Aviation-Meteorology Statistical Analysis and Pediatric Residency Scheduling

Final Report

Abstract

This paper summarizes Ji Wang's research work with the guidance of Professor Amy Cohn at Center for Healthcare Engineering and Patient Safety (CHEPS), University of Michigan.

During my 12-week work period at CHEPS, I worked on two projects, namely the Aviation Statistical Analysis, and Pediatric Emergency Department Residency Scheduling (PEDS). The two projects took approximately the same amount of time, hence this paper will cover both of them.

Wang, Ji
Industrial and Operations Engineering Department
University of Michigan

Faculty Advisor: Professor Amy Cohn

September 2014

Table of Contents

Table of Figures

I. Aviation Statistical Analysis

1. Overview

Airline industry is an important building block in economy and people's daily life. However, flight delays frequently impedes the smooth operation of the airborne transportation, causes airlines' loss in money, and damages their customer experience. In addition, serious flight delays may badly influence the journeys of passengers, especially for those who was to take connecting flights.

The aviation project at CHEPS aims at discovering the relationship of flight delays against multiple meteorology factors, and applying the findings to forecast possible delays in airline industry. For instance, the departure performance of flights might deteriorate if the visibility at the origin airport is low at the scheduled departure time.

My major job on the Aviation team was to investigate the influence of ceiling on the on-time performance of flights. The ultimate goal was to find a pattern on how the flight responds to the change of the ceiling level, and try to quantify the pattern and use it to forecast the future delays. The research during the summer did not proceed to the stage where quantitative model can be built and used to forecast flight delays based on weather. However, some notable findings will benefit the future research on the aviation project.

2. Background

i. Database of Aviation Project

Databases containing the necessary data for aviation analysis are built on a MySQL server. Two major parts of the databases are the flight database and the weather database.

There are 11 years of domestic flight data stored in the database. The team obtained these data from Research and Innovative Technology Administration, Bureau of Transportation Statistics (RITA BTS), which provided data of all transportations to the public. The Aviation has loaded into the flight database all domestic flight data from 2003 to 2013, covering all airlines in the BTS dataset.

The size of the flight database is 73,814,038. Each record corresponds to a single flight, and contains the flight date, scheduled departure time, carrier, origin, destination, and tail number, and the time of departure delay, taxi in, taxi out, arrival delay, scheduled elapsed time, and the actual air time.

Correspondingly, there are 11 years of meteorology data from 2003 to 2013 to match the flight data. The team obtained these data from National Oceanic and Atmospheric Administration (NOAA).

The size of the weather database is 45,822,843. Each record corresponds to a weather measurement, and contains the time and location information, as well as important meteorology measurements, including the wind direction, wind speed, air temperature, dew point temperature, visibility, ceiling, and sea level pressure.

ii. Definition of Delay and Cancellation

In our project, the term "delay" refers to the situation where the actual departure or arrival time is behind the scheduled departure or arrival time for 15 minutes or more; the term "cancellation" refers to the flights that do not actually fly the scheduled piece of route.

iii. Ceiling

In meteorology, the term "ceiling" refers to the height of the cloud base for the lowest broken or overcast cloud layer. (NOAA, 2009) The lower the ceiling, the more restricted is the aircraft operation, either when departing or landing.

For safety reasons, a group of planes whose departure was scheduled at or after the time when a low ceiling was measured might stay on the ground until the ceiling returns to the normal level, or they might take off at slower pace and longer intervals than when the sky is clear for safety reasons. This will probably delay the departure of flights, especially when there are multiple flights ready to leave the ground at normal intervals.

This idea inspired the team and an analysis task was brought up to investigate the relationship between the ceiling and delay of flights.

iv. Linking the Flight and Weather Table

The analysis requires the combination of the flight and weather data, since the impact of weather on flight on-time performance is under investigation. As recommended by Mr. Lonny Hurwitz and Mr. Keiron McEwen at Southwest Airlines, who are collaborating with the Aviation team on multiple projects, we should link each flight to the closest weather data in terms of time and location of our interest. For instance, to analyze the weather impact on departure performance of the flight, the closest weather record at the origin airport before the scheduled departure time should be taken.

3. Data Preparation

The time and location scope was narrowed down for an easy start of the analysis. To analyze a dataset with almost 73 million records all at once is beyond the team's capability. In addition, the dataset covers nationwide flight data. To treat all airports at one time can be misleading because of the underlying geographical and hence the climatological difference from place to place. For instance, the departure performance is not very likely to respond to the ceiling change in the same pattern at Alaska and Florida.

