ASEE CoED Journal
Published 31 July 2021

gruepr, a Software Tool for Optimally Partitioning Students onto Teams

Joshua L. Hertz[1*]

[1]**Northeastern University, Boston, 02115, MA, USA**

RESEARCH PAPER

Abstract

Abstract—Choosing how to split a group of students into teams for project work can be a time-intensive task for an instructor. An instructor might have a complex set of parameters to optimize, perhaps desiring each team to have a certain number of times throughout the week where they can meet, while also seeking to create teams that are homogeneous in some characteristics and heterogeneous in other characteristics. Demographic composition may also be considered, and perhaps the instructor has certain students that must be placed on the same team or must be placed on different teams. Maybe teams will be formed several times, and no student should have the same teammate twice. A few software tools can be found in the literature to assist an instructor with this task, but few of them seem to be easily and freely accessible. This paper describes a new software tool named gruepr, written in C++ by the author. The code has been released under an open source license, and both the code and compiled binaries with a modern, graphical user interface for Windows and macOS have been made freely available. An important design goal was that usage of the software would come at no cost to any instructor who wanted to use it, and accordingly the survey instrument used by gruepr to survey the students is the free Google Form platform. Other important design goals were that the software was easy to use and highly flexible to the instructor's desired definition of what constitutes an optimal team. Within gruepr, an instructor can create a Google Form survey with a highly customizable set of questions. The Google Form is created in the instructor's own Google Drive. After the students have submitted their survey responses, the instructor opens in gruepr the file of downloaded results and sets a flexible set of teaming options. Gruepr then uses a genetic optimization algorithm to partition the students onto teams. The code takes advantage of multi-threading parallelization and generally finds a reasonably optimal partitioning of students in a few minutes or less on a modern laptop computer.

Keywords: student teams, open source, grouping, teamwork

OPEN ACCESS

Volume
12

Issue
2

Corresponding author
j.hertz@northeastern.edu

Submitted *6 Mar 2021*

Accepted *16 Apr 2021*

Citation
Hertz J.L. "gruepr, a Software Tool for Optimally Partitioning Students onto Teams," *Computers in Education Journal*, vol. 12, no. 2, 2021.

1 Introduction

Students are increasingly asked to work in teams for projects and other class work [1]. The issue of how to best partition a set of students onto teams is a difficult one. The most common strategies are: a) randomized teams, b) student self-selection, or c) intentional selection by the instructor [2–6]. This last strategy may be used to improve the pedagogical and/or project outcomes, but it comes at a cost: instructor preparation time [6]. A robust body of literature has shown that a student's teammates can have a profound influence upon their group experience [7–12]. One particularly illustrative recent example showed that female first year engineering students verbally participated more and felt more positively challenged when working on teams with gender parity or female majority status [12].

A number of computer-based tools have been created to help instructors use student information to form optimal teams, with some of these tools also able to assist the instructor in surveying the students to collect that information [13–16]. Many of the solutions are command-line based and therefore somewhat awkward to use. Even worse, many of these tools are not easily available to

an interested instructor. In some cases, the code has not been released to the public, and in other cases, web-based systems have been taken offline since publication. A centralized server that 1) collects student data by sending to them a web-based survey, 2) processes that data, and then 3) delivers to the instructor the information about which students are on which teams is a very clean and organized approach to this problem; however, maintaining the server and making available the bandwidth and computational power required is labor-intensive and costly if the service is made available to a wide community.

Special mention must be made of CATME, as it is by far the most commonly used platform for computer-based team formation [3, 17, 18]. It is web-based, hosted on CATME's own servers. Many people find CATME to be flexible and easy to use. Still, the source code is no longer freely available, and a usage fee is charged to users in order to support the hosting costs. In addition, some people may find it objectionable that student data is collected and stored by CATME's servers. Nevertheless, CATME was a keen inspiration to the software described in this paper.

Finding an optimal partitioning of discrete objects (i.e., students) into sets (i.e., teams) is computationally difficult, in fact in the class of problems known as NP-hard. The search space for this optimization is calculated as

$$N = \frac{n!}{t!\prod(s!)}$$

where N represents the number of unique partitions, n represents the number of students, t represents the number of teams, and s represents the size(s) of the teams. For example, there are about 10^{18} ways to partition 30 students into 10 equal teams of size 3.

Optimal solutions are sought within this search space, and an instructor's definition of optimal can be very complex. For example, an instructor may desire each team to have simultaneously: a wide distribution among the teammates of academic majors and desired work roles, a narrow distribution among the teammates of past experience levels in a certain skill, at least 3 but ideally 6+ hours each week where all teammates are free to meet, and no isolated students from underrepresented gender or racial minority groups. The instructor might also want to allow each student to provide a few names of classmates with whom they desire to work or absolutely cannot work, or the instructor may want ensure that students work with new teammates each time teams are formed. Given the immensity of the search space and the potential complexity of the instructor's definition of team optimality, a number of optimization methods have been explored. These include genetic algorithms [19], the hybrid grouping genetic algorithm [20], evolutionary algorithms [21], heuristic algorithms [22], and ant-colony optimization [23], among others.

This paper will describe gruepr, a program written by the author to ease the creation of student teams. The design goals for the program and the methods used to create the software will first be discussed. Next, the genetic optimization algorithm that underlies how gruepr finds optimal partitioning onto teams is detailed. After this, results from the use of gruepr in typical cases are given. Finally, future prospects for the software will be detailed.

2 Design Goals and Methods

Since other solutions to the problem of partitioning students onto teams can be found described in the literature and, to some extent, have been made available for use, it was necessary to start with a clear set of distinguishing design goals for gruepr. One clear goal from the outset was to release gruepr as free and open source software [24]. Related to this, beyond simply opening the source code for free download and reuse by anyone, it was seen as important that the program would be easy for non-developers to use at no-cost and little time investment. Thus, binaries for both Windows and macOS were to be released. A cross-platform system for creating a graphical user interface would be needed.

During the author's initial search for software to help form student teams, several of the solutions found described in the literature were web-based systems that were no longer available for public

use. Since the goal for gruepr was to make an approach that many people could use, the decision was made to create a server-free solution. In other words, gruepr would rely only on publicly available systems and no cost or labor to maintain a server would be needed from the author. Relatedly, any centralization of the data collection can raise privacy concerns, and thus a solution was desired that didn't require student data to be stored on or even flow through a gruepr server system. As much as possible, instructor control over their students' data was to be maintained.

Additional design goals included maximal flexibility and ease of use for the instructor. To some extent, these two design goals lie in opposition to each other, as the former implies complexity and the latter implies simplicity. The flexibility of use includes allowing the instructor to largely choose what information they want to obtain from the students, including by writing their own survey questions and set of possible multiple-choice responses. In addition, the instructor can very flexibly determine how to use the student survey data so as to define an optimal team. The idea of flexibility also extends to how the student data is to be collected, with an open format of all data files desired.

