we assume he or she did so because it is less expensive than using parallel OR's.

Under this assumption the following relationship holds (where w_1 and s_1 refer to waiting and idle time in a single OR and w_2 and s_2 refer to waiting and idle time in parallel ORs).

$$c_w w_1 + c_i s_1 < c_w w_2 + c_i s_2 \Rightarrow \frac{s_2 - s_1}{w_1 - w_2} > \frac{c_w}{c_i}$$

For the cases where the block was performed in parallel ORs we obtain, by a similar argument, a lower bound in the cost ratio.

$$c_w w_1 + c_i s_1 > c_w w_2 + c_i s_2 \Rightarrow \frac{s_2 - s_1}{w_1 - w_2} < \frac{c_w}{c_i}$$

Given a single OR case, we compute s_2 and w_2 by assuming the same surgery sequence and the zigzag OR-sequence. Given a parallel OR case, we compute s_1=0 and w_1 by assuming that the surgeries with the largest setup and cleanup times are at the beginning and in the end positions in the surgery sequence respectively (note s_1=0 in the single OR case). The following plot presents the computation of the cost ratio bounds for all surgical blocks.

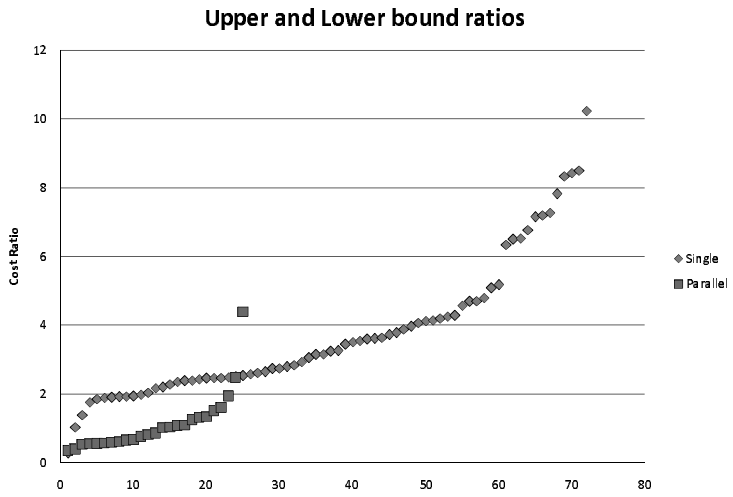

Figure 3.7: Upper and Lower bounds in the cost ratio based on historical decisions.

From this plot it appears that the implied cost ratio in our hospital is around 2. Of course this ratio may vary from hospital to hospital.

3.8 Conclusions

In this paper we developed an algorithm for sequencing surgeries for a single surgeon operating in parallel operating rooms under uncertainty. This decomposition based algorithm solves problems with sufficiently small run times for implementation on real size problems.

We also provide an upper bounding heuristic that can be used to generate reasonable solutions when fast computation is required. This methodology may also be useful in other application areas such as manufacturing and network computing. In particular, the lower bounds that were developed could be very useful when the number of jobs increases as one might expect in the other application areas. Using the algorithm to conduct experiments allowed us to develop insight regarding when the parallel OR approach is viable, what the basic cost tradeoffs are, and how one can measure the (typically unknown) implied cost ratio between surgeon waiting cost and OR idle time cost.

Chapter 4

Stochastic integer program based algorithms for adaptable open block surgery scheduling

4.1 Introduction

The daily schedule for a hospital's operating rooms is typically finalized the day prior to surgery. This schedule includes the operating room (OR) to which each surgery is assigned and its planned starting time. In many hospitals, surgeons perform all of their surgeries for the day in a single operating room. One disadvantage of this approach is that the surgeon is idle during the setup phase before, and the cleanup phase after each surgery. To avoid excessive surgeon idle time, many hospitals have moved to an "open block" scheduling approach in which the surgeons move between different ORs to perform their surgeries. If scheduled carefully, open block scheduling can significantly cut surgeon idle time Batun et al. [2011]. In this paper we address open block scheduling via stochastic programming assuming that the durations of each surgery, setup, and clean are random variables with known distributions.

Rescheduling. In practice the daily surgery schedule is routinely changed during execution as the actual durations of surgeries are observed and as emergency surgeries and other potential disruptions arise. One common way to revise the schedule during execution, or "reschedule", is to simply continue with the original OR assignments and surgery sequence but delay the starting times as required. In this paper we assume that surgeries cannot be started earlier than specified in the original schedule because the patient will not be available for surgery (it may be helpful to think of the scheduled starting time as the earliest possible starting time). The simply policy that maintains the original OR assignments and sequences but delays surgeries as needed to accomodate disruptions is called the "right shift" rescheduling policy. An alternative to the simple "right shift" rescheduling policy could be a policy that reassigns surgeries to different ORs by removing the surgery from its scheduled OR and inserting it into the sequence of another OR. Note that the starting time cannot by earlier than the scheduled starting time that was established the day before and reflects when the patient will arrive and be ready for surgery. This more complicated rescheduling policy must include details specifying how to change the schedule based on the current status of the system. Moving a surgery to a different OR may also have an additional cost associated due to communication, transportation, staffing, etc. This new schedule can be generated through a re-optimization procedure from a local or global perspective. The set of actions executed to produce revised schedules during execution are called rescheduling policies. For a thorough review of the rescheduling literature please see Vieira et al. [2003].

Previous literature on stochastic surgery scheduling implicitly assumed a "right shift" rescheduling policy Batun et al. [2011], Mancilla and Storer [2009], Mancilla and Storer [2010], Denton et al. [2010]. All of these papers use two-stage stochastic integer programming to find initial schedules. In the first stage, the initial schedule is computed so as to minimize expected cost. In the second stage, new start and finish times are computed for

each surgery by simply delaying (or "right shifting") starting times as necessary to accommodate the actual observed processing times in each scenario. The advantage of assuming a "right shift" rescheduling policy is that the recourse in stage two is easily computable. If the only action allowed during schedule execution is to delay starting times, then these methods provide schedules that are robust in that the uncertainty in processing times is explicitly considered in the optimization. However an initial schedule developed under the assumption of a naive and unrealistic rescheduling policy (e.g. right shifting) may perform poorly if a more complex rescheduling policy is in use.

Adaptable Scheduling. The problem of finding an optimal initial schedule given (1) processing time distributions and (2) a specified rescheduling policy has been previously called the robust scheduling problem Wu et al. [1993]. We have found that the term "robust scheduling" causes significant confusion. The field of "robust optimization" in which one seeks solutions to optimization problems that are insensitive to inaccuracies in problem data is well established. Thus robust scheduling is often misconstrued as finding schedules that are similarly insensitive. For this reason we prefer to use the term "adaptable scheduling". In adaptable scheduling the initial schedule must take into account what rescheduling policy is being used. Previous research in stochastic scheduling typically assumes a simple "right shift" rescheduling policy when building the initial schedule. In this paper we consider finding an initial schedule under more complex rescheduling policies.