Based on these consideration, it was decided to study four major airports from different regions in a smaller time scope. The four chosen airports were: Baltimore–Washington International Airport (BWI), Dallas Love Field (DAL), San Diego International Airport (SAN), and San Francisco International Airport (SFO); the time scope was 2009-2012.

The fields extracted for flight data include: date of the flight, scheduled departure time at the origin, the carrier, the airport code of both the origin and the destination, the departure delay

in minute, the taxi out time in minute, and the binary cancellation label. The fields extracted for weather data include ceiling.

4. Data Analysis

The data were first grouped and averaged by month, in order to build an intuitive view of how the ceiling affects the on-time performance of departure, and to verify or disprove our assumption that the on-time performance is positively correlated with these two measurements. In the following text the SFO 2011 data will be used as an example, since this dataset has been one of the most promising ones where we can obtain appreciable results.

The key performance indicator here was the percentage of flights that are not on time among all flights, named as "non-on-time percentage" in the following text. These flights include all the delayed and cancelled flights. For the on-time performance to be positively correlated with ceiling, the non-on-time percentage is supposed to be negatively correlated with ceiling.

i. Trend over the year

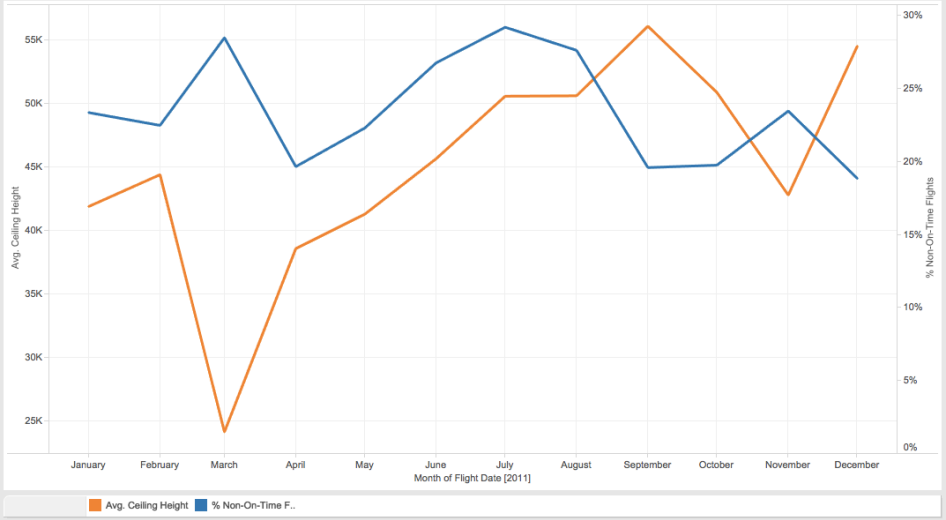

Figure 1 Abnormal Flight Percentage v.s. Ceiling & Visibility SFO 2011

In Figure 1, the yellow line refers to the monthly averaged ceiling over the year of 2011 at SFO, and the blue one is the monthly-averaged non-on-time percentage of flights.

From this graph it was suspected that the impact of ceiling on the departure performance may be different depending on the season or the month. Specifically at SFO, the ceiling has a strong influence on the departure performance during fall and winter, when the departure performance deteriorates when ceiling drops. Hoping to find how the response can be modeled, we studied the detailed statistics of the year excluding April to August.

ii. Correlation Value using Monthly-Averaged and Daily-Averaged Data

For the time of the year when the correlation seems obvious, the Pearson's correlation value was calculated as shown in Figure 2. A perfectly fitted line was plotted at SFO in 2011, giving a correlation value of -0.99.

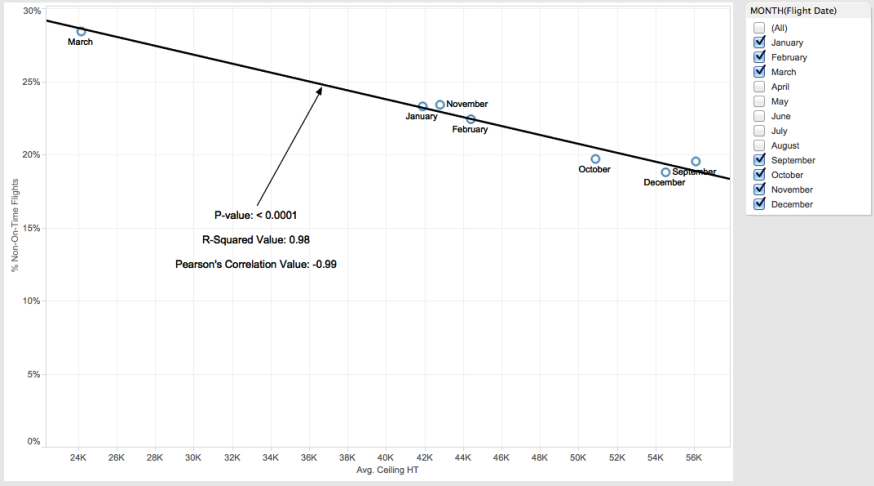

Figure 2 Fitted Line of Delay Percentage vs Ceiling, SFO 2011 (Whole Year)

