FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatik

Multi-Objective Optimization for Team Composition in Project Courses

Teodora Dobos

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatik

Multi-Objective Optimization for Team Composition in Project Courses

Mehrdimensionale Optimierung für die Teamzusammensetzung in Projektkursen

Author:	Teodora Dobos
Supervisor:	Prof. Dr. Bernd Brügge
Advisor:	Dr. Dora Dzvonyar
Date:	15.07.2020

I assure the single handed composition of this bachelor thesis only supported by declared resources,

Munich, 15.07.2020 Teodora Dobos

Acknowledgements

I am grateful to my advisor, Dr. Dora Dzvonyar, for giving me the opportunity to work on an interesting research topic, for her constant guidance and for the friendly, professional relationship that we have.

Zusammenfassung

Die Zusammensetzung ausgewogener Teams von Studenten in einem Multiprojektkurs, so dass sie bestimmte praktische und fachliche Anforderungen erfüllen, kann als ein Optimierungsproblem mit Nebenbedingungen modelliert werden. Für die Zusammenstellung von Teams im iPraktikum wurde eine einkriterielle Methode angewandt, mit dem Ziel, jedem Studenten eines seiner vorrangigen Projekte zuzuweisen und gleichzeitig gleichwertige Teams in Bezug auf das Kompetenz- und Erfahrungsniveau der Mitglieder zu bilden. Es hat sich jedoch als herausfordernd erwiesen, Nebenbedingungen z.B. für den Fähigkeiten Ausgleich in jedem Team festzulegen, und die Frage, ob es vernünftig ist, das Erfahrungsniveau eines Teams allein durch die Zählung der Studenten mit einem bestimmten Fähigkeitsniveau zu messen, ist umstritten. Daher ist die Modellierung und Lösung der Team-Zuweisung als ein mehrkriterielles Optimierungsproblem eine Alternative, die in Betracht gezogen werden sollte.
Das Ziel dieser Arbeit ist es, mehrkriterielle Optimierungsmethoden für die Zusammenstellung von Teams in Multiprojektkursen zu evaluieren. Um dieses Ziel zu erreichen, haben wir drei mehrkriterielle heuristische Algorithmen ausgewählt und zur Lösung des Teamzuordnungsproblems in einer konkreten vergangenen Instanz des iPraktikums angewandt. Wir evaluierten die Anwendbarkeit, Nutzbarkeit und Effizienz dieser Algorithmen in einem kontrollierten Experiment, in dem wir Leistungs- und Zuweisungsqualitätskriterien berücksichtigten. Für jeden Algorithmus identifizierten wir optimale Parameterkonfigurationen. Wir verglichen die mit den mehrkriteriellen Methoden erzeugten Lösungen mit der Lösung, die mit dem in TEASE implementierten einkriteriellen Methode berechnet wurde. Die Ergebnisse zeigen, dass unter den analysierten mehrkriteriellen Methoden der MOPSO-Algorithmus die beste Zuweisungslösung in Bezug auf die von uns definierten Metriken berechnet. Außerdem ist ein mehrkriterieller Ansatz im Vergleich zu einer einkriteriellen Methode für konkrete Szenarien des Teamzuordnungsproblems (z.B. wenn das Erfahrungsniveau des Studenten mit Fuzzy-Logik modelliert wird) besser, aber er berechnet Zuordnungslösungen, die niedrigere Prioritätszielwerte haben.

Abstract

Creating balanced teams of students in a multi-project course such that they satisfy specific practical and skill requirements can be modelled as a constrained optimization problem. A single-objective method has been applied to compose teams in the iPraktikum, with the purpose of assigning each student one of their top-priority projects, while also creating equal teams with respect to the skill and experience levels of the members. However, setting constraints for e.g. the balance of skills in each team has shown to be challenging and the question whether it is reasonable to measure the experience level of a team solely by counting the students with a specific skill level is debatable. Thus, modeling and solving the team assignment as a multi-objective optimization problem is an alternative which should be considered.
The goal of this thesis is to evaluate multi-objective optimization methods for composing teams in multi-project courses. In order to achieve this goal, we selected three multi-objective heuristic algorithms and applied them to solve the team assignment problem in a concrete past instance of the iPraktikum. We evaluated the applicability, usability and efficiency of these algorithms in a controlled experiment in which we considered performance and assignment quality criteria. For each algorithm, we identified optimal parameter configurations. We compared the solutions generated by the multi-objective methods with the solution computed using the single-objective approach implemented in TEASE. The results show that, among the analyzed multi-objective methods, the MOPSO algorithm calculates the best assignment solution with respect to the metrics that we defined. In addition, compared to a single-objective technique, a multi-objective approach is superior for concrete scenarios of the team assignment problem (e.g. if the student experience level is modelled using fuzzy logic), but it computes assignment solutions that have lower priority objective values.

Contents

Chapter 1

Introduction

Designing effective student teams in the context of a project course such as iPraktikum is a challenging task. Instructors have to identify the technical skill level of each student, which, together with related personal and social information forms a complete participant profile. In addition, a systematic approach is needed for assigning each student to the project that appears to be the most suitable for them. Given that a set of possibly conflicting criteria must be considered in this process (practical constraints, equal skill distribution, motivational factors and personal criteria [9]), a manual approach is extremely time-consuming and has been shown to produce a suboptimal assignment solution with respect to some predefined goals [8].
Numerous multi-project courses, such as the iPraktikum, have been developed at the Chair for Applied Software Engineering at the Technical University of Munich. These courses implement a similar team assignment process [10], whose steps follow the Project Preparation and the Team Initialization workflows as defined in the TEMPO framework [8].
In order to understand the complexity of a manual approach for solving the team assignment problem, one should briefly describe the criteria that the instructor must assess before allocating a team to each student. As discussed in [9], the criteria are clustered into four categories and are evaluated at the team or at the personal level.
The first set of criteria refers to the *practical constraints* that each team must satisfy. Thus, the final assignment solution must ensure that all teams have similar size (unless a project requires more manpower than others) and that a minimum number of test and development devices is available within each team. These practical constraints are important for creating equal technological contexts for all project groups and thus avoiding the situation in which a team cannot complete its tasks due to the lack of human or material resources.

The criteria related to *skill distribution* form the second cluster of the four previously mentioned. The goal is to "create a setting in which less experienced students can learn from their advanced peers" [9] and consequently to achieve the best learning outcome for all course participants. Hence, the assignment solution must guarantee a balanced software engineering experience, meaning that each team must include approximately the same number of students having technical skills and prior knowledge in different areas relevant to the project course (e.g. object-oriented software development, user experience design and software modeling). Moreover, possible project requirements must be considered in the team allocation process. For instance, if a project demands knowledge in a specific field or technology, the assignment solution must fulfil this requirement if possible (by e.g. allocating a student who has experience in the required domain to the corresponding team).
The *motivational factors* represent another set of criteria that must be considered in the team composition process. Based on the two-factor theory developed by Herzberg [14], the project priority can be seen as a *motivator*, that is, a factor that yields positive satisfaction. In this sense, it is believed that the student's motivation to work is significantly higher if the project to which the student is assigned is one of his top-priority options (we assume that each student creates his personal ranking by prioritizing all projects offered in the course). Therefore, an important aim of the assignment solution is to allocate each student one of his top-priority projects.
Last but not least, the gender distribution across the teams and the language skills of the students are two *personal criteria* that the instructor must take into account when composing the teams. It is believed that "fostering collaboration between genders can break down stereotypes" [9], and, therefore, a fair idea (and goal of our assignment process) is to distribute the female students equally over the teams. Furthermore, an ideal assignment solution should avoid the formation of "tight-knit communities", that is, groups of students within the same team who speak a language other than the course's official one. Although not being a fundamental aspect, but rather a hygiene-factor [14], the language skills criterion should be considered in the final assignment solution (unless its satisfaction would cause the break of other, more important constraints).
In a problem such as the team assignment for a project course, the four previously introduced criteria contradict each other. Thus, the manual team assignment approach constitutes a cumbersome process for the instructor (for the iPraktikum, it can last for up to five hours [8]), and it is generally impossible to decide whether the final solution is optimal, i.e. whether all criteria are satisfied. Moreover, the fulfillment of some criteria might be

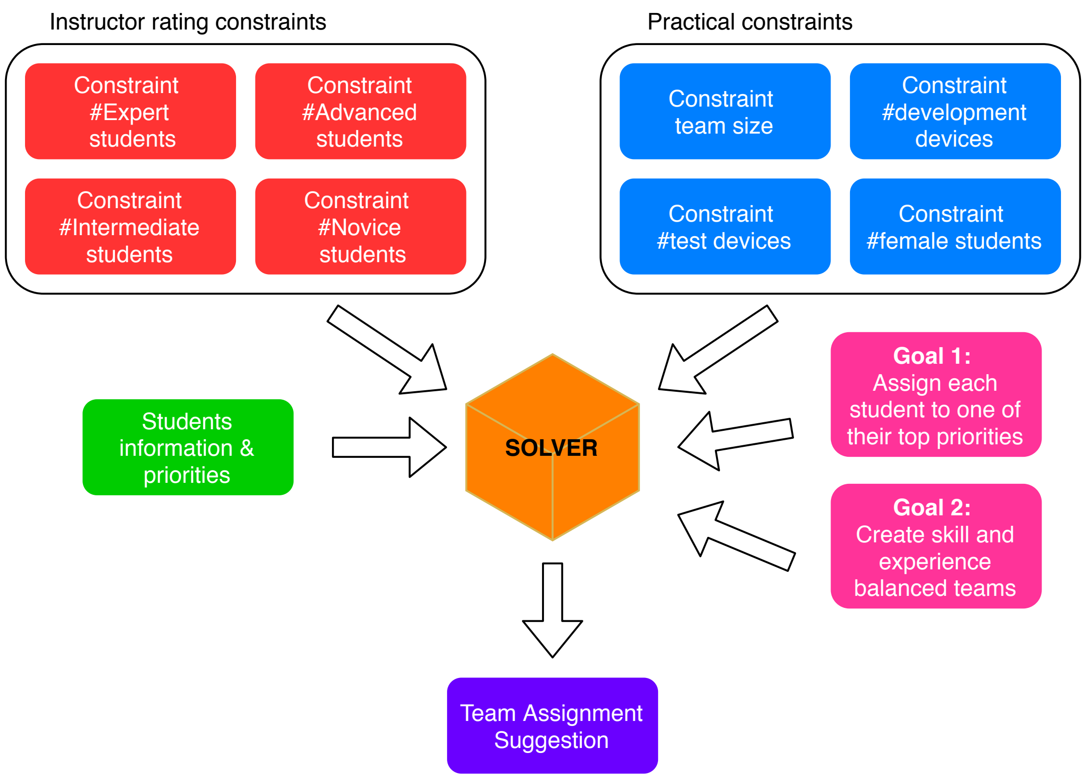

Instructor rating constraints
Constraint #Expert students
Constraint #Advanced students
Constraint #Intermediate students
Constraint #Novice students
Practical constraints
Constraint team size
Constraint #development devices
Constraint #test devices
Constraint #female students
Students information & priorities
SOLVER
Goal 1: Assign each student to one of their top priorities
Goal 2: Create skill and experience balanced teams
Team Assignment Suggestion

Figure 1.1: Factors considered in the team assignment process.

difficult to assess. For instance, there exists no mathematically verifiable rule to evaluate whether all students have been allocated to one of their top-priority teams in the final assignment solution (or, at least, there is no consensus definition of the "top").

Therefore, treating the assignment task as a *constrained single-objective optimization problem* (with the objective of allocating each student one of his top-prioritized teams) and solving it using an algorithmically suppored approach is a feasible idea which is implemented in the TEASE tool [10], [8]. Figure 1.1 illustrates the criteria related to the skill distribution (Instructor rating constraints), the practical constraints and the two goals that are considered in the team assignment process. The solver is the TEASE tool.

As expected, the assignment solution computed by TEASE has been shown to be more effective than the manually created one (not only is the solution found faster, but there also exists a metric to evaluate the optimality of the result - see Chapter 2). The tool is currently used for allocating the teams in the iPraktikum, aiming to solve the optimization problem while also ensuring the satisfaction of some hard constraints in each team. The problem has been formulated using linear programming and an implementation of the Simplex algorithm is used to solve it.

1.1 Problem

One of the requirements for the assignment solution is that all teams must be equal in terms of the skill distribution (unless otherwise specified). In the following, we present the original approach for achieving this goal, which is currently implemented in TEASE, and afterwards we introduce a new approach that addresses this requirement.

Using constraints for creating skill balanced teams

In the original single-objective team assignment problem in the iPraktikum, the skill balance requirement is modeled and solved as a constraint satisfaction problem being part of the global constrained optimization problem that TEASE solves. Thus, the instructor first has to specify bounds for the number of students having a particular skill level (i.e. instructor rating) that must be in each team and then the tool computes a team allocation solution that fulfils this requirement. In order to choose the appropriate number for each skill level and thus to guarantee an equal skill distribution across the teams, the instructor must have a comprehensive understanding of the problem, which may be difficult to achieve. This means that he has to know, for each type of instructor rating, the number of students that are labeled with it and what percent of the students this “skill level”-category represents, what the relation between the skill levels is: for instance, is it correct to assume that one expert student “counts” as three novice students?
Furthermore, the skill distribution constraint is likely to “interfere” with other constraints and, in the worst-case scenario, setting constraints such that the assignment problem is satisfiable might degenerate into a more time-consuming and complicated activity than the manual assignment. Hence, some questions that the instructor might encounter are: What if all expert students must be assigned to the same team because they are the only ones that have knowledge in a technology required in that team? Is it possible to create equally “experienced” teams in such a situation? What is more important: to have equal skill distribution or to ensure that the students from a particular team have the required knowledge to work on the corresponding project? Does a total order of importance of the constraints exist?
Thus, modeling and solving the team assignment task as a constrained single-objective optimization problem with the objective of distributing each stu-

dent to one of his top-priority teams might not guarantee an equal skill level distribution over the teams, despite the instructor's attempts to choose the "best" values for each constraint. The main reason for this problem lies in the nature of the "experience" attribute (which cannot be measured simply by counting how many students in each team have a specific instructor rating).

An experience objective function for creating skill balanced teams

Our proposed approach for creating skill balanced teams is to optimize an experience objective, such that the difference in the skill and experience levels between all teams is minimal. The new optimization problem is described in detail in Section 4.1. The goal is to minimize the mean absolute deviation of the experience level across all teams. The new objective function is defined based on the instructor ratings of the students. Thus, in the new assignment process, the instructor does not define constraints related to the skill distribution anymore.

1.2 Research Approach

In order to address the problem of composing teams such that each student is assigned to a high-priority project (given their personal project priority) and that all teams are equal in terms of the skill distribution, we propose a *constrained multi-objective optimization formulation* of the team assignment problem and analyze different methods to solve it.
Thus, we focus on three classes of multi-objective approaches - *a priori, a posteriori* and *interactive methods* - that differ in the decision maker's role (i.e. in our case, the instructor) in finding the "optimal" assignment solution. In a priori methods, the preferences of the decision maker are asked before running the algorithm and the best solution according to the given preferences is found. Contrary, in a posteriori methods, a set of solutions is generated and the decision maker may choose the best one among those. Nevertheless, in interactive methods, the decision maker is expected to state preferences at each iteration in order to get Pareto optimal solutions that are of interest to him [17].
The goal of this thesis is, on one side, to discuss the applicability of different multi-objective approaches to the team assignment problem in the iPraktikum. On the other side, we aim to investigate the strengths and weaknesses of multi-objective approach over a single-objective approach for solving this problem. To achieve these goals, we first analyze different multi-objective methods from a theoretical point of view, select the MOPSO, NSGA-II

and Tabu Search approaches and implement them to solve the team assignment task. We evaluate these algorithms considering e.g. performance and assignment-quality related criteria (Section 5.2). In the evaluation phase of our project (Chapter 5), we answer three research questions:

RQ1: What is the optimal parameter configuration for each algorithm?

RQ2: What are the strengths and weaknesses of each algorithm compared to the other evaluated approaches?

RQ3: What are the strengths and weaknesses of a multi-objective optimization approach over the existing single-objective optimization approach for the team assignment problem?

Nevertheless, we propose two new approaches that could be integrated in a multi-objective solver used in the team assignment problem in the iPraktikum (Sections 5.7.1, 5.7.2).

1.3 Outline

This thesis in structured as follows: in Chapter 2 we present the general concepts of single- and multi-objective optimization, describe methods for solving optimization problems and give a brief overview of the work carried out in the field of assigning teams using multi-objective approaches. In Chapter 3 we present the existing system for assigning teams and describe the scope and purpose of the proposed system. In Chapter 4 we formulate the multi-objective version of the team assignment problem in the iPraktikum and describe the NSGA-II, MOPSO and Tabu Search algorithms that we implemented to solve this problem. In Chapter 5 we discuss the results of the evaluation of the three mentioned algorithms and propose two new approaches that can be integrated in a multi-objective approach for solving the team assignment problem.

Chapter 2

Background & Related Work

In this chapter, we present the fundamental concepts of mathematical optimization (Section 2.1) and we briefly describe the research work in the field of algorithmic team composition (Section 2.3). In Section 2.1.1 we address the single-objective optimization problems, in 2.1.2 we explain the fundamental notions of multi-objective optimization and in 2.2 we give an overview of the most popular methods. In Section 2.4 we describe the original single-objective formulation of the team assignment problem in the iPraktikum.

2.1 Optimization

At its simplest definition, optimization is "a mathematical technique for finding a maximum or minimum value of a function of several variables subject to a set of constraints" [1]. Thus, given an objective function that must be minimized/maximized, an optimization technique aims to find the optimal value of that function, by systematically selecting input values from a corresponding domain. The method compares various solutions in multiple iterations until the optimal value of the function is found or, in case this value is unknown, until a satisfactory solution is discovered. Depending on the number of considered objective functions, we can distinguish between single-objective and multi-objective optimization.

2.1.1 Single-Objective Optimization

Single-objective optimization problems (SOPs) are the ones in which only one function must be minimized/maximized. In case of a minimization problem, the goal of an optimization algorithm is to determine the *global minimum*

[1] https://www.dictionary.com/browse/optimization

solution(s). Expressed mathematically: given the objective function $f : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}, \Omega \neq \emptyset$, for $\mathbf{x} \in \Omega$ the value $f^* \doteq f(\mathbf{x}^*) > -\infty$ is called *global minimum* if and only if

$$\forall \mathbf{x} \in \Omega : f(\mathbf{x}^*) \leq f(\mathbf{x}). \tag{2.1}$$

Ω is the feasible region of $\mathbf{x}$ and $\mathbf{x}^*$ is the global minimum solution.

2.1.2 Multi-Objective Optimization

Let us now define the general concept of a multi-objective optimization problem (MOP, also called multicriteria optimization, multiperformance or vector optimization problem [4]). As described in [20], a MOP is the problem of finding "a vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria which are usually in conflict with each other. Hence, the term "optimize" means finding such a solution which would give the values of all the objective functions acceptable to the decision maker".
The *decision variable space* contains all possible solutions for a MOP. Similarly, we use the term *objective space* to denote the coordinate space within which vectors resulting from evaluating a MOP's solutions are plotted. Thus, if we assume that a MOP has n decision variables aggregated in a vector $\mathbf{x} = [x_1, ..., x_n] \in \mathbb{R}^n$ and k objective functions $F(\mathbf{x}) = [f_1(\mathbf{x}), ..., f_k(\mathbf{x})]^T$, then the following two Euclidian spaces must be considered: an n-dimensional space of the decision variables in which each coordinate axis corresponds to a component of the vector $\mathbf{x}$ and a k-dimensional space of the objective functions in which each coordinate axis corresponds to a component vector $f_l(\mathbf{x})$. Moreover, every point in the first space represents a solution and gives a certain point in the second space, which determines the "quality" of this solution in terms of the objective function values [4].
The objective functions of a MOP almost always conflict, which leads to a partial, rather than total, ordering on the search space. In fact, finding the global optimum of a general MOP is an NP-Complete problem [4].
Although a SOP might have a unique optimal solution, for a MOP there possibly exists a set of uncountable solutions, which are found using Pareto Optimality Theory. Thus, the notion of "optimum" changes, because in MOPs, the aim is to find good compromises rather than a single solution as in global optimization (single-objective optimization). For the following definitions, we assume a minimization MOP.

We say that a vector $u = [u_1, ..., u_k]$ from the objective space *dominates* another vector $v = [v_1, ..., v_k]$ (denoted by $u \preceq v$) if and only if u is partially less than v, i.e. $\forall i \in \{1, ..., k\}, u_i \leq v_i$ and there exists an $i \in \{1, ..., k\}$ such that $u_i < v_i$. A solution $\mathbf{x} \in \Omega$ is said to be Pareto Optimal with respect to the universe Ω if and only if there is no $\mathbf{x'} \in \Omega$ for which $v = F(\mathbf{x'}) = [f_1(\mathbf{x'}), ..., f_k(\mathbf{x'})]$ dominates $u = F(\mathbf{x}) = [f_1(\mathbf{x}), ..., f_k(\mathbf{x})]$ [4]. In other words, we say that $\mathbf{x} \in \Omega$ is Pareto optimal if there exists no feasible vector $\mathbf{x'} \in \Omega$ which would decrease some objective without causing a simultaneous increase in at least one other objective.
Further, for a given minimization MOP and its corresponding vector function F, we define the *Pareto Optimal Set* as follows:

$$\mathcal{P}^* := \{\mathbf{x} \in \Omega \mid \neg \exists \mathbf{x'} \in \Omega \quad F(\mathbf{x}) \preceq F(\mathbf{x'})\}. \tag{2.2}$$

As equation 2.2 suggests, Pareto optimal solutions are those solutions within the decision space whose corresponding objective vector components cannot be all simultaneously improved [4]. Their corresponding vectors in the objective space are termed *nondominated.*
When plotted in the objective space, the solutions from the Pareto Optimal Set form the *Pareto Front* :

$$\mathcal{PF}^* := \{\mathbf{u} = F(\mathbf{x}) \mid \mathbf{x} \in \mathcal{P}^*\}. \tag{2.3}$$

Thus, the Pareto Front is composed of the nondominated solutions. In general, it is not easy to find an analytical expression of the line or surface representing the Pareto Front, in most cases it actually turns out to be impossible! The normal procedure to generate (or, more precisely, to approximate) the Pareto Front is to compute many points in the universe Ω and their corresponding $F(\Omega)$. When the number of these points is large enough, it is possible to determine the nondominated points and to produce the Pareto front [4]. After a specific algorithm has computed the Pareto optimal solutions set, a decision-maker (DM, in our case, the instructor) selects a solution from it that, from their perspective, represents the best trade-off between all objective functions.

2.2 Multi-Objective Methods

In this section we give an overview of multi-objective methods and global search and optimization algorithms. In 2.2.1 we describe a priori methods, in 2.2.2 we concentrate on a posteriori methods and in 2.2.3 we present interactive methods.

Figure 2.1 provides an overview of the most important global search and optimization approaches. Enumerative algorithms are the simplest search strategy (and the most inefficient, especially when the search space is large) and evaluate each possible solution within a defined search space. Deterministic algorithms incorporate problem domain knowledge [4], but are unsuitable for many irregular, real-world multi-objective problems, which are e.g. high-dimensional. For solving such irregular problems, stochastic algorithms have been developed and are currently used in many fields. In our project, we implemented three stochastic algorithms, as described in Section 4.2, 4.3 and 4.4.

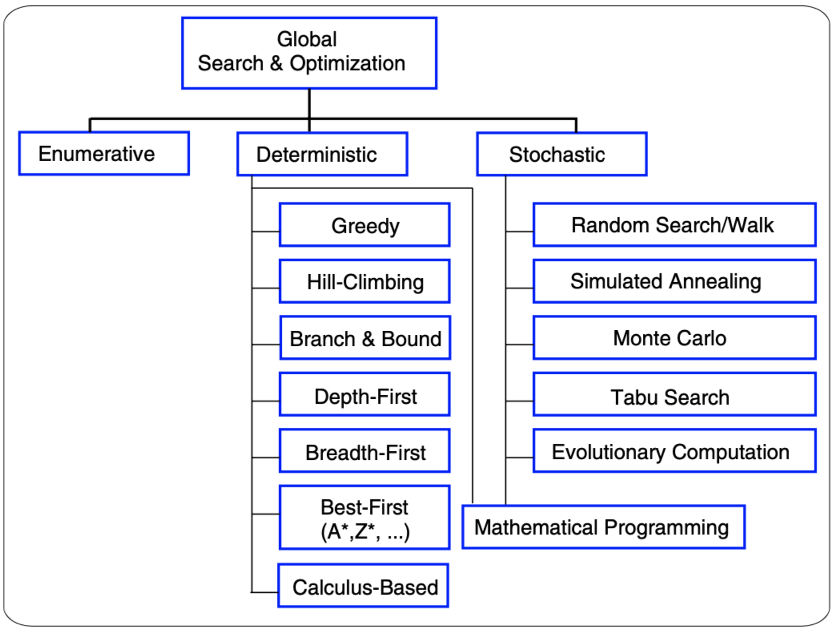


Figure 2.1: Global Optimization Approaches (Source: [4]).

