Paper WUSS 122-2018

# Optimizing Exam Schedules at Oklahoma State University

Thu Nguyen, Oklahoma State University

## ABSTRACT

Every semester, the Registrar Office of Oklahoma State University schedules hundreds of exams into rooms according to the Colleges' preference and room availability. In addition, these schedules need to incorporate several other requirements, such as schedule priority, proximity, and times to avoid conflicts with other exams. Manual scheduling is time consuming and will not be feasible as the number of requirements increase in the future. Nonetheless, the automated scheduling process must be flexible enough so that non-programmer users can conveniently alter the constraints and input parameters as requirements change from semester to semester. Moreover, the results need to be in a user-friendly format so that the Registrar Office staff can determine whether further adjustments are needed. This paper will demonstrate a program that extracts information entered by users on a spreadsheet to create dynamic SAS® scripts using macro programs. The program incorporates an optimization model built using SAS/OR® PROC OPTMODEL. The optimized schedules are then visualized using a dashboard on SASViya® (SAS Visual Analytics), which assists users in evaluating the results effectively. As the datasets are moving solely between SAS products throughout the entire process, the data incompatibility issue is minimized.

## INTRODUCTION

Oklahoma State University (OSU) schedules roughly 200 mid-term and final common exams. The requirements for these exam schedules are not too complex but many are exam specific and may change over time. These requirements are known best by the Registrar Office staff, who may not be familiar with SAS programming.

This paper outlines the process of extracting users' requirements from an input spreadsheet and building a dynamic optimization model based on the inputs using several macro programs and an OPTMODEL procedure. The outlined program allows non-programmer users to alter and customize certain types of parameters and constraints. Moreover, the paper details how the optimized result is visualized in SASViya (SAS Visual Analytics). This visualization allows users to quickly identify issues, such as rooms with high utilization, exams with conflicting times, etc.

This paper demonstrates the solution on 83 actual final exams during finals week of Spring 2018. In practice, this program will be run twice, once for mid-term and once for final exams. Most constraints and parameters used in this demonstration are actual requirements. A few others are simulated to demonstrate the program's flexibility. The paper will briefly describe the input data, the general constraints, objective types for the exams, and the optimization model. These are all specific to Oklahoma State University's requirements. Then, snippets of the codes used to produce the dynamic optimization model will be presented. Lastly, the paper will discuss the visualization dashboards and the solution's performance.

## DATA DESCRIPTION

Inputs for the optimization model come from three datasets and a source table. The three datasets contain the exams' and rooms' information, which define the optimization model's objectives and fixed constraints. The source table is used to construct the user-defined constraints dynamically.

### THREE INPUT DATASETS

The three datasets contain information aggregated from OSU's database, which are detailed as following.

1. Dataset Enrollment: identifies each exam (variable ExamID) and number of enrolled students (variable Enrollment).
2. Dataset Capacity: identifies each unique room (variable RoomID) and its capacity (variable Capacity).

3. Dataset Objective: includes all combinations of exams, rooms, times, and the preference weights that coordinate to such combination of exam, room, and time. Preference weights incentivize the optimization model to assign the exams to their preferred rooms and time blocks. Each preference weight is the sum of three preference types as following.

- General Preference type: incentivizes the model to assign exams to their preferred time and location combination. At OSU, the earlier the exam scheduling application is turned in, the higher this general preference.
- Time Preference type: incentivizes the model to schedule exams on the preferred time if the general preference is not achievable.
- Room and Building Preference type: the room preference incentivizes the model to schedule exams in the preferred room. The building preference incentives the model to schedule the exams in any room belonging to the preferred building if the general preference is not achievable.

