Balancing diversity and preferences: Mono-objective versus Multi-objective Approaches to the Educational Team Formation Problem

FRUTOS-RODRIGUEZ D., University of Twente, The Netherlands
BARRIOS-FLEITAS Y., Formal Methods and Tools, University of Twente, The Netherlands
LALLA E., High-tech Business and Entrepreneurship department, University of Twente, The Netherlands

This study explores the Team Formation Problem (TFP) in an educational case study that aims to create project teams balancing cognitive diversity and student preferences—a computationally complex task due to its NP-hard nature. The research investigates three nature-inspired metaheuristics: an enhanced mono-objective Genetic Algorithm (GA), a multi-objective Non-dominated Sortig Genetic Algorithm (NSGA-II), and a discrete Particle Swarm Optimization (PSO) approach adapted to the TFP. All algorithms are modified to strictly enforce domain-specific constraints and are evaluated on real and reduced datasets representing up to 278 students. Results show that PSO consistently outperforms both genetic algorithms in terms of solution quality, although at a higher computational cost. Meanwhile, NSGA-II retains its advantage of offering a diverse set of trade-offs via Pareto-optimal fronts, making it especially suitable for scenarios requiring flexible or value-driven decision-making. The study contributes open-source tools and empirical benchmarks to support reproducible research in team optimization and algorithmic approaches to educational design.

Keywords: Genetic Algorithms, NSGA-II, Particle Swarm Optimization, Team Formation Problem, Cognitive Diversity, Preference Alignment, Constraint Satisfaction

ACM Reference Format:
Frutos-Rodriguez D., Barrios-Fleitas Y., and Lalla E.. 2025. Balancing diversity and preferences: Mono-objective versus Multi-objective Approaches to the Educational Team Formation Problem. In . ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Team-based learning (TBL) has become a widely adopted instructional strategy in higher education due to its proven impact on deep learning, critical thinking, and student engagement [41, 43]. However, decades of research on working groups remind us that simply grouping students is not sufficient to guarantee such benefits [17, 39]. Instead, the success of TBL depends on the careful formation of effective teams: those capable of performing assigned tasks, maintaining member satisfaction, and remaining viable for future collaboration [18, 30].

TScIT 42, July 1, 2025, Enschede, The Netherlands

Traditional team formation strategies in education, such as random assignment, instructor-based matching, or self-selection, often fail to account for important task and social dynamics, which largely shape team effectiveness [37]. Team effectiveness models, such as the classical Input-Process-Output (IPO) and Input-Mediator-Output-Input (IMOI) frameworks highlight the critical role of input factors in the team formation stage, with particular emphasis on task relevance and interpersonal diversity [25]. Guided by this theory and evidence that cognitive variety often outperforms raw ability in complex problems [23], this paper focuses on two attributes instructors can measure before teams start work:

(1) Project-preference alignment, which consists of assigning students to projects they genuinely want to tackle. Project-preference alignment increases intrinsic motivation, and curbs social loafing [8, 20, 47].
(2) Cognitive diversity, which in the context of this study consists of balancing complementary cognitive styles captured via Belbin roles [2]. Cognitive diversity broadens the idea pool and buffers teams against impasses [5].

The challenge of optimizing team formation around these multifaceted objectives is known in operations research literature as Team Formation Problem (TFP) [28]. Although ideally instructors could create optimal teams based on their pedagogical criteria, the reality is that this problem often becomes computationally infeasible (also known as an NP-hard problem) [33], as the number of team members and teams to be formed grows, increasing the number of possible combinations of team members exponentially and gradually resembling the setup of the Knapsack problem [42].

As a consequence, researchers have increasingly turned to heuristics and metaheuristics as data-oriented approaches designed to guarantee a (nearly) optimal formation of teams. A recent systematic literature review found that approximately 60% of the reviewed studies implemented genetic algorithms (GA), with particle swarm optimization (PSO) a distant but growing alternative [34]. However, two key limitations in existing research motivate this study:

(1) **Most studies use generic metaheuristic operators** (e.g., standard crossover and mutation) without incorporating domain-specific constraints or knowledge (such as ensuring team feasibility or maximizing cognitive diversity).
(2) **Comparative studies between mono- and multi-objective algorithms remain scarce**, especially in the context of

educational TFPs with dual objectives (preference satisfaction and diversity).

This study aims to compare a constraint-compliant GA with a discrete PSO for the formation of project teams that jointly maximize project preference satisfaction and cognitive diversity in a cohort of 400 first-year computer-science students. Furthermore, it compares these mono-objective algorithms with a multi-objective NSGA-II implementation. These algorithms will first be tested in test environments in small-scale reduced datasets and then applied to large-scale datasets with real anonymized data from the case study discussed. In doing so, this paper addresses the following research questions:

> **Research Question 1:** Which metaheuristic, GA or PSO, produces higher-quality teams (in terms of our dual effectiveness objective) within practical runtime limits?
> **Research Question 2:** What type of metaheuristics, mono-objective or multi-objective, produce higher-quality teams, and what is the diversity-runtime tradeoff for these?
> **Research Question 3:** How do the findings of RQ 1 and 2 affect TBL in higher education, and how can the findings of this research be applied to similar TFPs?

In this paper, section 2 discusses relevant work in the research and formulation of the TFP and past metaheuristic implementations as attempts to solve it, section 3 formulates a methodology for the implementation and evaluation of the algorithms with respect to the datasets used, section 4 visualizes the experiments, section 5 contrasts the experiments' results with those of existing research, and discusses its applications to general educational contexts, section 6 concludes the findings of the conducted research, and section 7 illustrates valuable directions for future research.

2 BACKGROUND

2.1 The Team Formation Problem

A 2021 systematic literature review on TFPs [28] defines a taxonomy of the TFP, categorizing it into two classes, each with three subclasses, based on a comprehensive but non-exhaustive review of more than 100 articles. In summary, it defines assignment-based models as models that form teams by directly matching candidates to roles or groups based on suitability scores, costs, or interpersonal preferences typically focusing on structured optimization. Juarez et al. then define community-based models as models that select teams from social networks by optimizing skill coverage and collaboration strength, emphasizing candidate interactions and connectivity. The given case study falls into this specific subclass, making it a suitable candidate for exploratory research in this field. Moreover, the review discusses various approximation methods to (nearly) solve the TFP, focusing on heuristics, approximation algorithms, and metaheuristics.

2.2 Selected Attributes

As mentioned in the introduction of this paper, two attributes were selected to assess the suitability of candidate team formations. These were decided firstly based on data availability (as the data was collected prior to this research), and secondly based on the suitability of either attribute for measuring the suitability of team formations, as elaborated below.

2.2.1 Attribute 1: Satisfaction with the selected project and relevance in educational contexts. Previous research on TBL in higher education shows that student autonomy, particularly in task or project selection, enhances achievement motivation [8], which, in turn, has been shown to reduce social loafing and disengagement in group work [47]. When students feel ownership over the task, they are more likely to contribute actively and consistently. For this reason, satisfaction with project assignment was selected as a key attribute to evaluate team suitability in this study, based on the alignment between final team allocations and each student's ranked top five project preferences.

2.2.2 Attribute 2: Belbin roles and relevance in educational contexts. Cognitive diversity, as modeled by Belbin's team roles, has been shown to improve team performance and collaboration quality. Aranzabal et al. [2] found that teams balanced in Belbin roles in a project-based course performed better academically and reported more effective communication and coordination. Similarly, Borges, et al. [5] developed a group formation method that integrates Belbin roles with project preferences and skills, resulting in teams that students perceived as more balanced and fair. For this reason, Belbin role diversity was selected as a key attribute in this study, using precollected role assessments to support cognitive complementarity during team formation.

2.3 Metaheuristics

Blum and Roli [4] define metaheuristics as "high-level general-purpose strategies of stochastic search and optimization", meaning they are flexible, problem-agnostic methods that use randomness to efficiently explore complex solution spaces without guaranteeing optimality. Despite not guaranteeing optimality, their article found that these strategies proved useful in complex problems because of the way they sample the solution space with efficiency. Notably, the article found that GA and simulated annealing (SA) showed a higher use rate than other metaheuristic strategies, as did Maqtary, et al. [34]. However, the article emphasizes that this may be because these strategies are generally more used, also outside of the TFP.

2.3.1 Genetic Algorithms. The simple genetic algorithm, SGA, a subset of evolutionary algorithms, EAs, was first conceptualized by J.H. Holland in 1975 [22]. It can be explained as an iterative optimization process inspired by natural selection. The general workflow of a genetic algorithm is illustrated in Figure 3 (Appendix A.1), which shows the key steps in a typical SGA cycle [24]. The algorithm begins by generating an

initial population of encoded solutions (in the context of the TFP, possible team arrangements) called chromosomes, evaluates them through a fitness function, and then creates new generations through repeated cycles of selection, crossover, and mutation. This process is looped until a stopping criterion is met (a satisfactory standard), at which point the best solution found according to the relevant objective function is returned.

2.3.2 Particle Swarm Optimization. The Particle Swarm Optimization (PSO) algorithm was first introduced by Kennedy and Eberhart in 1995 [13, 29]. Inspired by the social behavior of bird flocking and fish schooling, PSO uses the collective intelligence of particles (candidate solutions) to explore the solution space collaboratively.

The general workflow of PSO is illustrated in Figure 4 (Appendix A.2). It begins by initializing a swarm of particles, each representing a candidate solution (like a GA). Each particle has a position (solution representation) and velocity (rate of change), both of which are iteratively updated. The algorithm evaluates the quality of each particle based on a defined fitness function. Each particle keeps track of its personal best position, and the swarm tracks the global best position found among all particles.

