Introducing junto: a Web Tool to Build Project Teams based on a Bidding Strategy

Akhil Krishna Mohan, University of Illinois at Urbana-Champaign

Mathematics and Computer Science major, Class of 2021.

Priyanka Dey, University of Illinois at Urbana Champaign

Priyanka Dey is currently pursuing a B.S. degree in Computer Science Statistics. Her research interests include natural language processing, data mining, and optimization algorithms.

Sizhi Tan, University of Illinois at Urbana Champaifn

Sizhi Tan is currently pursuing a Bachelor's Degree in Computer Science at the University of Illinois at Urbana Champaign.

Dr. Blake Everett Johnson, University of Illinois at Urbana - Champaign

Dr. Blake Everett Johnson is a lecturer and instructional laboratory manager in the Department of Mechanical Science and Engineering at the University of Illinois at Urbana-Champaign. His research interests include experimental fluid mechanics, measurement science, and engineering education. He oversees undergraduate laboratories in fluid mechanics, fluid dynamics, and heat transfer. Pedagogically, Dr. Johnson employs active learning, inquiry-based laboratory instruction, and any initiative that empowers students to do hands-on learning. Additional service interests include teaching and leadership training for graduate students, enhancing communication education for undergraduate engineering students, developing evidence-based design project team formation strategies, and improving engineering design curricula.

Prof. Wade Fagen-Ulmschneider, Computer Science, University of Illinois at Urbana-Champaign

Wade Fagen-Ulmschneider is a Teaching Associate Professor of Computer Science at the University of Illinois at Urbana-Champaign (UIUC). With a passion for data, he teaches thousands of students each year in his courses on Data Structures, Data Visualization, and Data Science. He was selected as one of the National Academy of Engineering's Frontiers of Engineering Education scholars, awarded the Collins Award for Innovation Teaching, and has been consistently ranked as an excellent instructor by his students for the past ten years. His work on data visualizations has been used by governors of multiple states, featured by websites including Popular Mechanics and The Verge, and has been viewed by millions of readers.

Prof. Mariana Silva, University of Illinois at Urbana-Champaign

Mariana Silva is a Teaching Assistant Professor in Computer Science at the University of Illinois at Urbana-Champaign. She has been involved in large-scale teaching innovation activities, such as the development of online course content and assessments for the mechanics course sequence in the Mechanical Science and Engineering Department and the numerical methods class in Computer Science. Silva is currently involved in two educational projects involving the development of online assessments for computer-based testing and creation of collaborative programming activities for computer science classes. She is also involved in a project that aims to create a software that facilitates collaborative problem-solving activities in classrooms, through which both the instructors and students learn more about collaboration skills. Silva is very passionate about teaching and improving the classroom experience for both students and instructors. She has been included in the List of Teachers Ranked as Excellent five times and has received the Engineering Council Outstanding Advisor Award every year since 2014.

©American Society for Engineering Education, 2020

Introducing `junto`: a Web Tool to Build Project Teams based on a Bidding Strategy

Abstract

This work presents a web application created to help instructors assign students to group projects, with an algorithm that optimizes student satisfaction, gives students the opportunity to select a team member, and reduces time needed for an instructor to create teams. Our approach focuses on two main aspects: (a) it gives the student the ability to apply weights to their project choices (instead of just ranking them) and (b) it provides students with the opportunity to select a classmate to be partnered with. We implemented a genetic algorithm that assigns students to projects in order to maximize the fitness function, defined as a multi-objective function to increase student satisfaction, decrease the variance of team sizes, and optionally decrease the GPA variance among team members.

Introduction

`junto` started off as a fast, portable solution to senior design project team formation. Releasing forms, collecting student data, validating student data, and assigning teams, all steps in the process, take place in the first few days of a semester. The massive amounts of work needed, and the scarcity of time to complete the process called for automation. `junto` seeks to provide solutions for a robust interface for instructors to set up their course, and students to select their preferences. It works on a bidding system, allowing students to indicate their relative interests in projects by dividing up their points into bids for the projects. `junto` also gives students the option of selecting a partner, in which case they are assured to be placed in the same group. For instructors, it allows inclusion of GPAs and group sizes as parameters for optimization. Using a genetic algorithm, it employs a novel and scalable solution strategy to an assignment problem that has mostly been solved by hand. The automation allows instructors to reduce their workload in a busy time of year, and also remove both human effort and human bias in the team assignments. While `junto` was created to help solve a niche problem, the final application could serve as a template to transform an algorithm into a portable web-service.

