Modeling Relaxed Temporal Constraints in Operations Research Using Constraint Programming

Daniel LaChance, Glastonbury High School
Mentored by Greg Johnson, University of Connecticut

Abstract

Constraint Based Programming is a means of solving complex constraint satisfaction problems characterized by a tuple of variables, domains, and constraints that must be met in order to find a feasible solution. Our focus is on scheduling problems using constraint programming. One method for creating these constraints is temporal, a method that creates a schedule with strict ordering of all constraints; they must happen one after the other. In this study, we analyzed constraint based programming and how we can properly test and model less restrictive temporal constraints. Empirical study was done by producing a schedule for a FIRST robotics competition. In this competition, there are two repeated tasks: a five-minute competition that must be completed three times and a set of ten-minute practices, where all practices must be completed before the last competition. We used Google-OR Tools to effectively create two models that could be compared to each other. One model uses old temporal methods to create a solution, and the other uses a proposed relaxed temporal method to find a solution. Overall, the new, relaxed constraint was able to produce a shorter competition day because it was able to output solutions that made better use of the resources.

Introduction

Constraint programming (CP), a branch of artificial intelligence, is used to efficiently solve constraint satisfaction problems for various applications. These are problems in which there are a number of constraints that must be met in order to find a feasible solution (Hnich, 2004). Scheduling is a major application domain in which CP can be used to its fullest extent; indeed, constraint programming has become the dominant form of modeling and solving these problems (Fromherz, 2001). A classic example for CP is scheduling nurses at a hospital. The constraints would be that they all need to be scheduled for three shifts with no more than two night shifts, and that they need to work in a ten day cycle (Hoeve, 2006). Different types of scheduling problems have been identified to narrow down useful models in problem solving, one of which being cumulative scheduling models. Cumulative models have resources that can execute tasks in parallel, as long as they do not exceed the supply cap (Baptiste, 2006). This work develops a strategy for scheduling repeated tasks, scheduled among multiple alternatives, in two or more categories where all tasks in one category must complete before the final task in another category. We begin our inquiry for schedule solving with cumulative scheduling.

Background:
There are three parts to constraint satisfaction problems. A constraint satisfaction problem (P) is represented by a 3-tuple where:

> *P = {X, D, C} where X is a set of (n) of variables, D is a set of (n) of domains, and C is a set of (m) constraints (Freuder, 2006).*

Constraints can expressed as a relation between variables:

R_S and S_j where R_S is the relation of variables in scope(C) = S_i
(Freuder, 2006).

Constraints specify whether the values in the domains of variables, in the relation, can or can not be used together. A solution is an assignment to all variables in the set X, and feasible solutions satisfy all constraints (Freuder, 2006). Constraint programs are divided into a model where P is represented and a search component that attempts to find an assignment to all variables to form a feasible solution (Freuder, 2006). These solutions will be returned; therefore, if none are found, the problem is labeled *unfeasible* and the set of solutions will be left blank (Freuder, 2006).

Constraints can range from logical, or "precedes" condition, in which Job A must complete before Job B; to global constraints which are constraints that must apply to all variables, for example, *alldifferent()* (Kanet, 2004).

Constraint programming models contain constraints that once applied to variables, reduce the domains from which solutions can be drawn, which in turn, will reduce the search space of all feasible solutions the algorithm will be able to output (Baptiste, 2006). This manipulation and necessary criteria will vary from problem to problem; once a working one is produced, it can be changed to meet new criteria (Rousseau, 2002).

Cumulative scheduling models can be broken down further into logical models containing temporal as well as partial-ordering models. Temporal constraints allow for the implementation of timed succession to different variables, allowing for spacing of tasks to implement set-up/takedown times (Baptiste, 1995). On top of temporal constraints, there are partial-ordering constraints which similarly affect the spacing of task variables. There is research which defines partial-ordering constraints as a complex graph that represents a partial-ordering of some subset of tasks (Policella, 2007). Partial-ordering constraints are expressed as flexible temporal models in which each task has multiple assigned start times that can all be feasible (Policella, 2007). These schedules are produced by adding more relaxed precedence constraints, in mass, to existing models, allowing for each task to have a window applicable start times (Policella, 2007). In order to be partial-ordering, not all precedence of tasks need to be met, allowing for a wider range of solutions than a total-ordering model (Raja,, 2019). The focus of that research was to generate schedules that are resilient to delayed start times or non-standard completion times.