One of the most popular classification of multi-objective optimization techniques was proposed by Cohon and Marks [5]. They distinguished between three classes, namely techniques which rely on prior articulation of preferences (1), generating techniques (2) and techniques which rely on progressive articulation of preferences (3). This classification is based on the way in which each technique handles the problems of searching and making (multicriterion) decisions [4]:

1. ***A priori* Preference Articulation** : make decision before searching (decide $\Rightarrow$ search).

2. ***A posteriori* Preference Articulation** : search before making decisions (search $\Rightarrow$ decide).

3. **Interactive Preference Articulation**: integrate search and decision making (decide $\Leftrightarrow$ search).

2.2.1 A Priori Methods

This class contains methods that assume that either concrete desired achievable goals or a pre-ordering of the objectives can be specified by the DM before the search is performed [5], [4]. In the following, we present three a priori methods and discuss their applicability for our team assignment optimization problem.
The first approach from this class is the *Weighted Sum Method.* Assuming that there are k objectives functions f_i which should be minimized, one could define a single aggregated objective function as follows:

$$\min_{x \in S} \quad \sum_{i=1}^{k} w_i \cdot f_i(x) \tag{2.4}$$

where $\sum_{i=1}^{k} w_i = 1, w_i \geq 0 \quad \forall i \in \{1, ..., k\}$.

The weights (w_i) are defined by the decision maker and suggest the importance of each objective (a high weight indicates a high importance of the corresponding objective). These can be set (and never changed) before a specific global search and optimization algorithm (that integrates such a method) is run or they can be adapted by the algorithm in each of its iterations. The first case describes the *conventional weighted sum method* (CWA), while the second is called *dynamic weighted sum method* (DWA) [15]. CWA has been applied in our MOPSO implementation, while DWA is used in the Tabu Search algorithm for solving the team assignment problem. We implemented this method because it provides an efficient strategy for computing solutions, given that one does not need to calculate the dominance of each solution and also to store the Pareto optimal solutions.
Another a priori approach is the *ϵ-Constraint Method* [4]. Given k objectives, the idea is to choose only one objective to be optimized, to give the others an upper bound and to consider them as constraints, as the following formulas indicate:

$$\text{minimize} \quad f_i(x) \tag{2.5}$$

$$f_j(x) \leq \epsilon_j \quad \forall j \in \{1, ..., k\} \setminus \{i\} \tag{2.6}$$

This approach is not suitable for solving our optimization problem, since the instructor (DM) does not know beforehand what the values of the objective functions of an 'ideal' assignment solution are and, thus, it is difficult to set reasonable bounds.
Goal Programming is another a priori method, in which the DM must define goals for each objective. Extra variables (e.g. $d_i^+ \geq 0$ and $d_i^- \geq 0$) are introduced and deal with the deviation from the goal for each objective. Thus, d_i^+ represents the amount by which the algorithm deviates upwards from the goal and d_i^- denotes the downwards deviation level from the goal. For example, the priority and experience objectives could be expressed using goal programming as follows:

$$f_{priority} = goal_{priority} + P^+ - P^- \tag{2.7}$$

$$f_{experience} = goal_{experience} + E^+ - E^- \tag{2.8}$$

$$P^+, P^-, E^+, E^- \geq 0 \tag{2.9}$$

There are two goal programming approaches, namely *weighted goal programming (non-preemptive goal programming)* and *lexicographic ordering (pre-emptive goal programming)*. In weighted goal programming, the DM should attach weights to each of the variables associated with upward/downward deviation and then an algorithm is used to solve a single problem whose objective is to minimize this weighted deviation sum. The weight (e.g. w_{P+}) is associated with one percentage deviation from the current goal for each variable. Continuing the above example, we have:

$$\begin{aligned}\text{minimize } & \mathrm{w}_{P+} \cdot \frac{P^{+}}{goal_{priority} \cdot \frac{1}{100}} - w_{P-} \cdot \frac{P^{-}}{goal_{priority} \cdot \frac{1}{100}} + \\ & \mathrm{w}_{E+} \cdot \frac{E^{+}}{goal_{experience} \cdot \frac{1}{100}} - w_{E-} \cdot \frac{E^{-}}{goal_{experience} \cdot \frac{1}{100}}\end{aligned} \tag{2.10}$$

$$\text{subject to equations 2.7, 2.8, 2.9} \tag{2.11}$$

In lexicographic ordering, the DM should decide priority levels for the goals (priority level 1 for the most important goal) and first satisfy priority one goals, then priority two goals and so on. A sequence of related problems is solved this way.
The goal programming method would be suitable for our team assignment optimization problem. The DM could set the goals for the priority and experience objectives 1 and 0, respectively, because these are the minimal values of these functions (see Section 4.1).

2.2.2 A Posteriori Methods

The a posteriori methods "do not require prior preference information from the DM" [4]. Evolutionary and swarm algorithms are included in this cluster [4] and are briefly presented in the following.
Evolutionary algorithms are population-based metaheuristic optimization methods inspired by biological evolution. A detailed explanation about this class of algorithms is provided in [4]. As part of this class, a genetic algorithm (GA) computes efficient solutions for optimization problems by relying on operators inspired from biological processes, such as mutation, crossover and selection [21]. Starting from an initial random set of candidate solutions, which form a population, a GA computes new solutions in multiple iterations (i.e. generations), by modifying the results of the previous generations. Each solution has a set of characteristics forming a corresponding chromosome and its own fitness, which is a 'composite' value of the objective functions. Given the minimization/maximization problem, the goal of a GA is to find better solutions in each generation, until e.g. a predefined number of iterations are completed. Thus, in each generation, after some operations were performed with a specific probability, a new population of children solutions is obtained from parent solutions, which are stochastically selected from the individuals of their generation. The common operations are the *crossover*, in which the parents 'exchange' genes of information, and the *mutations*, when random changes occur in a chromosome.

NSGA-II (*Nondominated Sorting Genetic Algorithm II*) is a popular genetic multi-objective evolutionary algorithm (MOEA) that is widely used for solving optimization problems due to its positive characteristics, namely good computational complexity, elitism approach or "passing good solutions down the generations", diversity of solutions, convergence speed [4]. We applied this algorithm to solve our team assignment problem. Details about the implementation of this method, the reasons why we chose it and also a discussion on the advantages and disadvantages that this approach has can be found in Section 4.2.

Proposed by James Kennedy and Russell Eberhart [16], PSO (Particle Swarm Optimization) is a population-based metaheuristic which mimics *swarm intelligence* in nature. This approach is inspired by the choreography of a bird flock and simulates the movements of the members of the group searching for food [4]. Swarm intelligence represents the collective behavior of a self-organized system consisting of independent intelligent agents which have no centralized unit control and seek to achieve a goal as a team [1]. The MOPSO (Multi-Objective Particle Swarm Optimization) algorithm is an extension of the single-objective PSO (Particle Swarm Optimization) algorithm that handles multiple objectives. We implemented this algorithm to solve the team assignment problem and the details of our approach are covered in Section 4.3. Other swarm algorithms are ant colony optimization algorithms [2] and the multi-objective grey wolf optimizer [19].

2.2.3 Interactive Methods

Interactive or progressive methods consist of three stages, as described in [4]: (1) find a nondominated solution, (2) get the opinion of the DM on this solution and modify the preferences of the objectives accordingly and (3) repeat steps (1) and (2) until the DM is satisfied with the solution or no further improvements are possible.

NIMBUS [3] is a method of such type. The idea is that the DM evaluates the values of the objective functions computed for the current solution and divides the objective functions into five categories. These classes include objectives whose values (1) should be improved, (2) should be improved up to some aspiration level, (3) are satisfactory at the moment, (4) are allowed to impair up to some bound and (5) are allowed to change freely. Afterwards, NIMBUS generates another solution by taking into account the preferences and the whole process is repeated until the DM is satisfied with the result.

[2] `http://www.scholarpedia.org/article/Ant_colony_optimization`

[3] `https://wwwnimbus.it.jyu.fi/info.html`

Zionts-Wallenius method is another interactive approach, in which the DM is requested to provide answers to yes and no questions regarding certain trade-offs between the objective functions that he likes or dislikes. A detailed description about this method can be found in the paper in which it was first described [24].

2.3 Algorithmic Team Composition

This section presents related work carried out in the field of composing teams using multi-objective optimization techniques. For each of the following approaches for solving the team allocation problem, we briefly describe its general idea and mention the algorithm that it implements.
Gonsalves and Itoh [22] propose a *skill-to-time* model for assigning the available personnel to the various tasks in a software development project (i.e. "allocating the right personnel to the right job, at the right time, and at the right cost"). The personnel skill levels are mapped onto a 1 to 5 scale and each day involves a cost of the staff members. The task processing time varies depending on the match between the skills required to perform the task and those possessed by the personnel (a skill mismatch increases the estimated task processing time). Also, the objectives are minimizing the project development cost and duration and the constraints which are considered in this approach refer to specific task precedence relations and the personnel availability. To solve this assignment problem the Multi-Objective Particle Sworm Optimization (MOPSO) Algorithm has been used.
By taking into account the interaction between the team members, a solution to the optimization Multiple Team Formation Problem (MTFP) based on sociomeric techniques is introduced by Esgario et al. [11]. The Sociometric Matrix (illustrating the cohesion degree of the group) and the Project Requirement Matrix (containing the number of people from a particular department which are needed for a specific team) are required for solving the MTFP. The goal is to maximize the overall cohesion degree, while also complying to each project requirements and assigning every member to exactly one team. In a first phase of this approach, all possible allocation matrices are generated using a Genetic Algorithm. For each solution, a penalty score is computed, which measures to what extent the solution violates the constraints of the problem. In a subsequent phase, the difference between the general cohesion of each solution and its penalty score is calculated and afterwards the solution having the greatest value is selected.
Another multi-objective optimization approach for composing a single team for new product development (NPD) projects is proposed by Lianying and

Xiang Zhang [23], who consider both the comprehensive capabilities and the interpersonal relationships of all possible members of the team. Thus, a fuzzy Analytic Hierarchy Process method based on fuzzy linguistic preference relations is implemented in order to assess the capabilities of the candidates. Moreover, the members' personalities are modeled using the Myers-Briggs Type Indicator (MBTI), which makes it possible to predict the cooperative abilities of the future teammates. The objectives considered in this approach are selecting the team members that best meet the requirements of the NPD project and choosing the people that have the best interpersonal relationships with the rest. The MOPSO algorithm has been used to solve this optimization problem.
In our literature review we did not find any work that focuses on assigning teams in an educational context or on multi-objective optimization techniques for composing teams in multi-project courses.

2.4 Problem Formulation - Single-Objective Optimization Version

In the following, we present the single-objective formulation of the team assignment optimization problem, as introduced in [8].
Let $T = \{1, ..., t\}$ with $|T| = t$ and $S = \{1, ..., s\}$, with $|S| = s$ be the team/project and the student sets respectively. We denote with $P_{ij} \in \{1, ..., t\}$ the priority student i gave to project j, with $P_{ij} < P_{ik}$ meaning that the student i expressed more motivation for project j compared to project k. Further, let $\mathit{Rating} = \{Expert, Advanced, Intermediate, Novice\}$ be the set containing the possible instructor ratings.
The vector $\mathbf{x} = [x_{11}, x_{12}, ..., x_{1t}, ..., x_{s1}, x_{s2}, ..., x_{st}]^T \in \Omega$ consists of $s{\cdot}t$ discrete decision variables (i.e. numerical quantities for which values in $\{0, 1\}$ are to be chosen in our optimization problem [4]) and denotes an assignment of each student to a team, where:

$$x_{ij} = \begin{cases} 1 & \text{if student i is assigned to team j} \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

The universe Ω is called the *feasible region* and contains all possible assignment solutions $\mathbf{x}$ that satisfy the problem constraints. Given the objective function $f_1 : \Omega \subseteq \{0, 1\}^{s \cdot t} \to \mathbb{R}$, the SOO problem is expressed as follows:

$$\text{minimize} \quad f_1(\mathbf{x}) = \sum_{i \in S} \sum_{j \in T} x_{ij} \cdot P_{ij}, \tag{2.13}$$

$$\text{subject to} \sum_{j \in T} x_{ij} = 1, \tag{2.14}$$

$$n_{min} \leq \sum_{i \in S} x_{ij} \leq n_{max} \qquad \forall j \in T, \tag{2.15}$$

$$\sum_{i \in S} x_{ij} \cdot d_i^{Mac} \geq m_{Mac,min} \qquad \forall j \in T, \tag{2.16}$$

$$\sum_{i \in S} x_{ij} \cdot d_i^{iDevice} \geq m_{iDev,min} \qquad \forall j \in T, \tag{2.17}$$

$$m_{r,min} \leq \sum_{i \in S} x_{ij} \cdot l_i^r \leq m_{r,max} \quad \forall j \in T, \forall r \in Rating, \tag{2.18}$$

$$\sum_{i \in S} x_{ij} \cdot g_i \geq m_{g,min} \qquad \forall j \in T \tag{2.19}$$

where:

$$d_i^{Mac} = \begin{cases} 1 & \text{if student } i \text{ has a Mac} \\ 0 & \text{otherwise.} \end{cases} \tag{2.20}$$

$$d_i^{iDevice} = \begin{cases} 1 & \text{if student } i \text{ has an iDevice} \\ 0 & \text{otherwise.} \end{cases} \tag{2.21}$$

$$g_i = \begin{cases} 1 & \text{if student } i \text{ is a female person} \\ 0 & \text{otherwise.} \end{cases} \tag{2.22}$$

$$l_i^r = \begin{cases} 1 & \text{if student } i \text{ received the instructor rating } r \\ 0 & \text{otherwise.} \end{cases} \tag{2.23}$$

Thus, the aim is to minimize the linear objective function f_1 and to implicitly assign each student to his top priority team.
Equation 2.14 suggests that each student must be assigned to exactly one team, while equations 2.15, 2.16 and 2.17 denote three practical constraints [9]: the size of each team must respect the lower and upper bounds n_{min} and n_{max} respectively, each team must have at least $m_{Mac,min}$ Mac devices (development devices) and $m_{iDevice,min}$ iDevices (test devices). Also, equation 2.18 denotes a constraint related to the skill distribution in each team [9]: there should be at least $m_{r,min}$ and at most $m_{r,max}$ students with the instructor rating r in each team. This constraint is applied for each of the four possible instructor ratings. Last but not least, equation 2.19 describes a constraint considering the gender distribution personal criterion [9]: every team should have at least $m_{g,min}$ female members.
The above presented SOO problem formulation treats all teams equally by defining specifc global constraints. The model can be adapted depending on possible constraints that specific teams must satisfy (for instance, if a team requires more development devices than the others, then a new constraint is defined for this team and considered in the algorithm that solves the optimization problem).

Chapter 3

Analysis

This chapter presents the results of the Requirements Analysis phase of our project. Section 3.2 depicts three visionary scenarios, Section 3.3 contains the functional and nonfunctional requirements that we identified and in Section 3.4 we describe the Analysis Object Model of the problem domain that we address in this thesis. Dynamic models can be found in Chapter 4, where we discuss the algorithms in detail.

3.1 Overview

In this section, we give an overview of the current system used for composing teams in the iPraktikum and we also present the aim of the system that we propose. In Section 3.1.1, we briefly present TEASE and in Section 3.1.2 we describe the scope and purpose of the new system.

3.1.1 TEASE as Existing System

TEASE (**TE**am **A**llocator for **S**oftware **E**ngineering courses) is a reference implementation of the TEMPO framework [8] and was developed at the Chair for Applied Software Engineering. The user interface was realized with the framework AngularJS[1] and the tool was written in TypeScript.
The user can import into it a data set which contains information about the students that should be assigned to teams. TEASE provides a GUI in which the user can define constraints that should be fulfilled in the assignment solution that the tool computes. These can be global, that is, applied for all teams, or team-specific.

[1]`https://angularjs.org`

TEASE generates an assignment solution using a single-objective approach, namely the Simplex algorithm [6]. It uses an open-source library for solving linear-programs[2]. The solution is displayed on a Dashboard, that shows, for each team, the students that were assigned to it. The user can then refine the suggested solution, by moving students from one team to another one. Moreover, the tool provides team statistics that show the average priority and the priorities distribution for each team. This way, the user can analyze the assignment quality of the solution.

3.1.2 Scope & Purpose of the Proposed System

The purpose of the proposed system is to provide three multi-objective approaches for solving the team assignment problem in the iPraktikum. These are the MOPSO, Tabu Search and NSGA-II algorithms and are implemented in Java[3]. The user can import a CSV file that contains information about the students that should be assigned to teams. He can define lower and upper bounds for a set of global constraints that relate to the team size, the number of development and test devices that should be available in each team and also to the number of female students that should be present in each team. Then, each algorithm computes an assignment solution for the defined constrained optimization problem (if it is solvable), by optimizing two objective functions (an experience and a priority objective function, as described in Section 4.1). The assignment solution is written on a CSV file, which can be imported into TEASE. For each student, the assigned team is indicated.

3.2 Visionary Scenarios

The following visionary scenarios illustrate the applicability of a multi-objective approach for solving the team assignment problem in the iPraktikum. The scenarios form a continuous sequence of events, which means that each scenario is built on the events presented in the previous ones.

Visionary Scenario I: First attempt to create teams using a single-objective approach

Emma, one of the instructors of the iPraktikum, wants to create the teams of all students who participate in this multi-project practical course. The Swift Bootcamp has just finished and all 85 students have filled in the iPraktikum

[2]`https://github.com/JWally/jsLPSolver`

[3]`https://java.com/en/download/`

Team Initialization Questionnaire [8]. The instructors have already assigned an instructor rating to each student, which is based on the student's evolution in the bootcamp and also on their questionnaire answers. This year's iPraktikum edition has 10 industry partners, so there are 10 team projects on which the students should work in the upcoming months. Emma has a CSV file containing data about all students, such as their personal information, knowledge and interests in software engineering and related fields, the priorities that they gave to each project and their assigned instructor rating. She opens TEASE, which implements a single-objective approach for composing teams, and imports the student data into it. She knows from the previous editions that she should compose the teams such that the students are assigned to one of their top priority projects. Also, the teams should be balanced in terms of their size, distribution of test and development devices, gender distribution and skill level.
Emma wants to set the global constraints and throws an eye on the statistics provided in TEASE. She sets the constraint that each team should consists of at least 8 and at most 10 students. She sees that there are 8 Expert, 28 Advanced, 36 Intermediate and 13 Novice students. Emma does some fast math calculations and sets the lower and upper bounds for the instructor rating constraints as follows: 0 <= #Expert <= 1, 2 <= #Advanced<= 3, 3 <= #Intermediate<= 4 and 1 <= #Novice<= 2. TEASE shows that there are 48 and 41 development and test devices, respectively, so she sets the following device constraints: 4 <= #development devices<= 5, 4 <= #test devices<= 5. There are 11 female students, so she sets the constraint that each team should be composed of at least 1 and at most 2 female students.
Emma lets TEASE compute an assignment solution. When she sees the result, she is very surprised. No constraint is broken, but the average team priority distribution is low. Most of the Expert and Advanced students were assigned to one of their last 4 priorities.
When she briefly analyzes the assignment solution, Emma notices that no Expert or Advanced student has a test or a development device and all female students have Intermediate or Advanced instructor ratings. Emma starts thinking..."What an odd situation! It looks like, if I focus on the priority distribution, I can't make every student happy when I apply these constraints... should I give up on the constraints regarding the instructor ratings? It seems like these are responsible for the low priority distribution. But it is important to have skill-balanced teams...". TEASE does not provide multi-objective algorithms for assigning teams...
After some minutes in which she has been trying to change the instructor rating constraints in the hope of improving the priority distribution, Emma remembers that Mia, another iPraktikum instructor, has been studying dif-

ferent multi-objective approaches for solving the team assignment problem. Emma goes to Mia's office and explains her the problem that she encountered while trying to create the teams. Mia tells her that she could use a multi-objective algorithm to solve the assignment problem. This way, Emma would need to define just the practical and gender constraints and, instead of optimizing only one priority objective, an algorithm would also optimize an 'experience' objective. The composed teams should be balanced in terms of the experience level, which is derived from the instructor ratings of the students. Mia tells Emma that she could use the MOPSO, Tabu Search and NSGA-II algorithms to create the teams. "I studied them and I know that they work well for the team assignment problem", Mia says. Emma does not know the difference between these algorithms. Mia tells her that NSGA-II is a genetic algorithm which computes multiple optimal solutions, that MOPSO seems to compute the best assignment solution with respect to the two objectives, but it might take a while until it does so, and that Tabu Search is a local search algorithm and probably the simplest one among these three.

Visionary Scenario II: Analyze and refine the assignment solution

Because she likes its explanation (and also its name), Emma chooses the Tabu Search algorithm. She uses her data set and Mia's Tabu Search implementation to create the teams. Once the team are composed, the algorithm displays the mean priority distribution and the mean deviation of the experience level corresponding to the generated assignment solution.
Emma imports into TEASE the resulting CSV file containing the initial participants data plus the allocated team for each student. She notices that all practical and gender constraints that she defined at the beginning are satisfied and analyzes the priority distribution in each team, by checking the statistics. Emma notices that there are teams in which the priority objective has a very good value (around 2), while some other teams have low average priority distribution (around 6). This is due to the fact that the multi-objective algorithm optimizes two objective functions simultaneously and, thus, it is not possible to generate an assignment solution that has global minimum values for both the experience and priority objectives.
She refines the initial solution, by moving some students from a team to another one while ensuring that the constraints are not broken. Once she is satisfied with the assignment, Emma goes back to Mia's office and asks her how does she know that the resulting teams are balanced in terms of the experience level. Mia tells her colleague that the algorithm solves the assignment optimization problem, such that the mean deviation of the ex-

perience level in each team and the mean priority are minimum. She also shows Emma the results of the work she has been doing to evaluate the three algorithms, which prove that they perform as expected.

Visionary Scenario III: Compare the assignment solutions generated by multi-objective algorithms and select one to apply in the iPraktikum

After some months, Mia integrated all three multi-objective algorithms (MOPSO, Tabu Search and NSGA-II) into TEASE. She asks Emma to use the new version of TEASE in order to assign teams for the new edition of the iPraktikum. Emma imports a CSV file into TEASE, defines some global constraints as described in Visionary Scenario I and then lets the tool compute assignment solutions. The tool displays the solutions generated by each algorithm - for each student, it shows the team to which it has been assigned. The tool displays a message: "No constraints are broken". Also, TEASE provides statistics related to the objective values of the assignment solutions. Emma notices that MOPSO solution has the best global priority objective value (around 4), while Tabu Search has the worst value of this objective (around 5). In terms of the experience objective, the statistics show that Tabu Search and NSGA-II have the best value of this objective function (around 0.2). Emma also sees the Dashboard with the students distribution into teams, for all generated assignment solutions. She notices that, in the MOPSO solution, almost half of the students were assigned to one of their top 3 priorities, while less than 10 students were distributed to one of their last 3 project priorities. After analyzing the strengths and weaknesses of all assignment solutions, Emma chooses the MOPSO assignment solution.

3.3 Requirements

In this section, we present the requirements for a multi-objective approach for composing teams. These are derived from the visionary scenarios described in Section 3.2. Section 3.3.1 contains the functional requirements, while section 3.3.2 outlines the nonfunctional requirements of a multi-objective approach for solving the team assignment problem in the iPraktikum.

3.3.1 Functional Requirements

In the following, we present the functional requirements of a multi-objective approach for composing teams.

FR1 Given a concrete number of projects, the instructor can assign exactly one team for each project.

FR2 The instructor can define global constraints, which are applied for all teams that will be formed. The constraints relate to the size of each team, the number of development devices, the number of test devices and the number of female students. For each constraint, a lower and an upper bound must be defined.

FR3 Each student is assigned to exactly one team.

FR4 The multi-objective approach should work with the student data collected from their questionnaire answers (the questionnaire can be found in [8]).

FR5 The multi-objective approach should minimize two objective functions, namely a priority and an experience objective (see Section 4.1). If the defined optimization problem is solvable, the generated solution must satisfy the constraints.

FR6 The instructor receives one or multiple assignment solutions if they exist.

FR7 The instructor can see the objective values corresponding to the generated assignment solution(s) and the team to which each student is assigned.

FR8 The instructor can refine the suggested assignment solution by moving students from one team to another one using the TEASE GUI.

3.3.2 Nonfunctional Requirements

The next nonfunctional requirements follow the URPS model, as described in [3].

NFR1 The multi-objective approach should be usable for generating teams without prior knowledge in the field of optimization (Usability).

NFR2 The multi-objective approach should generate an assignment solution in less than 20 seconds (Performance).

NFR3 The multi-objective approach should deal with student sets of variable size, up to 300 students. (Adaptability).

NFR4 The multi-objective approach should support the optimization of further objective functions, besides the priority and experience objectives. This can be done either using the a priori weighted sum method (each objective gets a weight that denotes its 'importance') or using other a posteriori methods that optimize multiple objective functions simultaneously. (Extensibility).

