Automatic Physician Scheduler for the Emergency Department of TBRHSC

Sahand Asri
Department of Computer Science
Lakehead University
Thunder Bay, Canada
sasri@lakeheadu.ca

***Abstract*—A Shift Scheduling Problem consists of a list of physicians, days, and shifts, along with a set of constraints and objectives to be satisfied. The aim of this project is to assign working shifts to the physicians in the Emergency Department of Thunder Bay Regional Health Sciences Centre. We also believe that the offered solutions in this project could also be generalized and performed in other kinds of timetable scheduling systems. In this report, firstly, we explain timetable scheduling and several different approaches to deal with these types of problems. Secondly, we describe our particular problem in details. Then, we present our approach which is based on Constraint Programming and Integer Linear Programming. Finally, we analyze the final solution compared to the imaginary optimal solution.**

***Index Terms*—shift scheduling, timetable, constraints, objectives, constraint programming, linear integer programming, optimization**

I. Introduction

Timetable scheduling is the NP-hard problem of time and resource assignment which typically follows a set constraints (rules). Teacher assignment, classroom assignment and final exam planning for colleges and universities are some examples of timetable scheduling which needs to be processed each term.

In this paper, we take a look on another kind of timetable scheduling which is usually referred to Physician Scheduling Problem where shifts should be assigned to doctors based on the availability, expertise, fairness and many other factors.

In the past, all timetable scheduling were done manually by a single person or a group of people. Even now, there are many companies and hospitals using the old method for timetable scheduling. With the growth of technology and increase in the number of medical and human resources, shift scheduling has became a major issue for hospitals. However, the process of manually planning the timetable could be time-consuming, error-prone and complex which also requires lots of human interactions. Therefore, designing an automatic physician scheduler would make this process faster, error-free and more convenient.

This paper, represents a real case study of a Physician Scheduling project for Emergency Department of Thunder Bay Regional Health Sciences Centre (TBRHSC) in Ontario, Canada.

II. Background

There are several different types of approaches to deal with timetabling (scheduling) problems such as Mathematical Programming, Graph Coloring, Clustering, Constraint Programming, Meta-heuristic, Multicriteria, and Case-based reasoning methods [1]. In this project, we are using Integer Linear Programming (ILP) and Constraint Programming (CP) to build the scheduler.

A. Linear Programming (LP)

Linear Programming is a popular mathematical modeling method to obtain the best results considering the desired goals. Linear Programming Problem consists of a set of variables, coefficients, constraints and an objective function. Generally, a Linear programming problem could be expressed in Standard Form (Canonical Form) as shown in equation 1 [2].

$$\begin{aligned} &maximize \sum_{i=1}^{n} c_i x_i \\ &subject\ to \sum_{i=1}^{n} a_{ij}\ \ x_i \le b_j \quad j = 1,2,...m \\ &\qquad\qquad\qquad x_i \ge 0 \quad i = 1,2,...m \end{aligned} \tag{1}$$

B. Integer Linear Programming

Integer Linear Programming (ILP) is a subset of Linear Programming problem where the variables are restricted to be integers (Equation 2).

$$\begin{aligned} &maximize \sum_{i=1}^{n} c_i x_i \\ &subject\ to \sum_{i=1}^{n} a_{ij} x_i \le b_j \qquad j = 1,2,...m \\ &\qquad x_i \in \mathbb{Z}^n \qquad\qquad i = 1,2,...m \end{aligned} \tag{2}$$

If the variables only accept either 0 or 1 integer values, it is called Binary Integer Linear Programming (Equation 3). Mixed Integer Linear Programming (MILP) is another type of Linear Programming Problem which is a mixture of the mentioned types.

$$\begin{aligned} &maximize \sum_{i=1}^{n} c_i x_i \\ &subject\ to \sum_{i=1}^{n} a_{ij} x_i \leq b_j \qquad j = 1, 2, ...m \\ &\qquad x_i \in \{0, 1\} \qquad\qquad i = 1, 2, ...m \end{aligned} \tag{3}$$

C. Constraint Programming (CP)

Constraint Programming is a popular and recent technique to solve Constraint Satisfaction Problems such as task assignment and timetable scheduling [3]. Unlike Linear Programming which is based on linear algebra, Constraint Programming is mostly inspired by graph theory and logic in computer science. At first, constraint programming works to find a feasible solution for the model. Then, it will try to search for more answers to improve the value of the objective function. While processing, the model may realize there are no remaining values for the variables in domain to reach a feasible solution. In this situation, Constraint Programming Optimizer uses pruning approaches to reduce unnecessary further search. Constraint Programming Optimizer may also use forward checking techniques to detect inconsistency earlier. As a result, these features made Constraint Programming more valuable than simple Integer Linear Programming for timetable scheduling problems.

D. Constraint Satisfaction Problem (CSP)

Constraint Satisfaction Problem consists of a set of variables $X = \{x_1, x_2, ..., x_n\}$, a set of possible domains $D = \{d_1, d_2, ..., d_n\}$, and a set of constraints C [4]. Each constraint in C is a pair of Variable $V = \{v_1, v_2, ..., v_m\} \in X$ and relation $R \in d_1 \times d_2 \times ... \times d_m$.

Depending on the number of variables involving in the relations, a constraint could be Unary, Binary or in general k-ary. A Unary Constraint is a constraint which only involves one variable at a time. For instance, $x_1 = 0$ is a unary constraint which restricts a variable to only accept a specific value 0 in its domain. A Binary Constraint requires two variables in a single relation equation. For instance, the relation $R_1 : x_2 = x_3$ is a Binary Constraint which wants to make sure that the value given to x_2 is equal to the value of x_3. At last, a Constraint is called k-ary Constriant, if it has k variables in its relation. $x_5 = x_3 \times x_4$ is a 3-ary Constraint which is also known as a Ternary Constraint. All k-ary, are transformable to a set of Binary Constraints.

E. Boolean Satisfiability Problem (SAT)

Boolean Satisfiability problem is a type of CSP which accept binary values (True, False) to satisfy the given formula (relations). In a SAT problem, the formula should be in Conjunctive Normal Form (CNF). Conjunctive Normal Form which is also referred to Clausal Normal Form is a conjunction of one or more clauses where each clause is a disjunction of literals (Boolean variables) [5]. In other words, CNF is a Product of Sums.

F. Interfaces and solvers

There are many different kinds of algorithms and solvers available for Linear Programming, Integer Programming and Constraint Programming for both commercial and research use. Groubi and CPLEX are two examples for commercial solvers, while SCIP, GLPK and GLOP are open source and both have free license.

Also, there are several types of languages, frameworks and visual modeling systems to design and implement the Optimization Problems such as Linear Programming Problems and Constraint Satisfaction Problems. Gurobi is an example of language interfaces for optimization problems which also delivers its own solver as well. OR-Tools and CPLEX are another examples of these language modeling systems which are offered in external packages for C, C++. Java, Python.

A different type of frameworks use visual interfaces for the users to solve linear programming and Constraint Satisfaction Problems such as GAMS (General Algebraic Modeling System), Lingo and Visual Math.

Compared to language modeling systems, visual frameworks are easier and quicker for novice users. However, language modeling systems are often more flexible and give more options for professional users.

III. Problem Description

In this paper, we present an automatic Physician Scheduling System for Emergency Department of Thunder Bay Regional Health Sciences Centre (TBRHSC). This scheduler is based on Integer Programming and Constraint Programming using ILP and SAT solver. There are three type of constraints which are Hard Constraints, Soft Constraints and Objectives. These constraints consider important factors for shift scheduling of doctors such as availability, experience, expertise, and fairness to obtain a better shift assignment.

A. Hard Constraints

Hard Constraints are the rules which should not be broken by the scheduler at any cost. Table I is the list of hard constraints for this project.

B. Soft Constraints

Soft Constraints are not as strict as hard constraints. Although, it is preferred to avoid breaking these constraints. Breaking each soft constraint gives the scheduler negative point (punishment). An optimal solution is the one which does not violate any soft constraints. However, in practice while working with real-world problems, the optimal solution might be inaccessible or even impossible.

Soft Constraints may have different level of importance. In this project they are classified in four levels, which are Priority 1, Priority 2, Priority 3, Priority 4 Soft Constraints. Priority 1 Soft Constraints are the most important ones while Priority 4 Soft Constraints have the least importance. As a result, the punishment for soft constraints varies in value. Table II shows the list of soft constraints as well as their importance level (Priority).

