Assigning Students to Groups Based on Preference and Traits

BY

Brendan McGuirk

BS in Computer Science, University of New Hampshire, 2019

THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Master of Science

in

Computer Science

May, 2020

This thesis/dissertation has been examined and approved in partial fulfillment of the requirements for the degree of Master of Science in Computer Science by:

Thesis Director, Wheeler Ruml
Professor of Computer Science

Marek Petrik
Assistant Professor of Computer Science

Christopher Bauer
Professor of Chemistry

On April 24, 2020

Original approval signatures are on file with the University of New Hampshire Graduate School.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Assigning Students to Groups Based on Preference and Traits

by

Brendan McGuirk

University of New Hampshire, May, 2020

The University of New Hampshire Chemistry Department has been organizing study groups for students called Peer Led Team Learning (PLTL) for years now. However, assigning hundreds of students into groups manually is a non-trivial task, and doesn't guarantee the best possible solution. Each student will have certain times that they can meet, and more specifically, certain times that they prefer to meet. Additionally, when assigning students to groups, the students' traits should be taken in to account (e.g. preventing having only one freshman in the group). In this thesis, we devise a mathematical formulation of this problem, and apply an integer linear program solver to it to optimize the results. This work was developed with frequent input and feedback from the users to ensure that it met their needs and built on their suggestions. We demonstrate that our method is sufficiently fast to be used for the current needs of the users, and can continue providing high quality results as the future needs of the department scales.

CHAPTER 1
INTRODUCTION

1.1 Motivation

At the University of New Hampshire (UNH), many classes offer extra resources for enhancing students' learning experience. For example, the UNH Chemistry department has helped over 8000 students taking general or organic chemistry by offering Peer Led Team Learning (PLTL), where student volunteers who have previously taken the course meet weekly with groups of about 6 students to work on challenging problems, or answer any questions the students may have. Each semester, more than 300 students between multiple classes with different lecturers and lecture times are organized by hand into groups of about 6 to 8 students, creating more than 35 PLTL groups.

Student leaders are picked to lead PLTL groups before the semester starts. Groups meet for an hour and twenty minutes, but the leaders are given their choice on when they want their group to meet. A list of these group times are compiled in to a Google survey which is sent out to students taking the general or organic chemistry courses via email in the beginning of the semester. In this survey, students are asked to put their name, email, gender, and year, and answer *preferred*, *possible*, or *impossible* for each group time. When a student answers this survey, their results are added to a comma separated value (CSV) file saved by Google. This file contains all of the results for every student, with name, email, gender, year, and each group time being the column headers. These results are available to the UNH Chemistry department, so they can manually sort the students in to groups. This process involves manually assigning students to groups in hopes of finding a combination

that satisfies the most student's preferences. There is no solution check that they can use to see if their combination of students is optimal, and refining their solution requires checking every student and seeing where switches can be made.

1.2 Problem Definition

In a perfect world, assigning students to these PLTL groups would be as simple as dividing up all of the students equally in to each group. Unfortunately, this would be of little benefit for the students, as the chance of a student not being able to attend their PLTL group at its assigned time is extremely high due to obligations such as class load. Assigning students to groups that meet at a time they can meet is important, but it would also be beneficial to give students their preferred meeting time. Students should be separated by what chemistry course they're taking, so PLTL groups can work on one subject. Additionally, students from different lectures for the same class should be in different PLTL groups, as sometimes professors will get behind or ahead, so both lectures might not be at the same point. Finally, the UNH Chemistry department might decide that they want an even distribution of traits such as year and gender, so as to not let any student feel left out, lesser than their peers, or uncomfortable to meet in their group. This now becomes a much more complex problem that takes hours to do by hand.

As the number of students and the number of PLTL groups increase, the number of combinations of students in groups increases. The number of different combinations is equal to the number of groups to the power of the number of students. In the case where 300 students need to go in to 35 groups, that means there are 35^{300} or $1.661 * 10^{463}$ different combinations. Finding which of these combinations is the *best* combination is a non-trivial task. To check every combination would be infeasible, as, assuming a computer could check one billion combinations per second, it would take it about 10^{443} years. In the field of computer science, this is called a combinatorial optimization problem.

For combinatorial optimization problems, in order to figure out what types of groups

make a *good* group, an objective function needs to be defined. This objective function is used to set penalties or rewards for certain combinations of groups. For example, there could be a penalty for a group that is too big, or a reward for giving a student their preferred class. Using these penalties and rewards, you can calculate a "score" for a combination. Finding the best "scoring" group is what is defined as the optimal combination. This objective function can easily be changed to create different types of groups. You can only estimate how much you reward or penalize a certain combination, so coming up with a good objective function is a challenge.

This creates a human aspect to the problem as well. The UNH Chemistry department needs to be able to use an application for any chemistry class, by inputting a file to the program, and having it output a combination of PLTL groups. This output would be a CSV file with all the students arranged in to meeting times. The application needs to be easy enough to not create confusion for the user, and fast enough to not make the user wait hours to get a good solution.

1.3 Outline

In this thesis, I will first talk about previous work to this problem specifically, and similar types of problems in Chapter 2. Next, I will discuss my approach to solving this problem in Chapter 3. In Chapter 4 I will show the results of my solution, and the performance of the program I made. In Chapter 5 I will talk in depth about the feedback I received from the UNH Chemistry members who used my program extensively, and discuss how their feedback improved the program.

CHAPTER 2
PREVIOUS WORK

Assigning things to groups is not a novel concept, and many different problems like this have been solved. The problem UNH Chemistry is facing, however, has some unique aspects to it that require a new method of solving. In order to assure that students are placed in to groups not only on their preferences, but also their traits, we need to consider more information from each person when creating these groups.

2.1 UNH Chemistry Problem

Previous undergraduate students created a computer program that attempts to solve this problem. Their implementation was part of their capstone Senior Project at UNH. Their approach used very generic solutions like hill climbing [1] and squeaky wheel optimization [2], because the exact requirements for grouping students was never finalized, and these approaches can easily adjust their settings to create different types of groups. The downside to these methods is that there is no guarantee that the solution they find is optimal. There are many problems that hill climbing and squeaky wheel optimization can run in to such as local optima [3], plateaus [4], and ridges [3], that prevents finding optimal solutions.

The UNH Chemistry department found that their program was not implemented well enough to use for their needs. They currently cannot get the program to run on their computers, because of a lack of instructions on how to install the JRE. This has forced the UNH Chemistry department to return to sorting students by hand. While this does accomplish the task, it creates problems for the department leaders who have to spend hours

creating these groups manually.

It is sufficient to say that this program did not succeed in solving UNH Chemistry's problem. The biggest reason the program didn't work is the distribution of the program. Professor Chris Bauer, the PLTL lead for UNH Chemistry, said that there were problems running their program on Mac vs. Windows. Also, in order to get the program on their machines, the file had to be shared between users, which is inefficient and impractical. These were significant problems that needed to be addressed in the new program.

2.2 Similar Problems

The problem of assigning students to groups has been attempted to be solved in a variety of different ways. There is a lot of research that has gone in to what makes an effective group [5, 6], and whether grouping students by ability, interest, or other categories affected performance and effectiveness of the groups. This is not the goal of our problem. This problem is strictly for finding different combinations of students in groups to optimize student preferences.

A similar type of problem is the problem of timetabling [7,8], in which classes are assigned to times such that teachers only teach one class at a certain time slot, and students can only be in one class at a certain time slot. The timetabling problem is a type of scheduling problem that is in the class of NP-Hard problems [9]. This problem differs from the problem of assigning students to groups because in the timetabling problem, students are being assigned to multiple classes, making sure that their class times are not overlapping. These problems are often solved using a linear integer programming model [10], which is a popular approach for these types of problems.

Another problem that is similar to assigning students to groups is the hospitals/residents (HR) problem [11] (also known as the college admissions problem [12]). It is a generalization of the stable marriage problem [13], where residents rank hospitals they prefer in a strict order, and hospitals rank residents they prefer in a strict order. The goal is to assign residents

to hospitals such that the amount of preferred matches is optimized. One difference between this problem and the problem of assigning students to groups is that the preference order for residents and hospitals must be in strict order, and residents can not say they prefer two hospitals equally.

There is an extension of the HR problem where the order of preference does not have to be in strict order, and there can be ties in the preferences. This extension of the problem is called HRT and is NP-Hard [12]. HRT is more similar to the problem of assigning students to groups than the basic HR problem, except that in both HR and HRT, hospitals also have preferences over residents. The problem of assigning students to groups is a special case of HRT where every hospital's preference list has all residents at a tied preference.

A one sided version of the HR problem is called the assignment problem [14]. In the assignment problem, agents are assigned to tasks with some cost depending on the agent-task pairing. The goal is to minimize the cost, while assigning as many tasks as possible to agents. The assignment problem is a fundamental combinatorial optimization problem. The equation for the assignment problem is as follows:

$$\begin{aligned} \text{Minimize: } & \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}x_{ij} \\ \text{Subject to: } & \sum_{i=1}^{N} x_{ij} = 1 \text{ for } j = 1, ..., N \\ & \sum_{j=1}^{N} x_{ij} = 1 \text{ for } i = 1, ..., N \\ & x_{ij} \in \{0, 1\} \end{aligned} \tag{2.1}$$

Where c_{ij} is the cost for assigning agent i to job j, and x_{ij} is whether or not agent i is assigned to job j, for N jobs and N agents.

The unbalanced assignment problem [15], where there are more tasks than agents, is a special case of the assigning students to groups problem, where there are multiple tasks for each spot in a group and group size limits are fixed. There is a polynomial time algorithm to

solve the assignment problem, called the Hungarian method [16], that solves the problem in $O(n^3)$ time. The unbalanced assignment problem can be reduced to the assignment problem by adding dummy agents until the number of agents is equal to the number of jobs. Then the Hungarian method can be applied to solve the problem in polynomial time.

The semi-assignment problem [17, 18] is a version of the assignment problem where jobs are grouped in to sets. This is more in line with the assigning students to groups problem, in that students are being assigned to groups instead of individual tasks. The semi-assignment problem can also be solved in polynomial time using the Hungarian method. The equation for the semi-assignment problem is as follows:

$$\begin{aligned} \text{Minimize: } & \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}x_{ij} \\ \text{Subject to: } & \sum_{i=1}^{N} x_{ij} \leq b_j \text{ for } j = 1, ..., N \\ & \sum_{j=1}^{N} x_{ij} = 1 \text{ for } i = 1, ..., N \\ & x_{ij} \in \{0, 1\} \end{aligned} \tag{2.2}$$

Where c_{ij} is the cost for assigning agent i to job j, x_{ij} is whether or not agent i is assigned to job j, and b_j is the limit on the number of agents that can be assigned to group j, for N jobs and N agents.

On the other side of difficulty, the generalized assignment problem [19] (GAP) is an NP-Hard problem [20] that is a generalization of the assignment problem. In the GAP, tasks are assigned to agents, but all agents have a budget such that the sum of the weight of all their tasks does not exceed their budget. The problem of assigning students to groups is a special case of the GAP where the weight of all students is one, and the budget for each group is

its size limit. The equation for GAP is as follows:

$$\begin{aligned} \text{Maximize: } & \sum_{i=1}^{M}\sum_{j=1}^{N} c_{ij}x_{ij} \\ \text{Subject to: } & \sum_{j=1}^{N} w_{ij}x_{ij} \leq b_i \text{ for } i = 1, ..., M \\ & \sum_{i=1}^{M} x_{ij} = 1 \text{ for } j = 1, ..., N \\ & x_{ij} \in \{0, 1\} \end{aligned} \tag{2.3}$$

Where c_{ij} is the cost for assigning job j to agent i, x_{ij} is whether or not job j is assigned to agent i, w_{ij} is the weight of assigning job j to agent i, and b_i is the budget an agent has, for M agents and N groups.

The GAP is too expressive to compare our problem to, because in our problem students always have the same weight. Additionally, our problem has more constraints than the semi-assignment problem, such as considering students' traits when assigning them to groups. Our problem is in between the NP-Hard GAP and the polynomial time semi-assignment problem, meaning that it is not obvious whether or not our problem is NP-Hard.

In a similar approach to the previous UNH undergraduate students' attempt at solving the problem of assigning students to groups, various heuristic methods can be used to find solutions to the GAP and similar problems. A popular heuristic method for solving this problem is Tabu Search [21], which uses basic hill climbing with the ability to escape local optima. Just like basic hill climbing, the results you get are not guaranteed to be optimal, and the quality of your results depends on where you start the search. However, the ability to escape local optima increases the chance of finding an optimal solution, or at least a better solution.

Genetic algorithms [22, 23] can also be used for this problem. Genetic algorithms treat assignments of students to groups as chromosomes, and at each iteration of search, it changes the weakest chromosomes to increase the objective function value. For similar scheduling

problems, genetic algorithms show to perform better than other local search methods [24]. Again, using this method does not guarantee finding an optimal solution. This is a problem across all local search methods, and is something we tried to avoid in our approach.

CHAPTER 3
APPROACH

Based on the previous work from past UNH undergraduate students, we knew that a new approach needed to be taken to meet all the requirements of the UNH Chemistry department. Our solution was to create a program that we called *Group Assign*.

3.1 Design

During the design stage of the program, we were able to sit down with Professor Chris Bauer and three PLTL leaders: Molly Hanlon, Ryan Dussault, and Haley D'Angelo. In this meeting, we made sure to understand their ideas on what made a good group, and what sorts of things they wanted the program to do. In this meeting, it was unanimously agreed that groups are best between 6-8 students, and that 10 or above and 4 or below was too big and small for group sizes.

Additionally, they told us that if a student puts impossible as their availability for a group, to never assign that student to that group. With this information, we also learned that in smaller classes, such as Organic Chemistry, where the number of groups is not enough to allow every student to be placed into a group, that students would have to be unassigned if they didn't have any groups to be placed into. This was the motivation for having an "Unassigned" section in the output.

A feature that they didn't deem necessary, but would help improve the quality of the results, was to take the students traits into account when assigning students to groups. This could include their gender and their class standing, and was mostly intended to not

single any student's identities out. We were told that since the majority of students in chemistry are female, it can be hard to avoid groups that have only one male in them. These males would often be uncomfortable in groups of only females, and would be less likely to participate. The same is true for class standings, such as a single freshman in a group with all upperclassmen. This became an optional feature in the program where the user can specify whether groupings should be made in consideration of multiple personal characteristics.

Lastly, the PLTL leaders urged that students with different professors for their lectures should be placed in different groups. They have had many problems in the past where professors get behind in lecture material compared to other professors teaching the same class, and having PLTL groups with mixed professors would lead to confusion and division among the group. When the PLTL leaders organize the groups by hand, this is one of the things that they pay attention to the most, so we agreed that this feature should be included in our program.

Of course, the most important thing to consider was the algorithm for sorting students into groups. We considered many approaches, but ended up focusing on integer linear programming.

3.2 Integer Linear Programming

A mathematical approach to solving this problem is using Integer Linear Programming (ILP). An ILP is a program that defines a problem mathematically with decision variables. If we have i students, j groups, and k lectures, we can have a decision variable for each student in each group (x_{ij} is whether or not student i is in group j), and a decision variable for each group and lecture (g_{jk} is whether or not group j allows students from professor k). Knowing this, we can define mathematical equations for scoring our objective function.

