THESIS

A CONSTRAINED OPTIMIZATION MODEL FOR PARTITIONING STUDENTS INTO COOPERATIVE LEARNING GROUPS

Submitted by

Matthew Alan Heine

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2016

Master's Committee:

Advisor: Michael Kirby

Olivier Pinaud
Kimberly Henry

ABSTRACT

A CONSTRAINED OPTIMIZATION MODEL FOR PARTITIONING STUDENTS INTO COOPERATIVE LEARNING GROUPS

The problem of the constrained partitioning of a set using quantitative relationships amongst the elements is considered. An approach based on constrained integer programming is proposed that permits a group objective function to be optimized subject to group quality constraints. A motivation for this problem is the partitioning of students, e.g., in middle school, into groups that target educational objectives. The method is compared to another grouping algorithm in the literature on a data set collected in the Poudre School District.

TABLE OF CONTENTS

1 Introduction

Arranging students into groups for various purposes is commonly done throughout the realm of education. There are many reasons to form such groups, but we will focus on those where friendship formation is the primary concern. For example, some researchers wish to address the problem of school disengagement by promoting healthy relationships for at-risk students. Groups are formed to do team-building activities with the goal of maximizing friendship formation, and thus raising the level of school bonding. In order to achieve this goal, one would need to have way to measure the likelihood of friendship formation for a set of students. In this paper, we use information gathered about the students' values and interests, and make the assumption that similarity in values and interests implies a higher probability of friendship formation.

One may also wish to place additional constraints on how these groups are formed. For example, in the group formation problem that spurred the development of the methods described in this paper, it is required that groups contain no more than one student with disciplinary history. There are many such constraints that are required for the project in question. Several methods are currently employed for finding such groups but, in contrast to the approaches proposed here, they have limited ability to handle constraints.

Given information about the social network and student values and interests profiles, one can do a simple 'nearest neighbor' approach as described by [2]. They developed a method with the goal of raising the level of school bonding of students that are identified as at-risk by forming groups with high probability of friendship formation. The algorithm described is for forming groups of size four. The at-risk students are held back from joining groups until the final step. First, students are selected to seed the groups. Then, pairs are formed one at a time by identifying the best match between the seeds and the remaining students. A third student is added to each group in the same way. Finally, the at-risk students are added to the groups in the same way. They also tried a slight variation to this algorithm, where instead of systematically picking the best matches, during each round they eliminated the

worst ones one at a time until the next additions were determined. In this algorithm, the only type of constraint that is implemented is the separation of certain subsets of students. In fact, they can separate at most four types and it's unclear what to do if a student counts as more than one of these types.

A more sophisticated method was developed by [5] and has been implemented as an open source web-based tool known as Teammaker. However, extensive changes were required to be applicable to the problem studied in this paper. In fact, we were able to modify Teammaker to fit the problem at hand and increase its speed significantly. For this algorithm, a score function is developed to rank the quality of a particular group. The score function contains all of the information about values and interests as well as punishments for the constraints. The algorithm involves first grouping the students randomly and then performing pair swap operations between groups. A swap is kept if it improves the minimum score of the two groups in question. By placing the constraints in the objective function, there is no guarantee that the solution will satisfy them, so each constraint must be checked. Also, the algorithm doesn't converge to the global maximum. It will instead converge to some local maximum determined by the initial conditions. Thus different starting conditions must be explored to ensure a high quality solution.

In this paper we present a new two-phase method for forming student groups subject to a wide variety of constraints. In phase 1, we apply a technique called latent class analysis (LCA) to identify abstract classes of students based on the values and interests data. Each student is assigned a probability of membership to each class. In phase 2, we formulate a binary integer program to form the groups. The objective function is crafted from the probabilities that come from the LCA. We will see that a wide variety of constraints for grouping problems, can be transformed into linear constraints on this binary integer program. The advantages of the proposed method over prior algorithms include its ability to handle a large number of constraints with ease, a guarantee that constraints will all be satisfied on the first try, even if we have to exit the optimization early due to time restrictions, and the objective function has a nice interpretation to intuitively understand the quality of a

solution. We will also describe the modifications made to Teammaker in order to have an alternative method to solve the same problem.

In Section 2, we investigate the principle assumption used in constructing optimal groups. In Section 3, we describe the new integer programming approach to define and search for an optimal grouping. In Section 4, we describe a variation of (teammaker citation) Teammaker method to give a slightly different definition of an optimal grouping and to find a solution. In Section 5, we compare the two methods and evaluate the pros and cons of each. Finally, in Section 6, we summarize the contents of the paper and make suggestions for future research.

2 Preliminary Analysis

2.1 Causality Assumption

Before we begin to develop a method that groups middle school students based on individual values and interests, it would be prudent to first examine the assumption that students with similar values and interests are more likely to become friends. We have a sample data set of 267 students that contains an attribute matrix and a friendship matrix. The attribute matrix contains the results of a survey where students ranked 27 values and interests on a scale of 1 (not important) to 5 (very important). Element A_{ij} in the friendship matrix is a value between 0 and 5 indicating how strong of a bond student i feels with student j. A value of 0 indicates that no bond is present and a value of 5 indicates a very strong bond.

To evaluate our assumption that friendships are formed between students with similar attributes, we will check how many friendships are predicted by a latent class clustering and compare this to an unrestricted k-means clustering and a random clustering. An R package called 'mclust' [1] allows us to obtain the probabilities of membership for each student to any given number of latent classes. We can then assign each student to his most likely latent class. R has a built-in function, 'kmeans' and for the random clustering, we can use the nested functions 'floor(runif()).'

Once we have a clustering, we can check to see how many pairs of friends were placed in the same group. We define a friendship between students i and j to mean that each student gave the other a score of at least 4 in the friendship matrix. Since the k-means algorithm is not optimal and the outcome depends on the arbitrarily chosen starting point, we will run it many times and average the results. Also, we need a baseline with which to compare these clusterings. For this reason, we will also create a randomly generated clustering of the students into k groups and compute the number of predicted friendships. If our assumption is valid, then the latent class clustering and k-means clustering should consistently predict more friendships than a random clustering.

