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

Camilo Mancilla

Robert H. Storer

Lehigh University

Abstract: We develop algorithms for a stochastic two-machine single-server sequencing problem with waiting time, idle time and overtime costs. Scheduling surgeries for a single surgeon operating in two parallel operating rooms (ORs) motivates the work. The basic idea is that staff perform cleanup and setup in one OR while the surgeon is operating in the other. The benefit is less waiting time for the surgeon between surgeries, but may also result in added idle time for staff if cleanup and set-up are completed prior to completion of surgery in the other OR. When surgeries are long relative to cleanup and setup times, parallel OR scheduling is not attractive as significant OR staff idle time will result. The problem we address consists of assigning surgeries to ORs and sequencing them, and is formulated as an integer stochastic program using sample average approximation. A decomposition based solution approach is developed. Computational testing based on real data shows that the proposed methods solve the problems to optimality in acceptable processing times. Using the solution methodology, we further provide insight as to the conditions when parallel operating rooms is cost effective.

Key words: Scheduling, Stochastic Programming, Health Care Management

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, the model is described and formulated. In Section 4 algorithms are proposed to solve the sequencing and assignment problems. In Section 5 computational results are presented. In Section 6 we present computational results with respect to the number of scenarios needed to solve the problem. In Section 7 we provided insight as to when parallel operating rooms is cost effective. In Section 8 conclusions and future research directions are discussed.

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. (2010) 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), Abdekhodaeea et al. (2006), Abdekhodaee and Wirth (2002). 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., (2000). 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. (2000) 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. Abdekhodaeea 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. Mathematical Formulation of the Sample Average Parallel Operating Room Scheduling (SAPOS) Problem

In this section we formulate a stochastic integer programming model for the problem described above. 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 OR's. 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 $\vec{X} \text{ and } \overrightarrow{M}$. 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 $\vec{X}\ and\ \overrightarrow{M}$, we have “simple recourse”. Finally we can compute the average (over scenarios) cost. The problem formulation appears below.

$$\text{minimize: } \sum_{k=1}^{K} \frac{1}{K}\left(\sum_{i\neq j} c^w W_{ij}^k + \sum_{r\in R} c^I I_r^k + \sum_{r\in R} c_r^l O_r^k\right) \quad (1)$$

s.t.:

$$\begin{array}{lll}
x_{ij} + x_{ji} = 1 & i,j \in J & (2)\\
x_{ij} + x_{je} + x_{ei} \leq 2 & 1 \leq i \neq j \neq e \leq n & (3)\\
a_{ijr} \leq x_{ij} & i,j,r & (4)\\
\sum_{r\in R} m_{ir} = 1 & i & (5)\\
a_{ijr} + a_{jir} \leq m_{ir} & i, j > i, r & (6)\\
a_{ijr} + a_{jir} \leq m_{jr} & i, j > i, r & (7)\\
a_{ijr} + a_{jir} \leq m_{ir} + m_{jr} - 1 & i, j > i, r & (8)\\
\sum_{t\neq i} x_{ti} + 1 = \sum_{u=1}^{n} u q_{iu}^1 & i & (9)\\
\sum_{u=1}^{n} q_{iu}^1 = 1 & \forall i & (10)\\
q_{ij} = \sum_{u=1}^{n-1} uq_{iju} & \forall i, j \neq i & (11)\\
uq_{iju} \leq q_{iu+1}^1 & \forall i, j \neq i, t < n & (12)\\
uq_{iju} \leq q_{ju}^1 & \forall i, j \neq i, t < n & (13)\\
uq_{iju} \geq q_{ju}^1 + q_{iu+1}^1 - 1 & \forall i, j \neq i, t < n & (14)\\
C_i^k \leq M_1 m_{ir} & \forall i, r & (15)\\
W_{ij}^k \geq \sum_{r\in R} C_{jr}^k - \sum_{r\in R} C_{ir}^k + post_i^k - sur_j^k - post_j^k - M_1(1 - q_{ij}) & \forall i, j \neq i, k & (16)\\
W_{ij}^k \leq \sum_{r\in R} C_{jr}^k - \sum_{r\in R} C_{ir}^k + post_i^k - sur_j^k - post_j^k + M_1(1 - q_{ij}) & \forall i, j \neq i, k & (17)\\
\sum_{r\in R} C_{jr}^k \geq \sum_{r\in R} C_{ir}^k + pre_j^k + sur_j^k + post_j^k - M_1(1 - a_{ijr}) & \forall i, j \neq i, k & (18)\\
C_{ir}^k \geq pre_i^k + sur_i^k + post_i^k - M_1(1 - m_{ir}) & \forall i, r, k & (19)\\
C_{ir}^k \leq l_r^k & \forall i, r, k & (20)\\
I_r^k = l_r^k - \sum_{j=1}^{n} m_{ir}(sur_j^k + pre_j^k + post_j^k) & r, k & (21)\\
l_r^k - d_r = O_r^k - g_r^k & \forall r, k & (22)
\end{array}$$