The value of each preference type is assigned partially automatically (e.g. based on the application date) and partially by users. Then, these values are aggregated to form the preference weights in the objective dataset shown in Table 1. For example, exam1's time preference value for time1 is 5 and room preference value for room5 is 5. Exam1 has no building preference but a preference value of 18 for the combination of room5 and time1. Thus, the preference weight of exam1 in room 5 during time1 is 28 as shown on the first line of Table 1.

| | ExamID | ID | time1 | time10 | time2 | time3 | time4 | time5 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 5 | 28 | 28 | 28 | 64 | 28 | 28 |
| 2 | 1 | 9 | 20 | 20 | 20 | 47 | 20 | 20 |
| 3 | 1 | 10 | 20 | 20 | 20 | 20 | 20 | 20 |

**Table 1: Snapshot of Objective Dataset's First Few Observations and Variables**

### SOURCE TABLE – SOURCE 3

The Source 3 table is imported from an Excel sheet in which a user defines the wanted constraints, the exam(s) the constraints are applied to, whether the constraints are negative, related time, and location.

The user-defined Excel sheet is shown in Table 2. In this example, the user has defined three constraints: the first prohibits exam1 to locate in room9, the second prohibits exam2 to be at 12pm on May 11th, and the last prevents exam20 and exam21 to locate next to each other.

| Exam1 ID | Negative Req | Req Category | Exam2 ID | Time | Date | Room ID |
|---|---|---|---|---|---|---|
| 1 | no | locate in | | | | 9 |
| 2 | no | at | | 12:00 | 5/11/2018 | |
| 20 | | not locate next to | 21 | | | |

**Table 2: User-Defined Excel Sheet Sample**

## GENERAL OBJECTIVE, CONSTRAINT TYPES, AND OPTIMIZATION MODEL

This section details the model's decision variables, objectives, and constraints that are deduced from Oklahoma State University's current and potential needs.

### DECISION VARIABLES

Each decision variable (denoted as assignment) is a combination of an exam (i), a room (k), and a time block (j). All decision variables are binary, having a value of 1 if a certain exam (i) is scheduled in a certain room (k) during a certain time block (j) and 0 otherwise. The decision variables are written as:

```
var Assignment {exams, times, rooms} binary;
```

## OBJECTIVE FUNCTION

The objective function maximizes all exams' assignment preferences. Each exam i's assignment preference is the product of the preference weight (denoted as preference) of exam i in room k during time block j and whether exam i is assigned room k during time block j.

As mentioned in the Data Description Section, each preference weight is the combination of the general, time, room, and building preferences. The objective function is written as:

```
max TotalPreference = sum {i in exams, j in times, k in rooms}
   Preference[i,j,k] * Assignment[i,j,k];
```

## CONSTRAINT FUNCTIONS

Constraints are either model-required or user-defined. Model-required are hard constraints that are either strictly required for every exam or are needed for the optimization model to perform the assignment correctly. User-defined constraints can be altered, add, or deleted over time. For OSU's scheduling problem, there are three model required constraints and four general types of user-defined constraints.

### Model Required Constraints

1. Model required constraint 1:

An exam can only be assigned to rooms that have at least as many seats as the exam's enrollment. Thus, enrollment [i] (number of students enrolled in exam i) must be less than or equal to capacity [k] (capacity of room k) if exam i is assigned to room k during any time block (j). The constraint is written as:

```
con req_cap {i in exams, k in rooms, j in times}:
   enrollment[i] * Assignment[i,j,k] <= capacity[k];
```

2. Model required constraint 2:

An exam is assigned to precisely one room and one time block. Thus, for each exam i, all assignments of this exam to all rooms (all k) during all time blocks (all j) must total to 1. The constraint is written as:

```
con req_exam {i in exams}:
   sum {j in times, k in rooms} Assignment[i,j,k] = 1;
```

3. Model required constraint 3:

Each room can host at most one exam during a time block. Thus, for each combination of room k and time j, the sum of all assignments related to that combination of room and time must be less than or equal to 1. The constraint is written as:

```
con req_room {k in rooms, j in times}:
   sum {i in exams} Assignment[i,j,k] <= 1;
```

### User-Defined Constraints

A user may specify any or all of the three user-defined constraint types. Each constraint type 1, 2 and 3 can be either a regular or a negative constraint. Each constraint type is detailed as following.