The objective function is an equation that defines what a good solution is. The objective function is created with decision variables and coefficients, and the solver tries different combinations of values for the decision variables until it finds either a maximum or minimum

value. This approach will always find the optimal solution, however the run time will vary depending on the number of decision variables and the amount of constraints.

In order to set limitation on what the decision variables can be set to, constraints can be defined on the solver. This is important to prevent situations such as one student being assigned to multiple groups. There were many constraints required to fit the problem, which are talked about more in depth in Section 3.2.1.

We also considered using heuristic search methods to solve this problem such as Tabu Search [25], Greedy Randomized Adaptive Search Procedure [25], and Large Neighborhood Search [25, 26]. These methods were proposed as possible solutions should the ILP take too long to find a solution. Our results in Chapter 4 prove that the ILP is fast enough for any realistic situation that could occur for UNH Chemistry, as well as future situations should PLTL expand.

There were a few choices to use for ILP solvers, but eventually we decided on a solver called Gurobi [27]. Initially, we tried a free solver by Google, called Google OR-Tools [28]. This solver worked for problems where the number of students and the number of groups were small, however it did not scale well, and when putting harder problems to the test, the solver took too long to find a solution, if it could find any before it timed out. Since the program might run on the UNH server Agate, which has a five minute timeout, this solver would not be acceptable.

The Gurobi Optimizer advertises itself as one of the best optimization solvers for ILPs. We knew that this solver would provide speedup from Google's free solver. An additional benefit of using Gurobi is that they offer free licenses for academic users. We were quickly able to request a license and get Gurobi working. The same problems solved using Google OR-Tools took on average 22.7 times longer than Gurobi as shown in Figure 3.1.

3.2.1 ILP Formulation

In this section, we will discuss the formulation of our ILP.

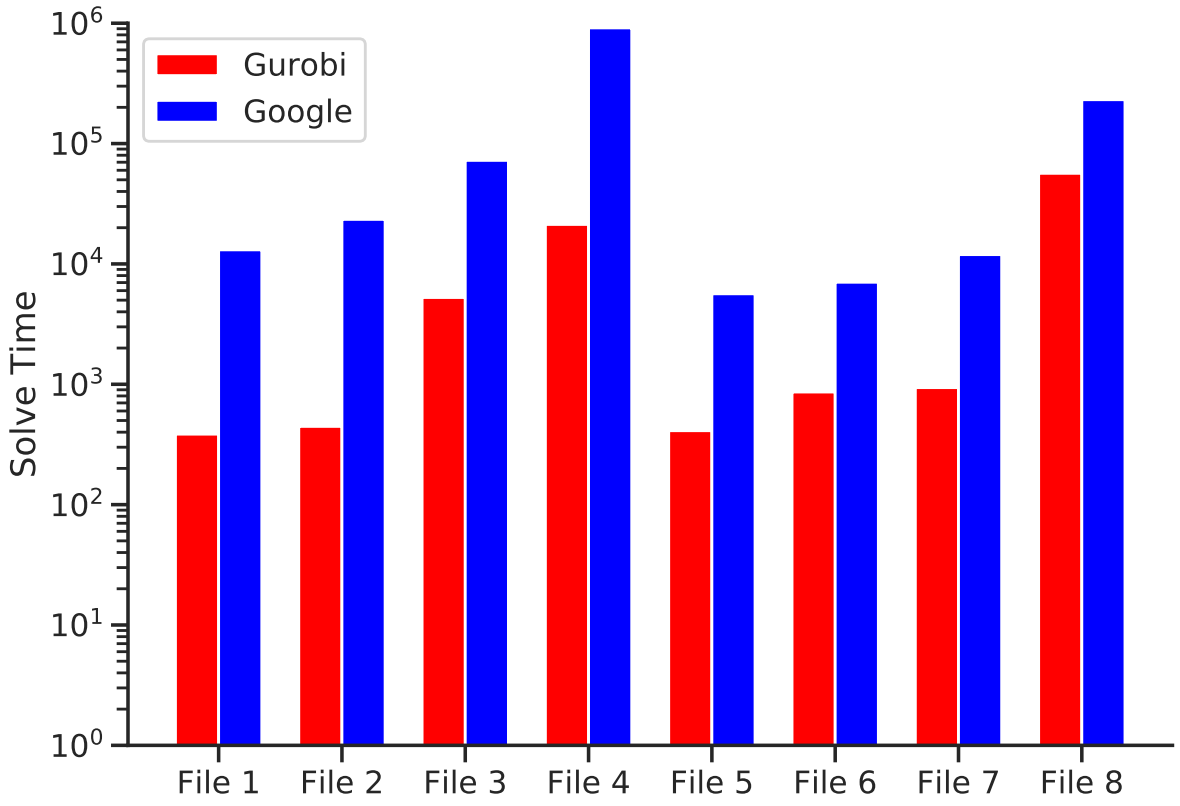


Figure 3.1: Run Times from Google-OR Tools and Gurobi

Solver Variables

In order to formulate the ILP to solve our problem, additional variables needed to be used. As described above, each student had one decision variable for each group. This decision variable could be either 1 if the student was assigned to that group, or 0 if the student was not assigned to that group. This variable will be x_{ij}, for whether or not student i is in group j.

Since groups could potentially have no students assigned to them, another decision variable was made for each group, which could be either 1 or 0, depending on whether the group was enabled or not respectively. If a group's decision variable is 0, that means that no students could be assigned to that group. This variable will be $enabled_j$ for whether or not group j is enabled.

To implement a limit on group size, two integer variables were created for each group; one for the minimum size and one for the maximum size. Since the PLTL leaders said that they wanted groups to be between 6 and 8 people, we needed two variables to define group size, so that multiple groups could have different sizes within those numbers with no penalty. The variables for minimum and maximum group size will be min_j and max_j respectively for group j. These variables are integer variables, which is different from the 0-1 decision variables in that they can be any integer number. Having these variables is necessary to encourage groups to be between the preferred minimum and maximum group sizes, without going past the absolute minimum and maximum group sizes. Since we want the user to be able to specify their preferred and absolute minimum and maximum group sizes, the minimum group size variable is constrained to be between the absolute minimum and preferred minimum, and the maximum group size variable is constrained to be between the preferred maximum and the absolute maximum.

Additionally, each group got a decision variable for each professor. This decision variable could be either 1 or 0, depending on whether that group allowed students who had that professor for their class assigned to that group or not respectively. This solves the problem of keeping students with different professors into the different groups. This variable will be g_{jk}, for whether or not group j allows students from professor k.

To have students' traits affect group assignments, each group has decision variables for enabling males, females, freshmen, sophomores, juniors, and seniors, for a total of six decision variables per group. These decision variables could be either 1 or 0, depending on whether the group allows students with that trait or not respectively. These decision variables will be $enabled_{jt}$ for whether or not group j allows students with trait t, where t is either male, female, freshman, sophomore, junior, or senior.

In the case where PLTL leaders want to avoid singling out traits in their groups (e.g. groups with only one male), each group has decision variables for whether or not there is only one member assigned to the group with a certain trait. Each group has a decision variable

for each trait, for a total of six decision variables per group. These decision variables are set by constraints to ensure that they are 1 when a student in the group is singled out by a certain trait, and 0 otherwise. These decision variables will be $single_{jt}$ for whether or not group j has a student singled out by trait t, where t is either male, female, freshman, sophomore, junior, or senior.

In the case where PLTL leaders want to avoid groups consisting entirely of a certain trait (e.g. all female groups, all freshmen groups, etc.), each group has decision variables for whether or not all members of a group belong to the same trait. Each group has a decision variable for each trait, for a total of six decision variables per group. These decision variables are set by constraints to ensure that they are 1 when all group members share the same trait, and 0 otherwise. These decision variables will be $shared_{jt}$ for whether or not all members of group j share the same trait t, where t is either male, female, freshman, sophomore, junior, or senior.

Since every group gets one decision variable for each student, every group gets one decision for each professor, every group has three decision variables for group size (enabled, min, and max), and every group has 18 decision variables for traits (enabled, singled, and shared *per trait*), the total number of variables the solver is using is

$$nm + np + 3n + 18n = n(m + p + 21) \tag{3.1}$$

where n is the number of groups, m is the number of students, and p is the number of professors.

Objective Function

The objective function in our program is defined to tell the solver to favor group assignments that met the criteria described by the PLTL leaders. The objective function is made of many separate equations that together capture what groups should resemble. In our problem, the

program is trying to be minimize the value of the objective function.

The full objective function is as follows:

$$\text{Minimize: } \mathit{Preferences} + \mathit{Sizes} + \mathit{SingleTrait} + \mathit{SharedTrait} \tag{3.2}$$

The first part of the objective function aims to give students their desired groups.

$$\mathit{Preferences} : \sum_{j=1}^{n} \sum_{i=1}^{m} (p_{ij} - v) * x_{ij} \tag{3.3}$$

Where p_{ij} is the cost for assigning student i to group j, v is the penalty for leaving a student unassigned, and x_{ij} is whether or not student i is assigned to group j. Subtracting the penalty for leaving a student unassigned from the cost of assigning a student to a group encourages the program to place students in to groups since the penalty for assigning a student to a preferred group is zero.

The second part of the objective function is used to create groups between the users' preferred minimum and maximum sizes.

$$\mathit{Sizes} : \sum_{j=1}^{n} (-a * min_j) + (b * max_j) \tag{3.4}$$

Where a is the penalty for decreasing the minimum group size min_j, and b is the penalty for increasing the maximum group size max_j. This part of the objective function keeps the minimum group size variable high by multiplying it by the negative penalty for decreasing the group size minimum, and to keep the maximum group size variable low by multiplying it by the penalty for increasing the group size maximum.

The two variables min_j and max_j work together to keep group sizes between the preferred values. This is shown in Figure 3.2, where the absolute minimum and maximum groups sizes are 4 and 10 respectively, the preferred minimum and maximum group sizes are 6 and 8 respectively, $a = 10$, and $b = 3$. Since the program is trying to minimize the value of

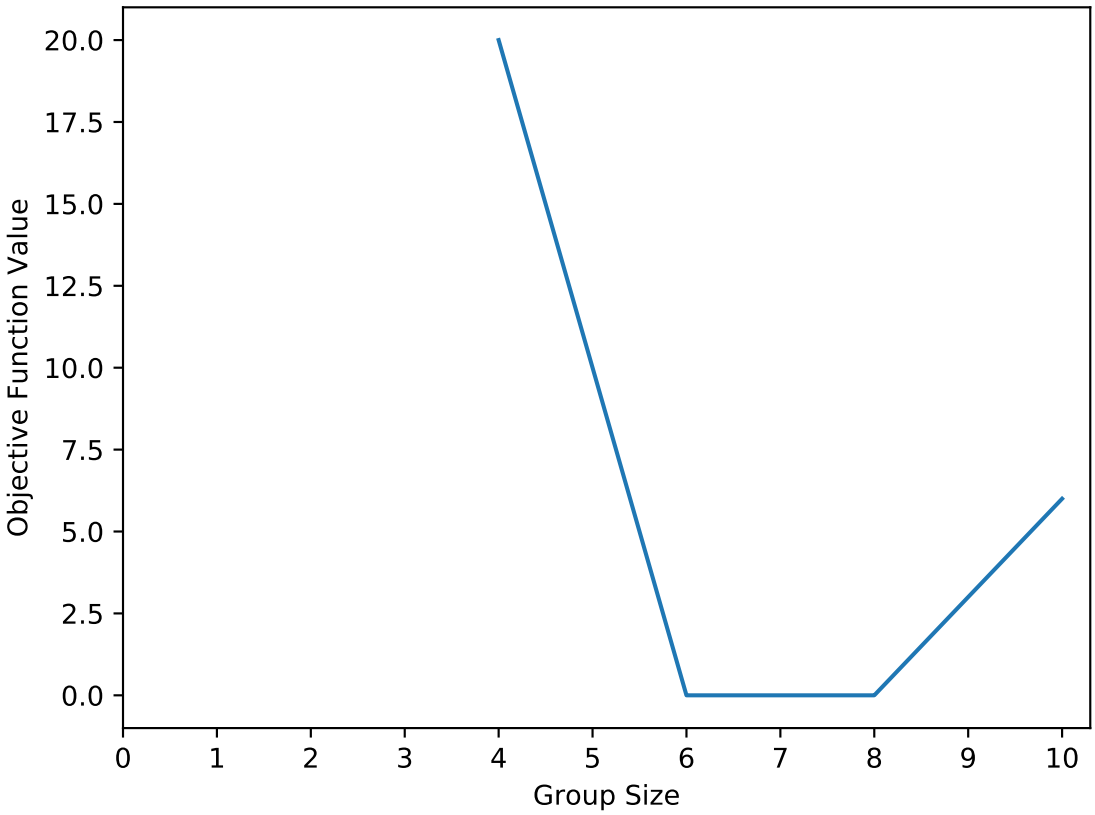


Figure 3.2: Group Size Variables Affecting Objective Value Score

the objective function, it will prefer to create groups between the preferred minimum and maximum group sizes, where the objective function value is lowest. Constraints are set in place to ensure that these variables reflect the actual number of students assigned to the group, which are described in the next section.

The last two parts of the objective function changes how groups are made based on students' traits. The first equation penalizes groups that have a student who is singled out by their trait.

$$SingleTrait : \sum_{j=1}^{n} c_t * single_{jt} \text{ for trait } t \in Traits \tag{3.5}$$

Where c_t is the penalty for having a single student in group j with trait t. The set of traits includes males, females, freshmen, sophomores, juniors, and seniors. Several PLTL leaders

have stated that when students are singled out by gender, they are often less willing to participate in the group discussion because they feel uncomfortable.

The last equation penalizes groups in which all students share a similar trait.

$$SharedTrait : \sum_{j=1}^{n} d_t * shared_{jt} \text{ for trait } t \in Traits \tag{3.6}$$

Where d_t is the penalty for every student in group j sharing trait j. The set of traits is the same as in Equation 3.5. These two equations are an important part of creating groups that the user wants.

The benefit of having different penalties for each trait is that it gives more control over how groups are made. The majority of students in PLTL are female freshmen, so limiting groups that are entirely female freshman might not be as important as preventing singling out males. This allows for the most control over how students are assigned to groups.

Constraints

In order to limit which decision variables can be set in combination with others, constraints are put on the system. These constraints are mathematical equations formed from constants and solver variables. Our solver has six different constraints, made up of many different equations.

The first constraint is limiting students to be assigned to either one group or no groups. In the following equation, we have m students and n groups. The sum of all decision variables for each student must be less than or equal to 1.

$$AtMostOneGroupPerStudent : \sum_{j=1}^{n} x_{ij} \leq 1 \text{ for } i = 1, ..., m \tag{3.7}$$

For constraining the group size variables, three constraints were used for each group. To constrain enabling groups, for each group, the number of students in that group must be greater than or equal to the enabled variable and less than or equal to the hard maximum

group size multiplied by the enabled variable, where the hard maximum group size is specified as part of the input.

$$SetGroupEnabled : enabled_j \leq \sum_{i=1}^{m} x_{ij} \leq hardMax * enabled_j \text{ for } j = 1, ..., n \tag{3.8}$$

To limit the group sizes to between a preferred minimum and a preferred maximum, the minimum and maximum group size variables are constrained to be less than or equal to the number of students in the group, and greater than or equal to the number of students in the group respectively. The objective function is trying to limit the minimum and maximum group sizes from moving past their preferred sizes, but the constraint makes sure if a group has more or less students than the preferred size, that the group size variables change accordingly.