In adaptable scheduling one would ideally incorporate the more complex and realistic rescheduling policy into the recourse. The result will be a detailed multi-stage stochastic integer program (MSSP) with a complex optimization problem to be solved at each stage. The schedule found in stage one would be the "adaptable schedule" and the decisions found at subsequent stages would give the optimal rescheduling policy over the various scenarios. However, solving this multi-stage stochastic program (MSSP) is clearly not possible due to its complexity.

4.2 Problem definition

In operating room scheduling, the following problem is common. There is a given set of surgeons each with a given set of surgeries to perform on a given day. The OR Manager would like to decide, the day before surgery, how to schedule these surgeries in an open block scheduling fashion. He/she must decide:

- The sequence in which each surgeon will perform his/her set of surgeries (in some cases this decision is pre-specified by the surgeon, in other cases the sequence may be decided by the scheduler).
- The sequence of surgeries in each OR.
- The planned (earliest) starting time for each surgery.

Since this initial schedule will undoubtedly change during execution, the OR manager is looking for an initial adaptable schedule that minimizes the overall cost at the end of the day given that rescheduling will almost surely occur. In this paper, costs considered include (1) idle time of surgeons between surgeries, (2) patient waiting time between the scheduled and actual start time, (3) idle time of the operating rooms and staff, (4) overtime of the OR staff and (5) the number of OR room assignment changes made during rescheduling. Further it is assumed that each surgery is composed of three phases: (1) setup, (2) surgery, and (3) clean-up. The distribution of the duration of each phase is assumed known, and the presence of the surgeon is only required during the surgery phase. Further, each phase of each surgery has a unique distribution.

We begin by describing the full multi-stage stochastic program (MSSP) formulation. We then approximate the MSSP with a two-stage stochastic integer program where the first stage decisions provide the adaptable schedule and the second stage decisions approximate the rescheduling policies. In the following, we show a simple example that provides intuition as to why this approach might work, then we explain the approximation and finally we propose a method to explicitly test its performance.

4.2.1 Motivating Example

Assume that we have to schedule 3 surgeries in 2 OR's and that we have only 3 scenarios as shown in Figure 4.1. In Figure 4.1 we show two different schedules. On the schedule A surgery 3 (red) is scheduled in OR 2. On the schedule B surgery 3 is scheduled in OR 1. In both schedules surgery 3 is initially scheduled to start at 1:00. For each schedule, the left figure shows what will happen when right shift is applied after realizing the outcomes in each scenario (note that surgery 3 must remain in its original OR under right shift rescheduling). Under a right shift rescheduling policy, then the optimal schedule will be the schedule A (surgery 3 in OR 2). Note that the average delay for the schedule A is (0+20+20)/3 (thus the expected waiting cost is $13.33c^w$) while the average delay for the bottom schedule is (50+0+0)/3 (thus the expected waiting cost is $16.67c^w$) where c^w is the per unit time cost of patient waiting. However, if we treat this problem as a multi-stage stochastic problem, the final assignment of surgery 3 to an OR can be scenario dependent. The two right hand schedules show the final optimal schedule for each scenario under the assumption that we can move surgery 3 to a new OR. Once moving surgery 3 to another OR is allowed we see that in the schedule A we would move surgery 3 to OR 1 in both scenarios 2 and 3. In the schedule B surgery 3 would be moved to OR 2 in scenario 1. After moving surgeries in this fashion, there will be no delay in the starting times of surgery 3 in either schedule in all scenarios (and thus the patient waiting cost will be zero in both cases). However, there may be costs associated with moving surgeries to another room. Letting c^c represent the cost of moving a surgery to another room, we see that the top schedule incurs two moves (scenarios 2 and 3) while the bottom schedule only requires one move (scenario 1) and thus is the better choice under this more complex rescheduling policy. (Note: this analysis requires that c^c is much less than c^w to be precise). The key point of this example is that the optimal initial schedule is different under the right shift rescheduling policy than under a more complex rescheduling policy in which shifting ORs is allowed. Thus an initial schedule should take into account both the uncertainty in

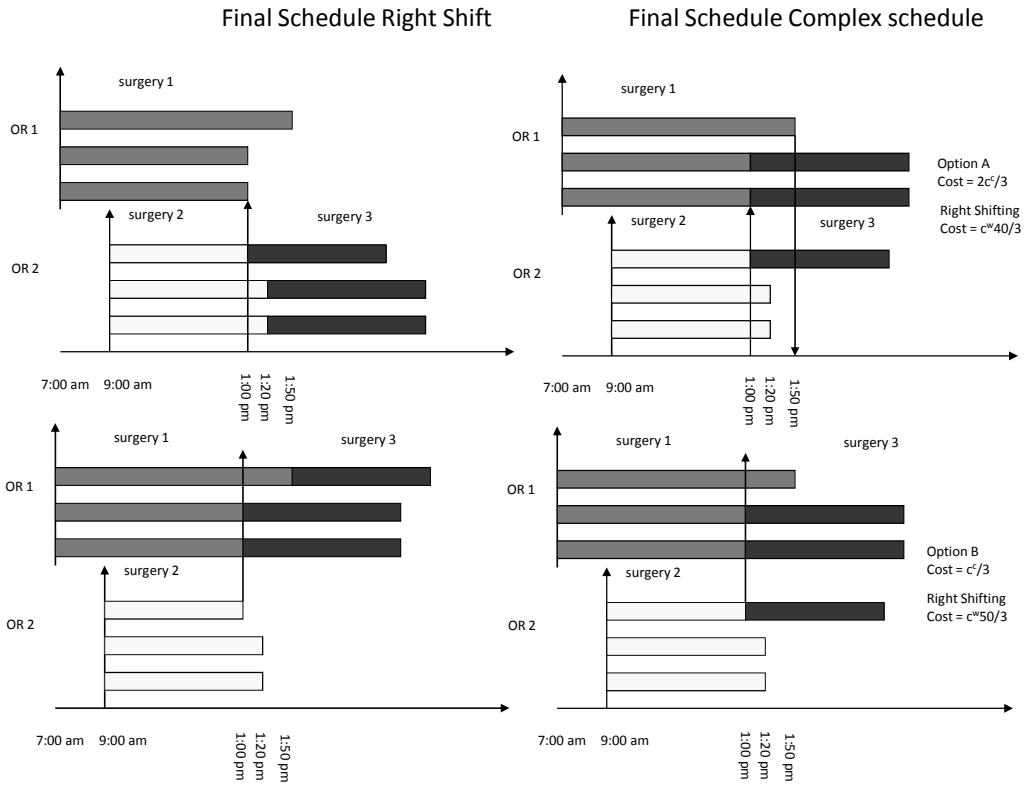

Figure 4.1: Small example.

processing times and the rescheduling policy being used as the schedule is executed.