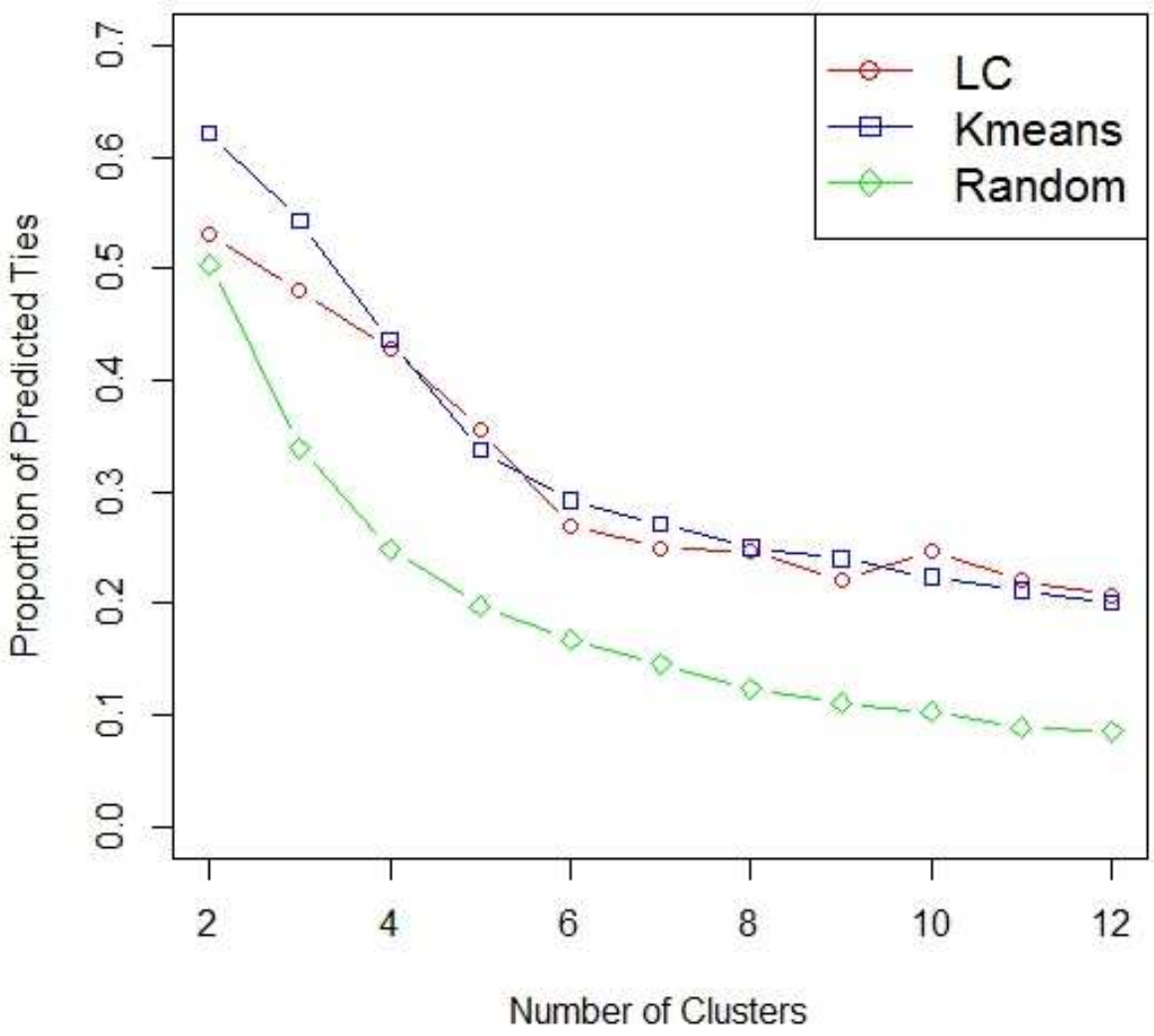

Figure 1: Friendships predicted by different clustering methods

Figure 1 shows the result of this analysis on our sample data set. For each k we took 50 each of k-means and random clusterings and averaged the proportions of predicted friendships. It is clear in the plot that latent class clustering and k-means clustering are consistently better at predicting friendships than random clustering. We conclude that similarity in values and interests is correlated with friendship. The question of causation remains unresolved because we have not determined if similarity induces friendship or if friendship induces sim-

ilarity. This analysis does not rule out the possibility that similarity induces friendship so we will continue to make that assumption.

It is not clear from Figure1 that latent class clustering advantageous over k-means. However, we are comparing one latent class clustering to an average of many k-means clusterings. The value of any individual k-means clustering can vary significantly. Also, k-means does not have a nice way of enforcing constraints. The LC probability matrix allows us to design a constrained optimization problem whose objective function has a nice interpretation. In addition, latent class analysis has been found to significantly outperform k-means at identifying subgroups in a population [4].

2.2 Combinatorial Complexity

Before searching for solutions to the problem, it would be enlightening to know the approximate size of the space we must search. Consider the simplified problem: How many ways are there to partition N students into groups of size 4, assuming $4|N$?

Working this problem out, we find that the number of partitions is given by $N!/(24^{\frac{N}{4}}(\frac{N}{4})!)$. The numerator grows much faster than the denominator, therefore this number blows up rapidly. In fact, even for the relatively small problem of $N = 40$, we find that there are 3.5×10^{27} possible partitions.

3 The Optimization Problem

Essentially, we are faced with the problem of grouping students in a way that maximizes compatibility subject to constraints. A natural approach is try to phrase the problem in the language of optimization. We define the decision variable, x, as follows:

$$x_{ig} = \begin{cases} 1 & \text{if student } i \text{ should join group } g \\ 0 & \text{otherwise} \end{cases}$$

After transforming x into a column vector, we can write the optimization problem:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & f(x) \\ \text{subject to} \quad & x \in \Omega, \end{aligned}$$

where Ω is the set of feasible solutions to the problem as determined by the set of constraints.

3.1 Objective Function

We need to find an objective function that allows us to maximize friendship formation based on values and interests. Student values and interests have been characterized using latent class analysis. The result of this analysis is a set of latent classes and probabilities of membership for each student where we define p_{ij} to be the probability that student i belongs to latent class j.

For each group to be formed, we assign a latent class. Let $\phi : \{\text{groups}\} \mapsto \{\text{classes}\}$ be such a map. Then we define the group membership weights as follows:

$$c_{ig} = p_{ij} \quad \text{where} \quad \phi(g) = j.$$

The objective function will naturally be a linear combination of the elements of x with these probabilities as coefficients, and the value of the objective function will be the expected number of students that are in a group corresponding to their respective latent classes:

$$f(x) = \sum_i \sum_g c_{ig} x_{ig}.$$

We are now faced with the problem of how many groups to assign to each latent class. We would like this to be proportional to the expected number of students belonging to each class. The expected number of students that belong to class j is given by

$$E_j = \sum_i p_{ij}$$

Using these expected values as proportions, we can apportion the groups to the latent classes. A perfect apportionment method is mathematically impossible, so we have many methods to choose from at this step. The Huntington-Hill method is simple and usually avoids the major pitfalls of apportionment so it is what we will use.

Let N be the number of students and k be the number of groups to be formed. The first step is to compute the divisor,

$$D = \frac{N}{k}.$$

Next, we need to find the population of each latent class. Students were not each mapped into a single latent class. We instead have a probability distribution for each student's membership in each class. So for the population of a latent class, we will use the expected number of students it contains, E_j. Now we compute the quota for each class,

$$Q_j = \frac{E_j}{D},$$

and the lower quota,

$$L_j = \lfloor Q_j \rfloor.$$

Finally, we compare Q_j to the geometric mean of L_j and $L_j + 1$ to determine the number of groups to assign. If $Q_j > \sqrt{L_j(L_j+1)}$, then we assign $(L_j + 1)$ groups to latent class j. Otherwise, it receives L_j groups.