TABLE I
LIST OF HARD CONSTRAINTS

1	Each doctor can work maximum 7 consecutive days
2	Each doctor can work maximum 1 shift per day
3	2 days off after last midnight shift (except on-call shift)
4	Certain physicians can only work on Fast-Track shifts
5	Maximum 2 midnight shifts in a row (except some physicians)
6	Physicians do not share the same shift (in the same day)
7	No midnight shifts for the new staff (first 6 months)
8	No late or midnight shifts prior to the day requested off
9	2 days off after 3 to 7 consecutive working days
10	All shifts should be assigned
11	Physicians can work the 9:30 AM shifts or earlier prior to working on call. They can work starting no earlier than 11 AM the day after on call

TABLE II
LIST OF SOFT CONSTRAINTS

#	Priority	Description
0	1	Work maximum 7 days in a row
1	2	No shifts that starts more than 1.5 hours earlier than the shift on the previous day
2	2	Max 1 midnight in a row
3	2	Maximum 3 late shifts in a row
4	2	Work maximum 5 days in a row
5	2	Maximum 5 late shifts in two weeks
6	3	General principle avoid shift times changing too much day to day
7	3	3 days off after midnight
8	3	3 days off when transitioning from late shift to day shift
9	3	2 days off when transitioning from late shift to afternoon shift
10	4	Shifts should have same start time to 2.5 hours later compared to previous shift
11	4	Avoid FT shifts on consecutive days

C. Objectives

Objectives are the states which are desired for the scheduler to reach as close as possible. Objectives and Soft Constraints are similar when it comes to the scheduler. Objectives have 4 level of priorities as well where Priority 1 has the most importance. Same as Soft Constraints, if the objective is not met, the solver receives negative points. The value of the punishment depends on the priority of the objective and also the distance of the offered solution by scheduler compared to the optimal solution. Again, the optimal solution might be infeasible. However, the scheduler is expected to search for the closest possible solution. Table III shows the list of objectives with its corresponding priority.

TABLE III
LIST OF OBJECTIVES

#	Priority	Description
1	1	Equalize weekends (equal number per person per FTE)
2	1	Equalize holidays worked
3	1	Equalize night shifts
4	1	Equalize late shifts
5	2	Minimize split weekends
6	2	Avoid scheduling people shifts that end after 5 PM Friday on weekends they have off
7	2	Equalize 2200 shifts
8	2	Equalize 2000 shifts
9	3	Equalize day shifts
10	3	Equalize afternoon shifts
11	3	Avoid working more than 4 days in a row
12	4	Equalize weekdays
13	4	Equalize other shifts
14	4	Avoid working more than 3 days in row

IV. MATHEMATICAL MODELING

In this section, we show and explain the mathematical modeling techniques and equations which is used for each constraint. Some of the following equations are conditioned with Linear Integer Programming modeling system such as Equation 7, 8, 9, 10, and 11. The rest of the equations are written in a Constraint Logic Programming modeling system such as Equation 6. A Constraint Programming solver like SAT Solver might be able to solve both of these equation types.

In all of the equations, $X(n, d, s)$ refers to the assignment state of the doctor n, in the day d, for the shift s. For instance, if $X(2, 3, 4) = 1$ it means doctor 2 is assigned to work in day 3 for shift 4. Although, if $X(4, 3, 2) = 0$ it means doctor 4 is not going to work for shift 2 in day 3. Also, $work(n, d)$ means a doctor n is assigned to work on day d. In other words, $work(n, d) = \sum_{s \in shifts} X(n, d, s)$. For more details and the ease of reference, a list of notations and symbols is offered in Table IV and Table V.

A. Hard Constraints

As mentioned in the previous section, we should strictly avoid breaking any hard constraints. Here are the list of hard constraints along with their mathematical modeling.

Hard Constraint 1: Each doctor can work maximum 7 consecutive days

$$\sum_{s \in shifts} \sum_{d_f=0}^{7} X(N, D + d_f, s) \leq 7 \qquad \forall N \in docs, \forall D \in days \tag{4}$$

TABLE IV
LIST OF NOTATIONS

$docs$	All doctors
$days$	All days
$shifts$	All shifts
N_{new}	Doctors with less than 6-month work experience
N_{ft}	Fast-track doctor
D^i_{off}	Day requested off for doctor 'i'
$D_{workdays}$	Weekdays: Monday to Friday (working days)
$D_{weekends}$	Weekends: Saturday and Sunday
$D_{holidays}$	Holidays
$D_{Saturday}$	Saturdays
S_{oncall}	On-call shifts
S_{FT}	Fast-track shift
S_{day}	Day shifts
S_{noon}	Afternoon shifts
S_{night}	Night shifts
S_{late}	Late shifts
S_{mnight}	Midnight shifts
$S^{time:t}$	Shifts starting at time 't'
$time(s)$	Starting time of shift 's'

TABLE V
LIST OF VARIABLES

$X(n, d, s)$	Doctor n is assigned to work on day d on shift s
$work(n, d)$	Doctor n is assigned to work on day d $= \sum_{s \in shifts} X(n, d, s)$

Explanation: The number of assigned shifts for current day plus the next 7 days $(D + d_f)$ for each doctor should be 7 or less (current day could be any day other than last 7 days).

Hard Constraint 2: Each Doctor can work maximum 1 shift per day

$$\sum_{s \in shifts} X(N, D, s) \leq 1 \qquad \forall N \in docs, \forall D \in days \tag{5}$$

Explanation: Everyday, the number (summation) of assigned shifts for every doctor should be equal or less than 1.

Hard Constraint 3: 2 days off after last midnight shift (except on-call shift)

$$\begin{aligned} &\neg X(N, D, S_{mnight}) \vee \\ &X(N, D+1, S_{mnight}) \vee \\ \neg((&\sum_{s \in shifts} X(N, D+1, s) + X(N, D+2, s)) == 0) \\ &\forall N \in docs, \forall D \in days \end{aligned} \tag{6}$$

Explanation: Everyday, for each doctor (row 4), either they are not working on midnight (row 1) or they are working midnight tomorrow (it is not the last midnight) (row 2) or there are 2 days off after the last midnight (row 3).

Hard Constraint 4: Certain physicians can only work on Fast-Track shifts

$$X(N, D, S) = 0 \qquad \forall N \in N_{ft}, \forall D \in days\ \forall S \in shifts \setminus S_{FT} \tag{7}$$

Hard Constraint 5: Maximum 2 midnight shifts in a row (except some physicians)

$$\sum_{d=0}^{2} X(N, D+d, S) \leq 2 \qquad \forall N \in docs, \forall D \in days, \forall S \in shifts \tag{8}$$

Hard Constraint 6: Physicians do not share the same shift (in the same day)

$$\sum_{n \in docs} X(n, D, S) \leq 1 \qquad \forall D \in days, \forall S \in shifts \tag{9}$$

Explanation: The number (count) of the physicians who are working in the same day and same shift should be 1 or less (0).

Hard Constraint 7: No midnight shifts for the new staff (first 6 months)

$$X(N, D, S) = 0 \qquad \forall N \in N_{new}, \forall D \in days, \forall s \in S_{mnight} \tag{10}$$

Hard Constraint 8: No late or midnight shifts prior to the day requested off

$$X(N, D-1, S) = 0 \qquad \forall N \in N_{ft}, \forall D \in D^N_{off}, \forall S \in S_{late} \tag{11}$$

Hard Constraint 9: 2 days off after 3 to 7 consecutive working days
Equation 12 shows the mathematical model for 2 days off after 3 consecutive working days. 3 could be generalized to any number of days.

$$\begin{aligned} &\sum_{s \in shifts} \sum_{d_f=0}^{2} X(N, D+d_f, s) == 0 \quad \vee \\ &\sum_{s \in shifts} \sum_{d_f=0}^{1} X(N, D+3+d_f, s) == 0 \\ &\forall N \in docs, \forall D \in days \end{aligned} \tag{12}$$

Explanation: The physicians are not working three consecutive days (first three days), OR they are having 2 days off after it (not working in the last two days).

Hard Constraint 10: All shifts should be assigned

$$\sum_{n \in docs} X(n, D, S) = 1 \qquad \forall D \in days, \forall S \in shifts \tag{13}$$

Explanation: Everyday, for each shift, there should be exactly one physician assigned for it.