However, the monthly-averaged data provide only 12 data point in the plot, and we filter out 5 of them because the spring-summer data did not show much correlation. The sample of size 7 is too small for the plausible p-value and correlation to be convincing.

Nevertheless, this -0.99 correlation provides a strong implication that there could be good correlation between the ceiling and the departure performance in winter and fall months, even with a larger sample of data.

To verify this, the time unit was further reduced to days, and the same procedure was repeated for the daily-averaged ceiling and non-on-time percentage. This time there are 212 data points, which should be abundant in size. The correlation is now -0.72, which implies strong negative correlation between the ceiling and the non-on-time percentage.

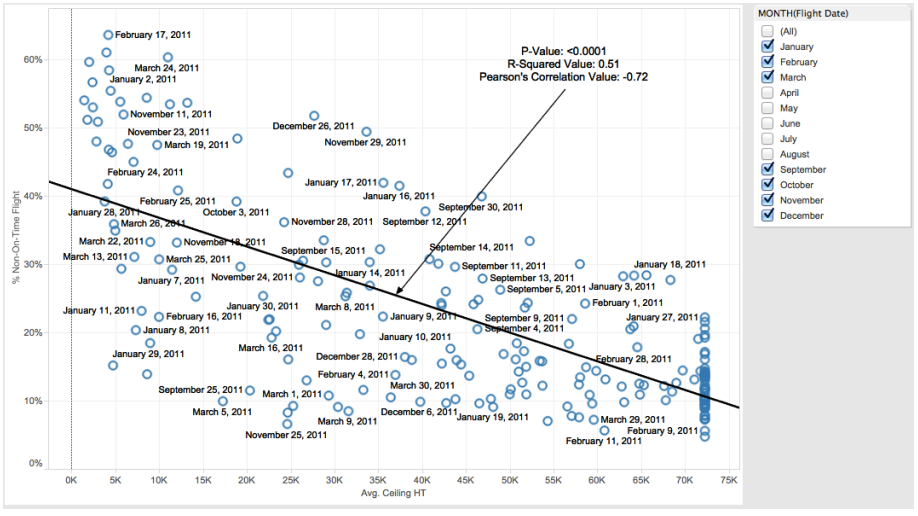

Figure 3 Fitted Line of Delay Percentage vs Ceiling, SFO 2011 (Winter and Spring)

iii. Summary

The correlation results from all the chosen dataset were listed in the following table. It is noteworthy that the correlation values of non-on-time percentage and ceiling at DAL and BWI are generally between -0.3 and -0.4, which shows weak correlation. At SFO, however, the correlations are basically between -0.6 to -0.7, which indicates a moderate to strong correlation.

Table 1 Summary of Best Correlation Values at Three Major Airport 2009-2012

Airport	Year	Month	Correlation
DAL	2009	10-12	-0.38
	2010	10-12	-0.34
	2011	9-10	-0.38
	2012	1 & 11-12	-0.33
BWI	2009	2-5	-0.35
	2010	3-4 & 9-12	-0.41
	2011	1-5 & 8-12	-0.35
	2012	1-2 & 9-12	-0.39
SFO	2009	1-3 & 9-12	-0.66
	2010	1-3 & 9-12	-0.69
	2011	1-3 & 9-12	-0.72
	2012	1-3 & 9-12	-0.58

5. Conclusions and Future Plans

At this point the summer research work has come to an end. We've investigated the influence of ceiling on departure performance during 2009-2012 at BWI, DAL, SAN, and SFO. The research shows that there is likely a clear relationship at SFO, while at the other three locations the relationships are vague. We haven't reach the point where we can accomplish our ultimate goal, that is, to quantify the relationships with confidence, and to build a tool to forecast delay occurrence based on the ceiling observations.

