Improving Automated Group Assignments in an Academic Setting

Prof. Petra Bonfert-Taylor, Dartmouth College

Petra Bonfert-Taylor is a Professor and an Instructional Designer at the Thayer School of Engineering at Dartmouth College. She received her Ph.D. in Mathematics from Technical University of Berlin (Germany) in 1996 and subsequently spent three years as a postdoctoral fellow at the University of Michigan before accepting a tenure-track position in the Mathematics Department at Wesleyan University. She left Wesleyan as a tenured full professor in 2015 for her current position at Dartmouth College. Petra has published extensively and lectured widely to national and international audiences. Her work has been recognized by the National Science Foundation with numerous research grants. She is equally passionate about her teaching and has recently designed and created a seven-MOOC Professional Certificate on C-programming for edX for which her team won the ”2019 edX Prize for Exceptional Contributions in Online Teaching and Learning”. Previously she designed a MOOC ”Analysis of a Complex Kind” on Coursera. The recipient of the New Hampshire High Tech Council 2018 Tech Teacher of the Year Award, the Binswanger Prize for Excellence in Teaching at Wesleyan University and the Excellence in Teaching Award at the Thayer School of Engineering, Petra has a strong interest in broadening access to high-quality higher education and pedagogical innovations that aid in providing equal opportunities to students from all backgrounds.

Mr. Christopher Miller, Dartmouth College

Improving Automated Group Assignments in an Academic Setting

Chris Miller* **and Petra Bonfert-Taylor**

Dartmouth College

ABSTRACT

This research paper outlines and evaluates a method to efficiently assign students to groups in an academic setting. Group work is an important part of academic learning. Prior research shows that data-oriented group assignment methods outperform self selected and random methods. We introduce the Group Assignment Tool, an automated method to perform data-oriented group assignment in an academic setting. The Group Assignment Tool maximizes the benefits of instructor-assigned groups by performing probabilistic weighted optimization of groups based on student survey data. We show that the Group Assignment Tool produces comparable outcomes to prior hill climbing algorithms for group formation with significantly faster runtime, and produces groups which score within a small factor of optimal scores. We present the GAT via a traditional presentation as well as a demonstration.

Keywords: Group Assignment, Group Formation, Team Assignment, Team Formation, Hill-Climbing, Collaborative Learning

1 OVERVIEW

College courses across a wide variety of departments rely on group work. Group work can promote collaborative learning, improve information retention, and provide students with valuable team skills which are vital for the modern workplace [Hansen (2006)]. However, groups can also suffer from issues such as lack of shared scheduling availability, lack of diverse skill sets, or marginalization of at-risk students [Bacon et al. (1999) and Dasgupta et al. (2015)]. This means that forming effective groups—which we define as groups characterized by sufficient diversity of experience and knowledge, shared scheduling availability, lack of isolation of at-risk students, and fulfilled student preferences—is an important task. The three primary methods of group assignment are self-selected (students choose their own groups), random assignment (students are sorted into random groups), and instructor selected (course instructors assign students to groups).

Literature suggests that random and self-selected groups are often ineffective. [Feichtner and Davis (1984)] reveals that students who formed their own groups were highly likely to rate the experience poorly, and [Bacon et al. (1999)] notes performance issues with random and self-selected groups. Intuitively, this makes sense. Groups formed by students are often chosen based on prior relationships rather than diversity of experience and scheduling availability, and randomly selected groups are equally unlikely to produce combinations with diverse levels of experience and similar schedules. Self-selected groups may also lead to isolation and lack of participation from students who do not have close friends in the class.

Instructor selected groups are clearly a better method for producing positive group outcomes. However,

*Funding for initial development of the Group Assignment Tool from September 2017 through March 2018 was provided by the Kaminsky Undergraduate Research Award.

manual group formation can be difficult and time consuming. Group formation is a high dimensionality constrained optimization problem. Instructors typically aim to maximize heterogeneity of groups with respect to certain attributes while minimizing heterogeneity with respect to others and fulfilling various constraints—for example, avoiding pairing certain students, or avoiding isolating students by gender or ethnicity (research shows that isolating at-risk students in groups can damage outcomes for them [Sullivan et al. (2018) and Dasgupta et al. (2015)].)

The Group Assignment Tool (GAT) addresses the difficulty of effective group formation by introducing an automated method to produce groups based on student survey data. Instructors can select the relative importance of each survey question and also indicate whether to optimize for heterogeneity or homogeneity, avoid isolating students by any attribute, or fulfill student preferences for a given question.