3.4 Analysis Object Model

The Analysis Object Model of our problem domain is illustrated in figure 3.4. Following the Analysis Object Model of TEMPO [8], we organized the classes into three packages, namely the `Assignment Package`, the `Team Package` and the `Student Package`.
For each `Student`, various `Student Information` exists. This can be either `Personal Information` about the student, such as their name, gender or the current semester of study, a `Skill` relevant for the iPraktikum (e.g. skill in the web development field) or an `Interest` in e.g. a technology used in a specific project. Each `Student` has their own `TeamPriority` list, which includes, for each team, the priority that the `Student` assigned to it. Also, a `Student` can possess multiple `Devices`, which can be either a `Mac` or an `IDevice`. Nevertheless, each `Student` is assigned one of the four possible `Instructor Ratings`.
Moving to the `Team Package`, we see that each `Project` might consist of multiple `Requirements`, that is, specific `Skills` that are needed for its development. For every `Project`, a `ProjectTeam` should be created.
The `Assignment Package` contains the classes required for creating an assignment solution, given a concrete `AssignmentProblem` which has multiple `AssignmentCriteria`.
Thus, an `AssignmentCriterion` can be either an objective, namely the `PriorityObjective` or the `ExperienceObjective`, or a `GlobalConstraint`. To solve the `AssignmentProblem`, one needs a `Solver`, which will generate one or multiple `TeamSuggestions`. We used the Strategy Pattern [12] to model how a suitable multi-objective algorithm can be chosen to solve the `AssignmentProblem`. `MOPSOSolver`, `TabuSearchSolver` and `NSGA-IISolver` provide different algorithms for solving the assignment problem and all im-

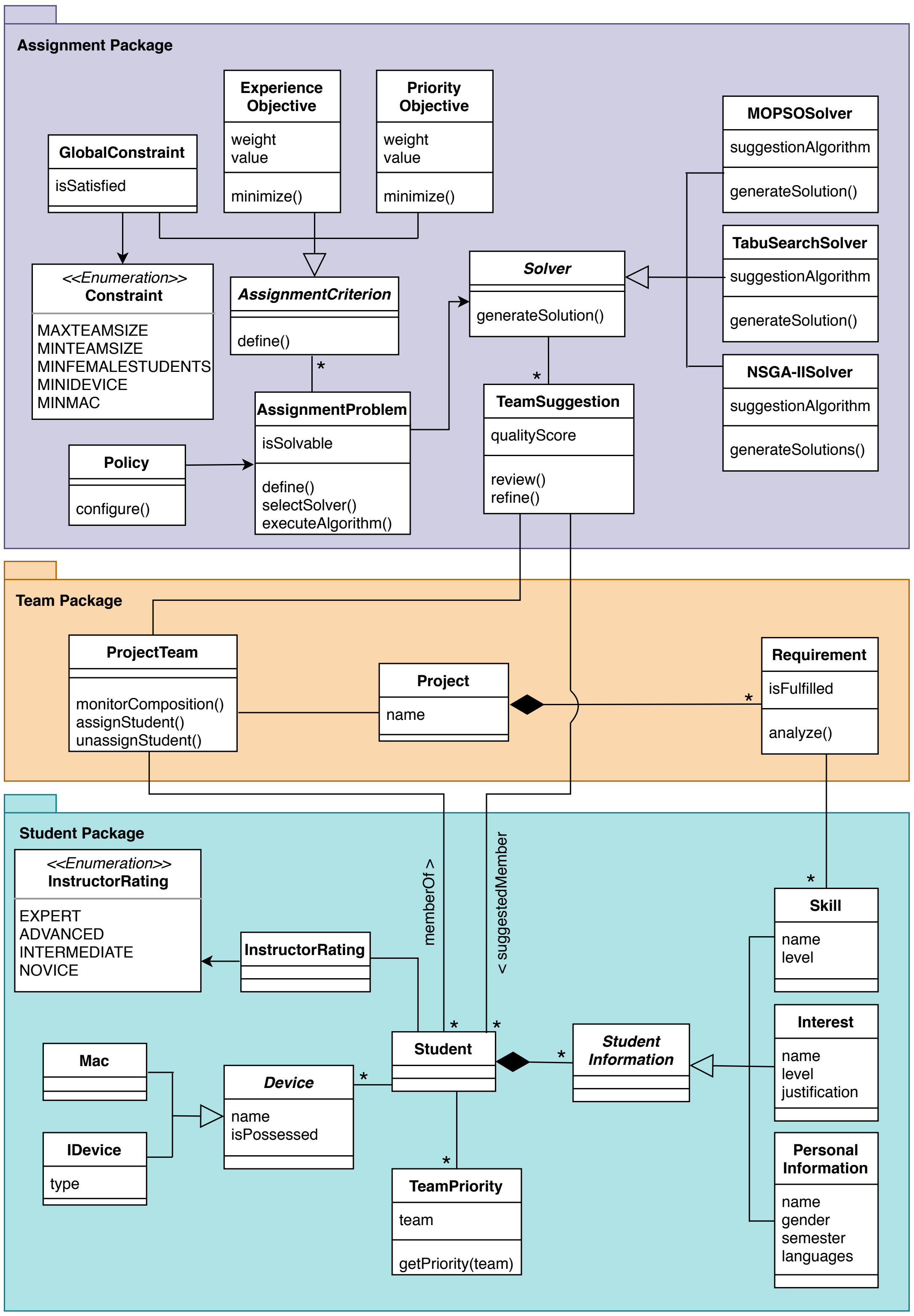


Figure 3.1: Analysis Object Model (UML Class Diagram)

plement the method `generateSolution()`. The `Policy` selects the `Solver` in the `AssignmentProblem` class by calling `selectSolver()`. This determines which algorithm is used to generate the teams when the method `executeAlgorithm()` is called. The decision regarding what `Solver` to use is based on different criteria that are evaluated in the `configure()` method. For instance, if one wants to receive multiple team assignment Pareto solutions, then the `NSGA-IISolver` should be used. Otherwise, if a local search approach is preferred, the `TabuSearchSolver` is the ideal decision. The `MOPSOSolver` is the best option if one wants to use a population-based algorithm that has good computational complexity (see Section 5).

Chapter 4

Algorithmic Design

In this chapter we present three multi-objective algorithms that we implemented to solve the team assignment problem in the iPraktikum. In Section 4.1 formulate the multi-objective version of the problem and in 4.2, 4.3 and 4.4 we discuss the NSGA-II, MOPSO and Tabu Search algorithms.

4.1 Problem Formulation - Multi-Objective Optimization Version

In the following, we present the MOO formulation of the team assignment problem. We chose the four constraints related to the instructor ratings (as described in the SOO problem formulation) and converted them into an experience objective function.

Let $\mathbf{x} = [x_{11}, x_{12}, ..., x_{1t}, ..., x_{s1}, x_{s2}, ..., x_{st}]^T \in \Omega$ be the vector of decision variables and $F(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x})]^T$ be the vector function, where $|T| = t > 0$ and $|S| = s > 0$ are the cardinalities of the team and student sets, respectively. The multi-objective optimization team assignment problem is:

$$\text{minimize} \quad f_1(\mathbf{x}) = \frac{\sum_{j \in T} avg_{priority(j)}}{|T|}, \tag{4.1}$$

$$\text{minimize} \quad f_2(\mathbf{x}) = \frac{\sum_{j \in T} |avg_{experience}(j) - \mu|}{|T|}, \tag{4.2}$$

$$\text{subject to} \sum_{j \in T} x_{ij} = 1 \qquad \forall i \in S \tag{4.3}$$

$$n_{min} \leq \sum_{i \in S} x_{ij} \leq n_{max} \qquad \forall j \in T \tag{4.4}$$

$$\sum_{i \in S} x_{ij} \cdot g_i \geq m_{min} \qquad \forall j \in T \tag{4.5}$$

$$\sum_{i \in S} x_{ij} \cdot d_i^{Mac} \geq m_{Mac,min} \qquad \forall j \in T, \tag{4.6}$$

$$\sum_{i \in S} x_{ij} \cdot d_i^{iDevice} \geq m_{iDev,min} \qquad \forall j \in T, \tag{4.7}$$

where:

$$avg_{priority}(j) = \frac{\sum_{i \in S} x_{ij} \cdot P_{ij}}{|j|} \qquad \forall j \in T, \tag{4.8}$$

$$avg_{experience}(j) = \frac{\sum_{i \in S} x_{ij} \cdot r_i}{|j|} \qquad \forall j \in T, \tag{4.9}$$

$$\mu = \frac{\sum_{i \in S} r_i}{|S|}, \tag{4.10}$$

$$g_i = \begin{cases} 1 & \text{if student } i \text{ is a female person} \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

$$r_i = \begin{cases} 1 & \text{if student } i \text{ is } \textit{Expert} \\ 2 & \text{if student } i \text{ is } \textit{Advanced} \\ 3 & \text{if student } i \text{ is } \textit{Intermediate} \\ 4 & \text{if student } i \text{ is } \textit{Novice.} \end{cases} \tag{4.12}$$

The goal is to minimize the mean priority (equation 4.1) and the mean deviation of the experience level (equation 4.2) across all teams.
We introduced a new variable r_i which maps the instructor rating for student i to a number in $\{1, 2, 3, 4\}$. Thus, the smaller the value of r_i is, the 'better'

the instructor rating for student i is. Equations 4.8 and 4.9 denote the mean priority and the mean experience levels of a specific team. μ represents the mean experience of the students.
Solving this MOP global optimization problem means to first find the *global minimum solution set* (i.e. $\mathcal{P}^*$) and then to select one solution from this set. Formally, given the vector function $F : \Omega \subseteq \{0,1\}^s \rightarrow \mathbb{R}^2, \Omega \neq \emptyset$, for $x^* \in \Omega$ the set $\mathcal{PF}^* = f(x_i^*) > (-\infty, ..., -\infty)$ is called the *global minimum* if and only if:

$$\forall \mathbf{x} \in \Omega : f(x_i^*) \preceq f(\mathbf{x}). \tag{4.13}$$

Then, $x_i^*, i = 1, ..., s$ is the *global minimum solution set*, F is the multiple objective function as defined before, and the set Ω is the feasible region [4].

4.2 NSGA-II Algorithm

In this section we describe the key concepts of the NSGA-II algorithm and how it has been applied to solve the team assignment problem. Detailed information about the algorithm and also a comparison (based on simulation results) to similar approaches can be found in the original paper in which NSGA-II was introduced [7].
We chose the NSGA-II algorithm to solve the team assignment problem for various reasons. Being an iterative algorithm, NSGA-II (and genetic algorithms in general) is well-suited for problems whose optimal solution(s) cannot be found in a single try and the team allocation is a problem of such type. Also, the way in which new candidate solutions are created - by combining the fittest available solutions and additionally performing mutations - ensures both the diversity of the assignment solutions and their convergence to an (ideal, possibly unknown) optimal result. Moreover, the elitist approach of NSGA-II (which will be discussed in the following) ensures that good solutions are not lost when performing e.g. the crossover operation, but are passed down the generations instead.
Furthermore, the NSGA-II algorithm works well with our data, as we will see in a subsequent paragraph in which we present the chromosome structure. Thus, the allocated team for each student can be easily "read" from the chromosome corresponding to a candidate solution, given its fixed structure which is maintained through the generations (each student will have the same bit positions in the chromosomes created in each iteration). Nevertheless, the good computational complexity of NSGA-II, in comparison to other genetic algorithms, represents an important reason for selecting and implementing this approach to solve the team assignment problem.

NSGA-II works by iteratively building populations of a fixed size (i.e. number of chromosomes). As its name suggests, the algorithm uses a *fast nondominated sorting approach* for computing the nondominated fronts, which leads to a significant performance improvement compared to the original NSGA (NSGA-II has $O(MN^2)$ computational complexity, while NSGA has $O(MN^3)$, where M represents the number of objectives and N denotes the population size). The first nondominated front contains all solutions that are not dominated by any solution, the second domination front includes the solutions wich are dominated by exactly one solution and so on. A fast nondominated sorting approach means that, instead of naively comparing all solutions with the others in order to identify the first nondominated front and repeating this process to compute all fronts, the algorithm calculates for each solution two attributes: the domination count, i.e. the number of solutions which dominate it, and the set of solutions that it dominates.
We think that this performance improvement is important for solving the team assignment problem, as our experiments (see Chapter 5) have shown that a large population size N and hence computing multiple potential solutions moght lead to a better final assignment result.
For ensuring the diversity of the solutions and implicitly computing a uniformly spread-out Pareto optimal front, NSGA-II implements a *crowding-comparison approach* (pseudocode from Figure 4.1).

```
crowding-distance-assignment($\mathcal{I}$)
$l = |\mathcal{I}|$                                          number of solutions in $\mathcal{I}$
for each $i$, set $\mathcal{I}[i]_{\text{distance}} = 0$      initialize distance
for each objective $m$
  $\mathcal{I} = \text{sort}(\mathcal{I}, m)$                 sort using each objective value
  $\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$   so that boundary points are always selected
  for $i = 2$ to $(l - 1)$                                    for all other points
    $\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[i]_{\text{distance}} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$
```

Figure 4.1: Pseudocode of the *crowding-distance-assignment procedure* (Source: [7]).

Each solution is assigned a crowding distance, which denotes the proximity of the solution to others (i.e. how “similar” the solutions are). Thus, the higher the crowding distance value is, the “more crowded” the corresponding solution is (and the more “similar” to other solutions it is). Also, between two solutions that belong to different nondominated fronts, the one from the front having a lower index is preferred (e.g. if one solution belongs to the third nondominated front and thus is dominated by two solutions and the another solution belongs to the fourth nondominated front, we choose the

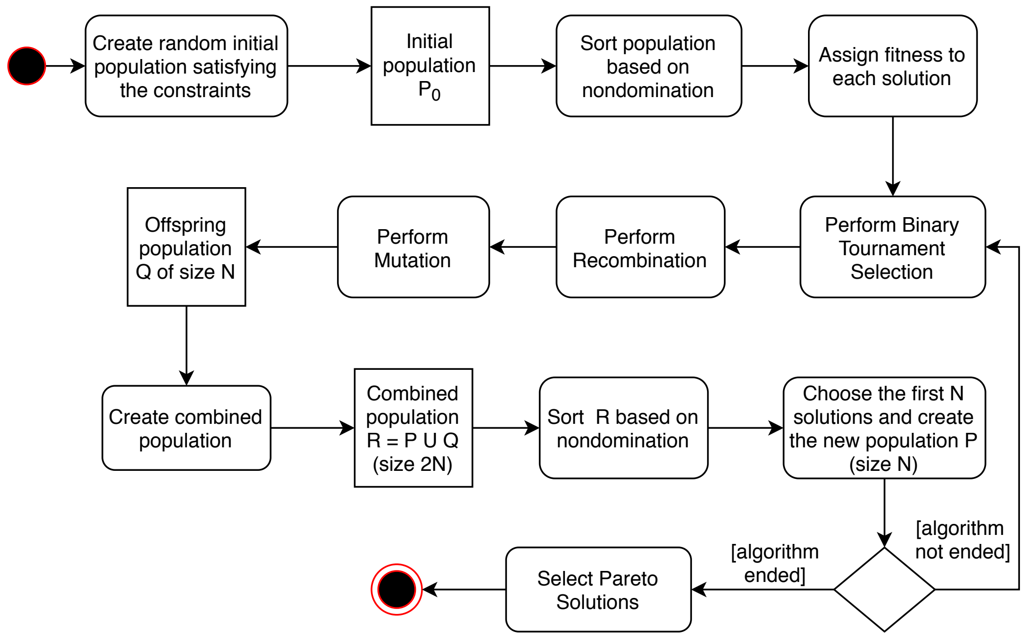

Create random initial population satisfying the constraints
Initial population P_0
Sort population based on nondomination
Assign fitness to each solution
Perform Binary Tournament Selection
Perform Recombination
Perform Mutation
Offspring population Q of size N
Create combined population
Combined population R = P U Q (size 2N)
Sort R based on nondomination
Choose the first N solutions and create the new population P (size N)
[algorithm not ended]
[algorithm ended]
Select Pareto Solutions

Figure 4.2: NSGA-II algorithm steps as implemented to solve the team assignment problem (UML Activity Diagram).

first solution). Otherwise, if both solutions belong to the same front, the one with the smaller crowding distance is chosen.

Figure 4.2 illustrates the steps of the NSGA-II algorithm as it was applied to solve the team composition problem. A pseudocode of the algorithm can also be found below. At first, a random initial population that satisfies the global constraints (see Section 4.1) is created. This is a modification to the original algorithm, which does not directly deal with constrained optimization problems and starts with a "pure" random population.

Next, the initial population is sorted based on nondomination (pseudocode from Figure 4.3) and each solution is assigned a fitness value, which equals its nondomination level (1 is the best level, 2 is the second best level etc. [7]). The following three steps (Perform Binary Tournament Selection, Perform Recombination, Perform Mutation) are realized in order to create a new child population. Before discussing these steps, let us briefly introduce how we modeled an assignment solution in our NSGA-II implementation.

A fragment of the structure of a chromosome encoding a solution can be seen in Figure 4.4. The chromosome is composed of genes which are represented as an array of bits having a fixed length ($|S| \cdot |T|$). The first $|T|$ bits relate to the first student, the next $|T|$ bits refer to the second student and so on. In the figure below, the bits at indexes 2 and 8 are 1, which means that the

```
fast-non-dominated-sort($P$)
for each $p \in P$
  $S_p = \emptyset$
  $n_p = 0$
  for each $q \in P$
    if $(p \prec q)$ then                     If $p$ dominates $q$
      $S_p = S_p \cup \{q\}$                   Add $q$ to the set of solutions dominated by $p$
    else if $(q \prec p)$ then
      $n_p = n_p + 1$                          Increment the domination counter of $p$
  if $n_p = 0$ then                            $p$ belongs to the first front
    $p_{\text{rank}} = 1$
    $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$
$i = 1$                                        Initialize the front counter
while $\mathcal{F}_i \neq \emptyset$
  $Q = \emptyset$                              Used to store the members of the next front
  for each $p \in \mathcal{F}_i$
    for each $q \in S_p$
      $n_q = n_q - 1$
      if $n_q = 0$ then                        $q$ belongs to the next front
        $q_{\text{rank}} = i + 1$
        $Q = Q \cup \{q\}$
  $i = i + 1$
  $\mathcal{F}_i = Q$
```

Figure 4.3: Pseudocode of the *fast-non-dominated-sort* procedure (Source: [7]).

first student was assigned to the third team, while the second student was allocated to the first one. Because a student must be assigned to exactly one team, only one bit out of the total $|T|$ bits that correspond to a student must be 1.

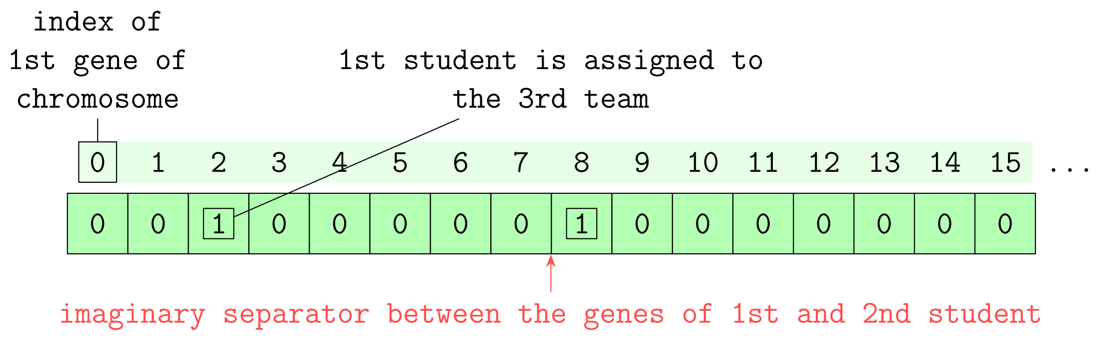


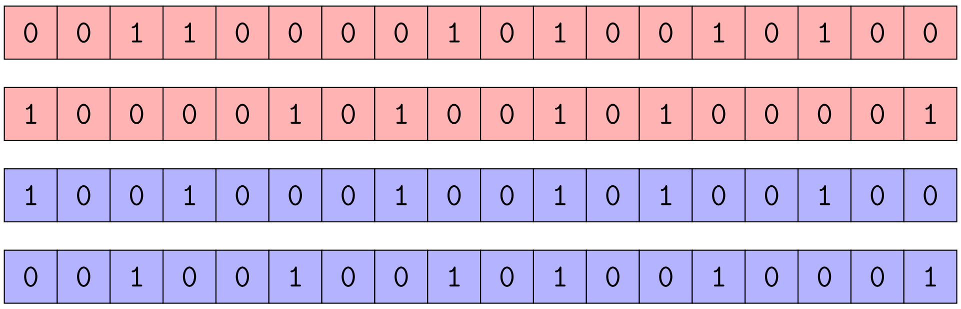
Figure 4.4: Fragment of the structure of a chromosome representing the assignment of the first 2 students across 8 teams.

A binary tournament selection means that, in multiple runs, out of two solu-

```
$R_t = P_t \cup Q_t$                                   combine parent and offspring population
$\mathcal{F} = \texttt{fast-non-dominated-sort}(R_t)$  $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$, all nondominated fronts of $R_t$
$P_{t+1} = \emptyset$ and $i = 1$
until $|P_{t+1}| + |\mathcal{F}_i| \leq N$             until the parent population is filled
    $\texttt{crowding-distance-assignment}(\mathcal{F}_i)$   calculate crowding-distance in $\mathcal{F}_i$
    $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$             include $i$th nondominated front in the parent pop
    $i = i + 1$                                        check the next front for inclusion
Sort$(\mathcal{F}_i, \prec_n)$                         sort in descending order using $\prec_n$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - |P_{t+1}|)]$   choose the first $(N - |P_{t+1}|)$ elements of $\mathcal{F}_i$
$Q_{t+1} = \texttt{make-new-pop}(P_{t+1})$             use selection, crossover and mutation to create
                                                         a new population $Q_{t+1}$
$t = t + 1$                                            increment the generation counter
```

Figure 4.6: Procedure for creating a new generation of chromosomes (Source: [7]).

tions, the fittest one is chosen for crossover - two tournaments are run in order to select the two parent solutions involved in a crossover. The recombination or the crossover is the moment when the parent solutions randomly exchange information and two new solutions are obtained as a result. Figure 4.5 illustrates two parent chromosomes (red) and the resulting children chromosomes (blue) after the crossover operation was performed. The chromosomes encode a solution for the allocation of six students over three teams.



0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	1	0	0
1	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0	0	1
1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	0	1

Figure 4.5: Crossover operation: the red-colored chromosomes are the parents and the blue-colored chromosomes are the resulting offsprings.

With a probability defined before running the algorithm, a mutation can occur in the chromosome created in the crossover step. We defined a mutation as the process in which exactly two randomly selected students swap their teams. In figure 4.4, the new set bits would be at indexes 0 and 10 if the first two students were involved in a mutation.
As a result of the steps presented above, an offspring population of the same size as the initial population (or the previous one: P_t) is created. By com-

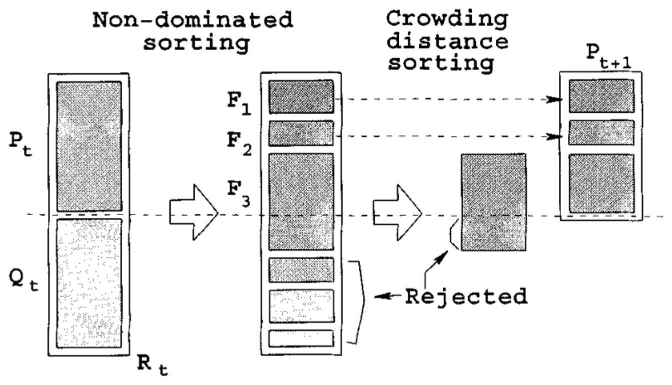
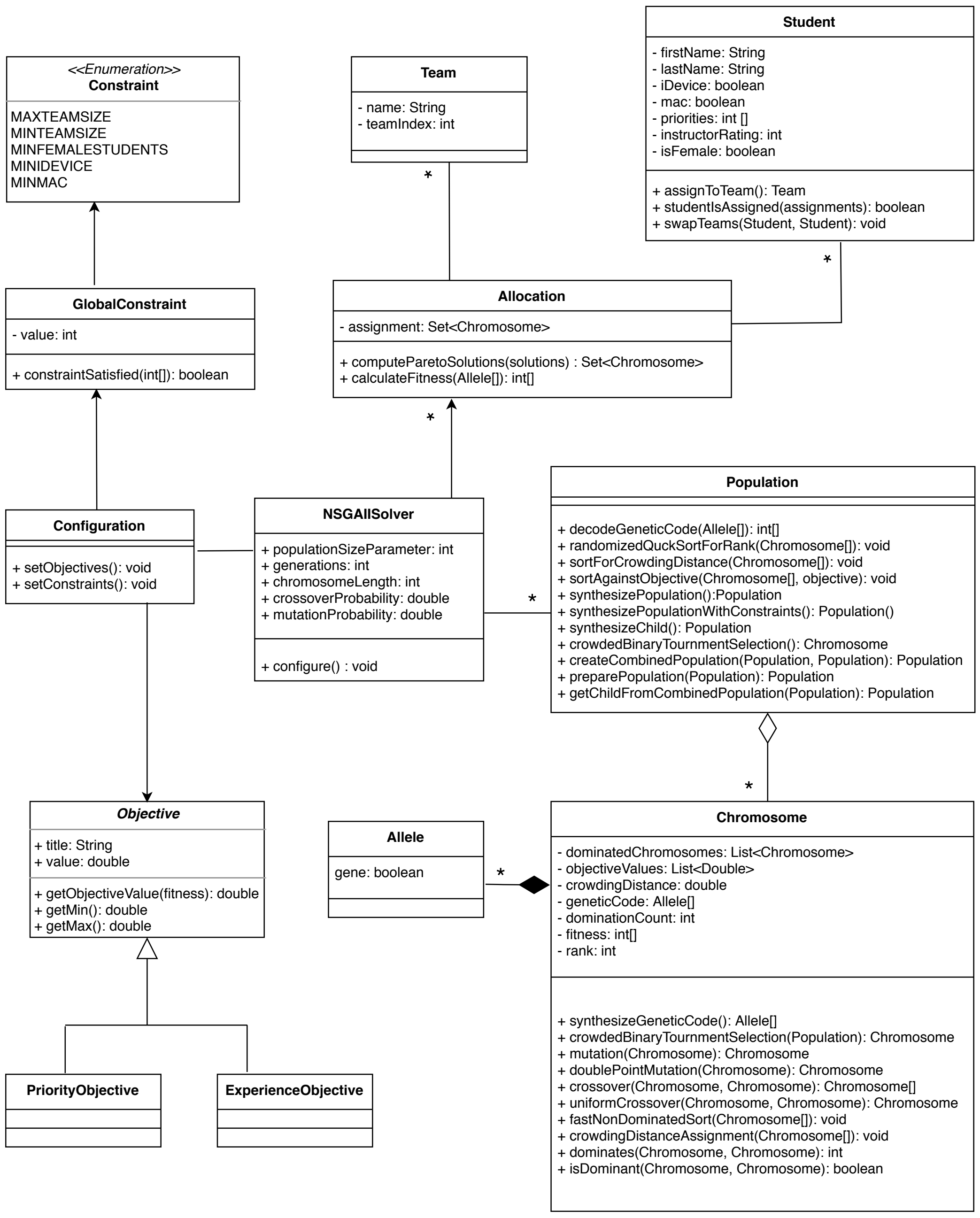


Figure 4.7: NSGA-II procedure (Source: [7]). F_1, F_3, F_3 denote the first three nondominated fronts.

bining the new and the previous populations, a double sized one is obtained, which is then sorted (apply fast-non-dominating-sort() procedure from Figure 4.3) and its best N solutions are chosen to form the population P_{t+1}, as shown in Figure 4.7. The steps previously described are illustrated in the pseudocode from Figure 4.6 and in Figure 4.7 the whole NSGA-II procedure can be visualized.