1. Type 1 – Location Constraint:

An exam (i1) must be in a certain room (k1) but can be at any time (any j) (function 1). The negative version of this type is: an exam (i2) cannot be in a certain room (k2) at any time (any j) (function 2).

- Function 1: All assignments related to the combination of i1 and k1 must total to 1. i1 and k1 are user-defined constants. The constraint is written as:

  ```
  con locCon: sum{j in times} Assignment[i1,j,k1] = 1;
  ```

- Function 2: All assignments related to the combination of i2 and k2 must total to 0. i2 and k2 are user-defined constants. The constraint is written as:

```
con locCon: sum{j in times} Assignment[i2,j,k2] = 0;
```

2. Type 2 – Time Constraint:

An exam (i1) must be at a certain time (j1) but can be in any room (any k) (function 3). The negative version is: an exam (i2) cannot be at a certain time (j2) regardless of its location (any k) (function 4).

- Function 3: All assignments related to the combination of i1 and j1 must total to 1. i1 and j1 are user-defined constants. The constraint is written as:

```
con timCon: sum{k in rooms} Assignment[i1,j1,k] = 1;
```

- Function 4: All assignments related to the combination of i2 and j2 must total to 0. i2 and j2 are user-defined constants. The constraint is written as:

```
con timCon: sum{k in rooms} Assignment[i2,j2,k] = 0;
```

3. Type 3 – Location and Time Constraint:

An exam (i1) must be in a certain room (k1) and at a certain time (j1) (function 5). The negative version is: an exam (i2) must not be in a certain room (k2) and not at a certain time (j2) (function 6).

- Function 5: Assignment[i1, j1, k1] is always equal to 1. i1, j1 and k1 are user-defined constants. The constraint is written as:

```
fix Assignment[i1,j1,k1] = 1;
```

- Function 6: Assignment[i2, j2, k2] is always equal to 0. i2, j2 and k2 are user-defined constants. The constraint is written as:

```
fix Assignment[i2,j2,k2] = 0;
```

4. Type 4 – Proximity to Others:

An exam (i1) must not be located next to another exam (i2) if the two exams are scheduled at the same times. As rooms with adjacent IDs are next to each other, if exam i1 is assigned to room k, exam i2 must not be assigned to room k+1. Thus, Assignment[i1, j, k] and Assignment[i2, j, k+1] must total to less than or equal to 1. Likewise, Assignment[i2, j, k] + Assignment[i1, j, k+1] must total to less than or equal to 1 as this constraint implies that exam i2 must not be next to i1 either. As the model required constraint 2 and 3 prohibit an exam to be assigned twice or a room to host two exams at once, the sum of all four assignments must be less than or equal to 1 as in function 7.

- Function 7: i1, i2 are user-defined constants and &Rcount specifies the total number of rooms. The constraint is written as:

```
con nextCon {j in times, k in rooms: k < &Rcount}:
   Assignment[i1,j,k] + Assignment[i2,j,k+1]+ Assignment[i2,j,k] +
   Assignment[i1,j,k+1]<=1;
```

## SAS PROGRAMS AND PROCEDURES

This section details the steps of the SAS program. A user will key in the desired time parameters (startT, endT, startD, and endD), which define the time period the user wishes to schedule the exams in, and execute the main macro. The main macro then calls other macros that create the input datasets, create the user-defined constraint code lines, and execute the procedure OPTMODEL with MIXLP solver option. An example on how the main macro is called for execution is:

```
%Macro Main (startT='12:00't, endT='16:00't, startD='7May2018'd,
endD='11May2018'd, blockD='4:00't);
```