We also introduce a scoring model for restrictive questions, which allow instructors to incorporate student preferences into group formation. This question type is used for applications such as allowing students to identify others with whom they do not wish to be paired.

The Group Assignment Tool has been used in its pilot phase to assign groups in multiple undergraduate Engineering, Sociology, Anthropology, and Chemistry courses, with positive feedback from students and instructors. In a modified application, we scheduled lab block assignments for an engineering school machine shop. We are currently developing an integration into a college online learning management system to make it easier for instructors to use the Group Assignment Tool in an effort to save time and improve student outcomes.

2 PRIOR WORK

Previous literature describes many approaches towards group formation. These include clustering, genetic algorithms, and hill climbing optimization. The latter is the approach we take.

2.1 Clustering

Clustering algorithms can be used to group data using some measure of similarity. They work by grouping points which are close together into the same clusters. In common approaches such as k-means or fuzzy c-means clustering, this is done by repeatedly assigning points to clusters based on cluster centroids and updating cluster centroids to reflect the newly assigned points until convergence. This approach is computationally efficient, but poses some challenges. Clustering algorithms are inherently designed to group similar points together, which limits these algorithms to producing homogeneous groups. And while clustering algorithms such as k-means and fuzzy c-means allow us to define the desired number of clusters, they typically are not able to set a specified number of points per cluster, which is a problem for assigning fixed-size student groups. Approaches such as Christodoulopoulos and Papanikolaou (2007) are forced to alter cluster compositions after formation to resolve that problem, producing less optimal groups.

2.2 Genetic Algorithms

Genetic algorithms follow an evolutionary approach. By merging sets of previous group assignments and introducing random mutations, they can solve certain optimization problems extremely effectively.

Systems such as Wang et al. (2007) are limited to selecting for heterogeneity or homogeneity only, and do not allow instructors to form groups that are optimized with regard to both or to introduce other criteria. Many also have limitations on the number of variables or group sizes.

The nature of the group assignment problem also limits the applicability of genetic algorithms. One cause is group size constraints. Moving an individual from one group to another requires swapping another individual back into the first group to avoid either group being over or under-filled. Another cause is that genetic algorithms typically generate multiple potential solutions in each generation and combine the best solutions in a crossover step. However this crossover step is problematic in the case of the group assignment problem since each student must exist in exactly one group in the class. The crossover step could lead to removed or duplicate students. In practice, this means that implementations of genetic algorithms select random groups and swap students between them, producing an implementation of a hill climbing algorithm.

2.3 Hill Climbing Algorithms

Hill climbing algorithms take an initial state, and evaluate local neighbor states to move to a state with a higher score. In a group assignment context, this entails taking an initial set of groups and moving students between groups as long as doing so improves scores. A significant limitation of hill climbing algorithms is that they can be caught in a local maximum along the objective function. Our approach addresses this problem by allowing random state changes which do not improve the score locally, but may allow the algorithm to escape a local maximum.

The hill climbing algorithm outlined by Cavanaugh et al. (2004) and implemented by Layton et al. (2010) as CATME Team-Maker is the closest to our work. Team-Maker permits arbitrary survey questions and pursues a similar swapping strategy to ours. As we show in Section 4, our approach is more effective at scale for large class sizes.

3 TECHNICAL DESCRIPTION

To produce groups, we first record student responses to survey questions determined by the instructor. We next assign students to initial groups, then follow the swapping strategy outlined below to switch students between groups with the goal of improving (maximizing) average group scores, defined in Section 3.3. We produce initial group assignments randomly.

3.1 Swapping

Group Selection

At each iteration, the algorithm selects two groups to swap students between. In the majority of cases these two groups are the lowest scoring current group and a random group excluding the highest scoring current group. But with probability $\varepsilon_g = .25$, these two groups are chosen entirely randomly instead. This effectively focuses the algorithm's effort on improving the current worst group (and not compromising the current best group), while still allowing swaps to improve other groups.

Student Swapping

Once groups have been selected, we use an epsilon-greedy search strategy inspired by the random walk approach, WalkSAT, for constraint satisfaction problems [Selman et al. (1996)].

At each iteration, once two groups have been selected as outlined above, we select between a greedy search and a random swap. We select a random swap with some small probability ε_s and greedy search with probability $1 - \varepsilon_s$, where ε_s decreases by a constant discount factor γ at each iteration. Random swaps allow the algorithm to escape local maxima. The discount factor γ produces stability, ensuring that random swaps occur with reduced frequency as group scores converge. This means that the algorithm's actions approach that of a greedy algorithm as epsilon approaches zero, so that a random swap towards the end of the program will not compromise scores.

