Math 480 - Course Project
Creating a schedule for the Week of Chaos

Derek Rhodehamel
Justin Zecak
Davis Zvejnieks

1 Abstract

MathILy-Er is an academic mathematics summer program in Oregon. Every summer in between regularly scheduled classes is an impromptu week-long set of classes decided by student interests. MathILy-Er needs a way to quickly create an assignment of students and teachers to classes, and classes to timeslots, so that are no scheduling conflicts. The organizer of this event, and our community partner, is Dr. Jonah Ostroff at the University of Washington.

2 Problem description

MathILy-Er is an academic Summer program for high school students. In the middle of regularly scheduled classes, there is the Week of Chaos. During the Week of Chaos a number of week-long classes are taught during some number of time slots (typically five). Each student is scheduled for a class during each of the time slots, and teachers are scheduled to instruct these classes. Before the Week of Chaos, students are asked to fill out a survey where they rate their levels of interest in the various classes on a scale of 0-3. The possible teachers for each class is also taken into account.

The program's staff must then create a Week of Chaos schedule based on student preferences and teacher eligibility. Our goal for this project is to create a method to schedule based on an artificial set of student and teacher preferences which maximizes that satisfaction of student preferences, such that students are never scheduled for the multiple classes during a given time slot, every student is assigned to a class during each time slot, and the teaching load is evenly distributed between the teachers.

3 Our approach

At root, the problem is to find an optimal 3-matching. Unfortunately, finding a 3-or-more-matching is proven to be NP-complete, so a polynomial time algorithm is not known at this time. A 3-matching is a problem in which the desired solution is a collection of paths between three disjoint sets that satisfy all given constraints. This type of problem can quickly grow out of proportion when the size of the sets increase. Thus we chose to formulate the problem as a linear program (LP), since this method is both flexible and relatively scalable.

The end result must contain all integer values, for example, we cannot assign a student to half of a class. Thus the problem is an integer programming (IP) problem, and we'll use an LP solution as a basis for our IP problem. To approach this problem we chose to implement the GNU Linear Programming Kit (GLPK) to solve the IP, as it is free to use with no license requirements or other restrictions.

Because there is no data yet for student preferences of 2016, we used randomized data fitting within the constraints and estimations of Dr. Ostroff. This is done to show the feasibility of using this approach for data sets larger than the 2015 data. While our approach is built on the requirements of MathILy-Er's Week of Chaos, our algorithm has been generalized to accept a variable amount for the scheduling requirements. This allows this method of scheduling to extend beyond just MathILy-Er's Week of Chaos.

3.1 Variables

Let x represent student assignment into n classes. y represents instructor assignment into n classes. z represents time slots assignment of n classes. We set up the variables to be adjustable depending on the requirements in different circumstances.

$$h \in \{1, 2, ..., l\}\text{ , where } l \text{ is the number of instructors}$$

$$i \in \{1, 2, ..., m\}\text{ , where } m \text{ is the number of students}$$

$$j \in \{1, 2, ..., n\}\text{ where } n \text{ is the number of classes}$$

$$t \in \{1, 2, ..., s\}\text{ where } s \text{ is the number of timeslots}$$

$$x_{ij} = 1 \text{ if student } i \text{ is assigned to class } j\text{, 0 otherwise}$$

$$y_{hj} = 1 \text{ if instructor } h \text{ is assigned to class } j\text{, 0 otherwise}$$

$$z_{tj} = 1 \text{ if class } j \text{ is assigned to timeslot } t\text{, 0 otherwise}$$

We also incorporate the student preference matrix P and instructor eligibility matrix E in calculations as we seek to maximize their preferences and eligibility.

$$P = (p_{ij})\text{, where } p_{ij} \text{ represents preference rating of student } i \text{ for class } j$$

$E = (e_{hj})$, where e_{hj} represents eligibility rating of instructor h for class j

$F = (f_{ij})$, where $f_{ij} = 1$ if student$i$ is forced to take for class j, 0 otherwise

For the sample problem of 2016, with 5 instructors, 24 students, 15 classes, and 5 timeslotes we let:

$$l = 5, m = 24, n = 15, s = 5$$

Our objective function is the sum of student ratings for their assigned classes plus the sum of teacher ratings for the classes they are assigned to teach.

