Tutoring Schedule for Statistics Tutor and Study Center

Authors: Liyuan Tang [1], Long Kim Tran [2], You Sen Wang [3] [4]

Abstract

The Statistics Tutor and Study Center provides free drop-in tutoring for all students enrolled at UW. They want a weekly tutor schedule based on each tutor's available time, the manager's preference and administrative constraints. We used mixed integer quadratic programming to create a Discrete Mathematical Model using Python with Gurobi package to help the Lead Tutor, Sheridan Grant, generating a schedule for all the tutors. We tried different methods to approach the optimal solution, such as changing the time slots and separating the schedule in two parts in order to decrease the model size. The result indicates the optimal schedule for the statistics tutoring center. We tested the resulting schedule by comparing with the current schedule to validate our model.

Problem Description

Tutoring centers are significant factors in students' academic success. By solving problems that tutoring centers have, it is beneficial not only to students who can get help from the tutors but also to the tutors themselves. Since most tutors are students too, improving the schedule helps them utilize their time more efficiently. Moreover, our project can make supervisors' jobs easier by creating a fair and balanced schedule among tutors which means every tutor should work a similar number of hours every week.

The focus of our project will be on helping the Lead Tutor Sheridan Grant of the Stat and Study Center to create a schedule. Sheridan generated the current schedule by hand, which is inefficient. Thus, the tutoring center is in need of a more efficient way to generate a proper schedule.

Our goal is to create a 'fair, balanced and generally consecutive' weekly working schedule for statistics tutors such that every time slot between 9:30am to 5:00pm and 7:00 pm to 9:00 pm on Monday to Thursday is covered by exactly one of the 12 tutors, no tutor works for more than 4 hours per week and all tutors are scheduled during their available time.

[1]Department of Applied Computational and Mathematical Science, University of Washington, liyuant@uw.edu
[2]Department of Applied Computational and Mathematical Science, University of Washington, tran6388@uw.edu
[3]Department of Applied Computational and Mathematical Science, University of Washington, youw208@uw.edu
[4]Date: December 9, 2018

Background Information

Scheduling problem is a very common problem that a lot of businesses and organizations encounter during their operation. Many people have tried to tackle the issue and each person would come up with different methods to tackle different situations. One such example is the scheduling proposal by Caleb White and his team [1]. They are the professor and undergraduates at the University of Washington. They tried to generate a schedule for the Math Study Center (MSC). The model they proposed implemented binary integer programming and they called it the penalty model. The data they collected from the tutors at MSC contained the preference of the tutor in their available time from 1 to 3 (1 indicate their most preferred time slot and 3 is the least preferred). After using computer programming to generate the schedule, they compared it with the 2003 Summer, Winter and Autumn schedules as "representative instances on which to test" their proposed schedule. The results of their paper indicated that their proposed schedule is faster and provided a better solution. It is more flexible for the tutors; but most importantly, it "does not require the overhead of sorting out and removing the in-feasible constraints" [1]. After their project was done, Patrick Perkins has been using this model to generate schedules for the Math Study Center at UW. The employees at the Center have shown satisfaction with the new schedules than their old ones.

Requirements, Simplifications, and Assumptions

In this project, we will need to make a few assumptions in order to create an optimal schedule for the tutors. From the data that we collected, we first need to assume that the schedule for drop-in tutors is from 9:30 am to 5:00 pm with 30 minutes shifts and 7:00 pm to 9:00 pm with a 2 hours shift.

Assume there is a number n of tutors that can be used for filling the schedule; in the case of this quarter, for example, there are 12 tutors. One of the big requirements is that our formulated schedule needs to respect the available time of each tutor based on the time availability table that our community partner gave us.

There are also some hard constraints like each tutor should work at least 1 hour and at most 4 hours per week. The director also prefers to have people's shift be consecutive in general, which is the soft constraints in our model. We also need to assume the schedule will not change during the quarter so we are solving a static

problem. We further assumed the demand for the tutoring center is constant over time and for each time slot the center needs exactly one tutor.

Mathematical model

The plan for the project is to use mixed integer quadratic programming as a nonlinear programming problem covered in *Operations Research*[3]. Moreover, we will use the four-stage modeling method as the way to develop the project. We will first translate the real world problem into a mathematical model. The objective function is to minimize the variances of total working hours for each tutor in Statistics Tutor and Study Center.