Swapping continues until either the class score (the average group score across all groups) converges, which we define as improving by less than .5% over 1000 iterations, or the program reaches a predefined iteration count limit.

As an example of how epsilon-greedy swapping can outperform purely greedy swapping, consider two groups of four students. Assume the optimal pairing is only achieved if two students from the first group are swapped with two students from the second group. If the initial state is a local maximum and any single swap of one student reduces the score, a purely greedy algorithm will never reach the optimal pairing. Our epsilon-greedy algorithm, by contrast, can randomly make a swap which temporarily reduces the class score, and then take a greedy step to reach the optimal assignment.

Greedy Search

In greedy search, we test swapping each possible pair of students between the two selected groups, scoring the groups after each swap. If no swaps improve the average of both groups' scores, the original groups are kept. Otherwise, we select the swap which produces the maximum average score between the two groups:

$$\underset{(i,j)}{\arg\max} \quad \mathrm{score}(\mathrm{swap}(i,j))$$

where i is a member of the first group and j is a member of the second and the function score() returns the average score of the two groups.

Random Swap

In random swap, we select a random student from each group and swap them, even if this swap decreases the average group score.

Selection of ε and γ

We determined the most effective values for ε_s and ε_g experimentally, and set them to .05 and .25 respectively. We set γ as

$$\gamma = \left(\frac{.001}{\varepsilon_s}\right)^{\frac{1}{N_{iter}}},$$

where N_{iter} refers to the iteration limit. This ensures that ε decays to a value of .001 (a 0.1% chance of a random swap) by the time it reaches the final iteration.

3.2 Search Strategy

GAT Random allows the user to provide a limit on runtime. The algorithm runs up to the time limit, repeating with different random initializations and returning the highest scoring set of groups before time expires.

Anytime Style

To avoid being limited by a poor random initialization, we use an approach inspired by anytime algorithms. Anytime algorithms produce increasingly good solutions as they continue to run, and return the best solution upon reaching a time limit or being interrupted [Zilberstein (1996)].

We generate random initializations and swap until we reach a given time limit. The tool keeps a running average of how long each initialization and swap step takes and stops re-running once elapsed time plus average runtime exceeds the time limit. At this point, the program outputs the highest scoring set of groups found.

This functionality allows users to choose their preferred compromise between scores and runtime. A longer runtime allows the algorithm to explore a greater portion of the state space by starting from more random initializations, while a shorter runtime may be desirable for a user who needs groups immediately.

In practice, 10 to 15 seconds produces excellent scores even for large classes of several hundred students, while remaining efficient.

Minimum runtime is constrained by the runtime of one initialization/swap pairing (which is itself constrained by the chosen number of iterations). If runtime is set to less than the runtime for one initialization/swap pairing (typically around 2 to 4 seconds for normal iteration counts), the program will not terminate until after the runtime of one pairing has completed.

3.3 Group Scoring

When evaluating whether or not to swap students between groups, we need to evaluate whether or not the swap improves group fitness by determining scores for the new groups. We follow the group scoring method outlined by Cavanaugh et al. (2004) for multiple choice, checkbox, and scheduling questions and describe it below. We extend their notion of pairing underrepresented students with isolation questions, and introduce a new question type, restrictive questions.

The purpose of the algorithm is to produce a set of groups which maximizes the average group score G_{avg}, given by

$$G_{avg} = \frac{1}{m} \sum_{j=1}^{m} G_j,$$

where m is the number of groups in the class and the score of group G_j, is given by

$$G_j = \sum_{i=1}^{q} W_i X_{i,j}.$$

Here, q is the number of scored questions in the survey, W_i is the weight of question i, and $X_{i,j}$ is the fitness measure for group j with respect to question i in the range $[0,1]$.

The value of the fitness measure, $X_{i,j}$, is dependent on question type and is defined below. For all questions, $X_{i,j}$ ranges from 0 to 1. Note that a high fitness measure may not be desirable for a given question type; for homogeneous questions and isolation or restriction questions (see below), a value of zero is often desirable. In this case, the question is assigned a negative weight so that the algorithm selects for lower fitness scores.

Multiple Choice Questions

We define multiple choice questions as questions which allow students to select exactly one option from those presented.