The algorithm stops when a fixed number of iterations have been completed. Being an a posteriori algorithm, NSGA-II involves the decision-maker only after the Pareto solutions have been computed. Hence, once the algorithm has ended, the instructor can choose one assignment solution from the set of calculated Pareto solutions.

After briefly discussing its high suitability for our problem, we should not ignore the possible drawbacks of the NSGA-II algorithm. Thus, it is important to mention that there is no guarantee that NSGA-II finds the true Pareto solutions (the MOEA algorithms are, in general, not certain to be optimal). Given that the optimal solution(s) of our team assignment problem cannot be computed using a simple brute-force approach (due to the large number of variables that are involved), there is no method for evaluating the optimality of NSGA-II without using a more difficult, optimal algorithm. Moreover, the increased complexity of the algorithm, compared to e.g. a priori methods, might constitute another drawback of NSGA-II.



<<Enumeration>>
Constraint
MAXTEAMSIZE
MINTEAMSIZE
MINFEMALESTUDENTS
MINIDEVICE
MINMAC
Team
- name: String
- teamIndex: int
Student
- firstName: String
- lastName: String
- iDevice: boolean
- mac: boolean
- priorities: int []
- instructorRating: int
- isFemale: boolean
+ assignToTeam(): Team
+ studentIsAssigned(assignments): boolean
+ swapTeams(Student, Student): void
GlobalConstraint
- value: int
+ constraintSatisfied(int[]): boolean
Allocation
- assignment: Set<Chromosome>
+ computeParetoSolutions(solutions) : Set<Chromosome>
+ calculateFitness(Allele[]): int[]
Configuration
+ setObjectives(): void
+ setConstraints(): void
NSGAIISolver
+ populationSizeParameter: int
+ generations: int
+ chromosomeLength: int
+ crossoverProbability: double
+ mutationProbability: double
+ configure() : void
Population
+ decodeGeneticCode(Allele[]): int[]
+ randomizedQuckSortForRank(Chromosome[]): void
+ sortForCrowdingDistance(Chromosome[]): void
+ sortAgainstObjective(Chromosome[], objective): void
+ synthesizePopulation():Population
+ synthesizePopulationWithConstraints(): Population()
+ synthesizeChild(): Population
+ crowdedBinaryTournmentSelection(): Chromosome
+ createCombinedPopulation(Population, Population): Population
+ preparePopulation(Population): Population
+ getChildFromCombinedPopulation(Population): Population
Objective
+ title: String
+ value: double
+ getObjectiveValue(fitness): double
+ getMin(): double
+ getMax(): double
Allele
gene: boolean
Chromosome
- dominatedChromosomes: List<Chromosome>
- objectiveValues: List<Double>
- crowdingDistance: double
- geneticCode: Allele[]
- dominationCount: int
- fitness: int[]
- rank: int
+ synthesizeGeneticCode(): Allele[]
+ crowdedBinaryTournmentSelection(Population): Chromosome
+ mutation(Chromosome): Chromosome
+ doublePointMutation(Chromosome): Chromosome
+ crossover(Chromosome, Chromosome): Chromosome[]
+ uniformCrossover(Chromosome, Chromosome): Chromosome
+ fastNonDominatedSort(Chromosome[]): void
+ crowdingDistanceAssignment(Chromosome[]): void
+ dominates(Chromosome, Chromosome): int
+ isDominant(Chromosome, Chromosome): boolean
PriorityObjective
ExperienceObjective
*
*
*
*
*
*

Figure 4.8: NSGA-II algorithm for solving the team assignment problem - object design (UML Class Diagram)

4.3 MOPSO Algorithm

In this section we first introduce the PSO (Particle Swarm Optimization) algorithm and afterwards we concentrate on the MOPSO algorithm, which is the multi-objective version of PSO and has been implemented to solve the team assignment problem.
We considered that this algorithm is suitable for our problem for two main reasons. Our literature review revealed that MOPSO, given its efficiency and moderate implementation difficulty, is one of the most used algorithms for solving the assignment problem, in its various forms [23], [22]. Although our team assignment problem is very different compared to the traditional assignment problem (or its variations covered in [23], [22]), primarly, due to the existence of the constraints, we considered that MOPSO may compute good solutions for it (with respect to the metrics discussed in Section 5.2). Moreover, in its search for new solutions, MOPSO keeps track of the feasible solutions that have already been found, as we will notice in this section when we discuss the impact of the coefficients c_1 and c_2. This fact is extremely important in our context, given that better team assignment solutions are usually derived from already existing ones, as described in [8] (where the manual approach is presented). These facts being said, let us now discuss the particle swarm optimization algorithm.
PSO works by creating a random initial population of particles (i.e. solutions to the optimization problem to be solved) and iteratively moving these particles around the search space. Each particle is represented as a vector and its position is updated in every iteration by considering both its *personal best* (the best objective function value that this particle has ever computed) and the *global best* (the best objective function value computed by all particles). Thus, the changes in the particle vector are determined by the *velocity*, which is computed for each dimension of the vector using equation 4.14 and is composed of three parts, as suggested with colors.
Let us now analyze equation 4.14. $X[i]$ is the result computed by the current particle for the *ith* dimension in the previous iteration. The parameter w' is called the *inertial weight* and indicates the degree to which the new particle position is influenced by the old position. Its value must always be set smaller than 1, given the accumulative nature of the updating equation (we want to prevent the velocity value to grow unbounded, otherwise the particle goes outside the search space) [19].

$$V[i] = w' \cdot V_{old}[i] + c_1 \cdot r_1 \cdot (PBEST[i] - X[i]) + c_2 \cdot r_2 \cdot (GBEST[i] - X[i]) \tag{4.14}$$

As we notice, PSO takes into consideration the 'past' when computing the current position, this fact representing a difference to the NSGA-II algorithm, which is 'memoryless'. Furthermore, r_1 and r_2 are two random numbers in $[0, 1]$, while c_1 and c_2 are coefficients to tune the impact of the *cognitive component* (red) and of the *social component* (blue), respectively. The cognitive component refers to the degree to which each particle maintains its 'autonomy' when searching for better solutions. On the other hand, the social component relates to the 'tendency' of a particle to guide its search by also taking into account the other particles' best values found so far. Thus, at a general level, if the value of the coefficient c_1 is greater than the value of c_2, then the particle predominantly searches around its personal best and is less influenced by the solutions found by the rest of the particles.
We denote with $PBEST[i]$ the personal best value found by the current particle for the *ith* dimension, while $GBEST[i]$ is the global best value that the entire population has computed for the same dimension. Hence, in each iteration, the new velocity is updated to a weighted sum between the old velocity and the distances of $X[i]$ to the personal and global best values. The values chosen for the coefficients c_1 and c_2, as well as for the inertial weight influence the exploitation and exploration of the search space (see Chapter 5).
The new particle value for the *ith* dimension is calculated using equation 4.15. The operation $\oplus$ can be differently defined in each algorithm implementation (e.g. can be the addition).

$$X_{new}[i] = X[i]_{old} \oplus V[i] \tag{4.15}$$

Let us now concentrate on the MOPSO algorithm for solving the team allocation problem. The pseudocode of the algorithm, as we have implemented it, is shown below, and a UML activity diagram illustrating the algorithm steps is presented in Figure 4.9.
The particle structure is identical to the chromosome structure implemented in the NSGA-II algorithm. Each particle has $|S| \cdot |T|$ dimensions. The algorithm starts by creating a random set of particles that satisfy the defined constraints. Then, for each particle, the velocities are randomly initialized with values in [0,1) (one value for each dimension). The personal best value of a particle is the particle itself at the beginning of the algorithm. Also, the global best particle is found after searching over all personal best values and comparing the fitness values of the particles.

We calculated the fitness using equation 4.17. λ_{min} and λ_{max} are the minimum and maximum values of the experience objective function without considering the priority objective. θ_{min} and θ_{max} are defined analogously.
We will now describe one iteration of the algorithm. For the next particle which was not processed in the current iteration yet, the velocity is computed using equation 4.14. Then, for each student, the following operations are repeated until an assignment has been found: a team is randomly chosen and afterwards the sigmoid function indicated in equation 4.16 is evaluated for the dimension of the particle corresponding to the current student and the chosen team. We apply the sigmoid function in order to ensure than the positions of the renewed particles are 0 or 1 [23].

$$S[i] = \frac{1}{1 + exp(-V[i])} \tag{4.16}$$

$$Fitness = w_1 \cdot (-f_{experience})/(\lambda_{min} - \lambda_{max}) + w_2 \cdot (-f_{priority})/(\theta_{min} - \theta_{max}) \tag{4.17}$$

If the sigmoid function value is greater than a random number, then we check whether the chosen team does not satisfy the *minSize* constraint (the size of the team is smaller than the required minimum size). If this is the case, then we assign the student to this team. Otherwise, we search for a team that does not satisfy this constraint and allocate the student to it. If such a team does not exist, then we assign the student to a random team which has size smaller than the maximum allowed size (we always have the guaranty that such team exists).
The *Judge-and-Repair Strategy* that we have just described is needed for ensuring that the MOPSO algorithm computes solutions that satisfy the defined constraints. The team size constraint is, intuitively, the hardest to be satisfied (among all constraints presented in Section 4.1) in a stochastic algorithm like MOPSO. Therefore, this strategy aims to ensure that the team size constraint is fulfilled when each particle is updated. This determines a higher convergence rate and exploration of the search space.
Once all students have been allocated to a team, we check whether the new particle satisfies the constraints and, if this is the case, we compare the fitness value of the new particle with the fitness value of the personal best of the current particle. If the new particle is ‘fitter’ than the personal best, then we update $PBEST$ and check whether the global best can be improved. The algorithm ends when all iterations have been completed.

Algorithm 4.1 MOPSO Algorithm for team allocation

```
Result: an assignment solution
Create random particles satisfying the constraints;
for all particles do
    for all dimensions do
        Randomly initialize velocity in [0,1)
    end
    PBEST = particle
end
Find GBEST
while maxNumber of iterations not achieved do
    for all particles do
        Update velocity acc. to equation 4.14
        for all students do
            while student not assigned to any team do
                Randomly choose a team T
                Evaluate sigmoid function acc. to equation 4.16
                if sigmoid function evaluation > random number then
                    if teamSizeConstraint for T satisfied then
                        Assign student to T
                    else
                        ;                          ▷ Judge-and-repair strategy
                        Randomly choose team that satisfies
                        teamSizeConstraint and assign student to it
                    end
            end
        end
        if particle satisfies all constraints then
            if PBEST ⪯ current particle then
                PBEST = current particle
            if GBEST ⪯ PBEST then
                GBEST = PBEST
    end
end
Return GBEST
```

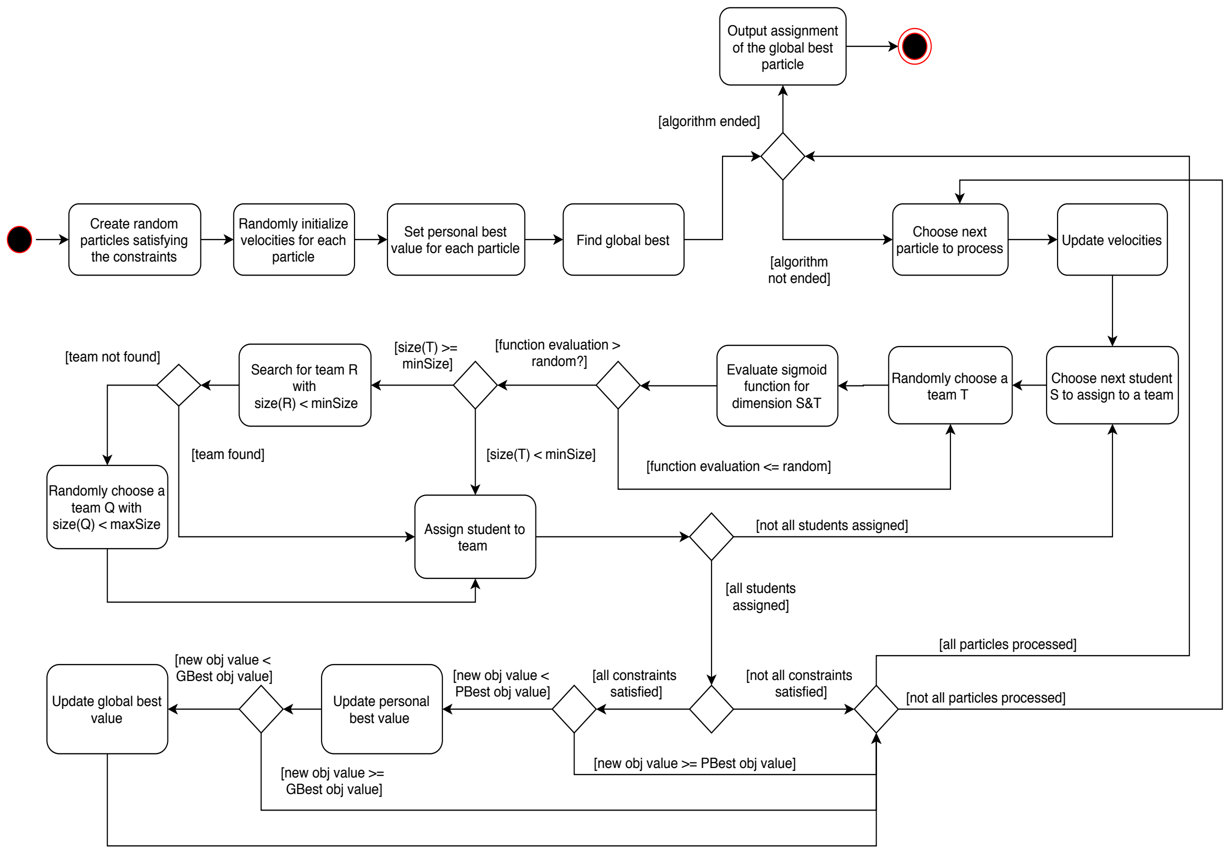

Output assignment of the global best particle
[algorithm ended]
Create random particles satisfying the constraints
Randomly initialize velocities for each particle
Set personal best value for each particle
Find global best
[algorithm not ended]
Choose next particle to process
Update velocities
[team not found]
Search for team R with size(R) < minSize
[size(T) >= minSize]
[function evaluation > random?]
Evaluate sigmoid function for dimension S&T
Randomly choose a team T
Choose next student S to assign to a team
[team found]
[size(T) < minSize]
[function evaluation <= random]
Randomly choose a team Q with size(Q) < maxSize
Assign student to team
[not all students assigned]
[all students assigned]
[all particles processed]
[new obj value < GBest obj value]
Update global best value
Update personal best value
[new obj value < PBest obj value]
[all constraints satisfied]
[not all constraints satisfied]
[not all particles processed]
[new obj value >= PBest obj value]
[new obj value >= GBest obj value]

Figure 4.9: MOPSO algorithm steps as implemented to solve the team assignment problem (UML Activity Diagram).

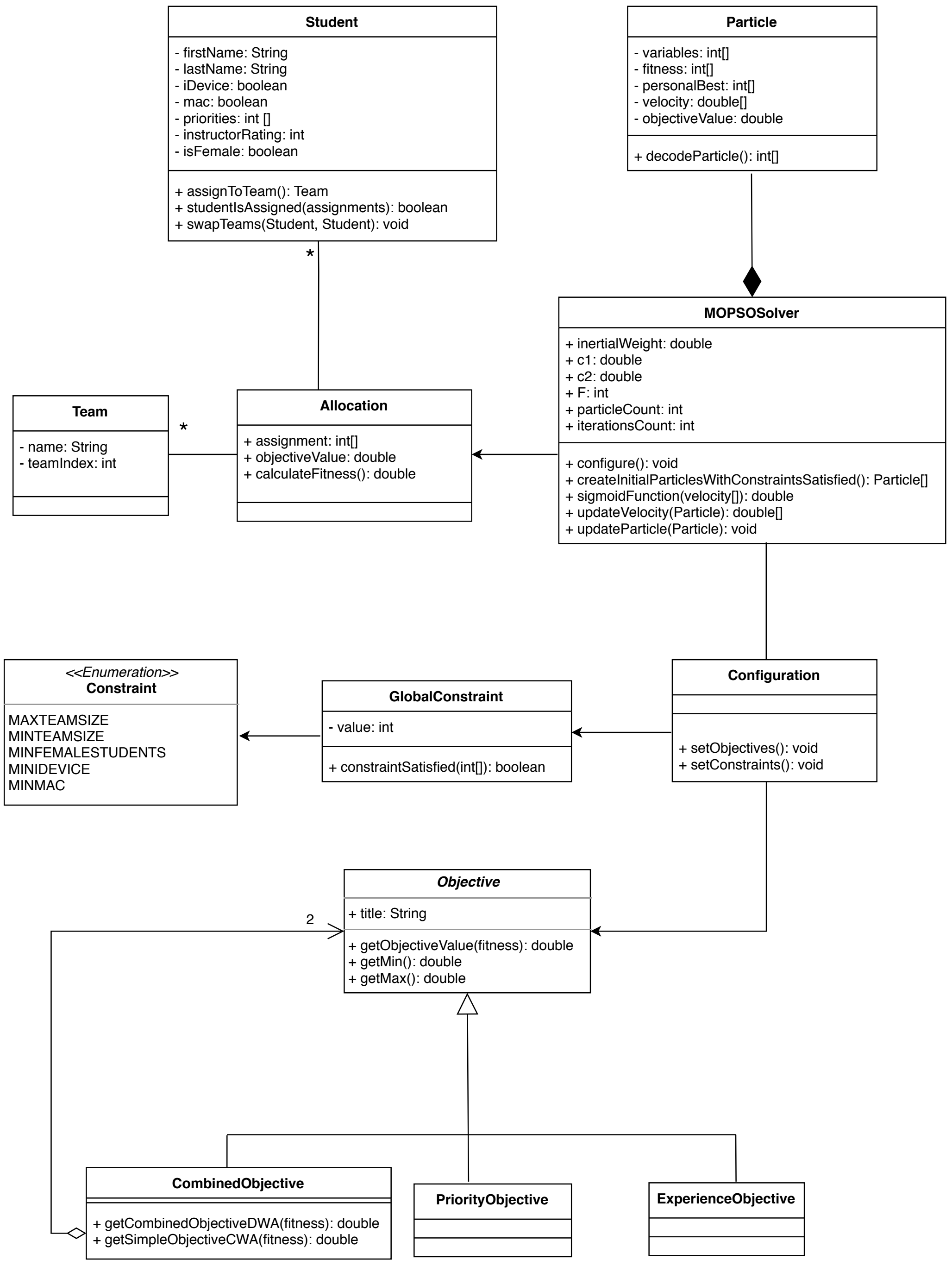


Figure 4.10: MOPSO algorithm for solving the team assignment problem - object design (UML Class Diagram).

4.4 Tabu Search Algorithm

In this section we present the general concept of Tabu Search and how it has been applied to solve the team assignment problem. Details about this method can be found in the paper in which it was introduced [13].

We selected Tabu Search to solve the team assignment problem because it is a local-search method and we want to evaluate whether such strategy is more promising that e.g. population-based algorithms (such as MOPSO and NSGA-II), with respect to the quality of the computed assignment solution. Moreover, this algorithm has similarities with a random search and our aim is to investigate whether good assignment solutions can be found by systematically exploiting the search space.

Tabu Search is a local-search metaheuristic approach for solving complex optimization problems. Is incorporates *adaptive memory* and *responsive exploration* [13] in order to find the optimum of a given function. Thus, TS uses *adaptive memory*, meaning that it implements a “procedure that is capable of searching the solution space economically and effectively” [13], as opposed to memoryless approaches which do not take into consideration the past solutions and rely on a semirandom process (e.g. a form of sampling) when searching for better solutions. An example of a memoryless approach is the NSGA-II algorithm.

The algorithm attempts to find the optimum of a function by starting with a randomly generated solution and iteratively refining it. For the solution found in the current iteration, a set of neighbors, which are solutions that differ from the original one only in some points, is calculated. Then, the fittest neighbor is selected and the process of generating its neighbors and choosing the fittest one among them to be refined in the next iteration is repeated until a stopping criterion is met (e.g. after a defined number of iterations is achieved). The adaptive memory of the algorithm is ‘represented’ by a Tabu list, which contains solutions that were already ‘explored’ and refined. The list is updated in each iteration and aims to prevent the algorithm from trying to refine solutions that were already updated in previous iterations.

The pseudocode of the algorithm as it was implemented to solve the team assignment problem is shown below. Figure 4.11 illustrates the steps of TS in an UML activity diagram and Figure 4.12 represents the object design of our Tabu Search implementation.

We defined the *neighborhood* of a solution as the set of solutions that are obtained from the original one by swapping the teams of a predefined number of students (e.g. 50). The Tabu list contains the best assignment solutions (w.r.t. the combined objective value) found by the algorithm. In each iteration, the neighborhood of the current solution is computed. For each solution in the neighborhood, we check whether it satisfies all constraints and, if so, we see whether it is already included in the Tabu list. If the solution is not in the Tabu list, we verify whether its combined objective value is smaller than the value corresponding to the current best assignment solution and, if so, we replace the current best solution with the newly found one. This solution is also added in the Tabu list, in order to prevent the algorithm from refining it in future iterations. The algorithm ends once a predefined number of iterations is achieved and the final assignment solution is the best solution (in the pseudocode, this is the *bestAssignment*).

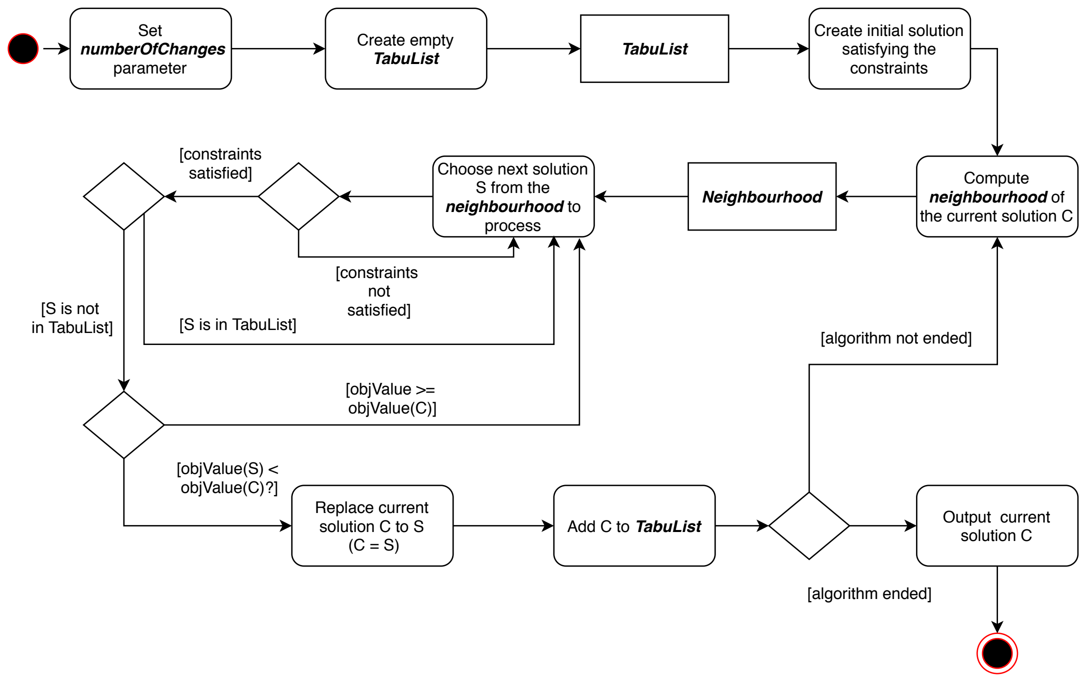

Set numberOfChanges parameter
Create empty TabuList
TabuList
Create initial solution satisfying the constraints
Compute neighbourhood of the current solution C
Neighbourhood
Choose next solution S from the neighbourhood to process
[constraints satisfied]
[constraints not satisfied]
[S is not in TabuList]
[S is in TabuList]
[objValue >= objValue(C)]
[objValue(S) < objValue(C)?]
Replace current solution C to S (C = S)
Add C to TabuList
[algorithm not ended]
[algorithm ended]
Output current solution C

Figure 4.11: Tabu Search Algorithm steps as implemented to solve the team assignment problem (UML Activity Diagram).