Hard Constraint 11: Physicians can work the 9:30 AM shifts or earlier prior to working on call.

$$\begin{aligned} X(N, D, S_{oncall}) + \sum_{s \in S_1} X(N, D-1, s) \leq 2 \\ \forall N \in docs, \forall D \in days, \\ \forall S_1 \in \{s \in shifts | time(s) > 930\} \end{aligned} \tag{14}$$

Physicians can work starting no earlier than 11 AM the day after on call.

$$\begin{aligned} X(N, D, S_{oncall}) + \sum_{s \in S_2} X(N, D+1, s) \leq 2 \\ \forall N docs, \forall D \in days, \\ \forall S_2 \in \{s \in shifts | time(s) < 1100\} \end{aligned} \tag{15}$$

B. Soft Constraints

Soft Constraint are a set of conditions which are not required but preferred to be satisfied. In this project, breaking each soft constraint gives a penalty to the optimization solver. Therefore, the solver should try to minimize the cost function of the soft constraints. In this section, we look at the penalty function (cost function) of each individual soft constraint.

Soft Constraint 0: Work maximum 7 days in a row.

First we need to find out if a physician is assigned to work in a specific day. For that, we can get the summation of assigned shifts for that particular physician and day. In Equation 16, if the doctor N is assigned to work on day D, the output would be 1 (it cannot be more than 1 since each physician may work only one shift per day), otherwise it would be 0.

$$\sum_{s \in shifts} X(N, D, s) \tag{16}$$

Now, it is desired to check forward for the next 7 days and find out if doctor N is working 8 consecutive day. We can do this by getting the product (multiplication) of Equation 16 for 8 consecutive day starting from day D. The output of Equation 17 equals to 1 if doctor N is working 8 days in row, which is not ideal for the solver. Otherwise, the output would be 0.

$$\prod_{d_f=0}^{7} \sum_{s \in shifts} X(N, D + d_f, s) \tag{17}$$

We would like the solver to check the Equation 17 for every doctor, starting from any day. Therefore, the solver should get the summation of Equation 17 and try to minimize it. Equation 18 shows the final cost function for soft constraint 0. Ideally, the output of Equation 18 would be 0, if there are no consecutive working days for any doctor. But, for every consecutive working days found for each doctor, the solver would receive (+1) as punishment.

$$minimize \sum_{n \in docs} \sum_{d \in days} \prod_{d_f=0}^{7} \sum_{s \in shifts} X(n, d + d_f, s) \tag{18}$$

Soft Constraint 1: No shifts that starts more than 1.5 hours earlier than the shift on the previous day

In the first step, Equation 19 gives a set of shifts where the second shift is more than 90min earlier than the previous shift.

$$\forall (s_0, s_1) \in \{s_0, s_1 | time(s_0) - time(s_1) > 90min\} \tag{19}$$

The product of $X(N, D-1, s_0)$ and $X(N, D, s_1)$ would be true if doctor N is working 90min earlier the previous day. Therefore, the Equation 20 gets the summation of the probable violations of Soft Constraint 1 (everyday for every doctor) and tries to minimize it.

$$\begin{aligned} minimize \sum_{n \in docs} \sum_{d \in days} X(n, d, s_0) \times X(n, d+1, s_1) \\ \forall (s_0, s_1) \in \{s_0, s_1 | time(s_0) - time(s_1) > 90min\} \end{aligned} \tag{20}$$

Soft Constraint 2: Max 1 midnight in a row

The Equation 21 get the number of 2 consecutive midnight shifts assigned to a every doctor.

$$\begin{aligned} minimize \sum_{n \in docs} \sum_{d \in days} X(n, d, S_{mn}) X(n, d+1, S_{mn}) \\ \forall S_{mn} \in S_{mnight} \end{aligned} \tag{21}$$

Soft Constraint 3: Maximum 3 late shifts in a row

$$\begin{aligned} minimize \sum_{n \in docs} \sum_{d \in days} X(n, d, S_1) X(n, d+1, S_2) \\ \forall S_1 \in S_{late}, \ \forall S_2 \in S_{late} \end{aligned} \tag{22}$$

Soft Constraint 4: Work maximum 5 days in a row.

Soft Constraint 4 is very similar to the Soft Constraint 0 with a difference in the number of consecutive days which is 5 instead of 8 in Soft Constraint 4. Equation 23 shows the cost function of Soft Constraint 4.

$$minimize \sum_{n \in docs} \sum_{d \in days} \prod_{d_f=0}^{5} \sum_{s \in shifts} X(n, d+d_f, s) \tag{23}$$

Soft Constraint 5: Maximum 5 late shifts in two weeks.

The Equation 24 gets the number of late shifts a physician is working in two weeks. Then, it checks if the number exceeds the threshold which is 5. In the end, it counts the number of violations (late shifts exceed the threshold) with every doctors and every starting day.

$$min \sum_{n \in docs} \sum_{d \in days} (\sum_{s_n \in S_night} \sum_{d_f=0}^{14} X(n, d+d_f, s_n)) \geq 5 \tag{24}$$

Equation 24 cannot be implemented using Linear Integer Programming. However, it can be implemented using Constraint Programming techniques. In the Implementation section, we can see some detail about a specific Constraint Programming modeling system and methods to deal with these kinds of problems.

Soft Constraint 6: General principle avoid shift times.

This Soft Constraint is same as Soft Constraint 1 and Soft Constraint and Soft Constraint 10. In other words, this it is the mixture of Soft Constraint 1 and 10. Therefore, we avoid repetition here.

Soft Constraint 7: 3 days off after midnight.

Equation 26 is similar to Equation 24. It can be implemented using Constraint Programming methods. The output of Equation 25 would be 1 only if doctor N is assigned to the midnight shift on day D **and** doctor D is assigned to work for any shift in the next 3 days.

$$X(N, D, S_{mnight}) \wedge (\sum_{d_f=1}^{3} \sum_{s \in shifts} X(N, D+d_f, s) > 0) \tag{25}$$

Equation 26 gets the summation of the violations for each doctor and day. Then, it tries to minimize the penalty function (number of violations).

$$\begin{aligned} minimize(\sum_{n \in docs} \sum_{d \in days} X(n, d, S_{mnight}) \quad \wedge \\ (\sum_{d_f=1}^{3} \sum_{s \in shifts} X(n, d+d_f, s) > 0)) \end{aligned} \tag{26}$$

A detailed Implementation for similar Soft Constraints is explained in Subsection V-H (OnlyEnforceIf) of Section V (Implementation Details).

Soft Constraint 8: 3 days off when transitioning from late shift to day shift.

For Soft Constraint 8, we use two different techniques. Then, we implement each one to see to pick the one with the best result (less number of violations), while considering the time it takes for the scheduler to solve the problem.

$$\begin{aligned} min \sum_{n \in docs} \sum_{d \in days} X(n, d, S_n) \times X(n, d+1, S_d) \times \\ \sum_{d_f=0}^{2} \sum_{s \in shifts} X(n, d+2+d_f, s) \\ \forall S_n \in S_{late}, \forall S_d \in S_{day} \end{aligned} \tag{27}$$

Equation 27, uses the Linear Integer Programming concepts to calculate the cost function. The cost function will receive $+1$ for any violation of Soft Constraint 8. In other words, if in the first day a doctor is assigned to a late shift **and** in the second day the same doctor is working day shift **and** he is working in any shift in the next 3 days, then the scheduler would get $+1$ as punishment.

In the second method, we may use Constraint Logic Programming techniques.

$$\begin{aligned} minimize \sum_{n \in docs} \sum_{d \in days} transition(n, d), \\ (transition(N, D) \vee \\ \neg X(N, D, S_{late}) \vee \\ \neg X(N, D, S_{day})) \\ transition(n, d) \in \{0, 1\} \\ \forall N \in docs, \forall D \in days \end{aligned} \tag{28}$$

In Equation 28, either transition(n,d) is true (happened) **or** a doctor n is not working in night shift in the first day **or** the same doctor is not working in day shift in the second day. Since, we solver is trying to minimize the transition (min for transition is 0), therefore, one of the next two conditions should met. In other words, by satisfying any of these conditions (not working night shift in the first day or not working day shift in the next day), we are avoiding the transition, therefore, transition variable could accept value 0.

Soft Constraint 9: 2 days off when transitioning from late shift to afternoon shift

Soft Constraint 9 is same as previous one, where only the transitioning shifts are different. Therefore, we avoid repeating the same concept and technique for this Soft Constraint.