3.2 Basic Constraints

We now need to find the feasible set of solutions that satisfy the requirements set forth for the groupings. There are two basic constraints that must always be included for the solution to make sense. First, each student must join exactly one group. This is easily accomplished with the linear constraints

$$\sum_g x_{ig} = 1, \quad \forall i.$$

The other basic constraint is on the group sizes. Let α and β be the minimum and maximum allowable group sizes. Then we have a set of linear constraints given by

$$\alpha \leq \sum_i x_{ig} \leq \beta$$

3.3 Separation Constraints

In the project for which this method was developed, one of the requirements was that no group may contain more than one student that was at risk for school disassociation. These students were identified by low school bonding scores that were calculated from their responses to survey questions. Let S be the set of indices corresponding to these students, and we have another linear constraint

$$\sum_{i \in S} x_{ig} \leq 1, \quad \forall g.$$

This constraint can easily be generalized to separate other sets of students by choosing a different index set. Also, one could change the relationship or the right-hand side accordingly. For example, another constraint that was required for this project was to promote diversity

among the groups. If we let S be the set of indices corresponding to a category of students that we want to be represented in each group, then we have

$$\sum_{i \in S} x_{ig} \geq 1, \quad \forall g.$$

3.4 Balance Constraints

We were asked to balance quantities such as GPA, level of school bonding and popularity across the groups. For each of these categories, each student has a score. In order to promote balance, we have designed a constraint that allows us to restrict the average score for each group to be within an interval centered on the average score of the whole population. In order to do this, we introduce a parameter ϵ that controls how tightly we constrict the group averages to the population mean. Let S_i be the score for student i. We can easily compute μ and σ_S, the mean and standard deviation of the population. We may then write constraints of the form

$$\mu_S - \epsilon\sigma_S \leq \frac{\sum_i S_i x_{ig}}{\sum_i x_{ig}} \leq \mu_S + \epsilon\sigma_S, \quad \forall g.$$

A little manipulation and we can write this as two nice linear constraints:

$$0 \leq \sum_i (S_i - \mu_S + \epsilon\sigma_S) x_{ig}, \quad \forall g,$$

$$0 \geq \sum_i (S_i - \mu_S - \epsilon\sigma_S) x_{ig}, \quad \forall g.$$

3.5 Gender Constraint

It is considered developmentally inappropriate to have just one boy or just one girl in a social grouping for middle school aged students. Students are more likely to befriend others of the same gender. This leads to the requirement that no group to contain exactly 1 male or exactly 1 female. Equivalently, each group must contain 0 males or at least two males,

and 0 females or at least two females. If we let M be the index set of males and F be the index set of females, then what we need are

$$\sum_{i \in M} x_{ig} = 0 \quad \text{or} \quad \sum_{i \in M} x_{ig} \geq 2, \quad \forall g$$

and

$$\sum_{i \in F} x_{ig} = 0 \quad \text{or} \quad \sum_{i \in F} x_{ig} \geq 2, \quad \forall g.$$

We would like to have all of our constraints in a linear form and the 'or' is preventing this. To circumvent the issue, we will introduce two binary variables for each group, $B_{m,g}$ and $B_{f,g}$. For the males, we introduce the following constraints:

$$\sum_{i \in M} x_{ig} + \beta B_{m,g} \geq 2, \quad \forall g,$$

$$\sum_{i \in M} x_{ig} - \beta(1 - B_{m,g}) \leq 0, \quad \forall g.$$

To see why this will work, plug in both possibilities for $B_{m,g}$ and reduce. If $B_{m,g} = 0$, then we have

$$\sum_{i \in M} x_{ig} \geq 2, \quad \forall g,$$

$$\sum_{i \in M} x_{ig} \leq \beta, \quad \forall g.$$

The first is the same as the expression right of the 'or.' The second is already true from the group size constraints. On the other hand, if $B_{m,g} = 1$, then the constraints reduce to

$$\sum_{i \in M} x_{ig} \geq 2 - \beta, \quad \forall g,$$

$$\sum_{i \in M} x_{ig} \leq 0, \quad \forall g.$$

The first is redundant from the constraint that $x \geq 0$ and the fact that $\beta \geq 2$ in practice. Since the sum cannot be negative, the second is equivalent to the left side of the 'or.'

3.6 Computation

Since the objective function and all of the constraints can be written as linear combinations of binary variables, we can easily write them in vector/matrix form. It is assumed that the reader is somewhat familiar with linear programming. However, this problem has the additional constraint that the variables are binary. What we really have is an integer program, or more specifically, a binary integer program. For this reason, the standard techniques applied to solving linear programs are insufficient. Integer programming, on the other hand, is generally a much slower procedure. The most common classes of algorithms are cutting plane methods and branch and bound methods. Each of these requires solving a large number of linear programs that are formed from relaxing the integrality constraints. The size of our problem is given by $n =$ (Number of Students) $\times$ (Number of Groups). The worst case scenario for solving an integer program with these methods, is that we solve a linear program of size n for every possible solution. The complexity of these integer programming algorithms is exponential in n, however they tend to behave much faster in practice.

Gurobi [3] is a well known commercially available solver that has a free license for academic use. It utilizes state-of-the-art techniques for solving optimization problems such as this. It is also available on many platforms including R. Since R also has nice packages for latent cluster analysis, it is a natural choice for this project.

3.7 Results

This method was used to group 149 students from our sample data set into teams of size 4 or 5. A parameter ϵ was used for three different balance constraints. We first arbitrarily

set $\epsilon = 1$ and show the results. Each student has a unique ID number. The code creates a text file where each line contains 4 or 5 ID numbers that represent a team (see Appendix for example output). The code also stores useful information like the objective function value and the runtime.

Gurobi [3] found the optimal solution to the integer program in 22.93 seconds. The value of the optimal solution is about 132. Our interpretation is that out of the 149 students, we expect to find that 132 of them were placed in groups corresponding to their actual latent classes. We define the quality of a solution to be $q = \frac{V}{N}$, where V is the value of the program, and N is the number of students. For this example, we have $q = \frac{132}{149} = 0.886$.

If $q \leq 0.5$, then less than half of the students are placed in groups corresponding to their latent classes. So on average, less than half of the students in any given group actually belong to that group's latent class. Thus the identification of that group with a particular latent class no longer holds meaning. This is clearly a problem and it likely means that there are too many constraints on the solution. The more constraints present in the system, the lower the quality of the resulting solution.