During each iteration, particles update their velocities and positions guided by both their individual experience and the swarm's collective experience. This collaborative movement allows the particles to progressively converge towards optimal or near-optimal solutions. The iterative process continues until a predefined stopping criterion is met, such as reaching a satisfactory solution quality or a maximum number of iterations. At this point, the algorithm returns the best solution identified during the optimization process.

2.3.3 Non-dominated Sorting Genetic Algorithm II. The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is an evolutionary algorithm designed to solve multi-objective optimization problems, first introduced by Deb et al. in 2000 [12]. Although it shares the foundational structure of standard GAs (selection, crossover, and mutation), NSGA-II differs in how it handles selection to preserve a diverse and high-quality set of solutions across multiple objectives.

As illustrated in Figure 5 (Appendix A.3), NSGA-II works by combining the parent population P_t with the offspring population Q_t to form an intermediate population R_t. This combined population is then sorted into different non-dominated fronts $F_1, F_2, \ldots$ based on Pareto dominance. The algorithm progressively fills the next generation P_{t1} using these fronts until the population limit is reached. When a front does not fully fit, its members are ranked by a crowding distance measure to ensure that the most diverse solutions (those farthest from others in objective space) are retained, while the rest are rejected. This combination of dominance ranking and preservation of diversity allows NSGA-II to converge toward the true (best) Pareto front while maintaining a well-distributed solution set [12]. That means that unlike single-objective GAs, NSGA-II outputs a series of dominating chromosomes with trade-offs (i.e., an increase in one objective in exchange for a decrease in the other).

2.4 Research Gaps and Subsequent Research Aim

In its discussion of various experiments that implemented (variations of) evolutionary algorithms, Juárez et al.'s 2021 literature review [28] identifies a research gap in the implementation of genetic algorithms aiming to solve the TFP in various contexts, primarily in sports and education. That is, most works have not explored GAs further than basic implementations that use simple crossovers such as Simulated Binary Crossover (SBX), n-point crossover, and uniform crossover. This restricts the problem-specific adaptability, as simple crossovers will often ignore problem-specific constraints, greatly reducing the applicability of the implementations when considering the diversity of TFPs. Therefore, there may be value in investigating and evaluating the implementation of more complex genetic algorithms.

For example, Nand and Sharma [35] performed a comparison of the Firefly Algorithm (FA), PSO (Particle Swarm Optimization) and IWO (Invasive Weed Optimization), introducing a GA implementation later in its study, but without a specification on the implementation of crossover functions of the algorithm itself. Similarly, Chen et al. [7] propose a genetic algorithm integrated with social network analysis to form heterogeneous student groups based on grade, gender, and sociometric status, but it also omits any formal description or categorization of the crossover operator used, instead relying on mutation and selection as primary evolutionary drivers.

An exception to this is the Enhanced Genetic Algorithm (EGA) proposed by Hwang et al. [24], which employs a multipoint crossover operator specifically designed to ensure that the difference in the number of students in any two groups does not exceed one. However, the authors acknowledge a key limitation: one of their defined constraints (i.e., each group must include at least one student who understands each required concept) is not strictly enforced. Instead, solutions violating this constraint are allowed during chromosome generation and are only penalized in the fitness function. As such, this implementation does not fully resolve the constraint satisfaction challenge, leaving room for improvement in their EGA implementation.

Another exception is the deterministic crowding evolutionary algorithm (DCEA) proposed by Yannibelli and Amandi [46], which employs a permutation-based representation and order crossover to form well-balanced teams according to the Belbin role model. Although this approach effectively ensures team size balance and role distribution within groups, it is limited by its narrow focus on a single fixed grouping criterion (i.e., role balance) without supporting multiobjective or constraint-flexible extensions. The algorithm does not allow instructors to specify alternative or additional criteria (e.g., academic performance, prior collaboration, or learning

styles), which are often critical in real-world educational settings. Therefore, despite its robust optimization mechanics, the lack of generalizability and customization means this line of research still leaves open opportunities for developing more adaptable, multi-criteria team formation algorithms.

Lastly, the crossover method used in the genetic algorithm described in Chen at al.'s research [6] is a custom form of team-wise random sampling, not a standard crossover method such as single-point, n-point, uniform, or SBX. However, the applicability of this research is limited, as it is executed on a small (publicly unavailable) sample and it is not tested or benchmarked against other algorithms. This problem is also common to many PSO implementations [14, 19, 21, 35, 48]. Moreover, some of these PSO implementations also lack complex constraint-compliant functions for velocity, with more complex implementations still lacking applicability to TFPs with global constraints.

As shown by these previous examples, valuable attempts have been made to produce complex implementations of the genetic algorithm, but shortcomings present either in the logic of the algorithms themselves, or the connection between the algorithms and the contexts in which they were tested allow space for improvement in the field. One variation of the genetic algorithm that has been used to solve multi-objective extensions of assignment-based kindred TFPs (which the selected case study falls under) and community-based with weighted-skills TFPs, as identified by Juárez et al. [28], is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [12]. Some studies have implemented the NSGA-II for solving various multi-objective TFPs, with various levels of success, but fail to present a meaningful benchmark or comparison with other genetic algorithms, or often other metaheuristics at all, instead illustrating comparisons of the NSGA-II against non-evolutionary algorithms like WSM or TOPSIS, or even randomly or self-selected team arrangements [1, 15, 27, 36, 44]. Therefore, due to its adequacy for TFPs that fall under the assignment-based kindred subclass, and the complexity of its crossover function, this algorithm will be studied in this paper.

2.5 Problem Definition

To understand how the selected case study falls under the subclass of multi-objective assignment-based kindred TFPs, the case study and its constraints must be formally defined.

The Data & Information Course at the University of Twente typically contains about 400 students, where instructors must divide these students into teams of 5-6 students for the course's project component. The restrictions of this TFP are listed below.

Team restrictions:

(1) Each team must be made up of exactly 5-6 students.
(2) Each team must be made up of no more than 4 TCS students, with the remaining students coming from other study programs.
(3) No more than 3 students may have the same native language, except Dutch.

Domain restrictions:

(1) Each student must be in exactly one team.
(2) The maximum number of teams per project is given by the formula below.

$$\text{Upper Limit} = \frac{\text{Ideal number of teams}}{\text{Number of projects}}$$

The mathematical model for this case study can be found in a publication by Barrios-Fleitas et al. (unpublished) [3].

3 IMPLEMENTATION

Firstly, the original data set was cleaned and reduced data sets were derived for early standardized testing, creating a benchmark to guide the early steps of implementation. More details on the process and mathematical computations involved in these steps can be found in Appendix B.

3.1 Genetic Algorithm

The early implementation of a genetic algorithm was inspired by the SGA presented by J.H. Holland (Figure 3). Python was chosen as the primary language for this implementation due to its extensive libraries for data handling and analysis. The pseudocode found in Appendix C.1 was developed for the planning of this implementation.

To aid the implementation, the following utilities were developed:

- a function to evaluate the fitness score of a team arrangement,
- a function to check whether a team arrangement is valid, according to the restrictions of the given case,
- an exhaustive solver that evaluates all possible arrangements, returning the best possible arrangement (feasible for testing in small datasets).

These utilities, as well as all code and relevant files in this paper can be found in a public GitHub repository with documentation [16]. After developing these key utilities, the basic algorithm and its main functions (crossover algorithm, mutation algorithm) were developed. This basic algorithm was first tested on small reduced datasets, where the performance of the algorithm could easily be weighed against the best possible solution computed by the exhaustive solver.

3.1.1 Initial Population Generation. The initial population generation of a GA is integral to both the quality of its solutions and its convergence speed [26]. For example, if the chromosomes generated in the initial population were to lack diversity, even robust crossover and mutation functions may fail to avoid overfitting its later chromosomes to a small subset of the full solution space [40]. The algorithm developed for initial population generation ensures both diversity through random assignment, as well as the validity of the generated team assignments by validating them against the restriction checker.

3.1.2 Fitness Function. The fitness function of a GA will fully determine the characteristics of the chromosomes selected and returned by the algorithm. For this case study, a linear fitness function was implemented that equally valued project preference alignment and cognitive diversity by assigning either of them equal weights. The formula of the fitness function is given below.

$$\text{Arrangement Score} = 0.5 \cdot \text{PS} \;\; 0.5 \cdot \text{CD}$$

where:

- **PS** = Project Satisfaction (normalized, 0 to 1 based on ordinal scale)
- **CD** = Cognitive Diversity (normalized, 0 to 1 using Blau Index)

Project-preference Alignment. In this fitness function, project satisfaction (project-preference alignment) is coded based on a list of preferences submitted by each individual student, where an assignment of the student's first-choice project will yield a score of 1, down to 0.8, 0.6, 0.4, and 0.2 where the student was assigned their fifth preferred choice. If a student is assigned a project that was not on their project preference list at all, this will yield a score of 0. For assessing the adequacy of any team in an arrangement, the mean value of all the team members' project-preference alignment is computed.

Cognitive Diversity. Diversity in Belbin Roles (cognitive diversity) represents a perfectly cognitively diverse team with a score of 1, and a team without cognitive diversity (e.g., a team made up of only six shapers) with a score of 0. The cognitive diversity value is assigned based on the Blau index [10], particularly the adjusted (normalized) formula given below.

$$\text{Blau Index}_{\text{adj}} = \frac{n}{n-1}\left(1 - \sum_{i=1}^{k} p_i^2\right)$$

For assessing an arrangement of teams, the mean fitness score of all the teams in the arrangement is computed, yielding a normalized value from 0 to 1.