These research results bring about the promising alternatives we can take in the future to further study the aviation-meteorology data:

i. To include arrival performance

The summer research focused on the influence of ceiling on the departure performances. The basic reason behind this was that the SQL query to obtain the departure data from the database was technically much easier and more efficient than the one for arrival data, based on the current database structure. In the future, we can modify the database so that an easy acquisition of arrival data is possible. Then we can investigate the ceiling influence on arrival performances as well.

ii. To incorporate multiple weather factors

One limitation of the summer research work was that we considered only the ceiling.

Our starting point was to figure out whether ceiling could be a potential influence on departure delays, and if yes, we would include the ceiling measurements into our forecast model in the future. Our results demonstrate noticeable correlation between ceiling and departure performances, at least within the sample we chose. Therefore, it would be reasonable to take ceiling into consideration in the future.

However, it is highly possible that multiple weather factors are influencing the flight on-time performances, more than just ceiling. The statistics we obtained could only support the weak to moderate correlation between the ceiling and departure performance, but could definitely not prove any causality or provide any quantitative descriptions. The weather factors, including ceiling, visibility, dew-point temperature, etc., may depend on one another, and it would be wrong to forecast the on-time performance with a single weather measurement.

Therefore, a research of a larger scale should be launched to build the forecast model. We would need to examine each possible weather impact, and take their interdependence into consideration. Only after that can we start to build quantitative models.

II. Pediatric Residency Scheduling

1. Overview

At the pediatric emergency department (PED) of C.S. Mott Children Hospital, the residents follow a pre-defined monthly schedule that determines their work time. These monthly schedules typically cover the time from the 27th of the last month to the last day of the current month. Foe every day in the covered period, there are 8 shifts, all of which have to be assigned to one resident, except for the noon shift, which is optional (termed as "flexible" at the hospital).

A schedule is not complete with the mere coverage of all shifts on every day. There are all kinds of restrictions to build a schedule, some as the hospital requirement, and others as residents' request. For instance, some resident cannot work during the time when they have weekly conference or clinics out of town; first-year residents cannot work on the first and last shift of the day because they lack the experience and efficiency; and almost every residents have a preferred period for vacations. The schedule has to fill every day-shift spot in the month without conflicting with any of the restraints, which becomes the hardest part of the process.

Traditionally, one of the chief residents at the PED, who take care of the administrative issues inside the department, is responsible for building the schedule of residents each month. This takes the chief resident around 25 hours of work per month, and involves laborious trial-and-error approaches to satisfy the need from both the hospital side and the resident side. Mistakes occur often, and not always could the chief resident complete a perfect schedule that caters to everyone's need. Chief Residents are one of the most relatively expensive human resources at the hospital and using a significant portion of their time each week to perform scheduling tasks may not be an efficient and effective use of that resource. (CHEPS, 2014, p. 9)

Coming to the rescue, the CHEPS started a program to help build the monthly schedule at the ED by utilizing the knowledge of linear programming and computer science implementation. The program started as a Pediatric Emergency Department Scheduling (PEDS) team two years ago, and have helped build monthly shift schedules with the chief residents ever since.

My major job on the PEDS team was to design a template that accommodates linear programs formulations generally, and to revamp the previous C++ code that builds monthly schedules. The template construction utilized knowledge of C++ object-oriented programming, and the interface of IBM Cplex optimizer. The PEDS code revamp also required the knowledge above, plus the basic linear programming techniques.

2. Background

i. Linear Programming

Linear programming (LP), also known as linear optimization, is a mathematical technique to provide the optimal solution to problems. The LP model describes the problem as comprised of **decision variables** to modify, **objective function** to optimize, and **constraints** to conform to. It is termed as "linear" because the objective function and the constraints have to be modeled as the linear functions with respective to the decision variables. (IBM, Linear programming)

LP problems are solved with the simplex method. The simplex method tries to find the optimal solution among the feasible solutions to the problem. (Morgan, 1997) The solution comes in the form of the complete assignment of all the decision variables.

In practice, LP problems are often solved by software packages able to use the simplex method. Typical tools include the Microsoft Excel embedded with simplex solver, and IBM ILOG CPLEX Optimization Studio, that will be introduced in the section iii.

ii. CHEPS LP Projects

At CHEPS, several research projects involves LP, such as University of Colorado Hospital resident scheduling project, and general surgery block scheduling project. These projects use LP to model their problem, most of which are schedule making, and utilize the IBM CPLEX solver to optimize the solution.

iii. IBM ILOG CPLEX Optimization Studio

The IBM ILOG CPLEX Optimization Studio (Cplex) is a software package that solves linear optimization problems using the simplex method. It provides optimal solutions to LP problems, and helps the users to make the best decisions.