A great illustration of this relation can be shown by varying the parameter ϵ used in the balance constraints of Section 3.4. If we make ϵ smaller, then we are reducing the size of the solution space and therefore expect the quality of the solution to diminish. We have run the code for a variety of ϵ's and plotted the results in Figure 2. Notice the expected behavior, that as ϵ decreases, so does q. We also notice that when ϵ reaches about 0.9, the quality becomes constant. This reflects the fact that for large enough ϵ, the balance constraints will not be binding, and therefore will not affect the size of the solution space.

We should also see how ϵ affects the computational time. In Figure 3, we have plotted the run times for each ϵ. First we notice that for small ϵ, the run time is constant. This is because the code has a time limit at which it terminates and returns the best solution it has so far. In this case, the solution returned is not the optimal solution. The time limit in this case was 10 minutes.

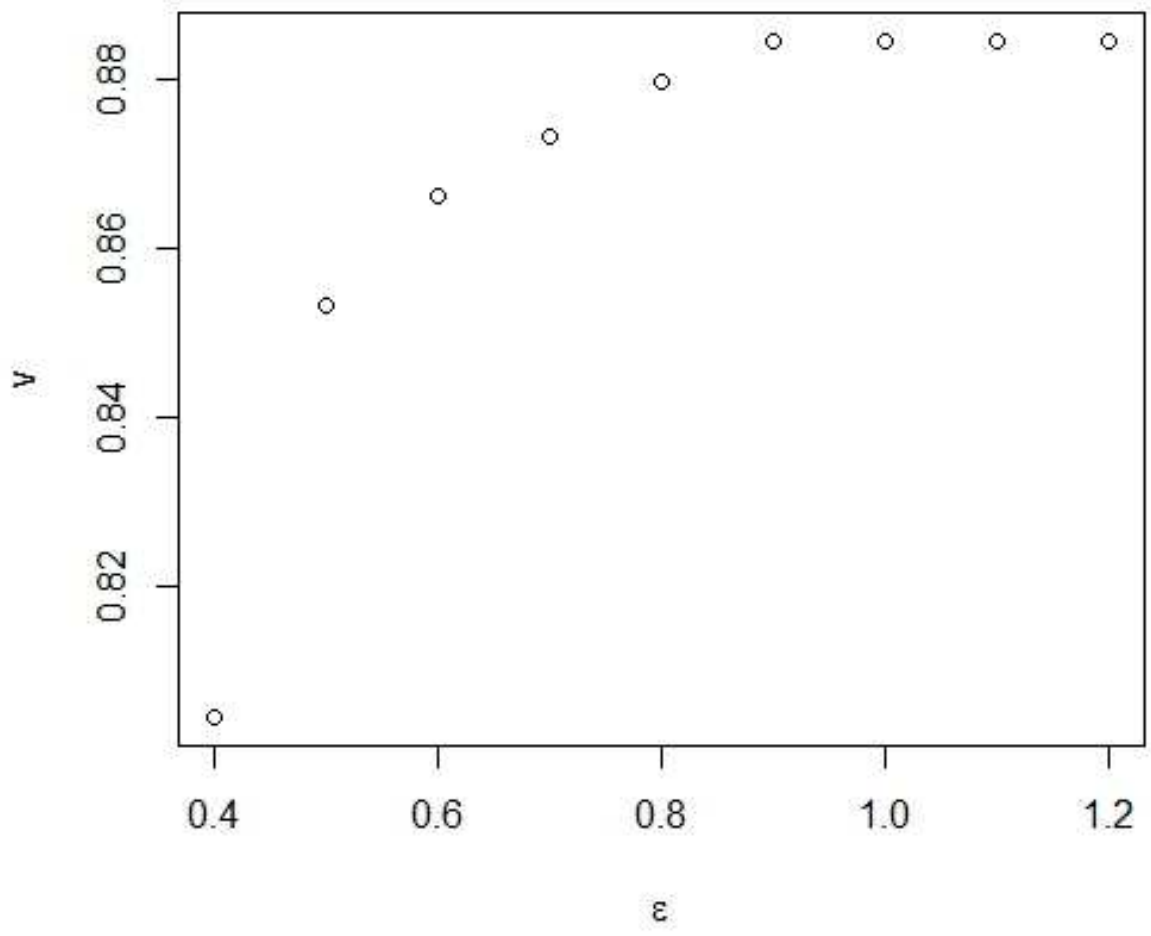

Figure 2: Value of solution found by Gurobi [3] for various ϵ's. The parameter ϵ controls the tightness of the balance constraints from Section 3.4.

The plot implies that as we make ϵ smaller, the time required to find the optimal solution increases. This is quite counterintuitive because we would expect that a smaller solution space would mean that it's easier to find the best solution. This interesting phenomenon merits further study. Notice however, that even cutting it off at 10 minutes, we are still able to find a pretty good solution. Referring to Figure 2, when $\epsilon = 0.4$, the value of the solution is only about 10% lower than the optimal solution found for $\epsilon = 1$. Since we expect the value

to be smaller anyway, this shows that the cost of terminating early due to time constraints is not devastating.

Finally, we note that for large enough ϵ, the run time flattens out again. This is another effect of the constraints no longer being binding. In other words, an increase in ϵ will not change the solution space, and therefore the time spent searching it is the same.