It has not yet been studied how to use constraint programming on a model of temporal in which the first task must complete before the other and there are alternative resources which can be used to complete a particular task. Constraint-based scheduling has only been studied through temporal and partial-ordering constraints, which are both more rigid than they need to be for some tasks. Temporal constraints are too restrictive because they have already laid out specific ordering of tasks. Partial-ordering constraints, while they may produce more resilient schedules, still restrict the pool of solutions by forcing a specific order for tasks. They also are more resource intensive and require specialized constraints that are not readily available with most constraint programming libraries. So, by relaxing partial-ordering constraints to fit new needs, the range of possible solutions will expand and more groups and tasks will be able to be accommodated. By simplifying the main ideas behind partial-ordering and implementing them using a standard set of constraints, the goal is to produce schedules that take less time/resources or support more tasks.

In order to satisfy this new relaxed version of the partial ordering constraint, this study aims to develop an efficient strategy to model the scheduling of tasks in two classes where all tasks of one class are scheduled before the last task of another class, and where they can be

scheduled in any order, except for the last task of the first class. This solution, being less restrictive than a temporal approach, is expected to produce schedules that accommodate more tasks or ones that complete in less time. To test our new constraint we will apply it to the scheduling of different repeated competition tasks in a First Robotics Competition. In this competition, there are a set number of teams at the competition event. At each event, each team has three types of tasks they must complete: three 10 minute practices, three 5 minute competitions and a 40 minute judging block, so all of these tasks must be scheduled without overlap. At the current state of scheduling for these competitions, there is an Excel spreadsheet that creates only a schedule for competitions and judging rooms. It leaves the scheduling of practices up to the coaches. Most importantly, however, we apply constraint programming to produce a fairer schedule that includes time for all practices to be completed before the last competition, so they are not useless. In order to schedule this problem efficiently, we used adapted constraint-based scheduling programs. Using discrete optimization tools allows us to schedule practice table time fairly to all teams so that it is useful since practicing after the last competition would not be useful. Adding these constraints to the state of practice scheduling tool, currently in Excel, is too complex and time consuming for a human to solve. The assistance of a constraint-based programming method will automate the process and quickly produce a new schedule if requirements change. If the tool is efficient enough, it can be used to compare different schedules with varying numbers of repeated tasks. Event organizers will be able to run a sensitivity analysis to test how the event's duration will change based upon changing the number of tasks or teams.

Methods

Creation of a Simple Cumulative Model (In a First Robotics Competition case study)

In the current case for First Lego League Competition scheduling there are 50 teams, all scheduled for three different types of tasks. A five minute competition, a ten minute practice and a forty minute judge-guided presentation. This study revolved around the optimization of the ten minute practice. These tasks were scheduled with four alternatives for the competition tables, four for the practice tables, and six for the judging rooms.

```
while count != int(team_number):
        jobs.append(
        [  # Job 0
            [(1, 0), (1, 1), (1, 2), (1, 3)],  # task 0 = Robot Game, 4 alt
            [(1, 0), (1, 1), (1, 2), (1, 3)],  # task 1
            [(1, 0), (1, 1), (1, 2), (1, 3)],  # task 2
            [(2, 4), (2, 5), (2, 6), (2, 7)],  # task 3 = Practice, 4 alt
            [(2, 4), (2, 5), (2, 6), (2, 7)],  # task 4
            [(2, 4), (2, 5), (2, 6), (2, 7)],  # task 5
            [(8, 8), (8, 9), (8, 10), (8, 11), (8, 12), (8, 13)],  # task 6 =
Judging, 6 alt
        ],)
        count += 1
```

To begin, a basic model must be outlined using the methods for later extrapolation: ten teams along with the five minute competition and forty minute presentation. Atom and Windows

Powershell, two programming tools, were used to build and test this basic model in the language of Python and the Google-OR Tools library. We chose Python as the programming language to build our research because it is a high-level language while still maintaining one of the most diverse data analytics portfolios.