While there are other works on efficient ways to create teams for the senior design project, our approach combines the robustness of some prior approaches with the portability of modern software solutions. A genetic algorithm was used by researchers at the University of North Carolina, Charlotte[1] for the same problem, to varying degrees of success. The inputs to the algorithm were binary choices (yes/no) for each project, based on the student's ranked top 3 preferences. Our approach, however, allows students to bid on projects, dividing up their points to

better indicate their relative interest in projects. Typically, the form will enforce splitting up points across at least $4 - 6$ projects, allowing us to gauge more than just the top 3 preferences for every individual.

Another paper by researchers in Rowan University[2] has proposed a portable solution to the same problem, and succeeded in its task to decentralize the work involved. Using Google Forms to collect student data paired with the Google Apps Script platform, they created a program that worked entirely on the cloud, serving as a fast and efficient solution. `junto` intends to take inspiration from the solution and the web-app with an end-to-end integration similarly portable solution for instructors and students. The algorithm is written in pure Python, and is more versatile than the Google Apps Script, forming a more robust application.

Two other papers, one from researchers at Eastern Washington University[3], and the University of Calgary[4], have proposed solutions to the problem by using a list of preferences from students as input. There exists no literature on the effectiveness of the bidding system in solving our problem, and `junto` intends to fill that niche. Another point of interest is that `junto` allows students to choose partners. `junto` might also allow us to study the effect of this choice in making effective teams.

The aforementioned papers consider the fairness of an algorithm in the assignment of teams. In some cases, the approach is to use a student's GPA to break ties when students compete for the same spot on a project. Using Junto, the instructor has the option to upload a roster with GPA and use this data to balance out average GPA across groups. Junto does not break ties using GPA, however, as its randomization takes precedence for the fair tie-breaking.

`junto` can also be compared to CATME[5], a software tool to make teams. CATME takes a more individual-centric approach, querying instructors to complete subjective surveys about their students. Combining peoples' schedules, work ethics, and interests with the survey data, CATME provides holistic teams, but requires many inputs. `junto`, in contrast, cares only about preference data, and optionally factors in student GPA while balancing teams. In the workflow for creating capstone project teams, `junto` provides a specialized option, tuned to its requirements.

The scalability of genetic algorithms is an important factor in making `junto` a portable web-app solution. The major computations involved are highly parallelizable and thus give us a more versatile tool, capable of handling many such jobs in a short time. The code will be open-source when the web-app launches and the app can be customized to be used in different environments.

Online Application

Users interact with `junto` through a web-based interface. Every instance of `junto` begins with a user who is interested in using the tool for team formation. That user (often a course instructor) creates a new "`junto` Instance" with two or more projects for participants (often students) to rank their personal project preferences. Additionally, the instance creator can set various parameters (ex: total number of points a user allocate, the minimum number of projects that must be allocated, the maximum number of points a user can allocate to a single project, and others) for their specific instances or use the default values.