Soft Constraint 10: Shifts should have same start time to 2.5 hours later compared to previous shift.

This Soft Constraint is same as Soft Constraint 1. The only difference is having 2.5 hours instead of 1.5 hours, and it is considering the later shifts instead of earlier shifts. Equation 29 is the penalty function for this Soft Constraint.

$$\begin{gathered} minimize \sum_{n \in docs} \sum_{d \in days} X(n, d, s_0) \times X(n, d+1, s_1) \\ \forall (s_0, s_1) \in \{s_0, s_1 \mid time(s_1) - time(s_0) > 150min\} \end{gathered} \tag{29}$$

Soft Constraint 11: Avoid FT shifts on consecutive days.

The Equation 30 finds out if each doctor is avoid any combination of Fast-Track shifts in any two consecutive days. Then, it counts the number of violations and returns it as penalty function and the solver should try to minimize the final value of the penalty function.

$$min \sum_{n \in docs} \sum_{d \in days} \sum_{s_1, s_2 \in S_{FT}} X(n, d, s_1) \times X(n, d+1, s_2) \tag{30}$$

C. Objectives

Objectives are similar to Soft Constraints. They are not required to fully satisfied by the scheduler. However, it is recommended to avoid any unnecessary violations in the objectives. Here we take a closer look to the objectives and some offered penalty functions (cost functions) for the violation of each individual objective.

Objective 1: Equalize weekends (equal number per person-per FTE)

Method 1: First, we calculate the number of days, each doctor is working on weekends. Second, we calculate the minimum and maximum number of days a doctor is assigned to work in weekends. The solver should try to minimize the difference between the minimum and maximum number of shift assignments to reach a balanced and equalized weekend shift assignment. For instance, in a timetable for 3 doctors where they are working 1, 2, and 4 weekends, the cost function would be 3 which is the difference between min and max weekend shift assignment. Equation 31 shows the penalty function for Objective 1.

$$\begin{aligned} & minimize \quad (D_{max} - D_{min}) \\ & D_{max} = \max_{N_1 \in docs} \sum_{d_w \in D_{weekends}} \sum_{s \in shifts} X(N_1, d_w, s) \\ & D_{min} = \min_{N_2 \in docs} \sum_{d_w \in D_{weekends}} \sum_{s \in shifts} X(N_2, d_w, s) \end{aligned} \tag{31}$$

Method 2: First, we calculate the number of days where each doctor is working on weekends. Second, we calculate the ideal weekend number of days for doctors to work on weekends by dividing the number of weekend shifts by the number of doctors (Equation 32). Note that the ideal weekend shift distribution may be infeasible.

$$D_{ideal} = \lfloor num(D_weekends) * num(shifts) / num(doctors) \rfloor \tag{32}$$

Then, we calculate the difference between the assigned weekend shifts and the ideal solution for every doctor and then we add the all. Finally, the solver should try to minimize the calculated values. Equation 33 shows the final objective function using the second method for Objective 1.

$$\begin{aligned} & min \quad (\sum_{n \in docs} \sum_{d_w \in D_{weekends}} \sum_{s \in shifts} X(n, d_w, s) - D_{ideal}), \\ & D_{ideal} = \lfloor num(D_{weekends}) * num(shifts) / num(doctors) \rfloor \end{aligned} \tag{33}$$

Method 2 may work better in the situations where a set of doctors are restricted to work on weekends but the scheduler could still try to equalize weekend shifts for the rest of the physicians.

Objective 2: The same methods on Objective 1 can be applied on this one as well. Therefore, we avoid repetition of the idea.

Method 1:

$$\begin{aligned} & minimize \quad (D_{max} - D_{min}) \\ & D_{max} = \max_{N_1 \in docs} \sum_{d_w \in D_{holidays}} \sum_{s \in shifts} X(N_1, d_w, s) \\ & D_{min} = \min_{N_2 \in docs} \sum_{d_w \in D_{holidays}} \sum_{s \in shifts} X(N_2, d_w, s) \end{aligned} \tag{34}$$

Method 2:

$$\begin{aligned} & min \quad (\sum_{n \in docs} \sum_{d_w \in D_{holidays}} \sum_{s \in shifts} X(n, d_w, s) - D_{ideal}), \\ & D_{ideal} = \lfloor num(D_{holidays}) \times num(shifts) / num(doctors) \rfloor \end{aligned} \tag{35}$$

Equalize Objectives: For Equalize Objectives which are Objective 1, 2, 3, 4, 7, 8, 9, 10, 12, 13 which are related to the fair shift assignment, the same method and instructions as Objective 1 and Objective 2 can be applied to them. Therefore, we do not repeat explaining the details again.

Objective 5: Minimize split weekends.

The Equation 36 tries to minimize the number of times a doctor works only on Saturday or Sunday on the same weekend.

$$min \sum_{n \in docs} \sum_{d_s \in D_{Saturday}} work(n, d_s) \neq work(n, d_s + 1) \tag{36}$$

Equation 36 could be implemented using Constraint Logic Programming techniques. In the Section V we will look into more details.

Objective 6: Avoid scheduling people shifts that end after 5 PM Friday on weekends they have off

$$min \sum_{n \in docs} \sum_{d_s \in D_{Saturday}} (work(n, d_s) \vee work(n, d_s + 1)) \wedge \sum_{s_l \in S^{t>5PM}} X(n, d_s - 1, s_l) \tag{37}$$

In Equation 37 whether doctor n is off on the weekend **or** they are not working after 5 PM on Friday evening. If we had a violation of the above statement, it comes with a $+1$ penalty for the scheduler. Since the scheduler is minimizing the penalty function, therefore, it avoids breaking the statement if possible.

Objective 11: Avoid working more than 4 days in a row.

For Objective 11, Equation 38 calculates the number of times a doctor is working for 5 consecutive days. Then it gives $+1$ penalty to each time the mentioned statement is broken by any doctor in any starting day.

$$min \sum_{n \in docs} \sum_{d \in days} \prod_{d_f=0}^{4} work(n, d + d_f) \tag{38}$$

For instance, if a doctor is assigned to work for 6 consecutive days, the scheduler receives $+2$ penalty, since there are two 5 consecutive working days. The first one, starts with the first day, and the second one starts from the next day after.

Objective 14: Avoid working more than 3 days in row

It is similar to Objective 11. The only difference is the scheduler should avoid working more than 3 days in row instead of 4. Equation 39 shows the cost function for Objective 14.

$$min \sum_{n \in docs} \sum_{d \in days} \prod_{d_f=0}^{3} work(n, d + d_f) \tag{39}$$

V. Implementation Details

A. OR-Tools

In this project, we use Google OR-Tools Python library which is an interface for modeling optimization problems. OR-Tools is an open source software which is licensed under the terms of the Apache License 2.0. OR-Tools is available Python, C++, C# and Java.

After modeling the problem using mathematical techniques in OR-Tools, we need to search through the variable state space to find the feasible solutions and then pick the best which maximize or minimize the objective function. However, in most of the cases, there are too many variables in the states space to explore even for modern computers. As a result, it is important how to compress the state space to search space and explore search state in a better order. This tasks are the responsibilities of an Optimization Solver. OR-Tools offers four different types of optimization solvers depending on the application, which are: Linear and Mixed-Integer Programming, Constraint Programming Solver, Vehicle Routing, and Graph Algorithms. In this project, we are using Linear Integer Programming and Constraint Programming techniques, therefore we need a solver which support both of them.

1) Linear and Mixed-Integer Programming: MPSolver is the main OR-Tools solver for Linear and Mixed-Integer Programming Problems. After modeling the MIP problem, it is time to pick a solver to for exploring the state space. Modeling the MIP problem gives us the chance to choose from a wide variety of well-known linear solvers, such as Gurobi, CPLEX, SCIP, GLPK, and GLOP.

2) Constraint Programming: OR-Tools also offers CP-SAT solver to deal with Constraint Programming Problems.

For solving an Integer Programming problem we may use both MIP Solver and CP-SAT. while dealing with Integer Programmin problems, MIP Solver might be faster than CP-SAT. However, CP-SAT solver offers more features and flexibility to the model.

In this project, we use CP-SAT solver for the physician scheduling.

B. Hard and Soft Constraints

Mainly there are two types of constraints, Hard Constraints and Soft Constraints. While hard constraint should be avoided at any cost, soft constraints there is no such restriction for soft constraints. However, they should be avoided whenever possible. In this project, Objectives are implemented and treated like Soft Constraints.