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} p_{ij} + \sum_{h=1}^{l} \sum_{j=1}^{n} y_{ij} e_{ij}$$

We seek to maximize this function subject to the follow sets of constraints:

3.2 Classes

3.2.1 Classes Taken

Each student must be enrolled in 5 classes during the Week of Chaos.

$$\forall i \in \{1, 2, ..., m\}, \sum_{j=1}^{n} x_{ij} = 5$$

3.2.2 Class Size

Each class should have around 7 students at most. We broke this up into two constraints, by setting a class size minimum and maximum. We found that without setting a minimum, class distribution was uneven. Some classes had only two or three students, and this violated the needs of Dr. Ostroff.

$$\forall j \in \{1, 2, ..., n\}, \sum_{i=1}^{m} x_{ij} \leq 8$$

$$\forall j \in \{1, 2, ..., n\}, \sum_{i=1}^{m} x_{ij} \geq 5$$

3.2.3 Instructors per class limit

First, each class should only have one instructor. While the 2015 data showed that Dr. Ostroff taught a class with another instructor, we decided this would add unnecessary complexity for the algorithm. We suggest that, if another teacher is not scheduled at that time, he or she may co-teach.

$$\forall j \in \{1, 2, ..., n\}, \sum_{h=1}^{l} y_{hj} = 1$$

3.3 Instructor eligibility

Information is gathered to determine teacher eligibility. Typically, of the selected classes there is already an understanding of who will teach what class. Occasionally, multiple instructors are able to teach a class, and we treat these as a backup option. It is a hard constraint that a teacher is never scheduled to teach a class that they are not qualified to teach. Thus for each class we add constraints stating that unqualified teachers cannot teach that class.

Eligibility for instructors is represented by the matrix E. This can be a matrix containing only binary numbers, or integer or real numbers. Using non-binary numbers may be useful in ranking instructors indicating possible back up instructors.

$$\forall j \in \{1, 2, ..., n\}, \sum_{h=1}^{l} y_{hj} e_{hj} \geq 0$$

3.4 Overrides

An interesting aspect of our problem is that, even though the goal is to make a schedule that maximizes student preferences, instructors will frequently override student preferences. This is because students usually do not have a clear idea of what goes on in the various classes at the time when they take the survey. If the instructors feel that a student underrated a class that fits well with her general interests, or ought to be in a class because it addresses a gap in her skill set, they will simply place the student in the relevant class regardless of their preferences. Conversely, some students are barred from taking a class.

Overrides constraints are calculated on a per student basis, reducing the number of constraints.

$$\text{if } f_{ij} = 1, \text{ then } x_{ij} = 1$$

$$\text{if } f_{ij} = -1, \text{ then } x_{ij} = 0$$

3.5 Time constraints

Each day there is a fixed number of time slots during which classes can be scheduled. In our artificial data set we set this number of time slots to 5. Clearly we cannot schedule teachers to teach multiple classes during a single time-slot or schedule students to take multiple classes during a single slot. We formulated these constraints in the obvious way: the sum of classes assigned to each teacher during each time-slot must be less than or equal to 1, and for students this sum must be equal to one, since the students must take exactly one class during every time slot.

3.5.1 Number of classes per timeslot

This depends on the scheduling decided ahead of time by the program's organizers. With 15 classes and 5 timeslots, we create a constraint such that only 3 classes can occupy any given timeslot.

$$\forall t \in \{1, 2, ..., s\}, \sum_{j=1}^{n} z_{tj} = 3$$

3.5.2 Each class is assigned to only one timeslot

Each class should only occur once in the schedule. This ensures that the algorithm does not disregard the time conflicts.

$$\forall j \in \{1, 2, ..., n\}, \sum_{t=1}^{s} z_{tj} = 1$$

3.6 Student time conflicts

To prevent students being assigned to classes that occur in the same timeslot, we must add a large number of constraints per student. For any given selection of three classes (there are $\binom{15}{3} = 455$) the total of the binary variables representing timeslots and student enrollment must be less than 1 plus the number of simultaneous classes.

For example, consider timeslot 1, student 2:

$$t_{13} + t_{14} + t_{15} + x_{23} + x_{24} + x_{25} \leq 4$$

Student 2 can be enrolled in classes 2, 3 and 4 as long as either only one of those classes are scheduled for timeslot 1, or none of them are. The issue arises when the total is above 4, meaning there is a time conflict. This same inequality is added as a constraint for every three classes, for every timeslot, and for every student.