$$C_i^k \geq 0,\ I_{ij}^k \geq 0, I_r^k \geq 0, l_r^k \geq 0, O_r^k \geq 0, g_r^k \geq 0\ \forall(i,j,k) \in (J,J,K)$$

$$x_{ij},\ a_{ijr},\ y_{ir},\ q_{iu}^1,\ q_{ij},\ uq_{iju} \in \{0,1\}$$

Indices and Sets

n is the number of procedures to be scheduled

K is the number of scenarios

j, i, t and e index the set of procedures: from 1,...,n.

u indexes the position in the sequence of procedures: u=1,...,n.

r indexes is the set operating rooms: r=1,2 k indexes scenarios: k=1,....,K.

Parameters
c^w surgeon waiting time cost per unit time.

c^O overtime cost per unit time.

c^I idle time cost per unit time.

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.

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

Constraints

(2), (3), (4) define the possible sequences for the surgeon.

(6), (7), (8) define the sequence inside each OR.

(9), (10), (11), (12), (13), (14) compute the immediate predecessor of surgeries.

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

(16), (17) define the surgeon waiting time.

(18), (19) define the completion time for each surgery for a given assignment.

(20) calculates the completion time for every OR.

(21) calculates the OR idle time.

(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).

3.1. 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.

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 2^n 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 (y= 0,1,0,1,0,1....).

$$
\begin{aligned}
&\min \sum_{k=1}^{K} \frac{1}{K} \left(\sum_{i=1}^{n-1} c^w W_i^k + \sum_{i=1}^{n-2} c^s I_i^k + c^s fs^k + \sum_{r \in R} c_r^l O_r^k \right) && && (23)\\
&\text{s.t. :} C_1^k = \sum_{j \in J} (pre_j^k + post_j^k + sur_j^k) v_{1j} && k && (24)\\
&W_i^k = C_{i+1}^k - C_i^k + \sum_{j \in J} post_j^k x_{ij} - \sum_{j \in J} (sur_j^k + post_j^k) v_{i+1,j} && i < n, k && (25)\\
&I_i^k = C_{i+2}^k - C_i^k - \sum_{j \in J} (pre_j^k + sur_j^k + post_j^k) v_{i+2,j} && i < n-1, k && (26)\\
&fs^k = C_2^k - \sum_{j \in J} (pre_j^k + sur_j^k + post_j^k) v_{2j} && k && (27)\\
&C_n^k - d = O_1^k - g_1^k && k && (28)\\
&C_{n-1}^k - d = O_2^k - g_2^k && k && (29)\\
&\sum_{j \in J} v_{ij} = 1 && i && (30)\\
&\sum_{i \in I} v_{ij} = 1 && j && (31)
\end{aligned}
$$

Indices and Sets

n procedures to be scheduled indexed by j=1,...,n.

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

Variables

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

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

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

(25),(26),(27) calculation of surgeon waiting time, OR idle time for the second surgery and OR idle time between surgeries for every scenario.

(28),(29) calculation of overtime for each OR and every scenario.

(30),(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.

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.

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

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

$$I^k_{ip(j)j} = C^k_j - C^k_{ip(j)} + post^k_{ip(j)} - sur^k_j - post^k_j$$

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 ∞ .