C. Unary, Binary, and k-ary Constraints

In Constraint Programming modeling system, each constraint consists of Relations and Variables. Based on the number of different Variables in the Relations, there are three types of Constraints, Unary Constraints, Binary Constraints, and k-ary Constraints.

1) Unary Constraints: Unary Constraints are the ones with only one variable in the relation. Therefore, they are easy to control and usually can be handled by restricting the domain of the variables. For instance, in a Unary Constraint with Relation $x \neq 0$, we only need to remove 0 from the variable x domain.

2) Binary Constraints: Binary Constraint involves two different Variable in the Relation. While in some Binary Constraints cases we may use MIP modeling system for implementation, in others we should use other Programming techniques such as Constraint Logic Programming to solve it. For instance, for $x_1 = x_2$, we could easily convert it to $x_1 - x_2 = 0$ to implement it in MIP. However, for $x_1 \times x_2 = 1$, we may need Constraint Programming to solve it.

3) k-ary Constraints: k-ary Constraints are the ones with k Variables in the relations. Similar to Binary Constraints, a $k \geq 2$ k-ary Constraint, might be convertible to a Linear Integer Programming problem or not. For instance, in $x_1 = x_2 + x_3$, we can convert it to $x_1 - x_2 - x_3 = 0$. while, we cannot do it for $x_3 = x_1 \times x_2$. k-ary Constraints such as latter one are harder to implement and usually need more time to be solved.

D. CreateSolver

In OR-Tools in order to create an MIP solvers, we use CreateSolver. In the argument of this method, we could specify which type of linear solver do we want to choose. In Listing 1, we create an SCIP Solver.

```
from ortools.linear_solver import pywraplp

# Create the mip solver with the SCIP backend.
solver = pywraplp.Solver.CreateSolver('SCIP')
```

Listing 1. Creates an SCIP MIP Solver

In OR-Tools in order to create a Google CP-SAT solver, we may use cp_model. In Listing 2, we create a CP-SAT Solver.

```
model = cp_model.CpModel()
```

Listing 2. Create a CP-SAT Solver

E. New Variable

To declare a new Boolean variable in OR-Tools, we use NewBoolVar and for declaring a new Integer variable we use NewIntVar. Note that for declaring a new Integer variable, we also need to give it an upper and lower boundary. Listing 3 is an example of creating a set of Boolean variables (the set of variables X in Table V) that we declare in this project.

```
doc_shifts = {}
for n in range(num_doctors):
    for d in range(num_days):
        for s in range(num_shifts):
            doc_shifts[(n,d,s)] = model.NewBoolVar("
                shift_n%i_d%i_s%i" % (n,d,s))
```

Listing 3. Create doc_shift variables (X in Table V)

Listing 4 is another example for declaration of another set of variables (work in Table V).

```
doc_shifts = {}
for n in range(num_doctors):
    for d in range(num_days):
        for s in range(num_shifts):
            doc_shifts[(n,d,s)] = model.NewBoolVar("
                shift_n%i_d%i_s%i" % (n,d,s))
```

Listing 4. Create is_working variables (work in Table V)

F. Add New Constraint

model.Add is a popular method in OR-Tools which is used for introducing a restriction (constraint) which has to be satisfied without considering any extra costs. Therefore, this method is very useful for modeling most of the Hard Constraints. Listing 5 shows an example where we used model.Add method to build Hard Constraint 2.

```
for n in range(num_doctors):
    for d in range(num_days):
        model.Add(sum(doc_shifts[(n,d,s)] for s in
            range(num_shifts)) <= 1)
```

Listing 5. Using model.Add method for Hard Constraint 2

G. Add Multiplication Constraints

AddMultiplicationEquality is another method in OR-Tools which we use for getting the multiplication of a set of variables. The final output of the multiplication goes to another IntVar which is needed to be declared in advance. The AddMultiplicationEquality method only search the domain of the output variable to find a solution which satisfies the multiplication. If the the value cannot be found in the output domain, the solver cannot find any feasible solution. This method could be useful for hard constraints. In Listing 6, we use AddMultiplicationEquality to solve Hard Constraint 12.

```
for n in range(num_doctors):
    for d in range(num_days-5):
        days_1_3 = model.NewBoolVar("")
        days_4_5 = model.NewBoolVar("")

        days = [is_working[(n,d+d_f)]
            for d_f in range(5)]

        model.AddMultiplicationEquality(days_1_3,
            [days[i] for i in range(3)])
        model.AddMultiplicationEquality(days_4_5,
            [days[i+3] for i in range(2)])
        model.AddBoolOr([days_1_3.Not(),
            days_4_5.Not()])
```

Listing 6. Using AddMultiplicationEquality method for Hard Constraint 12

In addtion to the hard constraints, this method is used to build many intermediate variables in soft constraints. In many cases in Binary and k-ary Constraints $(k > 3)$ where the relations are non-linear, we create these intermediate variables to be used in the penalty function. In Listing 7, we use AddMultiplicationEquality to declare and initialize some intermediate variables to develop Soft Constraint 0.

```
soft[0] = 0

for n in range(num_doctors):
    for d in range(num_days - 7 + 1):
        is_working_var_list = [is_working[(n,d+d_f)]
     for d_f in range(7)]
        is_working_days_1_7 = model.NewBoolVar("")

        model.AddMultiplicationEquality(
            is_working_days_1_7,is_working_var_list)
        soft[0] += is_working_days_1_7
```

Listing 7. Using AddMultiplicationEquality method for Soft Constraint 0

Timing Issue: Since this method is working with Integer Variables and uses Integer Multiplication, it requires more complicated hardware architecture compare to main logical operators. Usually, these hardware components are not available in a large scale. Therefore, this method is much slower than model.Add method and other logical operators in OR-Tools. We found this issue while we were declaring new intermediate variables using AddMultiplicationEquality for soft rules. Before dealing with this issue, the scheduler was not able to find any feasible solution in hours of work. This issue is solved by implementing logical AND (Bitwise AND) operator and AddBoolOr method, which is explained in the next following sections.

H. Conditional Constraints

Conditional Constraints are the ones which should be satisfied only if some condition met. In OR-Tools "OnlyEnforceIf" method does this task for the CP-SAT Solver. Usually "OnlyEnforceIf" goes after model.Add function to do this trick.

```
soft[7] = 0

for n in range(num_doctors):
    for d in range(num_days-14+1):
        is_soft7_broken = model.NewBoolVar("")
        num_late_shifts_in_two_weeks =
            sum(doc_shifts[(n,d+d_f,s_late)] for
            d_f in range(14) for
            s_late in late_shifts)

        model.Add(num_late_shifts_in_two_weeks > 5)
            .OnlyEnforceIf(is_soft7_broken)
        model.Add(num_late_shifts_in_two_weeks <= 5)
            .OnlyEnforceIf(is_soft7_broken.Not())

        soft[7] += is_soft7_broken
```

Listing 8. Using OnlyEnforceIf method for Soft Constraint 7

For example, in Soft Rule 7, we declare a new Boolean variable is_soft7_broken for Soft Rule 7. Then, we calculate the total number of shifts for the next two weeks. Then, we use "OnlyEnforceIf" function to check the Soft Rule 7 violation. If, is_soft7_broken is True, then, the "OnlyEnforceIf" function forces the scheduler to have more than 5 shifts in two weeks. Otherwise, if the variable is False, the scheduler should find a solution where there are 5 or less shifts in the next two weeks. Finally, the summation of these variables will be our penalty function. Since, the scheduler is trying to minimize the penalty function, therefore, it tries to avoid having $is\_soft7\_broken = True$ condition in each case. Listing 8 shows our detailed implementation using "OnlyEnforceIf" method for Soft Rule 7.

I. Add Boolean OR Constraints

"AddBoolOr" is a method for CP-SAT solver which forces the solver to satisfy at least one of the given arguments (literals) in the list. Listing 9 shows the implementation of $a \lor b$.

```
a = model.NewBoolVar("")
b = model.NewBoolVar("")

model.AddBoolOr([a,b])
```

Listing 9. Boolean OR Constraint

J. Add Boolean AND Constraints

"AddBoolAnd" is another method in CP-SAT solver which forces the solver to satisfy all of the given arguments (literals) in the list. Listing 10 shows the implementation of $a \land b$.

```
a = model.NewBoolVar("")
b = model.NewBoolVar("")

model.AddBoolAnd([a,b])
```

Listing 10. Boolean AND Constraint

K. Bitwise AND

As mentioned before, in Add Multiplication Constraints, there is a major timing issue when we are dealing with a huge number of variables and multiplications. To solve this issue, we implement our own Bitwise AND which is much faster than integer multiplication. There are two methods to implement, Bitwise AND. Let us assume that we are going to implement $c = a \land b$.