For the minimum group size, with group size absolute limits hardMin and hardMax, and preferred limits prefMin and prefMax, we limit the minimum group size variable with the following equation.

$$GroupMin :$$
$$-prefMin \leq (\sum_{i=1}^{m} x_{ij}) - min_j - (prefMin * enabled_j) \leq hardMax \tag{3.9}$$
$$\text{for } j = 1, ..., n$$

For the maximum group size, the same approach is used.

$$GroupMax :$$
$$-hardMax \leq (\sum_{i=1}^{m} x_{ij}) - max_j + (prefMax * enabled_j) \leq prefMax \tag{3.10}$$
$$\text{for } j = 1, ..., n$$

These two equations make sure that the number of students in the group is in between the minimum and maximum group size variable values.

The PLTL leaders made it very clear that it is important that students with different professors for their lectures do not end up in the same PLTL group. To ensure this, we used two constraints. The first constraint limits the number of professors a group can be associated with to 1. For every group j in n groups and every professor k in p professors, the sum of all the professor decision variables g_{jk} should equal 1.

$$OneProfessorPerGroup : \sum_{k=1}^{p} g_{jk} = 1 \text{ for } j = 1, ..., n \tag{3.11}$$

With the previous equation, we ensure that each group only is associated with one professor. To make sure that all students in a group share the same professor, for each group j and for each professor k, we sum the decision variables for that group of every student that has that professor (x_{ij} where student i has professor k). This equation should be less than or equal to the hard maximum group size multiplied by whether or not group j is associated with professor k.

$$\begin{gathered} StudentsShareProfessor : \\ \sum_{i \in M_k} x_{ij} \leq hardMax * g_{jk} \\ \text{for } j = 1, ..., n \text{ and } k = 1, ..., p \\ \text{where } M_k \text{ is the set of students with professor } k \end{gathered} \tag{3.12}$$

To give groups more flexibility, and increase the chances of successfully assigning every student into a group, we allow for a group leader to list multiple potential times they are available to host a group as part of the input. It is rare that a group leader will host more than one group per semester, so we want to limit the number of groups one person can host to one or zero. We can check this by adding all of the enabled decision variables for groups

led by each group leader.

$$OneLeaderPerGroup:$$

$$0 \leq \sum_{j \in N_l} enabled_j \leq 1 \tag{3.13}$$

$$\text{for } l = 1, ..., L \text{ where } N_l \text{ is the set of groups lead by } l$$

Using traits as part of the objective function required constraints to set decision variables based on group conditions. The decision variables we are trying to set are $single_{jt}$ and $shared_{jt}$. To set both of these, we need to use $enabled_{jt}$, to determine whether or not that group allows members with a certain trait. To set $enabled_{jt}$, we limit the sum of all people with trait t to be between $enabled_{jt}$ and $hardMax * enabled_{jt}$.

$$SetTraitEnabled:$$

$$enabled_{jt} \leq \sum_{i \in M_t} x_{ij} \leq hardMax * enabled_{jt} \tag{3.14}$$

$$\text{for } j = 1, ..., n \text{ where } M_t \text{ is the set of students with trait } t$$

Setting the decision variable $single_{jt}$ is done by finding the number of students in group j with trait t, and if there is one student with that trait, and the trait is enabled for the group, forcing $single_{jt}$ to be one.

$$SetSingleTrait:$$

$$0 \leq \sum_{i \in M_t} x_{ij} + (single_{jt} - 2 * enabled_{jt}) \leq hardMax \tag{3.15}$$

$$\text{for } j = 1, ..., n \text{ where } M_t \text{ is the set of students with trait } t$$

The same approach is used for setting $shared_{jt}$, where if the sum of all students in group j minus the sum of students in group j with trait t is zero and the trait is enabled for the

group, $shared_{jt}$ is forced to be one.

$$SetSharedTrait:$$

$$0 \leq \sum_{i=1}^{m} x_{ij} - \sum_{i \in M_t} x_{ij} + (shared_{jt} - enabled_{jt}) \leq hardMax \tag{3.16}$$

$$\text{for } j = 1, ..., n \text{ where } M_t \text{ is the set of students with trait } t$$

The full problem of assigning students to groups can be defined as follows:

$$\begin{aligned} \text{Minimize: } & Preferences + Sizes + SingleTrait + SharedTrait \\ \text{Subject to: } & AtMostOneGroupPerStudent \\ & SetGroupEnabled \\ & GroupMin \\ & GroupMax \\ & OneProfessorPerGroup \\ & StudentsShareProfessor \\ & OneLeaderPerGroup \\ & SetTraitEnabled \\ & SetSingleTrait \\ & SetSharedTrait \end{aligned} \tag{3.17}$$

3.3 Input

In recent years, the PLTL leaders would send surveys to all of the students in the class to get their availability for PLTL times. This survey was sent out via Google Forms, and students had about a week to respond. In the survey, students were asked basic questions, such as name, email, and professor, and for each PLTL time, the student would put their availability as either *preferred*, *possible*, or *impossible*. Google Forms automates the collection of this

data and combines all students' responses in a Google Sheet, where it can be exported as a CSV file. The PLTL leaders liked the ease and efficiency that Google Sheets provides, and recommended that they want to continue using it. We decided that since the data the PLTL leaders collect comes in a CSV format, that our input should also be a CSV file.

The input to the program needed to be able to cover all of the students' preferences and information as well as all of the parameters for the solver. We wanted to make it as easy as possible for the user to gather all their data to send to the program. To accomplish this, we came up with two different input formats for the user to use; a simple version, and a full version.

In the simple version of the input, the user can send a CSV file containing only the students' information. The first row of the CSV file contains column headers. The required column headers are the student's first name, last name, email, gender, year, professor name, and notes, followed by group times. The group time columns must be unique, and the program will not accept repeats. Each row following the header row is information about the students.

The only required columns to fill out are first name, last name, and email. The other columns may remain blank, as they are only used for customizing groups in the full version of the input. The professor name column may be used in the case where there are multiple professors for that class, and students with different professors should not be in the same lecture. The notes column is only used to keep track of notes that students may have told the PLTL leaders, and will not be used while creating groups. These columns seemed to appear frequently in Google Forms that different PLTL leaders sent out, but not all of the optional columns were asked in every survey, so making these columns optional prevents the PLTL leaders from needing to add extra information they might not know to the input.

For each group time column, the only possible options are "Preferred", "Possible", and "Impossible". This corresponds with what the PLTL leaders told us they ask their students on the Google Form, so these columns would not have to be edited at all from the data they

collect.

The goal of this simple version of the output was to limit the amount of information the PLTL leaders need to add. While they might need to remove columns if they ask extra questions in their survey, this simple version combines the common questions asked across all surveys, reducing the chance that information will be missing.

The groups created by the program will use the default parameters set in the program. We came up with default values for these parameters based on the feedback from the initial meeting with Professor Bauer and several PLTL leaders. The default parameters set the group size limits between 4 and 10, with preferred group sizes between 6 and 8. The penalties for increasing or decreasing the group size past the preferred sizes are 3 and 10 respectively. One of the reasons for having a higher penalty for decreasing the group sizes is because students will often drop the class, so small groups will become even smaller. The penalty for giving a student a group time that is possible but not preferred is 2, and the penalty for leaving a student unassigned is 50. These numbers reflect how important it is to assign students, and how while it's better to give students their preferred choice, a possible choice is almost as good.

If the PLTL leader wants more control over what groups are being created by the program, they can send the full version of input in. The full version of input still uses the CSV format, but it is broken up into three sections: groups, parameters, and students.

The first section is where the user can define groups. This section is important in cases where two group leaders want to lead different groups at the same time. Groups are defined by adding a row that starts with "∼∼Group". We chose this header because it could be easily distinguished from students names (by starting with "∼∼"), and because we wanted to define groups in a way that were distinctly different from other rows in this section. The group header is followed by the group leader's name, email, and group meeting time. Repeated leader names, emails, and group meeting times may appear across multiple groups, but there may not be any repeated combined group leader names and group meeting times.

The group meeting times do not need to be in any format, as long as there is a corresponding column for that group time in section three.

At the end of the first section, a row needs to be added that starts with "∼∼Unassigned". Again, the "∼∼" is used to distinguish this row from the students. This section is used in the output. One important feature of the input is to lock students into groups. This may be useful if a PLTL leader is contacted by a student who tells them that they need to be placed in a certain group for any reason. It is also important for hand making groups, and sending the input to the program to assign the remaining students. Students are considered locked into a group if a row of their information is added beneath a group. This row must contain the student's name, followed by the student's email. An example of the first section of the input file can be found in the Appendix in Figure A.1.

The second section is where the student can edit the default parameters for the program. All parameters are required in the input. These parameters include the parameters used in the simple version, as well as penalties for having groups with only one male, female, freshman, sophomore, junior, or senior, and penalties for having groups with all males, females, freshmen, sophomores, juniors, or seniors. An example of the second section of the input file can be found in the Appendix in Figure A.2.

The last section is for student information. This section is exactly the same as the simple version. In this full version, the gender and year columns are recommended to fill out, so the groups can be created with the students' traits in mind. Also, if a group meeting time appears in the first section, it must appear in this section, and vice versa. Again, this section can not contain any duplicate meeting times, so if multiple groups are listed with the same time, they will both use the students' preference from the same column in this section. An example of the last section of the input file can be found in the Appendix in Figures A.3 and A.4.

3.4 Output

When considering what the output should be, we also were thinking about what would be easiest for the PLTL leaders. A common problem that occurs every semester is that students will sign up for their chemistry class, get placed in their PLTL group, and then drop the class. Additionally, students may add their chemistry class late and wish to be placed in a PLTL group. This causes a lot of headache for the PLTL leaders, when they need to insert people into groups.

To make the process of rearranging the input file to lock students into groups who have already been placed into groups easier, we decided that the output should also be able to be used as input. After the program arranges students into groups, the output copies the information from the input, but adds student rows underneath each group to show which students were assigned to each group.

This output can be sent back to the program as input, if the PLTL leaders wish to remove students from groups, or add students to the last section of the input. If the output is sent directly back to the program, nothing will change because all students have been assigned to groups. PLTL leaders can delete a row containing student information from the first section, essentially removing the student from that group, and allowing them to be reassigned.

When a user sends in a simple version of the input, the output will be the full version, so group assignments can appear and so the PLTL leaders can see which parameters were used.

To make the PLTL leader easier understand which groups caused the most problems, above each group is a comment in the file that shows how much penalty that group accrued.

3.5 Application

We thought of many different ways to create the application for this program. Our first thought was to make a web application where users could upload files, and the output would

either appear on the screen, or be downloaded as a file. We quickly realized that this would be a lot of work, and wasn't as important as focusing on the ILP program. There are also many problems that can occur when developing with different web browsers and devices in mind.

Instead, we decided that an email application would be sufficient. Our idea was to have a front end program regularly be checking a third party email service for new emails. A user could send data to the program by sending their input as an attachment in an email to our email address. To accomplish this, we created a new Gmail account, and had our front end program regularly checking for new emails. This front end program was written in Python.

When a new email is received, the program first checks the email to ensure that the attached file is formatted correctly. If no CSV file is attached to the email, or multiple CSV files are attached, the program sends an email back to the sender with a user guide and a template file attached that they can fill out. If there is a CSV file attached, then the program reads it to make sure that it is formatted correctly. If there are formatting errors in the file, such as missing parts or invalid values, the program sends an email back to the sender with a user guide, a template file, and the file they sent attached. The reply email also lists all the errors in the file, with specific errors message and line numbers so the user can fix their mistakes and resend their file.

If the file is formatted correctly, the front end program calls the ILP solver with the file as input. The ILP solver was created by Gurobi, but to create the ILP model, we wrote a program in C++. The ILP solver takes the model, creates group assignments, and outputs the results. The results are saved in a file, which the front end program sends back to the user in a reply email.

We decided to use a third party email service to increase the longevity of the program. This should limit the chance of a UNH email being disabled, and prevent the email used by this program from receiving spam emails from within the network. Another way we plan on increasing longevity of the program is hosting it on either UNH server Agate or Professor

Wheeler Ruml's personal server Katsura. The program has been tested thoroughly on Mica, a clone of Agate that allows for longer CPU time, but should have no problem running on Katsura. Professor Ruml Wheeler offered to start the program for the UNH Chemistry department every semester they need it.

We received very positive feedback from several PLTL leaders about running the program through an email service, so we did not feel a need to create a web application, or any other form of service. One user said "I love the gmail format, and the details about what specific parts of the program is very helpful for problem solving. Its extremely quick which is amazing. Overall very usable and easy to work with."

While the program will be hosted on a UNH server, the email interface does not limit incoming emails strictly to UNH emails. Therefore, anyone who is aware of the email can use it. This allows the intended user to use any email service they want, and doesn't restrict them to using their UNH Outlook email.

3.6 System Requirements

To run *Group Assign* requires the following system requirements:

- Linux based OS (Ubuntu 18.04.4 LTS used)
- An installation of Python 3 (Version 3.6.9 used)
- GNU Make (Version 4.1 used)
- C++ compiler: g++ (Version 7.5.0 used)
- Pip3 (Version 9.0.1 used)
- A Gurobi license
- An internet connection[1]

[1]Internet is only required when using the email interface. Using the ILP solver on its own does **not** require an internet connection

Installation is done by running a shell script provided with the program. This code can be found in Appendix B.

CHAPTER 4

RESULTS

4.1 ILP Performance

To test the performance of the ILP, we used both real data from past years students, as well as synthetic data, based on statistical analysis of the past years' data. We received 14 files from Professor Chris Bauer containing student responses from multiple different chemistry classes across multiple years. These files are typical of current years data, although, Professor Chris Bauer did mention that this year had the most students that signed up for PLTL.