Cplex provides interfaces to multiple programming languages. Theses languages include C++, C#, Java, Python, Microsoft Excel, and MATLAB. (IBM, CPLEX Optimization Studio Interfaces, n.d.) At CHEPS, the C++ and Java interfaces are used because the program written in C++ is robust, easy to maintain, and involves moderate difficulty for student programmers.

The PEDS residency scheduling project uses the C++ interface to connect to Cplex and solve the linear program. C++ codes are written to convert mathematical formulations Cplex input by means of the ILOG Concert Technology. (IBM, CPLEX Optimization Studio Interfaces)

iv. Current Version of Code and Revamp

At CHEPS, there is a running version of the PEDS shift scheduling program, which has been generating monthly schedule over the past two years. It proves beneficial to the chief residents when building the schedules, and is still under use during the summer 2014.

However, the maintainability of the current program is not satisfactory. On one hand, the coding style of the whole program is not consistent, because it was created and modified by several student programmers who did not know each other since they worked at CHEPS during different terms. On the other hand, the comments in the current code are not sufficient enough to provide explanation on the coding details. The lack of consistency and in-text explanation makes it hard to read through the code smoothly, and to understand the thought behind the code. This eventually added to the difficulty to maintain the code efficiently.

In addition, it is desirable to utilize the template for every project for consistency. If major changes, such as on the output preference or file location paths, would ever take place related to the template, it can quickly get updated within the template for once, and applied to all projects using the template thereafter.

Based on these considerations, it was decided to revamp the current shift scheduling program.

v. My Contribution

I was among the three major members working on the LP coding template construction. To be specific, I blueprinted the template structure based on previous version of the PEDS code, and created the base class specification, which breaks the whole program into clear pieces. I also finished part of the function definitions. Lastly, I documented the instruction to use the template for future programmers at CHEPS.

I also led the PEDS team in coding the shift scheduling tool. I set the data structures according to the mathematical formulation, built robust and flexible reading functions, assigned tasks of coding the constraints within the team, and created the customized reports that help the chief resident to check and modify the schedule generated by the tool. In addition, I improved part of the mathematical formulation to enhance the efficiency of the previous version.

3. LP Coding Template Construction

i. Motivation

The motivation of building an LP coding template is to save the effort of programmers on repetitive processes, to standardize the procedure of coding LP inputs to Cplex, and to improve the maintainability of the codes.

Enhancement of Programming Efficiency

Before the summer, each project that involves LP generates the Cplex input separately and independently. Every project has its own code to complete the formulation in C++. This includes initializing the Cplex environment, reading the input files, building the LP model, solving the model, and finally outputting the results.

There are functional blocks that are essentially the same for every project, or at least resemble each other to a large extent. For example, the initialization of Cplex

environment is independent of the specific projects, and the C++ code to realize that can be the same. Noted of the time and effort spent on similar work across multiple projects, Professor Cohn decided that the teams should create a LP coding template that covers all the common parts, so that the programmers can focus on the project-specific part of the code.

Standardization of Coding Procedure

It was also decided to create a general coding framework on the specific part for each project. For instance, it was desirable to have a unique main file that works through all the procedures, each of which is unique to the specific project in detail, but similar in conceptual functionality. That way the functional blocks of each project will be forced to work in the same logical order of building the Cplex input.

The standardization helps programmers understand the LP coding as a whole, and enables them to investigate codes of other projects if ever necessary. This is beneficial to the whole research team in that student research assistants may participate in multiple projects at the same time.

Improvement on Maintainability

The great maintainability is another important expectation of the template. Professor Cohn found it desirable to be able to easily modify the code either when potential problems appeared, or when new features and requirements are needed.

With the new template being a clear framework, we can break the whole problem into specific functional units, and each unit can be revamped without running the risk of affecting the functionality of any other units. The framework should also be flexible enough to accommodate new features without affecting the existing functional blocks.

ii. Template specification

The essence of the template lies in that it provides a specific code structure to follow. This structure clearly defines the common part that cannot be modified by each individual project without permission, and also offered guidance of what have to be completed by every project to accomplish its own goal.

In practice, the template divides the program into the general section and the specific section. The general section declares a base class called CHEPS which defines all the common functional block detail; the specific section provides the outline of the specific procedures that have to be completed by each single project. The programmers have to create a derived class from the base class, and finish the specific part within the new class. The final program should operate on the derived class object to achieve the goal.

General Part Specification

The general part covers the coding detail of the shared function blocks of all projects. This provides a one-shot solution to the common functional blocks, which spares the programmers' time on repetitive work and helps them focus on the specific part. In

detail, the general part includes a main.cpp file, the base class data members, and base class method members (termed as "general functions" in the following text).