Let:

$$S_3 = \text{the set of all 3-size combinations of classes}$$

$$c \in S_3, c = \{j_1, j_2, j_3\}$$

$$\forall t \in \{1, 2, ..., s\}, \forall c \in S_3, \forall i \in \{1, 2, ..., m\} \sum_{b}^{3} z_{tj_b} + x_{ij_b} < 4$$

Because this creates 455 constraints for each student, in each timeslot, it adds a lot of time complexity. This method creates 54,600 constraints to the problem.

3.6.1 Instructor time conflicts

A similar method is used to prevent time conflicts for instructors. This adds 11,375 constraints.

$$\forall t \in \{1, 2, ..., s\}, \forall c \in S_3, \forall h \in \{1, 2, ..., l\} \sum_{b}^{3} z_{tj_b} + y_{ij_b} < 4$$

In the end this amounts to a very large number of constraints, and it still takes quite a while to find the optimal solution. Thankfully, given the nature of our problem it is not actually necessary to find the optimal solution, since the preference function we take as our objective function is essentially just a proxy for subjective and approximate measures of student and teacher preference. Therefore our model allows users to vary how long they want to let the linear program solver run, rather than waiting for the solver to run to completion. In a matter of minutes the user can compute a solution which is very good albeit mathematically suboptimal, but which will be optimal or close to optimal from the practical standpoint of students and schedulers.

To substantiate this claim we ran our model for just one minute using the preference data from 2015, and arrived at a schedule which was nearly identical to the actual schedule chosen by the MathILy-Er instructors. The few discrepancies are most likely due to the overrides for 2015, which we do not have access to.

It should be noted that if the user does want to compute the mathematically optimal schedule, this can be done in a matter of hours for small to medium groups of students. That is enough time to be inconvenient, but it is still an improvement over finding a matching of a graph or hyper-graph by hand.

4 Simplification

While this problem does not require a significant number of simplifications, some assumptions must be made about the data in order to satisfy our solutions. One simplification is that we must assume is that the preferences of teachers are evenly distributed over the total collection of classes to be taught. We also determined a maximum class size of 8 students, and minimum class size of 5 students, based on last years data. Additionally, our method doesn't determine if two or more teachers instruct a class, as was found in 2015. Our initial tests are also based on a small number of randomly generated student and teacher preferences. Thus, another simplification, is that computation times are based on these artificial ratings.

5 Mathematical Solution

Depending on the size of the data set, finding the absolute maximum student and teacher preference in the assignment of classes could take days. In our tests using randomized student preferences, teacher eligibility, and overrides we found that the largest total preference was found in 10 minutes. Letting the algorithm run longer, we found no improvement after this time. Given our tests, we recommend setting a time limit of one hour to find a suitable schedule.

The integrality gap is the ratio between the relaxed LP optimal solution and the IP solution. Something interesting to note is that the optimal solution of the IP without any time constraints, is the same value of the optimal LP solution. That means whatever the integrality gap is, it represents the ratio of satisfaction given the ideal schedule where every student receives his most preferred class.

We created many artificial datasets, but worked with one in particular to perform more testing on. The data generated for teacher eligibility, overrides, and student preferences are found in tables 1, 2, and 3 respectively. Teacher eligibility was created under the guidance that of the available classes, the staff have an idea of who will teach that class. Occasional backup teachers were generated, with lower eligibility scores. Overrides were generated with the assumption that a lower rated class is often misunderstood by students, and those classes were chosen for overrides. Student preferences were created with the restriction that no rating could be given out more than 5 times.

In this data set, we found that within 10 minutes, the algorithm arrived at a schedule with an IP solution with an integrality gap of 2.2%. That is, within 2.2% of a schedule where time conflicts are not considered. Allowing our algorithm to run for ten hours did not improve the solution, but narrowed the integrality gap to 2%, by finding a new upperbound of the solution.

We chose GLPK to solve the integer programming problem, not only for its lack of restrictions in public or commercial applications, but for its flexibility in setting solver parameters. Different optimization parameters were tested using the same artificial dataset for their speed in producing feasible solutions.

We found the following essential to produce the largest total preference with the fastest results:

- Branching by least-fractional value
- Backtracking by breadth-first search

We additionally ran tests with a varying number of overrides and different preference and eligibility ratings without finding much difference in results produced. All tests resulted in an integrality gap of less than 5% in an hour.