Given these design goals, gruepr was programmed in C++. As much as possible, the program is written modularly so that others can choose to reuse and/or replace any functions, classes, and other code structures as befits an open source model. The source code, binaries, and documentation are hosted [25] on the Atlassian Bitbucket git-based platform. Gruepr provides means for an instructor to use their own surveying platform (even a paper-based, offline survey) to gather student information and therefore have complete control over their student's data. Still, for ease of use, gruepr has integrated a means to create and use customized Google Forms as a widely available, familiar, no-cost surveying platform.

To provide a graphical user interface and other ease-of-use expectations of modern software, the Qt libraries [26] are used. These were found to be very full featured, relatively simple to implement, able to be compiled for both Windows and macOS without alteration, and available under the GNU General Public License. To add to the aesthetic value, a suite of complementary color icons from icons8 [27] were used. These icons are available under a Creative Commons license.

A genetic optimization algorithm is used and is described in detail in the next section of this paper. The algorithm is computationally intensive. To reduce the optimization time, the OpenMP parallelization libraries [28] were used. These libraries were simple to implement and significantly speed the optimization process by detecting the number of processor cores present on the computer running gruepr and then multithreading parallel tasks across those cores. The OpenMP libraries are highly recommended for any C++ program that could benefit from multithreaded processing.

3 Solution

3.1 Survey Creation and Resulting Data File

One component of the gruepr software is titled SurveyMaker. It creates a survey instrument to be delivered to the students. The instructor can flexibly choose which questions they wish to include in the survey, and a live preview is shown while the options are being set. Required questions in a gruepr survey include the student's first / preferred name, last name, and email address. The instructor may then choose to include up to 15 "attribute" questions. In gruepr, an attribute can be any question with a categorical or Likert-scale answer. For each attribute, the instructor writes the text of the question and the possible response values. A long list of Likert scales is provided to assist the instructor in writing the response values. Example attribute questions are academic major, GPA, work preferences, self-assessments of ability or confidence in performing certain skills, expected project grade, or truly any multiple-choice question. Next, the instructor may choose to include in the survey a place for students to input their weekly schedule. Students use a grid of checkboxes by hour and by day when they are free for team meetings, and the instructor can choose the span of which hours and days are included in the grid. The instructor

can optionally choose to include in the survey questions asking for: the student's gender identity, racial/ethnic/cultural identity, section of the class they are enrolled in, list of preferred teammates, and list of preferred non-teammates. At the end of the survey, the instructor can add any additional questions they like; these questions may collect data of interest to the instructor but will not be used by gruepr directly.

Once the instructor is satisfied with the survey, with the click of a button, SurveyMaker creates a survey file and a results file. For convenience, these two files can be created as, respectively, a Google Form and a Google Sheet that will be auto-populated with the submissions to the Form. SurveyMaker creates the Google Form and Sheet by packaging the instructor's survey as a URL sent to a custom Google script. After the instructor logs in to their Google account and authorizes the script, the Form and Sheet are created in the instructor's own Google Drive, and the instructor is shown a webpage with links and instructions on how to use these files. Students need only be sent the link to the Form to submit their survey, and the instructor need only use the results download link in order to download a comma-separated value (csv) file containing all the results. If the instructor prefers not to use a Google Form, SurveyMaker can instead output two text files, one a text file containing the questions and instructions and the other a pre-formatted csv file in which the survey results should be copied.

The survey results must be a csv file with a header row containing the text of the questions asked of the students and each subsequent row containing the answers from one student. There are some constraints on the order of the questions and the text content of the questions. There is no requirement to use SurveyMaker to create the survey—there is nothing tying a survey created in SurveyMaker to survey results files when forming teams in gruepr—but the use of SurveyMaker ensures that all of the formatting requirements for the csv file are met. When reading the csv file, gruepr auto-detects what types of questions were included in the survey based on the question texts, their order within the survey, and, to some extent, on the student answers.

3.2 Teaming Options

A variety of highly flexible teaming options have been implemented in gruepr, and it is likely that future versions will continue to add more options. By setting these options, an instructor determines what characteristics of a team are optimal. The options that are currently implemented are listed and described below:

- If the survey includes students from multiple class sections, the instructor may choose to team all students together regardless of section, or may choose to team only those students from one particular section (repeating the process as needed to form teams for the other sections).
- The team size(s) may be set by evenly splitting the students or by arbitrarily setting individual sizes of each team.
- If gender information has been collected, an instructor may choose to prevent teams that have an "isolated" woman, meaning a team with exactly 1 woman, or, similarly, an isolated man or an isolated nonbinary student. Alternatively or in addition, an instructor may choose to prevent teams where all students are the same gender.
- If racial/ethnic/cultural identity information has been collected, an instructor can choose to prevent teams that have an isolated student from an underrepresented minority group. When this question is included in the survey, it has a free-response type answer, allowing students to self-identify however they choose. If the instructor is using this information, gruepr will list all of the students' self-reported identities, and the instructor can decide which of these are to be considered underrepresented.
- The survey may include up to 15 attribute questions, which have a multiple choice response. The instructor may choose independently for each attribute whether the preference is for homogeneity or heterogeneity of responses. Homogeneity refers to all students on a team giving the same response, and heterogeneity refers to the students on a team giving a wide

range of responses. The instructor can set the relative importance of each attribute question using a numerical weighting factor. The weights can be set to any non-negative value; it is only their relative value that is used by gruepr.

- For any of the attribute questions, the instructor may also set pairs of incompatible responses. For example, an instructor might want to separate onto different teams those students who want to work on project option A from those more interested in project option B. Analogously, an instructor may want to make sure no two students who indicated their major is Physics are placed on the same team.
- If weekly schedules have been collected, the instructor may choose to require a certain number of times throughout the week where all students on a team are free to meet. The options available to the instructor are the desired number of meeting times for the team, an absolute minimum number of meeting times for a team, and the minimum length of a meeting time (whether 1 hour is sufficient to count as a meeting time or 2 hours are needed). As with the attribute questions, the relative importance of the schedule can be set using a numerical weighting factor.
- A requirement can be created for any pairs or larger sets of students that must be placed on the same team.
- A requirement can be created for any pairs or larger sets of students that must be placed on different teams.
- A list of requested teammates can be created for each student, and the instructor can require that a certain number of those requests be fulfilled. For example, each student might be able to give the 5 names of requested teammates, and the instructor can set that each student will be on a team with 2 of them.

3.3 Genetic Optimization Algorithm

In this section, the operation of the genetic optimization algorithm is described. Throughout this description, n represents the number of students, and t represents the number of teams. The algorithm performs its work using several arrays of integers, namely: one 1D "team size" array of size t that contains the number of students on each team, one 2D "genepool" array of size (30000 $\times$ n) that contains a set of 30000 possible ways to partition the students into teams, and one 2D "ancestry" array of size (30000 $\times$ 14) that stores, for each genome in the genepool array, the numerical indexes of its parent, grandparent, and great-grandparent genomes.