In the simple example the optimal solution is option B, and we could have figured this out by first computing the waiting times for every scenario after applying the rescheduling policy, and then counting the number of changes (move surgery 3 from OR 1 to OR 2 or vice versa).

4.2.2 Multiple Stage Integer Programming Formulation

In this section we describe a multi-stage stochastic integer programming model of the OR scheduling - rescheduling decision process discussed above. In this model the stages are defined by discrete points in time where we observe the system (for instance, every 30 minutes) and revise the schedule (using the rescheduling policy). In this model the decision variables are:

First stage decision variables:

- The surgery sequence for each surgeon.
- The initial sequence of surgeries in each OR.
- The scheduled starting time of each surgery.

Subsequent stage decision variables:

- The revised sequence of surgeries in each OR.

In each stage after stage one, and for every branch in the scenario tree, a new sequence of surgeries for each OR is computed. This OR is the result of observing the state of the system at the current stage and then solving a multi-stage stochastic IP from "now" until the end.

Note that the surgery sequence for each surgeon and the scheduled starting times for each surgery are decided and fixed in the first stage. This is the part of the schedule that must be decided ahead of time (e.g. the day before) so that the patients know when to report to the hospital. In subsequent stages, after observing outcomes up to the current stage, the OR manager may decide to move one or more surgeries to different ORs. He or she must also decide where in the OR-sequence this surgery should be inserted. Also note that the surgery cannot start before its original scheduled starting time as decided in stage 1.

We assume that each stage is separated by 30 minutes. That is, every 30 minutes, we observe the state of the system and make rescheduling decisions that may move surgeries to different ORs and positions in the OR sequence. As mentioned previously, when a surgery is moved, a fixed cost penalty is assessed.

Formulating this problem in the form of a multi-stage stochastic integer program is extremely complicated because the state definitions depend on the entire path and all decisions made up until the current stage. Thus for example, one would need variables with 20 indexes to define the system state in stage 20. Further, solving this problem to optimality is problematic due to the extremely rapid explosion of the state space (i.e. curse of dimensionality). In order to define the multi-stage problem, we provide the following multi-stage formulation using a general form provided in Shapiro et al. [2009],

$$\min_{x_0, x_1, \ldots x_T} \mathrm{E}\left[f_0(x_0) + f_1(x_1(\Xi_{[1]}), \Xi_1) + \ldots + f_T(x_T(\Xi_{[T]}), \Xi_T)\right]$$

$$\text{s.t.} x_0 \in X_0, x_i(\Xi_{[i]}) \in X_i(x_{i-1}(\Xi_{[i-1]}), \Xi_i), \text{i=2,..T}$$

Where x_0 are the first stage decisions: surgery sequence for each surgeon, initial sequence of surgeries in each OR and scheduled starting time of each surgery. The $x_1, \ldots x_T$ are the decisions executed after time 0, (in our case we make a decision every 30 minutes therefore the index can be translated to time by simply multiplying by 30 minutes). These decisions are the revised sequence of surgeries in each OR. The $\Xi's$ represent the random variables that in our case are the surgery durations. $\Xi_{[1]}$ represents the surgeries that have been executed or are in progress up to the end stage 1. The Ξ_i are the surgeries that will be executed in the future and the uncertain portion of the surgeries that are being executed at the end of stage i-1. The functions $f_1, .., f_T$ are the cost incurred between every stage. The sets X_0 and X_i represent the possible actions in every stage. For instance, in stage i we cannot move to another OR the surgeries that started before stage i.

In the next section we formulated a two-stage approximation based on relaxing the non-anticipativity constraints in the multi-stage problem. We conclude that even this (much simplified) approximation to the multi-stage problem is extremely difficult to solve, thus requiring further simplification. For example a problem with 2 surgeons, 3 surgeries per surgeon, 3 ORs, and 10 scenarios could not be solved in over five days of computation.

4.2.3 Approximation to the multi-stage stochastic problem

The mathematical formulation of the two stage approximation is given below. We call this the SAPIA (sample average perfect information approximation) problem. This model is a two stage stochastic program with integer variables in each stage. We note here that even this two stage approximation is difficult to solve in reasonable time and we will later describe further approximations required to produce solutions in reasonable time.

$$\min z_A = \sum_{k\in K} \frac{1}{K} \left(\sum_{i\in I_s, s\in S} c^w w_{is}^k + \sum_{j\in J} c^{pw} pw_j^k + \sum_{r\in R} c^s s_r^k + \sum_{r\in R} c^o O^k + \sum_{i\in I, s\in S, r\in R} c^c u_{isr}^k \right)$$

s.t.	$x_{1jr}^0(pre_j^k + post_j^k + sur_j^k) \le C_{jr}^k$	j,k,r	(4.2)
	$-M(2 - x_{i-1sr}^k - x_{iqr}^k) + C_{sr}^k + (pre_q^k + post_q^k + sur_q^k) \le C_{qr}^k$	$s,q,r,k,i>2$	(4.3)
	$-M(3 - y_{iq}^s - y_{i+1g}^s - \sum_{m\le n} x_{mgr}) + \sum_{r\in R} C_{qr}^k - post_q^k + sp_g^k \le C_{gr}^k$	g,q,r	(4.4)
	$-M(2 - y_{iq}^s - y_{i+1g}^s) - \sum_{r\in R} C_{qr}^k + post_q^k + \sum_{r\in R} C_{gr}^k - sp_g^k \le w_{is}^k$	$k, i<n_s, q,g,s$	(4.5)
	$C_{jr}^k - d_r \le O_r^k$	k	(4.6)
	$st_j + pw_j^k \ge C_j^k - (pre_j^k + sur_j^k + post_j^k) - M(1 - y_{pj})$	j,p	(4.7)
	$st_j \le C_j^k - (pre_j^k + sur_j^k + post_j^k) + M(1 - y_{pj})$	j,p	(4.8)
	$C_{jr}^k \le M \sum_{i\le n} x_{ijr}$	j,r,k	(4.9)
	$\sum_{p\in I_s} y_{pj}^s = 1$	j,s	(4.10)
	$\sum_{j\in J_s} y_{pj}^s = 1$	p,s	(4.11)
	$x_{irs}^0 - x_{irs}^k \le u_{irs}^k$	i,s,r,k	(4.12)
	$x_{irs}^k - x_{irs}^0 \le u_{irs}^k$	i,s,r,k	(4.13)
	$\sum_{s\in S} x_{irs}^0 \le 1$	i,r	(4.14)
	$\sum_{r\in R} x_{irs}^0 \le 1$	i,s	(4.15)
	$\sum_{r\in R} \sum_{i\in I_s} x_{irs}^0 = \lvert I_s \rvert$	s	(4.16)
	$\sum_{s\in S} x_{irs}^k \le 1$	i,r,k	(4.17)
	$\sum_{r\in R} x_{irs}^k \le 1$	i,s,k	(4.18)
	$\sum_{r\in R} \sum_{i\in I_s} x_{irs}^0 = \lvert I_s \rvert$	s,k	(4.19)
	$x_{ijr} \in \{0,1\}\ \forall (i,j,r) \in (J,J,R)$		

Indices and Sets

n is the number of procedures to be scheduled

n_s is the number of procedures for surgeon s

K is the number of scenarios

j indexes the set of procedures: from 1,...,n.

