Paper 154-2013

The Hospital Game: Optimizing Surgery Schedules to Save Resources, and to Save Lives

Andrew Pease and Aysegul Peker, SAS Institute, Inc.

ABSTRACT

Surgeons are required to perform vital operations on a daily basis, but often their planning is not optimized for the "downstream" care. This leads to under- or overutilization of postoperative nursing wards, which can either compromise a hospital's ability to provide the best possible care and ensure the best possible patient outcome, or result in precious hospital funding going toward underutilized resources.

This paper briefly reviews some of the academic research that is available for a data-driven, operations research approach to solving this challenge, which is dubbed the "Hospital Game" in some of this literature. The paper then proposes an optimization-based approach that uses the OPTMODEL procedure to derive the best surgery schedule for a major European hospital.

INTRODUCTION

Hospitals are confronted with growing patient populations, and they face growing demand for services. However, like any organization, hospitals also constantly grapple with budgetary restrictions and limited resources. The surgical ward is often considered the heart of the hospital, in terms of both patient care and revenue generation. Modern surgical schedule planning often takes into consideration only the surgeons' availability. However, when both lives and a hospital's financial well-being are at stake, it becomes essential for the hospital planning to optimize throughout its service chain toward the following key objectives: consistent use of resources, quality of care, and financial stability. This paper describes how to use the OPTMODEL procedure in SAS/OR® to formulate and resolve an optimized master surgical schedule (MSS). Optimization, only one important part of the entire solution (which involves data analysis, statistical analysis, optimization, and simulation) is the focus of the current paper.

THE MASTER SURGICAL SCHEDULE CHALLENGE

If you have never had to go to the hospital, that is wonderful news! If you have, your experience might have gone in one of two possible directions:

- Smoothly: The hospital operates like a swimming duck—calm on the surface, but paddling frantically under the surface to keep everything running relatively on schedule and to provide the best, most timely care to all patients.
- Chaotically: The hospital operates like a swimming puppy—madly thrashing in all directions because it clearly finds the situation overwhelming and unmanageable.

Either way, it is the nature of hospitals that operational planning is chaotic. It is impossible to know just when, and for which departments, surgeries will be required. Because surgeons are often central to a successful emergency intervention, surgical schedules are usually formulated to make most efficient use of scarce and expensive surgical resources. However, these schedules rarely take into consideration other influenced hospital departments such as surgical nursing wards.

Therefore, the number of patients who are admitted to and discharged from surgical nursing wards, along with the number of beds that are occupied in these wards, can vary considerably from day to day. These fluctuations greatly affect the nursing staff and their workload, resulting in inefficient use of resources. More beds and nursing staff are needed to cope with peaks in demand, whereas these same resources are underused in slow periods. These fluctuations are mainly due to the current way of planning surgeries, which disregards the effect of this planning on the surgical nursing wards where patients recover.

STATEMENT OF PROBLEM

Any successful application of operations research (OR), including the Hospital Game, involves three key steps:

1. You need to identify a real-world challenge as relevant to mathematical optimization formulation: Do you know enough about the real-life situation to allow for a "close-enough" mathematical formulation so that the resulting solution will be relevant to real-life applications?

2. You need to translate a "real-life problem formulation" into mathematical language, which consists of a preliminary mathematical statement that includes the control variables, the objective, and the constraints. For this step, you need a complete, in-depth understanding of all known influencing factors and outcome expectations. You often obtain this understanding through iterative interviews or workshops with the domain experts.

3. You develop and refine mathematics and algorithms:

    a. First, you model the structure of the mathematical problems that are outlined in step 2 as completely as possible.
    b. Then, you solve the problem by pushing it toward a desired optimum.

This paper largely concerns itself with the third step. It shows how to address the surgical schedule planning challenge while optimizing the downstream care, and it shows how PROC OPTMODEL provides the flexibility to do this. Although this paper focuses on step 3, the first two steps are also important. Step 2 takes an identified, real-world situation and structures it in a mathematically solvable format. This step is extremely critical to the resulting relevancy of any OR-derived solution. If a single constraint is missed, the solution will not yield a realistic implementation. If a single control variable is missed (or misinterpreted), crucial "pockets of potential" for improvements might be missed.

So before a single line of code is written, the OR analyst should carefully review the real-life situation with the domain experts. In the Hospital Game, this requires iterative dialogue with the hospital planners, both on the surgical side and on the nursing side. A key challenge that quickly emerges in such dialogues is that objectives often conflict in a multi-tiered service chain. For example, surgeries "produce" the patients, who then require post-operative care in the nursing wards. Surgeons want to ensure the best possible outcome for all patients for their individual operations. This means that surgeons need to be rested and ready, while at the same time the scheduling for the patient is equally crucial. Nurses, on the other hand, want a consistent number of patients in order to avoid stressful "peaks" (where nurses are taxed and patients might have to wait for crucial post-op care) and "valleys" (where nurses don't need to be quite as vigilant). In addition, the hospital has the financial objective of optimizing the use of all resources (for example, not having empty nursing ward beds) to enable spending for more patient-focused initiatives.

The overriding objectives are maximizing patient outcomes, optimizing personnel usage and satisfaction, and minimizing resource usage. The precise balance among these objectives can vary appreciably from one hospital or surgical ward to the next. Therefore, it is essential to have completed the first two steps (perhaps in iterations) before embarking on the implementation in SAS/OR. This paper should be viewed as a framework for the third step—using PROC OPTMODEL to develop a hospital-specific solution to the Hospital Game.

BRIEF REVIEW OF LITERATURE

This short section briefly discusses the key studies that were reviewed for the writing of this paper.

Research at Gelre Hospital in Apeldoorn, The Netherlands, showed that implementing an MSS can, for each specialty, steady the flow of patients who originate in the surgical department (Hans and Nieberg 2007). An MSS is a cyclical schedule in which slots of surgical time are reserved for a specific set of surgeries, enabling surgeons to decide which patients to treat in these slots. Limiting the choice of surgeries to be performed in a slot to surgeries that have similar medical and logistical properties enables the central planner to design a recurring sequence of slots that optimizes the efficiency of the operating room and the nursing wards. Vollebregt's excellent initial problem formulation is academic and largely exploratory in nature. The current paper offers some insight into how hospitals can take the next step toward actually implementing such an MSS.