Next, each student is assigned a unique ID number, from 0 to n-1. One particular partitioning of students into teams is encoded as an array containing a permutation of these ID numbers. Continuing with the example team size array [3, 3, 4], the array of ID numbers [0, 8, 6, 5, 3, 2, 1, 4, 9, 7] represents that the first three students (0, 8, and 6) are on one team; the next three students (5, 3, and 2) are on another team; and the last four students (1, 4, 9, and 7) are on the final team. This representation is depicted graphically in Figure 1. One array of IDs is a "genome", and a population or genepool can be created from a set of such arrays. Gruepr uses a population size of 30000, and thus the genepool is a size (30000 $\times$ n) 2D array of integers.

This simple, array-of-integers genome is memory efficient and requires very little programming overhead, but it creates significant genomic redundancy. Specifically, permuting the order of IDs within a team or all of the IDs between two teams of the same size results in a genome that is distinct but synonymous. In a sense, these two genotypes encode the same phenotype. An example of two synonymous genomes is given in Figure 1. The amount of redundancy is equal to $[t!\prod(s!)]$. This value can be quite large, and less redundant encoding schemes have been explored in the literature [29]. These more complex schemes are generally found to increase the efficiency of genetic algorithms solving grouping problems, but they have not been used in gruepr.

At the start of gruepr's optimization algorithm, a genepool is created from 30000 random permutations of the ID numbers. A "score" is calculated for each genome using a fitness function. This

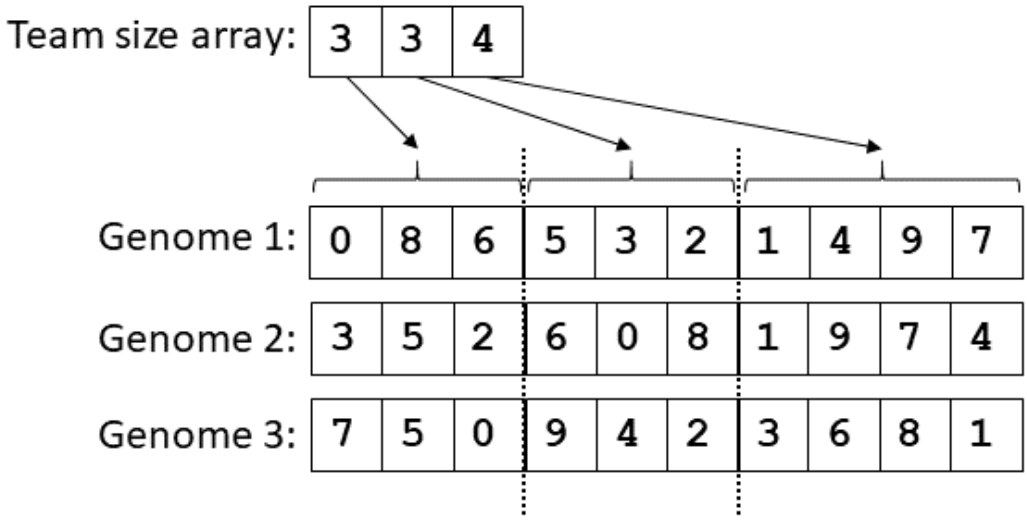

Figure 1. A graphical depiction of an example team size array and a genepool containing 3 example genomes. This team size array indicates that the first three values in a genome array are to be interpreted as the students in one team, the next three values as those in a second team, and the last four values as those in a third team. The dashed lines indicate "team boundaries" but are only displayed to aid visualization; a genome is simply a 1-D array of integers. Note that the first two genomes in this example genepool are different yet synonymous, as identical teams result from them.

function quantifies how optimal a solution each genome forms—in other words, how closely the teams in a genome meet the instructor's definition of optimal teams. The fitness function will be described in much more detail in the next section.

A genetic optimization algorithm converges towards increasingly optimal solutions by causing genomes with greater fitness function scores to be more likely to pass their genomic data onto subsequent generations [30]. In gruepr, the preferential selection of high scoring genomes is performed using the tournament selection method. In this method, a random subset of genomes is selected from the genepool, and two genomes are then selected from the tournament with preference for those with higher score. In gruepr, 60 genomes from the genepool are randomly selected for the tournament and put in rank order according to their fitness function scores. Higher scoring genomes are preferentially selected in the tournament by giving a 33% probability of selecting each genome in turn. In other words, the top scoring genome in the tournament is selected 33% of the time, the second-highest scoring genome is selected 33% of the remaining time, and so on down the line.

Two genomes are selected from a tournament in the same way, but the second genome is rejected in favor of a new one if it is the same as the first genome or "closely related" to it. The term "closely related" is an analogy to familial relationships. The second genome cannot be the sibling, first cousin, or second cousin of the first genome (in all cases, even partial relationships like half-siblings are prevented). Checking for genome relatedness is done in the interest of maintaining increased genetic diversity and thereby preventing premature convergence to a locally optimal solution [31]. Gruepr stores the indexes of each genome's two parent genomes, four grandparent genomes, and eight great-grandparent genomes, and thus the ancestry array is a size (30000×14) 2D array of integers. When selecting the second genome, if there is a match with the first genome among one or more of these values within a particular generation, a different second parent is selected. This method of saving ancestry information is similar to the method implemented in [32]. After successfully choosing two unrelated parents from the tournament, the ancestry of the soon-to-be-created child genome is saved using the indexes of the two parents and the indexes saved in the ancestry array for both parents' parents and grandparents. This process is shown graphically in

Figure 2.