6 Results

We were able to find a viable solution to this problem using integer programming within 30 seconds and an ideal solution within 10 minutes. For a more complex problem, this is faster than the method Dr. Ostroff used in 2015, which took approximately 2 hours by hand. It should be noted that the ideal solution is not necessarily the optimal solution. After running the algorithm for 10 hours, the solution was not improved upon, but all possibilities were not tested. The solution for a randomly generated data set is presented below.

6.1 Teacher Assignments by Eligibility

Table 1 refers to the data generated for the sample data in 2016. For a total of five teachers, each had an eligibility score to teach the classes. It was assumed that there was an even distribution of classes each teacher could instruct. Note that when a cell is colored green it corresponds to that teacher being scheduled to teach that class. It's important to note, that not only was no teacher assigned to aclass he or she could not teach, but the most eligible teacher was assigned in all cases. This satisfies the constraint of each class being instructed by an eligible teacher.

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Teacher a	0	0	🟩 10	0	0	0	4	0	0	🟩 10	0	0	0	🟩 10	0
Teacher b	🟩 10	🟩 10	0	0	7	0	0	0	0	0	🟩 10	0	0	0	0
Teacher c	0	0	0	0	9	0	🟩 10	0	🟩 10	0	0	🟩 10	0	0	0
Teacher d	0	0	0	🟩 10	0	0	0	🟩 10	0	6	0	0	0	0	🟩 10
Teacher e	0	0	0	0	🟩 10	🟩 10	0	0	0	0	0	0	🟩 10	4	0

Table 1: Teacher eligibility, green color represents assignment to classes by the algorithm

6.2 Student Assignments

Student assignments to classes as a result of the LP solution are shown in table 3 on page 9. This table shows class assignment among the preference ratings for the given classes. Note that this is in accordance to the overrides shown in 2 on page 9. The classes students were assigned are predominantly their highest rated classes.

Class	Include Student(s)	Exclude Student(s)
1	A, C	K
2		K, R
3	A, L	G
10	C	
13	C	
14	P	

Table 2: Overrides generated for sample data

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Student A	0	0	2	3	1	3	2	1	3	0	3	0	2	3	2
Student B	0	0	2	3	3	1	3	3	1	2	1	0	2	0	2
Student C	0	1	0	1	3	2	1	3	2	1	2	1	3	0	0
Student D	0	3	0	1	2	0	1	0	3	0	2	2	3	1	3
Student E	3	2	1	2	3	1	0	1	0	3	3	1	0	2	2
Student F	1	2	2	1	0	0	3	1	0	3	3	2	3	0	1
Student G	1	3	0	0	0	1	2	3	3	0	3	2	1	1	2
Student H	0	2	1	0	3	3	2	3	3	1	2	1	2	0	3
Student I	1	0	3	3	2	1	3	2	3	0	2	3	1	1	2
Student J	3	2	2	3	1	2	0	0	1	3	2	1	1	3	1
Student K	2	1	1	2	1	0	0	1	0	2	3	0	3	2	3
Student L	2	1	2	2	1	3	1	2	2	1	3	0	0	0	0
Student M	1	2	2	0	2	0	1	0	1	3	2	3	0	1	1
Student N	0	0	2	1	0	3	2	2	2	0	0	1	3	1	3
Student O	1	3	2	0	2	2	1	0	0	1	3	1	3	0	2
Student P	3	1	2	2	1	2	0	1	1	0	3	1	0	2	0
Student Q	2	0	1	3	2	2	2	1	2	0	3	3	1	1	3
Student R	2	0	2	2	0	1	2	3	3	3	1	2	1	0	3
Student S	3	2	0	3	3	0	3	2	1	2	1	1	3	0	1
Student T	2	1	3	2	1	0	2	2	0	3	3	0	1	2	1
Student U	1	3	0	2	0	1	3	2	2	0	2	3	1	0	0
Student V	1	3	0	1	0	3	3	0	2	1	2	2	1	0	2
Student W	0	3	0	0	2	1	3	2	1	2	1	1	2	3	1
Student X	0	2	2	3	2	0	1	3	0	3	3	1	0	1	3

Table 3: Student preference rating, with final assignment in green, overrides in blue (inclusion) and red (exclusion)

6.3 Timeslots

Organizing the data by classes, with their respective teachers and students is the end result that Dr. Ostroff and the MathILy-Er staff would need for "the Week of Chaos." This is shown in table 4. Note that in each timeslot, a student is only assigned to one class, and likewise with teachers. This satisfies the constraint of scheduling.