I is the set of positions for the OR sequence.

J's indexes the procedures for surgeon s.

I's indexes the positions for surgery sequence for surgeon s.

p indexes the position in the surgery sequence for a particular surgeon.

r indexes the set operating rooms: r=1,..,R

k indexes scenarios: k=1,....,K.

Parameters

c^w surgeon idle time cost per unit time.

c_{pw} patient waiting time cost per unit time.

c^o overtime cost per unit time.

c^s idle time cost per unit time.

c^c cost of changing one OR assignment.

d_r time beyond which overtime is incurred in O.R. r

M is sufficiently large "big M" number.

pre_j^k duration of setup for procedure j in scenario k.

sur_j^k duration of surgery for procedure j in scenario k.

$post_j^k$ duration of cleanup for procedure j in scenario k.

sp_j^k sumation of surgery and cleanup for j in scenario k.

Variables

W_{is}^k Surgeon waiting time before the procedure in position i of the sequence for surgeon s in scenario k.

pw_j^k patient waiting time for surgery j in scenario k.

I_r^k Total OR idle time in scenario k for OR r.

C_{jr}^k Completion time of procedure j in OR r in scenario k.

O_r^k Overtime time in OR r in scenario k.

x_{irs}^k a binary variable denoting the assignment of surgeon s in position i in OR r in scenario k.

x_{irs}^0 a binary variable denoting the assignment of surgery j in position i in OR r in the original schedule (first stage variable).

y_{pj}^s a binary variable denoting the assignment of surgery j in position p in surgeon sequence s.

u_{irs}^k a binary variable denoting the number of differences between the final OR assignment in scenario k and the original OR schedule x^0.

Constraints

(4.2), (4.3) define the completion time based on OR sequences.

(4.4) define completion time based on surgery sequence, the second surgery of surgeon s should start after the first surgery is completed.

(4.5) computes the waiting time for every surgeon.

(4.6) computes the overtime for every OR.

(4.7) computes the patient waiting time.

(4.8) guarantees that every surgery should start after its scheduled starting time.

(4.9) sets to zero the completion time of surgery j if this was not assigned to OR r.

(4.10), (4.11) assignment constraints that guarantees that every surgery is assigned to one OR.

(4.12), (4.13) assignment constraints that guarantees that every surgery in the surgery sequence gets assigned.

(4.14), (4.15) computes the OR schedule changes relative to the original schedule defined

at time 0.

This problem is a two-stage stochastic integer program where the first stage and the second stage have both continuous and binary variables. During experimentation we discovered that the problem behaves differently based on the value of the "cost changing ORs".

Case 1: The high cost of changing ORs (right shift) case:

When the cost of changing ORs is high enough, the OR manager will never change the initial OR assignments. This is equivalent to adopting the simpler "right shift" rescheduling policy. Thus solving the two stage problem with "high cost of changing ORs" will produce the optimal initial schedule under the right shift rescheduling policy. In this case the initial OR schedule is of critical importance since no future changes are possible. To solve this problem we use an approach similar to that proposed by Mancilla and Storer [2010] where the problem is decomposed by OR sequence. The idea is to solve a subset of problems defined by the OR sequence in each OR. Once we have fixed the OR sequence for every OR we will have a stochastic MIP that will search for the best surgery sequence for every surgeon. This sub-problem is very manageable for Cplex as it was found in Mancilla and Storer [2010]. To clarify, we define vector a^r as the "OR sequence for OR r". For instance, if ar is (A,B,C,C,0,0,0) then the first surgery in room r is from surgeon A, the second from surgeon B, then 2 from surgeon C. The search space of the vectors a^r is large (there are(S!)I possible OR sequences) but we can take advantage of various fast lower bounding techniques to quickly eliminate all but a few viable OR sequences. The basic outline of the algorithm is the following,

OR-Sequence decomposition algorithm

Set the upper bound (UB) to ∞ .

1. Initialize S as the finite set of all OR sequences.

   Initialize si as a promising OR sequence and remove from S.

   Solve the corresponding SAA MIP and initialize the UB.

2. Remove a new OR sequence (si) from S if S is not empty, otherwise end.

3. Solve the LP relaxation of the mean value problem for (si). If the solution is less than the UB go to 4. Otherwise go to 2.

4. Solve the sequencing sub-problem SAA for si. If the solution is less than the current UB update it, otherwise go to 2.

The idea behind this algorithm is that only a few sub-problem solutions will be close to optimal, while the vast majority of the OR sequences will be terrible. Since the LP relaxation of the mean value problem provides an easily computable lower bound, it can be used to very quickly eliminate all but a few OR sequences. In this decomposition approach we assume that the maximum number of surgeries per OR is 4 in order to minimize the number of possible OR sequences.

Initial Testing for Case 1

As was found in Mancilla and Storer [2010] the ratio between surgeries and setup and clean up times has an impact on the processing time of the algorithm. We define a parameter Bs for each surgeon s as follows:

$$B = \frac{\sum_{j=1}^{n} \sum_{k=1}^{K} sur_j^k}{\sum_{j=1}^{n} \sum_{k=1}^{K} pre_j^k + post_j^k}$$

Note that we only consider surgeries that belong to surgeon s in this equation. If, for example, the Bs parameter for a particular surgeon s is one, then it would be reasonable to have 2 ORs available since the surgeon is only required for half the time (on average) of each surgery. The parameters Bs are one of the factors we investigate in this computational experiment. Other factors include number of scenarios and the cost ratio as shown in the table below. The cost ratio is defined as surgeon waiting cost divided by OR idle cost. Test problems all consisted of 3 surgeons each with 4 surgeries scheduled in 4 ORs. Duration distributions were all lognormal with means and variances created as discussed in Mancilla and Storer [2010]. We ran 5 different sets of data. The "high cost ratio" corresponds to

a cost ratio of 2 and "low cost ratio" is 1. We say that the Bs parameters are "similar" when the coefficient of variation (computed based on the distributions for each surgeon) is below 0.1 and high otherwise. Table 13 shows the algorithm run time in seconds for the various problems tested. Processing times vary in the 1 to 3 hour range. The run time increases with the number of scenarios as one would expect and decrease when the cost ratio is high. The processing time decreases when the cost ratio is high is because the bound obtained by the mean value problem is tighter.

Table 4.1: Experiment Design

Factor	Possible Value
Number of Scenarios	100 , 150
B Parameter	Similar, Different
Cost ratio	High , Low

Table 4.2: Processing time in seconds

Factors	Scenarios	
B Parameter, Cost Ratio	100	150
Similar , Low	5568	9259
Similar , High	4890	10352
Different, Low	4433	11449
Different, High	3563	8682

Table 4.2 shows the time required to solve each of the cases in our experiment.