### OVERVIEW

The main macro is defined as (lines to include Macro 1 and 2 are not shown):

```
/* Declare main macro */
```

```
%Macro Main (startT=, endT=, startD=, endD=, blockD=);

/* Location of other macros */
FILENAME macInput "C:\Users";

/* Macro 1 and 2: define the time period and create the three datasets */
/* … */

/* Macro 3 to create the code lines for user-defined constraints */
%include macInput(UserDefinedConstraints)/source2;
%UserDefinedConstraints;

/* Macro 4 to execute the optimization model */
%include macInput(optimizationModel)/source2;
%OptimizationModel;
%Mend Main;
```

## MACRO 1 AND 2 – DEFINING TIME PERIOD AND CREATING THREE INPUT DATASETS

Macro 1 converts each exam's start time and end time to a time block. All exams will be scheduled within the defined time periods, starting from the start time (variable startT) of the start date (variable startD) until the end time (variable endT) of the end date (variable endD). The blockD variable defines the duration of each time block.

Macro 2 aggregates inputs from the database to create the three input datasets. Procedures used are those commonly used for shape manipulation, such as PROC TRANSPOSE and PROC SQL.

## MACRO 3 – CREATING USERS-DEFINED CODE LINES

This macro checks if any of the three user-defined constraint types is specified in Source 3 (the Excel spreadsheet). If there is any, Macro 3 calls the appropriate macro to process that constraint type. Then, it compiles the UserDefined macro that generates the scripts for existing user-defined constraints within the PROC OPTMODEL. The codes of Macro 3 are not shown. The codes of the UserDefined macro along with the macros used to process constraint type 1 and 4 are detailed below. Macros to process type 2 and 3 are similar to that of type 1 and are not shown.

### Macro UserDefined

This macro checks the existence of the constraint types and generates codes for the existing constraint types. Only the codes for type 1 and 4 are shown as:

```
%macro userdefined;
/* If type 1 exists, write the macro variables with type 1 code lines */
/* &LocCnt is the count of type 1 constraint(s) */
/* &&locCon&i is the macro that holds the code lines that defines each
type 1 constraint */
%if &LocCnt ne 0 %then %do;
   %Do i = 1 %to &LocCnt;
      &&locCon&i;
```

```
   %end;
%end;
/ *…* /
/* If type 4 exists, write the macro variables with type 4 code lines */
/* &nxtCnt is the count of type 4 constraint(s) */
/* &&nextCon&i is the macro that holds the code lines that defines each
type 4 constraint */
%if & nxtCnt ne 0 %then %do;
   %Do i = 1 %to &nxtCnt;
      &&nextCon&i;
   %end;
%end;
%mend userdefined;
```

**Macro Processing User-defined Constraint Type 1**

Source 3 (obtained from the Excel spreadsheet) is split into several datasets by the field Reg_Category. Each set contains one of the three user-defined constraint types. Part 1 details the codes used to generate the code lines dynamically. Part 2 shows a sample code line generated from part 1's codes.

1. Codes Generating Dynamic Code Lines:

```
%macro firstCons;
/* Create macro variables exam, room, and negative that contain information
 on all type 1 constraints */
proc sql noprint;
   select exam1_ID, room_ID, Negative_Req into :exam SEPARATED by ",", :ID
   SEPARATED by ",", :Negative SEPARATED by ","
   from locateCons;

/* Create locateCons to count the number of constraints with this type */
   select count(*) into :LocCnt from locateCons;
quit;

/* Loop through each constraint to generate the macro variables with
dynamic scripts */
%Do i = 1 %to &LocCnt;
   %global locCon&i;
   %let x = %scan("&exam", &i, ",");
   %let y = %scan("&ID", &i, ",");
   %let z = %scan("&Negative", &i, ",");

/* If the requirement is: Exam i must be in this room (Negative_req = is),
store function 1 into the locCon&I macro variable */
```

```
   %if &z=is %then
       %let locCon&i = con locCon&i: sum{j in times} Assignment[&x,j,&y] =
       1 %str(;);

/* If the requirement is: Exam i must not be in this room (Negative_req =
no), store function 2 into the locCon&I macro variable */
   %else
       %let locCon&i = con locCon&i: sum{j in times} Assignment[&x,j,&y] =
       0 %str(;);
%end;
%mend firstCons;
```