Research at the Catholic University Hospital of Leuven, Belgium, presents a broad overview of various heuristic methodologies for scheduling in hospitals (Belien 2005). This thorough summary of problem, formulation, and algorithmic approaches provides the reader with a solid theoretical background for the challenges that the current paper addresses. Further research at Gelre Hospital Apeldoorn includes an overview of a pilot application that was constructed for the Hospital Game (Schol 2012).

METHODOLOGY

This paper shows how you can use the OPTMODEL procedure to do the following:

- read the data from SAS data sets
- formulate the problem by defining decision variables, objectives, and constraints
- pass the decision variables, objectives, and constraints in a call to the mixed integer linear programming (MILP) solver
- output the solution to SAS data sets.

The paper describes each of these steps, and it provides source code examples of the statements that implement these steps in the OPTMODEL procedure.

Without optimization, surgeries might sometimes not be scheduled because resources are not available. To ensure that the maximum number of surgeries is scheduled, use the following two-step optimization (each of these steps is called a "solve"):

1. Solve for the maximum number of surgeries that can be accommodated with the available resources.

2. Impose a constraint to minimize the deviation of the recovery ward occupancy while ensuring that the maximum number of surgeries are actually scheduled. This step attempts to minimize the variation of time spent by each patient in the recovery room. The scope of this paper is limited to the L-infinity norm, which is an approximation for the true variation measure, standard deviation.

Figure 1 uses the resource optimization model to depict this high-level formulation of the optimization challenges.

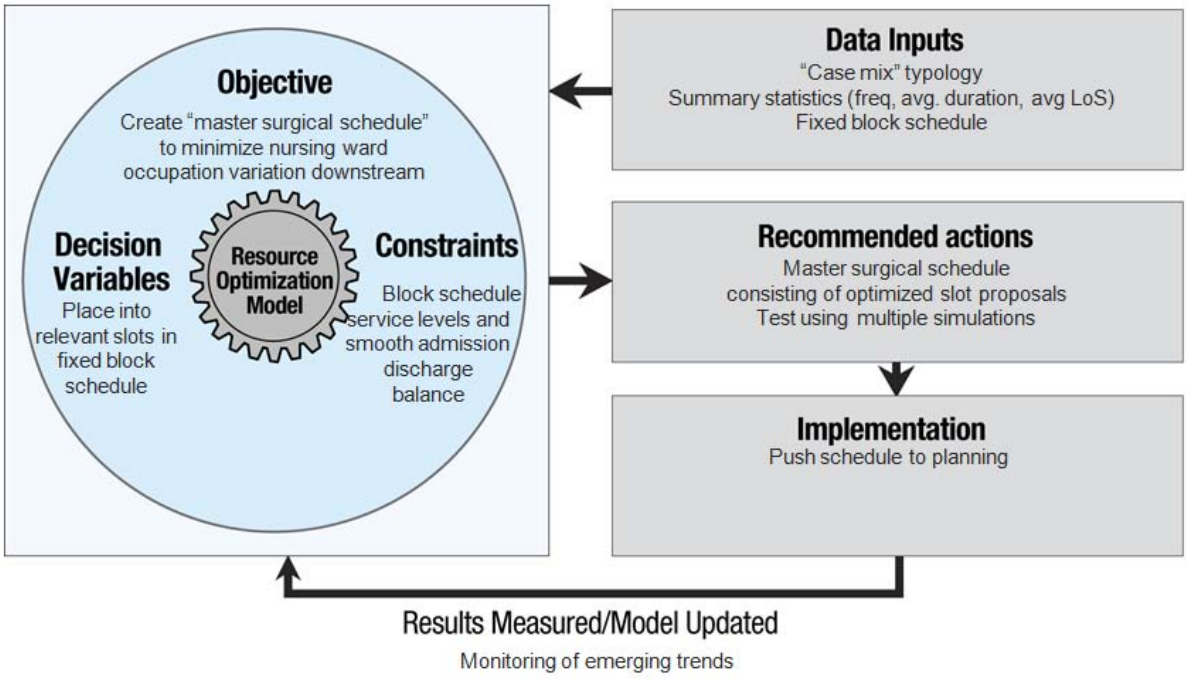


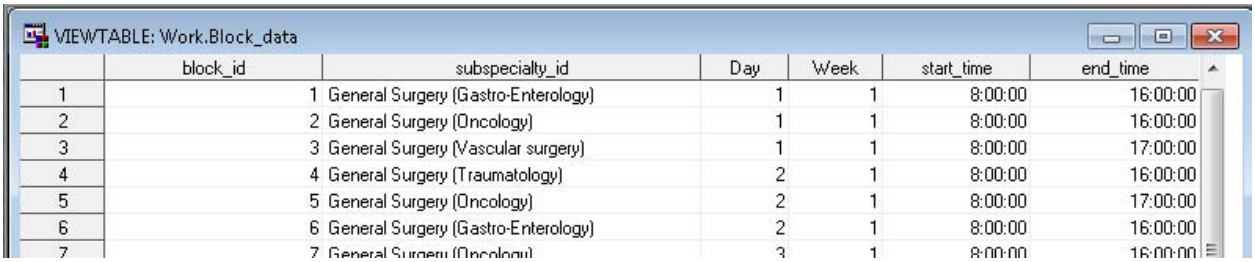
Figure 1. Optimization Model

The following sections describe the data and the preliminary formulation of three components (decision variables, objectives, and constraints) of the mathematical model and spell out additional concepts in more detail.

INPUT DATA

Two input data sets are required for the model: block data and case data. A block is defined as a time slot when an operating room is available (or not) for a particular surgical subspecialty. Block data contain the start and end hours for each block in addition to the day of the week, the week of the planning time horizon, and the subspecialty of the cases that can be assigned to the corresponding block.

Display 1 shows sample block data.