Case 2: Low cost of changing ORs (complex rescheduling) case:
When the "cost of changing ORs" is low, the OR manager will periodically observe the execution of the schedule, and will make rescheduling adjustments in order to diminish the impact of disruptions. Thus it is possible that every scenario may end up with a different OR sequence after the OR manager has executed the rescheduling policy. The mathematical formulation of our approximate problem (SAPIA) is a two-stage stochastic integer program with integer recourse. Since we have integer recourse it might be reasonable to try to solve this problem using decomposition techniques. We tried to use scenario decomposition Caroe and Schultz [1997] early on and found two problems. First the iteration

time was too expensive. Each iteration took 30 minutes because each scenario subproblem was a fairly difficult integer program. Further, the solutions were disappointing even when we let the method run for 4 hours. Mainly because of the time required per iteration, it seemed very unlikely that progressive hedging, or any other method based on scenario decomposition, could possibly produce good results in a reasonable amount of time.

Next we decided to divide the problem into different steps. The first step is the selection of each surgeon's surgery sequence. Once the surgery sequence for each surgeon is fixed we next compute the initial OR sequence and finally the scheduled starting times. The outline of this procedure is as follows:

Surgery-OR-Starting time decomposition algorithm

1. Find an initial surgery sequence for each surgeon (could be obtained from the high cost of change case, or from sort by variance, or from surgeon preferences, etc).

2. Solve every scenario independently in order to find the optimal final OR sequence for each scenario.

3. Solve the Initial "average OR-sequence" problem (see below) to get the initial OR-Sequence.

4. Compute the starting times (using the optimal starting time algorithm explained below) assuming the final OR-assignment for every scenario found in step 3.

The idea behind this heuristic is to find a good (adaptable) OR-sequence and starting times given the sequence for each surgeon. We will also investigate different ways to assign the surgeon's surgery sequence and report which methods work best. In the next section we describe how we find the initial "average OR-sequence" problem.

Initial average OR-Sequence.

Once we have fixed each surgeon's surgery sequence we compute the optimal OR-sequence for every scenario by solving a simple integer programming problem. We may end up with

as many as k different OR-sequences therefore we need a procedure to find a single initial OR-sequence given the optimal OR-sequence for each scenario. That is, we seek an initial OR sequence that is in some sense close to the various OR sequences in each scenario. We propose to solve the following optimization problem,

$$\min Z^c = \frac{1}{K} \sum_{k \in K} \sum_{i \in I} \sum_{r \in OR} \sum_{s \in S} c^c u^k_{irs}$$

$$\begin{aligned}
& x^0_{irs} - x^k_{irs} \leq u^k_{irs} && i, s, r, k \\
& x^k_{irs} - x^0_{irs} \leq u^k_{irs} && i, s, r, k \\
& \sum_{s \in S} x^0_{irs} \leq 1 && i, r \\
& \sum_{r \in OR} x^0_{irs} \leq 1 && i, s \\
& \sum_{r \in OR} \sum_{i \in I_s} x^0_{irs} = |I_s| && s \\
& \sum_{s \in S} x^k_{irs} \leq 1 && i, r, k \\
& \sum_{r \in OR} x^k_{irs} \leq 1 && i, s, k \\
& \sum_{r \in OR} \sum_{i \in I_s} x^0_{irs} = |I_s| && s, k \\
& x^k_{irs}, x^0_{irs} \in \{0, 1\}
\end{aligned}$$

The variables used in this problem are the same those we define in SAPIA with the addition of x^k variables that are obtained by solving the k scenarios problems separately. The idea is simply to find the initial OR sequence that minimizes the total number of changes over the k scenarios assuming that the k optimal scenario based OR sequences approximate the final schedule of the real problem (after rescheduling).

Optimal starting times given the OR-sequence. Once we have fixed (1) the surgery-sequence for each surgeon, (2) The (stage 1) initial OR sequence for each OR, and (3) the (stage 2) OR-sequences for every scenario (that is, all the integer variables are fixed) the SAPIA becomes a two stage stochastic linear programming problem that can be solved fairly quickly for the initial scheduled starting times. Once we have computed the starting

times the objective function can be easily computed from simple linear equations. For example the patient waiting time is given by:

$$pw_j^k = C_j^k - pre_j^k - sur_j^k - post_j^k - st_j^*$$

Other components of the objective function are computed in a similar way.

Further justification of the two-stage approximation In this section we present two results that help understand why the SAPIA approximation problem might be useful in finding promising solutions for the multi-stage problem.

Claim 1. *The objective function value of the optimal solution to the SAPIA approximation problem (z_S) is a lower bound to the optimal objective function value to the problem under a right-shift rescheduling policy (z_{RS}). ($z_S \leq z_{RS}$)*

Proof. This is true because the solution to problem RS is one of many possible solutions to the SAPIA problem. Problem RS is a special case of the SAPIA problem in which no rescheduling moves are allowed.

□

Claim 2. *The optimal objective function value of the (SAPIA) approximation problem is a lower bound for the multi-stage objective function zMS. Further, ($z_S \leq z_{MS} \leq z_{RS}$)*

Proof.

1. ($z_S \leq z_{MS}$) because SAPIA is a relaxation of the multi-stage problem.
2. ($z_{MS} \leq z_{RS}$) because the right shift solution is a special case of the multistage problem.

□

4.3 Algorithm Summary

In this section we present a summary of the models we have developed to help clarify the situation.

Multi-stage model: Our first model was a multi-stage stochastic integer program that finds the sequence of surgeries for each surgeon, the OR sequence for each OR, and the starting times for each surgery. The multi-stage model also incorporates a complex rescheduling policy in which some random durations are observed at each stage and an optimization algorithm is run that allows surgeries to be moved to other ORs during schedule execution (and inserted into that OR's sequence). There is little hope of solving this model to optimality.

Right shift rescheduling special case of the multistage model: If we assume the simple right shift rescheduling policy, the multi-stage model can be reduced to a 2-stage model. By extending the decomposition methods from our previous work on parallel OR scheduling (Mancilla and Storer [2010]), we can solve this problem to optimality for a reasonable number of scenarios.

SAPIA (sample average perfect information approximation) model: This model includes the more complex rescheduling policy but relaxes the non-anticipativity constraints in the multi-stage model. The result is a 2-stage approximation to the multi-stage model. In the first stage (representing decisions required the day prior to surgery) we must specify each surgeon's sequence, the scheduled starting time of each surgery, and an initial OR sequence for each OR. In the second stage, we assume all random variable outcomes are revealed, and that we are allowed to change the OR sequences (but not the surgeon sequences or starting times). However each change made to the OR sequence in stage 2 incurs a fixed cost (the "cost of changing ORs"). This model is very difficult to solve in reasonable time.