The main.cpp file belongs to the general part. It outlines the procedures the program will go through in a clear order. To be specific, the program 1) initializes the Cplex environment, 2) reads input files, 3) builds the Cplex model, 4) solves the model and outputs the solution, 5) generates reports, and 6) cleans up the memory at the end.

The data members consist of file locations, Cplex environment variables, and Cplex input parameters. The file locations include the path to the input files, and to the output folder where the program stores the results and user-defined reports. The Cplex environment variables provide concrete instance of Cplex environment that connects the C++ codes to the Cplex solver. The file location and Cplex environment is declared in the base class, and will be initialized by a single function called init(). The Cplex input parameters, also named as "general parameters", specifies the Cplex software options to solve the problem. These general parameters are declared in the base class, and will be loaded with user-defined value when the general parameter list file is being read.

The base class functions implement the shared part of the LP program. There function prototypes and descriptions are listed in the summary table below. They are all clearly defined by the base class CHEPS, and cannot be overloaded with new definitions by the derived classes. This ensures the consistency across all projects, and preludes the risk that programmers make mistakes in these building blocks. For simplicity, the function prototypes listed in the table may not appear with correct syntax in the strict sense.

Table 2 Function List in the General Part of the Template

Prototype	**Brief Description**
void init()	Initializes the Cplex environment and other data members
void read()	Reads the input files and store the inputs correspondingly, by means of implementing three sub-functions to process general parameters, customized parameters, and project data
void readGenParams()	One of the three sub-functions of read(); reads in the general parameters from formatted "genParams.txt"
void buildModel()	Builds the Cplex model by implementing three sub-functions to build variables, constraints, and objective
void buildObj()	Builds the objective expression in Cplex environment; if the user asks for only feasible solutions, then does nothing; Works under the top-level buildModel() function
void solveModel()	Solves the Cplex model, and outputs the solution
void outputCplexStats()	Outputs the statistics of Cplex per users' request
~CHEPS()	Cleans up the memory and destroys the CHEPS object
void convert_N_T(list)	Converts Cplex variables of N-dimension into corresponding T type of C++ variables; To be specific, theses "convert functions" have variations of vectors from 0 to 6 dimensions, and can convert Cplex variables into C++ int, double, or bool types.

Specific Part Specification

The specific part covers the conceptual framework in which each project will need to fill in their unique codes. It consist of functions that have to be designed by each single project according to its own interest and need. In the following text, these functions will be named as "specific functions".

Programmers are forced to code their own version of specific functions. This is ensured by declaring all the specific functions as pure virtual functions in the base class CHEPS and making the CHEPS class an abstract data type that cannot have its own instance, but have to be inherited by a derived class which finish all the definition of the pure virtual functions. Otherwise, the files will not compile.

The specific functions are listed below. Similarly to the general part, the summary table may contain incorrect syntax because of the tradeoff for paper simplicity and clarity.

Table 3 Function List in the Specific Part of the Template

Prototype	**Brief Description**
void readCustomParams()	Reads in the customized parameters from external files Works under the top-level read() function
void readMyData()	Reads in the project data from external files; Works under the top-level read() function
void writeMyData()	Outputs the data input to external files for debugging
void buildVar()	Builds variables in the Cplex environment; Works under the top-level buildModel() function
void buildSpecObj()	Builds specific objective functions in the Cplex environment; Works under the upper-level buildObj() function
void buildConstr()	Builds constraints in the Cplex environment by implementing multiple sub-constraints-functions; Works under the top-level buildModel() function
void report()	Outputs customized reports into external files
void convert()	Converts certain types of Cplex variables into corresponding C++ variables, and output them into external files per user's request Consists of possibly multiple sub-function to process different types of decision variables;
void cleanUp()	Cleans the memory space; Not necessary if no memory leak issues are involved

iii. Documentation

The template is expected to benefit all projects involved with LP at CHEPS, and hence the instruction on how to use template correctly should be documented in order that future programmers could understand it better and use it in the most efficient way.

The documentation contains quick instructions on how to use the template, necessary reminders to make the template function correctly, detailed descriptions on the data structure and class specifications, explanation on auxiliary functions, and delineation of the input file system.

4. PEDS Resident Shift Scheduling Tool Revamp

The template provides a clear direction on how the shift scheduling tool could be coded to solve a typical LP problem. The program has been broken into small pieces that were very easy to handle with basic C++ techniques, so the detail of the code will not be covered in this report.