After creating a working model, we will proceed to the second step which is the prediction step. We would use the mathematical model with all the constraints be entered into a computer that can solve this nonlinear programming problem to get a solution. Then, we will interpret our model and compare the current schedule the center has. Finally, we will process to the testing phase. As much as we like the supervisor of the tutoring center to try out our schedule, this quarter's tutoring schedule is already determined. Instead, we would use the previously used schedule, calculate its variance of total working hours per week of all tutors, and see whether we successfully create a fair and balanced schedule. We will refine our model according to the results.

First, we need to set up our input. We defined the auxiliary variable a_{ij} representing that tutor i is able to work at shift j with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, 64\}$. The value of a_{ij} is either 0 or 1. It is 0 if the tutors aren't available at that time slot; and 1 indicates they are available in that time slot. Then, we transformed the table that our community partner gave us into the auxiliary variable a_{ij}. We included a small section of the table in the figure below (Figure 1). Next, we replaced every green "OK" by value of 1, and filled the blank cells by 0 (Figure 2).

Initially, when we used this data which contains $12 \times 64 = 768$ tutoring slots as our input, our computer ran into the problem of "MEMORY ERROR" due to a large number of possible permutations. Thus, we need a way to scale down our table size to help running the program. The supervisor prefers tutors to work consecutively and with less shifts. We exploited this information and implemented this constraint by changing the size of time slots here. We will actually add this soft constraint into our program as another way which will be discussed in the remark.

2:00 PM – 2:30 PM	2:30 PM – 3:00 PM	3:00 PM – 3:30 PM	3:30 PM – 4:00 PM	4:00 PM – 4:30 PM	4:30 PM – 5:00 PM	7:00 PM – 9:00 PM
			OK	OK	OK	OK
						OK
OK			OK	OK	OK	OK
						OK
OK						
OK	OK	OK				

Figure 1: This is part of the original raw data set. Each row represents each tutor's available time during the week. Here we have the time availability information on Monday from 2:00 PM to 5:00PM and from 7:00 PM to 9:00 PM.

2:00 PM – 2:30 PM	2:30 PM – 3:00 PM	3:00 PM – 3:30 PM	3:30 PM – 4:00 PM	4:00 PM – 4:30 PM	4:30 PM – 5:00 PM	7:00 PM – 9:00 PM
0	0	0	1	1	1	1
0	0	0	0	0	0	1
1	0	0	1	1	1	1
0	0	0	0	0	0	1
1	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
1	1	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Figure 2: Converting 'OK' into 1 and filling empty cells into 0.

The size of the table that we have right now is 12 x 64 (12 represents the number of tutors, 64 represents the number of shifts with 30 minutes each, except for column 16, 32, 48 and 64 which are 2-hour shift at the end of each day). We decided to increase the size of a time slot during the day from 30 minutes to 90 minutes without changing the size of night shifts. We combined three adjacent columns in the old table by adding the values together to generate a new column except for column 16, 32, 48 and 64 since these 4 columns represent the night shift. For each single day, there used to be 16 time slots, 15 of them are 30 minutes and the last one is 2 hours. In the new table in Figure 3(a), we only have 6 time slots in one single day, the first five are 90 minutes and the last one is 2 hours. Thus, we scaled it down to a table with the size of 12 x 24.

Notice that the number of rows is the same, but we change the number of columns from 64 to 24 which highly decreases the size of our model. Therefore, we converted the table in Figure 2 to a new table in Figure 3(a) which is easier to run the code.

However, in order to import this table into Python as binary variables that could provide a solution, we need to further modify the table in Figure 3(a). If a value in a new cell in Figure 3 (a) during day shifts is either 0, 1, or 2, we convert the value to 0. This means that tutor is not available during the full 90-minute shift. If that value is 3, we convert it into 1. This means that this tutor can work the entire 90-minute shift consecutively. Finally, we keep the value during the night shift the same (Figure 3(b)).

2:00 - 3:30	3:30 - 5:00	7:00 - 9:00
0	3	1
0	0	1
1	3	1
0	0	1
1	0	0
0	0	0
0	0	0
0	0	0
0	0	0
3	0	0
0	0	0
0	0	0

(a) The number of available 30 mins time slots for each 90 mins time slot.