SAPIA-surgeon sequences fixed model (SAPIA-SSF): If we assume the sequence of surgeries for each surgeon is known in advance, the SAPIA problem simplifies. In many cases surgeons prefer to set the sequence themselves based on procedure difficulty, patient preference, etc. We may also be able to test various ways to produce surgeon sequences in a first step, then solve for OR sequences and starting times using this model. Even after

fixing the surgeon sequences, this SAPIA-SSF problem is still difficult to solve to optimality. We developed an approximate method based on scenario decomposition. We first found optimal (but different) OR sequences for each scenario separately. We then solved an optimization problem that finds the "average" initial OR sequence. This "average" initial OR sequence is the one that minimizes the total number of OR changes required to get to the final OR sequence in each scenario.

Optimal starting time model: Once both the surgeons sequences and initial OR sequences are determined, we must still find the scheduled starting time for each surgery. When both surgeon and OR sequences are fixed, the SAPIA reduces to a two stage stochastic linear programming problem that can be solved quickly by Cplex to produce optimal starting times for each surgery.

4.4 Evaluating the solutions by simulation

In order to evaluate the quality of the initial schedules the proposed methods produce, we develop a simulated environment that simulates the initial schedule under the more complex rescheduling policy. Given distributions of setup, cleanup and surgery times and the various cost factors, our algorithm produces initial surgeon sequences, OR sequences and starting times. We then evaluate this "adaptable" schedule's performance using the simulation of the more complex rescheduling policy. The simulation simulates the actions of the OR manager by observing the current state of the system every 30 (simulated) minutes and then solving a rescheduling optimization problem which gives a revised schedule. The simulation thus estimates the final cost after schedule execution and rescheduling. The simulation is repeated many times and the average total cost is reported. Obviously the processing time scenarios used in the simulation are random, and are different (independently generated) from those used to solve the original stochastic scheduling problem. Next we explain how we constructed this simulated environment in more detail.

The disruptions that we will consider are:

- Duration of surgery, setup and cleanup times.
- Emergency arrivals (indirectly).

The simulation of the dynamic (complex) rescheduling policy requires us to define the interval at which we observe the system, and how we will perform the rescheduling each period given the updated information on the state of the system. We assume that we observe the schedule execution (i.e. state of the system) every 30 minutes, and make a rescheduling decision by solving an optimization problem as defined below. When the system is observed at time t, we assume the duration of surgeries (or setups or cleanups) completed before t are known with certainty and that the distribution of surgeries that have not started by time t are those given in the problem statement. For durations started before t but not yet completed, we use the conditional distribution at time t given the initial distribution and actual starting time. As discussed previously, the actual optimization problem we would like to solve at each period is a multi-stage stochastic program (from period t to the end). In our simulation we approximate this using the mean value problem. For durations currently unknown, we simply plug in (conditional) distribution means. The result is an integer program that can be solved reasonably quickly. Solving this problem produces a revised OR-sequence for each OR at time t for future surgeries. These OR sequences are followed up to the next decision epoch. We also save the revised OR sequences made at each time period and use them to compute final costs. There is an additional important detail in this computation. Suppose a surgery is moved from OR 1 to OR 2 at time t = 30, then moved back to its original spot in OR 1 at time t = 60. In this case we do not charge an "OR change cost". Further, suppose a surgery is moved from OR 1 to OR 2 at time t=30, then moved from OR 2 to OR 3 at time t = 60, and that the surgery is ultimately executed as planned in OR 3. In this case we charge for only a single "OR change".

"We approximate emergency arrivals, not with additional surgeries to be scheduled,

but by assuming that they contribute to the duration of an already scheduled surgery. Bimodal distributions for the scheduled surgeries are used for this purpose." The idea is to construct this new distribution based on the probability that a particular surgery is interrupted by an emergency arrival. The assumption is that the probability of being interrupted increases as the surgery mean increase. Suppose that on a given day that we expect an average of λ emergency arrivals per hour. We then compute the probability that surgery j will be interrupted by an emergency as a function of its mean duration MS_j. Assuming Poisson arrivals, the probability that surgery j is interrupted is = 1 - exp($\lambda \frac{MS_j}{N}$) where N is the number of ORs. Finally we can construct a bimodal distribution where the mixture coefficient is the probability of being interrupted, the first unimodal distribution is the surgery distribution and the second unimodal distribution is the sum of the scheduled plus emergency surgery durations. In our experiments we assume emergencies arrive at a rate of one every two hours.

Finally we note that we also simulate initial schedules under the right shift rescheduling policy again using independently generated scenarios. This simulation is a straightforward exercise.

4.5 Computational Experience

In this section we present experiments designed to evaluate the scheduling methods discussed previously. In the following list we summarize these experiments, then provide more detail on the experiments followed by results. We created 4 problem instances for testing.

- We test 5 different methods for creating the initial schedule (which includes surgeon sequences, initial OR sequences and starting times)
- Each of the five methods is evaluated by simulation under both the simple right shift and the complex rescheduling policies.

- For each of the five methods and both rescheduling policies, evaluations are made at many different levels of the "cost of changing ORs" parameter.

Problem Instances: All four problem instances assume 3 surgeons, 4 surgeries per surgeon and 4 ORs. Log-normal distributions for setup, cleanup and surgery times were generated based on historical data obtained from our industry partner using the method described in Mancilla and Storer [2010]. In the first instance we assumed that the cost of patient waiting time was very low (0.01 USD/minute) compared to the other cost coefficients which were: surgeon idle time (3 USD/minute), OR staff idle time (3 USD/minute), and overtime (4.5 USD/minute). The second problem instance was the same as the first except that the cost of patient waiting time was increased by an order of magnitude to 0.1 USD/minute. The third problem instance included emergency arrivals modeled by the bimodal distributions described previously and the cost of patient waiting time was set to 5 USD/minute. The fourth problem instance included emergency arrivals and set the cost of patient waiting time to 0.01 USD/minute.

Scheduling Methods. We implemented five different scheduling methods as described in this section. The methods were selected in an attempt to compare our ideas to methods similar to what a typical OR manager might do. These methods are used to generate the initial schedule (initial surgeon sequences, OR sequences and scheduled starting times). These initial schedules are then evaluated by the simulation discussed previously.

- **Case 1: "RS" Scheduling.**As discussed previously, if we assume a right shift rescheduling policy, the multi stage problem reduces to a two stage problem that we can solve using our decomposition approach. In this case we use this method to generate the initial surgeon sequences, OR sequences, and starting times.

- **Case 2: "SBV-Opt" scheduling.** In this case we first find the sequence for each surgeon using the sort by variance heuristic (sequenced from smallest to largest variance). Next we determine the sequence in each OR using our SAPIA-SSF model and "average sequence" approximation algorithm. Finally we find the starting times by solving the optimal starting time algorithm (the two-stage stochastic LP that results when both the surgeon and OR sequences are fixed in the SAPIA problem).