Logistics:
In this base model each competition that was to be scheduled needed to be on one table out of four alternatives, and each judging block needed to be in one room out of eight alternatives. As per request by the competition administrator, for fairness, we made sure each team was on three different competition tables throughout the day.

Developing the New Constraint
We then adapted this program to include scheduling the practice times for every team. Similarly to the previous competition and judging tasks, we scheduled each practice from one table out of a variable number of alternatives. By creating two constraints to order team practice times: one being the previous temporal approach to the problem, and one being our new relaxed partial ordering approach, we were then able to compare the two. The first of these constraints used Google OR tools' "Add()" constraint where we partitioned a third of practice start times to begin before each competition:
"model.Add(starts[team_number, (i + 3 + int(int(practice_times)/3)))] < starts[(team_number, 1)])"
For the second of these constraints, we used Google OR tools' "AddMaxInequality()" function to define the start of the last possible competition and the start of the last possible practice. We then set the integer for the last competition start to be greater than the last practice start using the "Add()" constraint defined above:

```
count = 3
practice_starts = []
while count < (int(practice_times) + 3):
   practice_starts.append(
starts[(team_number, int(count))],
    )
if count > 3:
   model.Add(starts[(team_number, (int(count) - 1))] < starts[(team_number,
int(count))])
        count += 1
model.Add(final_comp_st > final_practice_st)
```

There is a difference between the number of solutions that can be drawn from the two different constraints. Consider a schedule that includes three practices and three competitions. The temporal constraint will only be able to produce one ordering, interweaving these two types of tasks; whereas, the more relaxed constraint proposed allows for 10 possible orderings and solutions instead of just one (see Figure 1).

Figure 1.