First Method: In the first method, we use "AddBoolAnd", "AddBoolOR" and "OnlyEnforceIf" methods. This method is only recommended for OR-Tools v8.0 (2020-10) and above, since there was an issue in "AddBoolAnd" in previous versions. Listing 11 shows detailed implementation for this method.

```
a = model.NewBoolVar("")
b = model.NewBoolVar("")
c = model.NewBoolVar("")

model.AddBoolAnd([a,b]).EnforceOnlyIf(c)
model.AddBoolOr([a.Not(),b.Not()]).
                            EnforceOnlyIf(c.Not())
```

Listing 11. Implementing Bitwise AND operator using first method

Second Method: In the second method, we only use "OnlyEnforceIf" and implement "Bitwise AND" in very basic detail. "OnlyEnforceIf" method acts like implication in logical terms. Therefore, first, we need to build logical AND operator using logical Implication (material implication) and Negation (Not). Equation 40 shows how we can build an "AND" operator using only implication and negation operators.

$$\begin{aligned} &((b \to a) \to c) && \land \\ &((b \to \neg a) \to \neg c) && \land \\ &((\neg b \to a) \to \neg c) && \land \\ &((\neg b \to \neg a) \to \neg c) && \land \end{aligned} \tag{40}$$

```
a = model.NewBoolVar("")
b = model.NewBoolVar("")
c = model.NewBoolVar("")

model.Add(c == 1).OnlyEnforceIf(a).OnlyEnforceIf(b)
model.Add(c == 0).OnlyEnforceIf(a.Not()).
                            OnlyEnforceIf(b)
model.Add(c == 0).OnlyEnforceIf(a).
                            OnlyEnforceIf(b.Not())
model.Add(c == 0).OnlyEnforceIf(a.Not()).
                            OnlyEnforceIf(b.Not())
```

Listing 12. Implementing Bitwise AND operator using second method

Finally, we can replace Implication operators with ”OnlyEnforceIf” method and use Not function for Negation operators. Listing 12 shows the full implementation of ”AND” opperation using ”OnlyEnforceIf” and ”Not” methods.

L. *Get Minimum and Maximum*

”AddMinEquality” and ”AddMaxEquality” the methods for finding the Min and Max values in a list of variables. Listing 13 explains the implementation of Objective 3 (first method) using both ”AddMinEquality” and ”AddMaxEquality” methods. For the mathematical modeling details refer to Equation 34 which uses the same technique.

```
# find the max working days on holidays
max_sum_holiday_worked = model.NewIntVar(0, num_days
    , "" % n)
model.AddMaxEquality(max_sum_holiday_worked, [
    sum_doc_holiday_worked[n] for n in range(
    num_doctors)])

# find the min working days on holidays
min_sum_holiday_worked = model.NewIntVar(0, num_days
    , "" % n)
model.AddMinEquality(min_sum_holiday_worked, [
    sum_doc_holiday_worked[n] for n in range(
    num_doctors)])

# minimize the difference of the max and min
obj[3] = (max_sum_holiday_worked -
    min_sum_holiday_worked)
```

Listing 13. Implementing Min and Max Equality for Objective 3

In this example, both maximum number and minimum number of working days can accept any number between 0 and $num(days)$.

M. *Optimize*

After implementing individual objective functions (cost functions), we need to aggregate them into a single one. For that, we can simply get the summation of all the objective functions. However, there are two problem. Firstly, each individual objective function is using a different method and also has a different scale compared to the rest of the objective functions. Secondly, each soft constraint and objective has a different level of importance (priority) in our scheduling problem.

To solve the first problem, we need to put all the objective functions in a same scale. In this project, we made the lower boundary of all soft constraints and objectives to be the same which is 0. In addition, since we are counting the number of violations in every cost function, they are having the same scale.

For the second issue, we need to prioritize some objective functions over the other one. Therefore, we can give a specific coefficient to each individual objective function. The value of the coefficient for the constraint with a higher priority should be more than a lower priority constraint.

N. *Coefficients*

Finding the perfect coefficients for most of the scheduling problems is not an easy task. For instance, in this project, there are some soft constraints and objectives which are violating each other like Soft Constraint 1 and 10 in contrast to Equalize Objectives. Both Soft Constraint 1 and 10 ask the scheduler to avoid shift changing for doctors. while, based on the Equalize Objectives (Fairness), shift change is necessary to reach a balanced and fair distributed timetable.

To deal with this problem, We perform a significant number of tests to see the influence of each adjustment in the value of a each individual coefficient over the whole solution (output). In addition, we need to actively be in touch with the Thunder Bay Regional Health Sciences Centre’s Emergency Department to receive feedback and understand their preferences and goals, so we can recommend a better scheduling system.

O. *Solver Final State*

There are 5 types of states for the solver in the end which are Optimal, Feasible, Unknown, Invalid, and Infeasible. The scheduler get one solution (best solution) in the end, which only occurs if the final state equals to Optimal or Feasible.

1) Optimal: Optimal means that the obtained solution is the best solution possible considering the objective function.

2) Feasible: Feasible means that the obtained solution may not be the best possible solution. However, it is acceptable.

3) Unknown: Unknown state means that the model could not find any feasible solution yet. However, there might be a feasible solution if the model search for more.

4) Invalid: This state occurs if the solver cannot pass the first validation step.

5) Infeasible: Infeasible means that the model could never find a feasible solution, since there are not feasible solution available.

P. *Hard Constraint Relaxation*

As mentioned previously, the scheduler cannot violate any hard constraint in any situation. Considering the rigidity of hard constraints, the solver should eliminate a significant number of solutions which are in contrast with them. However, this strictness comes with a price. After running some tests, we realized that the scheduler cannot find any feasible solution (within a specific time). Therefore, the solver reach a final state of ”INFEASIBLE” or ”UNKNOWN” which is not acceptable. To solve this issue, we transferred Hard Rule 1, to the Soft Constraints with the highest priority (highest coefficient).

Therefore, the scheduler would consider the violations of this constraint while it tries to avoid it more than any other soft constraint or objective.

VI. Experimental Analysis

In this section, we show the best solutions offered by the scheduler. Then, we analyze and compare the value of the objective functions.

The inputs of the scheduler are the list of physicians, number of the days to schedule, Ontario Calendar, and shifts with their detailed information. The output of the system, is a recommended timetable for shift scheduling along with the detailed graphs and charts for better visualization, and lastly a log file to display the solver's final state and the number of violations for each soft constraint (objective function).

A. Shift Equality

Shift Equality (Fairness) is a significant aspects of this scheduler. Most of the constraints especially in the objective section are considering shift equality.

Figure 1, 2, 3 demonstrates the equality (fairness) in shift distribution for the final timetable recommended by the scheduler. The in this particular example we scheduled 29 physicians for the duration of 30 days where there is 12 shifts per day.

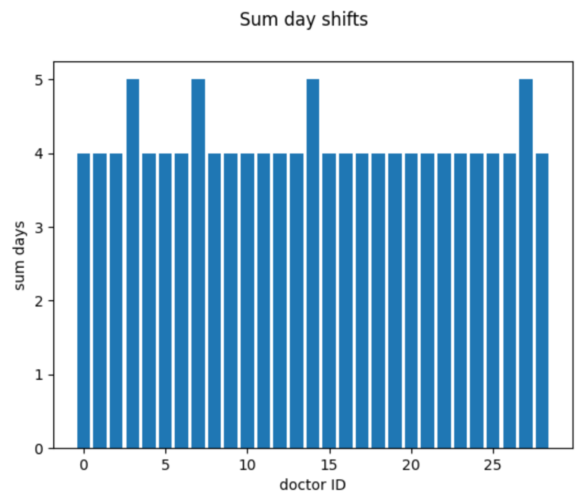


Fig. 1. Day shift distribution for objective 7 in bar chart

Figure 1 shows day shift (the shifts which start before 12 PM) equality using bar chart for objective 7. As shown, most of the physicians are assigned to work 4 days shifts in total and only few are working for 5 day shifts. Even though the chart in Figure 1 is not totally balanced, Since the number of the day shifts is not dividable by the number of the physicians, if we only consider objective 7, the following solution is optimal.