Class	Timeslot	Teacher	Students
1	1	b	A,C,E,J,L,P,S,T
7	1	c	B,F,G,I,M,U,V,W
15	1	d	D,H,K,N,O,Q,R,X
8	2	d	B,C,G,H,L,R,S,X
12	2	c	D,F,I,M,O,Q,U,V
14	2	a	A,E,J,K,N,P,T,W
9	3	c	A,D,G,H,I,R,U,V
11	3	b	E,J,L,M,P,Q,T,X
13	3	e	B,C,F,K,N,O,S,W
4	4	d	B,D,I,J,Q,S,U,X
6	4	e	A,G,H,L,N,O,P,V
10	4	a	C,E,F,K,M,R,T,W
2	5	b	D,F,G,J,O,U,V,W
3	5	a	A,I,L,N,P,R,T,X
5	5	e	B,C,E,H,K,M,Q,S

Table 4: Class assignments, organized by timeslot

6.4 Student satisfaction

To verify that students were assigned classes that they rated the highest, the average rating of their preferences was compared to the preference of their assigned classes, as shown in 5 on page 11. The difference from their average rating and actual assignment shows that no student was unfairly penalized in the assignment. Student A has the lowest preference of assigned classes due to abundant overrides given.

Another test was computed to compare the assignment. Using the same set of data, the IP was solved with a modified objective function. Instead, only teacher eligibility was maximized. The total of net satisfaction in this case was -1.466 compared to the original 23.332.

Student	Average Preference	Preference of Assigned Classes	Net Satisfaction
A	1.667	2.2	0.533
B	1.533	2.8	1.267
C	1.333	1.6	0.267
D	1.4	2.4	1.0
E	1.6	2.6	1.0
F	1.467	2.6	1.133
G	1.467	2.6	1.133
H	1.733	2.8	1.067
I	1.8	2.8	1.0
J	1.667	2.6	0.933
K	1.4	2.2	0.8
L	1.333	2.4	1.067
M	1.267	2.2	0.933
N	1.333	2.6	1.267
O	1.4	2.4	1.0
P	1.267	2.4	1.133
Q	1.733	2.2	0.467
R	1.667	2.8	1.133
S	1.667	2.6	0.933
T	1.533	2.6	1.067
U	1.333	2.6	1.267
V	1.4	2.0	0.6
W	1.467	2.4	0.933
X	1.6	3.0	1.4

Table 5: Student satisfaction, average rating given compared to rating of classes assigned

7 Improvements

The exponential growth of time needed to find the optimal solution for this Linear Program is something that cannot be avoided. With an NP-Complete problem and the structure of LPs, we can only hope to improve the time required to find a viable solution. By determining a proper heuristic to prioritize possibly viable solutions would improve the run time of finding a solution within a certain timeframe. Another possible source of improvement would be a custom preference scale. Our current preferences were pre-determined by our community partner, however, if we were given adequate time we might be able to produce a preference scale which would lead to either more accurate or quicker run times when it comes to discovered solutions.

Additional improvements need to be made to meet some of the additional, soft constraints of the community partner. For instance, our community partner mentioned incorporating gender ratios to the class schedules which our solution does not currently address.

Another beneficial option would be to place teachers with as many unique students as possible, preferably so that each student and teacher is paired in at least one class. The ability to turn these additional constraints on or off would allow our program to explore several possible solutions for our community partner with relatively little effort on their part.

Finally any other optimization that could be made to our LP structure might be beneficial. If we could combine constraints that produce the same results or in some other way reduce our total number of constraints, we could speed up our program to make it more convenient when testing several data sets.

8 Conclusions

This project directly connected to the material learned in class, regarding the subject of Linear Programming. While more complex than those covered in the homework, the basic knowledge provided in class proved instrumental in guiding our decision making process. Originally we tried to formulate a solution using graph theory, finding matchings between a tripartite graph so find a possible solution. Upon further consideration however, we determined that the novelty provided by this approach was outweighed by the scalability and customization provided by Linear Programming. When researching similar problems many published papers advocated for the use of Linear Programming over graphical methods.

9 Verification

Dr. Ostroff reported that he will be using our IP method in the upcoming session of MathILy-Er.

10 Appendix

All of our code was done using SageMath. We are releasing the code for free use and modification under the GPL.