3.1.3 Crossover and Mutation. What differentiates the genetic algorithm implemented in this research from a standard genetic algorithm is its crossover function. Unlike many GA implementations in TFP optimization research, the crossover function considers the entire solution space as a parameter, ensuring that both local restrictions of the TFP (e.g., number of nationalities in a team), and global restrictions (e.g., number of teams per project) are respected.

This is achieved by evaluating candidate offspring not only in terms of their individual team compositions but also in how they contribute to the overall feasibility of the full team arrangement. The crossover process begins by randomly selecting a subset of teams from one parent and ensuring no student duplication by filtering additional teams from the second parent. To preserve global feasibility, particularly project assignment balance, the function computes and respects a dynamically derived maximum number of teams per project.

Furthermore, the crossover function proactively fills any remaining student slots by invoking a team-generation routine that explicitly maintains compliance with global project limits and team-level constraints. If a feasible child cannot be produced within a fixed number of crossover attempts, the function defaults to generating a new valid solution from scratch, thereby ensuring continuity of the evolutionary process without stagnation.

The mutation function introduces variation into the population by swapping students between two randomly selected teams. This operation preserves most of the existing structure while potentially improving diversity. After each mutation, the function validates the modified arrangement against all defined constraints (e.g., student uniqueness, project quotas, team feasibility). If a valid solution is found, it is returned immediately. Otherwise, the process retries up to a fixed number of attempts before falling back to generating a new random valid arrangement. This ensures robustness by balancing exploration with feasibility preservation in the search space. With the early implementation of the GA complete, testing was conducted on a dataset of 15 students by setting the algorithm to run until maximum efficiency was reached (the optimal arrangement was output). This allowed for the identification of flaws in the algorithm presented. Details on the early testing and enhancement process can be found in Appendix D.

3.2 Particle Swarm Optimization

The PSO algorithm implemented in this study was based on the structure presented in Figure 4. Like the other 2 algorithms, it was implemented in Python. The implementation was guided by the pseudocode developed and found in Appendix C.2. As PSO is also a mono-objective algorithm, the utility functions developed for the GA were reused for this PSO implementation.

3.2.1 Particle Encoding and Initialization. Each particle in the swarm encodes a full solution (i.e., a complete assignment of students to teams and projects), but unlike the chromosomes generated in the GAs, stores additional information elaborated in the following subsections. Initial arrangements are generated randomly using the same random team-generation algorithm used in the GA's population initialization step that ensures that the initial population is composed of valid particles.

3.2.2 Velocity and Position Updates. In classical PSO, velocity is a vector in continuous space (e.g., a map of the real world). However, in this discrete implementation, velocity is interpreted as a list of swap operations (or other transformations) that will be applied to a particle to generate a new candidate solution, as team arrangements cannot be expressed continuously, but rather through references (discrete). Each particle updates its position using a composite of three components:

- **Inertia**: preserves part of the previous velocity (i.e., previously successful swap operations),
- **Cognitive component**: encourages a return toward the particle's own best-known arrangement,
- **Social component**: pulls the particle toward the best-known global solution.

Inside of the PSO loop, a velocity and position recongifuration attempts to conduct swaps between students in two teams with a series of hyperparameters that control the behavior of the function:

- $\boldsymbol{w}$: determines the probability of the new velocity vector adopting each of the values (swaps) of the previous velocity vector,
- $\boldsymbol{c_1}$: determines the probability of the new velocity vector adopting a swap that brings it closer to the particle's best-scoring arrangement,
- $\boldsymbol{c_2}$: determines the probability of the new velocity vector adopting a swap that brings it closer to the global best-scoring arrangement.

3.2.3 Constraint Compliance. Because transformations (i.e., student swaps) can result in invalid arrangements, each update cycle includes a validation check. If a transformation leads to an infeasible state, the algorithm either:

- Attempts other swaps (within a fixed retry limit), or
- Resets the particle to a new valid random solution.

This mechanism prevents premature convergence and ensures the search remains within the feasible region.

3.3 Non-dominated Sorting Genetic Algorithm II.

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is, unlike standard GAs, a multi-objective EA. This means that it is designed to simultaneously optimize various objectives (i.e., project-preference alignment and cognitive diversity) without requiring a manually weighted fitness function to do so. Like the earlier described GA, a NSGA-II that can tackle the specific TFP of this research was developed using Python. The pseudocode found in C.3, adapted from the original NSGA-II algorithm [12], was developed for the planning of this implementation.

3.3.1 Reusability. Since despite its multi-objective nature the NSGA-II is also a GA, several utils and functions from the GA implementation are reused, namely the restriction checker (as the same restrictions still apply), the fitness functions (as the functions are the same, but computed separately), and the crossover and mutation functions, as they are still a crucial component of NSGA-II, as shown in Figure 5. Moreover, the (reduced) datasets used for developing and testing the developed GA are maintained for the purpose of testability.

3.3.2 Chromosome Representation and Objective Evaluation. As previously mentioned, each chromosome generated by the NSGA-II still represents a complete valid team arrangement, meeting the same restrictions the developed GA meets as detailed in Section 2.5. Moreover, the same fitness functions for project-preference alignment and cognitive diversity explained in section 3.1.2 are used. However, since this algorithm is multi-objective, it ranks the chromosomes it generates via a function referred to as Pareto dominance, a form of comparing any two individuals in its population.

3.3.3 Fast Non-Dominated Sorting. The principle of Pareto dominance, and the basic implementation that this article derives thereof, is that any one individual (chromosome) in a population is said to Pareto dominate another if it is at least as good in all objectives, and strictly better in at least one [11]. In this implementation of fast non-dominated sorting, individuals that are non-dominated are assigned to Pareto fronts, a set of all the solutions that each generation of the algorithm generates. In terms of the code implementation, this means that for any chromosome that is output in each front, there exists no chromosome with the same or higher fitness score for one of the two objectives that also holds a higher fitness score in the second. In the GA loop of the implementation, the first front represents that with the best trade-offs and is prioritized in the selection function.

3.3.4 Crowding Distance and Diversity Preservation. The developed NSGA-II implementation uses crowding distance as a secondary selection criterion to maintain diversity among solutions. This metric approximates how isolated a solution is within the set of non-dominated solutions by comparing its objective values to those of its neighbors. Boundary solutions (those with the best or worst value in either objective) are assigned an infinite distance to ensure they are always preserved. The remaining solutions are given a score based on how far apart they are from others in normalized objective space. By favoring chromosomes with higher crowding distances, the algorithm avoids premature convergence and encourages exploration of a wide range of trade-offs.

3.4 Parameter Setting

The parameter setting process for tunable parameters in the developed GAs and PSO replicated the methodology of other EA implementations for tackling similar NP-hard problems [31, 32]. A detailed overview of the parameter setting process can be found in Appendix F.

4 EXPERIMENTS

4.1 Mono-objective Benchmarking Against Optimal Solutions (Small-scale)

Using the developed exhaustive solver and 3 reduced datasets of 15 students (Appendix B), the performance of both mono-objective algorithms was benchmarked against the optimal solution computed by the solver (the global maximum of the solution space). The performance of these algorithms with respect to these thresholds can be found in Figures 6, 7 and 8 in Appendix G.

4.2 Performance on Large-scale Datasets Against a Baseline

To assess the effectiveness of the evolutionary functions of the algorithms developed, the performance of the mono-objective algorithms was tested against a selective stochastic algorithm that replicates the selection and generation behavior of a Genetic Algorithm on randomly generated but restriction-abiding arrangements. The results of this comparison across differently sized datasets are visualized in Figures 9, 10 and 11 in Appendix G.

4.3 Mono-objective vs. Multi-objective: EGA and PSO vs. NSGA-II

To present a valuable benchmark between mono- and multi-objective EAs, the Pareto fronts generated by the NSGA-II implementation were visualized, highlighting the best single best solution generated by the GA and PSO expressed in terms of both of the objectives present in its objective functions. The average results of 30 runs across differently sized datasets are visualized in Figures 12, 13 and 14 in Appendix G.

4.4 Applicability to Real-World Scenario

To both visually and statistically verify the applicability to traditional team formation processes (i.e. self-selection or random assignment) in real-world applications, figures 1 and 2 illustrate the mean performance of 100 generations/iterations of both mono- and multi-objective algorithms against the self-selected team assignment formed for the original Data & Information project with 278 students.

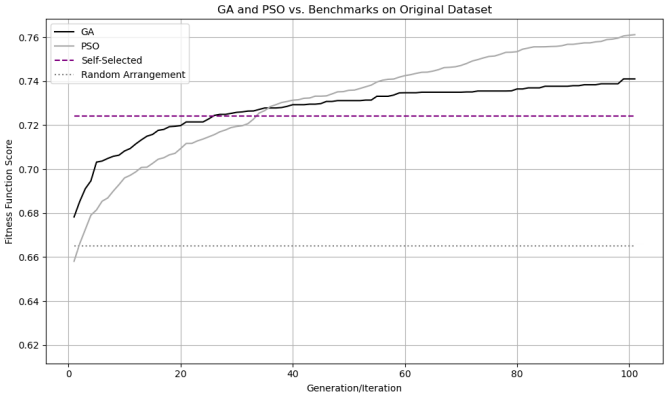

Fig. 1. Performance of GA and PSO against random arrangement and original selection in real-world case.