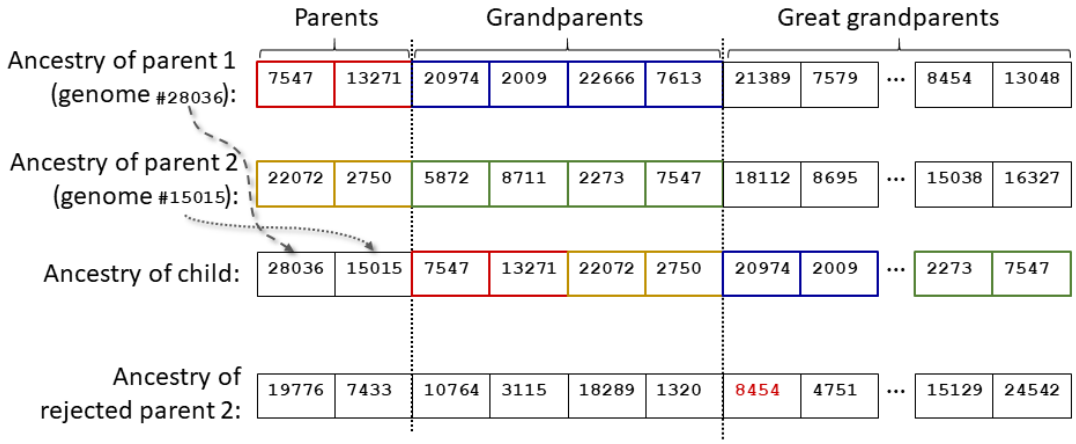

Figure 2. A graphical depiction of the creation of a child's ancestry array. From among the genepool of 30000 genomes, the child's first parent is genome 28036,and second parent is genome 15015. The child's ancestry array consists of, in order:the index of parent 1's genome, the index of parent 2's genome, the first two values of parent 1's ancestry array (i.e., parent 1's parents using the indexes from the previous generation's genepool), the first two values of parent 2's ancestry array, the next four values from parent 1's ancestry array, and the next four values from parent 2's ancestry array. Boxes are colored to help showwhere values are copied from parent to child. At bottom is the ancestry array of a rejected parent 2. After selecting a potential parent 2, its ancestry array is compared to that of parent 1. In this case, the value 8454 (emphasized in red text in the rejected parent's array) is found among the great-grandparents of both parent 1 and this rejected parent 2, indicating these genomes are partial-second-cousins. Note that genome15015 was not rejected as parent 2 even though it has a grandparent of 7547 and parent 1 has a parent of 7547. Comparisons are only meaningful within the same generation (parents-to-parents, grandparents-to-grandparents, etc.).

Once two parent genomes are selected, a child genome is created from them. It is important that the child's genome is distinct from those of the parents yet preserves some of the genetic data that likely led them to have a high fitness function score. In gruepr, this "mating" is performed using the ordered crossover method [30], modified by placing the crossover locations at locations within the genome that represent team boundaries. For example, continuing again with the example team size array [3, 3, 4], there are four genome locations that could be chosen as crossover sites: position 0 (before the first team begins), position 3 (boundary between team 1 and 2), position 6 (boundary between team 2 and 3), or position 10 (after team 3). Two sites are randomly chosen. The child's genome is created by copying the first parent's genome between these boundaries and then filling in the rest of the array with the other IDs in the order they are found within the second parent's genome. In this way, high scoring parents are likely to create high scoring children by passing on what's most likely good about their genome: specific teams and teammates. This process is represented in Figure 3

To create a new generation of 30000 genomes, the following steps occur 29997 times: 60 genomes are randomly selected for a tournament, two non-closely-related parent genomes in the tournament are chosen with preference towards those with a high score, and these two genomes mate to produce a child genome. The remaining three genomes come from directly cloning the three top scoring genomes of the previous generation. This process of "elitism" can increase the convergence speed of a genetic algorithm [30]. The ancestry of an elite genome is passed on as if the elite was parent 1 and parent 2 of its clone in the new generation.

Genetic diversity among a population is helpfully increased by allowing for genetic mutation. In gruepr, mutations are manifested as a swapping of values between two randomly selected

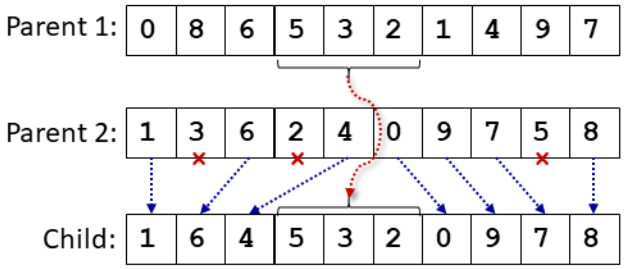

Figure 3. A graphical depiction of thecreation of a child genome from two parent genomes using ordered crossover. Thearray indexes 3 and 6 were randomly chosen as the crossover sites. To form thechild genome, the genomic data between the crossover sites (values 5, 3, and 2)are taken from parent 1, and then the remaining genomic data (all values except5, 3, and 2) are used in the order they are found in parent 2. The child genomeis distinct from both parents but contains teams and teammates found in theparents.

locations in a genome. An example mutation is shown in Figure 4. After all of the 30000 genomes of a generation are created, each genome is mutated with 50% likelihood. If a mutation occurs, an additional mutation occurs again with 50% likelihood. Thus, in each generation, roughly 15000 genomes will have no mutations, 7500 will have a single mutation, 3750 will have 2 mutations, etc. It should be noted that the single top scoring genome from the previous generation, which was cloned as one of the "elite" genomes into the current generation, is not allowed to mutate in gruepr. This is done to ensure that the highest fitness function score will never decrease from one generation to the next.

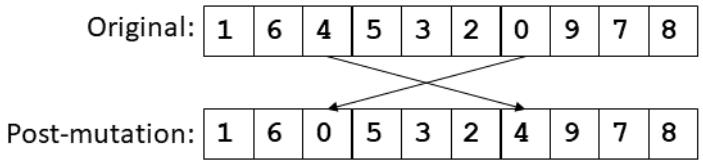

Figure 4. A graphical depiction of a mutation in a genome. Array indexes 2 and 6 were randomly chosen as the mutation sites.

The optimization process continues as each generation produces a subsequent generation that has, generally, increased values of the fitness function. The first generation of 30000 randomly created genomes typically has few, if any, with high fitness function values. In typical use cases of gruepr, the fitness function values increase rapidly for the first 25 generations or so. As might be intuitively expected, the optimization speed generally gets slower as the size and complexity of the search space increases, i.e., as the number of students increases, as the number of teams increases, and/or as the complexity of the instructor's teaming options increases. By default, gruepr chooses when to stop the optimization process, but the instructor may choose instead to let the optimization continue indefinitely until manually stopped. The default behavior stops the optimization after at least 40 generations have passed and the highest fitness function value found among all genomes has stayed within $\pm$1% for the previous 25 generations. As a failsafe, the default behavior also automatically stops optimization after 500 generations.

3.4 Fitness Function

A key component of a genetic optimization algorithm is a fitness function that quantifies how optimal of a solution that a particular genome forms. In gruepr, the genome represents a particular way to partition the students onto teams, as was shown graphically in Figure 1. The optimality of a

gruepr genome—the numerical value of its fitness function—is based on how well each individual team within that genome meets the instructor's definition of an optimal team. In gruepr, a "compatibility score" is calculated for each individual team within the genome. The set of compatibility scores from each team is then combined into a single fitness value.

The set of compatibility scores is combined into a single fitness function value using, in general, the harmonic mean. The harmonic mean is the reciprocal of the arithmetic mean of the reciprocals, and it was chosen over the arithmetic mean because the harmonic mean skews towards the low values in a set. By using the harmonic mean of the team compatibility scores for the genome's fitness value, the algorithm seeks to improve the compatibility score of all teams, but gives relative precedence to the improvement of teams with low compatibility scores. The calculation of a harmonic mean becomes problematic when there are non-positive values being averaged, and there are times where a team's compatibility score can be 0 or negative. Therefore, whenever any team in a genome has a non-positive compatibility score, gruepr instead uses for the fitness function value an arithmetic mean of the team compatibility scores that is further reduced by subtracting half of its absolute value. A flowchart summarizing the calculation of the fitness function value is given in Figure 5.

The remainder of this section will describe the calculation of an individual team's compatibility score. A compatibility score is a combination of a "base score" comprising a weighted average of the attribute and schedule scores and a "penalty counter" comprising an integer number of penalty points.