VIEWTABLE: Work.Block_data

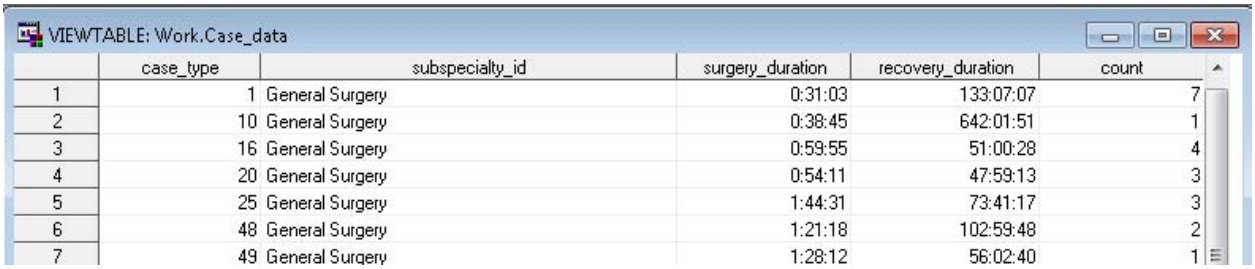
	block_id	subspecialty_id	Day	Week	start_time	end_time
1	1	General Surgery (Gastro-Enterology)	1	1	8:00:00	16:00:00
2	2	General Surgery (Oncology)	1	1	8:00:00	16:00:00
3	3	General Surgery (Vascular surgery)	1	1	8:00:00	17:00:00
4	4	General Surgery (Traumatology)	2	1	8:00:00	16:00:00
5	5	General Surgery (Oncology)	2	1	8:00:00	17:00:00
6	6	General Surgery (Gastro-Enterology)	2	1	8:00:00	16:00:00
7	7	General Surgery (Oncology)	3	1	8:00:00	16:00:00

Display 1. Sample Block Data

The case input data include the attributes of each case.

Display 2 shows a sample of case data.

`



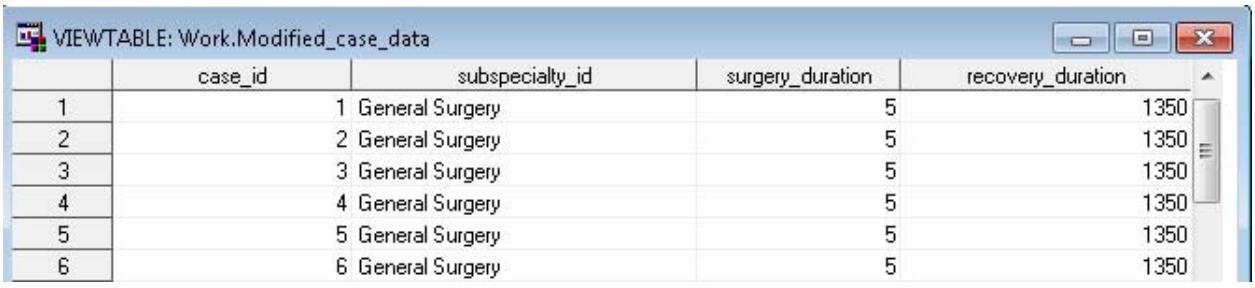
VIEWTABLE: Work.Case_data

	case_type	subspecialty_id	surgery_duration	recovery_duration	count
1	1	General Surgery	0:31:03	133:07:07	7
2	10	General Surgery	0:38:45	642:01:51	1
3	16	General Surgery	0:59:55	51:00:28	4
4	20	General Surgery	0:54:11	47:59:13	3
5	25	General Surgery	1:44:31	73:41:17	3
6	48	General Surgery	1:21:18	102:59:48	2
7	49	General Surgery	1:28:12	56:02:40	1

Display 2. Sample Case Data

Case data include the case type, subspecialty ID, duration estimate, and recovery time estimate in addition to the number of cases that need to be scheduled. For example, Display 2 shows that case type 1, with the subspecialty "General Surgery," has an estimated surgery duration of 31 minutes and 3 seconds and an estimated recovery time of 133 hours, 7 minutes, and 7 seconds. These data are derived from statistical methods that use the historical data, and these estimates are the mean values. Although this paper proposes a deterministic approach, distribution of these estimates in a real-world implementation should be considered in order to derive a more robust and relevant schedule.

To derive a robust schedule for case data, you convert the surgery duration and recovery duration to multiples of one-tenth of an hour and create single cases for each case type. The number of cases for each case type is determined by the count data. Display 3 shows the final input data.



VIEWTABLE: Work.Modified_case_data

	case_id	subspecialty_id	surgery_duration	recovery_duration
1	1	General Surgery	5	1350
2	2	General Surgery	5	1350
3	3	General Surgery	5	1350
4	4	General Surgery	5	1350
5	5	General Surgery	5	1350
6	6	General Surgery	5	1350

Display 3. Final Input Case Data

The block data are modified to match the same time unit. Start and end times for two blocks of different days might be the same. These times are converted to unique numbers based on the day of the week and the week of the planning horizon. The following DATA step makes this conversion for the block data:

```
data input.Modified_Block_Data;
set input.Block_Data;
start_time = start_time + 24 * ((week - 1) * 7 + day - 1);
end_time = end_time + 24 * ((week - 1) * 7 + day - 1);
run;
```

Display 4 shows a sample from the modified block data, which will become an input file to optimization in the next section.

VIEWTABLE: Work.Modified_block_data

	block_id	subspecialty_id	Day	Week	start_time	end_time
1	1	General Surgery (Gastro-Enterology)	1	1	80	160
2	2	General Surgery (Oncology)	1	1	80	160
3	3	General Surgery (Vascular surgery)	1	1	80	170
4	4	General Surgery (Traumatology)	2	1	320	400
5	5	General Surgery (Oncology)	2	1	320	410
6	6	General Surgery (Gastro-Enterology)	2	1	320	400

Display 4. Modified Block Data

Next, you map the time units in the modified data to the actual time to create the operational output from the optimization output. Display 5 shows a sample from this mapping.