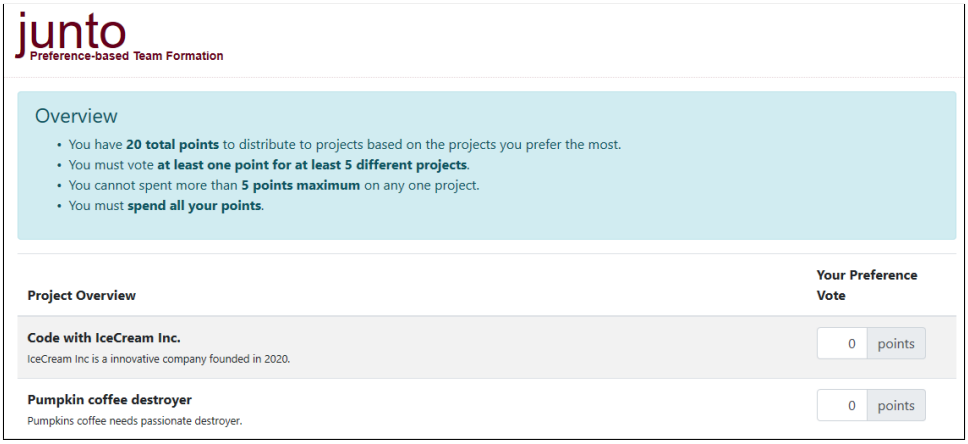

Figure 1: Screenshot of the participants' project preference selection interface inside of the `junto` web application.

As soon as the creator finalizes all projects, a unique URL is generated that is to be provided to participants. Participants use the unique URL to log in and provide their personal preferences for the set of projects. Figure 1 shows the participants' interface to bid on projects. At any time the creator can review the current participants project preferences via an interactive data visualization. If any projects have few participants, or low bids, or for any other reason, `junto` instance creators can decide to remove projects entirely from the set of projects.

Instance creators have several ways to customize the output of the algorithm, including the ability to manually "bind" students to set project, and to tune any of the algorithm parameters (described in detail later in the paper). Instance creators may iterate through this process, running the algorithm on demand until they are satisfied with the results. All results can be exported via a CSV format for use in Microsoft Excel, Learning Management Systems (LMS), or other programs/tools.

The `junto` web application is built using React JS[6] and Bootstrap HTML[7] for the front-end, ExpressJS[8] and Sequelize[9] for the middleware, and a Flask[10] backend paired with Python code. We represented our data using the following models:

- *JuntoInstance*: Each time an instructor submits a new set of projects with bidding instructions, a new JuntoInstance record is created
- *Project*: All projects entered by any instructor using Junto will be stored in this table
- *ProjectBid*: Each time a user bids on a project, a new record will be added to this table
- *User*: All users (students and instructors) are stored in this table

Problem Definition

Capstone projects are required senior level courses that represent the culmination of a student's engineering education. These projects tend to mimic real engineering projects, forcing students to

deal with real-world considerations. Given that students need to work in teams for these projects, we seek to provide a solution to:

- remove human effort in creating teams
- maximize overall student satisfaction to create motivated groups
- minimize variance in group average GPA (secondary, to create balanced groups)
- minimize the time spent in achieving the above

Student constraints

We use a bidding system to determine students' preferences in the projects. Students are given a total number of points P_{total}, and a maximum number of points that they can bid on a certain project, P_{max}. The model also enforces students to pick a minimum number n of projects to split their points across. This serves to create better data for the algorithm, trying to avoid problems without a solution that satisfies all the requirements. Students split up their bidding points across all the available projects, where higher bids indicate greater interest in a given project. When a student selects a partner, the preferences are duplicated, and the pair of students is treated as a single student for the purposes of the algorithm. The algorithm can be executed once all students have made their entries.

Optimization algorithm

We phrase the problem as an unconstrained maximization problem where the objective function f is defined as

$$f(x) = \frac{s_x}{s_{max}} + \gamma_1 \sigma^2_{num} + \gamma_2 \sigma^2_{gpa}$$

where

- s_x - total sum of student satisfaction in the current configuration,
- s_{max} - maximum theoretical satisfaction
- σ^2_{num} - (normalized) variance in number of students per group,
- γ_1 - weight of σ^2_{num}'s contribution to the fitness,
- σ^2_{gpa} - (normalized) variance in average GPA per group,
- γ_2 - weight of σ^2_{gpa}'s contribution to the fitness,

and the maximum theoretical satisfaction s_{max} are calculated assuming that every student was placed at their most preferred project, i.e., the sum of the highest bid from each student.

We use a genetic algorithm (GA) to solve this optimization problem, which is an example of a black-box optimization technique that comes from a broader class of algorithms called evolutionary algorithms. GAs, first introduced by John Holland in 1970[11], use operations inspired by Darwin's theory of evolution, namely mutation, crossover, and selection to arrive at their results. To setup the GA, we first model a potential solution to our problem as a *chromosome*. If

we have n students to split into teams for k projects, we encode the chromosome as a bit-string, an array of length n where each index of the array contains an integer value that can be $0, 1, 2, ...k-1$. Each index of the array represents a student and each value represents a project tentatively assigned to the student.