We opted for a TS implementation without a bounded Tabu list size. The motivation behind our choice is that we observed that the Tabu list does not grow considerably (in our simulations, the maximum size was ≈ 150) and, therefore, it does not make sense to restrict the 'memory' of the algorithm and thus to 'risk' refining solutions which were already refined in previous iterations.

Algorithm 4.2 Tabu Search Algorithm for team allocation

```
Result: an assignment solution
Create a random assignment solution bestAssignment satisfying the
  constraints
tabuList = ∅
tabuList.add(bestAssignment)
Set neighborhoodSize
neighborhood = ∅
while maxNumber of iterations not achieved do
    countneighbors = 0
    swapStudentsList = ∅
    while countneighbors != neighborhoodSize do
        Randomly choose a student s
        if S is not in swapStudentsList then
            swapStudentsList.add(s)
        countneighbors ++
    end
    for all s1 in swapStudentsList do
        Randomly choose a student s2
        ;    ▷ swap the teams of s1 and s2 in the current solution
        solution = bestAssignment.swap(team(s1), team(s2));
        neighborhood.add(solution)
    end
    for all solutions in neighborhood do
        if solution satisfies the constraints then
            if solution is not in tabuList then
                if fitness(solution) < fitness(bestAssignment) then
                    tabuList.add(solution)
                    bestAssignment = solution
    end
end
Return bestAssignment;
```

The evaluation results of the Tabu Search algorithm are presented in Chapter 5. As we will see, one of the greatest advantages of Tabu Search is its good execution time, while the objective values of the computed assignment solution tend to be lower than the values corresponding to the solutions generated with population-based approaches.

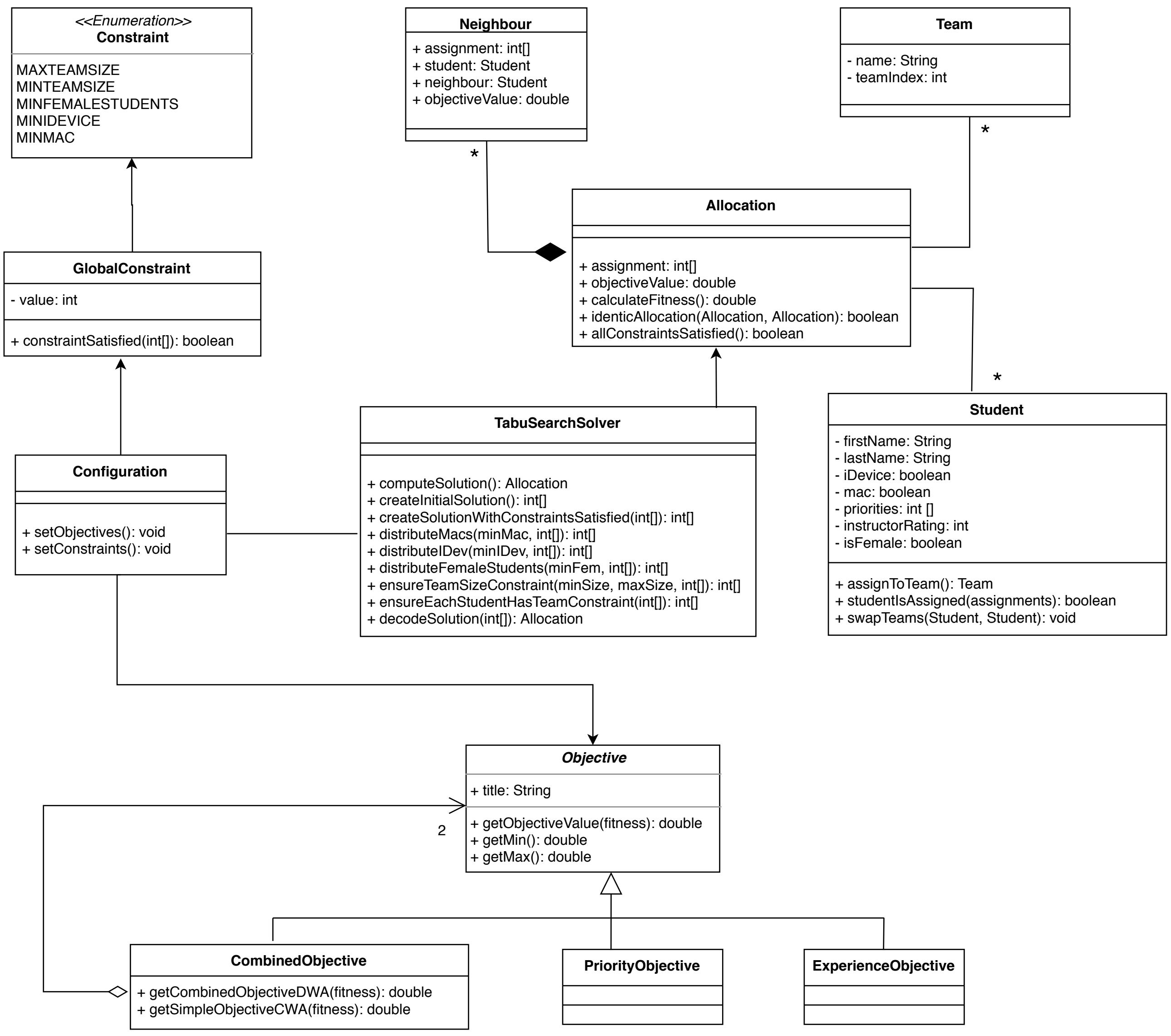

<<Enumeration>>
Constraint
MAXTEAMSIZE
MINTEAMSIZE
MINFEMALESTUDENTS
MINIDEVICE
MINMAC
Neighbour
+ assignment: int[]
+ student: Student
+ neighbour: Student
+ objectiveValue: double
Team
- name: String
- teamIndex: int
*
*
Allocation
+ assignment: int[]
+ objectiveValue: double
+ calculateFitness(): double
+ identicAllocation(Allocation, Allocation): boolean
+ allConstraintsSatisfied(): boolean
GlobalConstraint
- value: int
+ constraintSatisfied(int[]): boolean
*
TabuSearchSolver
+ computeSolution(): Allocation
+ createInitialSolution(): int[]
+ createSolutionWithConstraintsSatisfied(int[]): int[]
+ distributeMacs(minMac, int[]): int[]
+ distributeIDev(minIDev, int[]): int[]
+ distributeFemaleStudents(minFem, int[]): int[]
+ ensureTeamSizeConstraint(minSize, maxSize, int[]): int[]
+ ensureEachStudentHasTeamConstraint(int[]): int[]
+ decodeSolution(int[]): Allocation
Configuration
+ setObjectives(): void
+ setConstraints(): void
Student
- firstName: String
- lastName: String
- iDevice: boolean
- mac: boolean
- priorities: int []
- instructorRating: int
- isFemale: boolean
+ assignToTeam(): Team
+ studentIsAssigned(assignments): boolean
+ swapTeams(Student, Student): void
Objective
+ title: String
+ getObjectiveValue(fitness): double
+ getMin(): double
+ getMax(): double
2
CombinedObjective
+ getCombinedObjectiveDWA(fitness): double
+ getSimpleObjectiveCWA(fitness): double
PriorityObjective
ExperienceObjective

Figure 4.12: Tabu Search algorithm for solving the team assignment problem - object design (UML Class Diagram).

Chapter 5

Evaluation

In this chapter we present the evaluation of the MOPSO, NSGA-II and Tabu Search algorithms. In Section 5.1 we describe the method we followed in this phase of our project, in Section 5.2 we present the evaluation criteria and in Section 5.3 we discuss the test data we used. In Sections 5.4, 5.5 and 5.6 we answer the research questions which we introduce in Section 5.1 and in Section 5.7 we discuss two ideas that could be implemented in a multi-objective approach for solving the team assignment problem in the iPraktikum.

5.1 Method

We conduct a controlled experiment in order to evaluate the applicability and efficiency of the Tabu Search, MOPSO and NSGA-II algorithms for solving the team assignment problem in the iPraktikum. We study the following research questions:

RQ1: What is the optimal parameter configuration for each algorithm?

RQ2: What are the strengths and weaknesses of each algorithm compared to the other evaluated approaches?

RQ3: What are the strengths and weaknesses of a multi-objective optimization approach over the existing single-objective optimization approach for the team assignment problem?

The evaluation phase consisted of three steps, each focusing on one research question. We adopted an agile evaluation approach, meaning that every step

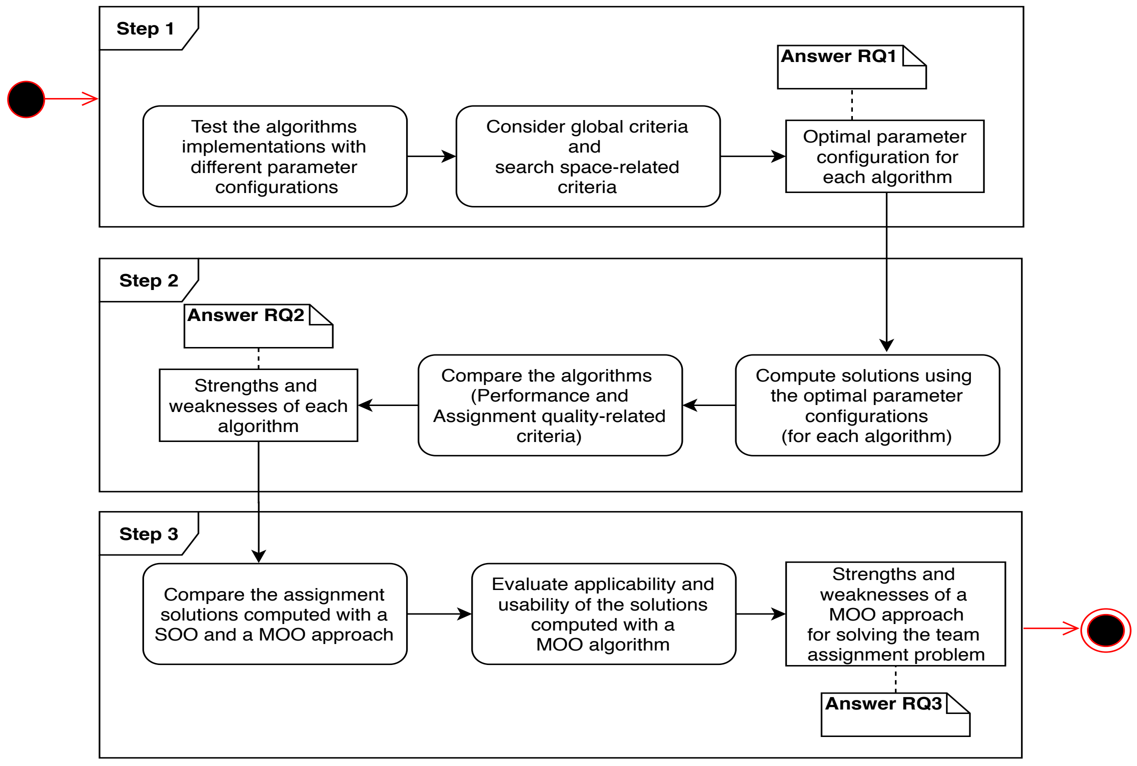

Step 1
Answer RQ1
Test the algorithms implementations with different parameter configurations
Consider global criteria and search space-related criteria
Optimal parameter configuration for each algorithm
Step 2
Answer RQ2
Strengths and weaknesses of each algorithm
Compare the algorithms (Performance and Assignment quality-related criteria)
Compute solutions using the optimal parameter configurations (for each algorithm)
Step 3
Compare the assignment solutions computed with a SOO and a MOO approach
Evaluate applicability and usability of the solutions computed with a MOO algorithm
Strengths and weaknesses of a MOO approach for solving the team assignment problem
Answer RQ3

Figure 5.1: Evaluation steps (UML Activity Diagram).

was based on the results obtained in the previous one and we did not create any preliminary hypotheses. Figure 5.1 illustrates the steps we followed to evaluate the algorithms. In the first step, we tested each algorithm with the same student data and global constraints, but in different parameter settings (the parameters are algorithm-specific). The purpose of this phase was to find an optimal parameter configuration for each algorithm, such that it computes an optimal team assignment solution. After we found such configurations, we compared the quality of the assignment solutions calculated by different algorithms and we also evaluated the three approaches considering different performance criteria (step 2). In the third step, we compared a solution generated by one of the three multi-objective algorithms with the assignment computed by the single-objective approach that already exists in TEASE. Moreover, we drew conclusions related to the applicability of a multi-objective approach for solving the iPraktikum team assignment problem.

Based on the *Goal-Question-Metric Model* [2], we derived metrics that we used for evaluating the assignment solutions computed with different multi-objective algorithms (RQ1 and RQ2). The evaluation criteria are described in the following section. Each research question is answered in its separate section (Sections 5.4, 5.5, 5.6).

5.2 Evaluation Criteria

In this section, we present the criteria we considered for the evaluation of the Tabu Search, MOPSO and NSGA-II algorithms. We clustered the evaluation criteria into three classes, as the taxonomy shown in figure 5.2 illustrates. For studying each research question we used specific criteria that are indicated by the labels in the taxonomy.

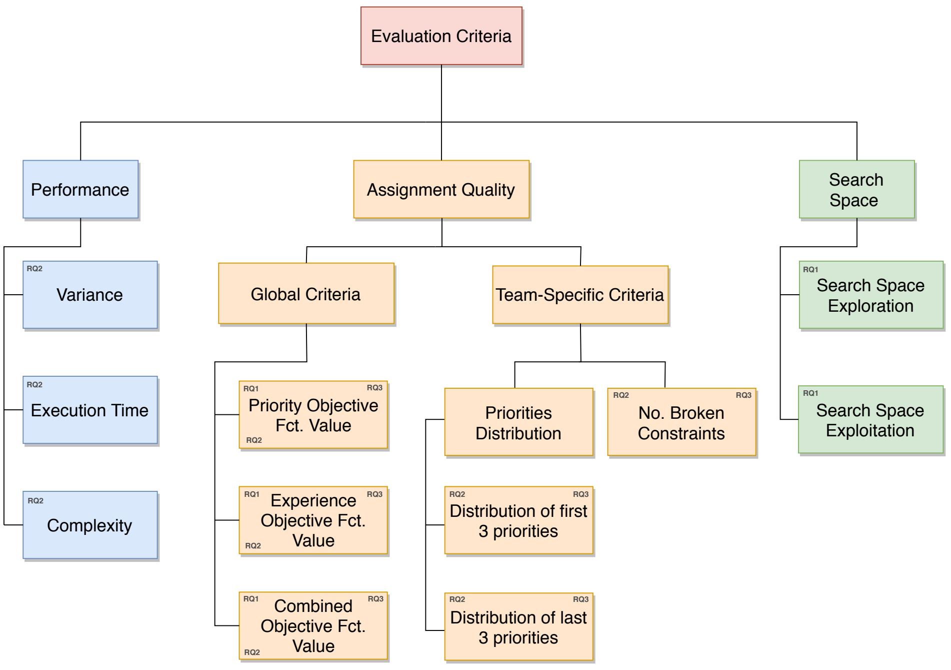

Evaluation Criteria
Performance
Assignment Quality
Search Space
RQ2 Variance
RQ2 Execution Time
RQ2 Complexity
Global Criteria
Team-Specific Criteria
RQ1 Search Space Exploration
RQ1 Search Space Exploitation
RQ1 RQ3 Priority Objective Fct. Value RQ2
RQ1 RQ3 Experience Objective Fct. Value RQ2
RQ1 RQ3 Combined Objective Fct. Value RQ2
Priorities Distribution
RQ2 RQ3 No. Broken Constraints
RQ2 RQ3 Distribution of first 3 priorities
RQ2 RQ3 Distribution of last 3 priorities

Figure 5.2: Taxonomy of the evaluation criteria.

Performance

We use the *sample variance* to measure how far a set of solutions (i.e. their objective values) computed by the same algorithm in multiple iterations are spread out from their average value. That is, a high variance value indicates that the algorithm computes solutions with significantly different objective values if we run it in multiple iterations, with identical constraints. On the other side, a small variance value indicates that the algorithm has (almost)

'deterministic' behavior and, given the same input, it computes similar solutions in each iteration (w.r.t. the objective values). Stochastic algorithms, such as Tabu Search, MOPSO and NSGA-II, are very unlikely to have null variance. However, we are searching for algorithms with small variance.
The *complexity* of an algorithm (time and space complexity) represents another performance criterion we consider in our evaluation. In the following sections, we will analyze the worst case complexity of each algorithm.
Although it is not a key evaluation criterion for our study, the *execution time* of an algorithm can reveal information about its efficiency and, therefore, we will consider it when we analyze the performance of a specific approach.

Assignment Quality - Global Criteria

Given that our project focuses on multi-objective optimization algorithms for the team assignment problem, an important evaluation criterion is related to the objective values of a computed solution. As discussed in section Section 4.1, each assignment solution has an experience and a priority objective value. Because we deal with a minimization optimization problem, the solutions that have smaller objective values are considered better than the ones having higher objective values.
The Tabu Search algorithm was implemented using a conventional weighted approach (CWA) in which the experience and priority objectives were given equal weights (0.5). The MOPSO implementation uses a dynamic weighted approach to compute the assignment solution, but in the following we will use the CWA method for evaluating this algorithm with the other two (MOPSO calculates both the DWA and CWA scores). The NSGA-II algorithm optimizes the two objectives simultaneously. Hence, it computes a set of Pareto solutions, as described in Section 4.2. In the next sections, we will also use the CWA method for evaluating the results computed by NSGA-II.
Thus, the combined objective value (equation 5.2) obtained using the CWA is an aggregation of the experience and priority objectives and is considered in the following to compare multiple solutions.

$$f_{combined} = 0.5 \cdot f_{experience} + 0.5 \cdot f_{priority} \tag{5.1}$$

Assignment Quality - Team-Specific Criteria

The team-specific criteria refer to the priorities distribution and to the number of broken constraints. One of the goals of the team assignment process

is to create a solution such that each student is allocated to a project that he likes the most, that is, a project to which the student assigned a high priority. Thus, for a computed solution, we analyze the distribution of the first 3 and last 3 priorities in each team. Moreover, we check how many practical constraints (i.e. number of test and development devices, number of female students) are broken in a given solution. Naturally, we want solutions that satisfy all the constraints defined by the user, assuming that such a solution exists (i.e. the optimization problem is solvable).
We define an *optimal assignment solution* as a solution that satisfies all constraints, has priority objective value close to 1 and experience objective value close to 0. Moreover, in this solution, a number of students close to the total number of students are assigned to one of their top 3 priorities and as few as possible students receive one of their last 3 priorities.

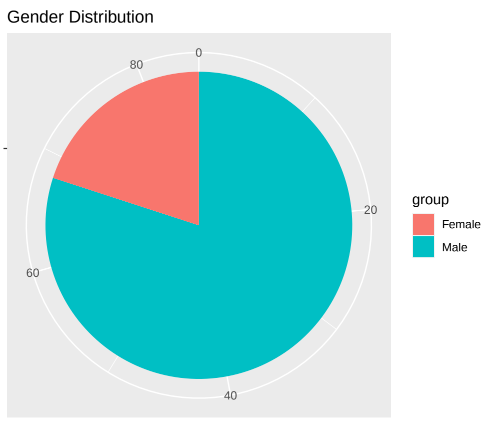
Search Space

In population-based evolutionary algorithms, such as MOPSO and NSGA-II, exploration and exploitation are two key operations that relate to the way in which new solutions are discovered in the search space.
Exploration refers to probing a large area of the search space, aiming to find other promising solutions that are yet to be improved. By doing this operation, the algorithm 'diversifies' the search space in order to avoid getting trapped in a local optimum [4]. Also, exploration is equivalent to a global search performed by the algorithm. Stochastic operators used in metaheuristics (Tabu Search, MOPSO, NSGA-II), such as mutation and crossover, increase their exploratory behavior [19].
Exploitation, however, means probing a limited, but promising region of the search space that has already been discovered by the algorithm (i.e. solutions located in this region have already been found). Thus, the algorithm attempts to improve a promising solution by intensifying the search in the vicinity of the region where this solution is situated. If exploration refers to performing a global search, exploitation is equivalent to a local search.
Ideally, exploration and exploitation should be combined in an algorithm's attempt to find optimal solutions and represent important metrics for evaluating the performance of evolutionary multi-objective algorithms. In the following, we will investigate these two operations mainly by analyzing the distribution of the feasible solutions over the search space.
We believe that the assignment quality criteria are the most important for an instructor of the iPraktikum that wants to create teams using a multi-objective approach. However, since we are evaluating algorithms, and not only the computed assignment solutions, criteria related to the algorithm's

performance and behavior over the search space should also be considered.

5.3 Test Data

In this section we present the test data that we used for evaluating the MOPSO, NSGA-II and Tabu Search algorithms. Anonymized student data from *iPraktikum* was utilized in this scope. Although the original file contains student information clustered into 54 attributes, in this section we describe only the information relevant for the team assignment process.
The test data consists of information about 85 students that must be distributed over 10 teams. The gender distribution of the students is shown in Figure 5.1. Out of 85 persons, 17 (20%) are female students.



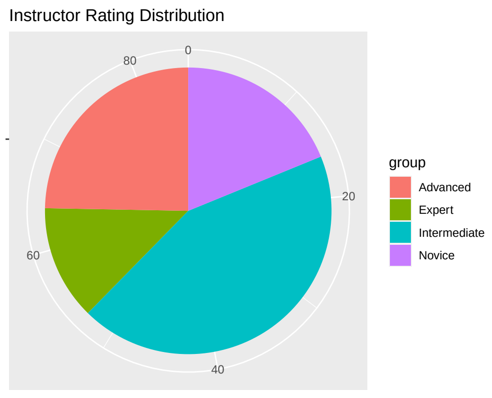


Figure 5.3: Gender (left) and instructor rating (right) distributions of the students

In terms of the instructor rating distribution, the majority of the students have been assigned the rating *Intermediate* (37 students, 43.52%), while 21 (24.70%) students have received the *Advanced* rating. Also, there are 16 (18.82%) *Novice* and 11 (12.94%) *Expert* students.
The test devices (iDevices) can be categorized into 5 classes: iPad, iPhone, Watch, iPadAR (ARKit compatible) and iPhoneAR (ARKit compatible). The distribution of each type of test device can be seen in table 5.1. Overall, there are 57 student that have at least one test device.

Test Device	**Total**
iPad	5
iPhone	15
Watch	6
iPadAR	17
iPhoneAR	40

Table 5.1: Test devices distribution.

Out of 85 student, 48 (56.47%) have a development device (Mac).

We recall that the goal of the algorithms aiming to solve the team assignment problem is to create well-balanced teams. Table 5.2 illustrates the 4 constraints that we set in our optimization process. If applicable, the lower and upper bounds are written for each constraint.

	Total	**Avg. per team**	**Lower bound**	**Upper bound**
Students	85	8.5	8	10
Students with dev. devices	48	4.8	4	-
Students with test devices	57	5.7	5	-
Female students	17	1.7	1	-

Table 5.2: Student data and constraints set during the experiment.

5.4 Optimal Parameter Configurations (RQ1)

In this section, we present the results we obtained in our first step of the evaluation phase. Figure 5.4 illustrates the goal of this step, the two questions which we tried to answer and the metrics we considered for the individual evaluation of the NSGA-II, MOPSO and Tabu Search algorithms.

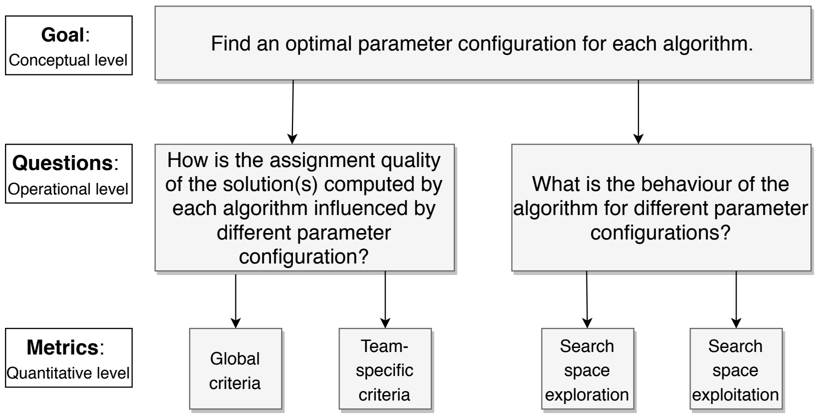

Goal: Conceptual level
Find an optimal parameter configuration for each algorithm.
Questions: Operational level
How is the assignment quality of the solution(s) computed by each algorithm influenced by different parameter configuration?
What is the behaviour of the algorithm for different parameter configurations?
Metrics: Quantitative level
Global criteria
Team-specific criteria
Search space exploration
Search space exploitation

Figure 5.4: Goal-Question-Metric for RQ1.