VIEWTABLE: Input.Mapping

	time_index	time
1	1	80
2	2	81
3	3	82
4	4	83
5	5	84
6	6	85
7	7	86

Display 5. Input Mapping

PROBLEM FORMULATION

To formulate the problem as a MILP problem, you need to define the decision variables, define the constraints, and define the objectives. These steps are described in the following subsections.

DEFINE THE DECISION VARIABLES

The primary decision is to determine which predetermined case type should be assigned to which available block. In the mathematical model, the decision variable is represented by a binary variable that indicates whether each case of a subspecialty starts at a certain time of a specific block. The remaining variables, except the measure for deviation among time intervals in the recovery room, are implied variables. This requires the development of five families of decision variables:

- case-to-block assignment
- binary assignment
- check-outs
- check-ins
- variation

The following sections outline these families of decision variables in more detail.

Case-to-Block Assignment Variables

The case-to-block assignment decision variables indicate whether a case of a given type is assigned to a block to start at time *t*. This family of decision variables is defined over all eligible triplets of case, block, and time.

The following OPTMODEL statements define the Assign decision variable over the eligible case, block, and time triplets:

```
set CASE_BLOCK{c in CASE} = {b in BLOCK: bl_subspecialty[b]=cs_subspecialty[c]};
set START_TIMES{b in BLOCK} = {t in bl_start_ind[b]..bl_end_ind[b] by &interval};
set CASE2BLOCK = {c in CASE, b in CASE_BLOCK[c], t in START_TIMES[b]:
t<bl_end_ind[b]-cs_duration[c]+2*&interval};
var Assign{CASE2BLOCK} binary;
```

The second SET statement creates the CASE2BLOCK set in three steps:

1. A set array over the CASE set is defined. Each element of this set gives the possible blocks for the corresponding case.
2. A set array is defined to keep the start times of the corresponding blocks.
3. If a case is assigned to a certain block, eligible start times are derived using the condition

   ```
   t<bl_end_ind[b]-cs_duration[c]+2*&interval
   ```

   where bl_end_ind [b] is the end time interval index for Block b, cs_duration[c] is the case duration for Case c, and interval is the macro variable that includes the time increments in the planning. For the conditions in set definitions, using equalities might not produce the expected output because of precision problems, so a less-than operation is used instead.

Binary Assignment Variables

The second family of decision variable defines whether or not a case is assigned. The following statement defines this family, which is named Unassign, over the CASE set:

```
var Unassign{CASE} binary;
```

The Assign variable implies the Unassign variable. The Assign_def constraint imposes the relationship between Assign and Unassign. The Assign_def constraint also requires that a case be assigned to no more than one block. The following statement defines the expression for the Assign_def:

```
con Assign_def{c in CASE}: sum{<(c),b,t> in CASE2BLOCK} Assign[c,b,t] + Unassign[c] = 1;
```

Check-Out and Check-In Variables

The third family of decision variables indicates the number of check-outs from the recovery ward, and the fourth family of decision variables indicates the number of check-ins. These two families impose the business restrictions on the number of check-outs, the number of check-ins, and difference between the number of check-outs and the number of check-ins. These two families are implied by the Assign variable and are defined over all possible time units, which form a much larger set than the set that is formed by all possible time units of the blocks. The following statements define this set, which is named ALLPERIODS_RECOVER:

```
num max_block_end = max{b in BLOCK} bl_end_ind[b];
num max_recover_end = max{b in BLOCK} bl_end_ind[b] + max{c in CASE} cs_recovery[c];
set ALLPERIODS_RECOVER = ALLPERIODS union BLOCKEND union max_block_end..max_recover_end;
```

The following statements declare the variables and impose the constraint of an implied relationship between the Assign variable and these two families of decision variables:

```
var Numcheckout{tr in ALLPERIODS_RECOVER} <= min(&numcheckout_UB,recovery_capacity) >= 0 integer;
var Numcheckin{tr in ALLPERIODS_RECOVER} <= min(&numcheckin_UB,recovery_capacity) >= 0 integer;
con Numcheckout_def{tr in ALLPERIODS_RECOVER}: Numcheckout[tr] =
sum{<c,b,ta> in CASE2BLOCK: tr>ta+cs_duration[c]+cs_recovery[c]-2*&interval
and tr<ta+cs_duration[c]+cs_recovery[c]} Assign[c,b,ta];
con Numcheckin_def{tr in ALLPERIODS_RECOVER}: Numcheckin[tr] =
sum{<c,b,ta> in CASE2BLOCK: tr>ta+cs_duration[c]-&interval
and tr<ta+cs_duration[c]+&interval} Assign[c,b,ta];
```

Both the third and fourth families have upper bounds that result from business restrictions. However, you can tighten the problem formulation by taking the minimum of the upper bound for each of these variables and the recovery room capacity, because the number of check-outs can never exceed the recovery room capacity.

In the Numcheckout_def constraint, the condition in the sum includes all possible assignments that can result in a check-out at the corresponding time, tr. The same logic is applied to the Numcheckin_def constraint. This constraint includes all possible assignments that can check in at the corresponding time, tr. Strict inequalities are used to formulate the conditions in the summations.

The decision variables to be explained after this point are needed only for the second solve. The number of cases that use the recovery room at each time unit is stored in the Occupancy variable. Business restrictions impose an upper bound, which is the capacity of the recovery room. The following statements define this family of decision variables over the ALLPERIODS_RECOVER set and impose constraints that show the implication between the Assign and Occupancy variables:

```
var Occupancy{ALLPERIODS_RECOVER} integer <= recovery_capacity >=0;
con Occupancy_def{tr in ALLPERIODS_RECOVER}:
Occupancy[tr] = sum{<c,b,ts> in CASE2BLOCK:
tr in ts+cs_duration[c]..ts+cs_duration[c]+cs_recovery[c]-&interval INTER ALLPERIODS_RECOVER}
Assign[c,b,ts];
```

Measure-of-Variation Variables

The fifth set of variables defines the measure of variation of times in the recovery room. The L-infinity norm is used to measure the variation. The following statement declares the decision variable:

```
var Maxoccupancy integer <= recovery_capacity >=0;
```

The constraints that imply the relation between the Occupancy and Maxoccupancy variables are explained in the section DEFINE THE OPERATIONAL CONSTRAINTS.