To determine how "good" any given solution is, we use the objective function as a proxy for fitness. Given a chromosome c, the fitness ($f(c)$), is given by calculating the above objective function, where s_{max} is fixed based on the input data, and γ_1 and γ_2 are constants supplied at run-time.

We now define the steps of the optimization process:

Initialization: We initialize a population of M randomly generated chromosomes. The run-time of the algorithm is proportional to the population size (M), which we had in the ballpark of $500 - 2500$. In our formulation of the problem, randomly generating a chromosome corresponds to setting each bit of the chromosome to an integer value in $[0, k-1]$ with equal probability. This step happens only once, at the start of the algorithm. The next steps are repeated in a loop.

Repair: We use a mechanism to fix the infeasibilities in projects, by attempting to smartly reassign students in crowded projects. If we find a student in a crowded project (i.e. a project with a team size greater than the maximum allowed), we try to reassign this student to a project he or she bid a higher amount for. Another strategy we use in tandem is to reassign a student if it increases the overall satisfaction in the assignment. If this is not possible, the algorithm changes nothing about the assignment. While this may seem ineffective, repair proves useful in early generations, and γ_1's effect on the fitness generally leads to balanced teams in later generations.

Selection: After assessing the fitness of each of the M chromosomes in the population, we sort the population based on the fitness. Our algorithm implements *elitism* which means the fittest chromosomes live untouched from one generation to the next generation. In our case, we have a parameter, n_{keep} that tells us how many chromosomes to retain across generation.

Crossover: Using the sorted population, a new population is generated from the best chromosomes of the previous generation. *Parent* chromosomes are selected by a randomized mechanism that gives relative weightage based on fitness (i.e. more fit individuals are more likely to be picked as parents). Once we have picked two parents, we splice the individual chromosomes together, selecting project assignments from each. We begin by copying over one parent's chromosome and swapping project assignments with the other. For each project $1, 2, ..., k$ we choose a team assigned by either one of the two parents. An additional parameter n_{cross} tells us the maximum number of team assignments that can be swapped between the parents. A successful swap (i.e. generating an individual from the selected parents) is called a *crossover*, which happens with crossover probability p_c. The other individuals, corresponding to a probability $1 - p_c$, are generated at random. We continue the process until we have generated $M - n_{keep}$ individuals for the new generation of the algorithm.

Mutation: In genetic algorithms, the mutation operator is required to maintain genetic diversity, avoid getting trapped in local solutions, and potentially speed up convergence. Here, mutation occurs with mutation probability p_m, swapping two randomly chosen indices in the

chromosome.

Termination: Picking a sufficiently large number K as a check, if the maximum fitness of the population remains within a given tolerance (tol) for K generations, we claim to have converged on a solution and can terminate. In our trials, we used a K-value of around 30. Convergence is a preferred outcome for the algorithm. However, if the algorithm fails to converge, and we reach a predetermined maximum number of generations (max_iter), we stop the algorithm and return the solution with highest fitness.

In summary, the GA algorithm requires 10 different input parameters, namely

- Population size (M): number of individuals generated in one generation (default = 500)
- Crossover Probability (p_c): determines how likely a crossover is to take place (default = 0.95)
- Mutation Probability (p_m): determines how likely mutation is to take place (default = 0.1)
- Maximum number of iterations (max_iter): termination condition if the algorithm does not converge (default = 1000)
- Fitness tolerance (tol): tolerance for the stopping criteria defined by the change in the fitness function (default = 10^{-6})
- γ_1: a float in $[0, 1]$ to determine relative weight of σ^2_{num}'s contribution to the fitness function (default = 0.5)
- γ_2: a float in $[0, 1]$ to determine relative weight of σ^2_{gpa}'s contribution to the fitness function (default = 0.0)
- n_{keep}: determines how many "elite" chromosomes survive into the next generation untouched (default = 2)
- n_{cross}: number of projects that are swapped between parent chromosomes in the crossover step (default = 2)
- $max\_per\_project$: maximum number of participants per group/project (default = 5)

Since this is an algorithm that inherently uses randomization, there can be significant differences in results with constant input parameters. Note that the default value for γ_2 is set to 0, which ignores the GPA calculations, since this data is optional to run the algorithm.