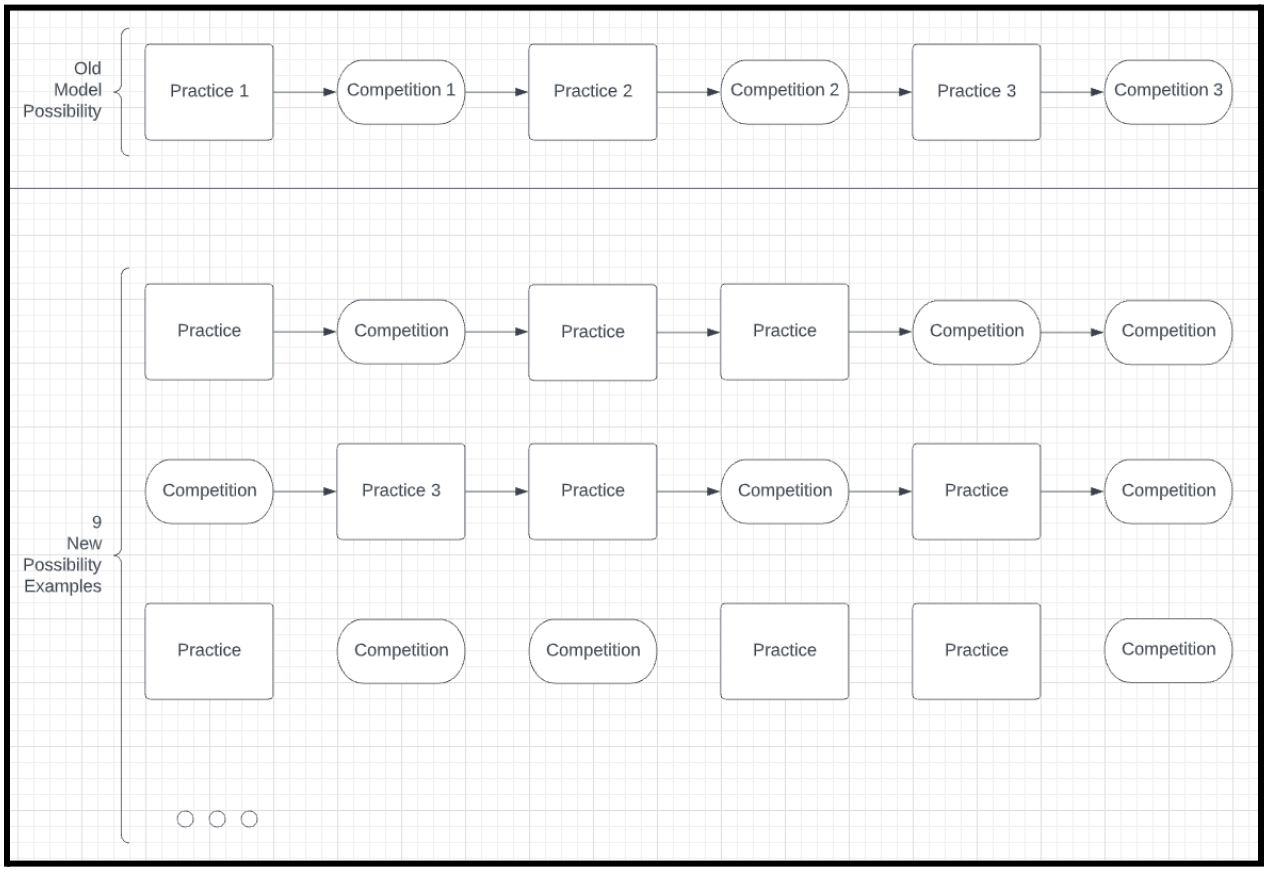


Figure 1: Comparison of temporal and relaxed-temporal constraints.

Collection of Data

Using the models in the previous section, we tested how three different variables (number of practices, number of practice tables, and number of teams) affected an array of measurable features from wall time, to maximum schedule length, to search effort measured by conflicts/branches. These features are directly affected by the constraints which we have been testing, so our goal is to test not only how our two constraints compare to each other, but where each constraint is unable to output a solution, making the solution infeasible.

A batch file method was used to run all analysis on the Windows command line. Trials for the number of practices measured the above dependent variables for three, four, five, and six practices while using four practice tables and forty teams:

- $i \in \{3, 4, 5, 6\}$ using 4 practice tables and 40 teams

Trials for the number of practice tables measured the same variables for four, six, eight, and ten tables while using three practice times and forty teams:

- $i \in \{4, 6, 8, 10\}$ Using 3 practice times and 50 teams

Trials for the number of teams measured the same variables for fifteen through forty teams, increasing by five each time. The same base values for three practice times and four practice tables were also used:

- $i \in \{15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 70\}$ using 3 practice times and 4 tables

This same batch file approach was repeated for program number two, containing the old, stricter temporal constraint.

Analysis of Data

Once all of the data were collected, four different types of graphs were generated in order to produce better visualizations of the differences. These graphs primarily focused on the maximum start time for competition tasks, or the time, in five minute increments, that the day would last.

The graphs most important to research are as follows:

- Number of practice tables vs. maximum time integer
- Number of practices vs. maximum time integer
- Number of teams vs. maximum time integer
- Program (temporal or relaxed) vs. Time for average solution

Results

All three parts of the research were conducted as stated in the methods, and the number of teams, the number of practice times, and the number of practice tables are shown with all mathematical outputs.