Class/Year	# Students	# Groups	# Professors	Time Taken
CHEM 411 Fall 2019	41	4	1	20.9 ms
ORGO 545 Fall 2016	132	5	1	21.3 ms
ORGO 652 Spring 2019	126	10	1	29.3 ms
ORGO 651 Fall 2018	180	11	1	33.4 ms
CHEM 651 Fall 2019	124	18	1	48.7 ms
Winans Spring 2018	75	23	1	64.4 ms
Buell Spring 2018	69	23	1	65.0 ms
Winans Fall 2016	188	26	1	74.6 ms
Bauer Fall 2017	78	27	1	77.1 ms
Bauer Spring 2018	64	23	1	177.3 ms
Bauer Fall 2016	76	26	1	217.1 ms
CHEM 403 Fall 2019	304	21	2	406.8 ms
CHEM 403 Fall 2018	197	25	2	847.3 ms
CHEM 403 Fall 2016	264	36	2	887.3 ms

Table 4.1: Past Years Data Performance Results

We did have to make modifications to the files in order to match the formatting of the input, but no modifications to the students data were made. The run times for the past years data is shown in 4.1. All files took less than one second to complete. The longest run

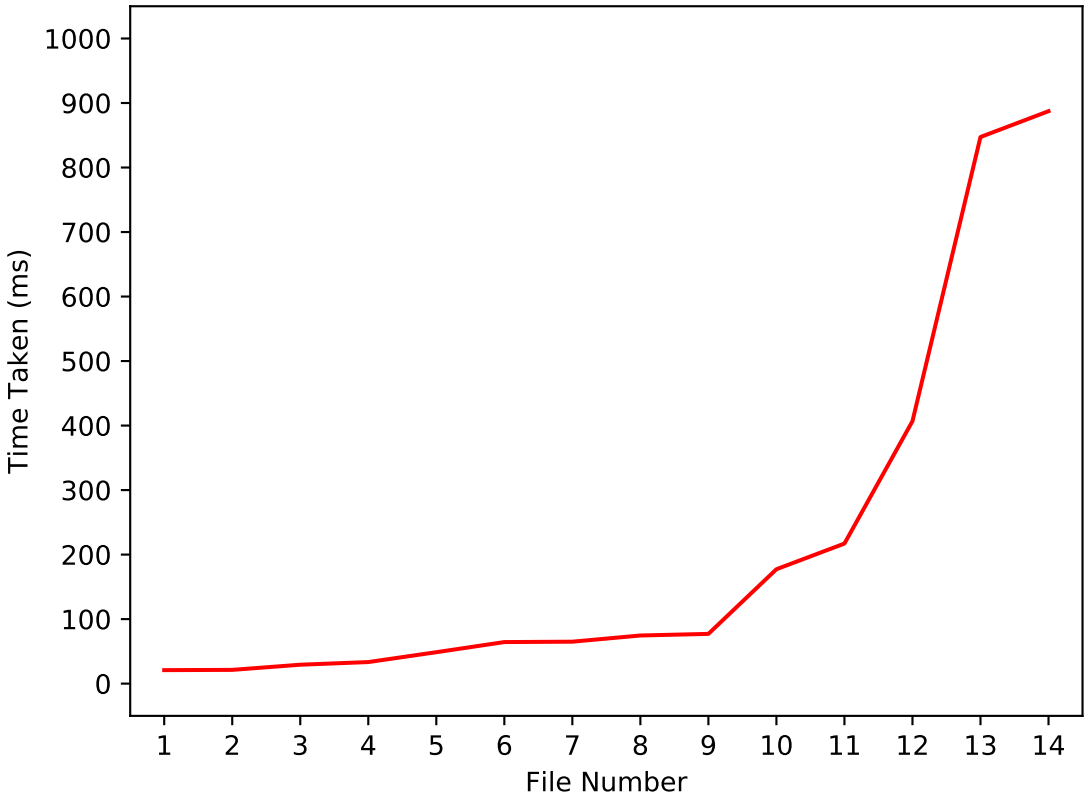


Figure 4.1: Past Years Data Performance Results

time took 887.3 ms, and was one of the largest input files. The average run time was 212.2 ms. It appears from the past years data that our program can scale to support any amount of input that the PLTL leaders might submit. However, we still needed to test with the current year's data.

File	# Students	# Groups	# Professors	Time Taken
Bauer (Students Locked In)	76	35	1	20.9 ms
Michael	82	4	1	21.9 ms
Sawyer	188	36	1	113.9 ms
Bauer (No Students Locked In)	76	35	1	223.8 ms
Molly	235	20	3	3227.0 ms

Table 4.2: Current Year Data Performance Results

We had four different people try running our program. The input they submitted was real data from multiple classes from the spring semester of 2020. Professor Chris Bauer ran his input twice, one with the students locked in and with different parameters. The results can be seen in Figure 4.2. Again, almost all files took less than one second to run, with the exception of Molly's General Chemistry class. The longest run time took 3.227 seconds, and was the largest input file. The average run time was 721.5 ms. While this average is higher than the past years, Molly's input is an outlier that skews the average significantly. Without Molly's file, the average would be 95.4 ms. We felt more confident that the ILP solver was going to be sufficient enough to justify not needing to implement heuristic methods and local search. To prove completely that the ILP provides optimal solutions in a reasonable amount of time, we ran trials of different sized inputs using synthetic data that was generated based on previous data.

Having previous data let us find statistics about an average class. Between all chemistry classes that we had past data on, about 23.2% of students were male, and about 77.8% of the students were female. Roughly 47.7% of the all students in every class were freshmen, 34.3% of students were sophomores, 13.8% of students were juniors, and 4.2% of students were seniors. For student preferences, among all students for all group times, students said they preferred a group time about 8.9% of the time, said a group time was possible about

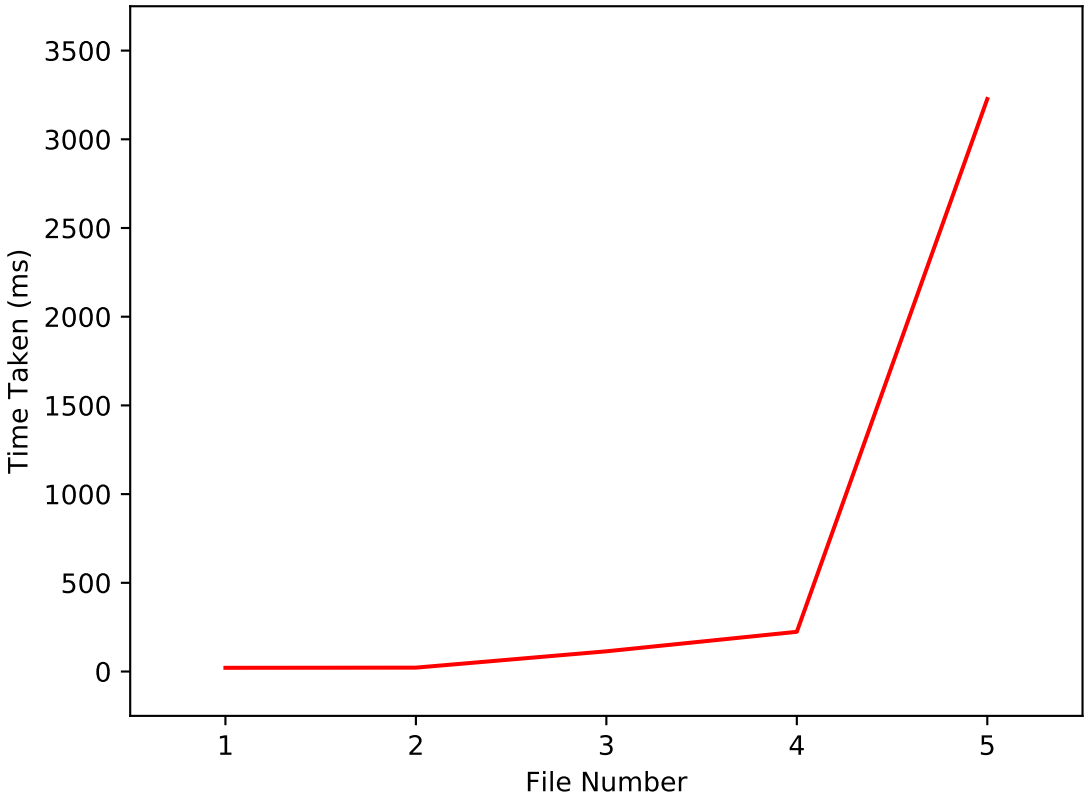


Figure 4.2: Current Year Data Performance Results

15.1% of the time, and said a group time was impossible 76% of the time. The percent of preferences showed variance depending on the number of groups, but we felt this was an accurate estimate for our testing.

Using this information, we generated synthetic data that matched these statistics to test the performance of the ILP solver. Since we know that the ILP works well for past and current data, we wanted to be able to predict the run time based on the number of students, groups, and professors in the input.

Since Gurobi supports multi-threading, we were able to change the amount of threads the solver used.

With only one professor, run time stays very low. This was shown in the past and current data as well. The results are shown in Figure 4.3. The longest run time was with 100 students and 40 groups, and took 1.739 seconds to complete. Often times, with a low number of students, randomness in the probability of generating the data can cause longer run times. For instance, the run time for 100 students and 40 groups is about 1600 ms longer than 150 students and 40 groups, which theoretically should be a harder problem. It can be expected for inputs with one professor will frequently take approximately one second or less.

When the input contains two professors, run time experiences a significant increase, as shown in Figure 4.4. The maximum run time for two professors took 464 seconds, or 7.73 minutes, when the input contained 250 students and 25 groups. This 10:1 student to group ratio also appears in the second longest run time of 304 seconds, or 5.07 minutes, when the input contained 300 students and 30 groups. Since the default absolute maximum on the number of students in a group is 10, this is likely due to the easy-hard-easy pattern [29], where the number of possible solutions is high and the difference between solutions is very small. Since the solver is trying to assign every student to a group, in order to find the optimal solution, more of the search space has to be explored. Input with two professors has varying run times depending on the number of students and groups, as well as the student to group ratio, but run time is usually less than 10 minutes.

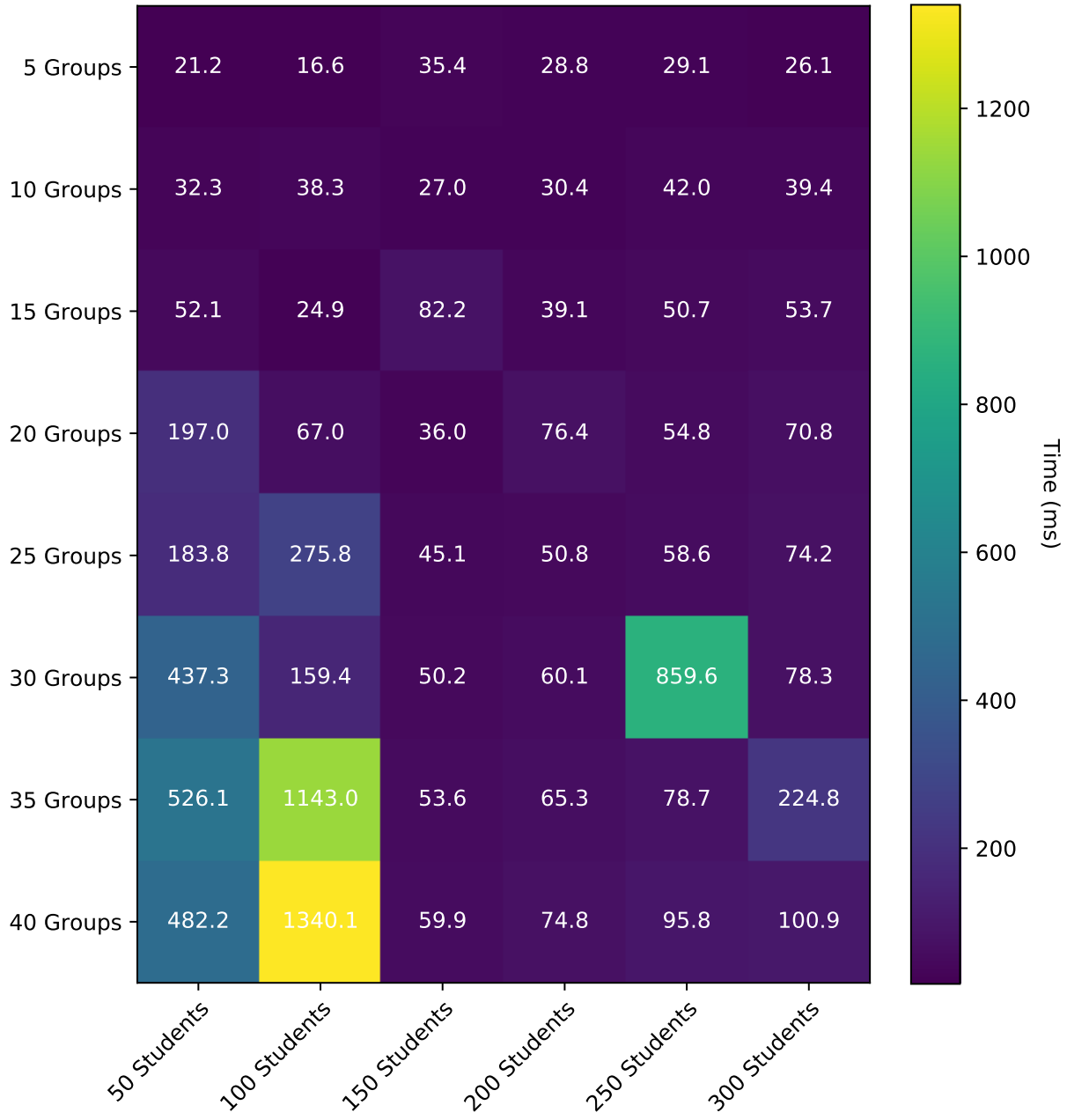


Figure 4.3: Synthetic Data Performance Results (1 Professor)

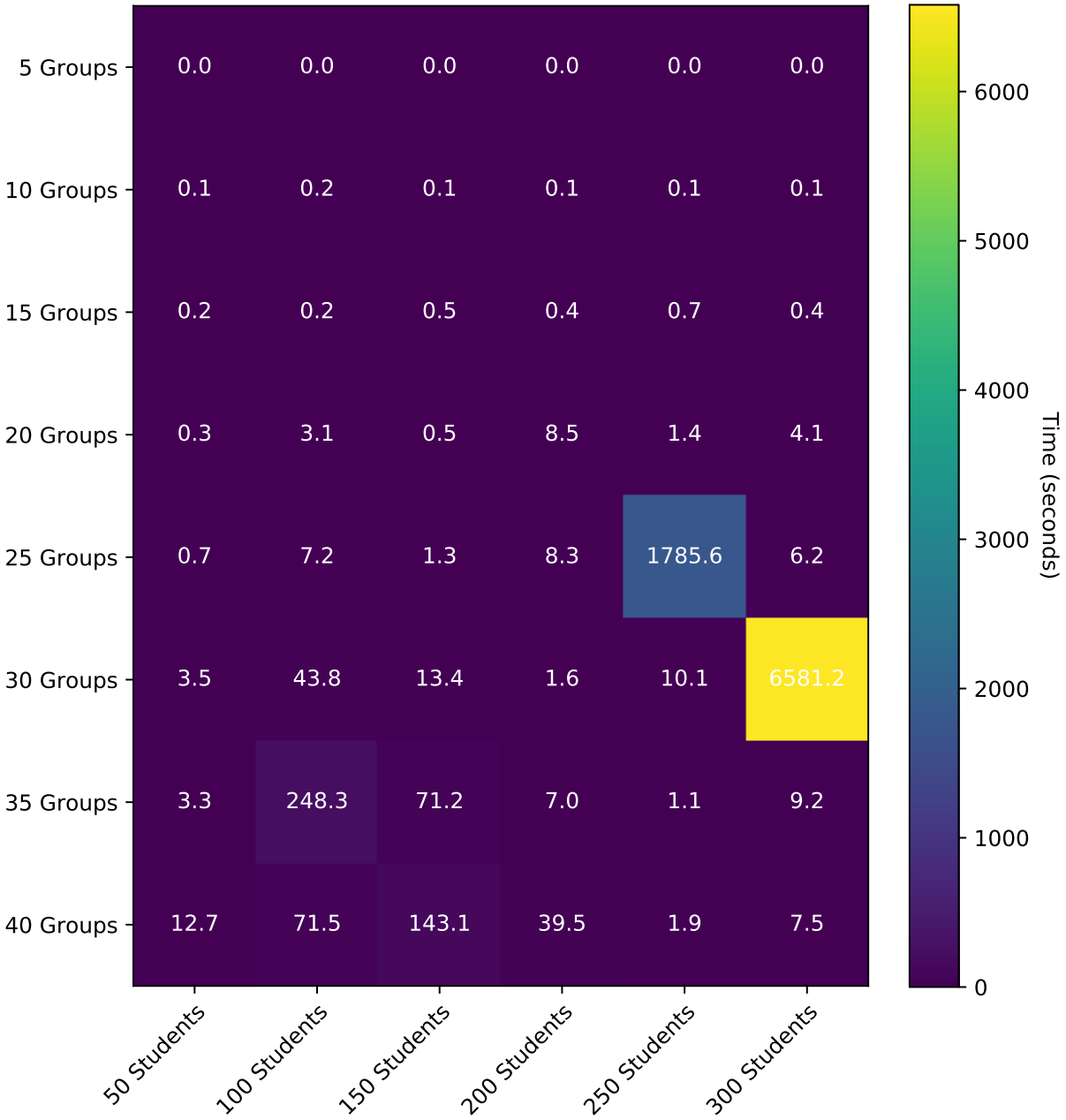


Figure 4.4: Synthetic Data Performance Results (2 Professors)