10.1 Global variables

These variables are used in both the random generation and creation of the schedules. The numbers set here reflect the predicted numbers for 2016.

```
    classNum = 15
    studNum = 24
```

```
tchNum = 5
timeSlotNum = 5
simClasses = 3
classSizeMax = 8
classSizeMin = 5
classesTaken = 5
classesTaughtMax = 4
```

10.2 Integer programming

```

#Set the solving time in minutes
solvingTime = 60

#Set the input data for the IP (Here it's using randomly generated data)
pref = gen_pref(studNum,classNum)
elig = gen_elig(classNum,tchNum)
override = gen_override(studNum,classNum, pref)

#Create the integer programming object, and set the variables.
p = MixedIntegerLinearProgram(solver="GLPK")

#Student class assignment
#Rows indicate students, columns indicate classes
#1 if a student is enrolled in a class, 0 otherwise
s = p.new_variable(binary=True)

#Teacher class assignment by eligibility,
#Rows indicate teachers, columns indicate classes
#1 if teacher is instructing the class, 0 otherwise
e = p.new_variable(binary=True)

#Set the objective function as the sum of student preference
#and teacher eligibility assignments
p.set_objective( sum( sum( s[i,j]*pref[i][j] for i in range(studNum)) for j
 for j in range(classNum) ))

#Each student is enrolled in set number of classes
for i in range(studNum):
    p.add_constraint( sum(s[i,j] for j in range(classNum)) == classesTaken)

#Set class size min and max
for j in range(classNum):
    p.add_constraint( sum(s[i,j] for i in range(studNum)) <= classSizeMax)
    p.add_constraint( sum(s[i,j] for i in range(studNum)) >= classSizeMin)

#Teachers can not instruct more classes than the maximum
```

```
    for h in range(tchNum):
       p.add_constraint( sum( e[h,j] for j in range(classNum)) <=
        classesTaughtMax)

    #Each class can not have more than 1 teacher
    for j in range(classNum):
        p.add_constraint( sum (e[h,j] for h in range(tchNum)) <= 1)

    #A teacher with 0 eligibility can never be assigned to a class
    for j in range(classNum):
        p.add_constraint( sum (e[h,j]*elig[h,j] for h in range(tchNum)) >= 0)

    #Set overrides
    for i in range(studNum):
        for j in range(classNum):
            if override[i,j] == 1:
                p.add_constraint(s[i,j] == 1)
            if override[i,j] == -1:
                p.add_constraint(s[i,j] == 0)

    #Add new variable to track classes in timeslots
    #Rows indicate timeslots, columns indicate classes
    #1 if class is in that timeslot, 0 otherwise
    t = p.new_variable(binary=True)

    #Set the number of classes per timeslot
    for k in range(timeSlotNum):
        p.add_constraint( sum(t[k,j] for j in range(classNum)) == simClasses)

    #Set the number of timeslots per class
    for j in range(classNum):
        p.add_constraint( sum( t[k,j] for k in range(timeSlotNum)) == 1)

    #Create a set of all combinations of classes of size simClassess
    mset = Set(range(classNum))
    w = Combinations(mset,simClasses)

    #Add constraints to prevent students being assigned to multiple classes
    #in the same timeslot
    for k in range(timeSlotNum):
        for l in w:
            for i in range(studNum):
            p.add_constraint( sum(t[k,l[n]] + s[i,l[n]] for n in range(leng))
              <= simClasses+1)

    #Do so likewise for teachers
    for k in range(timeSlotNum):
        for l in w:
            for h in range(tchNum):
                p.add_constraint( sum(t[k,l[n]] + e[h,l[n]] for n in
                 range(leng)) <= simClasses+1)

    #Set GLPK solver parameters for faster computation
    import sage.numerical.backends.glpk_backend as backend
    p.solver_parameter(backend.glp_simplex_or_intopt , backend.glp_intopt_only)
    p.solver_parameter("timelimit", solvingTime*60)
    p.solver_parameter("branching","GLP_BR_LFV")
    p.solver_parameter("backtracking","GLP_BT_BFS"

    #Solve the problem given the timelimit
    p.solve(log=3)

    #Get the variables from the solution
    timeslot = Matrix(ZZ,p.get_values(t))
    studentAssignment = Matrix(ZZ,p.get_values(s))
    teacherAssignment = Matrix(ZZ,p.get_values(e))
```