# of Teams	First Solution (s)	Last Solution (s)	Wall Time (s)	Max Value	Conflicts	Branches	Objective Value
20	0.21	0.24	300	44	57662	108819	32
25	0.28	0.28	300	46	52716	91567	40
30	0.44	0.45	16.32	62	3466	5876	40
35	0.596	0.69	20.6	71	3881	6744	48
40	1	1.84	300	80	25982	42313	56
45	0.92	1.82	300	87	33039	37674	64
50	1.25	1.25	300	95	26640	127163	72
55	1.43	1.43	300	104	14483	263885	80
60	1.74	3.37	62.63	112	6010	10786	80
70	2.41	2.41	300	134	2221	6967	96
20	0.21	0.22	300	44	80813	111127	32
25	0.35	0.37	300	46	121528	140156	30
30	0.45	0.45	10.76	62	1917	4137	40
35	0.59	0.59	300	77	2896	6045	48
40	0.8	1.66	300	82	35019	69289	56
45	1.04	1.04	300	92	47191	52345	64
50	1.38	1.38	300	98	29583	152407	72
55	1.76	1.76	300	108	31786	43423	80
60	2.07	3.17	66.4	126	4219	20346	80
70	2.99	3.08	300	140	1040	5834	96

# of Practice times	First Solution (s)	Last Solution (s)	Max Value	Conflicts	Branches	Solutions before final
3	1.15	1.15	98	21171	33332	0
4	2.14	2.14	120	1050	149760	0
5	2.31	2.31	146	23869	33776	0
6	2.86	2.86	174	19524	39816	0
3	1.35	1.35	102	31294	60283	0
4	1.88	1.88	134	27469	34703	0
5	2.34	2.34	158	21225	39708	0
6	4.12	4.12	184	20944	40814	0

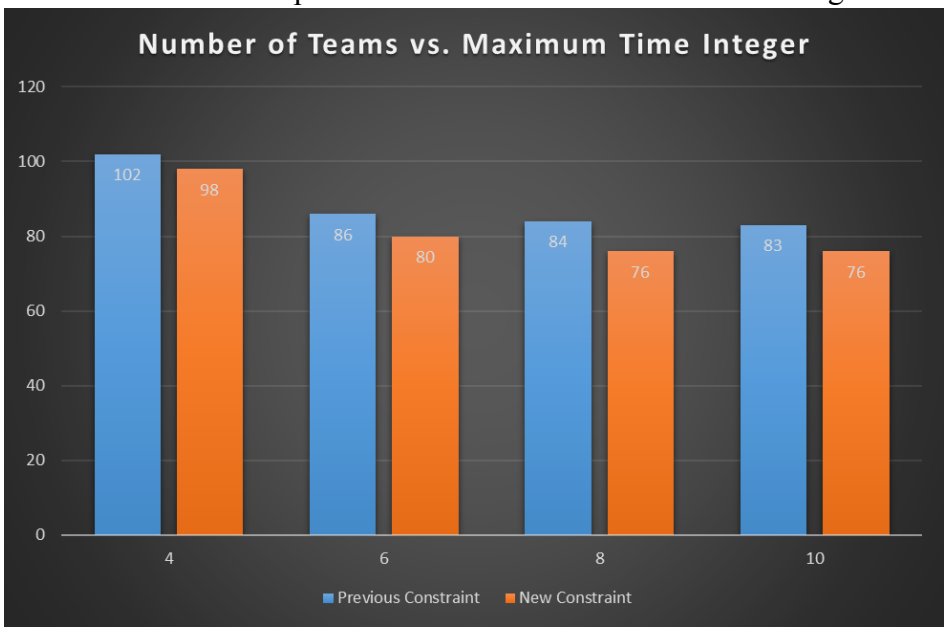
# of Practice Tables	First Solution (s)	Last Solution (s)	Max Value	Conflicts	Branches	Solutions before final
4	1.19	1.19	98	27151	129998	0
6	1.1	1.81	80	539169	746910	2
8	1.35	1.35	76	27121	53447	0
10	1.24	2.2	76	27011	40512	2
4	1.34	1.34	102	39553	54457	0
6	1.26	1.26	86	19725	211885	0
8	1.21	1.21	84	29312	58287	0
10	1.35	1.35	83	21949	203798	0

All of the data collected are shown here along with the time for the first solution, last solution, total wall time, maximum time integer, conflicts, branches, and objective value. Though we did not draw many conclusions from the data other than those conclusions we gained from the

maximum time integer (or value), the other data are presented above with trends similar to what we found with the maximum time integer.