Moreover, Wilcoxon signed-rank tests [45] were used to statistically compare the final generation fitness scores produced by the GA and PSO algorithms against the fitness score of the self-selected assignment, as well as their performance against each other, as shown in Figures 15 and 16

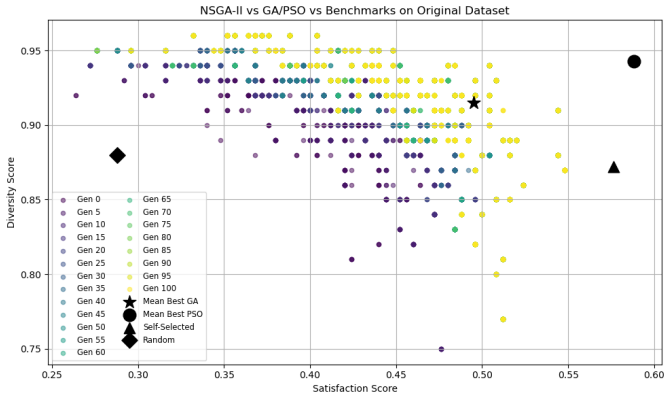

Fig. 2. Performance of the GA, PSO and NSGA-II against random arrangement and original selection in real-world case.

4.5 Efficiency and Computational Cost Analysis

To understand the technical feasibility of the development algorithms and their practicality in classroom contexts, the algorithms developed were assessed for statistics on algorithmic efficiency and time complexity. An overview and elaboration of these can be found in Tables 12, 13 and 14 inside of Appendix G.

5 DISCUSSION

Figures 6 to 8 demonstrate that the PSO and GA developed exhibit stable improvement across runs, further supported by early testing (Appendix D), where most tests converged to the optimum within 200 iterations, often under 100. Figures 9 to 11 evidence the algorithms' success in outperforming a selective stochastic baseline while fully respecting both local (e.g., CS student caps) and global (e.g., project team limits) constraints. This addresses the call by Juarez et al. [28] for problem-specific constraint handling, and builds on Hwang et al.'s work on constraint-focused GAs [24], improving feasibility without relying on penalization. Although the opposite was true in testing datasets, the PSO appears to outperform the GA in all experiments, likely due to its more rigorous discrete velocity function, relying less on stochasticity than the GA does.

Figures 12 to 14 visually compare the mono-objective algorithms with the multi-objective NSGA-II. The Pareto fronts produced by NSGA-II consistently align with or dominate the GA's best result, indicating comparable optimality. Nevertheless, PSO seems to notably dominate its fronts in the same number of generations/iterations. However, unlike the GA and PSO's single weighted output, NSGA-II offers a spectrum of trade-offs, which is particularly valuable in social science contexts. Notably, the diversity offered by NSGA-II does not appear to come at the cost of excessive computational complexity, unlike the visible performance-computation trade-off for PSO seen in 13. This contrasts findings in other evolutionary algorithms [46]. The literature reveals a technical-social

divide: while algorithmic research often overlooks practicality, social science lacks tools for constrained optimization. This work suggests that hybrid, constraint-conscious algorithms may bridge that gap.

Tables 12 , 13 and 14 further reveal that NSGA-II converges faster than the enhanced GA, despite producing a wider solution space. The high standard deviation in the mean of some of its computations per run suggests that this faster conversion may happen due to overly early conversion to local optimums in some runs. Although the observed drop in evaluations for larger datasets complicates conclusions about complexity, it may stem from improved constraint compliance. Empirical runtime trends suggest linear growth relative to dataset size for both genetic algorithms, while PSO's complexity trends toward $O\left(n^2\right)$, though this appears manageable in practice, supporting scalability.

Finally, the experiment design provides rare reproducibility in TFP optimization research, addressing issues of benchmarking and comparability noted in prior work [6]. The open-source tools, (reduced) datasets, and benchmark subsets developed in this study form a foundation for accessible and replicable research, and are already being used to support metaheuristic evaluation in related projects [16].

6 CONCLUSIONS

This research evaluated the effectiveness of two types of Genetic Algorithms, an enhanced mono-objective GA and a multi-objective NSGA-II, and a discretized Particle Swarm Optimization implementation for solving the Team Formation Problem with a focus on balancing member satisfaction and cognitive diversity. Results showed that all approaches are capable of generating high-quality team arrangements that lie close or ahead of the Pareto front, with neither genetic algorithm consistently outperforming the other in terms of objective trade-off. PSO largely outperformed the GA in larger datasets, although its computational rigour may be seen as a limitation. On the other hand, NSGA-II offers the added benefit of flexibility by producing a set of diverse trade-offs, enabling decision-makers to select solutions based on context-specific priorities rather than relying on fixed weightings.

In terms of practical efficiency, both genetic algorithms demonstrated acceptable performance within runtime constraints, with NSGA-II often converging faster despite generating a broader solution space. The practical efficiency of PSO should be sufficient for smaller classroom settings, but could be a drawback in larger-scale applications with a need for more dynamic decision-making. While the empirical time complexity was observed to grow roughly linearly with dataset size, NSGA-II's typical worst-case complexity of $O\left(n^2\right)$ did not appear to present a bottleneck at the tested scales. These findings suggest that multi-objective approaches like NSGA-II may offer a more robust and adaptable alternative for real-world TFP applications, particularly where trade-offs must be navigated or constraints dynamically balanced.

Finally, the demonstrated research and experiments yield answers to the research questions presented:

Research Question 1: As evidenced by the experiments conducted, the Particle Swarm Optimization implementation produced higher-quality teams within practical runtime limits (~10 minutes), but a tradeoff in solution quality and computational rigour is present.

Research Question 2: In the experiments conducted, mono-objective metaheuristics produced higher-quality teams than multi-objective metaheuristics, as PSO particles would often dominate the chromosomes in NSGA-II's Pareto fronts. However, the NSGA-II offers a much wider diversity of solutions in terms of the choice in attribute weighting it offers in educational contexts, and presents itself as a more dynamic alternative in these.

Research Question 3: The findings of **RQ 1** and **2** encourage an adoption of metaheuristics for the formation of educational teams in higher education. From a technical perspective, innovative adaptations like the constraint-compliant GA and discretized PSO using team-level swaps in its velocity function may inspire similar approaches for other TFPs or NP-hard problems. More broadly, this work highlights a promising intersection between algorithmic optimization and educational design, where computational methods support socially grounded decision-making.

7 FUTURE WORKS

Throughout the completion of this article, several directions for future research were identified, namely:

(1) Future operations research could experiment with different functions for initial population generation, crossover, mutation, velocity, and other evolutionary steps of the algorithms developed and conduct more robust testing and benchmarking than this research could offer.
(2) Manipulations could be done to the implementation of this research, such as different function codings or entirely different objective functions could be investigated. A more thorough exploration of literature regarding the influence of both personal and interpersonal factors on team effectiveness could help establish more reliable measurements for the "ideal" team, further bridging the gap between operations research and social science.
(3) The NSGA-II implementation could be further explored. The algorithm does not take into account the global front (as it only selects non-dominated individuals). However, logging the global front of the NSGA-II could help analyze the convergence of the algorithm in controlled datasets, and potentially enhance the algorithm.
(4) Technical studies could explore the effectiveness of (evolutionary) algorithms such as PSO, EGA or NSGA-II when ignoring some restrictions (for example, soft restrictions) as they explore the solution space. A suitable balance between ignoring restrictions and strictly enforcing them which allows for better exploration, and still ensures valid output, may be achievable.

A ALGORITHMS

A.1 Genetic Algorithm

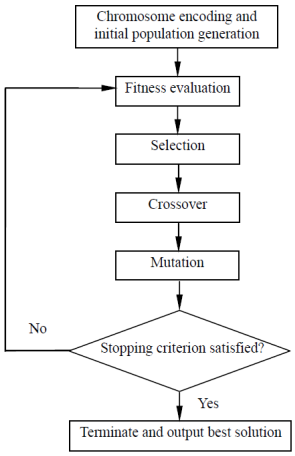

Fig. 3. Flowchart of the Simple Genetic Algorithm (SGA). Reproduced from [24].

A.2 Particle Swarm Optimization

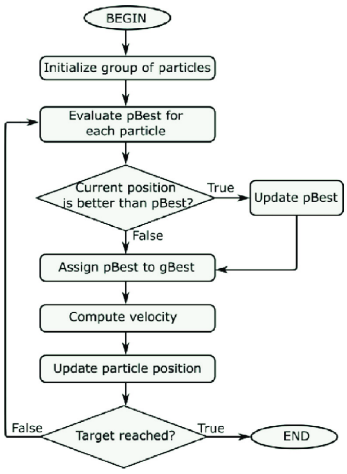

Fig. 4. Flowchart of the Particle Swarm Optimization (PSO) algorithm. Reproduced from [38].

A.3 Non-dominated Sorting Genetic Algorithm II

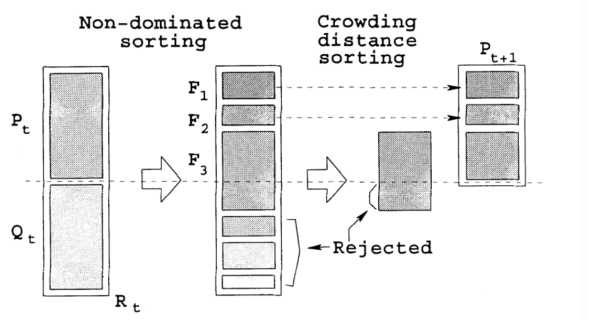

Fig. 5. Illustration of the NSGA-II selection process. The parent population P_t and offspring Q_t are combined and sorted into non-dominated fronts F_1, F_2, F_3. Individuals are selected for the next generation P_{t1} based on rank and crowding distance. Reproduced from [12].