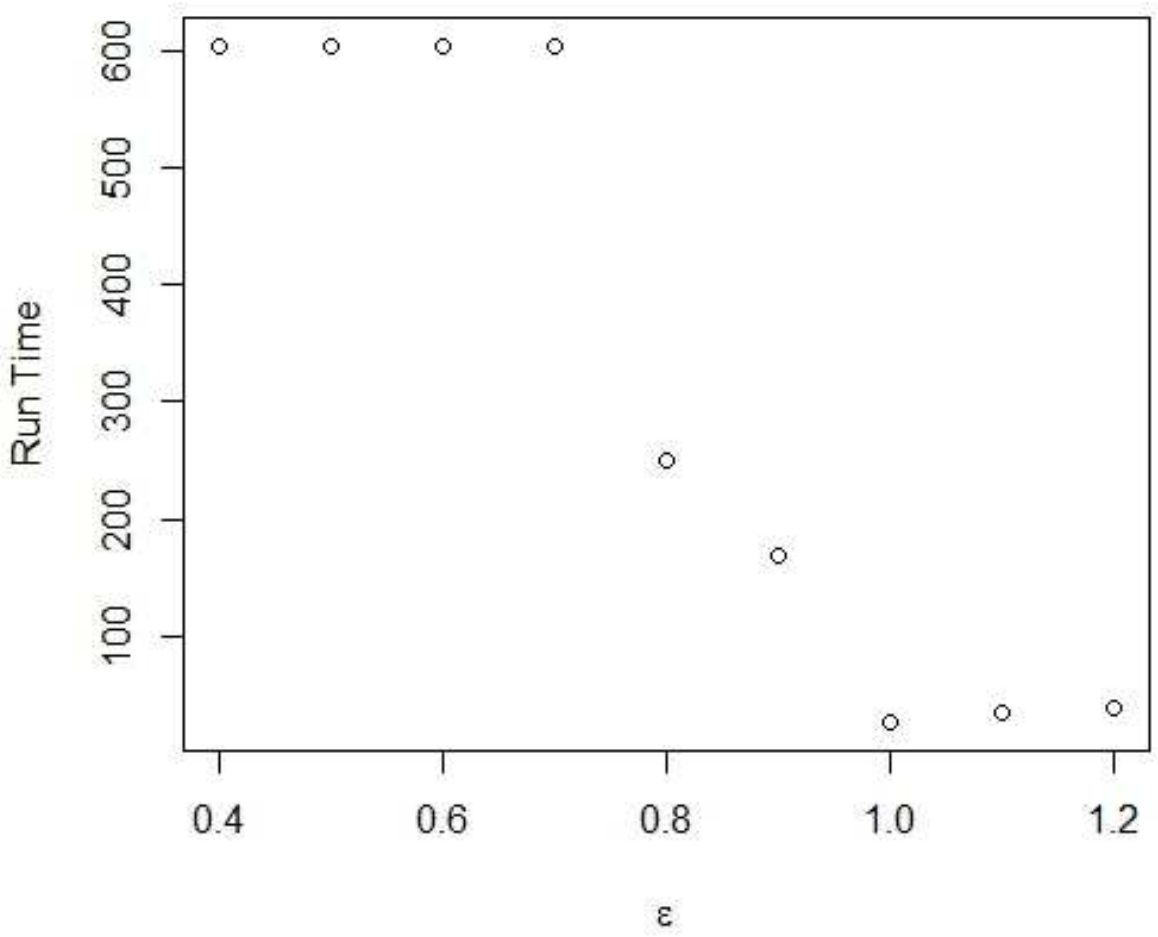

Figure 3: Time required for the Gurobi solver [3] to find the solution for various ϵ's. The parameter ϵ controls the tightness of the balance constraints from Section 3.4.

4 Teammaker

The Teammaker method developed by [5] also models group formation as an optimization problem. By reformulating their score function, we were able to adapt the procedure to solve an optimization problem with the same constraints described in Section 3. This allows us to compare the Integer-Programming method to the Teammaker method. In addition, we propose a modification to the Teammaker algorithm that results in much faster convergence without sacrificing the quality of the solution. In this Section, we will describe explicitly the modified Teammaker method for solving the optimization problem from Section 3. While we use a small, but important modification to the actual algorithm to boost speed, our objective function is quite different from the one presented in [5].

For the Teammaker approach, the score function is defined to evaluate the quality of a single group. Students are initially placed in groups randomly. Then an iterative pair swapping procedure takes place. The goal in this optimization is to maximize the minimum group score. Unfortunately, this algorithm results only in a locally optimal solution. To achieve a better solution, one must explore many different initial conditions. Also, since the constraints are placed in the objective function, we can use a parameter to increase the likelihood that they are satisfied, but the only way to know for sure is to check the solution.

Mathematically, a grouping can be described as a partition, P, of the set of students with the added requirement that each element, P_i satisfy the group size criteria. The optimal solution is defined to be the partition with the largest possible score for its lowest scoring member. This problem can be stated as

$$\begin{aligned} &\max_{P} \ \min_{i} \ \ Score(P_i) \\ &\text{subject to} \ \ \alpha \leq |P_i| \leq \beta, \end{aligned}$$

where α and β are the constraints on group size.

4.1 The Score Function

We will develop our score function that will be used to measure the quality of any given group. The score function will have several terms. First, we must have a term that scales with the similarity in values and interests of the group members. Next, we must try to incorporate all of the constraints of the problem. The Basic Constraints of Section 3.2 are satisfied by construction. The remaining constraints will be incorporated as penalty terms that will be negative when a constraint is violated, and 0 otherwise. Our score function is formulated similarly to the one in [5], but adapted to suit the problem at hand.

The survey data from this project consisted of 27 values and interests that students rated on a scale of 1 to 5 (see Appendix C). For each question, we can compute the variance in the responses provided by the students. Smaller variance implies that the students are more similar. Since we wish to maximize similarity, we will add a term to the score function that is the negative of the sum of the variances. Let s_i^2 be the variance in the responses for question i for the group. To be more general, let m be the number of survey questions. Then the attribute term can be calculated as

$$S_A = -\sum_{i=1}^{m} s_i^2.$$

Terms for the separation constraints can easily be added. Let Φ be a set of students that must all be placed in separate groups. Let n be the number of students from Φ that are in the group. If $n > 1$, then a constraint is violated, which must be punished in the score function. In fact, the larger n is, the more seriously the constraint has been violated. This leads us to the following term:

$$S_\Phi = \begin{cases} -n & \text{if } n > 1 \\ 0 & \text{otherwise} \end{cases}$$

For the balance constraints, we want to bring the group average for a particular quantity as close to the population average as possible. This leads us to write

$$S_B = -\frac{|\bar{x} - \mu|}{\sigma},$$

where $\bar{x}$ is the group mean, μ is the population mean and σ is the population standard deviation. Groups whose average lies far from the population average will receive a large penalty in the score function.

For the gender and ethnicity constraints, we need to penalize groups that have exactly 1 male, exactly 1 female or exactly 0 members of the category we want represented in each group. Let a be the number of males in the group, then we naturally have

$$S_{GE} = \begin{cases} -1 & \text{if } a = 1 \\ 0 & \text{otherwise} \end{cases}.$$

Scores for the female and ethnicity requirements can be written in a similar fashion.

Finally, we wish to consolidate all of these pieces into a single *Score* function. To be as general as possible, and for simplicity because our problem has several constraints of each type, we label each of the constraints with an index: S_i. For each constraint term, we introduce a positive constant, γ_i. We want to choose the γ_i's large enough that the solution is likely to satisfy the constraints. For this problem, it was found that $\gamma_i = 100 \ \forall i$ was sufficient to satisfy all of the constraints most of the time. However, given the random nature of the algorithm, one should always check the solution to ensure that no constraint is violated. Putting all of the terms together, we have

$$Score = S_A + \sum_i \gamma_i S_i.$$

4.2 Swapping Algorithm

Now that we have a metric, *Score*, to evaluate a group, we can begin the optimization process. Our goal is to maximize the score of the lowest scoring team in the grouping. Algorithm 1 outlines the iterative pair swapping procedure described in [5]. Two teams are

selected as well as one student from each group. The *Score* function is evaluated for each group before and after swapping. If the change increases the minimum score, then the swap is kept, otherwise it is reverted. In one iteration, every possible swap is tried. We continue until reaching an iteration where no swaps are kept.