2:00 - 3:30	3:30 - 5:00	7:00 - 9:00
0,	1,	1,
0,	0,	1,
0,	1,	1,
0,	0,	1,
0,	0,	0,
0,	0,	0,
0,	0,	0,
0,	0,	0,
0,	0,	0,
1,	0,	0,
0,	0,	0,
0,	0,	0,

(b) Converting 2 and 1 into 0 and converting into 1 if and only if 3 consecutive 30 mins time slots are available for the day shift.

Figure 3: Converting 30 mins available time slots to 90 mins available time slots.

We set up the decision variables as x_{ij} to take as binary integers 0 and 1. If x_{ij} is 0 then tutor i does not work at time slot j; if x_{ij} is 1 then tutor i works at time slot j. We then defined a new variable t_i for convenience, which represents the total working hours for tutor i per week. Here t_i can be expressed in terms of x_{ij},

$$t_i = \sum_{j=1}^{5} 1.5x_{ij} + \sum_{j=7}^{11} 1.5x_{ij} + \sum_{j=13}^{17} 1.5x_{ij} + \sum_{j=19}^{23} 1.5x_{ij} + 2(x_{i6} + x_{i12} + x_{i18} + x_{i24}). \quad (1)$$

Here we use 1.5 and 2 since the night shifts are 2 hours and we changed the time slots during the day from 0.5 hour to 1.5 hours. Let $\bar{t}$ be the average number of working hours for n tutors per week, where n is 12 here.

$$\bar{t} = \frac{\sum_{i=1}^{12} t_i}{12}. \quad (2)$$

At this point, we will have our objective function which is

$$VAR(t) = \min \sum_{i=1}^{12} \frac{(t_i - \bar{t})^2}{12 - 1}. \quad (3)$$

This objective function represents the formula of calculating the variance of the working hours for each tutor and we want to minimize the variance to create a balanced schedule as the goal for our project. However, in order to easily program in Python, we need to simplify the equation further without any constants. Thus, we will have the objective function as

$$\min \sum_{i=1}^{12} (t_i - \bar{t})^2. \tag{4}$$

Now, we will need to add constraints into our model to make it work. For the first hard constraint, we need to make sure each shift will have exactly one tutor working. Therefore, for a fixed j,

$$\sum_{i=1}^{12} x_{ij} = 1. \tag{5}$$

For the second hard constraint, we need to make sure each tutor will work at least 1 hour and no more than 4 hours per week. For a fixed i,

$$1 \leq t_i \leq 4. \tag{6}$$

Moreover, in order to create a balanced working schedule, we added more constraints that the total working hours for every tutor on each day should be less than or equal to 3 hours. This is because tutors get tired when they work too long. Limiting their working hours to at most 3 hours a day would ensure the tutors' working performance. Another hard constraint is each tutor should be assigned during their available time. We add this constraint by using the auxiliary variables a_{ij},

$$0 \leq x_{ij} \leq a_{ij}. \tag{7}$$

a_{ij} is the time availability for tutor i at time slot j which we have defined in the previous steps.

A soft constraint comes from the preference of the director of the Statistic Study Center is that the total working hours for a tutor should be greater than or equal to 1.5 times the number of distinct shifts. A distinct shift is defined as one single shift or some continuous shifts. For example, if a tutor works from 9:30 to 11:00 without any break, he only has 1 distinct shift even though he takes three

30-minute shifts. Another thing that needs to be mentioned is that if a tutor works from 4:30 pm to 5:00 pm and then continues to work from 7:00 pm to 9:00 pm, it will be counted as 2 distinct shifts instead of 1 distinct shift. This soft constraint means that a tutor should work at least 1.5 hours for 1 distinct shift. Since we changed the time slot to be 1.5 hours in the current model and it automatically guarantees that the solution meets this constraint, we no longer need to add it to our model.

By using the model that we developed above, we will be able to predict a working schedule that would create a fair and balanced schedule that would respect to time availability of all the tutors. For the future years, the director of the Statistic Tutor Center can use the model for any number of tutors and availability time constraints. Therefore, the model can be a great tool for the department.

Solution

We have used Python-Gurobi as the program to generate the schedule base on the model that we developed above. The whole programming code can be found in the Appendix located at the back of the paper. Below is the pseudo code for the program that runs on Python.

Pseudo code:
Import all the necessary files from python.
Import the time availability matrix (a_{ij}) which size is 12×24.
Create a list of tuples which size is 12×24.
Select the tuples where the corresponding values in the matrix are 1.
Update variables by using these selected tuples.