0. Initialize S as the finite set of all *surgeon-OR sequences* (with cardinality 2^{n-1}).

   Initialize s_i as the zig-zag *surgeon-OR sequence* and remove from S

   Solve the corresponding SAA MIP and initialize the UB

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

2. 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 1.

3. Solve the *surgery sequencing sub-problem* SAA for s_i. If the solution is less than the current UB update it, otherwise go to 1.

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.

4.3 An Alternative Lower Bound

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 (0001101010111) 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_i^s 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 2a.(In L_i^I the function $ind_{>2}$ has this definition it takes 1 if i>2 otherwise 0)

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

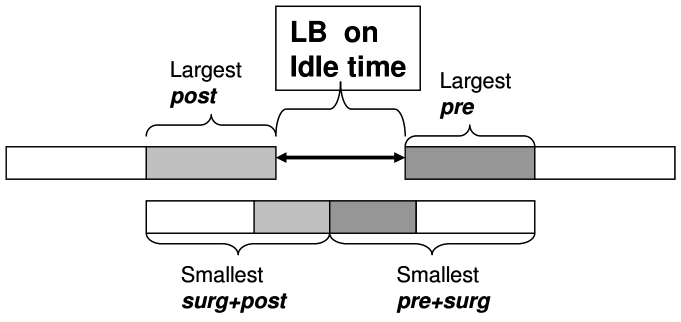

Figure 2a. Illustration of Lower Bound on idle time for P_2

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

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

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

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

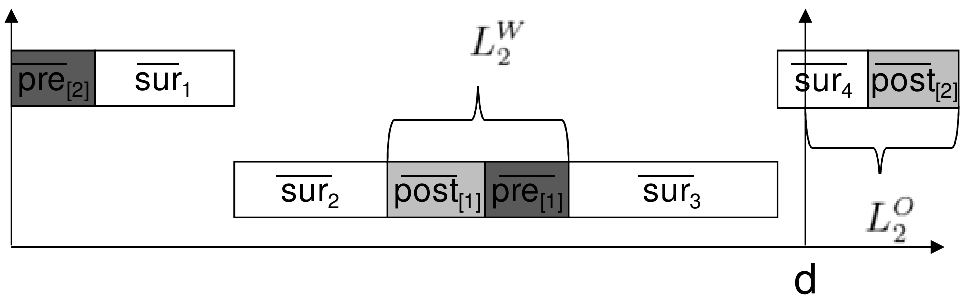

Figure 3b. Illustration of Lower Bound on wait and over time for P_2

Finally, we get our lower bound for partition i L_i.

$$L_i = c^I L_i^I + c^w L_i^W + c^o L_i^O$$

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.

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 zig-zag 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 O.R 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 follow:

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

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

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 5) we found that the gap between the optimal solution and the mean value solution (evaluated using the zig-zag OR-sequence) was 79% on average. The gap between the optimal solution and the heuristic solution was 44%. Thus the heuristic ultimately saved significant computation time.

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:

4.5.1 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.

4.5.2. Recycling Benders' Cuts

Since we have to solve many surgery sequencing sub-problems, and since they all have the same integer feasible region, we can store the integer feasible solutions from the first sub-problem and then recompute Benders' cuts in succeeding sub-problems. We hope to save computing time in the Benders' decomposition scheme although there is no guarantee that the cuts are not redundant in all the new sub-problems.

4.5.3. 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.

4.5.4. Cut strengthening

We observed significant problems with degeneracy in certain instances. We therefore used methodology from Magnanti and Wong (1981) 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_o) dominates (π^1, π_o^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.

4.5.5 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 (Wolsey, 1998). 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.

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 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.

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

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., (2010).

Factors	**Possible values**
Number of scenarios	10, 50, 100
B Parameter	high, low
Cost ratio	high, low

Table 1 Experimental Design

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, 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 bound. 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 disappointing results for the Benders’ algorithm appear to be caused by the fact that the technology matrix is not fixed (the coefficient of this matrix are scenarios for the different times modeled) and also that the right hand sides were usually zero. We observed that the Benders’ cuts were weak. Further we saw that the number of Benders’ cuts required to solve each subproblem varies with respect to the number of scenarios.