5.4.1 NSGA-II Algorithm

In this section, we discuss the simulation results of the NSGA-II algorithm. Detailed information about this algorithm can be found in Section 4.2. We tested the algorithm with different values for its specific parameters, i.e. the crossover and mutation probabilities, population size and number of generations.

Crossover Probability

We recall that the crossover is the genetic operation in which two chromosomes randomly exchange information and two new chromosomes are obtained as a result. In our problem, a chromosome represents an assignment solution and the information exchange is equivalent to swapping teams between students.
We ran our NSGA-II implementation with different crossover probabilities and with the other parameter values as follows: population size = 15, no. generations = 10, mutation probability = 0.5. The most relevant results are shown in Tables 5.3 and 5.4 and in Appendix A.1, A.2 additional tables are provided. We observed that high values for this parameter, such as 0.7 and 0.8, determine a high diversity of the Pareto solutions. On the other side, a small crossover probability like 0.02 leads to the fact that two random solutions (i.e. chromosomes) almost never exchange information, which means that the algorithm does not diversify the initial solutions computed at the beginning. Consequently, the Pareto solutions are the optimal ones from the initial population because almost no improvements in the initial solutions can be performed.

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	5.418	0.122	2.770
2.	5.088	0.198	2.643
3.	5.411	0.157	2.784

Table 5.3: Objective values of the Pareto solutions computed by NSGA-II, for crossover probability = 0.02.

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	4.823	0.126	2.474
2.	5.292	0.111	2.701
3.	4.803	0.153	2.478
4.	4.770	0.264	2.517

Table 5.4: Objective values of the Pareto solutions computed by NSGA-II, for crossover probability = 0.7.

Mutation Probability

As described in Section 4.2, a mutation in a solution computed by NSGA-II is represented by the team exchange between two randomly selected students. The higher the mutation probability is, the more likely such an exchange in an assignment solution is.

We tested our NSGA-II implementation with different values for the mutation probability. In each experiment, the other parameters values were: population size = 15, no. generations = 10, crossover probability = 0.75. The most relevant results are shown in Tables 5.5 and 5.6 and in Appendix A.3, A.4, A.5 additional tables are provided. Each table illustrates the priority, experience and combined objective values for the Pareto solutions that were computed with a specific mutation probability.

The best results (w.r.t. the combined objective values) were obtained for mutation probability between 0.5 and 0.7. For extreme probability values (i.e. 0.02 and 1), the quality of the assignment solution was low (high objective values).

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	5.240	0.163	2.701
2.	5.566	0.153	2.859
3.	5.004	0.237	2.620

Table 5.5: Objective values of the Pareto solutions computed by NSGA-II, for mutation probability = 0.02.

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	4.652	0.250	2.451
2.	4.750	0.218	2.484

Table 5.6: Objective values of the Pareto solutions computed by NSGA-II, for mutation probability = 0.5.

We recall that a mutation in a genetic algorithm is a stochastic operator, whose main purpose is to enhance the exploration of the search space. The fact that the best solutions were obtained for ‘middle’ mutation probability values is not surprising, because, for such parameter value, the algorithm has a behavior that maintains an equilibrium between the exploration and the exploitation of the search space.

Population Size

The population size parameter of NSGA-II is similar to the particle count parameter of MOPSO, meaning that it determines the range of the solutions from which the algorithm can choose the Pareto optimal ones. Thus, a large population size leads to the fact that the algorithm can choose the optimal solutions from a wide ‘variety’ of solutions. However, it also means that the algorithm has a smaller chance to refine the solutions that were improved in a previous generation, because choosing two chromosomes (i.e. solutions) for the crossover is done arbitrary. Moreover, due to the large size of the test data, our implementation works very slowly when the population size is above 90.
The objective values of the assignment results computed by NSGA-II ran with population size of 15, 50 and 80 are shown in Tables 5.7, 5.8 and 5.9. As the results indicate, setting the population size to a value between 15 and

50 leads to the best trade-off between the diversity of the Pareto solutions and good combined objective values.

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	4.930	0.315	2.622
2.	5.176	0.241	2.708
3.	5.153	0.242	2.697

Table 5.7: Objective values of the Pareto solutions computed by NSGA-II, for population size = 15.

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	4.908	0.113	2.510
2.	5.076	0.091	2.583

Table 5.8: Objective values of the Pareto solutions computed by NSGA-II, for population size = 50.

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	5.093	0.112	2.602
2.	5.364	0.085	2.724

Table 5.9: Objective values of the Pareto solutions computed by NSGA-II, for population size = 80.

Number of Generations

In each generation, the NSGA-II algorithm tries to improve the solutions computed in the previous generation. A large number of generations means that the algorithm can ‘better’ improve the initial solutions and, thus, has higher chances to find optimal solutions. For our problem, we observed that the objective values of the computed solutions are not improved significantly if the number of generations is high (see Tables 5.10 and 5.11).

Pareto Solution	Priority obj.value	Experience obj.value	Combined obj.value
1.	4.887	0.255	2.571
2.	4.965	0.247	2.606
3.	4.954	0.297	2.625
4.	5.139	0.225	2.682
5.	5.404	0.206	2.805

Table 5.10: Objective values of the Pareto solutions computed by NSGA-II, for no. generations = 25.

Pareto Solution	Priority obj.value	Experience obj.value	Combined obj.value
1.	4.913	0.279	2.596
2.	5.380	0.167	2.773
3.	4.892	0.314	2.603

Table 5.11: Objective values of the Pareto solutions computed by NSGA-II, for no. generations = 100.

Results: Optimal parameter configuration for NSGA-II (RQ1)

The results we obtained from the individual evaluation of the NSGA-II algorithm are presented in Table 5.12.

Coefficient	Experiment results
Crossover probability	High values, such as 0.7 and 0.8, determine a high diversity of the assignment solutions.
Mutation probability	A value between 0.5 and 0.7 leads to assignment solutions with good combined objective values.
Population size	A value between 15 and 50 leads to the best trade-off between the diversity of the Pareto solutions and good combined objective values.
No. generations	A significantly higher value does not necessary guarantee better assignment solutions w.r.t. the combined objective values (compared to a smaller value of this parameter).

Table 5.12: NSGA-II: RQ1 results.

5.4.2 MOPSO Algorithm

In the following, we discuss the simulation results we obtained from the evaluation of the MOPSO algorithm which was applied to solve the team assignment problem. The implementation details of the algorithm were discussed in Section 4.3. We analyze the algorithm's behavior for different values of the coefficients that appear in the velocity formula (presented in Section 4.3).

Inertial Weight

The inertial weight is denoted with w' in the velocity formula and represents a coefficient aiming to tune the influence of the previous found solution on the current solution to be computed by a given particle. Intuitively, the larger the value of w' is, the more influenced the current solution is by its 'past'. The inertial weight must be a value in $[0, 1)$, given the accumulative nature of the velocity updating equation. Otherwise, if w' is greater than 1, the velocity value may grow unbounded, which is, of course, an undesired situation for the MOPSO algorithm.
We ran our MOPSO implementation with different values in $[0, 1)$ for the inertial weight. The parameter setting was the same in each experiment: 200 particles, 1000 iterations, identical values of the cognitive and social coefficients $c_1 = c_2 = 2$. The most relevant results are synthesized in table 5.13 and the plots illustrating the particles created by the algorithm for each inertial weight value can be seen in Figures 5.5 and 5.6. The numbers were truncated to 3 digits after the comma.

w'	c_1	c_2	**Combined obj.**	**Experience obj.**	**Priority obj.**
0	2	2	2.265	0.307	4.223
0.1	2	2	2.392	0.296	4.489
0.5	2	2	2.426	0.182	4.671
0.7	2	2	2.475	0.185	4.766
0.9	2	2	2.497	0.248	4.747

Table 5.13: Combined, experience and priority objectives of the assignment solutions computed by MOPSO with different values for the inertial weight w' (1000 iterations, 200 particles).

We noticed that the best *combined objective* values were obtained when the inertial weight had small values. Thus, the minimum combined objective value (2.265) was computed with $w' = 0$, while the maximum objective value (2.497) corresponds to an inertial weight of 0.9. In terms of the *experi-*

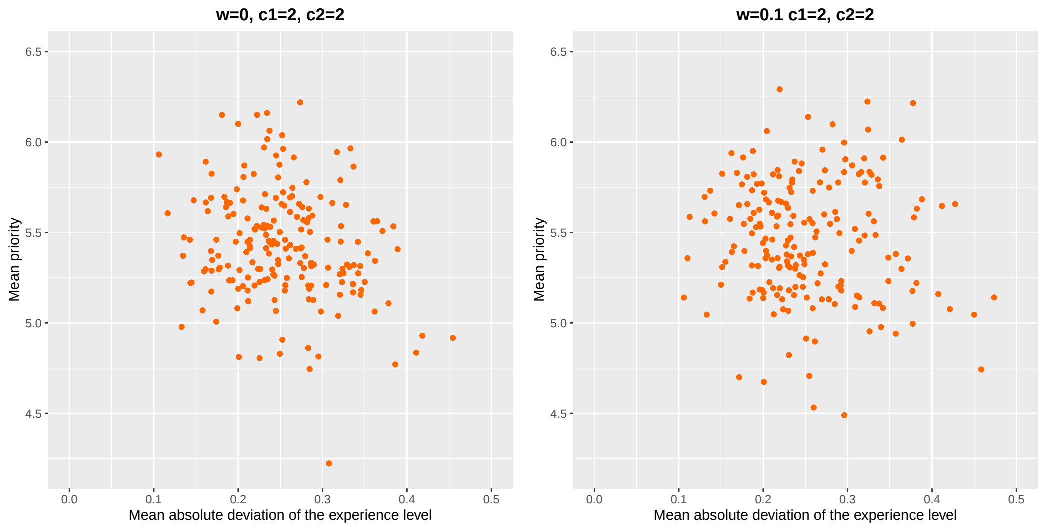


Figure 5.5: Particles computed by MOPSO for w=0 (left) and w=0.1 (right), c1=2, c2=2, 200 particles.

ence objective, we could not identify any pattern that describes the relation between this objective and the value of the inertial weight. However, the *priority objective* tends to be directly proportional with the inertial weight, as the experimental results from the table above indicate. Although a small decrease in the priority objective value can be observed for $w' = 0.9$ in comparison to $w' = 0.7$, the 'simultaneous' increase in the objective value and in the inertial weight is evident.

The distribution of the particles over the search space provides information about the algorithm's behavior with respect to exploration and exploitation, which are metrics that we consider in our evaluation. In Figures 5.5 and 5.6 we can observe the spread of the particles (their personal best values) at the end of the MOPSO algorithm.

A first fact that can be observed is related to the high distance of some isolated points (i.e. particles) from the rest of the points. There are two cases which must be analyzed: isolated particles that have a high objective value and particles with a small, very good objective value.

A remote particle corresponding to the first case can be seen in the first plot (for $w' = 0$) and has experience objective value of approx. 0.3 and priority objective value of approx. 4.25. The reason why such a 'segregated' particle appears in the search space is that the particle is most likely formed in the last iteration of the algorithm, and thus it is impossible that the other points 'follow' it.

An example for the second case can be observed in the third plot that cor-

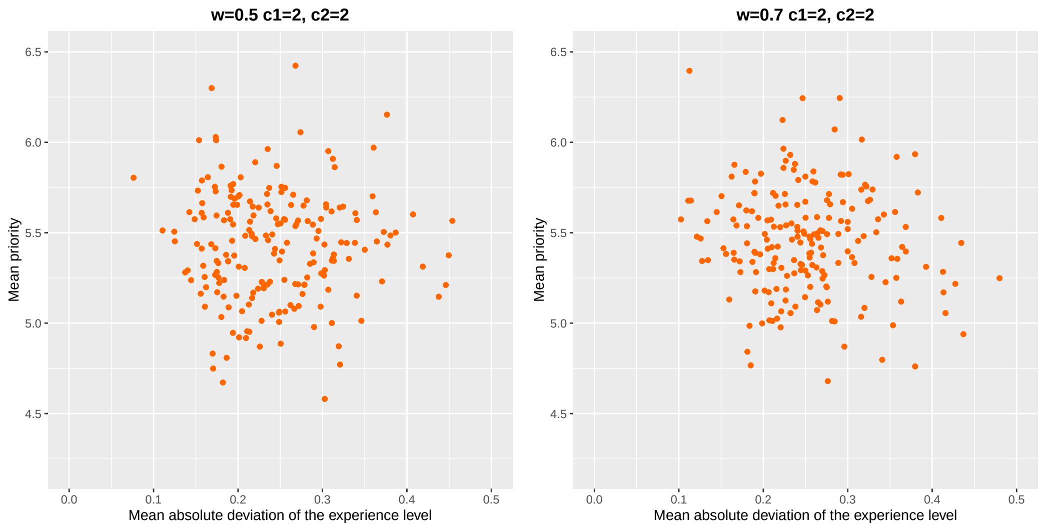


Figure 5.6: Particles computed by MOPSO for w=0.5 (left) and w=0.7 (right), c1=2, c2=2, 200 particles.

responds to an inertial weight of 0.5. The particle has the highest value of the priority objective from the entire swarm and, thus, cannot be the global best particle. Therefore, the algorithm behaves correctly and stops the other particles from following it.

In terms of the *exploration* (global search) of the algorithm, we notice that for $w' = 0.1$, the particles are distributed over a wide area of the search space, which means that the algorithm has high exploratory behavior. It is important to analyze in which part of the search space the exploratory behavior is predominantly developed. For $w' = 0.1$, we observe that the crowded region is extended between priority values of approx. 5-5.75 and experience values of approx. 0.175-2.275. Also, a significant number of particles are distributed over the region of the search space for which the priority and experience objective values are high. This means that, although most of the particles have rather small objective values, the algorithm has also created numerous particles having high objective values, which could not be 'brought' to the crowded region of the search space. This behavior is indeed expected for this extremely small value of the inertial weight. The particles 'forget' about their past when they try to update their position and, therefore, it is difficult to improve their objective values. On the other hand, we notice that for the extreme value $w' = 0.9$, the particles are more concentrated and the area of the search space that they cover is smaller, which means that the exploration rate is low.

Consequently, we can claim that small values of the inertial weight generally

lead to a high exploratory behavior of the algorithm.
At a general level, high *exploitation* is suggested by small regions of the search space in which numerous particles are concentrated. For $w' = 0$, $w' = 0.1$, $w' = 0.7$ and $w' = 0.9$ we notice that the algorithm has indeed behaviors that manifest high exploitation, given the crowded regions of the search space. However, these regions are not fully separated, as the plots indicate.
For the inertial weight $w' = 0.5$ we observe an interesting distribution of the particles. In this case, the algorithm achieves high exploitation, but the crowded regions are separated (one could see three 'holes' in the plot). This distribution of the particles indicates that the algorithm behavior is characterized by a balance between exploitation and exploration. Thus, MOPSO "takes 'bad' actions occasionally to find more promising regions" [19] (exploration) and it then 'exploits' (i.e. tries to refine) the solution that it has initially discovered in its attempt to explore the search space.
To sum up the ideas discussed in this section, it seems that there is no value which can be assigned to the inertial weight such that an optimal behavior of the MOPSO algorithm is guaranteed. While choosing a small inertial weight may result in a low value of the combined objective, it turned out that the experience objective is negatively influenced for this parameter setting. Also, for high inertial weights, the algorithm tends to create particles that are rather gathered in a compact region of the search space, which may cause a reduced exploratory behavior. We think that choosing the inertial weight of 0.5 leads to the best trade-off between all the points we discussed (combined, experience, priority objective values, exploration, exploitation).

c_1 and c_2

As described in section Section 4.3, c_1 and c_2 are coefficients used to tune the impact of the cognitive and social components on the solution computed by each particle in a given iteration. We ran our MOPSO implementation with different values for these coefficients. As for the other algorithm parameters, our goal was to identify a 'behavioral' pattern of the algorithm, that is, to draw conclusions regarding the relationship between the objective values of the computed assignment solution and the c_1 and c_2 values.
Unfortunately, we could not identify any reliable correlation. However, what seemed to determine good objective values for the particles is setting identical values for each coefficient. In Figure 5.8 we observe the particles computed by MOPSO when the coefficients have equal (left) and different values (right). For the right hand side plot, we notice that some particles, having high experience and objective values, tend to be 'separated' by the rest of the

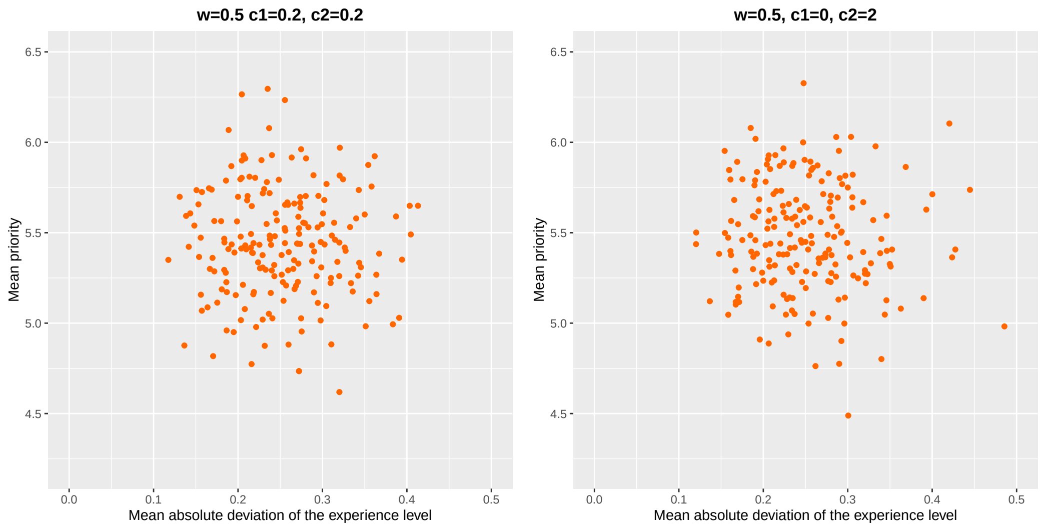


Figure 5.7: Particles computed by the MOPSO algorithm when the coefficient values of c_1 and c_2 are equal (left) and different (right). The other parameter values are w = 0.5, particle count = 200.

swarm. On the other side, in the plot corresponding to equal coefficient values, we see that the particles are concentrated on a smaller area of the search space. Moreover, the approximation of the Pareto front is better for the right solution (can be visually observed because the best particles of the right plot have smaller experience and priority objective values compared to the particles on the left plot).

Particle Count

The particle count C denotes the number of assignment solutions that the MOPSO algorithm computes. The particles (i.e. assignment solutions) are randomly created at the beginning of the algorithm, such that all constraints defined by the user are satisfied. Then, in each algorithm iteration, the particles are improved, meaning that the combined objective value corresponding to each of them becomes smaller or remains equal to the value from the previous iteration.

The global best particle is one of the C particles and is updated in each iteration. The output assignment solution of the algorithm is the solution corresponding to the global best particle, that is, the particle that has the smallest combined objective value.

In figure 5.14 the particles corresponding to different MOPSO algorithm runs with 100, 200 and 300 particles respectively are illustrated. For each particle

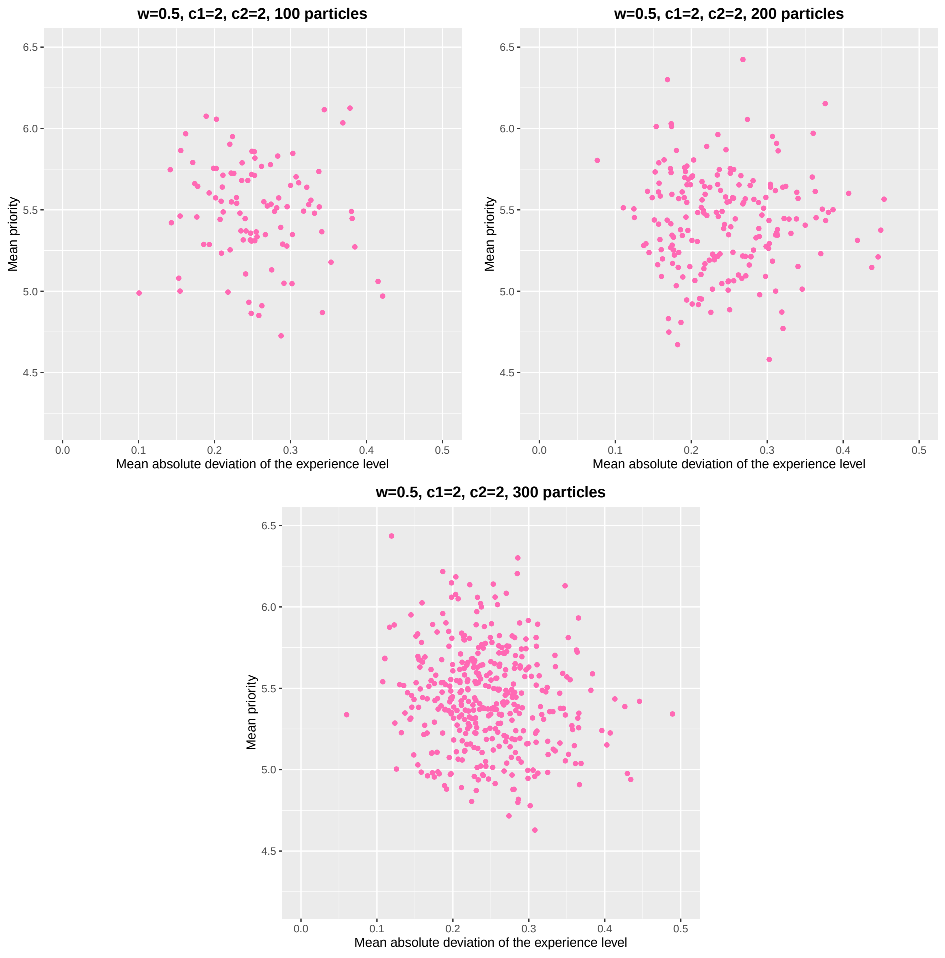


Figure 5.8: Particles computed by the MOPSO algorithm with particle counts equal to 100 (left), 200 (right) and 300 (bottom). The other parameters values are $\mathrm{w} = 0.5$, $\mathrm{c}_1 = 2, c_2 = 2$.

(i.e. point in the plot), the experience and the priority objective values can be observed. We notice that for $C = 300$, the algorithm computes the best approximation of the Pareto front (i.e. the particles with best experience and priority objective values), and also the exploration and exploitation levels are high, as opposed to the algorithm's behavior corresponding to a particle count of 100.

Our simulations revealed that the algorithm computes a better assignment solution when the number of particles was high. This fact is not surprising, given that the initial particles are randomly computed at the beginning of the algorithm, and thus, the probability that multiple particles (assignment solutions) are identical is low. This fact ensures the diversity of the solutions.

Hence, running the algorithm with many particles results in a better global best particle (w.r.t. the objective value).
However, a high particle count comes at the cost of a high running time of the algorithm, as the results from table A.6 in the Appendix show. The time was measured only for the main method of the algorithm and does not include the time required for processing the user-defined constraints. We notice that the running time of the algorithm executed with 300 particles is six times greater than the algorithm's running time corresponding to 50 particles. This difference can become even bigger, depending of the number of constraints and on the cardinality of the students and teams sets. Thus, there is a trade-off between the particle count and the algorithm running time.
Results: Optimal parameter configuration for MOPSO (RQ1)

The results we obtained from the individual evaluation of the MOPSO algorithm are synthesized in the following table.

Coefficient	**Experiment results**
Inertial weight	1. Small values are associated with high experience objective values. 2. Small values are associated with small priority objective values. 3. Small values are associated with small combined objective values. 4. Small values lead to high exploration of the search space. 5. The value 0.5 leads to the best trade-off between all objective values, exploration and exploitation of the search space.
c_1 **and** c_2	Equal values for both coefficients determine better objective values for each particle in the swarm.
Particle count	1. Greater values lead to a better assignment solution w.r.t. the combined objective values. 2. Greater values cause a significantly long execution time of the algorithm. 3. Greater values lead to high exploration of the search space.

Table 5.14: MOPSO: RQ1 results.

5.4.3 Tabu Search Algorithm

In the following, we present the results from the individual evaluation of the Tabu Search algorithm. We studied the algorithm behavior for different values of the neighborhood and of the Tabu list sizes, which are parameters specific to this algorithm.