Add Constraints into the model:
  every tutor work at least 1 hour per week
  every tutor works no more than 4 hours per week
  every tutor works no more than 3 hours per day
  every shift has exactly 1 tutor.

Objective function:
var = sum($(t_i - \bar{t})^2$ for i in range(12)).

m.setObjective(var, GRB.MINIMIZE).

Another trick we did here is to only add variables that can be non-zero into our model. For example, if tutor 1 is not available at time slot 1, which means $a_{11} = 0$, then x_{11} has to be 0 so we don't add x_{11} into our model. So we don't need to include equation (7) in our model once we use binary as the type of our variable.

If one tutor suddenly quits, we can just change the input data by deleting the row that represents that tutor and run the program again. With 11 tutors, we can still fill the schedule with each tutor works less than 4 hours per week.

If there is no tutor available at a particular time slot, the program would not give us a solution because there is a constraint that there should be exactly one tutor work in each time slot. The input for that column/time slot will just be all zeroes. When that happens, we should modify our input beforehand using a function. The function adds one more row representing Sheridan's available time slots and initialize it with all zeroes. Then, it checks whether each column/time slot has at least one tutor available. If there is not, it changes the initialized 0 in that column in Sheridan's row into 1. If this row is all zero, which means each time slots is covered by at least 1 tutor, then we just used the original data. If this row has at least one nonzero element, then we should use this new data with n+1 row as our input. By using this method, it should not change the current schedule since Sheridan will be only assigned to the shift that no tutors are available.

Results

Our solution can be interpreted in two ways as a complete schedule (Figure 4) and individual schedule (Figure 5). One can verify that this result is feasible since it meets all the constraints. In this schedule, all tutors work no more than 4 hours per week and each time slot has exactly 1 tutor. Also, it's guaranteed to meet the soft constraints. Since we changed the time slot to be 1.5 hours, if a tutor has one shift, she/he must work for at least 1.5 hours. As a result, the proposed schedule would meet the condition that total working hours is greater or equal than 1.5 times the distinct shift.

When we compared the bar charts of total working hours in two schedules (Figure 6), it also shows that our schedule is more balanced. The variance of our solution is 0.1060606. At the same time, in order to validate our results, we calculated the variance of the current schedule that is on their website, which is 1.072727. We have 12 tutors in our model and there are currently only 11 tutors working because

one tutor quit. However, since we are comparing the statistics of two schedules, the sample size doesn't matter. Obviously, the variance from the proposed schedule is much smaller than the current one so we believe this is a better schedule.

Time	Monday	Tuesday	Wednesday	Thursday
9:30 – 10:00	9th tutor	11th tutor	9th tutor	11th tutor
10:00 – 10:30	9th tutor	11th tutor	9th tutor	11th tutor
10:30 – 11:00	9th tutor	11th tutor	9th tutor	11th tutor
11:00 – 11:30	8th tutor	12th tutor	8th tutor	2nd tutor
11:30 – 12:00	8th tutor	12th tutor	8th tutor	2nd tutor
12:00 – 12:30	8th tutor	12th tutor	8th tutor	2nd tutor
12:30 – 1:00	5th tutor	12th tutor	5th tutor	1st tutor
1:00 – 1:30	5th tutor	12th tutor	5th tutor	1st tutor
1:30 – 2:00	5th tutor	12th tutor	5th tutor	1st tutor
2:00 – 2:30	10th tutor	3rd tutor	10th tutor	2nd tutor
2:30 – 3:00	10th tutor	3rd tutor	10th tutor	2nd tutor
3:00 – 3:30	10th tutor	3rd tutor	10th tutor	2nd tutor
3:30 – 4:00	3rd tutor	6th tutor	1st tutor	4th tutor
4:00 – 4:30	3rd tutor	6th tutor	1st tutor	4th tutor
4:30 – 5:00	3rd tutor	6th tutor	1st tutor	4th tutor
7:00 – 9:00	**4th tutor**	**7th tutor**	**7th tutor**	**6th tutor**