B Parameter , Cost ratio	Scenarios		
	10	50	100
B High , Low Cost ratio	607	1651	4979
B High , High Cost ratio	415	1375	3554
B Low, Low Cost ratio	677	1673	5841
B Low, High Cost ratio	368	1738	5144

Table 2: Computational Results (average processing time second).

We see in table 2 that experiments with low cost and high B parameter take more time to solve. Further, instances with high B always take more time than respective instances with low B.

6. Choosing the number of scenarios

In order to determine an appropriate number of scenarios, we applied the method proposed by (Linderoth, 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 4 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.

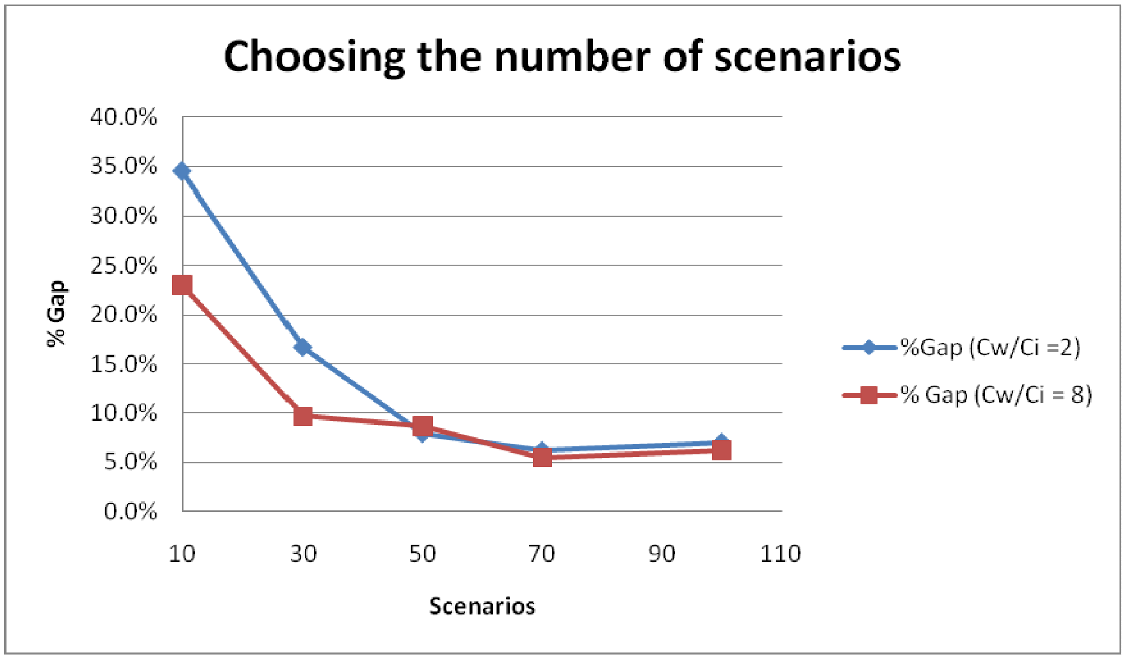

Figure 4: Tradeoff between number of scenarios and statistical gap.

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 industry partner the B factor typically ranges from 0.7 to 2 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=10 scenarios. The means and variances were generated as described in section 5. Let $pre_j^k, sur_j^k, and\ post_j^k$, 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:

$$\begin{aligned}
&pre_j^k(B) = (1+\phi)\,pre_j^k \\
&post_j^k(B) = (1+\phi)\,post_j^k \\
&sur_j^k(B) = sur_j^k - \phi\,pre_j^k - \phi\,post_j^k \\
&\text{where } \phi \text{ can be varied to generate desired B value}
\end{aligned}$$

Selecting the appropriate value of ϕ allows us to generate a problem with the same total work load for each procedure in each scenario, but with a given value of *B*. Next we solved the parallel OR scheduling problem for values of *B* from 0.5 to 2.0.

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 5A.