10.3 Random dataset generator

```
from sage.misc.randstate import random
import numpy
import itertools as it

set_random_seed()

#Randomly generate student preferences
#takes number of students and number of classes and returns
#a preference table
def gen_pref(studNum, classNum):

    tempRating = []

    data = []

    #Students can not give more than 5 classes the same rating
    ratings = [0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3]

    #Append ratings if  the number of classes is larger than 20
    while classNum - len(ratings) > 5:
        ratings.append(0)
        ratings.append(1)
        ratings.append(2)
        ratings.append(3)


    for i in range(studNum):

        #Copy the default list of available ratings and reset tempRating
        possibleRatings = list(ratings)
        tempRating=[]

            for j in range(classNum):

                #Randomly choose from available ratings
                r = (random())%(len(possibleRatings))

                #Append the rating from the random number
                tempRating.append(possibleRatings[r])

                #Remove the given rating from possible ratings
                possibleRatings.remove(possibleRatings[r])

            #Add student's rating to list of all ratings
            data.append(tempRating)
            #Create a matrix from the list of ratings
            prefTable = matrix(data)

        return prefTable

    #Randomly generates overrides
    #Takes number of students, class number, and preference table as parameters
    #Returns override matrix
    def gen_override(studNum,classNum,prefTable):

        theList = []

        #Create a list of the total rating for each class
        for j in range(classNum):
            theList.append( sum(prefTable[i][j] for i in range(studNum)) )

        #Choose a number of classes for overrides, between 2 and 3
        r = (random()%2)+2
        k = 0

        disliked = []

        #Find the most disliked classes r times
        while k < r:

            min = 200000
            mindex = 0

            #Find the most disliked course
            for i in range(len(theList)):
                if not i in disliked:
```

```
                if theList[i] < min:
                    min = theList[i]
                    mindex = i

        disliked.append(mindex)
        k = k + 1

    overrides = Matrix(ZZ,studNum,classNum)

    #Randomly assign overrides for the most disliked courses
    for j in range(len(disliked)):
        for i in range(studNum):
            r = random()%101
            if r < 10:
                overrides[i,disliked[j]] = 1
            if r > 96:
                overrides[i,disliked[j]] = -1


    #Create additional overrides with a lower frequency
    #This simulates overrides being used to suit a student's tastes
    for j in range(classNum):

        if j not in disliked:
            for i in range(studNum):
                r = random()%101
                if r < 2:
                    overrides[i,j] = -1
                if r > 94:
                    overrides[i,j] =  1


    return overrides

#Generates random teacher eligibility
#Takes number of classes, and number of teachers, returns eligibility table
def gen_elig(classNum,tchNum):

    eligTable=Matrix(ZZ,tchNum,classNum)

    #Match the assumption that the classes chosen
    #are evenly distributed among who can teach
    num = round(classNum/tchNum)+1

    #Create a list of each teacher represented num times
    teachLot = []
    for k in range(tchNum):
        for n in range(num):
            teachLot.append(k)

    #For each class randomly select an ideal teacher for that class
    #then remove an instance of that teacher from the list
    for j in range(classNum):
        r = (random())%(len(teachLot))
        eligTable[teachLot[r],j] = 10
        teachLot.remove(teachLot[r])

    #Randomly assign backup teachers with eligibility less than 10
    for j in range(len(teachLot)):
        r = random()%(classNum)
        elig = (random()%7)+3
        eligTable[j,r] = elig


    return eligTable
```

11 Acknowledgments

First off we would like to thank Professor Sara Billey for being understanding when our previous two projects did not end up working out. The stumbles suffered throughout the quarter made progress difficult and uncertain, but Professor Billey was very accommodating and made sure that we kept moving forward. We would also like to thank Dr. Jonah Ostroff for providing this interesting problem.

12 References

Chan, Yuk Hei. ”On Linear Programming Relaxations of Hypergraph Matching.” 18 Oct. 2009. Web.

Gunawan, Aldy, Kien Ming Ng, and Kim Leng Poh. ”Solving the Teacher Assignment-Course Scheduling Problem by a Hybrid Algorithm.” 07 Jan. 2014. Web.

Gunawan, Aldy, and Kien Ming Ng. ”A Genetic Algorithm for the Teacher Assignment Problem for a University in Indonesia.” 01 Mar. 2008. Web.

Gunawan, Aldy. ”Solving the Teacher Assignment Problem by Two Metaheuristics.” 01 Jan. 2011. Web.