Figure 4: Optimized Drop-In Stat Tutoring Schedule (1.5 hours)

```
Tutor 1 works from 15:30 to 17:00 on Wednesday.
Tutor 1 works from 12:30 to 14:00 on Thurday.
Tutor 2 works from 11:00 to 12:30 on Thurday.
Tutor 2 works from 14:00 to 15:30 on Thurday.
Tutor 3 works from 15:30 to 17:00 on Monday.
Tutor 3 works from 14:00 to 15:30 on Tuesday.
Tutor 4 works from 19:00 to 21:00 on Monday.
Tutor 4 works from 15:30 to 17:00 on Thurday.
Tutor 5 works from 12:30 to 14:00 on Monday.
Tutor 5 works from 12:30 to 14:00 on Wednesday.
Tutor 6 works from 15:30 to 17:00 on Tuesday.
Tutor 6 works from 19:00 to 21:00 on Thurday.
Tutor 7 works from 19:00 to 21:00 on Tuesday.
Tutor 7 works from 19:00 to 21:00 on Wednesday.
Tutor 8 works from 11:00 to 12:30 on Monday.
Tutor 8 works from 11:00 to 12:30 on Wednesday.
Tutor 9 works from 9:30 to 11:00 on Monday.
Tutor 9 works from 9:30 to 11:00 on Wednesday.
Tutor 10 works from 14:00 to 15:30 on Monday.
Tutor 10 works from 14:00 to 15:30 on Wednesday.
Tutor 11 works from 9:30 to 11:00 on Tuesday.
Tutor 11 works from 9:30 to 11:00 on Thurday.
Tutor 12 works from 11:00 to 12:30 on Tuesday.
Tutor 12 works from 12:30 to 14:00 on Tuesday.
```

Figure 5: Our optimized working schedule

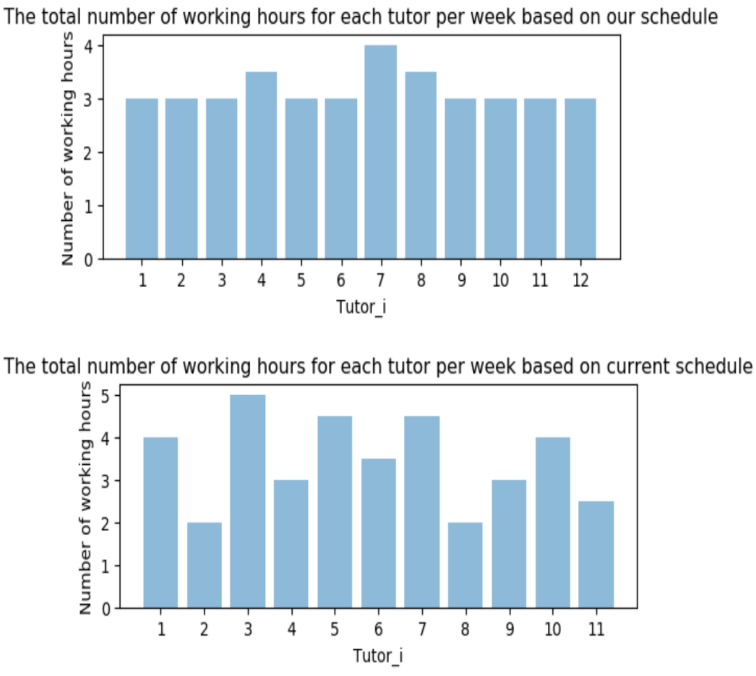

Figure 6: Bar chart of total working hours for each tutor in current schedule and our optimized schedule

Remark

Beside using Gurobi package in Python, our group also tried another approach by implementing MATLAB as the running programming. However, MATLAB's 'quadprog' function can only handle quadratic objective function with linear constraints; so, we cannot solve our problem with it. Even if the command is able to handle non-linear quadratic constraints, MATLAB requires users to write the objective functions and all the constraints in vector forms and in terms of the decision variables. It is would take unreasonably long time since we need to write out every t_i in terms of x_{ij} by hand. Thus, Gurobi is a better approach because it not only handles both quadratic constraints and quadratic objective function but also allows us to shorten our notations by redefining variables in the objective function.

Another way to implement the soft constraint is to encode it with other constraints in Python. This turned it into a hard constraint. However, we still have that memory overload issue so we also need a new way to reduce our input size. Based on UW's course schedule, most courses offered on Monday would also be offered at the same time on Wednesday. We utilized this fact and split the availability table with 30-minute time slots into two tables. One with 6 tutors for Monday and Wednesday and the other with the other 6 tutors for Tuesday and Thursday. Here we assumed that the first set of tutors could only work on Monday and Wednesday and the second set of tutors could only work on Tuesday and Thursday. During the assigning process, we first checked if there is any time slot that only has 1 tutor available to work. If it does, then we have to assign that tutor on that day. For example, in Figure 1, tutor 10 is the only person available from 2:30 pm to 3:30 pm on Monday which means that tutor 10 has to be assigned into the set of Monday and Wednesday. We tried to use a greedy algorithm to separate the rest of the tutors. At this point, we only needed to solve the problem with a 6×32 sized table twice. After implementing this process, we found a schedule with a variance which is two times bigger than the variance of the current schedule. Because of this high variance, we decided not to implement this into our model. This also further shows that the current proposed schedule is optimal so far.