Algorithm 1 Original algorithm [5]

```
1: while #swaps > 0 do
2:     for i in 1 : M − 1 do
3:         for j in i : M do
4:             for a in Team(i) do
5:                 for b in Team(j) do
6:                     oldScore ← min{Score(Team(i)), Score(Team(j))}
7:                     swap a ⟷ b
8:                     newScore ← min{Score(Team(i)), Score(Team(j))}
9:                     if newScore < oldScore then
10:                        swap b ⟷ a
11:                    end if
12:                end for
13:            end for
14:        end for
15:    end for
16: end while
```

Clearly, the combinatorics of this algorithm are not favorable; however, the code runs in a reasonable amount of time for typical N and M in the context of grouping middle school students. In our sample data set, $N = 149$ and $M = 37$. An average run on Algorithm 1 takes around 12 minutes. We can improve the speed of the algorithm significantly if, rather than trying every pair of teams on a given iteration, we instead focus on improving the worst team. This leads us to Algorithm 2, where we identify the lowest scoring team and try all possible pair swaps. This reduces the number of operations by about a factor of $\binom{M}{2}/M$, a significant amount as M gets larger.

4.3 Results

Unfortunately, the value of the score function does not have any meaningful interpretation, other than that it provides a metric with which to compare groups. We define the value of a grouping to be the score of its lowest scoring group, since this is what we were trying to maximize with the hill climbing algorithm. To see how this method performs and to

Algorithm 2 Improved algorithm

```
while #swaps > 0 do
    worstTeam ← min_i Score(Team(i))
    for j in 1 : M, j ≠ index(worstTeam) do
        for a in worstTeam do
            for b in Team(j) do
                oldScore ← Score(worstTeam)
                swap a ⟷ b
                newScore ← min{Score(worstTeam), Score(Team(j))}
                if newScore < oldScore then
                    swap b ⟷ a
                end if
            end for
        end for
    end for
end while
```

demonstrate the speed boost provided by using the improved algorithm, we have run the each algorithm 20 times with random initial conditions. The results are shown in Figure 4.

First, we notice that Algorithm 2 is indeed much faster than Algorithm 1. The average run-times were 736 seconds and 73.3 seconds, so it is about 10 times faster. Such a big increase in speed begs the question of what are the trade-offs. There was concern that by rushing the hill climbing algorithm, we may not explore the solution space as thoroughly, and result in a lower value solution. We found that this does not seem to be the case. The best solution found by each algorithm had value 468.25. We also see that the times for Algorithm 1 vary significantly, whereas Algorithm 2 is much more consistent.

We see that most of the solution values lie close to the maximum. Both algorithms had outliers that were far below the rest. These are examples of solutions that violated 1 or more constraints. Increasing the γ_i's would reduce the probability that constraints are violated. We have learned from this example, that although the solutions are not necessarily optimal, we do not have to run the code very many times to obtain a close approximation to optimality.

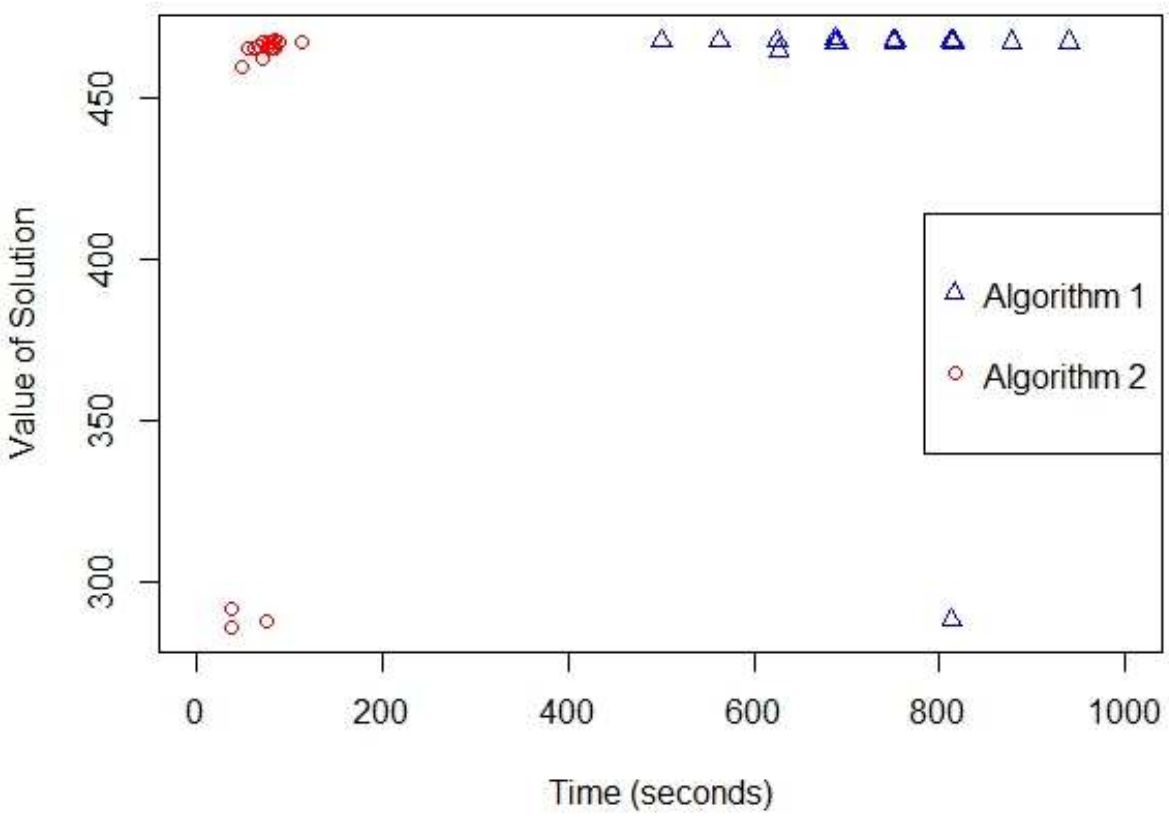

Figure 4: Results from running the both the original and improved algorithms. Each point represents a solution. On the x-axis, we have the amount of time required for the algorithm to terminate, and on the y-axis, we have the score of the minimum scoring group from the solution.

5 Comparison

Thus far, we have developed a new method for grouping students using integer-programming techniques, and we have modified an existing method to handle the same types of constraints. In this final section, we will compare the two methods and evaluate the pros and cons of each.

We expect to find that each method performs best under its own metric. We will first take a look at how the integer programming solution compares to the Teammaker solution when evaluated by the score function from Section 4.1. As we have seen, the balance constraints of Section 3.4 require us to choose a parameter ϵ that controls the strength of the blanance constraints. This poses a problem when we look to compare with the Teammaker method that has no such parameter. In the interest of having a fair comparison between the methods, we will throw out the balance constraints for this analysis.

We ran the Algorithm 2 with 5 different initial conditions and chose the solution with the highest minimum group score. We then found the integer program solution with balance constraints removed. Next, we evaluated the score function for every group in each solution. A histogram of this data is found in Figure 5.