For multiple choice questions, the fitness measure $X_{i,j}$ (recall that this is the fitness measure for group j with respect to question i) is a measure of heterogeneity where a value of 1 indicates a perfectly heterogeneous group and a value of 0 indicates a perfectly homogeneous group. A group is heterogeneous with respect to a given question if students in the group select mostly different answers to the question. A group is homogeneous with respect to a given question if students in the group select mostly the same answers to the question. For these questions, the fitness measure is given by:

$$X_{i,j} = \frac{1}{n} \sum_{k=1}^{c} \bigvee_{s=1}^{n} r_{s,k},$$

where n is the number of students in the group, c is the number of choices for the question, and $r_{s,k}$ is 1 when student s has selected option k and 0 otherwise. The expression $\bigvee_{s=1}^{n} r_{s,k}$ is the logical or operator over values of $r_{s,k}$ as s ranges from 1 to n and is equal to 1 if any value of $r_{s,k}$ is 1 and 0 if all values of $r_{s,k}$ are zero.

Effectively, $X_{i,j}$ measures the number of unique answers in group j to multiple choice question i, normalized by the number of students in the group. When all students in group j select the same option, the fitness measure is low: $\frac{1}{n}$. When all students select different options, the fitness measure reaches 1.

Checkbox Questions

We define checkbox questions as questions which allow students to select zero or more options.

For these questions, the fitness measure $X_{i,j}$ is again a measure of heterogeneity where a value of 1 indicates a perfectly heterogeneous group and a value of 0 indicates a perfectly homogeneous group.

$$X_{i,j} = \max\left\{0, 1 - \frac{1}{nc}\sum_{k=1}^{c} b_k^2\right\},$$

where n is the number of students in the group, c is the number of choices for the question, and b_k is given by:

$$b_k = \begin{Bmatrix} 0, & \text{if } \sum_{s=1}^{n} r_{s,k} < 2 \\ \sum_{s=1}^{n} r_{s,k}, & \text{else} \end{Bmatrix}$$

i.e., b_k is the number of students who selected option k if two or more students selected it, otherwise it is zero.

This measure falls to zero (indicating high homogeneity) as the number of students who select each option grows, with the square of the number of students selecting each option allowing us to indicate that four students selecting the same option is much more homogeneous than two students selecting the same option. If all students select different options, the function returns $1 - \frac{1}{nc}$, a value close to 1 which is indicative of high heterogeneity.

Scheduling Questions

We define scheduling questions as questions where students select blocks of time in which they are unavailable.

For scheduling questions, the fitness measure represents the group's level of homogeneity in free schedule blocks. A value of 1 indicates a high level of schedule compatibility and a value of 0 indicates a low level of schedule compatibility. We use

$$X_{i,j} = \frac{1}{c}\left(c - \sum_{k=1}^{c} \bigvee_{s=1}^{n} r_{s,k}\right),$$

where c is the total number of options (scheduling blocks), n is the number of students in the group, and $r_{s,k}$ is 1 when student s has selected option k and 0 otherwise. Recall that a student selecting option k indicates that this student is not available during scheduling block k. Intuitively, this measures the total fraction of all scheduling blocks for which no student in the group is busy.

Restrictive Questions

Restrictive questions are questions which allow students to indicate their preferences for certain attributes of their team members. As an example, instructors may wish to allow students to select other students with whom they wish to work or would prefer not to work. By providing a positive weight to the question, the instructor can allow students to indicate that they do not wish to work with students who provided a specified response to another question. By providing a negative weight, on the other hand, users can allow students to indicate that they do wish to work with students who provided a specified response to another question.

This question type can link to any other question or questions. As an example of linking to multiple questions, consider a set of three questions asking students for their top three preferred group project topics. By linking students' top choices to questions recording students' first, second, and third choices, instructors can reward groups in which each student's first choice topic is among all of their team-members' top three choices.

For restrictive questions, a value of 1 indicates that the group violates the restriction, while a value of 0 indicates that the group is in compliance with the restriction.

$$X_{i,j} = \bigvee_{s=1}^{n} Z_{s,i},$$

where

$$Z_{s,i} = \begin{cases} 1, & \text{there exists } b \in S \ s.t.\ a_{s,i} = a_{b,i_a} \\ 0, & \text{else} \end{cases}$$

S is the set of all students, $a_{s,i}$ indicates the response of student s to question i, and i_a is the associated question for restrictive question i.