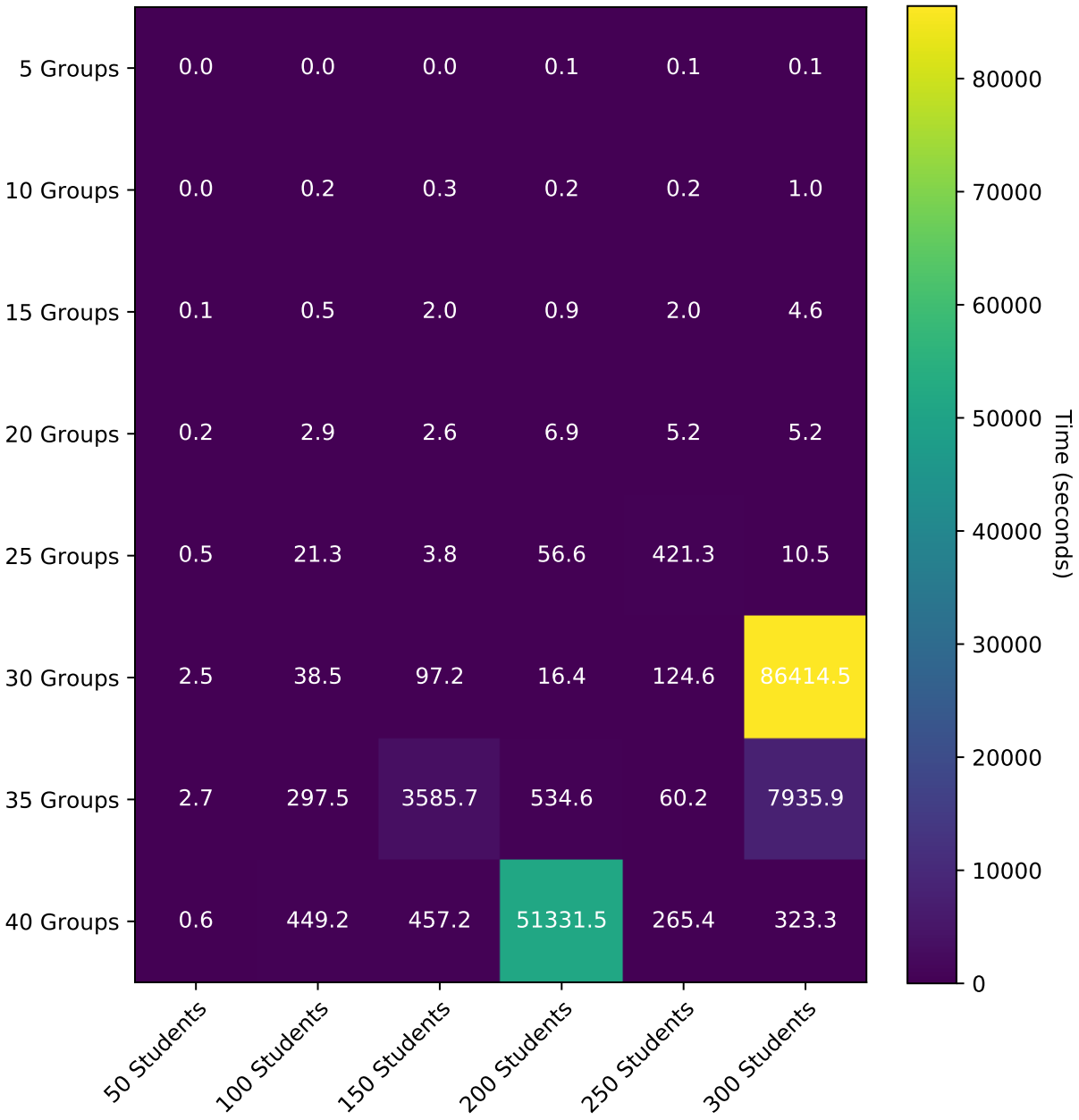


Figure 4.5: Synthetic Data Performance Results (3 Professors)

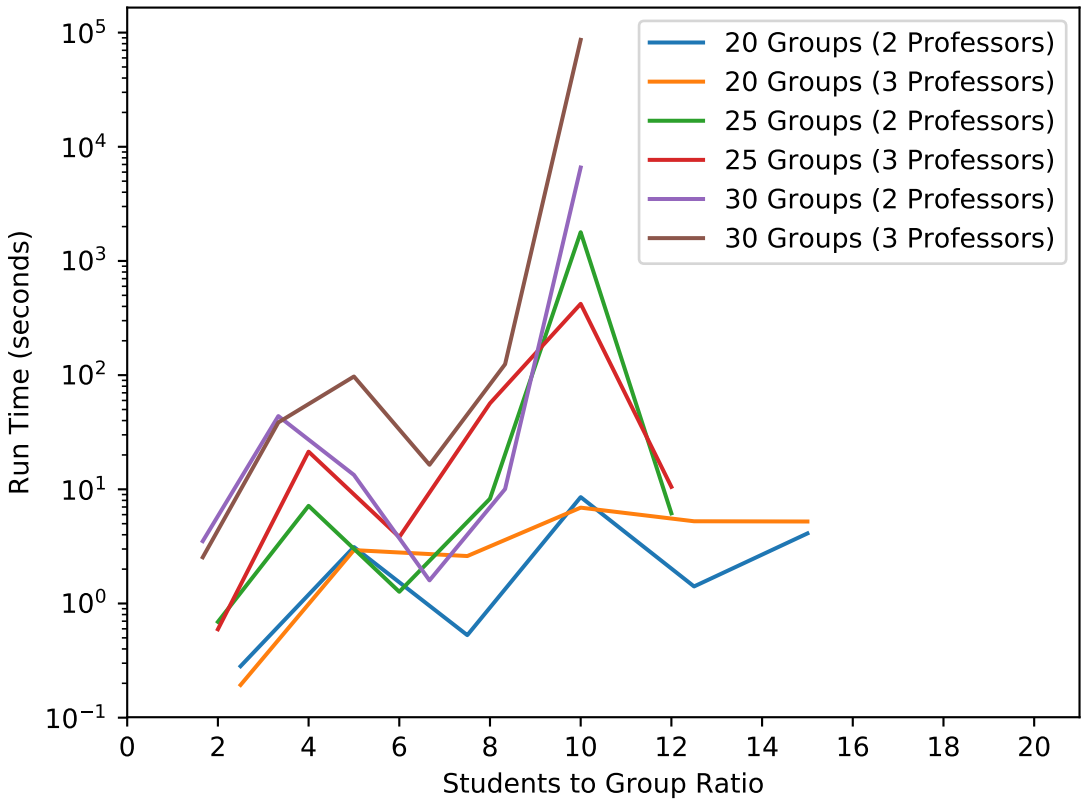


Figure 4.6: Easy-Hard-Easy Pattern Shown

Finally, with three professors, run time becomes harder to predict, and randomness in the input creates a lot of variance in run times. The results are shown in Figure 4.5. The maximum run time for three professors took one day, when the input contained 300 students and 30 groups. This also follows the easy-hard-easy pattern found in two professors, with a student to group ratio of 10:1. Run time with three professors is affected greatly by the statistics of the students as the input. Students can never be assigned to a group they declare is impossible for them, so the less possible groups a student has, the less total number of possible solutions the solver can explore. With three professors, run times range from less than a second to a whole day.

The easy-hard-easy pattern can be more easily seen in Figure 4.6, where the student to group ratio of 10 shows the longest run times. There is a large spike in run time at the ratio

of 10, compared to the ratios before and after it.

Gurobi is very efficient with memory use. When running the program on Mica using ten cores, a file that had been running for 12 hours had only consumed 20% of the total memory, which was around 25GB of RAM. Mica has 126GB of memory, and our program limits run time to one day, so memory should never be a problem. If, however, the amount of cores used is changed, or a system with less memory is used, memory use can be reduced by telling Gurobi to use less cores, or by changing the solving method to a method that uses less memory.

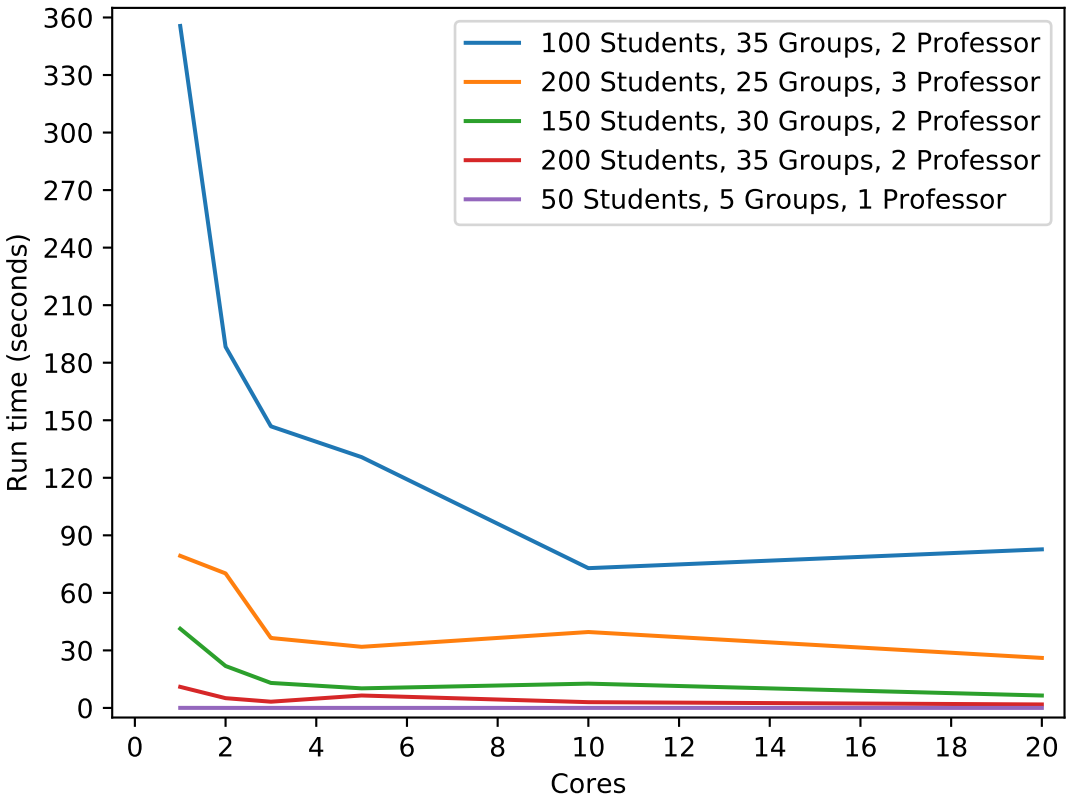


Figure 4.7: Run Time as Cores Increase

When memory is not a problem, such as on Mica, it is useful to increase the number of cores Gurobi uses to solve the ILP. We decided that ten cores was a good amount for Mica, because we wanted to get good performance without using all of the CPUs the server has.

Figure 4.7 shows how run time is affected by the number of cores used.

Generally, as the number of cores increases, the run time is better. However, this is not always the case, as some problems get no benefit from using more cores, thus increasing the run time by setting up threads for the program to run on. When the input is small, the number of cores does not significantly improve run time. However, with an input of 100 students, 35 groups, and 2 professors, using 10 cores showed a 4 times speed up compared to one core.

A big problem that UNH Chemistry currently faces is finding rooms for their PLTL groups. Therefore, it is unlikely that PLTL has the capacity for more than 300 students and 40 groups. At the moment, our program has quickly been able to assign students to groups with all of the real data the PLTL leaders have sent to it. However, Professor Chris Bauer said that the most students he could ever imagine signing up for PLTL for one class is 700 students, between three professors. In order to accommodate for the potential scaling of the PLTL program, we needed to test how well the solver worked under a time constraint.

It is extremely likely that inputs that reach 700 students between three professors could take days or longer to solve. In these situations, the ILP solver needs to return a plausible solution that is as close to optimal as it can get. To test this, we ran files with varying numbers of students, groups, and professors, that took between five and nine minutes to solve, and recorded objective function values at set time intervals. This should theoretically scale for files that take days to solve.