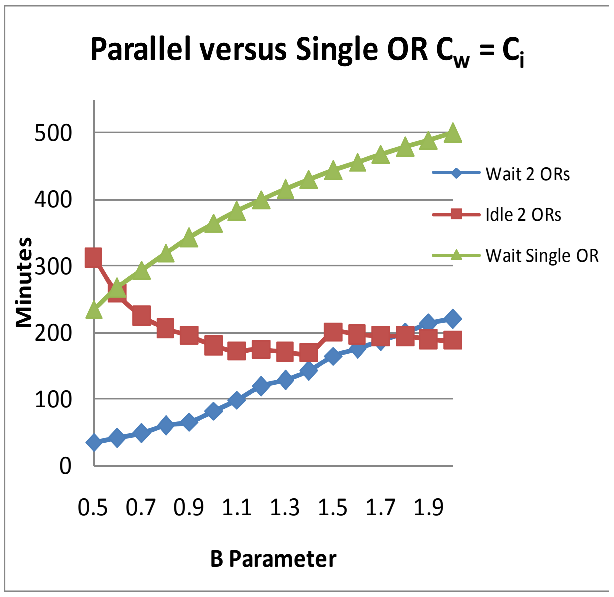

Figure 5 A :Comparison between single and parallel ORs, cost ratio = 1

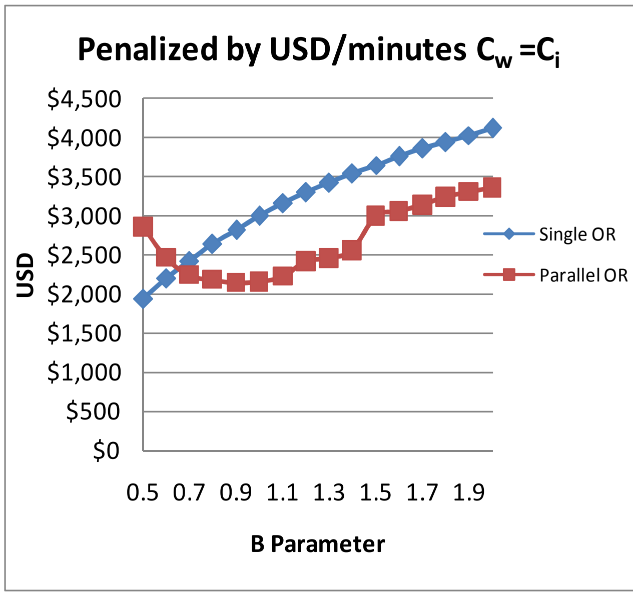

Figure 5 B :Cost curves when cost ratio = 1

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 6A.