Again, in the same particular example, the scheduler found 3 holidays (using Ontario's calendar). Therefore, there are $3 \times 12 = 36$ shifts to distribute among physicians. Figure 2 shows how the scheduler distribute these holiday shifts fairly between all the physicians. In another bar chart in Figure 3, we can also see the 8 PM shift distribution for the physician.

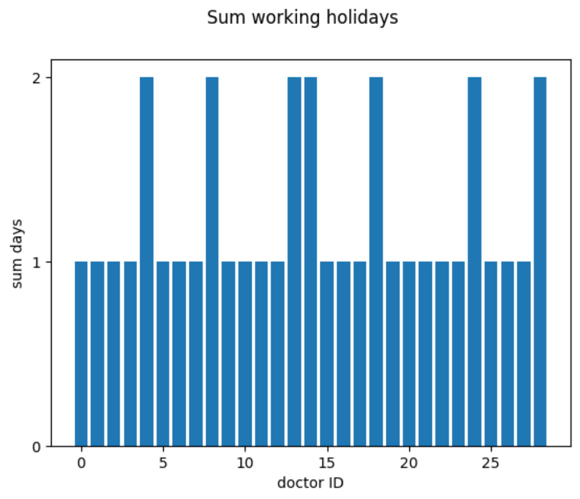


Fig. 2. Holiday shift distribution for objective 3 in bar chart

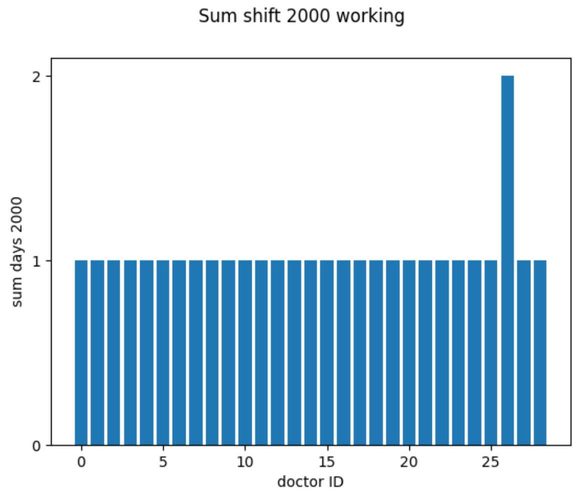


Fig. 3. 8 PM Shift distribution for objective 9 in bar chart

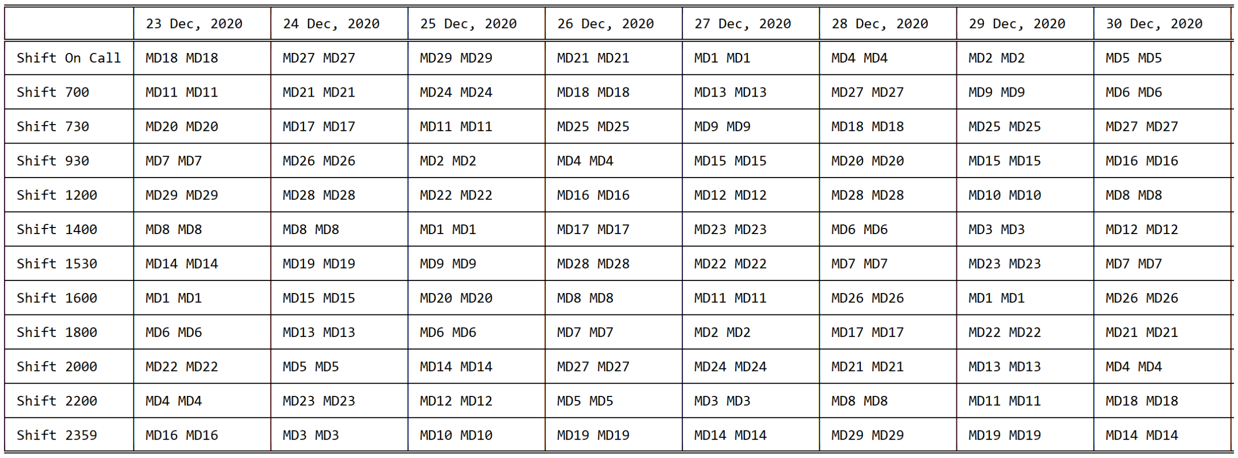
Note that, the provided bar charts in Figure 1, 2, 3 are for the same timetable (solution), which means the offered timetable by the scheduler satisfies all these objectives simultaneously. This could be applied to all the other shift equality objectives as well.

B. Timetable

Timetable is the final output of the scheduler which represents the shift assignments for the physicians during the scheduling time. Figure 4 displays the content of "timetable.txt" files for the first few days of a demo example. Each column of the timetable belongs to a specific day, and each row refers to a shift. For instance, in Figure 4 the cell in row 2, column 3 mentions Physician "MD11 MD11" is assigned to work on 25 Dec. 2020 in the 7:30 AM shift.

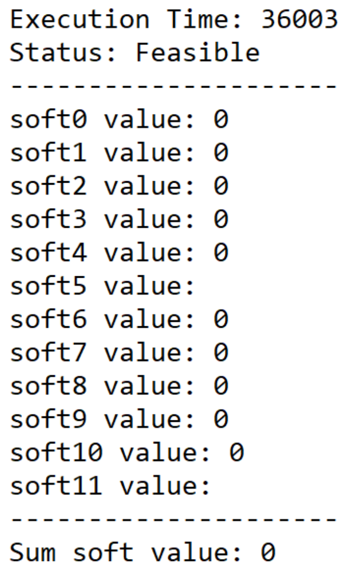
C. Soft Constraint and Objective Values

In addition to the shift equality objectives, there are other constraint that we need to satisfy as many as possible. Figure



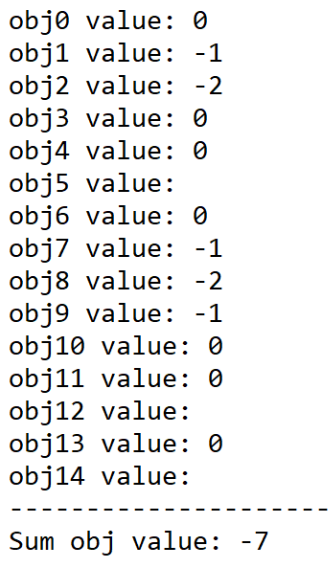
	23 Dec, 2020	24 Dec, 2020	25 Dec, 2020	26 Dec, 2020	27 Dec, 2020	28 Dec, 2020	29 Dec, 2020	30 Dec, 2020
Shift On Call	MD18 MD18	MD27 MD27	MD29 MD29	MD21 MD21	MD1 MD1	MD4 MD4	MD2 MD2	MD5 MD5
Shift 700	MD11 MD11	MD21 MD21	MD24 MD24	MD18 MD18	MD13 MD13	MD27 MD27	MD9 MD9	MD6 MD6
Shift 730	MD20 MD20	MD17 MD17	MD11 MD11	MD25 MD25	MD9 MD9	MD18 MD18	MD25 MD25	MD27 MD27
Shift 930	MD7 MD7	MD26 MD26	MD2 MD2	MD4 MD4	MD15 MD15	MD20 MD20	MD15 MD15	MD16 MD16
Shift 1200	MD29 MD29	MD28 MD28	MD22 MD22	MD16 MD16	MD12 MD12	MD28 MD28	MD10 MD10	MD8 MD8
Shift 1400	MD8 MD8	MD8 MD8	MD1 MD1	MD17 MD17	MD23 MD23	MD6 MD6	MD3 MD3	MD12 MD12
Shift 1530	MD14 MD14	MD19 MD19	MD9 MD9	MD28 MD28	MD22 MD22	MD7 MD7	MD23 MD23	MD7 MD7
Shift 1600	MD1 MD1	MD15 MD15	MD20 MD20	MD8 MD8	MD11 MD11	MD26 MD26	MD1 MD1	MD26 MD26
Shift 1800	MD6 MD6	MD13 MD13	MD6 MD6	MD7 MD7	MD2 MD2	MD17 MD17	MD22 MD22	MD21 MD21
Shift 2000	MD22 MD22	MD5 MD5	MD14 MD14	MD27 MD27	MD24 MD24	MD21 MD21	MD13 MD13	MD4 MD4
Shift 2200	MD4 MD4	MD23 MD23	MD12 MD12	MD5 MD5	MD3 MD3	MD8 MD8	MD11 MD11	MD18 MD18
Shift 2359	MD16 MD16	MD3 MD3	MD10 MD10	MD19 MD19	MD14 MD14	MD29 MD29	MD19 MD19	MD14 MD14