Isolation Questions

Isolation questions allow users to prevent groups in which students with a specified majority response outnumber other students. For example, studies in STEM courses have shown that female students report higher self-confidence and are more likely to major in a course's subject if placed in groups with a higher percentage of women, and student outcomes for all students in a group improve as the percentage of women in the group increases [Dasgupta et al. (2015) and Sullivan et al. (2018)]. Isolation questions allow users to penalize isolation of students by gender, ethnicity, or other attribute.

For isolation questions, a value of 1 indicates that a student is alone with respect to their question response in a group, a value of 0.5 indicates that students from minority answer choices make up less than half of the group, and a value of 0 indicates that no isolation is present. These questions thus require a negative weight to indicate that groups exhibiting isolation will be penalized.

$$X_{i,j} = \begin{cases} 1, & Y_{i,j} = 1 \\ \frac{1}{2}, & 1 < Y_{i,j} < \frac{1}{2}n \\ 0, & Y_{i,j} \geq \frac{1}{2}n \ \text{ or } \ Y_{i,j} = 0 \end{cases} ,$$

where $Y_{i,j}$ is the number of students in group j who selected a non-majority answer choice for question i, defined as

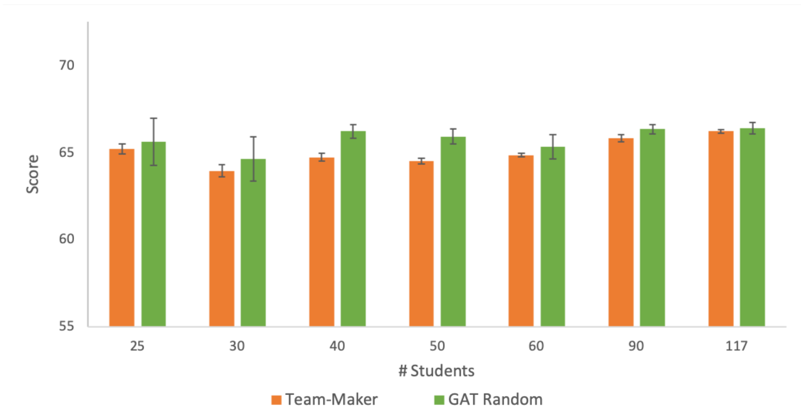
$$Y_{i,j} = \sum_{s=1,\ k \in A_i}^{n} r_{s,k},$$

where k refers to any non-majority answer choice in the set A_i of non-majority choices for question i.

4 EVALUATION

We compare the results from the Group Assignment Tool (GAT) to an implementation of the Team-Maker algorithm described in [Cavanaugh et al. (2004)] and [Layton et al. (2010)].

Figure 1. Score comparison between the Group Assignment Tool and Team-Maker



4.1 Scores

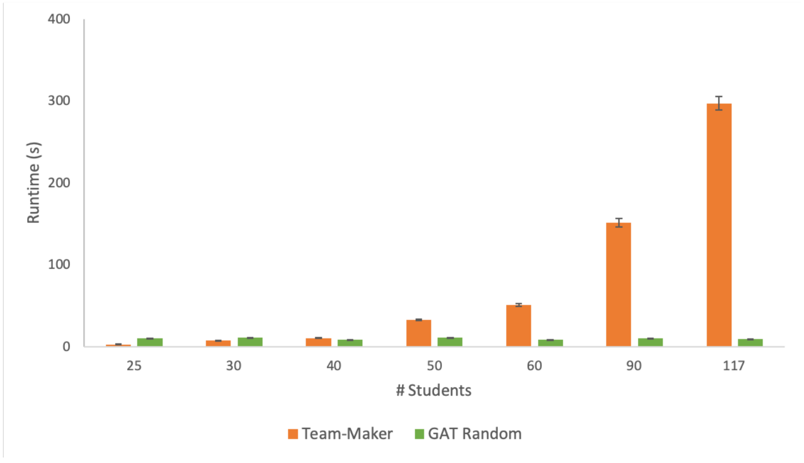
Figure 1 depicts mean group scores averaged over ten runs of each program. Each run was performed under a different random seed. Error bars depict the 99% confidence intervals for the data, meaning that we can say with 99% confidence that the true mean lies within the error bars.

GAT Random refers to the Group Assignment Tool run with random initializations. Team-Maker refers to the Team-Maker algorithm introduced in [Cavanaugh et al. (2004)].

4.2 Speed

Figure 2 depicts the average runtime, in seconds, for each algorithm across all class sizes tested. Error bars depict the 99% confidence intervals for the data.