- **Case 3: "SBV-Manager".** This is an attempt to schedule in a manner similar to what an OR manager might do "by hand". First the surgeon sequences are set by the sort by variance heuristic as in case 2. The OR sequences are then found by solving the mean value relaxation of the SAPIA-SSF. Given the surgeon sequences and the mean values for each surgery one can imagine an OR manager trying to fit the surgeries into a Gantt chart. This is our attempt at approximating an OR manager moving these "puzzle pieces" around in a Gantt chart to find a good OR sequence. Once the OR sequence is set we also used a more ad hoc method to find the surgery starting times (although we could have solved the 2 stage stochastic LP as in case 2). Here we imagine that each surgeon's cases were all scheduled in a single OR. We then solve for the starting times using the single OR scheduling method described in Mancilla and Storer [2009].

- **Case 4: "SBM-Opt".**The only difference between case 4 and case 2 is that we set the sequence for each surgeon by sorting by mean surgery time from largest to smallest. Since many surgeons prefer to schedule their more complex surgeries first, this represents what might happen when surgeons decide their own sequence.

- **Case 5: "SBM-Manager".** The only difference between case 5 and case 3 is that we set the sequence for each surgeon by sorting by mean surgery time from largest to smallest.

Evaluating Schedules by Simulation. We evaluated each of the five methods by simulating the schedules under both the simple right shift and more complex rescheduling policies. To simulate the more complex policy, we used the method described in section 4.4. We used 250 replicates of this simulation making sure that the scenarios in the simulation were generated independently from those used in the algorithms. To simulate performance under the simple right shift policy we used the same 250 scenarios. Note that in the three figures below in which results are presented, one can easily distinguish between cases where right shift and complex rescheduling are used. When there is no change in performance over the various levels of "cost of changing ORs" (i.e. a horizontal line) we know that right shift rescheduling is being used.

Cost of Changing ORs. For each combination of problem instance, scheduling method, and rescheduling policy we used 9 different "cost of changing ORs" parameter values. The value of the more complex rescheduling policy depends to a large extent on how easy and cheap it is to move surgeries around during schedule execution. We used values of 0, 15, 30, 45, 60, 75, and 90 USD as the cost of moving a single surgery to another OR.

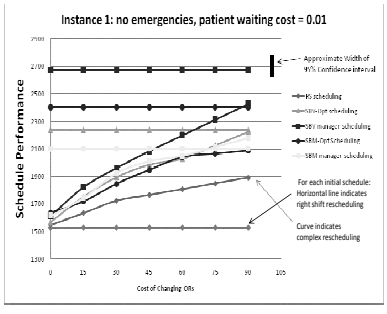

Figure 4.2: Benchmark non emergency presence.

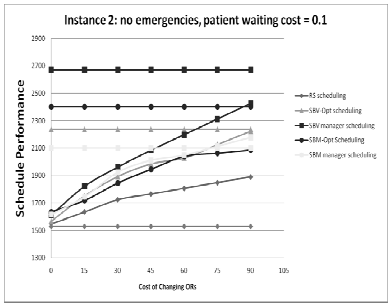

Figure 4.3: Benchmark non emergency presence, patient waiting cost 0.1.

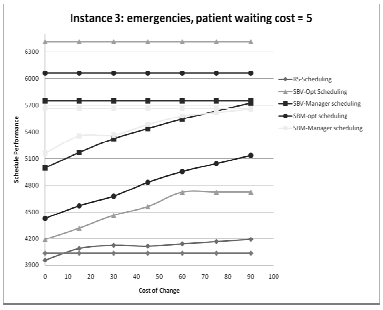

Figure 4.4: Benchmark of different initial schedules in presence of emergencies, high patient cost.

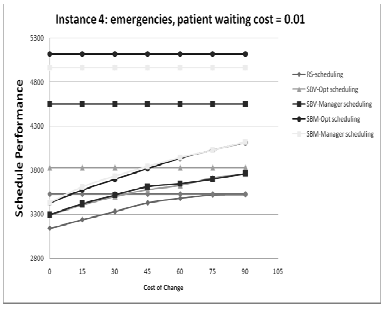

Figure 4.5: Benchmark of different initial schedules in presence of emergencies, low patient cost.

Based on the results presented in the figures above, we can draw some conclusions.

- Under the right shift rescheduling policy, the initial schedule is very important. In this case using the proposed method to find an initial schedule can cut cost by up to 50%.
- If sub-optimal or arbitrary methods are used to find the initial surgeon sequences, one can still achieve much better performance by implementing the more complex rescheduling policy, especially when the cost of changing ORs is low. In the non-emergency cases with low cost of changing ORs, the initial schedule appears to be unimportant since complex rescheduling can "fix the problems" in the initial schedule.
- As cost of changing increases, and in the case with emergencies, there is benefit to using the better (adaptable) initial scheduling methods even under complex rescheduling.
- When the patient waiting cost is low, scheduled starting times tend to be earlier reserving more time and flexibility. Thus in this case, we observe greater benefit from the more complex rescheduling policy.
- Given that surgeon sequences are specified in advance, there appears to be an advantage to using the proposed methods for finding the OR sequences and starting times over the "manager" methods meant to approximate what an OR manager might do by hand. The advantage is clearest in the case with emergencies, and when right shift rescheduling is used.
- There is no clear winner between sort by variance and sort by mean for this problem.
- By far the best initial scheduling method in our experiments was right shift scheduling. This method produces an initial schedule under the assumption of right shift rescheduling. Given this initial schedule, the more complex rescheduling policy was

not helpful unless the cost of changing ORs was zero. Even in this case the benefit was marginal. Looking at this in a different way, if one uses our right shift method to produce the entire initial schedule, there appears to be little need for the more complex rescheduling procedure.

- If one were able to solve the full blown multi-stage stochastic program, one could certainly do better, and benefit from the more complex rescheduling policy. However, this problem seems to be quite intractable.

4.6 Conclusions

In this paper we use stochastic programming methods to investigate adaptable scheduling in the case of open block scheduling in operating rooms. The adaptable scheduling problem, first addressed in the 1990's and called "robust scheduling" seeks good initial schedules under the assumption that rescheduling policies more complex than simple right shifting may be in use. We first formulated the problem as a multi-stage stochastic integer problem and then proposed approximation models and algorithms that help find reasonable solutions for adaptable scheduling. To solve the adaptable scheduling problem under simple right shift rescheduling we proposed a new decomposition algorithm and were able to solve the problem to optimality on small problems (but realistically sized in the realm of OR scheduling). We also created a new rescheduling policy based on solving mean value versions of the multi-stage problem every 30 minutes. We then implemented a simulation model with this rescheduling policy embedded, and used it to test various scheduling methods. The main conclusions seem to be that (1) building near optimal initial schedules under the assumption of right shift rescheduling seems to produce excellent initial schedules, and (2) When sub optimal methods are used to build the initial schedule (e.g. when each surgeon specifies his/her own sequence), significant benefit can be achieved be using the more complex rescheduling policy.