2. Sample Code Line Generated by Part a Codes

Example 1 below is used illustrate part 1's codes. In this example, a user defines in the Excel sheet (source table 3) that exam1 should not locate in room 9.

| Exam1 ID | Negative Req | Req Category | Exam2 ID | Time | Date | Room ID |
|---|---|---|---|---|---|---|
| 1 | no | locate in | | | | 9 |

**Table 3: Example 1 – Location Constraint**

With example 1, part a's codes will generate one macro variable locCon1 with the value of:

```
con locCon1: sum{j in times} Assignment[1,j,9] = 1;
```

**Macro for User-defined Constraint Type 4**

Similar to the previous macro, this macro generates the code lines depending on users' requirements. Part 1 below outlines the codes used to generate the dynamic code lines. Part 2 details a sample code line generated from part 1's codes.

1. Codes Generating Dynamic Code Lines:

```
%macro fourthCons;
/*Create macro variables exam, room, and negative that contain the
information on all type 4 constraints*/
proc sql noprint;
   select exam1_ID, exam2_ID
   into :exam1 SEPARATED by ",", :exam2 SEPARATED by ","
   from LNCons;

 /*Create macro nxtCnt to count the number of constraints with this type */
   select count(*) into :nxtCnt from LNCons;
quit;

/*Loop through each constraint to store appropriate code lines into macro
variable(s) */
%Do i = 1 %to &nxtCnt;
   %global nextCon&i;
```

```
   %let x = %scan("&exam1", &i, ",");
  %let y = %scan("&exam2", &i, ",");
   %let nextCon&i = con nextCon&i {j in times, k in rooms : k<&countR}:
   Assignment[&exam1,j,k] + Assignment[&exam2,j,k+1]+
   Assignment[&exam2,j,k] + Assignment[&exam1,j,k+1]<=1 %str(;);
%end;
%mend fourthCons;
```

2. Sample Code Line Generated by Part 1's Codes

Example 2 below is used illustrate part 1's codes. In this example, a user defines in the Excel sheet (source table 3) that exam20 should not be located next to exam21.

| Exam1 ID | Negative Req | Req Category | Exam2 ID | Time | Date | Room ID |
|---|---|---|---|---|---|---|
| 20 | | not locate next to | 21 | | | |

**Table 4: Example 4 – Proximity Constraint**

With example 2, part 1's codes will generate one macro variable nextCon1 with the value (&Rcount stores the total number of rooms):

```
con nextCon1 {j in times, k in rooms: k < &Rcount}:
Assignment[20,j,k] + Assignment[21,j,k+1]+ Assignment[21,j,k] +
Assignment[20,j,k+1]<=1;
```