Decision Variable Restrictions

When the decision variables are built, the following restrictions for using sets or bounds apply:

- A case cannot be assigned to a block if it finishes after the end time of the block.
- The start and end times of a block conform to the data that are given.
- A case can be assigned to a block only if the subspecialties match.
- A case cannot be assigned to more than one block.
- The number of check-outs from the recovery ward has an upper bound of &numcheckout_UB.
- The number of check-ins from the recovery ward has an upper bound of &numcheckin_UB.
- The number of patients recovering from surgeries has an upper bound of recovery_capacity.

DEFINE THE OBJECTIVES

This section shows the PROC OPTMODEL statements that define the objectives in the mathematical model.

- The following statement specifies the first objective function by constructing the constraint that imposes the maximum number of assignments:

  ```
  min Numunassign = sum{c in CASE} Unassign[c];
  ```

- The following statements specify the optimal value of the first objective function by forming the constraint that ensures that the solver assigns the maximum number of cases:

  ```
  num max_unassign init Numunassign.sol;
  con Numunassign_max: Numunassign <= max_unassign;
  ```

- The primary objective function is used to minimize the variation of occupancy of the recovery ward. The ideal way to measure the variation is standard deviation that is derived from historical cases. Although there is no efficient generic or universal methodology in literature to solve mixed integer nonlinear problems, you can use linear functions to approximate this standard deviation. This paper uses the L-infinity norm to even out the workload in the recovery wards, given the following constraint: Minimize the maximum recovery ward occupancy at the beginning of each time interval of all days.

The following PROC OPTMODEL statements formulate the objective function and constraints:

```
con Linfcon{t in ALLPERIODS_RECOVER}: Occupancy[t] <= Maxoccupancy;
min Linf = Maxoccupancy;
```

You can use measures other than the L-infinity norm, such as the L1 norm or linearized L2 norm. Each of these measures has advantages and disadvantages that affect performance, depending on the data. For the scope of this paper, the formulation is limited to the L-infinity norm.

DEFINE THE OPERATIONAL CONSTRAINTS

This section describes an initial set of constraints for MSS optimization. This set can vary based on the specific needs of an individual hospital.

The following CON statement imposes the first constraint—that no more than one surgery can consume the operating room of a block:

```
con Conflict_procedure{b in BLOCK, t in START_TIMES[b]}:
  sum {<c,(b),tin> in CASE2BLOCK: tin<t+&interval and tin>t-cs_duration[c]} Assign[c,b,tin] <= 1;
```

The following range constraint imposes the second set of preferential constraints—to avoid too many check-outs with too few check-ins or too many check-ins with too few check-outs (that is, operational restrictions limit the absolute value of the difference between the number of check-outs and the number of check-ins):

```
con Absdiff_range{tr in ALLPERIODS_RECOVER: tr>minstart}:
-3 <= Numcheckin[tr] - Numcheckout[tr] <= 3;
```

The condition in the constraint definition eliminates the indices for the time units during which there are no check-outs or check-ins. In other words, this condition improves the performance by removing the redundant constraints.

As a side note, the presolver can remove this type of constraints without your having to remove them from the constraint definition.

MIXED INTEGER PROGRAMMING (MILP) SOLVER

PROC OPTMODEL offers the following types of solvers for different types of mathematical programming problems:

- linear programming (LP) solver
- mixed integer linear programming (MILP) solver
- nonlinear programming solver
- quadratic solver
- integer programming solver

The MSS problem falls into the mixed integer programming problem category. The MILP solver implements a LP-based “branch-and-bound” algorithm. Branch-and-bound strategy divides the feasible region in to disjoint subproblems. The MILP solver then incorporates presolving, cutting planes, and additional heuristics to increase the efficiency in finding an optimum.

To solve the MSS challenge, the following SOLVE statement uses the MILP solver:

```
solve with milp/primalin;
```

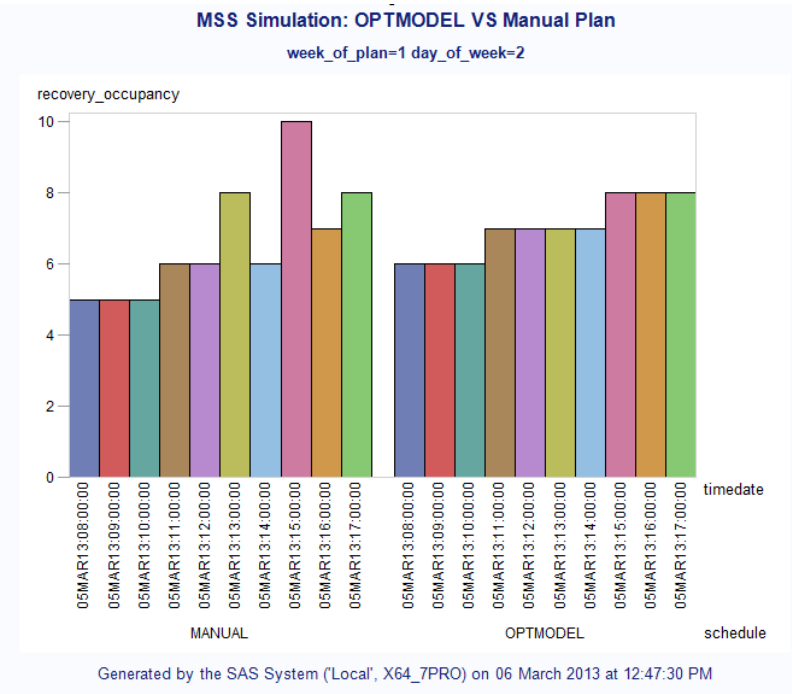
The PRIMALIN option enables you to input an integer feasible solution in PROC OPTMODEL before invoking the MILP solver. Adding the PRIMALIN option to the SOLVE statement requests that the MILP solver use the current variable values as a starting integer feasible solution (warm start). If the MILP solver finds that the input solution is valid, then the input solution provides an incumbent solution and a bound for the branch-and-bound algorithm. If the solution is not valid, then the PRIMALIN data are ignored. When it is difficult to find a good integer feasible solution for a problem, this warm start can reduce solution time significantly.