Each of the attribute scores and the schedule score range from 0 to 1 (with, as described later, the possibility that schedule scores can occasionally exceed 1). As mentioned in the Teaming Options section, the instructor sets the relative importance of each attribute question and the schedule question using numerical weights. These weights are applied to the attribute scores and the schedule score and then all of these scores are summed in such way that a final, single value that ranges from 0 to roughly 1 results. This calculation is accomplished by normalizing the set of instructor's weight values to "real weights" that sum to 1, then multiplying each attribute score and the schedule score by their respective "real weights", and then summing the results. This value is the team's "base score".

Together with the base score, the instructor may set up a number of absolute requirements for a team. Each time a requirement is not met, the penalty point counter is incremented. There are five categories where a penalty may be applied: schedule, incompatible attributes, gender, underrepresented minority status, and specifically named teammates. A penalty counter is used so that a team that fails to meet only one of the requirements is nevertheless preferred to a team that fails to meet two requirements, and so on for three or more failed requirements.

In summary, a team's compatibility score, c, is calculated according to:

$$c = (\sum (w_a \bullet a) + w_s \bullet s - p] \bullet 100 \quad (1)$$

where w_a and w_s refer to the normalized weight assigned to an attribute question and the schedule question, respectively, a and s refer to the score for an attribute question or the schedule question, respectively, and p refers to the integer value of the penalty counter. The summation in Equation 1 is over all attribute questions. Details on the calculation of *a*, *s*, and *p* are given next.

For each attribute question in the survey, gruepr first gathers the set of response values provided by the students on the team. If this question has ordered responses, such as a Likert scale question, then the total range of values is calculated and normalized to give a value between 0 and 1. For example, if a question has a Likert scale response from 1 (strongly disagree) to 7 (strongly agree) and a team of 4 students gave responses of 1, 4, 1, and 2, then the range of values on the team is 4 – 1 = 3, the total range of values is 7 – 1 = 6, and the normalized range is 3 / 6 = 0.5. In other words, students in this team represent $\frac{1}{2}$ of the total possible range of response values. If the question instead has categorical responses, such as a question asking for a student's academic major, then the "range" of values makes no sense. For these questions, the number of unique responses is counted and normalized to a range of 0 to 1. Using either of these two calculations, a value of

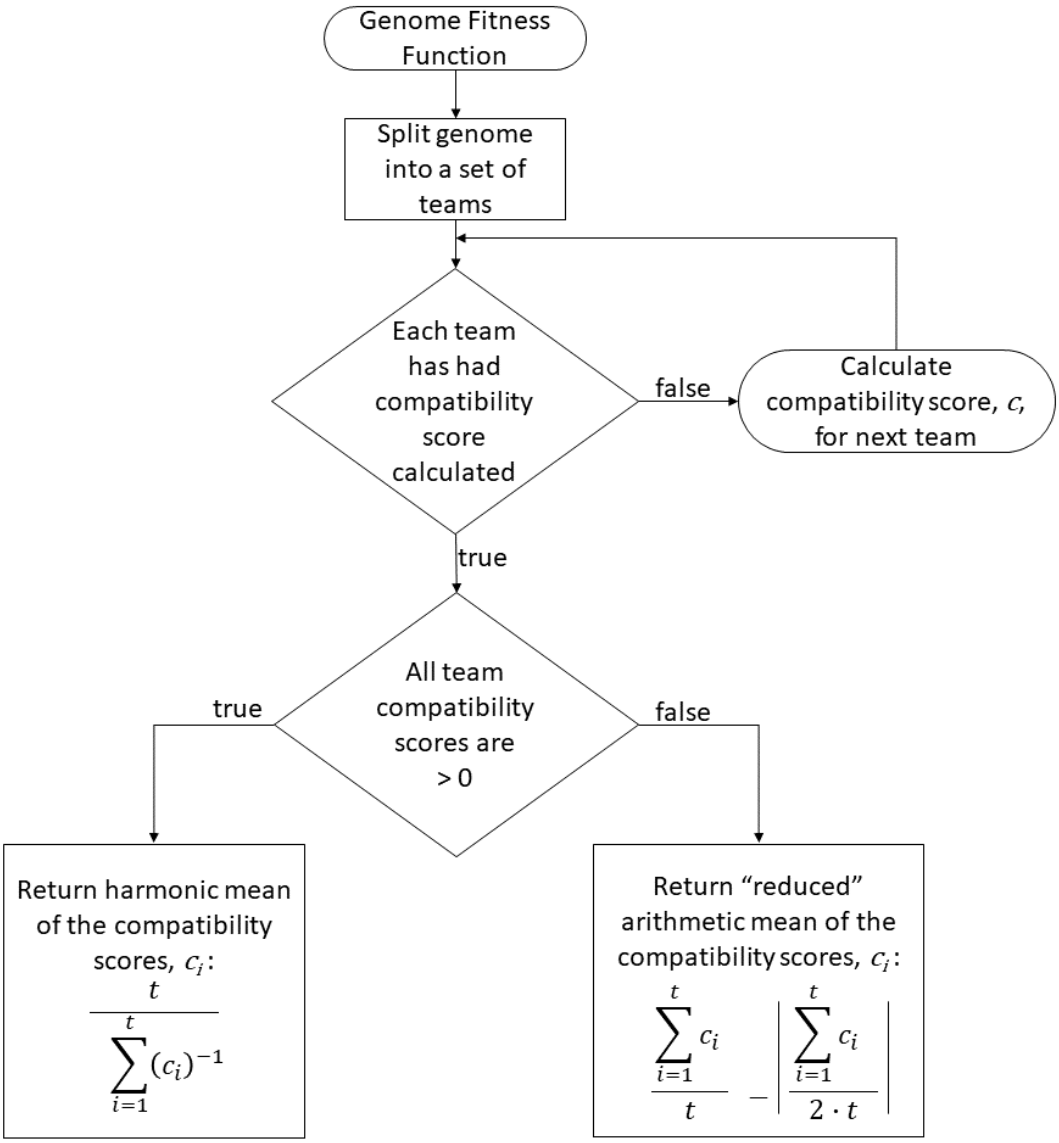

Figure 5. A flowchart describing the calculation of the fitness function value for a genome. The symbol t in the equations refers to the number of teams in the genome.

0 represents complete homogeneity of responses to this question and a value of 1 represents maximal heterogeneity of responses. If the instructor desires heterogeneity of responses on a team, this value is used the value is used directly as the attribute score; if homogeneity is desired, this value subtracted from 1 is used instead in order to reverse the quantification.