Fig. 4. Timetable of a demo example for the first few days



```
Execution Time: 36003
Status: Feasible
---------------------
soft0 value: 0
soft1 value: 0
soft2 value: 0
soft3 value: 0
soft4 value: 0
soft5 value:
soft6 value: 0
soft7 value: 0
soft8 value: 0
soft9 value: 0
soft10 value: 0
soft11 value:
---------------------
Sum soft value: 0
```

Fig. 5. Soft Constraint Scores (Soft Constraint 5 and 11 were exempt)



```
obj0 value: 0
obj1 value: -1
obj2 value: -2
obj3 value: 0
obj4 value: 0
obj5 value:
obj6 value: 0
obj7 value: -1
obj8 value: -2
obj9 value: -1
obj10 value: 0
obj11 value: 0
obj12 value:
obj13 value: 0
obj14 value:
---------------------
Sum obj value: -7
```

Fig. 6. Objective Scores (Objective 5, 12 and 14 were exempt)

5, 6 show the content of ”stat.txt” file, which is another output of the scheduler to show the execution time, the final state of the solver, and the soft constraint and objective scores (-1 for any violation). In the following example, we do not use coefficients to give weight to the constraints and we exempt Soft Constraint 5, 11 and Objective 5, 12 and 14 because of our current timing issues.

D. *Objective value improvement*

As CP-SAT Solver is running, it tries to find the first feasible solution which are not violating any Hard Constraint. Then, it calculates the value of the objective function and searches for another feasible solution with a lower punishment. The solver, would keep going until no feasible solution remains or the maximum scheduling time is over. After that, it returns the solution with the best cost function value.

Figure 7 displays the comparison of the cost function values (for each Soft Constraint) of the first feasible solution and the best one until the scheduler is run out of time.

In another example, in Figure 8 we can see the comparison of the objective values for the first feasible solution and the best one.

Note that the following results in Figure 7, 8 may even improve more (the punishment would decrease) if we increase the maximum execution time.

VII. Conclusion and Future Work

Scheduling is a critical and time consuming task which even need requires more attention when it comes to the emergency department in a hospital. Many companies, organizations and hospitals are scheduling their personnel and staff manually. The purpose of this project was to provide an Automatic Scheduling system to take the load off the administrative

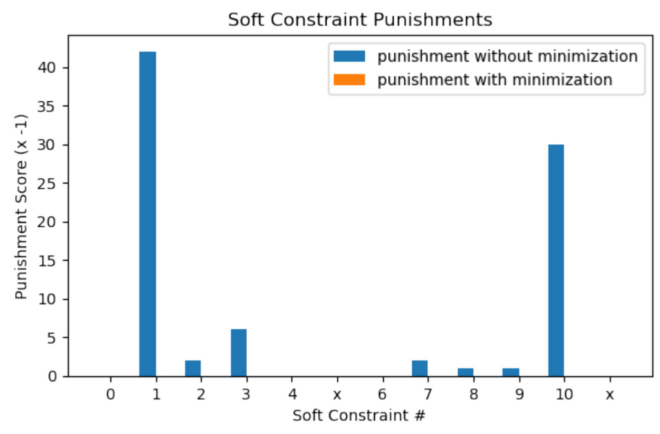


Fig. 7. Soft Constraint punishments (Soft Constraint 5 and 11 are exempt)

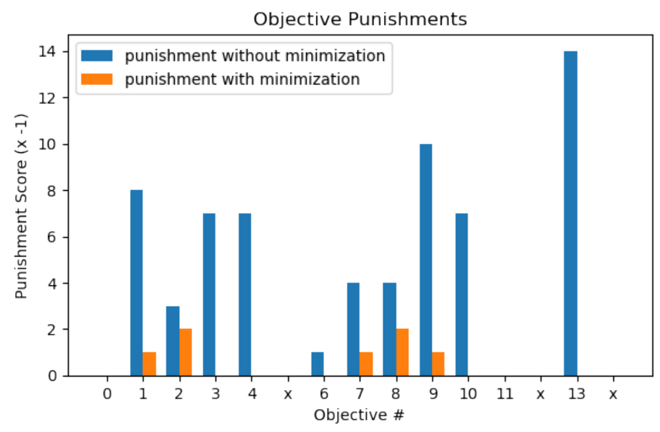


Fig. 8. Objective punishments (Objective 5, 12 and 14 are exempt)

professionals who are scheduling the timetable manually. The offered scheduler considers the rules and objectives of the Emergency Department and tries to find and recommend the best possible solution.

For future work we are planning to continue developing the scheduler, and we recommend the following possible improvements:

- Working on a refined technique which solves the issue of avoid shift changing and fairness objectives where they have negative interactions on each other. This could be done by improving the objective functions of these constraints or trying to relax some of them.
- Working on the equations and try to find new algorithms or implementation techniques which could result in an even faster computation.
- develop a scheduler which is scalable so we can distribute the work load in a distributed system. Since calculating the value of objective function each solution is independent to the others, we just need to make sure that each pair of computers are searching a different set of possible solutions.
- Adding more visual presentations to make it more convenience for later decisions in any possible manual adjustments.
- Expanding the experimental analysis which will discover improved solutions considering the quality and usability. For instance, having more experimental analysis and feedback would give us better chance to come up with a better coefficients for the objective function.
- Connect the scheduler to a software called ”OnCall Scheduler” by ”AMiON” which is currently used by the Emergency department of TBRHSC to distribute the final timetable among physicians on their devices (phone and computer).

References

[1] S. Petrovic and E. K. Burke, “University timetabling.” 2004.

[2] R. J. Vanderbei *et al.*, *Linear programming*. Springer, 2015, vol. 3.

[3] A. Schaerf, “A survey of automated timetabling,” *Artificial intelligence review*, vol. 13, no. 2, pp. 87–127, 1999.

[4] P. Van Beek and X. Chen, “Cplan: A constraint programming approach to planning,” in *AAAI/IAAI*, 1999, pp. 585–590.

[5] J. Gu, “Local search for satisfiability (sat) problem,” *IEEE Transactions on systems, man, and cybernetics*, vol. 23, no. 4, pp. 1108–1129, 1993.

[6] P. Hansen and B. Jaumard, “Algorithms for the maximum satisfiability problem,” *Computing*, vol. 44, no. 4, pp. 279–303, 1990.

[7] M. F. Zibran *et al.*, “A multi-phase approach to university course timetabling,” Ph.D. dissertation, Lethbridge, Alta.: University of Lethbridge, Faculty of Arts and Science, 2007, 2007.

[8] R. Mansini and R. Zanotti, “Optimizing the physician scheduling problem in a large hospital ward,” *Journal of Scheduling*, pp. 1–25, 2019.

[9] M. Gendreau, J. Ferland, B. Gendron, N. Hail, B. Jaumard, S. Lapierre, G. Pesant, and P. Soriano, “Physician scheduling in emergency rooms,” in *International Conference on the Practice and Theory of Automated Timetabling*. Springer, 2006, pp. 53–66.

[10] R. Bruni and P. Detti, “A flexible discrete optimization approach to the physician scheduling problem,” *Operations Research for Health Care*, vol. 3, no. 4, pp. 191–199, 2014.

[11] B. Jaumard, F. Semet, and T. Vovor, “A generalized linear programming model for nurse scheduling,” *European journal of operational research*, vol. 107, no. 1, pp. 1–18, 1998.

[12] L. Perron, “Operations research and constraint programming at google,” in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2011, pp. 2–2.

[13] K. A. Dowsland, “Nurse scheduling with tabu search and strategic oscillation,” *European journal of operational research*, vol. 106, no. 2-3, pp. 393–407, 1998.

[14] G. Weil, K. Heus, P. Francois, and M. Poujade, “Constraint programming for nurse scheduling,” *IEEE Engineering in medicine and biology magazine*, vol. 14, no. 4, pp. 417–422, 1995.