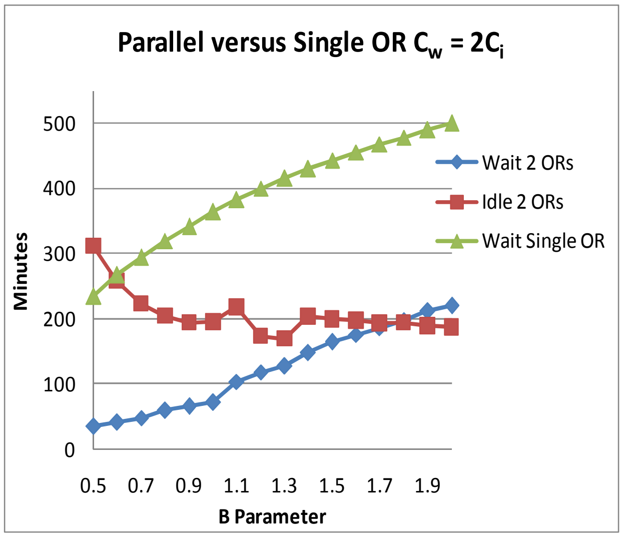

Figure 6: Comparison between single and parallel ORs, cost ratio = 2

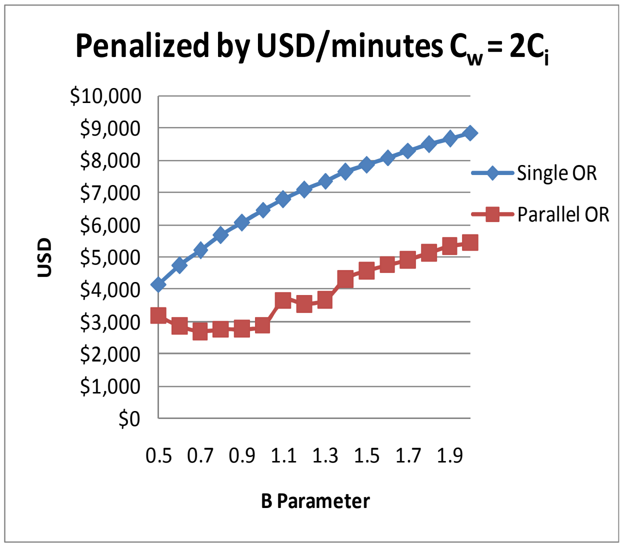

Figure 6 B :Cost curves when cost ratio = 2

In figure 5B we observe a cutoff value for the B parameter after which parallel ORs becomes cost effective (B≈0.7). This value changes with changing cost ratio. In figure 6B we can observe that parallel OR is always less costly.

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 selected 6 test problems each with 10 scenarios and 10 surgeries from among those 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 curves in Figure 6 below. By varying the cost ratio C_W/C_I and repeating, we generated one full curve in Figure 6. We then repeated the experiment for all 6 problem instances resulting in 6 total curves. 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.

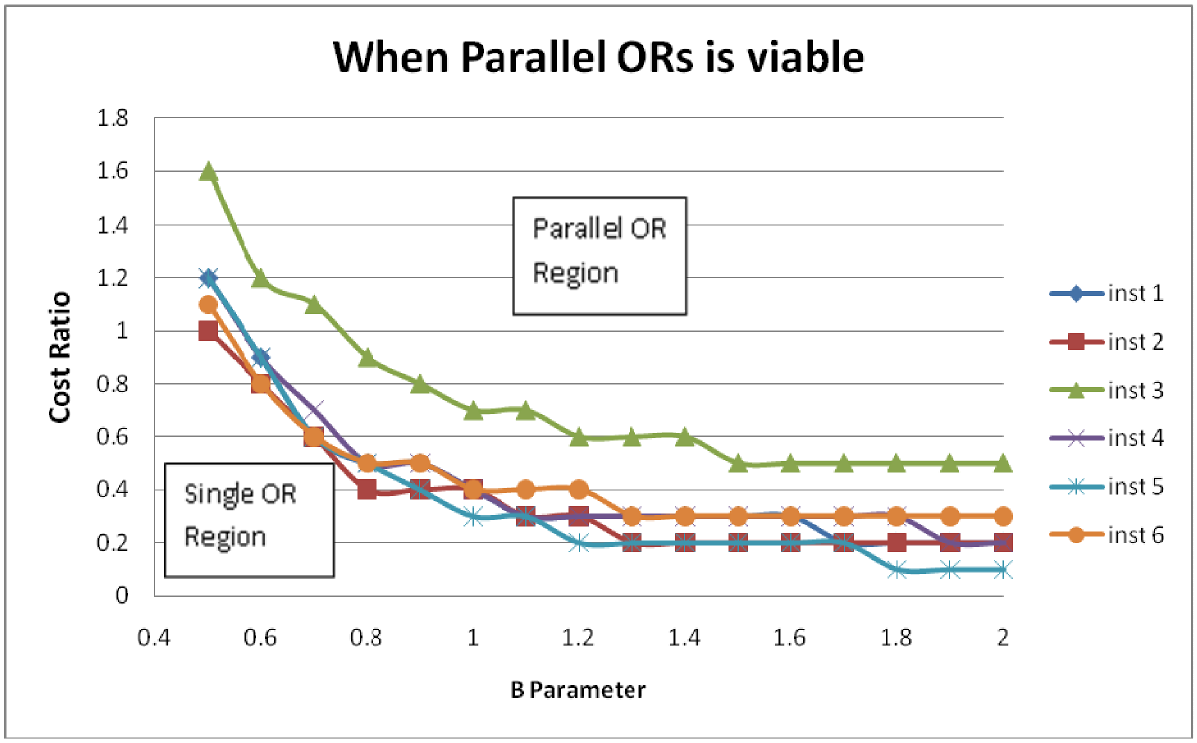

Figure 7: When to use parallel scheduling as a function of cost ratio and B parameter.