After completing each of the required part in the template, everything is compiled with Visual Studio, and the program is created. When provided with the required group of input files, the program will run the schedule building process. If the solution is feasible based on the LP formulation and user inputs, the resulted schedule will be generated in multiple views; otherwise, the optimizer will announce the infeasibility of the problem, and provide information of constraints where the conflict is detected.

The program has been tested on August inputs, and it generated a schedule successfully. Two customized reports generated by the tool are shown below.

	27-Jul	28-Jul	29-Jul	30-Jul	31-Jul	1-Aug	2-Aug	3-Aug	4-Aug	5-Aug	6-Aug	7-Aug
7a-4p	LPH12	LPH15	LPH7	LPH4	LPH18	Si	Ma	Br	Cu	Du	EMS	Du
9a-6p	Ja	Ja	To	Ja	LPH20	Du	Wa	Wa	Co	Co	De	Co
12p-9p	Wa	LPH17	LPH16	To	LPH5	-	To	Co	Du	Le	Si	Ma
4p-1a	To	LPH10	Wa	Wa	Wa	Br	Co	To	Br	Wa	Le	EMS
5p-2a	LPH13	LPH9	LPH6	LPH11	To	Co	De	Ma	Ja	Cu	Wa	To
8p-5a	LPH8	LPH21	LPH22	LPH19	Ja	Ja	Ja	De	Ma	Br	To	Ja
11p-8a	LPH14	LPH2	LPH23	LPH3	LPH1	TPH1	Si	Si	De	Ma	Cu	De

Figure 4 August Schedule Generated by the Revamped PEDS Shift Scheduling Tool

Figure 4 shows the monthly view of the August schedule generated by the revamped program. Each shift is represented at the left as a string consisting of the start time and end time, where “a” refers to “am”, and “p” refers to “pm”. At the top are the dates of the calendar. The assignment is represented by the abbreviation of resident names in the cells. A hyphen means there is no assignment for that day-shift, only possible to appear for the “12p-9p” flexible shifts. The LPHs are placeholders, filling the shifts which will be covered by special residents outside the given roster.

This view of the schedule provides the overall information on the specific assignments of each shift. The chief resident can immediately examine which resident should work on which shift on each day of the month without confusion. If any modification is needed, the chief resident can also look at multiple residents at the same time, and then make the decision.

Tomlinson	Ped1		27-Jul	11-Aug				
	Sun	Mon	Tue	Wed	Thur	Fri		Sat
						27-Jul		28-Jul
7a-4p						I		C I
9a-6p								C
12p-9p								C
4p-1a						Tomlinson		C
5p-2a								C
8p-5a						C		
11p-8a						C I		I

	Sun	Mon	Tue	Wed	Thur	Fri	Sat
	29-Jul	30-Jul	31-Jul	1-Aug	2-Aug	3-Aug	4-Aug
7a-4p	I	I	I	I	I	I	C I
9a-6p	Tomlinson						C
12p-9p		Tomlinson			Tomlinson		C
4p-1a						Tomlinson	C
5p-2a			Tomlinson				C
8p-5a						C	
11p-8a	I	I	I	I	I	C I	I

Figure 5 Weekly Schedule of One Resident in August

In figure 5, part of the weekly schedule of a resident is provided. At the very top of the schedule is the basic information of the resident, including his name, his residency type and year, and the start/end date of his work period in the month. These information offer a quick look-up whenever the chief resident needs them for reference while checking the quality of the schedule for individual residents.

Each block in the weekly schedule is the shift assignment in a whole week for a single resident, starting on Sunday and ending on Saturday. The weekday and date are given as the columns, and the shifts are given as the rows. If the resident is scheduled to take a shift, his/her name will appear at the corresponding cell. Otherwise, the cell will be left blank or with special labels introduced below.

Additional information other than the shift assignment is overlaid on the weekly schedule. They serve as prohibitions that the resident cannot work on the day-shift of which the corresponding cell is labeled. In this snapshot, for instance, there are labels such as “I”s and “C”s in the cells. They refer to the unavailability of the resident for the labeled day-shift. To be specific, “I” represents intern prohibitions, which dictates that the first-year residents cannot work on the “7a-4p” and “11p-8a” shifts, and they should be precluded from taking these shifts; “C” refers to continuity clinic requirements, which precludes the residents from being scheduled to work at the hospital when they are already scheduled to work out of town for regular clinics.

These information provide the chief residents the reason why the residents cannot be scheduled on the labeled day-shift. When the generated schedule is not the most satisfactory, the chief resident will make minor changes manually. With the clear labels, the chief can avoid running the risk of breaking the hard rules, and manipulate the assignments by filling only the blank cells that are available for the residents.