Figure 2. Runtime comparison between the Group Assignment Tool and Team-Maker



Both approaches have very similar runtimes for small class sizes ($N \leq 40$). However, as class sizes grow, the GAT approach outperforms the Team-Maker approach from a speed perspective. This is likely because the Team-Maker approach exhaustively analyzes every pair of groups when swapping, and the number of possible group pairs grows exponentially with the number of groups (a linear function of class size).

These results indicate that the Group Assignment Tool approach is suitable for large classes. GAT Random produces equivalent to slightly stronger scores than Team-Maker and provides stable runtime.

Our algorithm also allows runtime to be explicitly set by the user according to their needs. Stable runtime means that it can be used for extremely large-scale applications without runtime issues.

4.3 Consistency at Scale

The Group Assignment Tool is consistent at scale. While Team-Maker achieves consistency in scores by continuing to exhaustively analyze every possible student swap between each pairing of groups, the Group Assignment Tool's use of probabilistic swapping allows it to avoid the explosive runtimes Team-Maker entails for large class sizes. This shows that the Group Assignment Tool is highly scalable for use in large courses.

4.4 Proximity to an Optimal Solution

To provide context for the performance of the Group Assignment Tool, we introduce two methods to compare performance of the GAT and Team-Maker to an optimal solution (defined as a set of groups for a given class which produces the highest possible average group score).

For a given class size, n, and a given group size, p, there are C ways to partition the class into groups of p students. Assuming p divides n,

$$C = \frac{n!}{\left(\frac{n}{p}\right)!\,(p!)^{\frac{n}{p}}}$$

Thus exhaustively analyzing every grouping of students into groups of 4 is impractical for large class sizes. For 16 students, $C = 2,627,625$, for 20 students, $C = 2,546,168,625$ and beyond 20, the number C continues to grow at an extremely rapid rate.

As proof of the tool's effectiveness in a fully optimal scenario, we took a 12 person subset of our dataset and exhaustively analyzed all partitions of a twelve person class into groups of four. We then found the group assignment with the maximum average group score. GAT Random produced groups scoring within 2.64% of optimal. This indicates that for small class sizes, the behavior of the Group Assignment Tool is very close to optimal.

Since determining the optimal groups for our dataset is not a tractable problem for large class sizes, to compare performance of the Group Assignment Tool with optimal performance we instead generated new data with known optimal groups. This data generation allows us to measure how close groups formed by the Group Assignment Tool are to optimal groups, and allow evaluation on larger datasets than was possible with collected data. Details of how we generated optimal groups are provided in Section 4.5. Class sizes differ slightly from those used for collected data; we only consider classes and group sizes where group size evenly divides class size, and data generation allows us to evaluate all tools on two chosen larger class sizes of 200 and 400 students.

Figure 3 shows mean group score (averaged across ten runs with different random seeds) on generated datasets for each approach vs number of students being grouped. We also include optimal scores to contextualize our results. Error bars depict the 99% confidence intervals for the data.

As Figure 3 depicts, all approaches achieve optimal scores for the smallest class size of 20 students.

Figure 4 shows mean runtime, in seconds, (averaged across ten runs with different random seeds) for each approach (GAT Random, and Team-Maker) vs number of students being grouped. We depict runtimes on a log scale, because the long runtimes of Team-Maker for class sizes 200 and 400 require a y-axis scale which makes runtimes difficult to interpret for small class sizes. Error bars again depict the 99% confidence intervals for the data.

The introduction of larger class sizes shows the rapid growth of runtimes for Team-Maker, which makes the Team-Maker tool impractical to use in large class settings. GAT Random shows similar performance

Figure 3. Comparison of Group Assignment Tool and Team-Maker scores with optimal scores