## MACRO 4 – EXECUTING THE OPTIMIZATION MODEL USING PROC OPTMODEL

The SAS/OR PROC OPTMODEL is used to optimize the schedules using a Mix Integer Solver. The three datasets - enrollment, capacity, and objective - are input into the model. The UserDefined macro is then called to generate the scripts for the user-defined constraints dynamically. The PROC OPTMODEL codes below are an adaptation from the Assignment problem in *SAS/OR® 13.2 User's Guide: Mathematical Programming the OPTMODEL Procedure*.

```
%Macro OptimizationModel;
proc optmodel;

/* Declare index sets */
set exams;
set <str> times;
set rooms;

/* Declare parameters */
num preference {exams, times, rooms};
num enrollment {exams} init 0;
num capacity {rooms} init 0;

/* Read the set of times */
read data time into times=[timeBlock];
```

```
/* Read the set of exams and enrollment */
read data enrollment into exams=[examID] enrollment;
/* Read the set of rooms and capacity */
read data capacity into rooms=[roomID] capacity;
/* Read preference data */
read data objective into [examID roomID] {j in times}
< preference[examID, j, roomID]=col(j)>;

/* Declare the model */
var Assignment {exams, times, rooms} binary;

/* Objective function */
max TotalPreference = sum {i in exams, j in times, k in rooms}
preference [i,j,k] * Assignment[i,j,k];

/* Model-required constraint type 1 */
con req_cap {i in exams, k in rooms, j in times}:
enrollment[i] * Assignment[i,j,k] <= capacity[k];

/* Model-required constraint type 2 */
con req_exam {i in exams}:
sum {j in times, k in rooms} Assignment[i,j,k] = 1;

/* Model-required constraint type 3 */
con req_room {k in rooms, j in times}:
sum {i in exams} Assignment[i,j,k] <= 1;

/* User-defiend constraints code lines generated by macro UserDefined*/
%UserDefined;

/* Solve with Mix Integer LP algorithm */
solve with MILP;

/* Write results into dataset solution*/
create data solution
from [exam time room] = {i in exams, j in times, k in rooms}
assigned=Assignment;
```

```
quit;
%mend OptimizationModel;
```

### SOLUTION PERFORMANCE

The program executes much faster comparing to the old manual process. To schedule 83 classes into 40 rooms during 10 time blocks with 10 user-defined constraints, the main macro executes in under 15 minutes. With this project, the model has optimized 33,200 decision variables, which would be difficult for manual processing.

## RESULT VISUALIZATION WITH SAS VIYA – SAS VISUAL ANALYTICS

The resulting dataset from the model is joined back with the source files to obtain information such as the exact exams' times (instead of time blocks), exams' names, buildings, department, etc. Two dashboards are made to visualize the results. The first dashboard shows the exams' schedules only. The second shows the overall room utility when the exams are scheduled along with other classes.

### EXAM INFORMATION DASHBOARD

This dashboard incorporates only exam schedule details. From the calendar-like example below, users can see that most exams are scheduled on May 8, 2018 (Figure 1). Users can also drill down to certain exams as in Figure 2 by changing the options on the right panels. In this view, users can quickly identify existing issues. For example, as in the Figure 2, exam 5, 6, 7, and 12 are scheduled at the same time in four rooms that are next to one another. Users can assess whether this is a desirable situation and may add another constraint to avoid this assignment.