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. (2010) 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).

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

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

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

Given a single OR case, we compute s_2 and w_2 by assuming the same surgery sequence and the zig-zag OR-sequence. Given a parallel OR case, we compute $s_1=0$ and w_1 assuming that the surgeries with the biggest 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.

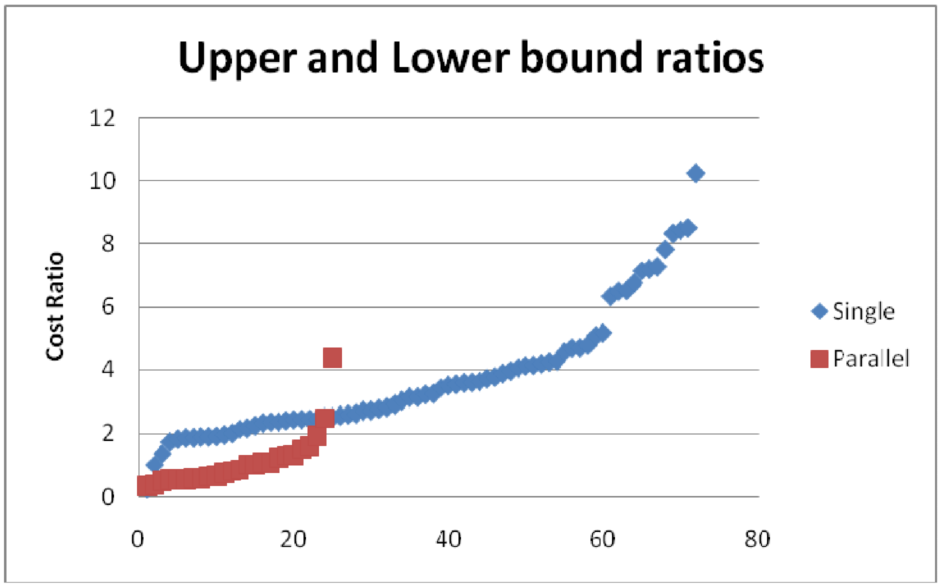

Figure 8: Upper and Lower bounds in the cost ratio based on historical decisions.

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.

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.

References

Abdekhodaee, A H, Andrew Wirth. 2002. Scheduling parallel machines with a single server some solvable cases and heuristics. Computers & Operations Research (29) 295 - 315.

Abdekhodaeea, A H, AndrewWirthb, Heng-Soon Gana. 2006. Scheduling two parallel machines with a single server: the general case. Computers & Operations Research (33), 994 - 1009.

Batun, Sakine, Brian T. Denton, Todd R. Huschka, Andrew J. Schaefer. 2010. The benefit of pooling operating rooms and parallel surgery processing under uncertainty. Informs Journal on Computing.

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

Glass, Celia A, Yakov M Shafransky, Vitaly A. Strusevich. 2000. Scheduling for parallel dedicated machines with a single server. Naval Research Logistics 47(1) 304 - 328.

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

Koulamas, C P. 1996. Scheduling two parallel semiautomatic machines to minimize machine interference. Computers & Operations Research 23 (10) 945 - 956.

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

Linderoth, J. T., A. Shapiro, and S. J. Wright, 2006 The empirical behavior of sampling methods for stochastic programming," Annals of Operations Research, vol. 142, pp. 219 - 245.

Magnanti, T L, R T Wong. 1981. Accelerating benders’ decomposition:algorithmic enhancement and model selection criteria. Operations Research (29) 464 - 484.

May JH, Strum DP, Vargas LG, 2000 Fitting the lognormal distribution to surgical procedure times. Decision Science; 31:129–48

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

Wolsey, L. (1998). Integer Programming. John Wiley & Sons.