The real time for solving this instance of the MSS problem is about 30 minutes.

RESULTS

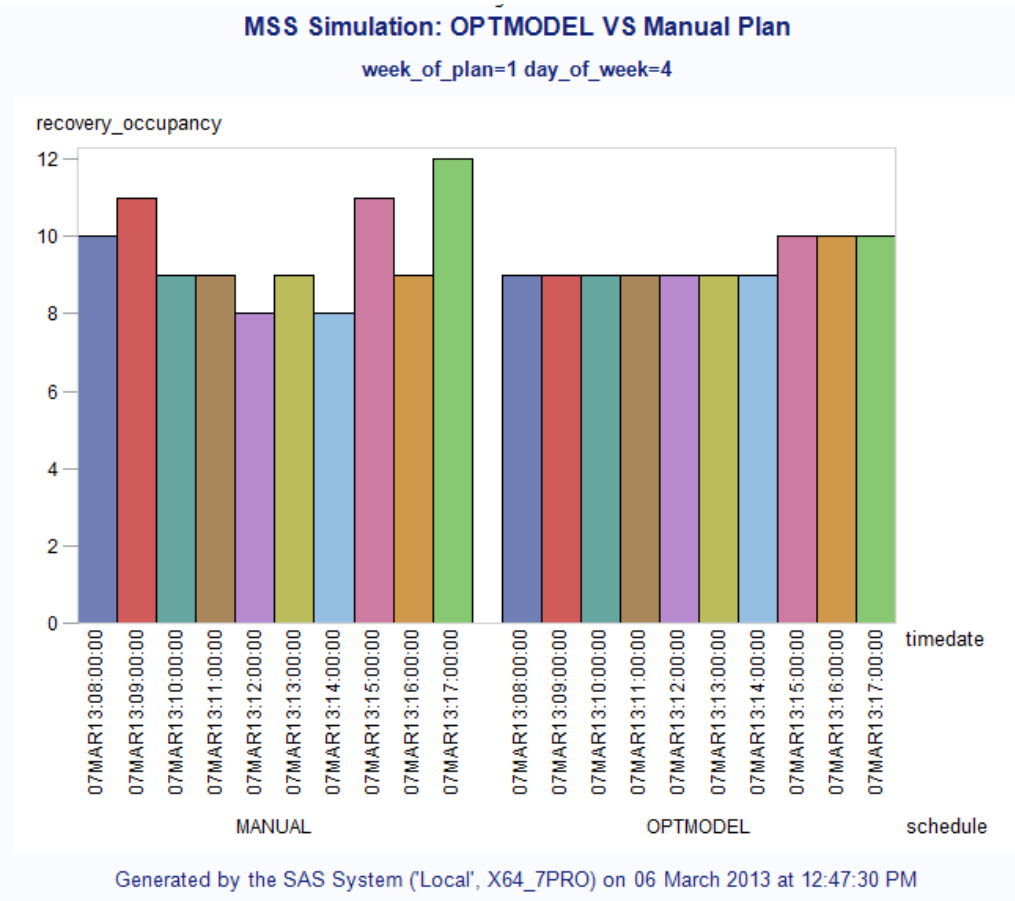
This section looks at the resulting length of stay in the nursing wards for a schedule that was generated for a single operating room in a two-week period in March 2013. This simulation assumes that the nursing wards were empty on the first day and gradually fill up.

Display 6 demonstrates that the OPTMODEL schedule creates a much more steady buildup in occupancy, already demonstrated on day 2.



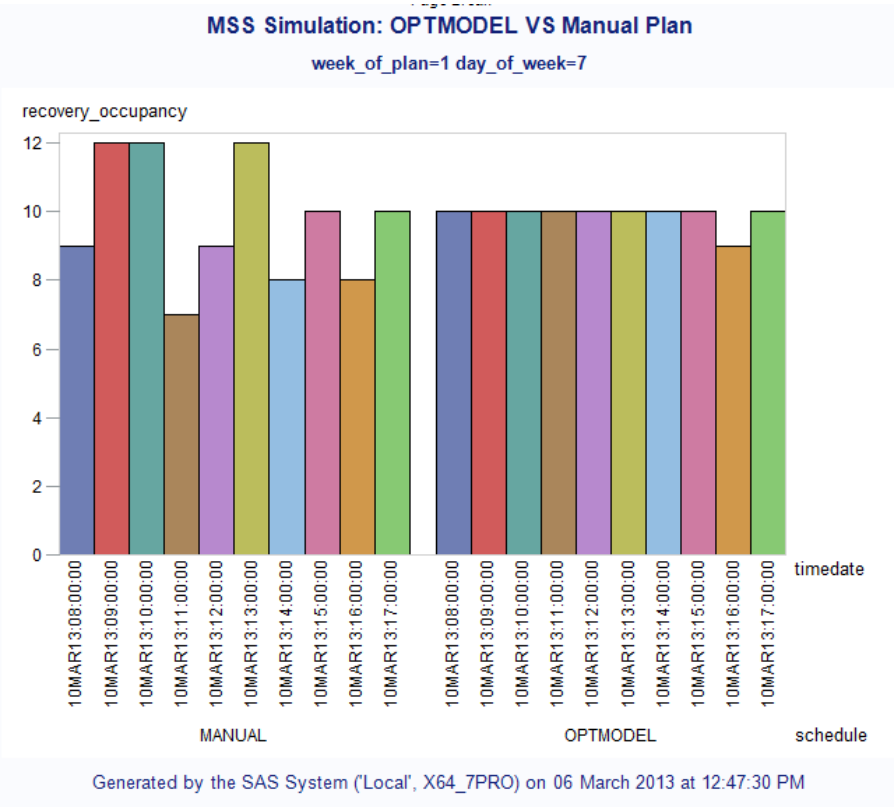
Display 6. Day 2 of the Schedule Simulation

The advantages of the optimized schedule become even more apparent on day 4, as shown in Display 7. The manual schedule requires a bed to be ready for up to 12 patients; in the optimized schedule, the occupancy maximizes at 10 beds.