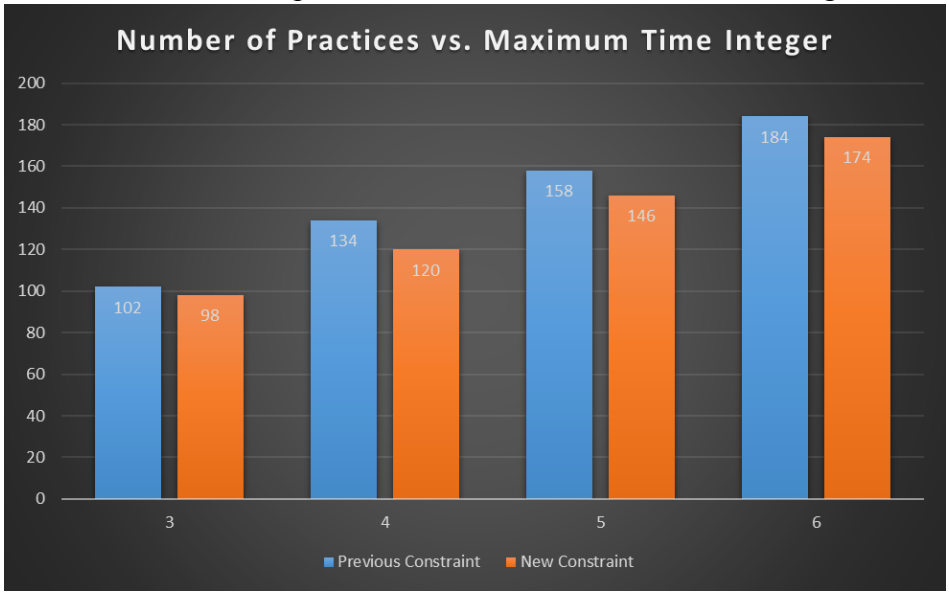
The four different graphs we created from this data are:

- The number of practice tables vs. the maximum time integer:



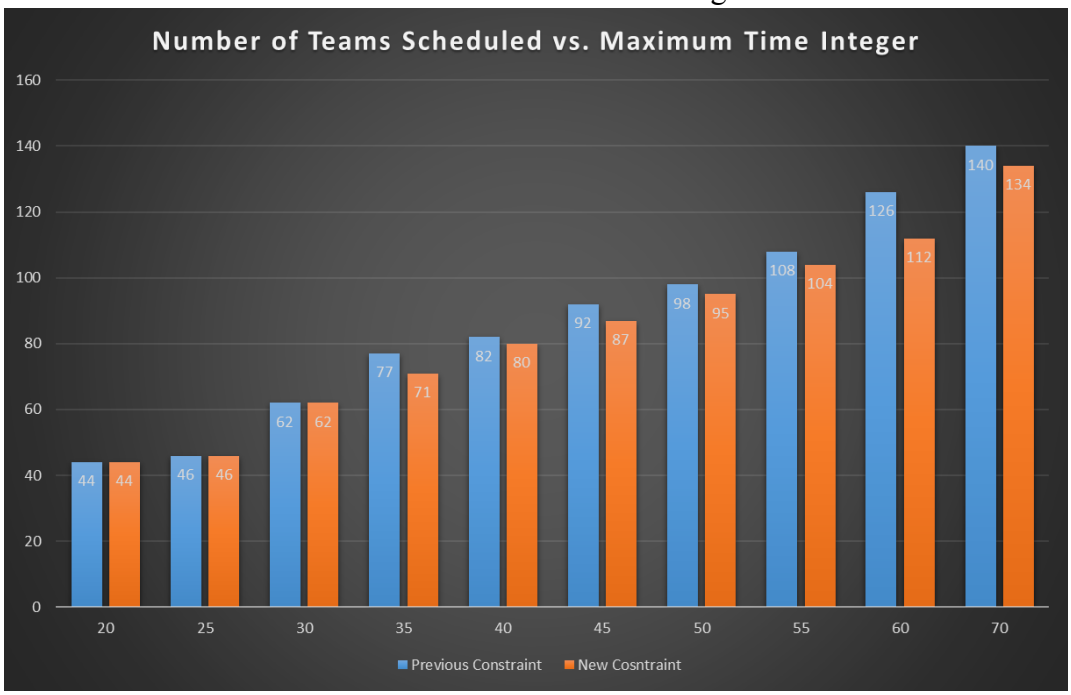
The maximum time integer, or the length of the competition day (in 5-minute increments), was compared to an increasing number of practice tables for both constraints. The number of practice tables was constant at 3, and the number of teams was constant at 40.