As shown in Figure 4.8, most of the time, the solver was able to find the minimum objective function value for a file before the program returned a solution. This was only not the case in the file with 250 students, 25 groups, and 3 professors. Therefore, on files that take between five and nine minutes, you can expect to be within a few values of optimal within a couple minutes, *most of the time*. Of course, this will depend on the students data in the input, and it is never guaranteed. Gurobi also offers the ability to output an optimality gap, which can be used to know how far from optimal you might be. In testing, we observed

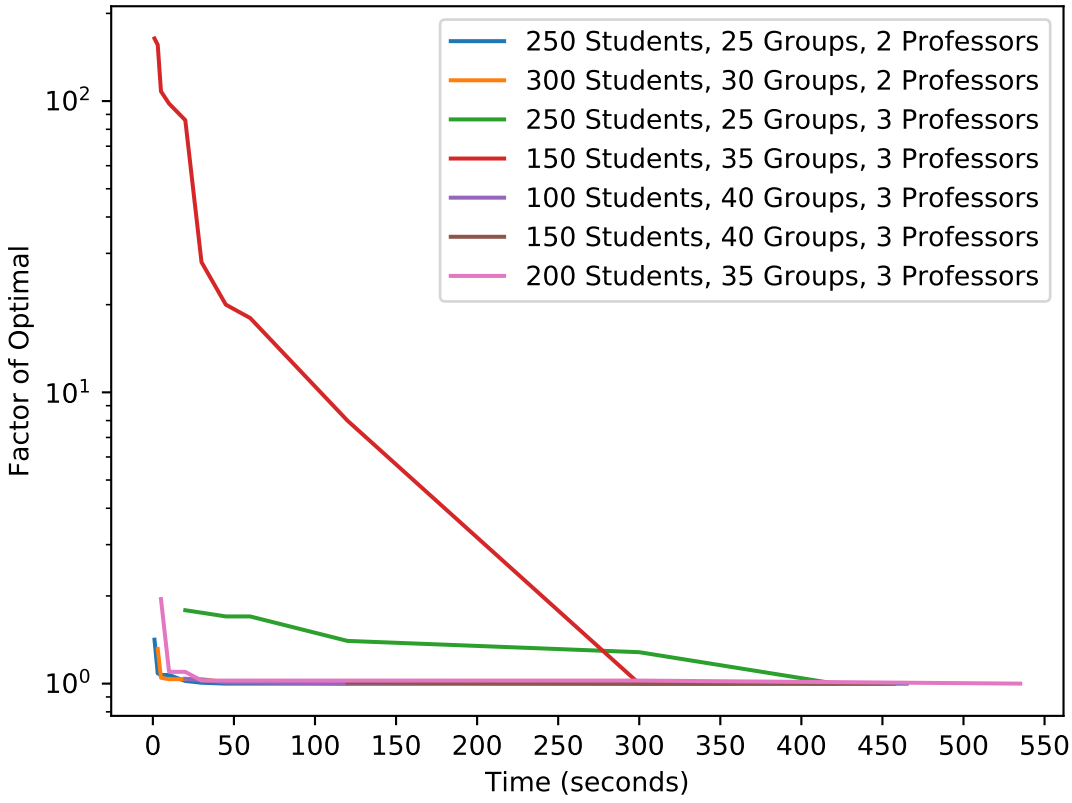


Figure 4.8: Difference from Optimal After Stopping ILP at Time Intervals

that the reported optimality gap is on average 0.00061% off of the actual optimality gap. To calculate this, we took the optimal objective function value for each file, subtracted the found objective function value at each time limit, and then divided that value by the found objective function value. The full equation is shown in Equation 4.1. The value *Gap* is what was compared against the reported optimality gap by Gurobi.

$$Gap = |Optimal - Found|/|Found| \tag{4.1}$$

The objective value found would often be less than the actual minimum objective value solution with very small run times (1-30 seconds), and this is because while the solver tries combinations, it may try solutions that are not possible. The ILP solver will return the best solution it has found when the time limit is reached, and when this happens, there is no guarantee that all of the constraints were met. The majority of these cases happen before 30 seconds, and therefore, no matter what time a user submits to the program, the minimum time limit is set to 30 seconds.

As the input scales and increases to files that take a full day, we expect that the time it takes to find a solution close the the optimal one would scale at the same rate. If a file that takes seven minutes will find an objective value close to optimal in a couple minutes, a file that takes a day could expect to find an objective value close to optimal in a 6 hours. This is not guaranteed, and will depend on the student data in the input.

While the number of students participating in PLTL for a given class is usually less than 300, in the event that PLTL does increase in size, our program will be able to support the increased input, as long as users are confident.

We feel we have enough evidence to show that our program is fast enough to avoid implementing heuristic methods and local search. The ILP implementation is sufficient for the current data being sent in, and all future data according to Professor Chris Bauer.

4.2 Unit Testing

When dealing with files as input, there's a lot that can go wrong. To ensure that files are formatted correctly when they are read in by the ILP solver, our front end program tests the file thoroughly for any possible errors. These errors include if the file is blank or empty, group information missing, duplicate groups (leader and time), student information missing, missing sections, missing parameters, invalid parameters, duplicate parameters, group meeting times missing from the students section, group preference times missing from the groups section, duplicate group preference times, and invalid preferences.

As more PLTL leaders used the program during its development and testing phase, more errors were able to be detected. Most of the errors we ran in to had to do with extraneous characters in files, such as duplicate carriage returns or a Byte Order Mark. Some errors were simple problems like spelling errors or invalid values. It is extremely important that any possible errors could be detected and addressed now, to ensure the program will work years in to the future.

To accomplish this, we created a suite of unit tests to confirm that problems were being detected by the front end program, and came up with as many possible tests we could. The results of the tests are shown in Table 4.3.

If the front end program found no problems with the file, we also ran it on the ILP program to make sure the results looked correct for the input. This was to make sure that not only our front end program worked correctly, but that our ILP model was correct.

Unit testing was very important for this program to ensure that years from now, no changes to the program need to be made. The testing done on the program will help to increase the longevity of the program, allowing it to be used by UNH Chemistry for years.

Test Name
Valid File
Minimal File
Empty File
Non-Related/Irrelevant File
Random White-space and Capitals
Simple Version
Most Students Preferences Impossible
All Students Preferences Impossible
Large Group of Students Locked In
Small Group of Students Locked In
Student Locked In Not In Student Section
Duplicate Student Locked In
Missing Group Info
Duplicate Groups
Misspelled Group Info
No Groups
Unassigned Section Missing
Missing Parameters
Duplicate Parameters
Misspelled Parameters
Invalid Parameter Values
Extraneous Lines in Parameters
Parameters Placed Out of Order
Student Header Row Missing
Student Header Row Misspelled
Student Header Row No Groups
Student Header Row Extra Groups
Missing Student Info
Duplicate Students
Invalid Student Info
No Students

Table 4.3: Unit Tests

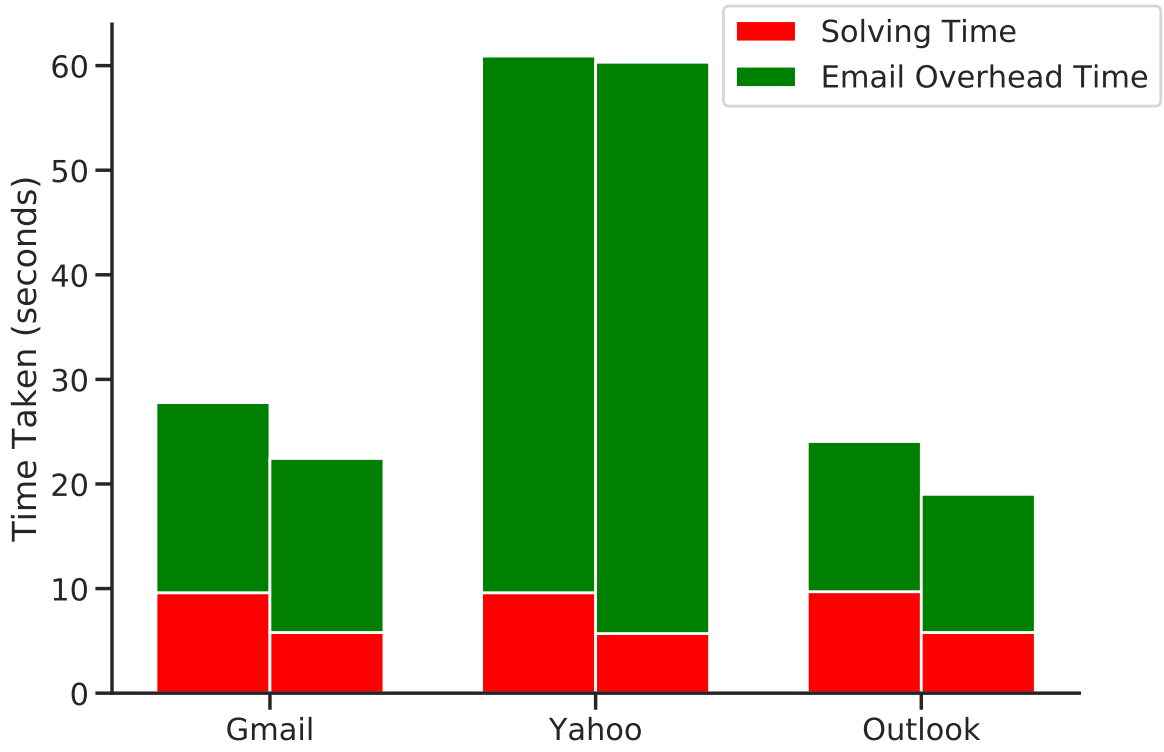


Figure 4.9: ILP solver and Email Overhead Times

4.3 Usability

Using an email application on top of the ILP solver introduces an element of latency in the performance of the application. While all of the tests above were performed directly on the ILP program, without using the email, it is important to show that even with an email interface on top of the program, it still behaves as expected. The round trip time it takes for a user to send their email and receive their results should depend on what email service they are using, and be independent of the ILP solver.

Two files with different numbers of students, groups, professors, and parameters were used for testing, and the average of three trials is displayed in Figure 4.9. The first column of each group are the results from the first file, and the last column of each group are the results from the second file. The red bars show that no matter what email service is being

used, the time it takes the ILP program to run remains the same. The green columns show that each email service has its own overhead time, and is independent of the ILP solver. Additionally, the overhead time remains fairly constant for each email service, meaning that if you know how long it takes the ILP program to run, you can estimate how long it will take for the full round trip time of using the program.

CHAPTER 5
FEEDBACK

Since the development of this program relied so heavily on feedback from the UNH Chemistry department, we worked closely with PLTL leaders, as well as Professor Chris Bauer, to ensure that the program met their needs.

5.1 In-Person Feedback

During development, we sat in with students trying the program for the first time to see what they had questions on, and see how someone who is unfamiliar with the program would behave using it. The focus was to make sure that our understanding of how the program would be used is actually what would happen when the PLTL leaders who were unfamiliar with our program used it.

After the initial prototype program was made, we were able to sit in with PLTL leaders Molly Hanlon and Ryan Dussault to see how they handled the program for the first time. We gave them a quick introduction to the program and explained the process, and then let them try to use it on their own. There was a lot of confusion in setting up the input file, but we were able to guide them through the process. Once they understood how to set up the file, they were able to send it to the program and get their results. At this stage in the design, our program required that every student is placed in a group. Unfortunately, we didn't plan for input where there were not enough groups for every student to be assigned to a group. Molly and Ryan both agreed that it would be beneficial for the program to be able to leave students unassigned, and we were able to add the change to their program.

Molly and Ryan were having troubles with formatting the input file, but we were not sure if this would be a problem for other PLTL leaders. Molly and Ryan both admitted that they were not very good with technology, and that the format was confusing for them. As more PLTL leaders tried using the program, we found that it usually took each leader between three and five tries to successfully format the input. Because of this, we felt that we needed an easier way for PLTL leaders to submit their input to the program, while still capturing all the requirements they need, which became the motivation for the simple version of the input.

5.2 Survey Feedback

As PLTL leaders used the program, we sent them a Google Forms survey asking for their feedback. We got responses from four PLTL leaders, with a variety of positive and negative feedback. Three students answered that to do this task manually would have take them hours (90 minutes, 3 hours, and 4 hours). The fourth student said that they had never had to do this task for a large group. We can assume that since the group was large, it would take that student hours as well.

The speed up this program provided was significant. The student that had never done this task manually, as well as the student that said it would have taken them 90 minutes, said it took them 30 minutes to figure out how to use the program and get their results. For these students, our program was able to provide at least a three times speedup from the manual process. Molly and Ryan, who learned how to use the program with our help, said it took them five minutes, for a speedup of 36 times the manual process. One student was unable to get the program to work.

For the students who were successful in running the program, all three liked the groups suggested by the program. One even said "They were organized as well as any manual group". They were most satisfied with the group sizes, and the amount of students that got a preferred time slot. While two of the students did not have any suggestions for improvements

to the groups, one student suggested that the gender diversity of the groups were mostly good, but they didn't like that one group had only females. This was not brought up in our initial meeting with PLTL leaders, but we decided that this could be an optional feature. In addition to avoid singling out traits, we added the optional feature to our program to avoid creating groups of only traits.

The PLTL leaders felt the program was very usable, and they enjoyed the Gmail format. Two of them mentioned that they felt the program was daunting, but after explanation and experimentation that it was easy and efficient. In the following question, both of these students suggested that there be some sort of separate document that provides instructions on setting up the file. Ryan and Molly mentioned that if they were not with us to help them set up the input file, that it would have been harder. This feedback was very important to hear, because it told us that the instructions we left in the template file were not clear enough. From these suggestions, we decided to create a user guide that is sent to the user when ever there is an error with their input file.

This user guide provides step by step instructions, as well as pictures, to help the user correctly format the input. It talks about both the simple version and the full version, and also discusses fixes for any error messages it responds with.

A suggestion for what additional information the program should take into account was students comments. While the program does offer the ability to write comments above a line, these comments do not follow the student around to the groups they are placed in. Because of this, we added the *Notes* column to the header row to allow any notes the PLTL leader might have on the student to follow that student around the file.

Another suggestion was to take in to account what order the students filled the survey out. This is especially significant for this year, since the number of students who signed up for PLTL was much higher than expected. Due to the excess of students, not all students were able to be placed. Because of this, the PLTL leaders suggested that there should be priority given to those who respond to the survey first. to accomplish this, we add a small

penalty to every student based on their order in the input.

$$p_{ij} + \frac{1}{5 + m - i} \tag{5.1}$$

In equation 5.1, p_{ij} is the cost of assigning student i to group j, for m students.

All of the PLTL leaders had positive things to say about the program, talking about how easy it was to use, the amount of time it saves, and the email format. The negative feedback about the program was mostly about the learning curve and how the input was confusing. This was another motivation to creating a user guide, so future PLTL leaders will have instructions to follow and solutions to their problems.

This survey was the best feedback we received about our program. Almost all of the users had no help using the program, so their experience was unbiased. Since we were so involved in designing and making the program, we often had trouble taking a step back and seeing things from the bigger picture. The feedback from these new users helped us see problems that we couldn't see before, and inspired new features to the program that the leaders wanted to see.

CHAPTER 6
CONCLUSION

6.1 Summary

UNH Chemistry has been sorting students in to groups based on preference and availability for years, but they've been doing the process by hand. This process is tedious, error-prone, and inefficient. Previous students have attempted to make a solution for them by creating a program to automate the process. This program did not meet their standards, and did not last long due to issues in the deployment of the program, as well as cross platform operating system errors. We decided that the problem was well enough understood to lock in a certain set of constraints, and solve their problem more efficiently.

To solve their problem, we created an email application that users can send their group information to that will reply with a file containing groups assignments for students based on their preferences and traits. To accomplish this, we used the Gurobi ILP solver, which is sufficiently quick to solve the input. In extreme scenarios, the average run time still remained under one minute. It is not obvious to us that the problem is NP-hard, but we used an ILP solver for convenience, to guarantee optimal solutions, and to still get reasonable run times.

The program was designed with a strong influence from the users' feedback. Many features of the program were requests from Professor Chris Bauer and PLTL leaders, and we listened to their feedback during the development process. Additionally, their testing of the program helped us to discover problems, as well as provided more test data for us. We demonstrated that the system we've developed is useful for classes that run PLTL groups.

In the future, Professor Wheeler Ruml has offered to host the program. He will work

with Professor Chris Bauer to coordinate when to start the program every semester, and will not have to interact with the program afterwords. Performance testing and unit testing have ensured that the program will last for years to come.

6.2 Limitations

This program can only solve the task of assigning students in to groups based on their preferences and traits. Unlike the timetabling problem, this program assumes that meeting times have already been created, and does not attempt to schedule the groups for non-overlapping times. Additionally, this program only uses the students' availability for specific times, and does not create groups from blocks of available times.

6.3 Possible Extensions

As part of the feedback from PLTL leaders who used the program, there were some ideas that we did not implement because it went outside the scope of Professor Chris Bauer's requests.