Next, if the schedule question was asked in the survey, a count is made of the number of blocks within the schedule where all teammates have indicated they are free to meet. Only blocks that meet the instructor's meeting-length requirement are counted. This count is determined by performing a Boolean and operation across the schedule arrays collected from each student. If the count is less than the instructor's minimum requirement for number of meeting blocks, then a schedule score of 0 results and a penalty point is applied. If the count meets the minimum requirement and does not exceed the instructor's desired number of meeting blocks, then the count is divided by the instructor's desired number of meeting blocks. Where a team has even more than the instructor's desired number of meeting blocks, the calculation is the same except

that the excess meeting blocks are discounted by a factor of 1/6. As examples of these three cases, if the minimum number of meeting blocks is 5 and the desired number of meeting blocks is 8, then a team where the students have 4 common free blocks in their weekly schedules would have a schedule score of 0; a team with 5 common blocks would have a schedule score of 5 / 8 = 0.625, and a team with 12 common meeting blocks would have a schedule score of [8 + (12 – 8) / 6] / 8 = 1.083. The schedule score thus ranges from 0 up to roughly 1, with values possibly exceeding 1. The use of three levels of schedule scores is done so that the schedule score monotonically increases with more compatible student schedules, but with an absolute requirement to reach the minimum number of meeting blocks and reduced emphasis on having more than the desired number.

If the instructor has specified incompatible responses for an attribute question (such as preventing a chemistry major and a physics major from being placed on the same team), then all of the response values for that attribute on a team are compiled. For each pairing of incompatible response values found in this set, the penalty counter is incremented.

If the instructor chooses to consider students' gender, then the number of men, the number of women, and the number of nonbinary students on the team are counted. If the number of women equals exactly 1 and isolated women are to be prevented, then the penalty counter is incremented. The same happens if the number of men equals exactly 1 and isolated men are to be prevented, or if the number of nonbinary students equals exactly 1 and isolated nonbinary students are to be prevented. Finally, if either the number of women equals 0 or the number of men equals 0 and single gender teams are to be prevented, then the penalty counter is incremented.

If the instructor chooses to consider students' underrepresented minority (URM) status, then the number of URM students on the team are counted. As described previously, student self-identify their identity in this regard, and it is up to the instructor to select which identities should be considered as an URM. If the number of URM students on a team is exactly 1 and isolated URM students are to be prevented, then the penalty counter is incremented.

If the instructor has named any prevented teammates, then each pair of teammates on the team is examined to see if they are prevented. Each time a prevented pairing is found, the penalty counter is incremented. Similarly, if an instructor has named any required teammates, then each teammate is examined to see if their required teammate(s) are found on their team. Each time a required pairing is not found, the penalty counter is incremented. Finally, if an instructor has named any requested teammates, then each teammate is examined to count how many of their requested teammates are found on their team. Each time this value is less than the number of requests that the instructor requires to be fulfilled, the penalty counter is incremented.

If any of the penalties have been applied, the intent is for the final team compatibility score to have a maximum value of 0. A team with a highly aligned schedule, however, might have a schedule score greater than 1, and thus a final score greater than 0. To prevent this issue, if any penalty is applied, the base score is capped at 1. The expected range of compatibility scores is 0 to 100, with negative values possible when a team fails to meet a requirement and values over 100 possible when a team meets all requirements and has students with very highly compatible schedules.

The calculation of a fitness function value for each of the 30000 genomes in a generation was found to be the slowest part of gruepr's optimization algorithm. Accordingly, much effort has gone in to reducing the fitness function calculation time. For example, gruepr skips the calculations related to any attribute question or the schedule question when it has been given a weight of 0. It likewise skips the determination of penalty points whenever the relevant requirement has not been set by the instructor.

3.5 Reporting Results

After the instructor loads into gruepr the file of survey results and then sets all of the desired teaming options, the optimization process begins. Gruepr provides a real-time display of the progress, showing the number of generations that have evolved, the top fitness value found thus

far among all genomes created, and a stability metric that suggests to what degree the optimization process seems to have settled on a solution. The instructor can choose to expand the progress indicator window to view additional details in the form of a graph showing box-and-whisker plots of the fitness function values in the genepool as a function of the generation number, an example of which is shown in Figure 6.

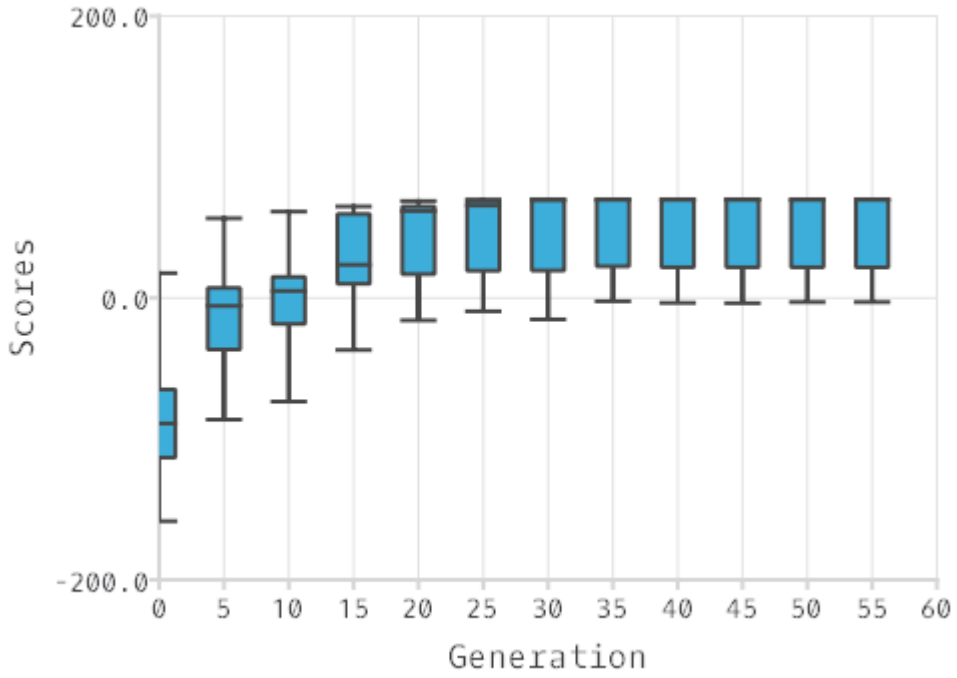

Figure 6. The progress of the optimization algorithm is shown as a box-and-whisker of the distribution of the 30000 compatibility scores within the genepool over the course of 55 generations. For clarity, values are displayed every 5 generations. After about 25 generations, the distribution of values is heavily skewed to larger values, with a long tail of lower values.

Upon completion of the optimization process, the single genome with the highest fitness value is returned from the optimization function. This particular partitioning of the students onto teams is displayed to the instructor in a tree display, initially sorting the teams and the students within each team according to the students' last names. The instructor can re-sort the teams according to the teams' compatibility scores, gender composition, responses to the attribute questions, or any of the other data collected in the survey. The instructor can also use drag-and-drop functionality to manually reorder the teams or move students around. Each team is given a default numerical team name, but these can be changed to one of the various built-in options (e.g., roman numerals, letters, chemical elements) or to instructor-input custom names.