Neighborhood size

As described in Section 4.4, the neighbors of a solution are solutions that are similar to it. In our implementation, the neighbors of a solutions are formed by swapping teams between a fixed number of students. Thus, the neighborhood size represents the number of (random) students, whose teams are exchanged, such that a new solution is derived from the current one.
We tested the Tabu Search algorithm with different values of the neighborhood size. Our aim was to see 'how much' the initial solution (i.e. assignment solution which satisfies all constraints) is improved in the iterations that the algorithm performs. The most relevant results are shown in table 5.9. We observe that the highest improvement in the combined objective value (0.403) was obtained when the neighborhood size was 50. Also, small improvements can be seen in the final solutions which were iteratively obtained when either too many (85) or too few students (5) were randomly selected to exchange their teams. Moreover, it is important to notice that swapping teams between all students (i.e. neighborhood size = 85) does not lead to a high improvement in the objective value of the final solution, which might be counter-intuitive. Actually, this is a normal behavior of the algorithm, because Tabu Search employs a local search strategy to refine an initial solution in multiple iterations. However, exchanging the teams of all students means creating a new solution that does not have anything in common with the previous solution. In this case, the Tabu Search degenerates into random search.

Neighborhood size	**Initial combined obj.value**	**Final combined obj.value**	**Improvement obj. value**
5	2.818	2.752	0.066
10	2.672	2.571	0.101
20	3.00	2.908	0.092
50	3.178	2.775	0.403
60	2.857	2.725	0.132
85	2.626	2.548	0.078

Table 5.15: Initial & final combined objective values and the improvement in the combined objective values of the solutions computed by Tabu Search with different neighborhood sizes (iterations = 300).

Tabu list size

The Tabu list contains assignment solutions that are neighbors of other solutions which were the 'local best' of the algorithm in a previous iteration. Thus, avoiding to refine a solution which is already in the Tabu list prevents the algorithm from 'going back' to a solution which was already improved. Our experiments have shown that the size of the Tabu list never goes above 200 (in our algorithm simulations, the average size was around 180). Therefore, we think that a dynamic Tabu list with unfixed size is suitable for solving the team assignment problem.

Besides the neighborhood size and the Tabu list size, Tabu Search does not have any specific coefficients. However, given that we are analyzing an iterative algorithm, the number of iterations is another parameter which must be set by the user. Our experiments have shown that the higher the number of iterations is, the better combined objective values tend the final solutions to have (and the improvement in the initial solution is higher).

Results: Optimal parameter configuration for Tabu Search (RQ1)

The results we obtained from the individual evaluation of the Tabu Search algorithm are synthesized in the following table.

Coefficient	**Experiment results**
Neighborhood size	A value of about $\|S\|/2$ leads to high improvements in the combined objective value of the initial solution.
Tabu list size	A dynamic Tabu list with no fixed size leads to high improvements in the combined objective value of the initial solution.

Table 5.16: Tabu Search: RQ1 results.

5.5 Algorithms Comparison (RQ2)

In this section, we compare the MOPSO, Tabu Search and NSGA-II algorithms considering different performance and assignment quality criteria.

The goal of this steps was to find the strengths and weaknesses of each algorithm compared to the other approaches. Also, the two questions which we investigated and the metrics that we used in this step of the evaluation phase are illustrated in the GQM model from the figure below.

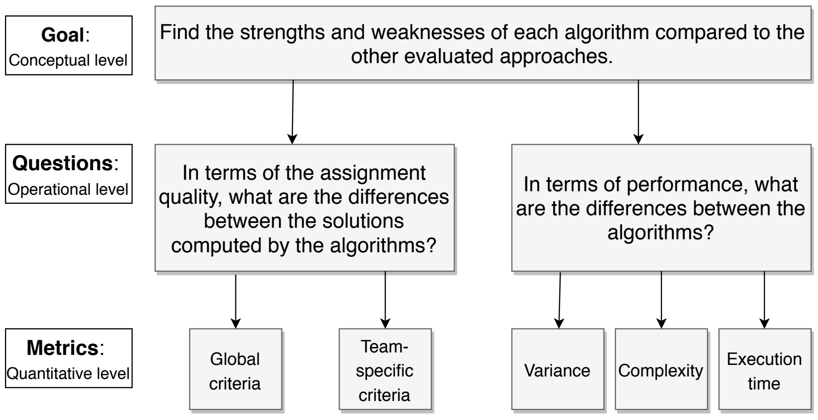

Goal: Conceptual level
Find the strengths and weaknesses of each algorithm compared to the other evaluated approaches.
Questions: Operational level
In terms of the assignment quality, what are the differences between the solutions computed by the algorithms?
In terms of performance, what are the differences between the algorithms?
Metrics: Quantitative level
Global criteria
Team-specific criteria
Variance
Complexity
Execution time

Figure 5.9: Goal-Question-Metric for RQ2.

5.5.1 Assignment Quality Criteria

In this section, we compare the assignment quality of the solutions generated by the MOPSO, Tabu Search and NSGA-II algorithms. As explained in the previous sections, NSGA-II computes a set of Pareto optimal solutions. In our experiment, this algorithm computed three such solutions. Thus, in the following, we evaluate these three solutions generated by NSGA-II and two solutions computed by MOPSO and Tabu Search, respectively. We will focus on the global criteria first and then we will consider team-specific metrics, as indicated in the GQM corresponding to RQ2.

Global Criteria - Objective Function Values

The tables illustrating, for each team, the priority, experience and combined objective function values of the assignment solutions computed by MOPSO, Tabu Search and NSGA-II can be found in Appendix A.7, A.8, A.13, A.15, A.17.

In the table below, the team priority objective values (i.e. mean priority) of the assignment solutions calculated by each algorithm are displayed. We implemented the priority objective as described in Section 4.1. With green are marked the best priority objective value for every team. A smaller value for each team is desired, meaning that the students assigned to a given team

gave a high (i.e. low number) priority to the respective team.
We notice that the MOPSO solution has the best priority objective values for 6/10 teams, while Tabu Search did not compute the best objective value for any team. The NSGA-II solutions are not way better than the Tabu Search solution, given that at most 2 best priority objective values correspond to one of the NSGA-II Pareto solutions.

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
MOPSO solution	4.7	4.8	6.78	4.13	3.25	3.88	5.63	3.88	1.88	5.25
Tabu Search solution	5.22	3.4	6.88	4.44	4.5	5.11	7	7.13	3.75	5.88
NSGA-II PS. 1	6.60	3.33	6.5	3.89	3.38	3.63	6.25	5.25	5.25	5.89
NSGA-II PS. 2	5.8	4.22	7.75	4.56	3.38	3.25	6.5	4.38	5.63	7.22
NSGA-II PS. 3	5.78	7.13	7.13	3.5	3.5	4.9	6.89	5.25	4.25	6.67

Table 5.17: Priority objective values of the assignment solutions generated by each algorithm.

The team experience objective values of the assignment solutions computed by each algorithm are shown in the following table. As described in Section 4.1, the experience objective is the mean deviation of the experience level across the teams. For the given students data set, we calculated the mean experience (around 2.6) and the aim of the algorithms is to minimize the mean deviation of the experience level for all teams. Thus, the smaller the absolute difference between a team experience level and the mean experience is, the 'better' the assignment is. A large value (e.g. above 0.3), however, indicates that the team is either 'too experienced' or 'too inexperienced'.
We notice that the Tabu Search and the third NSGA-II Pareto optimal solution have the best experience objective values for 4 teams, while MOPSO calculated the best value only for one team (tie with the Tabu Search solution).

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
MOPSO solution	0.22	0.18	0.24	0.30	0.32	0.57	0.30	0.32	0.43	0.07
Tabu Search solution	0.13	0.38	0.19	0.24	0.13	0.2	0.57	0.07	0.05	0.07
NSGA-II PS. 1	0.48	0.13	0.18	0.24	0.57	0.19	0.06	0.32	0.07	0.09
NSGA-II PS. 2	0.08	0.32	0.13	0.57	0.32	0.32	0.06	0.07	0.13	0.11
NSGA-II PS. 3	0.02	0.32	0.32	0.07	0.13	0.08	0.13	0.08	0.07	0.24

Table 5.18: Experience objective values of the assignment solutions generated by each algorithm.

Table 5.19 illustrates the combined objective values of the assignment solutions calculated by each algorithm. We observe that the MOPSO solution has the best objective values for 6 teams, while Tabu Search did not compute the best combined objective value for any team. The NSGA-II solutions

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
MOPSO solution	2.46	2.49	3.51	2.21	1.78	2.22	2.96	2.10	1.15	2.66
Tabu Search solution	2.67	1.89	3.53	2.34	2.31	2.65	3.78	3.60	1.90	2.97
NSGA-II PS. 1	3.54	1.73	3.34	2.06	1.97	1.91	3.15	2.78	2.66	2.99
NSGA-II PS. 2	2.94	2.27	3.94	2.56	1.85	1.78	3.28	2.22	2.88	3.66
NSGA-II PS. 3	2.90	3.72	3.72	1.78	1.81	2.49	3.51	2.66	2.16	3.45

Table 5.19: Combined objective values of the assignment solutions generated by each algorithm.

As we noticed, the MOPSO solution has better team priority and combined objective values than the Tabu Search and NSGA-II solutions. In terms of the team experience objective, the Tabu Search solution is better than the other solutions. The global priority, experience and combined objective values are shown in the table below.

Algorithm		**Priority obj. value**	**Experience obj. value**	**Combined obj. value**
MOPSO		4.41	0.29	2.35
Tabu Search		5.33	0.20	2.76
NSGA-II	**Pareto Sol. 1**	4.99	0.23	2.61
	Pareto Sol. 2	5.26	0.21	2.73
	Pareto Sol. 3	5.50	0.14	2.82

Table 5.20: Priority, experience and combined objective values of the solutions computed by each algorithm.

Given the previous observations, it is not surprising that the MOPSO solution is better than the others w.r.t. the priority and combined global objectives. The analyzed MOPSO solution has priority objective value 4.41 and combined objective value 2.35. Both values are considerably better than the objective values of the solutions computed by the Tabu Search and NSGA-II algorithms, as the table illustrates. In terms of the global experience objective, NSGA-II solutions are, on average, better than the solutions computed by the other two algorithms.

We performed the Friedman test [18] to check whether the medians of the priority, experience and combined objective distributions of assignment solutions computed by the three algorithms are the same. The median of a data sample/distribution is the value such that half of the sample values are greater and half are smaller than this value. We used the assignment solutions computed by MOPSO and Tabu Search and the first Pareto solution calculated by NSGA-II. The subjects were the objective function values for each team. The null and alternative hypotheses are:

H_0: there is no difference between the distributions of the priority/experience/combined objective values in the assignment solutions computed by MOPSO, Tabu Search and NSGA-II. Alternatively, this means that the medians of the three distributions are approx. equal.

H_1: at least two distributions of the priority/experience/combined objective values corresponding to the assignment solutions computed by MOPSO, Tabu Search and NSGA-II differ from each other.

We chose the significance level $\alpha = 0.05$. The degrees of freedom are $df = 2$ and the number of subjects $N = 10$. From the table of Friedman critical values we get the critical value 6.2, corresponding to our N and df. Thus, if the Friedman test statistic is greater than 6.2, we reject the null hypothesis. Using R we get:
For the priority objective values we get: FM test statistic = 6.2, $p - value = 0.04505$, so we do not reject H_0.
For the experience objective values we get: FM test statistic = 2.8108, $p - value = 0.2453 > 0.05$, so we do not reject H_0. However, the result is not significant.
For the combined objective values: FM test statistic = 6.2, p-value = 0.04505, so we do not reject H_0.
Therefore, we can say that the assignment solutions computed by the three algorithms do not differ in their median objective values. However, it is important to mention that the Friedman test is not 'very powerful' when the number of subjects is low, such as 10, in our case. Moreover, for the distributions of the priority and combined objective values, the FM test statistic is equal to the FM critical value, which means that we are very 'close' to reject the null hypothesis

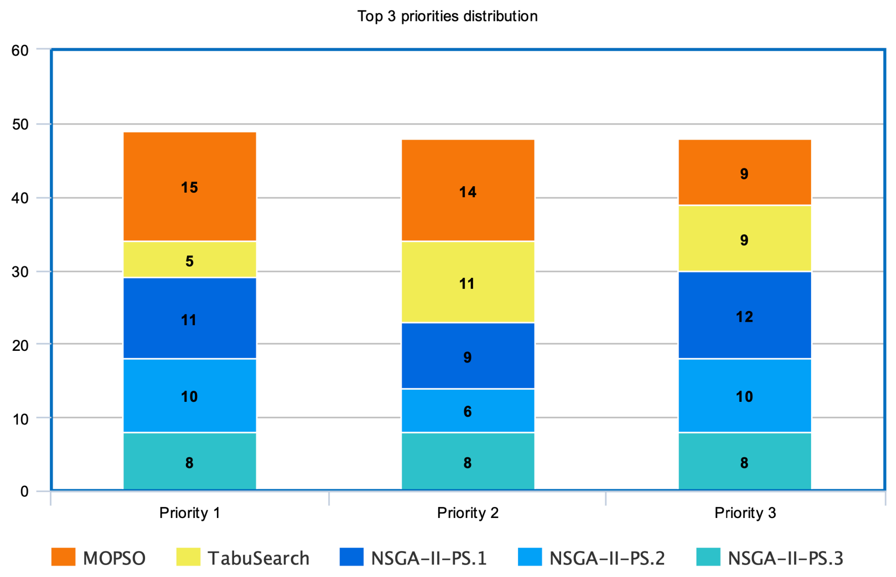


Figure 5.10: Stacked bar chart illustrating the distribution of the top 3 priorities in the assignment solutions computed by each algorithm.

Team-Specific Criteria - Priority Distribution & No. Broken Constraints

The tables showing the priority distribution of the assignment solutions computed by MOPSO, Tabu Search and NSGA-II can be found in Appendix A.10, A.18, A.19, A.20, A.21. In the cell associated to team i and priority j, the number of students assigned to team i who gave priority j to team i is displayed. The colours green and red are used to suggest how 'good' the assignment solution is w.r.t. the distribution of the first 3 and last 3 priorities, respectively. Thus, the greener a solution's table is, the better the distribution of the first 3 priorities is. Similarly, the redder the table is, the worse the distribution of the last 3 priorities is. Tables 5.10 and 5.11 show the distributions of the top and last 3 priorities, respectively. Also, we synthesized the results in table 5.21, which shows how many students were allocated to one of their first 3 and last 3 priorities, respectively. We notice that the best results, for both 'situations', correspond to the MOPSO solution, in which 38 students were allocated to one of their first 3 priorities and 17 students to one of their last 3 priorities. The Tabu Search and the third NSGA-II Pareto optimal solutions are considerably worse than the MOPSO solution, as the numbers indicate (25/24 vs. 38 and 23/26 vs. 17).

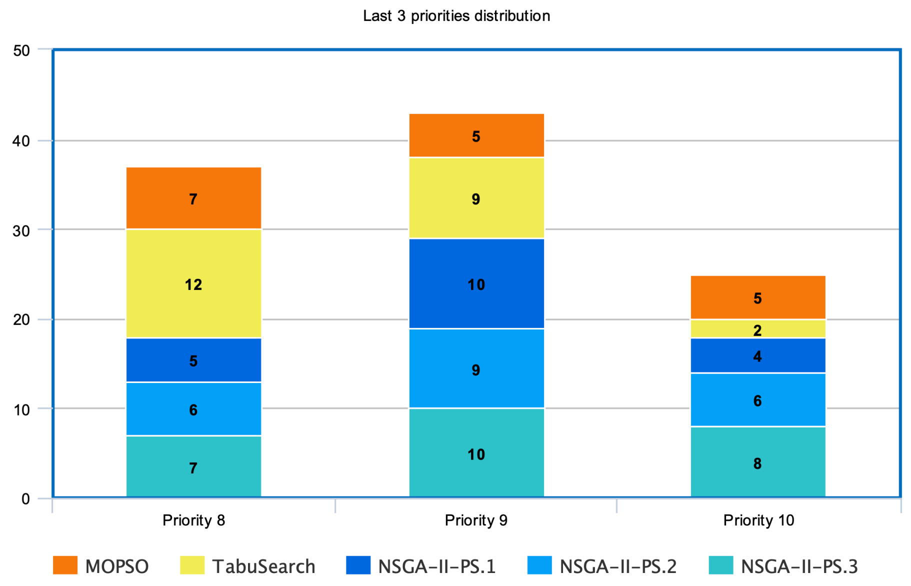


Figure 5.11: Stacked bar chart illustrating the distribution of the last 3 priorities in the assignment solutions computed by each algorithm.

Algorithm		**Distribution of the first 3 priorities**	**Distribution of the last 3 priorities**
MOPSO		38	17
Tabu Search		25	23
	Pareto Sol. 1	32	19
NSGA-II	**Pareto Sol. 2**	26	21
	Pareto Sol. 3	24	25

Table 5.21: Distribution of the first and last 3 priorities in the assignment solutions computed by each algorithm.

In terms of the number of broken constraints, we implemented all three algorithms such that the computed solutions never broke any user-defined constraint.

5.5.2 Performance Criteria

Variance

The table below illustrates the variance of each algorithm, calculated with a sample size of 50. We ran each algorithm for 50 times, with the same constraints (defined in Section 4.1), using its optimal parameter configuration which was discussed in a previous section. For each algorithm, we stored the combined objective values of the assignment solutions calculated in each run. We calculated the variance w.r.t. the combined objective values, that is, the degree to which the combined objective values of the solutions computed in each iteration are spread out from the average combined objective value. The results are shown in Table 5.22.

Algorithm	**Variance**
MOPSO	0.0078
Tabu Search	0.0206
NSGA-II	0.0144

Table 5.22: Variance of the algorithms (sample size = 50).

We observe that MOPSO has the smallest variance (0.0078), while Tabu Search has been proven to have the highest variance (0.0206) among the three algorithms. This means that MOPSO can be seen as a 'safe bet' for solving the team assignment problem, meaning that, given the same constraints, it computes assignment solutions with similar combined objective values if we run it multiple times.
However, all three algorithms have small variance. This fact is very important for our problem and indicates that we can *expect* the algorithms to compute assignment solutions with almost identical combined objective, whenever we run them with the same constraint and parameter configurations.

Complexity

In the following, we will derive the worst case complexity of each algorithm, based on its corresponding pseudocode. Let C be the set of constraints, S the set of students, T the set of teams and M the set of objectives.

MOPSO complexity:

We recall the pseudocode of MOPSO (Algorithm 4.2 defined in Section 4.3). The number of iterations, the dimension of each particle and the number of particles determine the complexity of MOPSO. Let n be the number of particles and t the number of the iterations of the algorithm. Thus, we have:

Random initialization of the particles: $O(n|C||T|)$.
Random initialization of velocity: $O(n|S||T|)$.
Main loop: $O(tn|S|)$.

Given that the number of students is greater than the number of constraints and the number of iterations is greater than the number of teams we have: $O(n|S||T|) >> O(n|C||T|)$ and $O(tn|S|) >> O(n|S||T|)$.
Overall: $O(n|C||T|) + O(n|S||T|) + O(tn|S|) = O(tn|T|)$.

Tabu Search complexity:

The pseudocode of the Tabu Search algorithm can be found in Section 4.4. Let G be the neighborhood set, t the number of iterations of the algorithm and l be the maximum Tabu list size (i.e. the size obtained after the last iteration). The algorithm complexity is:

Random initialization of the initial solution: $O(|C||T|)$.
Main loop: $O(tl|G|)$.
We have $O(|C||T|) << O(tl|G|)$.
Overall: $O(|C||T|) + O(tl|G|) = O(tl|G|)$.

NSGA-II complexity:

The pseudocode of NSGA-II was presented in Section 4.2. Let N be the population size. Thus, adapting the calculations presented in [7] to our problem, we have:

Random initialization of the chromosomes: $O(N|C||T|)$.
Fast nondominating sorting: $O(M(2N)^2) \approx O(MN^2)$.
Crowding distance calculation: $O(M(2N)log(2N))$.
Sorting: $O(2Nlog(2N))$.

It holds $O(MN^2) >> O(M(2N)log(2N)$.
Overall: $O(N|C||T|) + O(MN^2) + O(M(2N)log(2N)) = O(N(|C||T| +$

$MN)$).

Without the initialization required for creating a population with chromosomes that encode solutions satisfying the constraints, the algorithm would have complexity $O(MN^2)$. In fact, $O(MN^2)$ is the standard complexity of the NSGA-II algorithm, as described in [7]. The initialization significantly contributes to the time complexity of this algorithm, so we cannot neglect it when we calculate the computational complexity.
Table 5.23 shows the complexity of each algorithm.

Algorithm	**Complexity**
MOPSO	$O(tn\|T\|)$
Tabu Search	$O(tl\|G\|)$
NSGA-II	$O(N(\|C\|\|T\| + MN))$

Table 5.23: Complexity of the algorithms.

Comparison of complexity:

Tabu Search and MOPSO both depend on the number of iterations (t). In our experiments, we chose the coefficient values $n = 200$, $|G| = 50$ and we observed that the average maximum Tabu list size l is approx. 180. Thus, for the given test data, MOPSO has the best worst case computational complexity among the three algorithms.
As opposed to the other algorithms, the NSGA-II complexity depends on the number of objectives. This algorithm has the worst complexity among the three approaches, which is due to the fast nondominating sorting procedure. Also, NSGA-II is the only algorithm that does not compare different solutions based on a single, combined objective and, in the end, it computes a set of Pareto solutions (unlike the other two algorithms, which compute a single assignment solution).
Therefore, if we consider computational complexity, then MOPSO and Tabu Search are better approaches in comparison to NSGA-II.

Execution time

The execution time of each algorithm is shown in Table 5.24. We notice that Tabu Search and NSGA-II have considerably smaller execution time in comparison to MOPSO. What is interesting to notice is that MOPSO's high execution time does not necessarily depend on the number of iterations of the algorithm. In our simulations, the algorithm had approximately the same execution time when we run it with 100 and with 1000 iterations.

Algorithm	**Execution time (milliseconds)**
MOPSO	≈ 13000
Tabu Search	≈ 155
NSGA-II	≈ 200

Table 5.24: Execution time of the algorithms.

Thus, if we consider the execution time, Tabu Search and NSGA-II are more suitable than MOPSO for solving the team assignment problem.
Table 5.25 illustrates the most important results of the second step of the evaluation phase. For each algorithm, we listed its strengths and weaknesses over the other two approaches.
MOPSO has numerous advantages over the other approaches, such as small variance, best team and global priority and combined objective values and best distribution of the first and last 3 priorities. However, it has the worst execution time. Tabu Search has the highest variance among all three algorithms, but it computes solutions with good team experience objective value and has a good execution time. In terms of the NSGA-II algorithm, it has the worst complexity. However, it computes assignment solutions with good team experience objective values and its execution time is moderate. Also, the fact that it computes multiple Pareto solutions can be seen both as a positive and negative characteristic. On one side, this leads to a diversity in the assignment solutions. On the other side, it might be difficult for the user to choose the most ‘appropriate’ solution.
The results show that, overall, MOPSO is the most suitable among all three approaches for solving the team assignment problem in the iPraktikum.

Results: Strengths and weaknesses of each algorithm (RQ2)

Algorithm	Strengths	Weaknesses
MOPSO	1. Smallest variance 2. Best team priority objective values 3. Best team combined objective values 4. Best global priority objective value 5. Best global combined objective value 6. Best distribution of the first 3 priorities 7. Best distribution of the last 3 priorities	1. Worst execution time
Tabu Search	1. Good team experience objective values 2. Good execution time	1. Highest variance
NSGA-II	1. Good team experience objective value 2. Computes multiple Pareto solutions → diversity & trade-offs between different assignment quality criteria 3. Good execution time	1. Worst complexity 2. Difficult to choose the most 'appropriate' Pareto solution

Table 5.25: Strengths and weaknesses of each algorithm compared to the other two algorithms.

5.6 Multi-Objective vs. Single-Objective Approaches for the Team Assignment Problem (RQ3)

In this section, we compare the assignment solution generated by the single-objective approach implemented in TEASE with the MOPSO solution. We concentrate on the following two research sub-questions:

RQ 3.1: In terms of the assignment quality, what are the differences between these two approaches?

RQ 3.2: In which scenarios is a multi-objective approach preferred over a single-objective approach?

We evaluate the applicability and usability of a MOO assignment solution in the context of iPraktikum and we also discuss the strengths and weaknesses of a multi-objective approach over a single-objective one.

We computed an assignment solution using the approach implemented in TEASE [8], namely the Simplex algorithm. We set the same practical and gender constraints that we used to compute assignment solutions with the MOO approaches. Also, the following constraints related to the skill distribution in each team were defined, such that equally 'experienced' teams are formed:

Instructor rating	**Lower bound**	**Upper bound**
Expert	1	2
Advanced	2	3
Intermediate	3	4
Novice	1	2

Table 5.26: Skill constraints.

The top and last 3 priorities distributions of the Simplex and MOPSO solutions are shown in Figure 5.12 and 5.13, respectively. The team priority objective values of the Simplex and MOPSO solutions are illustrated in Table A.22.
In the following, we summarize the results of the evaluation of multi-objective approaches for solving the team assignment problem in the iPraktikum. We concentrate on the applicability and usability of the assignment solutions computed with such an approach.
Given that all assignment solutions discussed in the previous sections satisfy all practical and gender constraints, we think that a MOO solution is applicable to a real instance of the iPraktikum multi-project course. However, in terms of the usability of such a solution, there are some issues that are addressed in the following.