A potential extension on this program could be creating suggested groups for the unassigned students. This could either be done by looking at what times they said they were available and creating suggested groups from known availability, or by looking across past data to find the most available times among all students and creating suggested groups from unknown availability.

Another extension could be extending the priority system for students. Currently, priority is assigned at a flat rate based on order of appearance in the input. This could be expanded to either be a parameter, or a new column in the input, where students can be assigned a priority, which won't guarantee them a spot, but will strongly influence their assignment.

Finally, if the PLTL system expands to UNH Biology, a possible extension would be allowing students to be assigned to two different groups; one for chemistry, and one for biology. These times would need to not overlap, and group leaders would need to be assigned

to either chemistry or biology as part of the input. This would likely increase run times significantly, and there might be a more efficient way of implementing this change.

LIST OF REFERENCES

[1] Deniz Yuret and Michael De La Maza. Dynamic hill climbing: Overcoming the limitations of optimization techniques. In *The Second Turkish Symposium on Artificial Intelligence and Neural Networks*, pages 208–212. Citeseer, 1993.

[2] D. E. Joslin and D. P. Clements. Squeaky Wheel Optimization. *Journal of Artificial Intelligence Research*, 10:353–373, May 1999.

[3] Craig A Tovey. Hill climbing with multiple local optima. *SIAM Journal on Algebraic Discrete Methods*, 6(3):384–393, 1985.

[4] Steven Hampson and Dennis Kibler. Large plateaus and plateau search in boolean satisfiability problems: When to give up searching and start again. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:437–455, 1996.

[5] Lt Col James L Brickell, Lt Col David B Porter, Lt Col Michael F Reynolds, and Capt Richard D Cosgrove. Assigning students to groups for engineering design projects: A comparison of five methods. *Journal of Engineering Education*, 83(3):259–262, 1994.

[6] K R Baker and S G Powell. Methods for assigning students to groups: a study of alternative objective functions. *Journal of the Operational Research Society*, 53(4):397–404, April 2002.

[7] B. Naderi. Modeling and Scheduling University Course Timetabling Problems. *International Journal of Research in Industrial Engineering*, (Online First), August 2017.

[8] A.S. Asratian and D. de Werra. A generalized class–teacher model for some timetabling problems. *European Journal of Operational Research*, 143(3):531–542, December 2002.

[9] Hamed Babaei, Jaber Karimpour, and Amin Hadidi. A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, 86:43–59, August 2015.

[10] Sophia Daskalaki and Theodore Birbas. Efficient solutions for a university timetabling problem through integer programming. *European Journal of Operational Research*, 160(1):106–120, 2005.

[11] David F Manlove. Hospitals/Residents Problem. page 5.

[12] Alvin E. Roth and Marilda Sotomayor. The College Admissions Problem Revisited. *Econometrica*, 57(3):559, May 1989.

[13] David G McVitie and Leslie B Wilson. The stable marriage problem. *Communications of the ACM*, 14(7):486–490, 1971.

[14] David W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793, January 2007.

[15] Ventepaka Yadaiah and VV Haragopal. A new approach of solving single objective unbalanced assignment problem. *American Journal of Operations Research*, 6(1):81–89, 2016.

[16] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800020109.

[17] J. Kennington and Z. Wang. A Shortest Augmenting Path Algorithm for the Semi-Assignment Problem. *Operations Research*, 40(1):178–187, February 1992.

[18] Richard Barr, Fred Glover, and Darwin Klingman. A new alternating basis algorithm for semi-assignment networks. *Computers and mathematical programming*, pages 223–232, 1978.

[19] Temel Öncan. A Survey of the Generalized Assignment Problem and Its Applications. *INFOR: Information Systems and Operational Research*, 45(3):123–141, August 2007.

[20] Dirk G. Cattrysse and Luk N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3):260–272, August 1992.

[21] Roland Hübscher. Assigning Students to Groups Using General and Context-Specific Criteria. *IEEE Transactions on Learning Technologies*, 3(3):178–189, July 2010. Conference Name: IEEE Transactions on Learning Technologies.

[22] Luis E. Agustín-Blas, Sancho Salcedo-Sanz, Emilio G. Ortiz-García, Antonio Portilla-Figueras, and Ángel M. Pérez-Bellido. A hybrid grouping genetic algorithm for assigning students to preferred laboratory groups. *Expert Systems with Applications*, 36(3):7234–7241, April 2009.

[23] Putu Indah Ciptayani, Kadek Cahya Dewi, and I Wayan Budi Sentana. Student grouping using adaptive genetic algorithm. In *2016 International Electronics Symposium (IES)*, pages 375–379, September 2016.

[24] L Darrell Whitley, Adele E Howe, S Rana, Jean-Paul Watson, and Laura Barbulescu. Comparing heuristic search methods and genetic algorithms for warehouse scheduling. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, volume 3, pages 2430–2435. IEEE, 1998.

[25] Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[26] Snežana Mitrović-Minić and Abraham P. Punnen. Local search intensified: Very large-scale variable neighborhood search for the multi-resource generalized assignment problem. *Discrete Optimization*, 6(4):370–377, November 2009.

[27] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020.

[28] Laurent Perron and Vincent Furnon. Or-tools.

[29] Dorothy L Mammen and Tad Hogg. A new look at the easy-hard-easy pattern of combinatorial search difficulty. *Journal of Artificial Intelligence Research*, 7:47–66, 1997.

APPENDIX A

Example Input File

	A	B	C	D	E	F	G
1	######################						
2	# Section 1: Groups						
3	######################						
4	~~Group	Luke Skywalker	luke.skywalker@unh.edu	Monday 12:10-1:30 PM			
5	R2 D2	r2d2@unh.edu	Preferred	Male	Sophomore	Chris Bauer	Beep boop
6	C3 PO	c3po@unh.edu	Preferred	Male	Sophomore	Chris Bauer	
7	~~Group	Han Solo	Han.solo@unh.edu	Monday 3:10-4:30 PM			
8	~~Group	Princess Leia	Princess.leia@unh.edu	Monday 4:10-5:30 PM			
9	~~Group	Baby Yoda	baby.yoda@unh.edu	Tuesday 12:10-1:30 PM			
10	~~Unassigned						
11	Chewbacca	Chewwy@unh.edu					

Figure A.1: Example Input File: Section 1 (Groups)

	A	B
12	########################	
13	# Section 2: Parameters	
14	########################	
15	Smallest Possible Group Size	4
16	Largest Possible Group Size	10
17	Smallest Preferred Group Size	6
18	Largest Preferred Group Size	8
19	Increase Preferred Group Size Penalty	3
20	Decrease Preferred Group Size Penalty	10
21	Student Non-Preferred Assignment Penalty	2
22	Unassigned Penalty	50
23	Singling Out Male Penalty	0
24	Singling Out Female Penalty	0
25	All Males Penalty	0
26	All Females Penalty	0
27	Singling Out Freshman Penalty	0
28	Singling Out Sophomore Penalty	0
29	Singling Out Junior Penalty	0
30	Singling Out Senior Penalty	0
31	All Freshmen Penalty	0
32	All Sophomores Penalty	0
33	All Juniors Penalty	0
34	All Seniors Penalty	0
35	Time Limit	600

Figure A.2: Example Input File: Section 2 (Parameters)

	A	B	C	D	E	F
36	########################					
37	# Section 3: Students					
38	########################					
39	First Name	Last Name	Email	Gender	Year	Professor Name
40	Darth	Vader	darth.vader@unh.edu	Male	Freshman	Chris Bauer
41	Obiwan	Kenobi	obiwan.kenobi@unh.edu	Male	Freshman	Chris Bauer
42	Chewbacca	Wookie	chewwy@unh.edu	Male	Freshman	Chris Bauer
43	R2	D2	r2d2@unh.edu	Male	Sophomore	Chris Bauer
44	C3	PO	c3po@unh.edu	Male	Sophomore	Chris Bauer
45	Master	Yoda	yoda@unh.edu	Male	Senior	Chris Bauer
46	Sheev	Palpatine	sheev.palpatine@unh.edu	Male	Junior	Chris Bauer
47	Kylo	Ren	kylo.ren@unh.edu	Male	Freshman	Chris Bauer
48	Rey	Jedi	rey@unh.edu	Female	Freshman	Chris Bauer
49	Boba	Fett	boba.fett@unh.edu	Male	Sophomore	Chris Bauer
50	Jabba	Hutt	jabba.hutt@unh.edu	Male	Sophomore	Chris Bauer
51	Padme	Amidala	padme.amidala@unh.edu	Female	Sophomore	Chris Bauer
52	Lando	Calrissian	lando.calrissian@unh.edu	Male	Freshman	Chris Bauer
53	Ashoka	Tano	ashoka.tano@unh.edu	Female	Freshman	Chris Bauer
54	Mace	Windu	mace.windu@unh.edu	Male	Freshman	Chris Bauer
55	Jyn	Erso	jyn.erso@unh.edu	Female	Freshman	Chris Bauer
56	Jar Jar	Binks	jarjar.binks@unh.edu	Male	Freshman	Chris Bauer
57	Rose	Tico	rose.tico@unh.edu	Female	Freshman	Chris Bauer
58	Darth	Maul	darth.maul@unh.edu	Male	Freshman	Chris Bauer
59	Admiral	Akbar	admiral.akbar@unh.edu	Male	Freshman	Chris Bauer
60	Jango	Fett	jango.fett@unh.edu	Male	Freshman	Chris Bauer

Figure A.3: Example Input File: Section 3 (Students) Part 1

G	H	I	J	K
Notes	Monday 12:10-1:30 PM	Monday 3:10-4:30 PM	Monday 4:10-5:30 PM	Tuesday 12:10-1:30 PM
	Possible	Impossible	Preferred	Possible
	Impossible	Preferred	Possible	Possible
	Impossible	Preferred	Possible	Possible
Beep boop	Preferred	Possible	Possible	Possible
	Preferred	Impossible	Impossible	Possible
	Preferred	Possible	Possible	Possible
	Preferred	Possible	Impossible	Possible
	Possible	Possible	Preferred	Possible
Actually Prefers Tuesday	Impossible	Impossible	Preferred	Possible
	Impossible	Preferred	Impossible	Possible
	Possible	Impossible	Preferred	Possible
	Possible	Preferred	Possible	Possible
	Preferred	Possible	Impossible	Possible
	Preferred	Impossible	Impossible	Possible
	Impossible	Impossible	Preferred	Possible
	Impossible	Preferred	Possible	Possible
Sith lord	Impossible	Possible	Preferred	Possible
	Preferred	Impossible	Possible	Possible
	Possible	Preferred	Possible	Possible
	Impossible	Preferred	Impossible	Possible
	Impossible	Impossible	Preferred	Possible

Figure A.4: Example Input File: Section 3 (Students) Part 2

APPENDIX B

Installation Script Code

```
########################################
# Get Gurobi files
########################################

# Make folder for UNH Group Assign to live in
mkdir UNHGroupAssign

# Download Gurobi
wget https://packages.gurobi.com/9.0/gurobi9.0.1_linux64.tar.gz

# Extract the files and remove the tar
tar xvzf gurobi*
rm -rf gurobi*.tar.gz

# Move all of the files in to the current folder
mv gurobi*/linux64/* UNHGroupAssign
rm -rf gurobi*

########################################
# Untar UNH Group Assign files
########################################
```

```
# Extract files
tar xvzf UNHGroupAssign.tar.gz -C UNHGroupAssign
rm -rf UNHGroupAssign.tar.gz

########################################
# Setup Gurobi
########################################

# Enter the UNH Group Assign Folder
cd UNHGroupAssign

# Get the license from a license key
# YOU WILL BE PROMPTED TO ENTER A LICENSE KEY
./bin/grbgetkey

# Make the Gurobi C++ file
cd src/build
make
cp libgurobi_c++.a ../../lib
cd ../..

########################################
# Setup the UNH Group Assign program
########################################

# Make the c++ program
make
```

```
# Install the python packages
pip3 install google-api-python-client google-auth-httplib2 google-auth-oauthlib
    --upgrade --user
```

APPENDIX C

User Guide

UNH Group Assign

User Guide

Contents

How to Use

So you're a PLTL leader, and you want to use UNH Group Assign to make your life a little easier… No problem! UNH Group Assign will take your students' responses and create your PLTL combinations for you.

SETUP

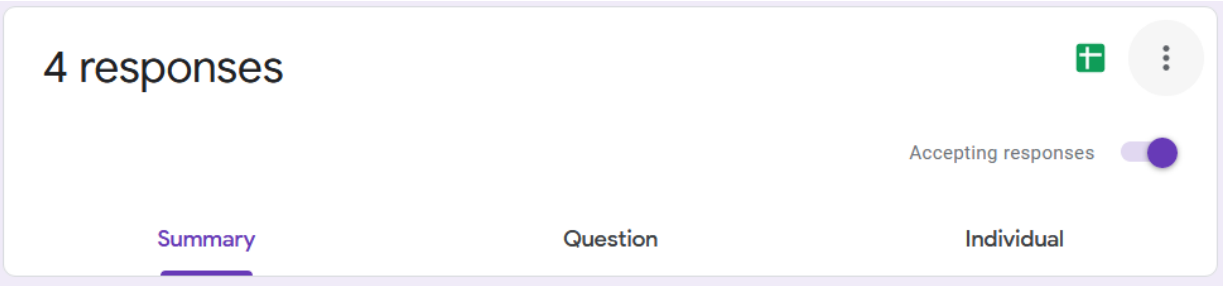
Before we start, we want to make sure that you have all the necessary information you need to use the program. You may have already sent out a Google Form to collect all your data, but to use this program, the following information is necessary:

- First Name
- Last Name
- Email

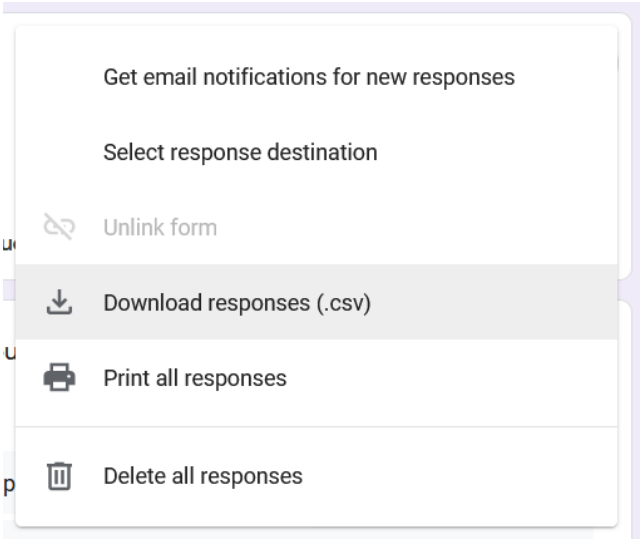
Also, the following information is not *necessary*, but is optional if you want more customizable groups:

- Gender
- Year
- Professor

Now that you have everything you need, go to your Google Form responses, and click on the three dots in the top right corner.



Next, click "Download responses (.csv)" to export all your students' responses in to a CSV (comma separated values) file.



SIMPLE VERSION

Depending on what questions you asked in your Google Form, you may be able to use this file without changing it at all. The first row in your .csv file should be:

| First Name | Last Name | Email | Gender | Year | Professor Name | Notes |

Followed by your groups.