B DATA CLEANING AND PREPARATION

The original data set (406 students) was cleaned by removing students with incomplete or inconsistent data (128 removed, 51 kept with corrections), leaving 278 students. The cleaning process included dealing with missing Belbin roles, project preferences, and dropouts, and ensuring partner-based data consistency. The inclusion-exclusion criteria for the data cleaning process are found in Table 1.

Category	**Description**	#
Removed (128)	Only doing additional assignment	9
	Not doing project	66
	Dropped out	23
	Missing study	3
	Missing study and Belbin role	1
	Missing Belbin role	18
	Missing project preferences	4
	Crossed out in original dataset	4
Kept (51)	Missing project preferences, but could assign partners'	16
	Two different project preference lists (partners' used)	2
	Dropped out, but data complete	1
	Belbin role scores didn't add up, but dominant role was found	27
	Extra digit in student number (corrected)	2
	Fulfilled first and third condition	1
	Fulfilled first and fourth condition	2

Table 1. Summary of Data Cleaning Decisions

Three reduced data sets were then created to simulate realistic team formation scenarios, preserving key demographic distributions (e.g., 67% TCS, 33% BIT, 35% Dutch, 17% Female). These small data sets were used to test the performance

of the algorithms in controlled scenarios. Moreover, a Python util was developed to ensure that future subsets of the full dataset (as seen in section 4) maintained the aforementioned demographic distributions.

The size of the three reduced data sets was determined based on the following mathematical function:

$$\text{Total Arrangements} = \frac{n!}{\left(\prod_{i=1}^{k} s_i!\right) \cdot \left(\prod_{j=1}^{m} m_j!\right)} \times \text{Projects}^{\text{Teams}}$$
$$= \text{Partitions} \times \text{Project Assignments}$$

where:

- n is the number of students,
- s_i is the size of team i,
- $m_j!$ corrects for duplicate team sizes (i.e., m_j is the number of teams of the same size j in the partition).

A population size of 12 students was first arbitrarily selected, which yielded the following.

$$\frac{12!}{\left(6!6!\right) \cdot \left(2!\right)} \times 7^2 = 22,638 \text{ Total Arrangements}$$

The exhaustive solver took 5.0375 seconds to solve this and, from this, a formula for the expected computation time for n total arrangements was developed*:

**All computations were done on the same JupyterLab servers, so minimal variation in computation power can be assumed.*

$$\text{Time} = \text{Total Arrangements} \times \frac{5.0375}{22{,}638} \text{ seconds}$$
$$\approx \text{Total Arrangements} \times 0.000223 \text{ seconds}$$

From this formula, n was maximized while keeping the expected computation time below 3 hours, which yielded $n = 15$:

$$\frac{15!}{\left(5!5!5!\right) \cdot \left(3!\right)} \times 7^3 = 43,261,218 \text{ Total Arrangements}$$

$$\text{Expected Time} = 43{,}261{,}218 \times \frac{5.0375}{22,638} \text{ seconds} \approx 2.67 \text{ hours}$$

C ALGORITHMS PSEUDOCODE

C.1 Genetic Algorithm

```
while (initial population < n):
    create random team arrangement
    if valid arrangement, add to initial population
while (iterations < number of generations):
    give fitness scores to current generation
    select 2 parent arrangements with best score
    while (offspring < number of offspring):
        create child arrangement as crossover of parents
        ensure validity of arrangement
        mutate child arrangement to introduce randomness
        ensure validity of arrangement
return highest-scoring arrangement
```

C.2 Particle Swarm Optimization

```
initialize empty swarm
while (swarm size < n):
    create random team arrangement
    if valid, add to swarm as new particle

for each particle:
    set personal best = current position

set global best = best particle in swarm

while (iterations < number of generations):
    for each particle in swarm:
        generate new velocity using:
            - inertia component (previous velocity)
            - cognitive component (toward personal best)
            - social component (toward global best)
        apply velocity to update position
        if new position is valid:
            evaluate fitness
            update personal best if fitness improved
            update global best if fitness improved
        else:
            try alternative transformations or reset

return global best solution
```

C.3 NSGA-II

```
F = [[]]
for individual p in population P:
    S[p] = []   # Solutions dominated by p
    n[p] = 0    # Number of solutions that dominate p
    for q in P:
        if dominates(p, q):
            S[p].append(q)
        elif dominates(q, p):
            n[p] += 1
    if n[p] == 0:
        p.rank = 1
        F[0].append(p)
i = 0
while F[i] != []:
    Q = []  # Next front
    for p in F[i]:
        for q in S[p]:
            n[q] -= 1
            if n[q] == 0:
                q.rank = i + 2
                Q.append(q)
    i += 1
    F.append(Q)
return F
```

D EARLY TESTING AND ENHANCEMENTS

Although early results for the GA implementation were promising, showing that some test runs output the optimal solution in under 30 generations (checking less than 1000 of over 43

million possible arrangements), some runs would reach a peak efficiency early on (e.g., reach a score with 95% efficiency within the first 30 generations), with said score not increasing for thousands of generations thereafter.

This suggested that the algorithm heavily relied on the initial population generation yielding two strong parent chromosomes, meaning if all 10 randomly generated arrangements in the initial population were far from optimal, the algorithm would constantly create offspring based on two parent chromosomes that would never yield the optimal solution.

Two solution approaches for this problem were identified:

(1) **Increasing the size of the initial population:** Increasing the size of the (randomly generated) initial population allows the GA to cover more variation before the crossover-mutation loop. This avoids executing a loop on two initial parent chromosomes that will never yield the optimal solution.
(2) **Introducing randomness in the algorithm:** While mutations in the SGA introduce some randomness, it is very limited (i.e., one small swap) and fails to substantially change the structure of a parent arrangement. Hence, it fails to uncover largely different arrangements that could yield the optimal solution. Therefore, systematically introducing complete randomness (such as in the initial population) every n generations where the score does not increase would introduce large randomness at times when the much-needed variation in the generation of arrangements stagnates.

The first approach improved performance (test runs of the algorithm would more often reach the optimal solution), but would still sometimes fail to output the optimal solution when the initial population generation did not yield high-scoring chromosomes. Moreover, this approach would not be scalable to larger datasets, as substantially more combinations become possible, meaning that finding a strong candidate in the initial population generation eventually becomes an NP-hard problem in the same way that the TFP itself becomes an NP-hard problem as the number of students increases.

Therefore, introducing randomness in the algorithm ensures that even if the initial population is suboptimal, the search space can be periodically reexplored. This systematic reintroduction of randomness provides new candidate solutions that may not be accessible through traditional crossover and mutation alone. In this way, the algorithm becomes less dependent on the initial population and more robust to local optima, improving its ability to consistently find the global optimum within a feasible runtime (i.e., sufficiently fast for classroom applications). These adjustments significantly improve the real-world applicability of the genetic algorithm to the team formation problem, especially in scenarios with large, highly constrained datasets.

After 100 test runs for various efficiency thresholds in all three reduced data sets, the standard GA yielded the results in Table 2, evidencing the performance of the GA. Individual performance data for each separate reduced data set can be found in Appendix E.

Efficiency	Generations	Arrangements Computed
90%	9.20	193.91
95%	46.68	840.80
100%	146.41	2638.41

Table 2. GA mean performance on reduced datasets at various efficiency thresholds

Although this implementation showed promising results across these reduced datasets, later testing on large datasets showed that the GA would sometimes violate general constraints (i.e., number of teams per project). Therefore, the crossover function was adjusted to ensure that any random arrangement generation (as a fallback for missing students due to overlap in the crossover) would respect not the global restrictions of that subset specifically, but the global restrictions of the full chromosome.

This enhanced implementation not only solved the restriction violation problem but yielded significantly better chromosomes in the full dataset. However, the restriction-violating GA yielded better chromosomes and would always converge to the optimal solution in reduced datasets, unlike the corrected GA. This suggests that some evolutionary (or other) algorithms may be more efficient in searching their solution spaces when ignoring global restrictions, potentially being more effective in finding optimal solutions.

The early testing of the GA developed (the first algorithm implemented in this experiment) helped uncover implementation challenges, for which solutions suitable for all three algorithms were found. Hence, early testing on the paper's NSGA-II and PSO implementations was not as rigorous.

E RAW TEST DATA

Efficiency	Generations	Arrangements Computed
90%	9.20	188.85
95%	78.70	1392.73
100%	97.46	1727.28

Table 3. GA mean performance on reduced dataset 1 at various efficiency thresholds

Efficiency	Generations	Arrangements Computed
90%	13.23	283.07
95%	16.79	330.26
100%	124.32	2202.05

Table 4. GA mean performance on reduced dataset 2 at various efficiency thresholds

Efficiency	Generations	Arrangements Computed
90%	5.16	109.80
95%	44.54	799.40
100%	217.44	3985.91

Table 5. GA mean performance on reduced dataset 3 at various efficiency thresholds

F PARAMETER SETTING

Following the methodology of previous work in the field [31, 32], datasets composed of data from 50, 100, and 200 students respectively were derived through random selection from the full dataset of 278 students. Tuning parameters were determined for each algorithm, and the derived configurations were run on all datasets.