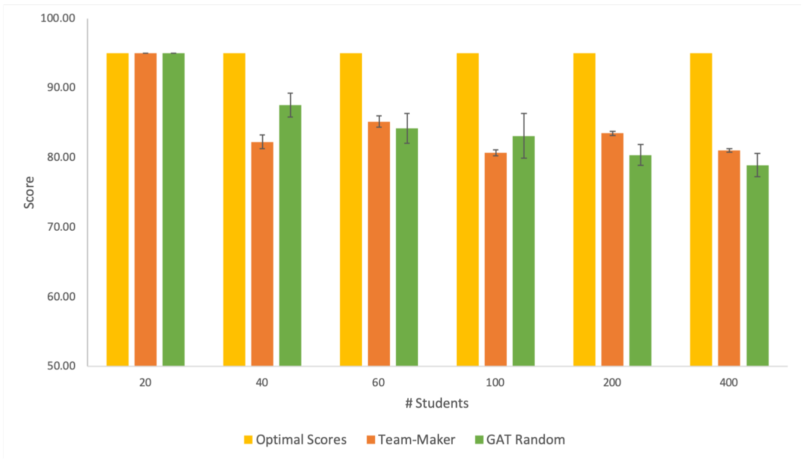
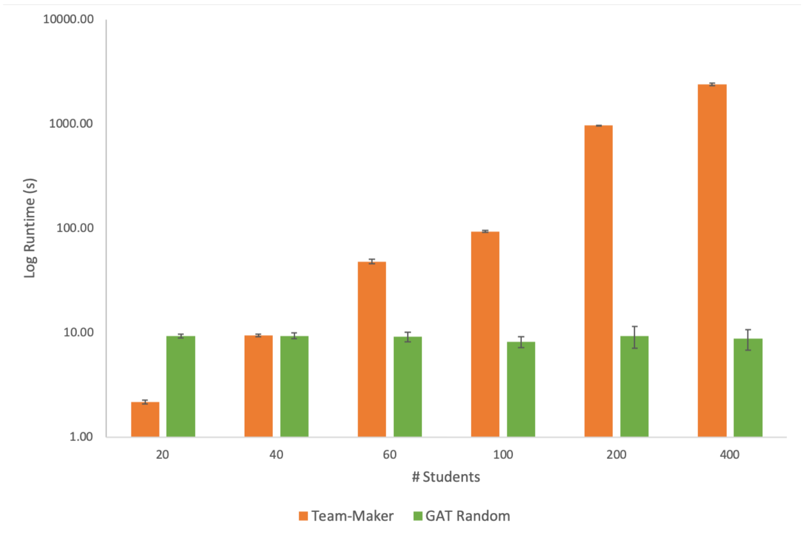


Figure 4. Comparison of runtimes on generated data, log scale



to Team-Maker and requires only ten seconds of runtime, showing its value for applications where users require extremely rapid group generation or group generation for large courses.

4.5 Parameters and Data

GAT Random Initialization Parameters

We set $\varepsilon = .25$, and used a discount factor of $\gamma = \left(\frac{.01}{\varepsilon}\right)^{\frac{1}{n}}$, where n was the number of iterations, set to 15000. For iteration, we gave the algorithm a time limit of 15 seconds. Finally, we set the convergence threshold to .005.

Team-Maker Parameters

We set Team-Maker's outer loop counter (the number of random initializations the algorithm starts from) to 50, and set the inner loop counter (the number of times it will iterate over all combinations of groups and attempt to perform swaps) to 20. These parameters were used by both [Cavanaugh et al. (2004)] and [Layton et al. (2010)], and we use them here to remain consistent with their work.

Data
We gathered real student data from a pilot run of the Group Assignment Tool in an undergraduate on-campus course. This included 117 survey responses for questions on topics including gender, ethnicity, class year, major, difficulty of a prerequisite course, schedule availability, teamwork style, leadership style, and hobbies. We obtained multiple class sizes for analysis by selecting subsets of this dataset.

Generated Data
To generate optimal groups, we first initialize each group with an empty student object, then generate responses on a group level according to question type.

For homogeneous multiple choice questions, we randomly select a response from the list of possible responses and assign the entire group that response, producing a multiple choice fitness measure of $\frac{1}{n}$. This is the lowest possible value, since all students must select at least one option.

For heterogeneous multiple choice questions, we randomly select a unique response for each student. If the number of possible responses is less than the number of students in the group, repeat selection is allowed for the additional students. This produces a fitness measure of $\frac{c}{n}$, the maximum possible measure.

For homogeneous checkbox questions, we randomly assign between one and three possible responses to the group such that each student has the same responses. This produces a checkbox fitness measure of 0. This measure cannot be lower because the checkbox scoring measure has a minimum value of 0.

For heterogeneous checkbox questions, we randomly assign each student a unique choice. We allow repeat selection if the group size is greater than the number of options, but this did not occur for our evaluation question set. For our evaluation set, this produces a fitness measure of 0, the maximum possible checkbox fitness measure.