Here's an example:

	A	B	C	D	E	F	G	H
1	First Name	Last Name	Email	Gender	Year	Professor Name	Notes	Monday 12:10-1:30 PM
2	Darth	Vader	darth.vader@unh.edu	Male	Freshman	Chris Bauer	I can't do Wednesdays	Possible
3	Obiwan	Kenobi	obiwan.kenobi@unh.edu	Male	Freshman	Chris Bauer		Impossible
4	Chewbacca	Wookie	chewwy@unh.edu	Male	Freshman	Chris Bauer		Impossible
5	R2	D2	r2d2@unh.edu	Male	Sophomore	Chris Bauer	Beep boop	Preferred
6	C3	PO	c3po@unh.edu	Male	Sophomore	Chris Bauer		Preferred
7	Master	Yoda	yoda@unh.edu	Male	Senior	Chris Bauer	Monday, I prefer	Preferred
8	Sheev	Palpatine	sheev.palpatine@unh.edu	Male	Junior	Chris Bauer		Preferred
9	Kylo	Ren	kylo.ren@unh.edu	Male	Freshman	Chris Bauer		Possible
10	Rey	Jedi	rey@unh.edu	Female	Freshman	Chris Bauer		Impossible
11	Boba	Fett	boba.fett@unh.edu	Male	Sophomore	Chris Bauer		Impossible
12	Jabba	Hutt	jabba.hutt@unh.edu	Male	Sophomore	Chris Bauer		Possible
13	Padme	Amidala	padme.amidala@unh.edu	Female	Sophomore	Chris Bauer		Possible
14	Lando	Calrissian	lando.calrissian@unh.edu	Male	Freshman	Chris Bauer		Preferred
15	Ashoka	Tano	ashoka.tano@unh.edu	Female	Freshman	Chris Bauer		Preferred
16	Mace	Windu	mace.windu@unh.edu	Male	Freshman	Chris Bauer		Impossible
17	Jyn	Erso	jyn.erso@unh.edu	Female	Freshman	Chris Bauer		Impossible
18	Jar Jar	Binks	jarjar.binks@unh.edu	Male	Freshman	Chris Bauer	Please give meesa my preferred!	Impossible
19	Rose	Tico	rose.tico@unh.edu	Female	Freshman	Chris Bauer		Preferred
20	Darth	Maul	darth.maul@unh.edu	Male	Freshman	Chris Bauer		Possible
21	Admiral	Akbar	admiral.akbar@unh.edu	Male	Freshman	Chris Bauer		Impossible
22	Jango	Fett	jango.fett@unh.edu	Male	Freshman	Chris Bauer	I can't do any days	Impossible

While only one group is shown here, you can list as many groups as you want!

FILLING IN THE DATA

Here's some things you should know:

- The first row is **REQUIRED**. Even though some of the data in these columns is optional, the titles in each column are required to be there.
- You can have repeat first and last names, and both columns should be filled in.

- Each row should have a unique email; watch out for students who submit responses twice and remove the duplicates!
- Gender is optional, but the only supported answers are Male and Female; if you have answers other than this, leave the cell blank. This information can be used to take gender into account when making groups.
- Year is optional, but the only supported answers are Freshman, Sophomore, Junior, and Senior; if you have answers other than this, leave the cell blank. This information can be used to take year into account when making groups.
- Professor Name is optional. If the class you lead has multiple professors, this column is used to make sure students with different professors don't appear in the same group.
- Notes is optional. Sometimes students have additional comments they want the leaders to be aware of. This is only used for your convenience, so you can keep track of what extra notes students have for you.
- Under each group is what the students' preferences go. This can either be "Preferred", "Possible", or "Impossible". If a cell is left blank, it will be treated as Impossible.

Rearrange your csv file until it follows the format above.

FULL VERSION

For more control over the groups being made, attached to the email is a file called GroupAssignTemplate.csv. This can be filled out with your data to make the groups more customizable.

This file is made up of three sections: Section 1 (Groups), Section 2 (Parameters), and Section 3 (Students). We're going to describe how to fill out each section in depth.

NOTE: Any line in the file that starts with '#' will be ignored by the program but will be kept in the output. You can write comments in the file by starting the line with #. Any line that is completely blank will be removed from the output.

Section 1: Groups

Section 1 is where you list your group leaders, their emails, and their group times. To add a group to section 1, add a new row that starts with "~~Group", followed by the group leader's name, email, and group meeting time. Below is an example of Section 1 filled with groups.

	A	B	C	D
1	~~Group	Luke Skywalker	Luke.Skywalker@unh.edu	Monday 12:10-1:30 PM
2	~~Group	Han Solo	Han.Solo@unh.edu	Monday 3:10-4:30 PM
3	~~Group	Princess Leia	Princess.Leia@unh.edu	Monday 4:10-5:30 PM
4	~~Group	Baby Yoda	Baby.Yoda@unh.edu	Tuesday 12:10-1:30 PM

You can list multiple groups per leader if the group times are different (e.g. Luke Skywalker could also be listed for a group at Wednesday 11:10 AM-12:30 PM). You can also list multiple groups that have the same time if the group leader's name is different (e.g. Princess Leia's group time could be the same as Baby Yoda's group time). The group's day and time do not need to be in any format, as long as they match a group time in the students' preferences section in Section 3.

Below all the groups should be a row that starts with ~~Unassigned. Here is where all students who weren't assigned to a group will be listed. Below is an example.

	A	B	C	D
1	~~Group	Luke Skywalker	Luke.Skywalker@unh.edu	Monday 12:10-1:30 PM
2	~~Group	Han Solo	Han.Solo@unh.edu	Monday 3:10-4:30 PM
3	~~Group	Princess Leia	Princess.Leia@unh.edu	Monday 4:10-5:30 PM
4	~~Group	Baby Yoda	Baby.Yoda@unh.edu	Tuesday 12:10-1:30 PM
5	~~Unassigned			

If you need to lock a student in to a group to make sure that they are assigned to that group, add their name and email below that group. You can optionally list

the student's preference for that group, their gender, year, professor, and any notes about that student. These optional values will appear in the output whether you include them or not. Below is an example.

	A	B	C	D	E	F	G
1	~~Group	Luke Skywalker	luke.skywalker@unh.edu	Monday 12:10-1:30 PM			
2	R2 D2	r2d2@unh.edu	Preferred	Male	Sophomore	Chris Bauer	Beep boop
3	C3 PO	c3po@unh.edu	Preferred	Male	Sophomore	Chris Bauer	
4	~~Group	Han Solo	Han.solo@unh.edu	Monday 3:10-4:30 PM			
5	~~Group	Princess Leia	Princess.leia@unh.edu	Monday 4:10-5:30 PM			
6	~~Group	Baby Yoda	baby.yoda@unh.edu	Tuesday 12:10-1:30 PM			
7	~~Unassigned						

Section 2: Parameters

The parameters are how you change what the groups look like. In this section, you will find values that you can set for the program to use.

	A	B
1	Smallest Possible Group Size	4
2	Largest Possible Group Size	10
3	Smallest Preferred Group Size	6
4	Largest Preferred Group Size	8
5	Increase Preferred Group Size Penalty	3
6	Decrease Preferred Group Size Penalty	10
7	Student Non-Preferred Assignment Penalty	2
8	Unassigned Penalty	50
9	Singling Out Male Penalty	0
10	Singling Out Female Penalty	0
11	All Males Penalty	0
12	All Females Penalty	0
13	Singling Out Freshman Penalty	0
14	Singling Out Sophomore Penalty	0
15	Singling Out Junior Penalty	0
16	Singling Out Senior Penalty	0
17	All Freshmen Penalty	0
18	All Sophomores Penalty	0
19	All Juniors Penalty	0
20	All Seniors Penalty	0
21	Time Limit	600

- Smallest Possible Group Size

    - This is absolute minimum number of students a group can contain (besides 0)
- Largest Possible Group Size
    - This is the absolute maximum number of students a group can contain
- Smallest Preferred Group Size
    - This is the smallest group size you would prefer. Groups can be larger than this, or smaller than this up to the Smallest Possible Group Size for a penalty.
- Largest Preferred Group Size
    - This is the largest group size you would prefer. Groups can be smaller than this, or larger than this up to the Largest Possible Group Size for a penalty.
- Increase Preferred Group Size Penalty
    - Here you can set the penalty for increase the size of a group past the Largest Preferred Group Size. For every group, every number past Largest Preferred Group Size will accrue this penalty.

- Decrease Preferred Group Size Penalty
    - Here you can set the penalty for decreasing the size of a group past the Smallest Preferred Group Size. For every group, every number past Smallest Preferred Group Size will accrue this penalty.
- Student Non-Preferred Assignment Penalty
    - This is the penalty for assigning a student to a possible group rather than a preferred group.
- Unassigned Penalty
    - This is the penalty for not assigning a student to any group.
- Singling Out Male/Female/Freshman/Sophomore/Junior/Senior Penalty
    - Groups that have just one male/female/freshman/sophomore/junior/senior in them will be given this penalty. If this parameter is 0, then the program will not avoid singling out these traits in groups.
- All Males/Females/Freshmen/Sophomores/Juniors/Seniors Penalty
    - Groups that are entirely male/female/freshmen/sophomores/juniors/seniors will be given this penalty. If this parameter is 0, then the program will not avoid creating groups with these traits shared between all members.

- Time Limit
    - This parameter is the time limit on the solver. The maximum limit is 600 seconds (5 minutes), and the minimum is 20 seconds.

Section 3: Students

Section 3 is where you put the student information. This is exactly the same format as the simple version.

USING THE PROGRAM

To use the Group Assign program, all you need to do is send an email to *UNHGroupAssign@gmail.com*! The subject can be left blank, or anything you want it to be. The body of the email can be left blank or anything you want it to be. Make sure you attach your csv file in this email and hit the send button.

After your email is sent, our program will receive it and send you an email notification when you have reached the front of the queue. In most cases, moments later you should receive a result with your group assignments. Only in cases where the input contains a lot of students and groups might it take more

than a minute. Additionally, the program will automatically stop at 5 minutes and send back the best results it could find.

If you don't see a response soon, it could be for a few reasons.

- If you used your UNH email, you might need to whitelist UNHGroupAssign@gmail.com to your account in order to receive emails. (https://www.thebalancesmb.com/whitelist-email-sender-3515045).
- If you are using Gmail, Google delays the sending of emails, so you have time to undo a send. Check your Gmail settings to see how long your delay is.

ERRORS

So, you sent in your file and there were errors. Don't worry, they can be fixed. We send back an email saying what parts of the file are wrong so you can change it and send it back. We try to be as specific as possible when sending back error messages, but sometimes it might be confusing as to why you got an error. Here's a list of all errors and potential fixes.

- Group leader name missing
    - Every group in section 1 must start with ~~Group, followed by a group leader's name. This must not be blank.

- Group leader email missing
    - Every group in section 1 must start with ~~Group, followed by a name, and then the group leader's email. This must not be blank.
- Group meeting time missing
    - Every group in section 1 must start with ~~Group, followed by a name, email, and then the group's meeting time. This must not be blank.
- _ is already leading a group at _
    - This means that there were two groups listed in section 1 with the same group leader at the same time. Change either the leader's name, the time the group meets, or remove the duplicate.
- No groups were found in the file
    - You either had an ~~Unassigned section with no ~~Group's before it, or the program got to the end of the file and couldn't find any lines that started with ~~Group. Add ~~Group's to section 1 or use the simple version to save time.
- The "~~Unassigned" section was not found in the file
    - If you have ~~Group's listed in section 1, add a line below the last group that starts with ~~Unassigned. See the template file for an example.

- No parameters found in the file
    - The program reached the end of the file and couldn't find any parameters. If you got this error in conjunction with "No groups were found in the file" or "The "~~Unassigned" section was not found in the file", it is likely because the program reached the end of the file already, and this problem will go away when the first problem is fixed.
- Parameter value for _ is missing
    - Every parameter in section 2 starts with the name of the parameter, followed by a positive integer value. This must not be blank.
- Parameter value for _ must be a number
    - Every parameter in section 2 starts with the name of the parameter, followed by a positive integer value. This must be a number (e.g. 1), and not spelled out (e.g. one).
- Parameter value for _ must not be negative
    - Every parameter in section 2 starts with the name of the parameter, followed by a positive integer value. This must not be a negative number (0 is not considered negative).

- Smallest Possible Group Size must be smaller than the other group sizes
    - Smallest Possible Group Size parameter must be smaller than Smallest Preferred Group Size, Largest Preferred Group size, and Largest Possible Group Size.
- Largest Possible Group Size must be larger than the other group sizes
    - Largest Possible Group Size parameter must be larger than Smallest Possible Group Size, Smallest Preferred Group Size, and Largest Preferred Group Size.
- Smallest Preferred Group Size must be larger than the Smallest Possible Group Size
    - Change the parameters' values to fix this error
- Smallest Preferred Group Size must be smaller than Largest Preferred/Possible Group Sizes
    - Change the parameters' values to fix this error
- Largest Preferred Group Size must be larger than Smallest Preferred/Possible Group Sizes
    - Change the parameters' values to fix this error
- Largest Preferred Group Size must be smaller than Largest Possible Group Size
    - Change the parameters' values to fix this error

- Invalid Parameter Name _
    - Your file had a parameter name spelled incorrectly, or an extraneous line was placed in the parameters section. Check the template file for reference. This error can also occur if not all of the parameters are found, and the program reaches the end of the file trying to find the missing parameters.
- Missing Parameters: [_, …, _]
    - If the program reaches the end of the file because not all of the parameters were found, this error will list which parameters were missing from section 2.
- Section 3 not found in file
    - This error occurs when the program reaches the end of the file without finding the header line. The header line must start with First Name, Last Name, Email, Gender, Year, Professor Name, and Notes. If any of these are spelled wrong, you will see this error. If you can't figure out what's wrong, try copy and pasting the header line from the template file into your file and try again.

- Duplicate time _ in preferences
    - If your header row in section 3 has repeat column names for group time preferences, this error will tell you which time has been repeated.
- Group time _ missing associated preference column
    - When this error occurs, it means that you have a ~~Group listed in section 1 that meets at time _, and there is no column in the header row in section 3 that has the same time. Either remove the ~~Group row from section 1 or add a column for that time in section 3.
- Preference time _ has no associated group
    - The opposite of the error above, this error occurs when you have a time listed in the header row in section 3 that doesn't appear in any ~~Group's in section 1. To fix this, either remove the column from section 3 or add a ~~Group to section 1 with this time.
- Student name missing
    - This error can occur for two reasons: either a student row that appears under a group in section 1 doesn't start with a name, or a student row in section 3 doesn't have a first and last name listed under the First Name and Last Name columns. These cells must not be blank.

- Student email missing
    - This error can occur for two reasons: either a student row that appears under a group in section 1 doesn't have an email after the student's name, or a student row in section 3 doesn't have an email listed under the Email column. These cells must not be blank.
- Student email '_' repeated
    - Every email listed in the Email column must be unique. If a student email is repeated, it is likely that a student filled out the Google Form twice. Remove the duplicate row, or change one of the duplicate emails.
- Student preference <_> is invalid
    - Student preferences must be either "Preferred", "Possible", or "Impossible". This can be left blank to represent impossible, but any other misspellings of these words will not be accepted.