F.1 Genetic Algorithm

The tunable parameter (which could affect the algorithm performance) for the developed GA is the parent reset threshold P_r, in other words, the stagnation threshold at which parents are completely reset to explore a different subset of the solution space. Table 6 shows reasonable parameter values to assess during parameter setting.

Parameter	Value
P_r	$\in \{5, 10, 15, 20\}$

Table 6. Parameter tuning range used for GA experiments.

For the GA developed and the parameter values mentioned, the mean values of 10 runs of 1000 generations shown in Table 7 were found.

Size	5	10	15	20
50	0.7186	0.7235	0.7204	0.7187
100	0.7042	0.7038	0.6963	0.7015
200	0.6777	0.6787	0.6783	0.6751

Table 7. Average scores for different values of P_r across dataset sizes.

A Friedman test [9] was performed using the scipy.stats library to evaluate the effect of different values P_r in different datasets of different sizes. The test resulted in a chi-square value of 3.8000 and a p-value of 0.2839, indicating that the performance differences were not statistically significant at a standard $\alpha < 0.05$. Consequently, P_r was set to 10 for the computation of the final results based on the best average performance, but it can be assumed that the parameter does not significantly affect the performance of the GA following the Friedman test.

w	c_1	c_2		
		0.25	0.5	0.75
0.25	0.25	0.8089	0.8176	0.8016
0.25	0.5	0.7994	0.8014	0.7960
0.25	0.75	0.7833	0.7769	0.7891
0.5	0.25	0.8090	0.8164	0.8138
0.5	0.5	0.8081	0.7959	0.8073
0.5	0.75	0.7743	0.7675	0.7718
0.75	0.25	0.7997	0.8139	0.8126
0.75	0.5	0.7968	0.8000	0.800
0.75	0.75	0.7858	0.7724	0.7805

Table 9. Average PSO fitness scores on small (50 person) dataset for each combination of w, c_1, c_2.

w	c_1	c_2		
		0.25	0.5	0.75
0.25	0.25	0.7795	0.7769	0.7690
0.25	0.5	0.7691	0.7659	0.7661
0.25	0.75	0.7547	0.7468	0.7450
0.5	0.25	0.7748	0.7782	0.7834
0.5	0.5	0.7670	0.7704	0.7503
0.5	0.75	0.7550	0.7396	0.7371
0.75	0.25	0.7764	0.7787	0.7774
0.75	0.5	0.7668	0.7663	0.7669
0.75	0.75	0.7381	0.7584	0.7542

Table 10. Average PSO fitness scores on medium (100 person) dataset for each combination of w, c_1, c_2.

F.2 Particle Swarm Optimization

The tunable parameters (which could affect the algorithm performance) for the developed PSO are those mentioned in Section 3.2.2. These hyperparameters $\boldsymbol{w}$, $\boldsymbol{c_1}$, and $\boldsymbol{c_2}$ represent stochastic probabilities of adopting swaps from different particles in the swarm, ranging from 0 to 1. Table 8 shows reasonable parameter values to assess during parameter setting.

Parameter	Value
w	$\in \{0.25, 0.5, 0.75\}$
c_1	$\in \{0.25, 0.5, 0.75\}$
c_2	$\in \{0.25, 0.5, 0.75\}$

Table 8. Parameter tuning range used for PSO experiments.

For the developed PSO and the parameter values mentioned, the mean values of 10 runs of 250 iterations shown in Tables 9, 10 and 11 were found for the respective dataset sizes.

w	c_1	c_2		
		0.25	0.5	0.75
0.25	0.25	0.7359	0.7369	0.7335
0.25	0.5	0.7281	0.7315	0.7266
0.25	0.75	0.7109	0.7117	0.7134
0.5	0.25	0.7364	0.7388	0.7399
0.5	0.5	0.7307	0.7323	0.7292
0.5	0.75	0.7094	0.7095	0.7075
0.75	0.25	0.7377	0.7370	0.7341
0.75	0.5	0.7301	0.7251	0.7204
0.75	0.75	0.7117	0.7094	0.7121

Table 11. Average PSO fitness scores on large (200 person) dataset for each combination of w, c_1, c_2.

As for the GA developed, a Friedman test was performed using the scipy.stats library to evaluate the effect of different value tuples w, c_1, c_2 across datasets of different sizes. The test yielded a Friedman statistic of 54.0000 with a corresponding p-value of 0.0000, indicating statistically significant differences in performance between at least some of the parameter configurations at a standard significance threshold $\alpha < 0.05$.

The test was applied over the average fitness scores across three datasets (small, medium, and large), treating each combination of w, c_1, c_2 as a separate treatment. The ranking of these combinations, as shown in the resulting mean rank values, enabled the identification of the most effective parameter settings. The combination $w = 0.5, c_1 = 0.25, c_2 = 0.75$ achieved the best overall rank, closely followed by $w = 0.5, c_1 = 0.25, c_2 = 0.5$ and $w = 0.75, c_1 = 0.25, c_2 = 0.5$, suggesting that these configurations lead to superior fitness performance across datasets.

Consequently, $w = 0.5, c_1 = 0.25, c_2 = 0.75$ was selected as the final parameter setting for the computation of the main PSO results, based on both statistical evidence and consistently high average scores.

F.3 NSGA-II

Unlike the enhanced GA developed earlier, the NSGA-II implementation used in this project does not incorporate explicitly tunable parameters. This is due to the design of the implementation, which follows a minimalistic and standard approach based on the original formulation [12]. Parameters such as population size and number of generations are treated as fixed computational constraints rather than elements to be tuned for performance, as they directly scale the total number of evaluations (i.e., increasing the population size will give more exploration diversity and hence find better solutions, but at the cost of computation time). Furthermore, advanced features such as adaptive mutation or restart strategies were not implemented, leaving no algorithm-specific parameters that influence performance in a tunable manner.

G RESULTS

G.1 Mono-objective Performance Against Optimal Solutions

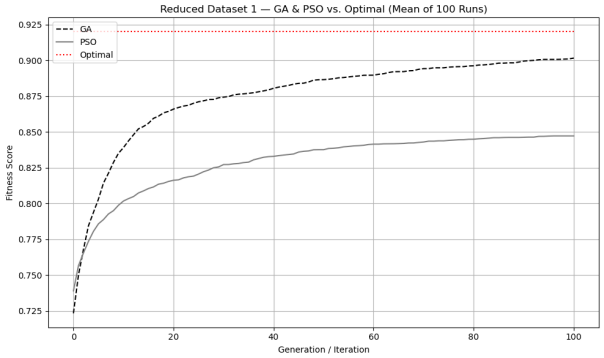

Fig. 6. Performance of GA and PSO in reduced dataset 1 benchmarked against optimum.

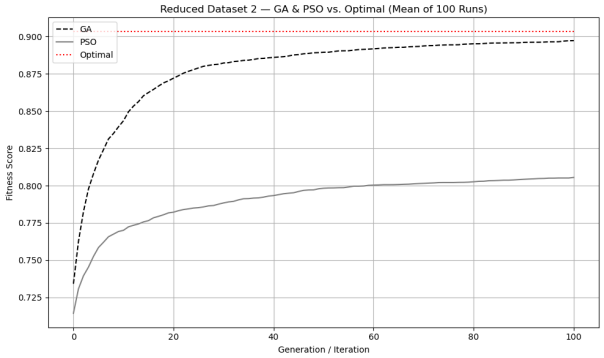

Fig. 7. Performance of GA and PSO in reduced dataset 2 benchmarked against optimum.

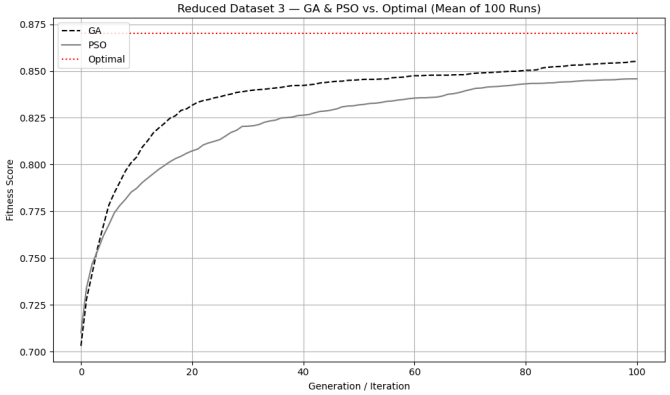

Fig. 8. Performance of GA and PSO in reduced dataset 3 benchmarked against optimum.

G.2 Performance on Large-scale Datasets Against a Baseline

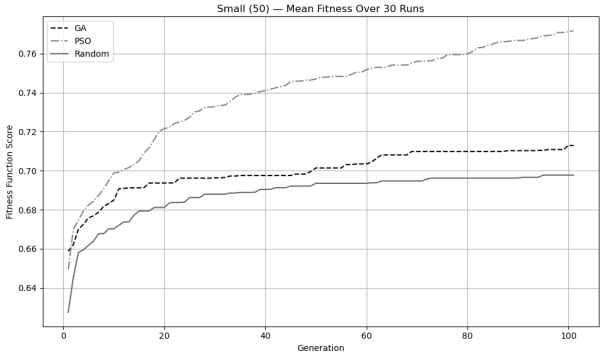

Fig. 9. Performance of GA and PSO against random baseline in a small (50 person) dataset.