First notice that by this metric, Teammaker clearly outperforms the Integer program. The minimum group scores were -51.4 and -32.4. However, a slightly different way to examine this is to compare the average group scores. Teammaker has an average group score of -26.5, and the integer program's is -29.4. Teammaker still comes out ahead, but these numbers are much more comparable. Looking at the histogram in Figure 5, we see that the scores for Teammaker are clustered tightly around the mean, whereas the scores for the integer program are much more spread out. This is a result of the fact that the Teammaker algorithm seeks to make the worst group as good as possible, while the integer program is maximizing its metric over all the groups simultaneously.

Now we evaluate the Teammaker solutions using the objective function from the integer program. Since the integer program finds the optimal solution (assuming that it didn't run

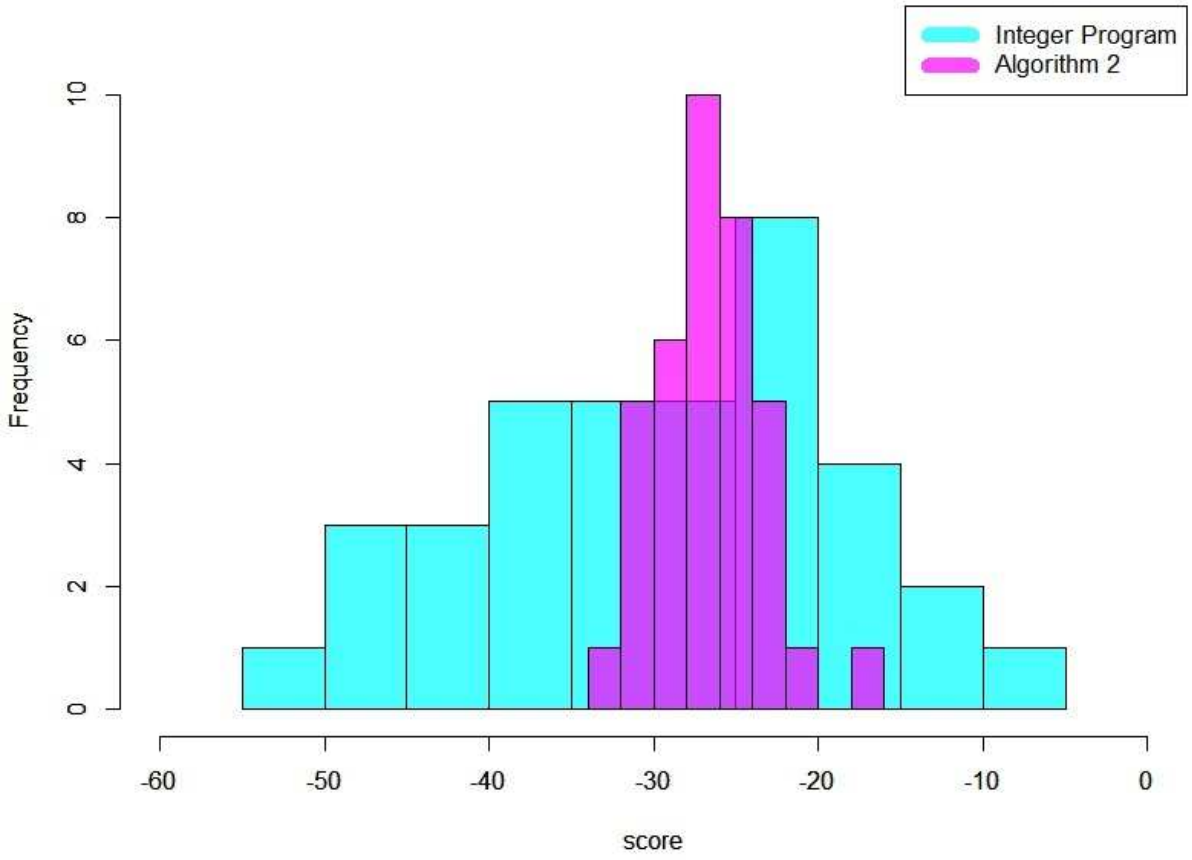

Figure 5: The *Score* function, defined in Section 4.1, was used to evaluate every group in each of the two solutions.

out of time), we know for a fact that the Teammaker solution cannot be better. In addition, the objective function does not contain any information about the constraints. It uses only the values and interests data, processed with LCA. An important part of the definition of the objective function was that each group was assigned a latent class. In order to compare the Teammaker solution, we will identify the best latent class for each group, and use it to evaluate the objective function.

With balance constraints removed, we find that the quality of the integer programming solution is $q = 0.885$. We ran Teammaker with 10 different initial conditions and computed the quality for each solution. We found an average quality of $\bar{q} = 0.623$ and a maximum quality $q_m = 0.653$. These numbers are lower than the quality of the integer programming solution as expected. They are greater than 0.5, so thinking of the groups in terms of latent classes does seem to make sense; however, recall that even with a highly restrictive balance constraint and a early timeout, the integer program found solutions above 0.8. This is evidence that the approaches used to promote similar attributes in the groups are fundamentally different.

In terms of computational time, the integer program clearly has the advantage. A single run of Algorithm 2 took more than 3 times longer than the integer program to find a solution. In addition, Algorithm 2 must be run several times in order to make sure that the solution is of high quality. More data is needed to see how these methods scale with the size of the problem. Another advantage of the integer program is that we have a nice interpretation of the objective function that gives immediate insight into the quality of the solution. The downside is that state-of-the art integer programming techniques are fairly complicated, and one may prefer to use one of the available solvers, such as Gurobi [3]. For applications outside the realm of acadamia, this could get expensive. On the other hand, Algorithm 2 and the definition of the *Score* function from Section 4.1 are easy to understand and program from scratch. Finally, it is difficult to compare the quality of the solutions generated from the two methods because of how differently they treat the optimization. While the integer program

tries to optimize the whole grouping at once, the teammaker method seeks to make the worst group as good as possible.

6 Conclusions and Summary

In this paper, we have described two methods for partitioning students into cooperative learning groups. The first was a binary integer program designed from scratch to solve a specific grouping problem, but kept as general as possible. The second was modified from an existing method presented by [5] to solve the same grouping problem. The two methods were then compared. It was found that the integer program was able to find a solution faster than the Teammaker approach, but is a far more complicated algorithm, requiring an LCA package, mclust [1], and a solver, Gurobi [3]. Finally, a solution to the integer program is guaranteed to satisfy all of the constraints. The same is not true for a Teammaker solution, so we must always check the solutions to verify the constraints.

We recommend further study to better understand how these methods scale with larger problems. More student data should be collected or artificially generated and tested with both methods. It is also worth exploring the possibility of using a more sophisticated algorithm with the Teammaker method. A genetic algorithm or simulated annealing may prove to be more efficient than the naive pair swapping approach of Algorithm 2, especially for a larger scale grouping problem.