- The number of practice times vs. the maximum time integer:



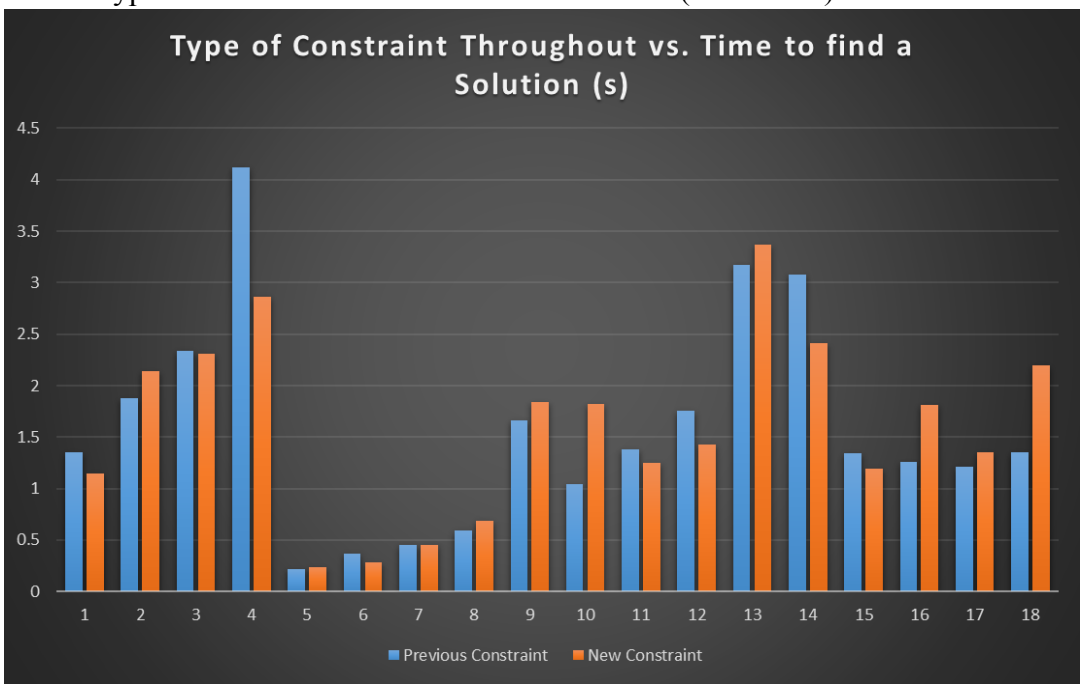
The maximum time integer, or the length of the competition day (in 5-minute increments), was compared to an increasing number of practices for both constraints. The number of teams was constant at 50, and the number of practice tables was constant at 4.

- The number of teams vs the maximum time integer:



The maximum time integer, or the length of the competition day (in 5-minute increments), was compared to an increasing number of teams for both constraints. The number of practices was constant at 4, and the number of practice times was constant at 3.

- Type of constraint vs. time to find a solution (in seconds)



Throughout this graph, the y-value is the average time to find a solution (s), and on the x-axis:

- 1-4 represents the trials in which the number of practice times was changed.
- 5-14 represents the trials in which the number of practice tables was changed.

- 15-18 represents the trials in which the number of teams was changed.

Average time for a solution came out to be 1.587222 seconds for the old constraint and 1.5994444 seconds for the new constraint.