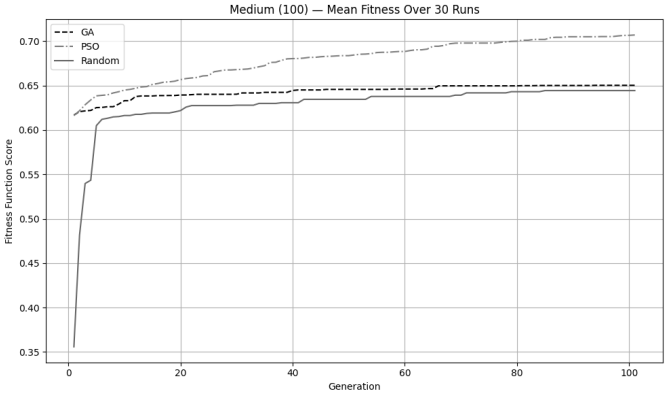

Fig. 10. Performance of GA and PSO against random baseline in a medium (100 person) dataset.

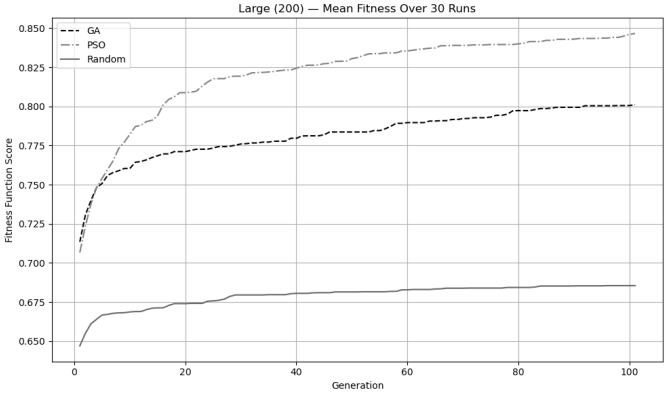

Fig. 11. Performance of GA and PSO against random baseline in a large (200 person) dataset.

G.3 Mono-objective vs. Multi-objective: GA and PSO vs. NSGA-II

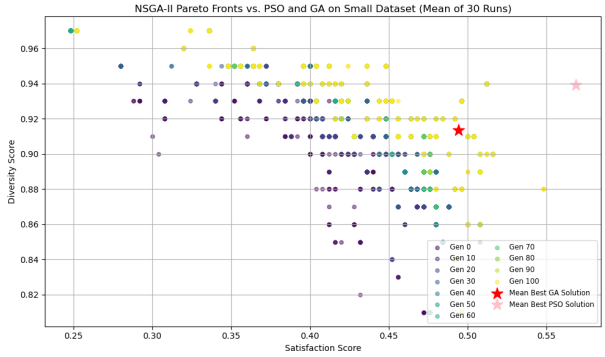

Fig. 12. Visual comparison of NSGA-II Pareto fronts and PSO and GA fitness.

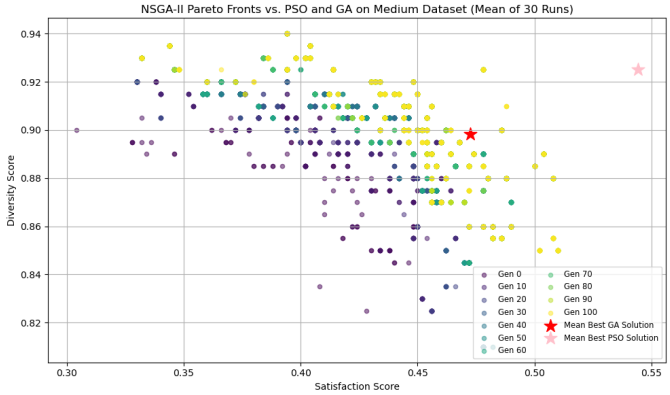

Fig. 13. Visual comparison of NSGA-II Pareto fronts and PSO and GA fitness.

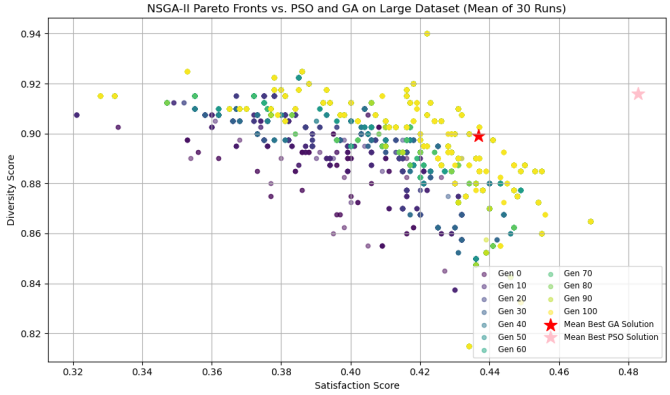

Fig. 14. Visual comparison of NSGA-II Pareto fronts and PSO and GA fitness.

G.4 Wilcoxon Signed-Rank Tests

Comparison	Statistic	*p*-value
GA vs Self-Selected	14.0	6.96×10^{-6}
PSO vs Self-Selected	1.0	1.92×10^{-6}

Fig. 15. Wilcoxon signed-rank test results comparing the final fitness of GA and PSO to the self-selected student arrangement.

For each run of Figure 1, the best score at generation/iteration 100 was extracted, and then compared to the benchmark value (the self-selected score) using a paired non-parametric test. The low statistics and p-values indicate that both the GA and PSO significantly outperform the self-selection threshold. Moreover, the following Wilcoxon signed-rank test compares the performance of the GA to the of the PSO in a dataset of 200 students, showing that the PSO significantly outperforms the GA implementation.

Comparison	Statistic	*p*-value
PSO vs GA	0.0	1.73×10^{-6}
PSO vs Random	0.0	1.73×10^{-6}
GA vs Random	0.0	1.73×10^{-6}

Fig. 16. Wilcoxon signed-rank test results comparing final fitness of PSO, GA, and Random on the 200-person dataset.

G.5 Efficiency and Time Complexity

Based on an average of 30 runs, each of 100 generations/iterations of the developed GA, PSO and NSGA-II, the values seen in tables 12, 13 and 14 were computed.

Size	GA	PSO	NSGA-II
50	81.91	45.63	54.35
100	88.32	194.00	70.90
200	118.61	752.22	96.13

Table 12. Mean runtime per algorithm (30 runs, seconds).

While the GAs roughly follow a trend of linear growth in computation time ($O\left(n\right)$), the time complexity of the PSO can be estimated with the following:

By assuming that the runtime can be modelled as

$$Tn = c \cdot n^k$$

where

- Tn is the total runtime for a dataset of size n,
- c is a constant coefficient,
- k is the degree that defines the growth rate,

we analyze empirical runtime values using ratios:

$$\frac{T100}{T50} = \left(\frac{100}{50}\right)^k = 2^k$$
$$4.25 = 2^k$$
$$k \approx \log_2 4.25 \approx 2.09$$

$$\frac{T100}{T50} = \left(\frac{200}{100}\right)^k = 2^k$$
$$3.88 = 2^k$$
$$k \approx \log_2 3.88 \approx 1.95$$

In both cases, the estimated exponent $k \approx 2$ suggests that the runtime of PSO grows approximately at ($O\left(n^2\right)$).

Size	GA	PSO	NSGA-II
50	25,684	134,232	23,640
100	32,937	285,623	33,167
200	17,410	565,463	15,821

Table 13. Mean evaluations per algorithm (30 runs).

Size	GA	PSO	NSGA-II
50	598	2,193	496
100	500	11,300	705
200	343	22,116	5,241

Table 14. Standard deviation of evaluations per algorithm (30 runs).

REFERENCES

[1] Faez Ahmed, Kalyanmoy Deb, and Abhilash Jindal. 2013. Multi-objective optimization and decision making approaches to cricket team selection. *Applied soft computing* 13, 1 (2013), 402–414. https://doi.org/10.1016/j.asoc.2012.07.031

[2] A. Aranzabal, E. Epelde, and M. Artetxe. 2021. Team Formation on the Basis of Belbin's Roles to Enhance Students' Performance in Project Based Learning. *Education for Chemical Engineers* 37, September (2021), 111–200. https://doi.org/10.1016/j.ece.2021.09.001

[3] Yeray Barrios-Fleitas, Eduardo Lalla, and Arend Rensink. 2025. Empirical Validation of a Metaheuristic Taxonomy for the Team Formation Problem in Educational Contexts: A Multi-Paradigm Comparative Study. (2025). Manuscript in preparation.

[4] Christian Blum and Andrea Roli. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)* 35, 3 (2003), 268–308. https://doi.org/10.1145/937503.937505

[5] José Borges, Teresa Galvão Dias, and João Falcão e Cunha. 2009. A new group-formation method for student projects. *European Journal of Engineering Education* 34, 6 (2009), 573–585. https://doi.org/10.1080/03043790903202967

[6] Rong-Chang Chen, Chien-Ting Chen, Ming-Hung Chen, and Shu-Ping Suen. 2011. A Genetic Algorithm for Team Composition Optimization in a Physical Education Program. In *International Conference on Information and Business Intelligence*. Springer, Springer, Chongqing, China, 51–56. https://doi.org/10.1007/978-3-642-29087-9_8

[7] Rong-Chang Chen, Shih-Ying Chen, Jyun-You Fan, and Yen-Ting Chen. 2012. Grouping partners for cooperative learning using genetic algorithm and social network analysis. *Procedia Engineering* 29 (2012), 3888–3893. https://doi.org/10.1016/j.proeng.2012.01.589

[8] Simon Cullen and Daniel M. Oppenheimer. 2024. Choosing to learn: The importance of student autonomy in higher education. *Science Advances* 10 (2024). https://doi.org/10.1126/sciadv.ado6759

[9] Wayne W. Daniel. 1990. *Applied Nonparametric Statistics* (2nd ed.). PWS-Kent Publishing Company, Boston, MA.