Improvements

The project successfully generated a working schedule that could be used by the Statistic Study Center. However, the schedule seems to be lacking in flexibility

in helping the tutor moving their time more efficiently. In particular, each tutor needs to work for at least 90 minutes per shift. This may not work with for all the tutors. Therefore, our group would make a recommendation to Sheridan to change the way the center collect data in Figure 1. For example, the center could collect data on the available time of tutor based on their preference ranking scores for each time slot from 1, 2 and 3. It means that the tutors can rank their available time in number to indicate the time they want to work most and the time they want to work the least. They can rank the available time slot they want most as 1 and the time slot they want to work the least as 3. Therefore, we would have a data set that would reflect the real reference of all the tutor. Thus, we could improve our model by adding more constraints that would generate a schedule that would satisfy all the tutors.

Besides, we can change the time slot from 90 minutes to 60 minutes. This can increase the flexibility of scheduling. There are 7.5 working hours during the day so we would have an extra 0.5 hours. We need to decided which 30-minute time slot to represent that 0.5 hours and put it into our model as an extra term. Nonetheless, we still need to make sure our computers are able to handle the memory needed for the programming to run.

One more improvement that we could apply to the project is to create a user interface that could help our community partner interact with the Python program more easily. Currently, our Python program requires the users to enter all the code by hand into the Jupyter Shell. We understand that not all people have the Python and Gurobi on their computer, so running the code might be difficult. It is a great limitation of our programming in re-usability. In the future, it could be more efficient if there is an interface that the users can enter their data and get the schedule out in a file that they can locate.

Last but not least, we made the soft constraint a hard one as we changed the time slot to 1.5 hours. In "Scheduling the Italian National Volleyball Tournament", we saw that we can add a soft constraint by adding an extra term in the objective function with a weight that is less than the weight of the main term which represents hard constraints [2]. When minimizing the objective function, the term with more weight will decrease the objective value more than the term with less weight. Therefore, the model will treat the hard constraint as a priority to minimize instead of treating both hard and soft constraints the same.

Conclusion

In this paper, we considered the scheduling problem for the Statistics Tutor and Study Center based on each tutor's available time and manager's preference. We constructed a mixed integer quadratic programming model to include several constraints. The Gurobi package in Python was able to solve our model within a relatively short amount of time after the time slot was changed from 30 minutes to 90 minutes. We were able to transform our solution to an actual schedule which can be used by our community partner. Finally, we evaluated our schedule by comparing it with the current schedule and the result showed that the variance of our schedule is approximately 10 times smaller than the variance of the current schedule. Thus, our model generated a more fair and balanced schedule.

Acknowledgements

For this project, we thank our community partner Sheridan Grant, Lead Tutor of Statistic and Study Center for the interview, suggestions and valuable data. We also thank Dr. Matthew Conroy for giving his time to listen to our project. Moreover, we greatly appreciate the help of our teacher Prof. Sara Billey and Nico Courts for giving us some suggestions about how to approach our goal and modify our code. We also thank the input of Nhi Ngo, Reva Kane and Wenjun Yang for their advises to improve the project.

References

[1] Caleb Z. White, Youngbae Lee, Yoonsoo Kim, Rekha Thomas, and Patrick Perkins. "Creating Weekly Timetables to Maximize Employee Preferences". 2004. Website: https://sites.math.washington.edu/ thomas/papers/MSCRevised.pdf.
[2] Guido Cocchi, Alessandro Galligari, Federica Picca Nicolino, Veronica Piccialli, Fabio Schoen, Marco Sciandrone. "Scheduling the Italian National Volleyball Tournament." Interfaces 48(3):271-284, 2018.
[3] Winston, Wayne. "Operations research: Applications and algorithms." Belmont, CA: Thomson/Brooks/Cole, Fourth Ed, 2004.