Trials, Testing, and Tuning

We tested the `junto` algorithm using unidentifiable student preference data provided by a instructor of senior design courses. The data consisted of 134 students that had to be distributed among 27 projects. 34 students selected a partner, and hence they were both considered as one entry for the algorithm. Thus, the algorithm had 100 rows of preference data, with students having to split 25 points across the 27 projects. They had to choose a minimum of 8 projects, and the maximum preferred team size was 5. We used all the default values, except for $\gamma_2 = 0.2$.

With the testing data, `junto` consistently placed all students in a project they had bid for. It also consistently made 26 teams of 5 students, and 1 team of 4 students, which was the best possible with 134 students and 27 students. The average variance for GPA average per group was found to be near 0.0225, a parameter the algorithm was trying to minimize. All runs of `junto` achieved over 80% of the theoretical satisfaction ($\frac{s_x}{s_{max}}$), based on every student getting their first preference. On average, it took 350 generations to converge at a solution, which translates to under 3 minutes on most systems.

User study 1: a Mechanical Engineering capstone course

`junto` was used by the instructor of a senior capstone course in the Mechanical Engineering Department of a large public research university in Fall 2019 and Spring 2020 semesters. While the front end was not yet complete, the backend code proved effective both semesters. This section summarizes background information about the course, and the method previously used to assign students to projects, the results from `junto` and the instructor feedback.

Course description

The capstone design course is a one-semester program that serves approximately 100 students (about 20% female) in the senior class each semester. Twenty-four teams of 3–5 students each are formed to work on real-world engineering problems, most of which are conceived and funded by industry partners, who make a financial gift to the department in return. Each industry sponsor deserves a team of students that possesses a fair level of academic ability, so the instructor seeks to minimize the variance of average student GPA across the 24 teams. The students are allowed to specify one colleague to be partnered with, but the rest of the team is assigned by the course instructor. Partners then complete a single project preference survey where they assign numerical weights to their most-preferred projects; students who do not specify a partner complete their own project selection survey. Numerical weights of 1–5 must be assigned to a minimum of eight projects and a maximum of 15 projects, and they are required to sum to 25.

Students submit project preference surveys by midnight of day one of the semester. The course instructor guarantees project and team assignments by midday of day three. From the student deadline to submit project preferences to the time they receive their assigned project and team, minus two nights of 7 hours' sleep, there are about 22 waking hours remaining to complete this process. This time must also be split between home and other work responsibilities, including assigning TAs and faculty advisers to the projects. Based on the Spring 2020 calendar, the course instructor estimates 10 hours of reasonable time available for working on the team assignments.

Creating the teams "by hand"

As a baseline for comparison, a manual procedure for assigning students to teams begins with identifying the least-popular project, according to the project preferences survey results, and assigning team members to that project based on those who rank the project most highly. The rest of the teams are then constructed using this method in order of increasing project popularity. As the teams are built, the instructor monitors the average GPA of each and performs spot-switching of students from team to team to adjust the average GPA of each team as near as possible to the

mean GPA of the class, and seeking to maximize the student satisfaction of each team. While the manual algorithm is sensible and fairly efficient, the entire process nonetheless takes many hours to perform with no way to know whether the best set of team assignments is found.

Using `junto` to create the teams

Using `junto`, many generations of team assignments are rapidly created and sorted according to total student satisfaction. The instructor of the course used all the default parameters, and $\gamma_2 = 0.5$. `junto` was able to quickly provide the instructor with a starting point of 24 complete team assignments with high student satisfaction from which to perform spot-switching, if needed. For a class of this size, the instructor was able to decrease the time spent making the teams by at least three hours when compared with the manual process.

Student impressions

Students in the Fall 2019 Mechanical Engineering capstone course were surveyed late in the semester for their opinions of how the project assignments had been made. The survey was given to all students and did not gather any personal identifiers. Out of 87 enrolled students, 72 took the survey. Of these, ten students indicated that they had been pre-assigned to their team. The results of the survey were processed by filtering those students out, yielding 62 completed surveys, for a response rate of 82% among students who were not pre-assigned to their project.