[10] Jeremy Dawson. 2012. *Measurement of work group diversity*. Ph. D. Dissertation. Aston University. https://publications.aston.ac.uk/id/eprint/16437/

[11] Kalyanmoy Deb. 2011. *Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction*. Springer London, London, 3–34. https://doi.org/10.1007/978-0-85729-652-8_1

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[13] R. Eberhart and J. Kennedy. 1995. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. 39–43. https://doi.org/10.1109/MHS.1995.494215

[14] Walaa H. El-Ashmawi, Ahmed F. Ali, and Mohamed A. Tawhid. 2019. An improved particle swarm optimization with a new swap operator for team formation problem. *Journal of Industrial Engineering International* 15, 1 (March 2019), 53–71. https://doi.org/10.1007/s40092-018-0282-6

[15] Bo Feng, Zhong-Zhong Jiang, Zhi-Ping Fan, and Na Fu. 2010. A method for member selection of cross-functional teams using the individual and collaborative performances. *European Journal of Operational Research* 203, 3 (2010), 652–661. https://doi.org/10.1016/j.ejor.2009.08.017

[16] Daniel Frutos-Rodríguez. 2025. Algorithm Implementations for Team Formation Problem. https://github.com/danielfrgz/tfp-algorithms/.

[17] J Richard Hackman. 1998. Why teams don't work. In *Theory and research on small groups*. Springer, Boston, MA, 245–267.

[18] J Richard Hackman. 2002. *Leading teams: Setting the stage for great performances*. Harvard Business Press, Boston, MA.

[19] Bikhtiyar Hasan and Amine Boufaied. 2023. Educational Group Formation Problem Resolution Based on an Improved Swarm Particle Optimization Using Fuzzy Knowledge. In *2023 6th International Conference on Information and Computer Technologies (ICICT)*. IEEE, Honolulu, HI, USA, 13–21. https://doi.org/10.1109/ICICT58900.2023.00010

[20] Myeong hee Shin. 2018. Effects of Project-based Learning on Students' Motivation and Self-efficacy. *English Teaching* 73, 1 (2018), 95–114. https://doi.org/10.15858/engtea.73.1.201803.95

[21] Tsu-Feng Ho, Shyong Jian Shyu, Feng-Hsu Wang, and Chris Tian-Jen Li. 2009. Composing High-Heterogeneous and High-Interaction Groups in Collaborative Learning with Particle Swarm Optimization. In *2009 WRI World Congress on Computer Science and Information Engineering*, Vol. 4. 607–611. https://doi.org/10.1109/CSIE.2009.876

[22] John H. Holland. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge, MA. https://doi.org/10.7551/mitpress/1090.001.0001

[23] Lu Hong and Scott E Page. 2004. Groups of diverse problem solvers can outperform groups of high-ability problem solvers. *Proceedings of the National Academy of Sciences* 101, 46 (2004), 16385–16389. https://doi.org/10.1073/pnas.0403723101

[24] Gwo-Jen Hwang, Peng-Yeng Yin, Chi-Wei Hwang, and Chin-Chung Tsai. 2008. An enhanced genetic approach to composing cooperative learning groups for multiple grouping criteria. *Journal of Educational Technology & Society* 11, 1 (2008), 148–167.

[25] Daniel R. Ilgen, John R. Hollenbeck, Michael Johnson, and Dustin Jundt. 2005. Teams in organizations: From input-process-output models to IMOI models. *Annual Review of Psychology* 56 (2005), 517–543. https://doi.org/10.1146/annurev.psych.56.091103.070250

[26] D. Jakobović, Ivan Vlasic, and Marko Durasevic. 2019. Improving genetic algorithm performance by population initialisation with dispatching rules. *Comput. Ind. Eng.* 137 (2019). https://doi.org/10.1016/j.cie.2019.106030

[27] Julio Juárez and Carlos A Brizuela. 2018. A multi-objective formulation of the team formation problem in social networks: preliminary results. In *Proceedings of the genetic and evolutionary computation conference*. 261–268. https://doi.org/10.1145/3205455.3205634

[28] Julio Juárez, Cipriano Santos, and Carlos A Brizuela. 2021. A comprehensive review and a taxonomy proposal of team formation problems. *ACM Computing Surveys (CSUR)* 54, 7 (2021), 1–33. https://doi.org/10.1145/3465399

[29] J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, Vol. 4. 1942–1948. https://doi.org/10.1109/ICNN.1995.488968

[30] Steve WJ Kozlowski and Daniel R Ilgen. 2006. Enhancing the effectiveness of work groups and teams. *Psychological science in the public interest* 7, 3 (2006), 77–124. https://doi.org/10.1111/j.1529-1006.2006.00030.x

[31] Eduardo Lalla-Ruiz, Christopher Expósito-Izquierdo, Belén Melián-Batista, and J Marcos Moreno-Vega. 2016. A hybrid biased random key genetic algorithm for the quadratic assignment problem. *Inform. Process. Lett.* 116, 8 (2016), 513–520.

[32] Eduardo Lalla-Ruiz, José Luis González-Velarde, Belén Melián-Batista, and J Marcos Moreno-Vega. 2014. Biased random key genetic algorithm for the tactical berth allocation problem. *Applied Soft Computing* 22 (2014), 60–76.

[33] Theodoros Lappas, Kun Liu, and Evimaria Terzi. 2009. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, New York, NY, USA, 467–476. https://doi.org/10.1145/1557019.1557074

[34] Naseebah Maqtary, Abdulqader Mohsen, and Kamal Bechkoum. 2019. Group formation techniques in computer-supported collaborative learning: A systematic literature review. *Technology, Knowledge and Learning* 24 (2019), 169–190. https://doi.org/10.1007/s10758-017-9332-1

[35] Ravneil Nand and Anuraganand Sharma. 2019. Meta-heuristic approaches to tackle skill based group allocation of students in project based learning courses. In *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, IEEE, Wellington, New Zealand, 1782–1789. https://doi.org/10.1109/CEC.2019.8789987

[36] M Niveditha, G Swetha, U Poornima, and Radha Senthilkumar. 2017. A genetic approach for tri-objective optimization in team formation. In *2016 Eighth International Conference on Advanced Computing (ICoAC)*. IEEE, 123–130. https://doi.org/10.1109/ICoAC.2017.7951757

[37] Barbara Oakley, Richard M Felder, Rebecca Brent, and Imad Elhajj. 2004. Turning student groups into effective teams. *Journal of student centered learning* 2, 1 (2004), 9–34.

[38] R S M Lakshmi Patibandla and Veeranjaneyulu N. 2021. *Clustering Algorithms: An Exploratory Review*. IntechOpen. https://doi.org/10.5772/intechopen.100376

[39] Irene Poort, Ellen Jansen, and Adriaan Hofman. 2020. Does the group matter? Effects of trust, cultural diversity, and group formation on engagement in group work in higher education. *Higher Education Research & Development* 41 (2020), 511–526. https://doi.org/10.1080/07294360.2020.1839024

[40] Pablo Ramos-Criado, D. B. Rolanía, Daniel Manrique, and E. Serrano. 2020. Grammatically uniform population initialization for grammar-guided genetic programming. *Soft Computing* 24 (2020), 11265 – 11282. https://doi.org/10.1007/s00500-020-05061-w

[41] Diana Sachmpazidi, Alice Olmstead, Amreen Nasim Thompson, Charles Henderson, and Andrea Beach. 2021. Team-based instructional change in undergraduate STEM: Characterizing effective faculty collaboration. *International Journal of STEM Education* 8 (2021), 1–23. https://doi.org/10.1186/s40594-021-00273-4

[42] Victor Sanchez-Anguix, Juan M Alberola, Elena Del Val, Alberto Palomares, and Maria Dolores Teruel. 2023. Comparing computational algorithms for team formation in the classroom: a classroom experience. *Applied Intelligence* 53, 20 (2023), 23883–23904. https://doi.org/10.1007/s10489-023-04748-5

[43] Marina Sousa and Eunice Fontão. 2025. Team-based learning—An approach to enhance collaboration and academic success in engineering education: A comprehensive study. *Forum for Education Studies* 3 (2025), 2239–2239. https://doi.org/10.59400/fes2239

[44] Shanu Verma, Millie Pant, and Vaclav Snasel. 2021. A Comprehensive Review on NSGA-II for Multi-Objective Combinatorial Optimization Problems. *IEEE Access* 9 (2021), 57757–57791. https://doi.org/10.1109/ACCESS.2021.3070634
[45] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968
[46] Virginia Yannibelli and Analía Amandi. 2012. A deterministic crowding evolutionary algorithm to form learning teams in a collaborative learning context. *Expert Systems with Applications* 39, 10 (2012), 8584–8592. https://doi.org/10.1016/j.eswa.2012.01.195
[47] Haining Zhang and Chia Ching Tu. 2023. An Empirical Study on the Effect of Achievement Motivation and Self-Efficacy on Social Loafing. *Journal of Logistics, Informatics and Service Science* (2023). https://doi.org/10.33168/jliss.2023.0419
[48] Zhilin Zheng and Niels Pinkwart. 2014. A Discrete Particle Swarm Optimization Approach to Compose Heterogeneous Learning Groups. In *2014 IEEE 14th International Conference on Advanced Learning Technologies*. 49–51. https://doi.org/10.1109/ICALT.2014.24