Finally, the teams can be printed directly from gruepr or saved to a text or pdf file. Whether printed or saved, the output can be in several formats. A "student's file" is suitable for distributing to the students, since it contains only each team's name, the names and email addresses of the students on the team, and a table showing what percentage of the team is free to meet at each hour during the week. An "instructor's file" is more useful for the instructor, since it contains, in addition to the content of the student's file, student demographic and other survey data, information about the teaming options, and details regarding the optimization process. A "spreadsheet file" contains only condensed, tabularized data on which students are on which teams in a format that is suitable for reuse in gruepr or loading into other software. As an example of how the spreadsheet file might be reused in gruepr, if new teams are to be formed later in the class and the instructor wants everyone to have all new teammates, gruepr's Prevented Teammates option is able to open a spreadsheet file and use the contents to see all of the previous teammates as prevented this time.

Multiple spreadsheet files can be processed this way if a third, fourth, or nth set of teams with new teammates is to be created.

4 Results

For an individual instructor splitting a set of 32 students into 8 teams of 4 with a typical set of tea

ming options, gruepr took less than a minute to find a reasonably optimal partitioning on a laptop computer (a 2-core / 4-thread Intel® Core™ i7-6600U processor @ 2.60 GHz). The algorithm usually requires about 50 generations to have settled on a solution. During repeated runs with the same students and teaming options, very highly similar teams result, perhaps only 2-3 students switch between different teams each time. This result suggests that something close to a globally optimal solution has been found. Optimization takes longer as the optimization criteria get more complicated and, especially, as the number of students and/or teams increases. Still, splitting a set of 224 students into 56 teams of 4 using the same conditions as before took only about 8 minutes.

As a test of gruepr's ability to find an optimal set of teams, a mock class of students was created comprising 3 copies each of 32 student survey submissions (for a total of 96 "students"). Gruepr was then asked to form 32 teams of 3 students. The following optimization criteria were used so that optimal teams would have as teammates the 3 copies of each student record: isolated men and women prevented, homogeneous values of all attributes desired, and highly overlapping schedules desired. Notably, no Required or Prevented Teammates were set, so that there was no manual imposition of the known optimal solution. This test represents a difficult case for a genetic optimization algorithm, since there is a clear, single, well-ordered optimal solution that must arise from initial randomness.

As opposed to the roughly 50 generations and < 1 minute of optimization required before, this test took about 150 generations and about 3 minutes. Still, the single, optimal solution of each team comprising the three copies of one of the students was successfully found by gruepr. The idea of this test was pushed even further by creating a set of 288 "students", 9 copies of each of the 32 student records. Using the same teaming options, this time gruepr was asked to partition the students into a set of 96 teams of 3. This time, it took gruepr 285 generations and approximately 20 minutes to complete. The scaling of the number of generations and the amount of time is non-linear, since each generation takes longer to process with more students in the genome. When the algorithm auto-stopped in this test, 15 of the 96 teams created by gruepr consisted of 2 copies of one student and a different student (albeit a highly similar one) as the third teammate. All of the remaining teams were the expected 3 copies of a single student record.

Response from instructors who have used gruepr has been highly positive, and both quantitative and qualitative feedback from several instructors was published recently [33, 34]. As a software "product," gruepr has found use for a number of users. The binaries have been downloaded over 100 times. The binaries were statically compiled, meaning for both Windows and macOS, there is no installation, just a single program file that can be easily shared. Thus, the number of installations may exceed the number of direct downloads. An optional registration page was added to the software as an informal way for the author to collect the names and associated institutions of gruepr users. There are currently 34 registered users at 20 different institutions worldwide.

4.1 Conclusions and Future Outlook

This paper has presented a free and open source tool to ease the intentional formation of student teams with a high degree of flexibility for the instructor in defining an optimal team. The genetic algorithm used in gruepr is able to find reasonably optimal teams in a short amount of time, often under a minute for a class size of about 30 students. It should be noted that gruepr itself only solves the problem of creating a survey to collect student information and then using that information to form the teams. A second important component of systems like CATME is peer evaluation. While gruepr itself does not currently have an integrated peer evaluation tool, it does work well

with a free and open source peer evaluation system known as TEAMMATES [35]. This system is, like gruepr, highly flexible, and it is possible to create with it a web-based peer evaluation instrument that very nearly replicates an instructor's current peer evaluation instrument. In fact, the "spreadsheet file" output from gruepr was designed specifically for immediate upload in the TEAMMATES system.

A number of future improvements are planned for gruepr. The community of instructors using gruepr regularly contributes suggestions and requests for improved functionality. For example, attribute questions will soon be expanded to allow the possibility of more than one answer, as in a "check all that apply" response type. A more granular scheduling option will soon be explored, too. For example, allowing the times listed in the schedule grid to be broken into half-hour increments. Finally, better documentation in the form of text and video explanations will be created.

Currently, gruepr uses Google Forms as the only automated way to survey students. Alternative survey instruments are possible, since gruepr only needs a csv file of results to create teams. As described previously, gruepr's SurveyMaker can output text files to help, but much of the administration of an alternate survey instrument is labor for the instructor. In the future, automating to additional survey instruments, such as Qualtrics or SurveyMonkey, will be explored. Of particular interest is integrating gruepr with various learning management systems like Canvas or Blackboard. In these cases, not only will automation of the survey instrument be added but also automatic uploading of the results to the class site. These systems generally have published APIs that allow software tools to download data like class rosters and upload data like surveys and the composition of student groups. Automation of software to work with learning management systems is made easier through the Learning Tools Interoperability (LTI) standard [36].

References

[1] L. K. Michaelsen, N. Davidson, and C. H. Major, "Team-based learning practices and principles in comparison with cooperative learning and problem-based learning," *Journal on Excellence in College Teaching*, vol. 25, no. 3&4, pp. 57–84, 2014.

[2] B. Oakley, R. M. Felder, R. Brent, and I. Elhajj, "Turning student groups into effective teams," *Journal of Student Centered Learning*, vol. 2, no. 1, pp. 9–23, 2004.

[3] R. A. Layton, M. L. Loughry, M. W. Ohland, and G. D. Ricco, "Design and validation of a web-based system for assigning members to teams using instructor-specified criteria," *Advances in Engineering Education*, vol. 2, no. 1, pp. 1–28, 2010.

[4] E. F. Barkley, C. H. Major, and K. P. Cross, "Collaborative learning techniques: A handbook for college faculty." Jossey-Bass, 2014.

[5] L. K. Michaelson, A. B. Knight, and L. D. Fink, *Team-Based Learning: A Transformative Use of Small Groups in College Teaching*. Sterling, VA, USA: Stylus Publishing, 2004.

[6] W. Mckeachie and M. Svinicki, "Mckeachie's teaching tips." Wadsworth, 2014.