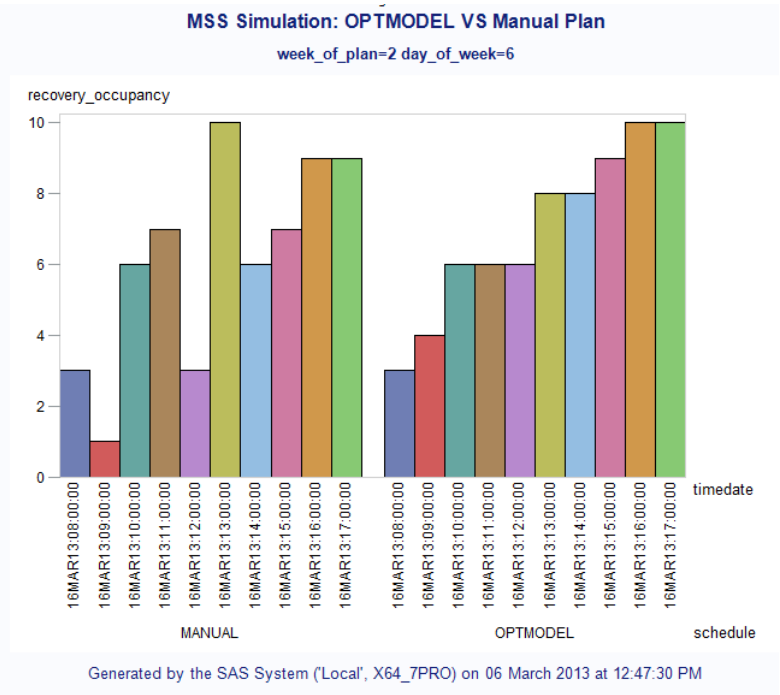
Display 7. Day 4 of the Schedule Simulation

This trend continues on through day 7, as shown in Display 8.



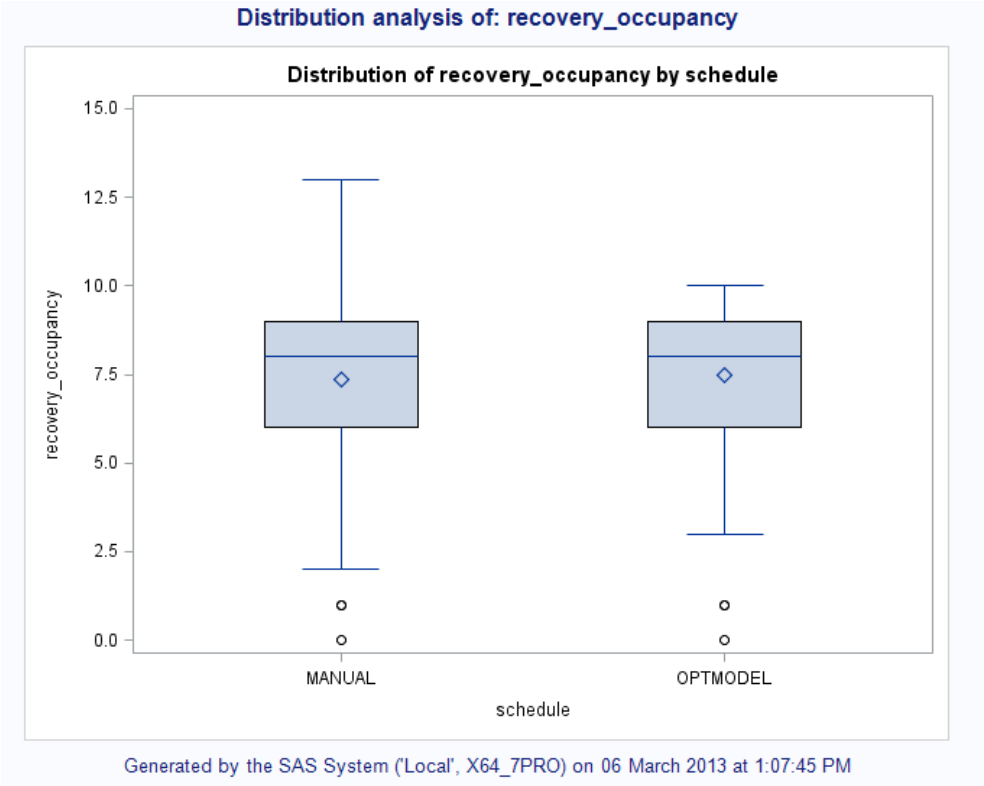
Display 8. Day 7 of the Schedule Simulation

Finally, at the end of the second week, there are numerous discharges in both plans. However, in the optimized plan, the ”shock” is less and the buildup is more stable, as shown in Display 9.



Display 9. Day 13 of the Schedule Simulation

Not only does the optimization manage to keep the number of required beds down, but the actual average occupancy is slightly higher even for this very small sample, as the box plot in Display 10 demonstrates.



Display 10. Box Plot Comparing Distributions of Both Plans

IMPLEMENTATION NOTES

The optimization that is described so far serves as the theoretical basis for implementation. However, for actual implementation, there are a few additional considerations.

For the actual operations, this optimization runs in a rolling horizon fashion, which means that the initial states of the surgery and recovery rooms are not empty. However, the initial states of the recovery and surgery room are assumed to be empty for the current formulation. There are several ways to integrate the initial states of the surgery blocks and recovery rooms. One way is to define dummy cases. Two types of dummy cases are required: one for the cases that are assigned for surgery blocks, and the other for the cases that are in the recovery rooms. For the first type of dummy case, assignment is already done and fixed. Before you call the solver, use the following statements to fix these cases to 0 for all the blocks and time slots to which they are not assigned, and to fix these cases to 1 for the time slot and block to which each of them is assigned:

```
for {c in DUMMY_BLOCK_CASES, <(c),b,t> in CASE2BLOCK} fix Assign[c,b,t] = 0;
for {c in DUMMY_BLOCK_CASES} fix Assign[c,assigned_block[c],assigned_slot[c]] = 1;
```

In the preceding statements, the block and time slot to which a case is already assigned are named `assigned_block[c]` and `assigned_slot[c]`, respectively.