Other views of the schedule are also created and they each provide a unique feature that helps the chief resident and the team to look at the schedule and change upon them.

5. Conclusion and Future Plans

   The revamped version of the PEDS shift scheduling tool now has the basic functionality to build monthly schedules. It is coded based on the CHEPS template, and can convert the mathematical formulation of an LP problem into C++ language, and solve the problem via the ILOG Concert Technology to Cplex.

   However, the program is not complete yet. The tool can be improved in different aspect to provide better schedules in a more efficient and user-friendly way.

   i. Constraint Generalization

      The individual constraints can be further grouped together to enhance the program maintainability.

      The current mathematical formulation contains 15 constraints. When coding these constraints, the similarity between some of them inspired me and the team. Since it is redundant to code constraints that behave similarly, and cumbersome to change every one of them if any general mistake is made, it will be a good practice to group them together, and treat them as a whole.

      Our next step is to identify the constraints that resemble each other in terms of the behavior and structure, and to write new code to convert the new generalized constraints.

   ii. Optimization Options

      The program should enable the users to choose between different result options.

      The current program produces only feasible solutions to the problem. In other words, the program ends as long as a schedule is found that does not violate any of the constraint, and never attempts to optimize the solution.

      In many cases, however, optimization is necessary. Although it obeys every rule in our list, a barely acceptable schedule may be frustrating when some residents have superior shift assignments, and others have uncomfortable ones that harms their own well-being, such as working 10 night shifts in a row.

      In the future we would like to add optimization options to the program and let the user decide which mode to use. The optimized factor could be various kinds of metrics that the chief resident deem as important but not enforced in the hard rules, such as bad sleep patterns, and tiresome consecutive working periods.

III. Acknowledgements

I would like to express my sincere gratitude to Professor Amy Cohn, my faculty advisor, for offering me the opportunity to work at Center for Healthcare Engineering and Patient Safety, and providing me with her patient guidance, strong technical support, and continuous confidence in my ability.

I would also like to offer my greatest appreciation to Mr. Tony Wang for his munificent financial support to my summer research. The generosity of Mr. Wang has been one of my greatest motivations to aim for the best I can do in the research. I also appreciate Mr. Mikhail Zolikoff for coordinating the funding issues and making the whole process smooth and successful.

My gratitude extends to Mr. Rodney Capps, Mr. Christopher Conrad, Mr. Gene Kim, and Ms. Elizabeth Fisher at Industrial and Operations Engineering Department. Their technical and administrative assistance made my 12-week summer research so much smoother than it could have been without their help.

I would like to appreciate Mr. Lonny Hurwitz, Mr. Keiron McEwen at Southwest Airlines for their strong support in aviation-meteorology statistical analysis, and Dr. Edmond O'Brien at C.S. Mott Children Hospital for his dedicated assistance in PEDS shift scheduling.

I also wish to acknowledge the help from the other student research assistant at CHEPS. My summer research has been a wonderful experience thanks to Jeremy Castaing, Ryan Chen, Aaron Cohen, Joanna Fleming, Mark Grum, Daniel Hazlett, Young-Chae Hong, Nathan Janes, Jared Kott, Brian Lemay, Elizabeth Olin, William Pozehl, Hannah Schapiro, Nina Scheinberg, George Tam, Zachary Verschure, and Yicong Zhang.

Lastly, I would like to give my special thanks to my friend Qijun Jiang, Kedi Wu, and Yang Yang, for their technical support in computer programming, data analysis, and Excel techniques, as well as for their wholehearted encouragement throughout the summer.

I could not have accomplished what I have so far in the summer without the help of these acknowledged people. Once again, I would express my deepest appreciation to every one of them for making my summer research a fruitful and valuable experience.

References

CHEPS. (2014). *The Engineer's Guide to Medical Resident Scheduling.* Ann Arbor.

IBM. (n.d.). *CPLEX Optimization Studio Interfaces*. Retrieved from IBM Software: http://www-01.ibm.com/software/commerce/optimization/interfaces/

IBM. (n.d.). *Linear programming*. Retrieved from IBM Software: http://www-01.ibm.com/software/commerce/optimization/linear-programming/

Morgan, S. S. (1997). *A Comparison of Simplex Method Algorithms.* Retrieved from http://www.mathtools.net/: http://www.cise.ufl.edu/research/sparse/Morgan/

NOAA. (2009). *Glossary*. Retrieved from National Oceanic and Atmospheric Administration: http://w1.weather.gov/glossary/index.php?letter=c