Finally, to fully validate and compare the grouping strategies described in this paper, a real world trial is necessary. We suggest implementing the grouping strategies, as well as a randomized grouping as a control, over several schools, or within a volunteer pool. One could design activities for group interaction to take place over a trial period or semester. Finally, data can be collected before and after the experiment and analyzed for changes in the social network that indicate the performance of the different grouping methods.

References

[1] Chris Fraley and Adrian Raftery. *mclust: Normal Mixture Modeling for Model-Based Clustering, Classification, and Density Estimation*, 2015. R package version 5.0.1 — For new features, see the 'Changelog' file (in the package source).

[2] Bill Hansen. A comparison of social network and person-centered strategies to assign adolescents to academic teams.

[3] Gurobi Optimization Inc. Gurobi optimizer reference manual, 2015.

[4] Jay Magidson and Jeroen Vermunt. Latent class models for clustering: A comparison with k-means. *Canadian Journal of Marketing Research*, 20(1):36–43, 2002.

[5] Matthew W. Ohland Richard A. Layton, Misty L. Loughry and George D. Ricco. Design and validation of a web-based system for assigning members to teams using instructor-specified criteria. 2010.

Appendix A Example Solver Output

```
Optimize a model with 10398 rows, 5587 columns and 70115 nonzeros
Found heuristic solution: objective 45.3603
Presolve removed 4958 rows and 0 columns
Presolve time: 0.61s
Presolved: 5440 rows, 5587 columns, 67784 nonzeros
Variable types: 0 continuous, 5587 integer (5587 binary)

Root relaxation: objective 1.320451e+02, 1541 iterations, 0.11 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  132.04510    0   21   45.36032  132.04510   191%     -    0s
H    0     0                      100.1365222  132.04510  31.9%     -    0s
     0     0  132.04510    0   55  100.13652  132.04510  31.9%     -    1s
     0     0  132.04510    0   19  100.13652  132.04510  31.9%     -    1s
H    0     0                      107.0325901  132.04510  23.4%     -    1s
     0     0  132.04510    0   39  107.03259  132.04510  23.4%     -    1s
H    0     0                      127.0477683  132.04510  3.93%     -    1s
     0     0  132.04510    0   47  127.04777  132.04510  3.93%     -    1s
     0     0  132.04510    0   32  127.04777  132.04510  3.93%     -    2s
     0     0  132.04510    0   26  127.04777  132.04510  3.93%     -    2s
     0     0  132.04510    0   23  127.04777  132.04510  3.93%     -    2s
     0     0  132.04510    0   27  127.04777  132.04510  3.93%     -    2s
     0     0  132.04510    0   25  127.04777  132.04510  3.93%     -    2s
     0     0  132.04510    0   19  127.04777  132.04510  3.93%     -    2s
     0     0  132.04510    0   20  127.04777  132.04510  3.93%     -    3s
     0     0  132.04510    0   16  127.04777  132.04510  3.93%     -    3s
     0     2  132.04510    0   16  127.04777  132.04510  3.93%     -    4s
    26    33  132.04510   10   37  127.04777  132.04510  3.93%  38.2    5s
H   27    33                      127.0500734  132.04510  3.93%  36.8    5s
*  381   381              45      129.6603287  132.04510  1.84%  21.4    9s
   437   427  131.95082   80   61  129.66033  132.04510  1.84%  22.6   10s
H  757   739                      130.7942965  132.04510  0.96%  17.1   14s
H  758   740                      130.8956559  132.04510  0.88%  17.1   14s
H  759   645                      131.8181850  132.04510  0.17%  17.1   14s
   896   742  132.04510   52   31  131.81818  132.04510  0.17%  16.8   15s
  1120   914  132.04500  105   15  131.81818  132.04510  0.17%  17.5   20s

Cutting planes:
  Gomory: 7
  Cover: 6
  Clique: 9
  MIR: 4
```

```
  GUB cover: 10
  Zero half: 2

Explored 1124 nodes (47881 simplex iterations) in 22.93 seconds
Thread count was 4 (of 4 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 1.318181849546e+02, best bound 1.318186593764e+02, gap 0.0004%
```

Appendix B Example Solution

The following is an example of the final output of the grouping algorithms. Each student has been assigned with a unique ID Number. Each row of the output represents a team of four or five students.

```
94 97 105 109
44 57 58 162
27 72 117 143
11 78 112 130
119 126 137 158
34 38 60 101
28 36 107 136
46 69 123 149
12 64 133 151
10 87 139 153
39 83 160 161
4 50 70 82
32 47 90 135
6 53 104 113
14 23 92 102
17 134 144 171
128 129 140 172
68 110 111 118
56 65 89 142
3 29 37 41
2 13 22 170
43 132 147 169
7 8 54 59
30 35 76 106
40 71 74 99
80 88 163 168
73 79 100 103
45 61 108 127
81 96 145 148
33 84 138 146
48 67 77 125
9 49 62 98
20 24 63 86 116
25 91 124 157
1 51 52 95
16 21 120 131
19 93 115 159
```

Appendix C Data Summary

Summary of Data Set	
varname	description
gender	Gender
i1	Singing or playing a musical instrument
i2	Participating in plays or drama
i3	Reading books or magazines
i4	Doing art, crafts, or hobbies
i5	Playing sports or joining a sports team
i6	Attending a sports event
i7	Doing things to improve your fitness
i8	Thinking about fashion and style
i9	Playing computer, board, or fantasy games
i10	Doing outdoor activities
i11	Doing things with animals
i12	Doing things with engines, machines, or technology
v1	Acceptance -- having friends, being cared about
v2	Achievement -- accomplishing difficult or important things
v3	Adventure -- exploring the world, doing new things
v4	Character -- doing what is right; having self-respect
v5	Creativity -- imagining, inventing, or building new things
v6	Education -- gaining knowledge and understanding
v7	Faith -- having religious thoughts and beliefs
v8	Fame -- having the attention and respect of others
v9	Fitness -- physical health and strength
v10	Independence -- having freedom to do what you want
v11	Inner Strength -- being able to deal with life's challenges
v12	Peace -- being in harmony with yourself and others
v13	Stewardship -- taking care of people and the earth
v14	Talent -- developing special skills and abilities
v15	Wealth -- having money and what money can buy
sb1	I feel like I belong at this school.
sb2	I nave no interest in school.
sb3	The teachers at this school like me.
sb4	I wish I went to a different school.
sb5	I like the teachers at this school.
sb6	I am often bored at school.
sb7	This school is a good school to go to.
gender	1=boy, 2=girl
interests	1=low, 2, 3, 4, 5=high
values	1=not important, 2, 3, 4, 5=absolutely most important
school bonding	Strongly Disagree, Disagree, Agree, Strongly Agree
For adjancency matrix (0=no tie, 1=tie)	
For weight matrix (1=less than an hour, 2=1-2 hours, 3=2-5 hours, 4=5-10 hours, 5=more than 10 hours)	