The students were asked to rate the project selection process with 5-point Likert scale from Extremely Bad to Neutral to Extremely Positive. The distribution of student answers to the question are shown in Figure 2, indicating that only about 10% of the respondent students had a negative impression of the project selection process, while the other 90% of students held a neutral or positive opinion of it. Indeed, more than 75% of the respondents held a positive opinion of the tool that was used to make the team assignments. It is unclear whether the students were all referencing their answer against a common baseline method for team and project assignments, such as CATME, which is used in other courses for making team assignments. Nonetheless, their responses indicate overwhelmingly positive impressions of the `junto` system.

Students were also asked to rate how they felt about their assigned projects and team-mates. These survey questions were given to students immediately after the team assignments and again at the end of the semester. Figure 3 shows that 70% of the students were satisfied with their assigned project at the beginning of the semester, and 12% indicated negative feelings. By the end of the semester, 62% of the students reported being satisfied with the project assignment while 20% were not satisfied.

Similar results were obtained when students were asked to rate their sentiments about their assigned teams. Figure 4 shows that 73% of the students were satisfied with their assigned team-mates at the beginning of the semester, and 2% indicated negative feelings. By the end of the semester, 75% of the students reported being satisfied with their assigned teams while 7% were not satisfied (there was decrease in the number of students with neutral opinions).

Overall, these results are very encouraging, since it reveals an overall positive perception and it does not reflect a drastic change of opinion as the semester progresses. Indeed, based on anecdotal feedback, instructors have noticed a decrease in the number of interpersonal conflict within the

student teams compared with previous versions of this course. Unfortunately, the instructors did not collect any data prior to the use of `junto` for a more rigorous comparison.

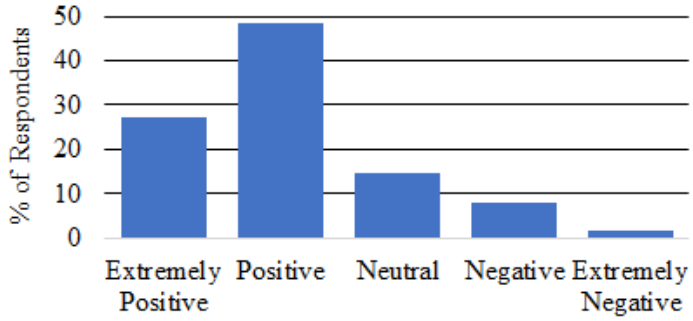

Figure 2: Survey results illustrating student's perception about the team selection and project assignment process.

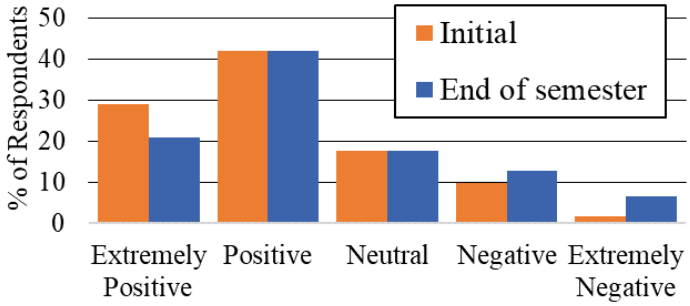

Figure 3: Survey results illustrating student's sentiments about their assigned project. Students responded to the same survey the week after the assignments were made (Initial), and at the end of the semester.

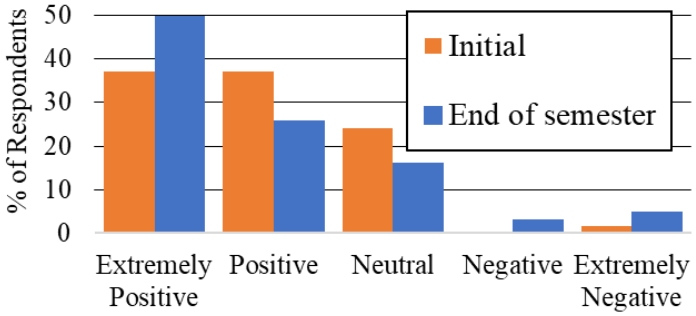

Figure 4: Survey results illustrating student's sentiments about their assigned team-mates. Students responded to the same survey the week after the assignments were made (Initial), and at the end of the semester.