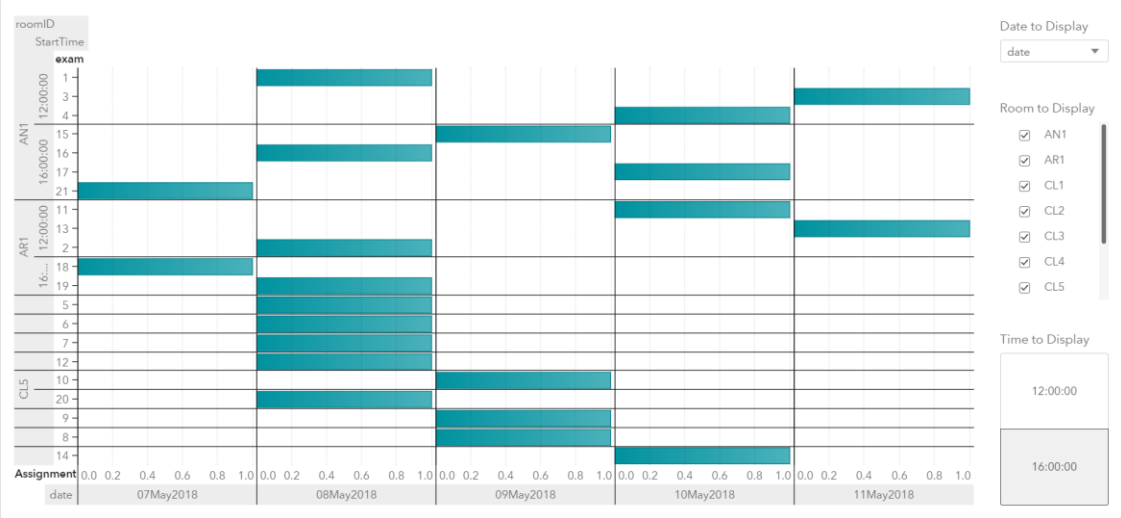


**Figure 1: Exams' Information Overview**

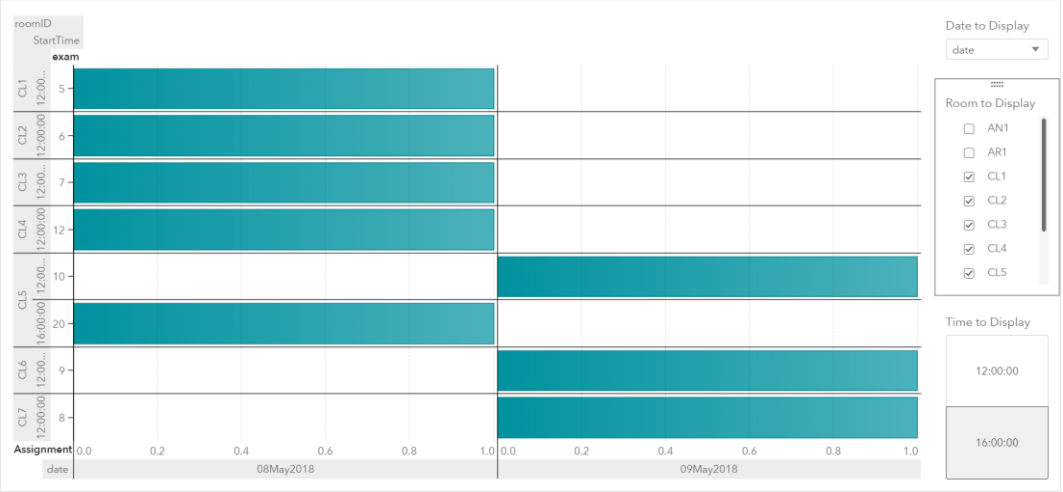


**Figure 2: Exams' Information Drilled Down View**

## ROOMS' UTILITY DASHBOARD

This dashboard shows the utilization of each building and room. The bar shows the percentage of time a room is occupied while the line shows the rooms' seat utilization rate, which indicates whether the rooms are used for exams or classes with enrollment much lower than the rooms' capacity. Double clicking on any department in the department view (Figure 3) will generate the drilled down view on each building (Figure 4) or each room (not shown).

In this example, users can identify whether a room is under or over utilized during a specific time by selecting the options on the left side panel. With this information, users can adjust the preferences on each exam to encourage the model to schedule exams in rooms that have lower utilization rate. Likewise, if a room has low seat utilization rate, users can decrease the location preferences of exams with fewer enrollments on this room. With the dashboards, a user can test out different combination of preference weights and constraints and are able to view the results quickly.