For scheduling questions, we select three random schedule blocks for each group and assign all students in the group those blocks. This produces a maximal scheduling fitness measure of $\frac{3}{4}$ for the scheduling blocks we used. Since all students select three blocks, this value cannot be higher than $\frac{3}{4}$.

For isolation questions, we randomly select between a minority population of 0 and a minority population of $\text{ceil}(\frac{1}{2}n)$. Either option produces the minimum isolation penalty of 0.

As shown above, the generated groups have optimal fitness measures for each question (maximum for positively weighted questions, minimum for negatively weighted questions). Thus the generated groups and scores are the best possible groups and scores for the set of students they apply to, and we can judge the Group Assignment Tool by proximity of its group scores to these optimal group scores.

Questions and Weighting
Group scores calculated by the Group Assignment Tool are dependent on questions, question types, and question weights. For evaluation, we held all three of these factors constant so that scores are directly comparable between different runs of the two Group Assignment Tool approaches and Team-Maker.

5 LIMITATIONS
The Group Assignment Tool may be less effective in settings involving mostly strict constraints (such as isolation or restriction constraints), especially ones which are difficult to fulfill or contradictory to each other. The Group Assignment Tool can produce groups which respect simple isolation criteria such as avoiding gender isolation. However, if multiple isolation criteria are combined with multiple restriction criteria (for instance, if students in the class select an unusually high number of other students they do not wish to work with) the tool may fail to produce groups which respect all of the constraints.

This is because the Group Assignment Tool handles all constraints via score penalties, and our probabilistic approach to maximizing class scores produces an inherent risk of failing to find a unique solution in a large state space. This approach does, however, allow users to weigh which constraints are more important, and can minimize constraint violations even when a perfect solution does not exist.

6 CONCLUSION & FUTURE WORK

Future research is important to determine specific attributes which are relevant to successful group outcomes. Research into significant attributes can help ensure that high scoring groups produced by the Group Assignment Tool correspond to effective groups. Since the Group Assignment Tool maximizes scores according to weights and questions set by the user, if ineffective questions are provided or weights are not appropriate, the tool may produce ineffective groups.

However, our work shows that the Group Assignment Tool is more scalable for large classes than similar previous approaches to the group formation problem while remaining highly effective. It allows efficient optimization across mixed metrics and question types, and can be used to ensure that groups contain students with diverse sets of experience and knowledge and shared availability for meeting outside of class, and that group assignments fulfill student preferences while not isolating at-risk students.

REFERENCES

Bacon, D. R., Stewart, K. A., and Silver, W. S. (1999). Lessons from the best and worst student team experiences: How a teacher can make the difference. *Journal of Management Education*, 23(5):467–488.

Cavanaugh, R., Ellis, M., Layton, R., and Ardis, M. (2004). Automating the process of assigning students to cooperative-learning teams. In *in proc. 2004 ASEE Annual Conf.*

Christodoulopoulos, C. E. and Papanikolaou, K. A. (2007). A group formation tool in an e-learning context. In *19th IEEE international conference on tools with artificial intelligence (ICTAI 2007)*, volume 2, pages 117–123. IEEE.

Dasgupta, N., Scircle, M. M., and Hunsinger, M. (2015). Female peers in small work groups enhance women's motivation, verbal participation, and career aspirations in engineering. *Proceedings of the National Academy of Sciences*, 112(16):4988–4993.

Feichtner, S. B. and Davis, E. A. (1984). Why some groups fail: A survey of students' experiences with learning groups. *Organizational Behavior Teaching Review*, 9(4):58–73.

Hansen, R. S. (2006). Benefits and problems with student teams: Suggestions for improving team projects. *Journal of Education for Business*, 82(1):11–19. Copyright - Copyright Heldref Publications Sep/Oct 2006; Document feature - ; Tables; Last updated - 2017-10-31.

Layton, R. A., Loughry, M. L., Ohland, M. W., and Ricco, G. D. (2010). Design and validation of a web-based system for assigning members to teams using instructor-specified criteria. *Advances in Engineering Education*, 2(1):n1.

Selman, B., Kautz, H. A., and Cohen, B. (1996). Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532.

Sullivan, L. L., Ballen, C. J., and Cotner, S. (2018). Small group gender ratios impact biology class performance and peer evaluations. *PloS one*, 13(4):e0195129.

Wang, D.-Y., Lin, S. S., and Sun, C.-T. (2007). Diana: A computer-supported heterogeneous grouping system for teachers to conduct successful small learning groups. *Computers in Human Behavior*, 23(4):1997–2010.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73–73.