To accommodate the patients who are already in the recovery room, create dummy cases with zero surgery duration and assign the remaining recovery time as the recovery duration. To impose the start of the recovery for the remaining times, fix the assignment variable for these cases to 1 for an eligible block. Because the surgery duration for these cases is 0, the block resource is not used and recovery starts immediately.

You also need to create a robust schedule that takes into consideration the stochastic nature of the process. The current optimization model considers only the situation for which both the cases of each subspecialty and the duration times are deterministic. For the nondeterministic duration times, a two-step approach is suggested:

1. To the duration times, add buffer times that are based on the standard deviation of the surgery duration, as follows:

```
for {c in CASE} case_duration[c] = case_duration[c] + &factor * case_std_dev[c];
```

   Specifying an inflated large factor can cause underutilized blocks and recovery rooms, whereas specifying a small factor can result in a non-robust, irrelevant solution. Use domain expertise and experimentation to best determine this factor.

2. Handle the second source of stochasticity, which are the emergency cases. Based on the historical occurrence of the emergency cases of each subspecialty, create dummy cases and input them to the optimization. Because emergency cases need to be scheduled within a short time window, these cases have a time-dependent weight factor for the occupancy. For the time intervals within the acceptable time window, the weight is 1; after the time window, these weights are assigned high values. Assignment before the time window starts is fixed to 0 to prevent any assignment before the emergency case occurs. Time window information is read from the data that are generated from the historical distribution of emergency cases. The following statements define the additional parameters that are required to incorporate emergency cases into the stochasticity:

```
num weight{CASE2BLOCK} init 1;
num timewindowLB{EMERGENCYCASES}; /* read form data */
num timewindowUB{EMERGENCYCASES}; /* read form data */
num largenumber init card(CASE);
```

   Use a large number to discourage any assignment after the time window of the emergency cases. The maximum possible occupancy of recovery room is the cardinality of the CASE set. The following statements prevent the assignment of the emergency cases outside their time window by assigning the weight of the occupancy to be the cardinality of the CASE set:

```
for {c in EMERGENCYCASES, <(c),b,t> in CASE2BLOCK: t<timewindowLB[c]} fix Assign[c,b,t] =
0;
for {c in EMERGENCYCASES, <(c),b,t> in CASE2BLOCK:
t>timewindowUB[c]+cs_duration[c]+cs_recovery[c]-&interval} weight[c,t] = largenumber;

con Occupancy_def{tr in ALLPERIODS_RECOVER}:
Occupancy[tr] = sum{<c,b,ts> in CASE2BLOCK:
tr in ts+cs_duration[c]..ts+cs_duration[c]+cs_recovery[c]-&interval INTER
ALLPERIODS_RECOVER} weight[c,tr] * Assign[c,b,ts];
```

   With these requirements, optimization might not assign any emergency cases, because the time window restriction for emergency cases is more restrictive than it is for regular cases. The following constraint imposes a service-level requirement for the emergency cases by subspecialty:

```
set EMERGENCY_SUBSPECIALTY{s in SUBSPECIALTY}
= {c in EMERGENCYCASES, <(c),b,t> in CASE2BLOCK: case_subspecialty[c] = s};
con Emergency_service_level{s in SUBSPECIALTY}:
sum{<c,b,t> in EMERGENCY_SUBSPECIALTY[s]} Assign[c,b,t]
>= sla[s] * card(EMERGENCY_SUBSPECIALTY[s]);
```

The service levels for the emergency cases of each subspecialty are user input.

When you have finished optimizing the MSS, you should validate its robustness. Although validation details are beyond the scope of this paper, in general, you can measure performance and robustness of the schedules by using a simulation model that you run multiple times. The simulation model can incorporate the stochastic duration time and stochastic occurrence of the emergency cases. You can measure the following performance characteristics for each simulation run:

- block utilization per subspecialty
- service level per subspecialty

CONCLUSION

The operations and their subsequent scheduling at one hospital can be very different from those at another hospital. The successful implementation of an optimization is closely related to how well the business requirements are formulated. The implementation needs to strike a balance between approximate and exact implementations. In other words, incorporating too many business restrictions can result in an unstable solution, whereas incorporating too few can result in an irrelevant solution that does not work.

This paper demonstrates that PROC OPTMODEL is flexible enough to incorporate differing business requirements. Any analyst who wants to develop a schedule optimizer for a master surgical schedule can follow the steps in this paper. The benefits of an optimized master surgical schedule are two-fold:

- You can limit the maximum number of resources that are required downstream for post-operative nursing care.
- You can achieve a more steady utilization of those available resources.

It is up to the operations research expert to determine the scope of the solution and to implement it successfully.

The authors hope that this document will result in more hospitals optimizing their surgical schedules. This type of optimization will benefit not only the financial health of those hospitals, but also the hospital staff, surgeons, and patients.

REFERENCES

Beliën, J., Demeulemeester, E., and Cardoen, B. (2008). “A Decision Support System for Cyclic Master Surgery Scheduling with Multiple Objectives.” *Journal of Scheduling* 12:147–161.

Hans, E. and Nieberg, T. (2007). “Operating Room Manager Game.” *INFORMS* 8:25–36. Available at http://ite.pubs.informs.org/Vo8No1/.

Schol, G. (2012). “Designing a Master Surgical Schedule for Gelre Apeldoorn.” Master’s thesis, University of Twente, Netherlands.

ACKNOWLEDGMENTS

The authors are grateful to Rob Pratt and Anne Baxter of the SAS Advanced Analytics Division for their valuable assistance in the preparation of the paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the authors at:

Andrew Pease
SAS Institute Inc.
Email: Andrew.Pease@sas.com

Aysegul Peker
SAS Institute Inc.
Email: Aysegul.Peker@sas.com