Chapter 5

Conclusions

5.1 Conclusions

The research problems presented in this thesis are of significant relevance. From an operational and tactical perspective, operating rooms are one of the main drivers of hospital activity and therefore improved efficiencies will have a large, system-wide impact. Furthermore, greater efficiency will increase elective capacity and therefore have a significant impact on costs related to capacity investment and strategic planning. The following are the principal contributions of this study,

- Develop optimization methods for real size problems in surgical suites under uncertainly.
- Propose new optimization methods for stochastic scheduling problems.
- Develop optimization methods for scheduling in dynamic stochastic environments (adaptable schedule).

The new optimization methods proposed have been developed based on known stochastic programming techniques combined with heuristic methods to allow timely solutions. It is reasonable to think that they can be used in many other applications areas.

From an academic point of view, these problems involve cutting-edge research in Operations Research applied to Health Care Management. The methodologies to be developed may also be applied to similar problems faced in health care (e.g. primary care clinics, outpatient procedure centers, etc.) and also other areas like project scheduling, container vessel and terminal operations, airport gate and runway schedules, extending their impact.

Bibliography

K. Aardal and T. Larsson. A benders decomposition based heuristic for the hierarchical production planning problem. *European Journal of Operational Research*, 45(1):4 – 14, 1990.

A. H. Abdekhodaee and A. Wirth. Scheduling parallel machines with a single server some solvable cases and heuristics. *Computers Operation Research*, 29:295 – 313, 2002.

A. H. Abdekhodaee, A. Wirth, and H. S. Gan. Scheduling two parallel machines with a single server: the general case. *Computers Operation Research*, 33:994 – 1009, 2006.

S. Batun, B. T. Denton, T. R. Huschka, and A. J. Schaefer. The benefit of pooling operating rooms and parallel surgery processing under uncertainty. *To apper in Informs Journal on Computing*, 2011.

C. C. Caroe and R. Schultz. Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24:37–45, 1997.

G. Cote and M. A. Laughton. Large-scale mixed integer programming: Benders-type heuristics. *European Journal of Operational Research*, 16(3):327–333, 1984.

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

B. Denton, J. Viapiano, and A. Vogl. Optimization of surgery sequencing and scheduling decisions under uncertainty. *Health Care Management Science*, 10:13–24, 2007.

B. Denton, A. Miller, H. Balasubramanian, and T. Huschka. Optimal allocation of surgery blocks to operating rooms under uncertainty. *Operation Research*, 58(4), 2010.

M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Math. of Operations Research*, 1:117–129, 1976.

C. Glass, Y. M. Shafransky, and V. A. Strusevich. Scheduling for parallel dedicated machines with a single server. *Naval Research Logistics*, 47 (1):304–328, 2009.

D. Gupta. Surgical suites operations management. *Production and Operations Management*, 16 (6):689–700, 2007.

HFMA. Achieving operating room efficiency through process integration. *Health Care Financial Management Association Report*, 2005.

G. C. Kaandorp and G. Koole. Optimal outpatient appointment scheduling. *Health Care Manage Science*, 10:217–229, 2007.

A. Kleywegt, A. Shapiro, and T. H. de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12:479–502, 2002.

Q. Kong, C.-Y. Lee, C.-P. Teo, and Z. Zheng. Scheduling arrivals to a stochastic service delivery system using copositive cones. *working paper*, 2010.

C. P. Koulamas. Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers Operation Research*, 23 (10):945–956, 1996.

G. Laporte and F. V. Louveaux. The integer l-shaped method for stochastic integer programs with complete recourse. *Operations research letters*, 13:133–142, 1993.

J. T. Linderoth, A. Shapiro, and S. J. Wright. The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research*, 142:219–245, 2006.

T. L. Magnanti and R. T. Wong. Accelerating benders' decomposition:algorithmic enhancement and model selection criteria. *Operations Research*, 29:464–484, 1981.

C. Mancilla and R. H. Storer. Stochastic sequencing and scheduling an operation room. Technical report, Industrial and System Engineering, Lehigh University., 2009.

C. Mancilla and R. H. Storer. Stochastic sequencing in parallel ors. Technical report, Industrial and System Engineering, Lehigh University., 2010.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1999.

L. W. Robinson and R. R. Chen. Scheduling doctors appointments: Optimal and empirically-based heuristic policies. *IIE Transactions*, 35:295–307, 2003.

R. Schultz. Stochastic programming with integer variables. *Mathemitical Programming*, 97:285–309, 2003.

A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on Stochastic Programming: Modeling and Theory*. SIAM, 2009.

R. H. Storer, D. S. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38 (10):1495 – 1509, 1992.

P. M. Vanden Bosch and D. C. Dietz. Minimizing expected waiting in a medical appointment system. *IIE Transactions*, 32:841–848, 2000.

P. M. Vanden Bosch and D. C. Dietz. Scheduling and sequencing arrivals to an appointment system. *Journal of Service Research*, 4 (1):1525, 2001.

G. Vieira, J. Herrmann, and E. Lin. Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, 6:39–62, 2003.

P. P. Wang. Optimally scheduling n customer arrival times for a single-server system. *Computers & Operations Research*, 24(8):703 – 716, 1997.

E. Weiss. Models for determining the estimated start times and case orderings. *IIE Transactions*, 22 (2):143–150, 1990.

D. S. Wu, R. H. Storer, and P.-C. Chang. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operation Research*, 20(1):1–14, 1993.

Biography

Name:	Camilo Mancilla.
Place of birth:	Vallenar, Chile.
Date of birth:	August 04, 1976.
Parents:	Mrs. Nuris Silva and Mr. Washington Mancilla.

Education

- Ph.D., Industrial Engineering, Lehigh University, Bethlehem, PA, August 2006 – August 2011.
- Degree of Industrial Engineering (Equivalent of a Masters Degree) (Hons.), University of Chile, Santiago, Chile, Aug 1998 – Mar 2000.
- Bachelor Degree in Engineering Sciences Major: Industrial Engineering, Mar 1994 – Aug 1998

Professional Experience

- Principal Process Engineer, Airproducts and Chemicals, Bethlehem, PA, May 2011 – Present.
- Project Manager, SCM Consulting, Santiago, Chile, Jan 2005 – Jul 2006.
- IT Project Leader , Consorcio Nacional de Seguros Vida, Santiago, Chile, May 2001 – Dic 2004.

BIBLIOGRAPHY

Publications

- C. Mancilla and R. H. Storer, Stochastic Sequencing and Scheduling of an Operating Room, submitted, 2009.

Talks and Presentations

- C. Mancilla and R. H. Storer, Network Simplex-like Algorithms for Stochastic Appointment Scheduling, INFORMS Annual Conference, Washington DC, October, 2008.

- C. Mancilla and R. H. Storer, Stochastic Sequencing and Scheduling of an Operating Room, Bethlehem PA, August, 2009.

Memberships and Professional Activities

- Member, Institute for Operations Research and the Management Sciences (INFORMS) and INFORMS Health Care.