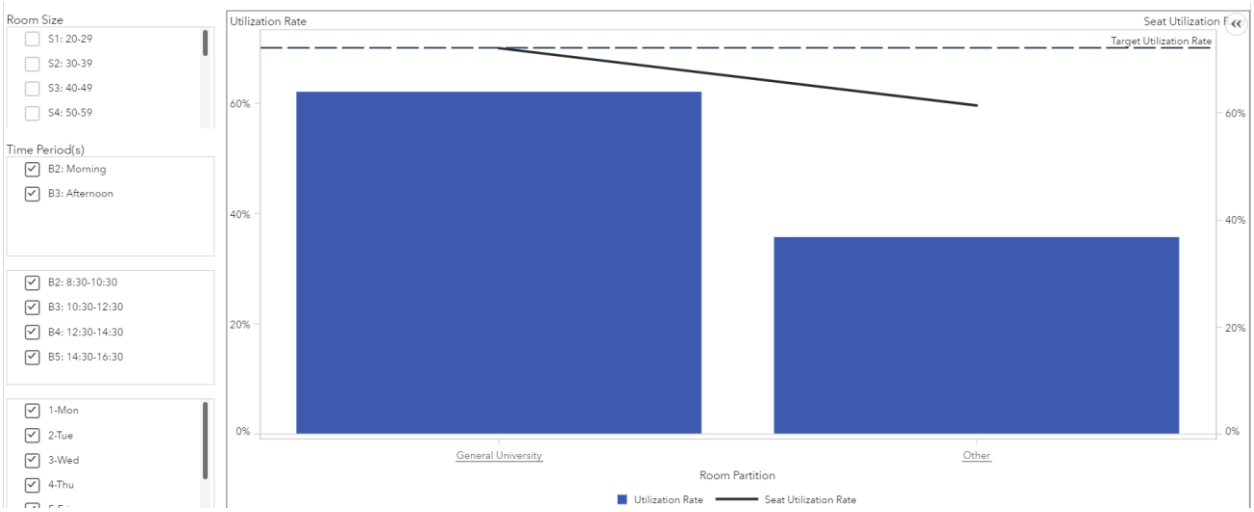


**Figure 3: Room's Utility Dashboard – Department (also Referred to as Partition) View**

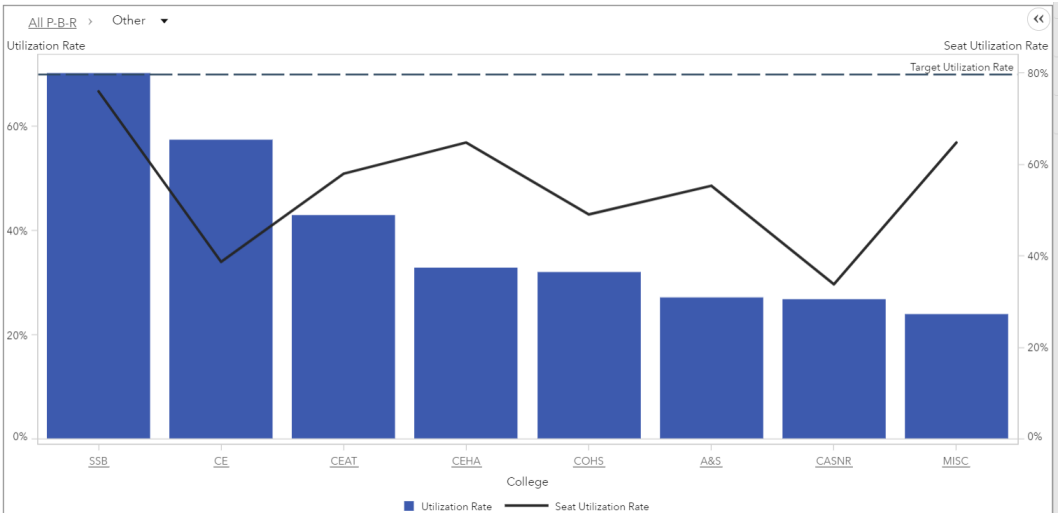


**Figure 4: Room's Utility Dashboard – Building (also Referred to as College) View**

## CONCLUSION

The exam scheduling process at Oklahoma State University can be automated and optimized using several SAS products. This paper has introduced a solution that aggregates user-defined information from outside sources to create a dynamic optimization model. Thus, the optimization model can be altered without changing the codes. This solution also offers accompanied dashboards that help users identify areas for adjustment.

## REFERENCES

SAS Institute Inc. 2014. SAS/OR® 13.2 User's Guide: Mathematical Programming. Cary, NC: SAS Institute Inc.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Thu Nguyen
Oklahoma State University
thunguy@okstate.edu