User study 2: a Bio-Engineering capstone course

We also had the chance to benchmark `Junto`'s team assignment against the currently-in-use human assignment, using it on data from a capstone design course in the Bioengineering Department in the same public research university. The instructor of the course reported spending over 6 hours creating the team assignments in Fall 2019. We were able to provide the same data to the `junto` application, and obtain results in less than 10 minutes. For this team assignment, the instructor did not use GPA or the option to select partners. The average satisfaction of the teams generated "by-hand" was 13.11, whereas the average satisfaction from `junto` was 13.05. These are very similar numbers, however `junto` would have saved the instructor over 5 hours of work. The instructor is currently gathering data for this one-year course (surveys, reports, and student performance) and has plans to use `junto` next year.

Conclusions

`junto` sought out to introduce an accessible, portable, quick, and efficient tool for the capstone team formation workflow. Given the time constraints for instructors and students, we believe this implementation has successfully met our goals to minimize the effort and time spent in creating good team assignments.

Despite successful results, our approach has some drawbacks. Since the algorithm used is a black-box optimization technique, it can be difficult to debug if results are not ideal. Extreme data and faulty parameters can also lead to suboptimal results. While the program has been tested extensively with real-world data, it has only been used in few contexts. Widespread use of `junto` is needed to discover and address more specific issues with the application. The open source project lends a mechanism to track ongoing issues, and continually update the application to ensure its effectiveness in a wider range of situations.

There is already scope for widespread use of the application, however instructors often have to visualize and pre-process the data before using it as input. A future update to the app will allow a variety of pre-processing and visualization features. These serve to give the instructor more feedback on the student survey data, and serves to create a more informed team assignment. Further improvements can be made by combining these pre-processing steps with the existing software, for example, excluding projects with low student interest. Another future improvement will be to support a ranked preference list as student input. While the bidding system is central to `junto`'s philosophy, instructors should be given the choice to revert to a ranked preference list if preferred.

`junto` shows promise as a robust, customizable tool in the workflow, and has attracted interest from instructors from other universities. While the project is in its infancy, we are hoping the portability and accessibility of `junto` will allow us to collect data from its use in various environments upon launch. Using a data-driven approach, we can quantitatively measure the effectiveness of `junto`'s approach in the sphere of capstone team formation.

References

[1] P. L. Schmidt, D. Hoch, W. F. Heybruck, D. L. Sharer, S. G. Teng, and E. Sharer, "An optimization routine for assigning students to capstone project groups," ASEE, 2011.

[2] S. Bakrania and B. J. Johnson, "A cloud-based tool for assigning students to projects," ASEE, 2015, paper Id #11538.

[3] B. M. Michaelis and D. H. Bae, "Optimizing capstone team selection," ASEE, 2019, paper Id #26682.

[4] T. Freiheit and J. Wood, "An algorithm for project assignment in capstone design," ASEE, 2007.

[5] 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, pp. 1–28, 2010.

[6] F. I. Community, *React*, v16.5.2 ed., Facebook Open Source, https://reactjs.org/, 2013, a JavaScript library for building user interfaces.

[7] T. J. Otto. M, *Bootstrap*, v3.3.7 ed., https://getbootstrap.com/, 2011, an open source toolkit for developing with HTML, CSS, and JS.

[8] S. Holowaychuk. T.J., *Express.JS*, v4.16.3 ed., https://expressjs.com/, 2010, node.js web application framework.

[9] *Sequelize*, v5.21.3 ed., https://sequelize.org/, an Object-Relational Mapping Software for SQL databases.

[10] A. Ronacher, *Flask*, v1.1.1 ed., https://www.palletsprojects.com/p/flask/, 2010, a lightweight WSGI web application framework.

[11] J. H. Holland, *Adaption in Natural and Artificial Systems*. MIT Press (re-issue), 1992.