[7] J. L. Bailey and J. Skvoretz, "The Social-psychological Aspects of Team Formation: New Avenues for Research," *Sociology Compass*, vol. 11, no. 6, pp. e12 487–e12 487, 2017. [Online]. Available: 10.1111/soc4.12487;https://dx.doi.org/10.1111/soc4.12487

[8] S. E. Baroudi, S. N. Khapova, P. G. Jansen, and J. Richardson, "Individual and contextual predictors of team member proactivity: what do we know and where do we go from here?" *Human Resource Management Review*, vol. 29, no. 4, pp. 100 671–100 671, 2019. [Online]. Available: 10.1016/j.hrmr.2018.10.004;https://dx.doi.org/10.1016/j.hrmr.2018.10.004

[9] M.-I. Sanchez-Segura, M. Hadzikadic, G.-L. Dugarte-Peña, and F. Medina-Dominguez, "Team Formation Using a Systems Thinking Approach," *Systems Research and Behavioral Science*, vol. 35, no. 4, pp. 369–385, 2018. [Online]. Available: 10.1002/sres.2536;https://dx.doi.org/10.1002/sres.2536

[10] M. Borrego, J. Karlin, L. D. McNair, and K. Beddoes, "Team Effectiveness Theory from Industrial and Organizational Psychology Applied to Engineering Student Project Teams: A Research Review," *Journal of Engineering Education*, vol. 102, no. 4, pp. 472–512, 2013. [Online]. Available: 10.1002/jee.20023;https://dx.doi.org/10.1002/jee.20023

[11] F. P. Morgeson, M. H. Reider, and M. A. Campion, "Selecting Individuals in Team Settings: The Importance of Social Skills, Personality Characteristics, and Teamwork Knowledge," *Personnel Psychology*, vol. 58, no. 3, pp. 583–611, 2005. [Online]. Available: 10.1111/j.1744-6570.2005.655.x;https://dx.doi.org/10.1111/j.1744-6570.2005.655.x

[12] N. Dasgupta, M. M. Scircle, and M. Hunsinger, "Female peers in small work groups enhance women's motivation, verbal participation, and career aspirations in engineering," *Proceedings of the National Academy of Sciences*, vol. 112, no. 16, pp. 4988–4993, 2015. [Online]. Available: 10.1073/pnas.1422822112;https://dx.doi.org/10.1073/pnas.1422822112

[13] T. Henry, "Creating effective student groups: an introduction to groupformation.org," in *Proc. of the 44th ACM technical symposium on computer science education (SIGCSE '13)*, 2013, pp. 645–650.

[14] D. Y. Wang, S. S. J. Lin, and C. T. Sun, "DIANA: A computer-supported heterogeneous grouping system for teachers to conduct successful small learning groups," *Computers in Human Behavior*, vol. 23, no. 4, 1997.

[15] K. Holmberg, "Formation of student groups with the help of optimisation," *Journal of the Operational Research Society*, vol. 70, no. 9, pp. 1538–1553, 2019. [Online]. Available: 10.1080/01605682.2018.1500429;https://dx.doi.org/10.1080/01605682.2018.1500429

[16] N. Maqtary, A. Mohsen, and K. Bechkoum, "Group formation techniques in computer-supported collaborative learning: a systematic literature review," *Technology, Knowledge and Learning*, vol. 24, pp. 169–190, 2019.

[17] CATME, "Welcome to CATME - Smarter Teamwork," 2010. [Online]. Available: https://info.catme.org/

[18] R. A. Layton, M. W. Ohland, and H. R. Pomeranz, "Software for student team formation and peer evaluation: CATME incorporates Team-Maker," in *Proc. of the American Society for Engineering Education Annual Conference & Exposition*, 2007.

[19] Z. C. Ani, A. Yasin, M. Z. Husin, and Z. A. Hamid, "A method for group formation using genetic algorithm," *International Journal on Computer Science and Engineering*, vol. 2, no. 9, pp. 3060–3064, 2010.

[20] L. E. Agustín-Blas, S. Salcedo-Sanz, E. G. Ortiz-García, A. Portilla-Figueras, and Ángel M. Pérez-Bellido, "A hybrid grouping genetic algorithm for assigning students to preferred laboratory groups," *Expert Systems with Applications*, vol. 36, no. 3, pp. 7234–7241, 2009. [Online]. Available: 10.1016/j.eswa.2008.09.020;https://dx.doi.org/10.1016/j.eswa.2008.09.020

[21] M. Craig, D. Horton, and F. Pitt, "Forming reasonably optimal groups (FROG)," in *Proc. of the 16th ACM International Conference on Supporting Group Work*, 2010, pp. 141–150.

[22] T. R. Henry, "Forming productive student groups using a massively parallel brute-force algorithm," in *Proc. of the World Congress on Engineering and Computer Science*, 2013, pp. 23–25.

[23] S. Graf and R. Bekele, "Forming heterogeneous groups for intelligent collaborative learning systems with ant colony optimization," in *8th International Conference on Intelligent Tutoring Systems*, 2006, pp. 217–226.

[24] D. Bretthauer, 2001. [Online]. Available: https://opencommons.uconn.edu/libr_pubs/7

[25] J. Hertz, "gruepr - wiki," 2021. [Online]. Available: http://bit.ly/Gruepr

[26] T. Q. Company, “Qt | Cross-platform software development for embedded & desktop,” 2021. [Online]. Available: https://www.qt.io/

[27] Icons8, “Icons8 Homepage,” 2021. [Online]. Available: https://icons8.com/

[28] OpenMP, “Home - OpenMP,” 2018. [Online]. Available: https://www.openmp.org

[29] E. Falkenauer, “A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems,” *Evolutionary Computation*, vol. 2, no. 2, pp. 123–144, 1994. [Online]. Available: 10.1162/evco.1994.2.2.123;https://dx.doi.org/10.1162/evco.1994.2.2.123

[30] H. O. G. Algorithms and L. Davis. New York, NY, USA: Van Nostrand Reinhold, 1991.

[31] L. J. Eshelman and J. D. Schaffer, “Preventing premature convergence in genetic algorithms by preventing incest,” in *Proc. 4th International Conference on Genetic Algorithms*, 1991, pp. 115–122.

[32] A. Wibowo and P. Jamieson, “Using simple ancestry to deter inbreeding for persistent genetic algorithm search,” in *Proc. International Conference on Genetic and Evolutionary Methods (GEM’12)*, 2012, pp. 23–29.

[33] J. L. Hertz, D. Davis, B. O’connell, and C. Mukasa, “Grueprː an open source program for creating student project teams,” in *2019 ASEE Annual Conference and Exposition*, 2019, pp. 26 537–26 537.

[34] J. L. Hertz and S. F. Freeman, ““gruepr, an open source tool for creating optimal student teams,” *2020 ASEE Virtual Annual Conference and Exposition*, 2020.

[35] TEAMMATES, “TEAMMATES - Online Peer Feedback/Evaluation System for Student Team Projects,” 2021. [Online]. Available: https://teammatesv4.appspot.com/web

[36] MS Global Learning Consortium, “Learning Tools Interoperability Core Specification 1.3,” 2021. [Online]. Available: https://www.imsglobal.org/spec/lti/latest/