Discussion

From the results, it can be concluded that the new constraint does indeed open up new possibilities that allow for a lower time integer/shorter day in our case study. The three graphs and data pertaining to the maximum time integer show a significant difference between the models. By opening up these new possibilities explained in Figure 1, it is possible to shorten the running time of the competition. In every circumstance, the day was shorter with the new constraint. The most notable trial was that with 60 teams practicing three times each on four tables. The difference between the two constraints in this case was 126 5-minute increments for the old constraints and 112 5-minute increments for the new one. By relaxing the constraints, it was possible to open up nine new possibilities, allowing for a shorter day in most cases. By shortening the entire day by 14 5-minute increments, 1:10 of time is given back to every participant in this case. If this time was not taken out of the schedule, all teams and judges would be staying there for an empty 1:10 longer where they would not be doing anything productive. Though it would be dispersed throughout the day, it is unnecessary time and our model can do away with it by relaxing constraints. This new model has a real effect on the experience of everybody at these competitions.

It was expected that the new constraint would take slightly longer to output a solution because it would be parsing through 10 times the possible outputs as well. This miniscule amount of time opened up new possibilities and outputs consistently shorter events with less downtime.

We also assessed branches and conflicts and how they are affected by the different constraints. Branches are where it made a guess and will explore it through until it either finds a solution or a conflict. A conflict is where no solution can be found and the program backtracks to an earlier point. We concluded that they are mainly inconsistent but the number of branches for the new model is always greater, meaning; using this model, the search considers more alternatives in the same amount of time.

Moving forward, it would be important to expand on where this new model actually breaks, or can not find a solution. These are powerful constraints, and it is possible that the program could find a solution with a much larger number of constraints if it had much more time than the allotted 5 minutes. With more research, it could be discovered that these models actually produce better schedules if they had more time.

Using Google OR-Tools allows us to model this problem using a core set of constraints and provides a generic search. Using more sophisticated constraint programming libraries would allow us to write custom constraints as well as custom search procedures, so it is possible that different conclusions could be drawn from other methods and programming languages. With these new procedures, the search process may be able to finish in the allotted time. Other than

the programming language and tools, there were few outliers in the data that would suggest an issue.

By relaxing the constraints, we were indeed able to shorten the outputted day. This new schedule is also fairer because it forces teams to run competitions on separate tables each time. The tool is able to schedule times on practice tables fairly so all teams get the same number of practice sessions and it removes the burden of trying to sign up for practice times manually from coaches and parents. It also allows organizers to have new schedules in minutes and it is much quicker, simpler, and easier than current methods.

Acknowledgements

I would like to express my gratitude to the Glastonbury Board of Education for providing me with the opportunity to complete this original research.

References

Baptiste, Philippe, et al. "Constraint-based scheduling and planning." *Foundations of artificial intelligence* 2 (2006): 761-799.

Baptiste, Philippe, Claude Le Pape, and Wim Nuijten. "Incorporating E cient Operations Research Algorithms in Constraint-Based Scheduling." *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research, Timberline Lodge, Oregon*. 1995.

Fromherz, Markus PJ. "Constraint-based scheduling." *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*. Vol. 4. IEEE, 2001.

Google. (n.d.). *OR-tools | google developers*. Google. Retrieved April 21, 2022, from https://developers.google.com/optimization

Hnich, Brahim, Steven Prestwich, and Evgeny Selensky. "Constraint-based approaches to the covering test problem." *International Workshop on Constraint Solving and Constraint Logic Programming*. Springer, Berlin, Heidelberg, 2004.

Kanet, John J., Sanjay L. Ahire, and Michael F. Gorman. "Constraint programming for scheduling." (2004).

Policella, Nicola, et al. "From precedence constraint posting to partial order schedules." *Ai Communications* 20.3 (2007): 163-180.

Raja, A. George Louis, F. Sagayaraj Francis, and P. Sugumar. "Deriving the Partial Order of Documents to Extend Clustering Applications." (2019).

Rousseau, Louis-Martin, Michel Gendreau, and Gilles Pesant. "Using constraint-based operators to solve the vehicle routing problem with time windows." *Journal of heuristics* 8.1 (2002): 43-58.