1. Optimality of an assignment solution generated with a multi-objective approach

Since we evaluated heuristic multi-objective algorithms, the optimality of the generated solutions is not guaranteed, as opposed to the solution computed with Simplex, which is optimal. Thus, there might be solutions with better experience and objective values that the MOPSO, Tabu Search and NSGA-II algorithms do not find. However, the evaluated heuristic algorithms have better complexities (see Section 5.5) than the Simplex algorithm, which has worst-case exponential complexity. Therefore, if the complexity represents

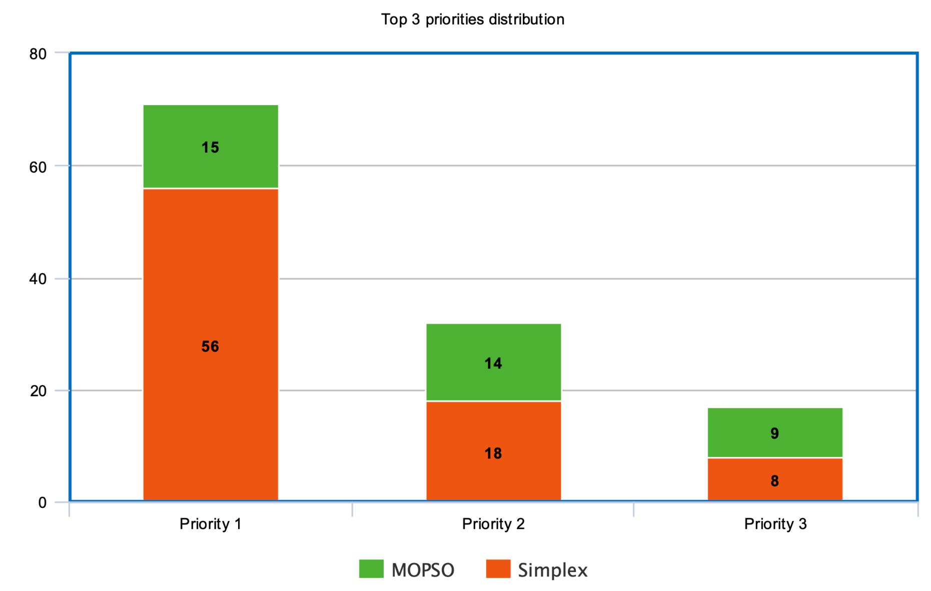


Figure 5.12: Stacked bar chart illustrating the distribution of the first 3 priorities in the assignment solutions computed by Simplex (TEASE) and MOPSO.

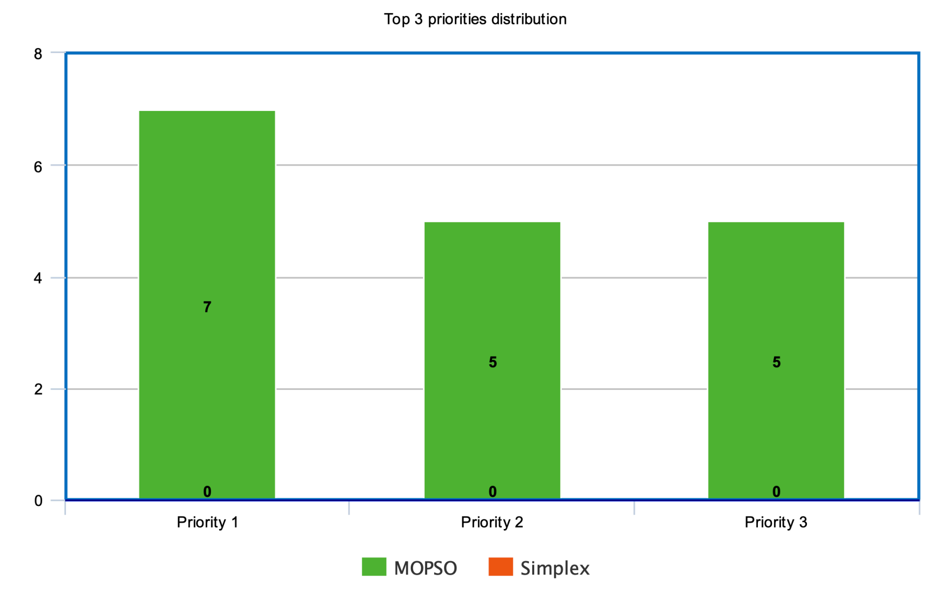


Figure 5.13: Stacked bar chart illustrating the distribution of the last 3 priorities in the assignment solutions computed by Simplex (TEASE) and MOPSO.

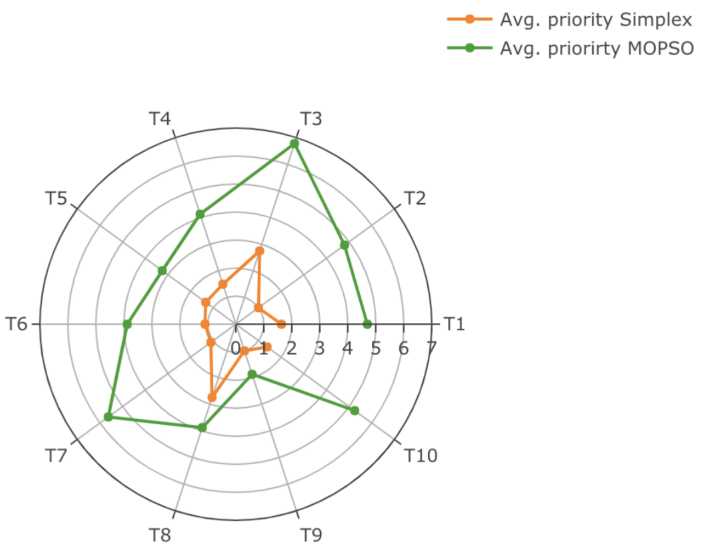


Figure 5.14: Spider plot illustrating the average priority of the assignment solutions calculated by Simplex (TEASE) and MOPSO.

an important criterion for the instructor, then the multi-objective algorithms are preferred over the single-objective method implemented in TEASE.

2. Priority objective

As described in the first sections of this thesis, a multi-objective optimization approach aims to optimize multiple functions simultaneously and, in general, it is impossible that all objective values are optimal in the Pareto solutions. The first thing that can be observed when analyzing the tables illustrating the objective values of the assignment solutions generated with single-objective and multi-objective approaches is that the team and global priority objective values are higher in the MOPSO solution. Thus, if the priority is the most important objective for the instructors, then the solution computed with a single-objective approach is preferred.
However, we recall that in our MOO algorithms we set equal weights (0.5) for the priority and experience objectives. If one objective is more important than the other one, then the user could assign it a higher weight. We implemented Tabu Search with a conventional weighted aggregation method and MOPSO with a dynamic weighted aggregation approach. Therefore, assigning different weights to the objectives can be easily done. This way, the optimization of one objective will be 'more important' than optimizing the other one and the final assignment solution will have better objective values corresponding to the first objective.

3. Experience objective

The experience objective was defined by transforming the constraints related to the instructor ratings into an objective function. All the MOO assignment solutions that we evaluated in the previous sections computed good values of this objective. We recall that the goal of our algorithms is to minimize the mean deviation of the experience level in each team. Thus, we saw that the final assignment solution distributes students in teams such that these are equally 'experienced'. However, the MOO algorithms do not directly consider the instructor ratings when computing assignment solutions (except for the mapping of each rating to a number in {1, 2, 3, 4}, which was discussed in Section 4.1). This is due to the fact that these MOO algorithms optimize real functions. In Section 5.7.1 we describe how the instructor ratings could be integrated in a multi-objective approach for solving the team assignment problem.
As we notice, a multi-objective approach has both advantages and disadvantages over a single-objective method for solving the team assignment problem.

5.7 Extended Experience and Skill & Interest Objectives

In this section, we present two scenarios in which a multi-objective approach is superior to a single-objective approach for solving the team assignment problem in the iPraktikum. In Section 5.7.1 we describe a fuzzy logic approach for modelling students experience and in 5.7.2 we discuss a team skill & interest objective that could be integrated in the team assignment process.

5.7.1 A Fuzzy Logic Approach for Modelling Students Experience

In the following, we present an alternative method for describing the experience levels of the students and a new experience objective that could be integrated in a multi-objective approach for solving the team assignment problem. The experience objective that has been implemented in the evaluated algorithms is presented in Section 4.1 and aims to minimize the difference in 'skill levels' between the teams. We think that describing the experience levels of the students solely by assigning them one of the labels {Expert, Advanced, Intermediate, Novice} does not represent an optimal

starting point for comparing the skill levels between teams, since only four values are possible for deciding the experience level of a person. Also, this approach is designed for working with constraints, rather than with an objective function.
Therefore, we believe that using a fuzzy set for deciding the experience level of each student is a good alternative. A possible idea would be to define a linguistic variable *Experience* and its corresponding linguistic terms could be the instructor ratings {Expert, Advanced, Intermediate, Novice}. The associated crisp set could be the interval $[0, 10]$, as shown in Figure 5.15. Thus, for every student, instead of assigning a single value from the set {Expert, Advanced, Intermediate, Novice}, the instructors could choose a value from the interval $[0, 10]$ to describe their experience level. The membership degree μ for each linguistic term is decided based on this value. Afterwards, an experience score ($s_{experience}$) based on the membership degrees can be computed for every student and a new experience objective function defined accordingly.

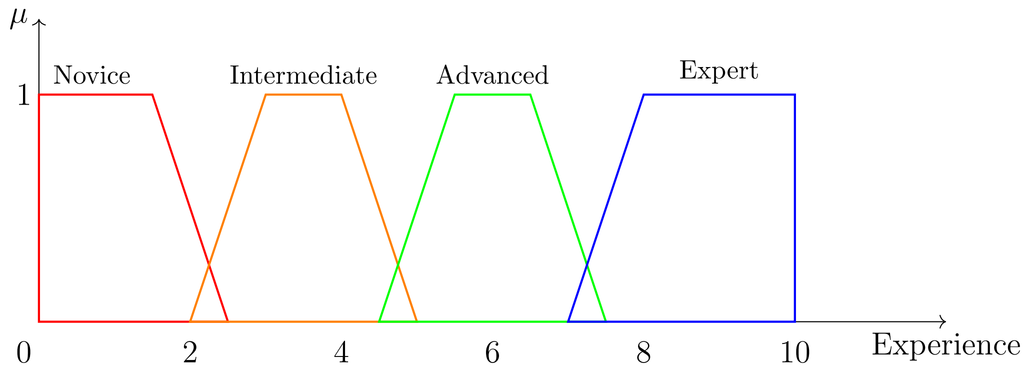


Figure 5.15: Membership function of the Instructor Rating fuzzy set.

Let x be the value from the interval $[0, 10]$ that the instructors assign to a student, based on the skills that the student has. We define the experience score of a student as follows:

$$s_{experience} : [0, 10] \to \mathbb{R} \tag{5.2}$$

$$\begin{aligned} s_{experience}(x) = {} & \mu(x, [0, 10], Novice) \cdot 2x + \mu(x, [0, 10], Intermediate) \cdot 2^2 x + \\ & \mu(x, [0, 10], Advanced) \cdot 2^3 x + \mu(x, [0, 10], Expert) \cdot 2^4 x \end{aligned} \tag{5.3}$$

The membership degrees (e.g. $\mu(x, [0, 10], Novice)$) can be read from the Figure 5.15. The new experience objective function is defined now as in Section 4.1, using $s_{experience}$ instead of r_i.
In the iPraktikum context, this approach would imply additional effort for the instructors. Assigning each student a value from $[0, 10]$ might be a difficult task. However, the form [8] that each student must complete before the team assignment is computed contains information that could help the instructors to set 'appropriate' experience levels.

5.7.2 Team Skill & Interest Objective

As described in [9], assigning students to a project that requires knowledge in the fields that the student is interested in positively influences the student's involvement in the course and also the development of the assigned project. In our evaluation, we did not consider team-specific criteria, but, in reality, most of the projects have specific skill requirements that should be fulfilled by the students from their allocated teams.
We believe that a multi-objective approach could be used for solving the team assignment problem in the iPraktikum when teams have specific skill requirements. Each project for which a team should be created focuses on specific fields or technologies, such as computer vision or embedded programming [9]. The iPraktikum Team Initialization Questionnaire [8] contains questions that ask students to state their skills and interests in software engineering and related fields. Thus, based on the students' answers, one could define the following objective function for each team:

$$f_{team_t} = w_1 \cdot f_{skill_t} + w_2 \cdot f_{interest_t} \quad \forall t \in T \tag{5.4}$$

where:

$$w_1 + w_2 = 1 \tag{5.5}$$

$$f_{skill_t} = \sum_{field \in F} \mathbb{1}_{field}(t) \cdot \sum_{i \in S} x_{it}, \cdot skill(i, field), \tag{5.6}$$

$$f_{interest_t} = \sum_{field \in F} \mathbb{1}_{field}(t) \cdot \sum_{i \in S} x_{it} \cdot interest(i, field), \tag{5.7}$$

$$\mathbb{1}_{field}(t) = \begin{cases} 1 & \text{if team } t \text{ requires knowledge in } \textit{field} \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

$$skill(i, field) = \begin{cases} 0 & \text{if student } i \text{ has no skills in } \textit{field} \\ 1 & \text{if student } i \text{ has beginner skills in } \textit{field} \\ 2 & \text{if student } i \text{ has average skills in } \textit{field} \\ 3 & \text{if student } i \text{ has advanced skills in } \textit{field} \\ 4 & \text{if student } i \text{ has expert skills in } \textit{field} \end{cases} \tag{5.9}$$

$$interest(i, field) = \begin{cases} 0 & \text{if student } i \text{ is not interested at all in } \textit{field} \\ 1 & \text{if student } i \text{ is hardly interested in } \textit{field} \\ 2 & \text{if student } i \text{ has average interest in } \textit{field} \\ 3 & \text{if student } i \text{ has high interest in } \textit{field} \\ 4 & \text{if student } i \text{ is extremely interested in } \textit{field} \end{cases} \tag{5.10}$$

$$F = \{\text{Frontend Development, Server-side Development, Embedded Development, Virtual and Augmented Reality, Machine Learning and Algorithms, UI/UX Design}\}$$

The objective functions defined for each team as in equation 5.4 could then be maximized (together with the optimization of the other objectives defined in Section 4.1) using a multi-objective solver that computes an assignment solution. This approach represents an alternative to the original way of dealing with team-specific requirements using constraints (as implemented in TEASE).

Chapter 6

Conclusion & Outlook

In this thesis we evaluated multi-objective approaches for solving the team assignment problem in multi-project courses such as the iPraktikum. We proposed a multi-objective formulation of this problem and solved it using the MOPSO, NSGA-II and Tabu Search algorithms, which were selected after a theoretical analysis of various multi-objective methods was conducted. A system has been implemented in Java, that allows a user (e.g. instructor of the iPraktikum) to define global constraints that should be satisfied in the assignment solution(s) that one of the three solvers generates.
The above mentioned algorithms were evaluated in a controlled experiment in which performance, assignment quality and search space related criteria were considered. We conducted both an individual evaluation of each algorithm, aiming to identify an optimal parameter configuration, and a pairwise comparison of the three multi-objective methods, with the purpose of determining the strengths and weaknesses of each approach. Moreover, the advantages and disadvantages of a multi-objective approach over a single-objective approach for solving the team assignment problem in the iPraktikum were identified. We proposed a fuzzy logic approach for modelling students experience and a team skill & interest objective that could be integrated in a multi-objective solver to compute assignment solutions.
The future work includes integrating a multi-objective algorithm into TEASE, such that the user can choose what type of optimization approach (single- or multi-objective) he wants to use for creating teams. The user interface of the tool could be accordingly extended, such that the details of the generated assignment solution(s) (e.g. objective values, priorities distribution) are displayed in a user-friendly way. Furthermore, other multi-objective algorithms, such as interactive methods, could be evaluated in the context of the iPraktikum.

Appendix A

(Appendix Chapter 5)

Pareto Solution	Priority obj.value	Experience obj.value	Combined obj.value
1.	5.119	0.198	2.658
2.	5.441	0.162	2.801
3.	4.996	0.190	2.593

Table A.1: Objective values of the Pareto solutions computed by NSGA-II, for crossover probability = 0.5.

Pareto Solution	Priority obj.value	Experience obj.value	Combined obj.value
1.	4.939	0.168	2.553
2.	4.701	0.354	2.527
3.	5.557	0.109	2.883
4.	4.912	0.243	2.577
5.	5.444	0.167	2.805
6.	4.813	0.291	2.552

Table A.2: Objective values of the Pareto solutions computed by NSGA-II, for crossover probability = 0.8.

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	4.803	0.211	2.507
2.	4.661	0.232	2.446
3.	4.884	0.205	2.544

Table A.3: Objective values of the Pareto solutions computed by NSGA-II, for mutation probability = 0.7.

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	5.310	0.112	2.711
2.	5.303	0.118	2.710

Table A.4: Objective values of the Pareto solutions computed by NSGA-II, for mutation probability = 0.9.

Pareto Solution	**Priority obj.value**	**Experience obj.value**	**Combined obj.value**
1.	5.269	0.165	2.717

Table A.5: Objective values of the Pareto solutions computed by NSGA-II, for mutation probability = 1.

Particle count	**Execution time (milliseconds)**
50	3428
100	6639
150	9914
200	14347
300	20602

Table A.6: The execution time of the MOPSO algorithm for different particle count values.

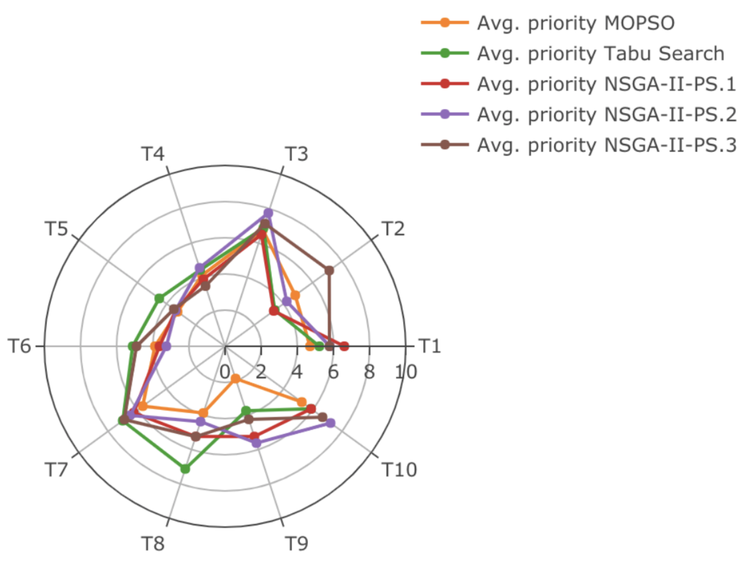


Figure A.1: Spider plot illustrating the average priorities of the assignment solutions generated by MOPSO, Tabu Search and NSGA-II.

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Priority obj. value	4.7	4.8	6.78	4.13	3.25	3.88	5.63	3.88	1.88	5.25
Experience obj. value	0.22	0.18	0.24	0.30	0.32	0.57	0.30	0.32	0.43	0.07
Combined obj. value	2.46	2.49	3.51	2.21	1.78	2.22	2.96	2.10	1.15	2.66

Table A.7: MOPSO Solution (priority objective value: 4.41, experience objective value: 0.29, combined objective value: 2.35)

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Priority obj. value	5.22	3.4	6.88	4.44	4.5	5.11	7	7.13	3.75	5.88
Experience obj. value	0.13	0.38	0.19	0.24	0.13	0.2	0.57	0.07	0.05	0.07
Combined obj. value	2.67	1.89	3.53	2.34	2.31	2.65	3.78	3.60	1.90	2.97

Table A.8: Tabu Search Solution (priority objective value: 5.33, experience objective value: 0.20, combined objective value: 2.76)

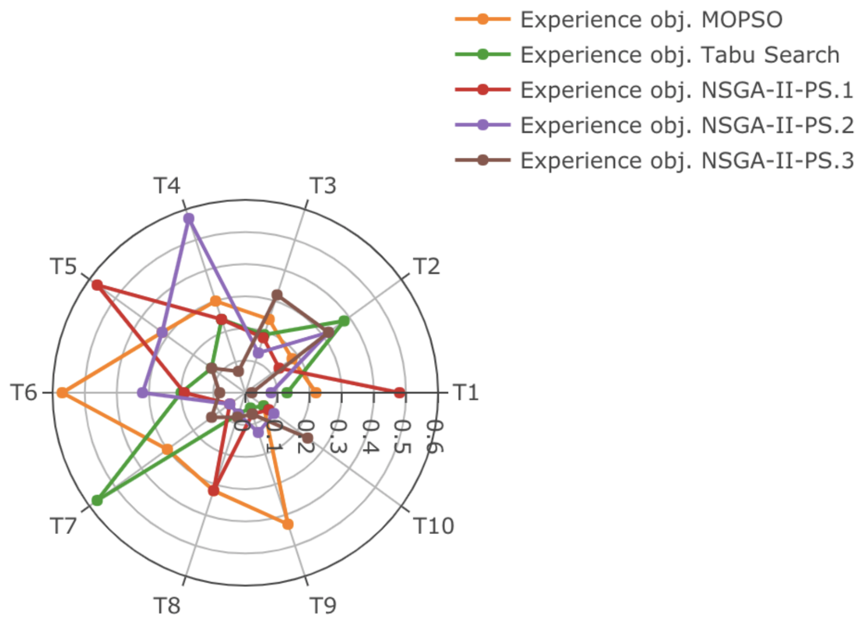


Figure A.2: Spider plot illustrating the experience objectives of the assignment solutions generated by MOPSO, Tabu Search and NSGA-II.

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
#Priority 1	2	0	1	1	1	3	0	3	3	1
#Priority 2	1	3	1	1	3	0	0	1	3	1
#Priority 3	1	0	0	1	2	1	0	0	2	2
#Priority 4	2	3	0	3	1	0	1	2	0	0
#Priority 5	0	0	0	0	0	2	3	0	0	0
#Priority 6	0	2	0	0	0	1	2	0	0	1
#Priority 7	2	0	1	1	0	0	2	0	0	1
#Priority 8	1	1	4	1	0	0	0	0	0	0
#Priority 9	0	0	1	0	1	1	0	2	0	0
#Priority 10	1	1	1	0	0	0	0	0	0	2

Table A.9: MOPSO: Priority distribution.

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Expert students	1	3	1	2	0	0	1	1	1	1
Advanced students	1	1	4	1	2	2	4	0	4	2
Intermediate students	6	4	3	5	4	2	2	5	3	3
Novice students	2	2	1	0	2	4	1	2	0	2
Broken constraints	0	0	0	0	0	0	0	0	0	0

Table A.10: MOPSO: Instructor rating distribution and number of broken constraints.

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
#Priority 1	0	3	0	1	0	1	0	0	0	0
#Priority 2	2	2	1	1	4	0	0	0	1	0
#Priority 3	1	0	0	2	0	1	1	0	3	1
#Priority 4	0	3	1	1	0	0	0	1	1	3
#Priority 5	1	0	1	0	1	3	0	2	3	0
#Priority 6	2	1	0	2	0	2	2	0	0	1
#Priority 7	1	0	0	1	1	1	1	1	0	0
#Priority 8	2	0	2	1	2	1	2	1	0	1
#Priority 9	0	1	2	0	0	0	2	2	0	2
#Priority 10	0	0	1	0	0	0	0	1	0	0

Table A.11: Tabu Search: Priority distribution.

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
#Priority 1	0	4	0	2	0	3	1	1	0	0
#Priority 2	1	1	0	1	4	1	0	1	0	0
#Priority 3	1	0	1	3	1	0	0	1	3	2
#Priority 4	0	1	0	0	1	0	0	2	1	2
#Priority 5	0	0	2	1	1	2	2	0	1	1
#Priority 6	1	1	2	0	0	1	1	0	0	0
#Priority 7	4	2	0	1	1	0	1	0	1	1
#Priority 8	0	0	1	0	0	1	1	0	1	1
#Priority 9	3	0	1	0	0	0	2	2	1	1
#Priority 10	0	0	1	1	0	0	0	1	0	1

Table A.12: NSGA-II, Pareto Solution 1: Priority distribution.

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Priority obj. value	6.60	3.33	6.5	3.89	3.38	3.63	6.25	5.25	5.25	5.89
Experience obj. value	0.48	0.13	0.18	0.24	0.57	0.19	0.06	0.32	0.07	0.09
Combined obj. value	3.54	1.73	3.34	2.06	1.97	1.91	3.15	2.78	2.66	2.99

Table A.13: NSGA-II Pareto Solution 1 (priority objective value: 4.99, experience objective value: 0.23, combined objective value: 2.61)

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
#Priority 1	1	1	0	2	1	3	0	2	0	0
#Priority 2	1	2	0	0	2	1	0	0	0	0
#Priority 3	0	0	0	2	2	1	1	1	2	1
#Priority 4	0	3	0	1	1	0	0	3	0	2
#Priority 5	1	1	1	1	1	2	1	0	3	1
#Priority 6	1	0	1	0	0	0	2	0	0	0
#Priority 7	5	1	0	2	1	0	2	0	1	0
#Priority 8	0	0	4	0	0	1	0	0	1	0
#Priority 9	1	1	1	0	0	0	2	2	1	1
#Priority 10	0	0	1	1	0	0	0	0	0	4

Table A.14: NSGA-II, Pareto Solution 2: Priority distribution.

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Priority obj. value	5.8	4.22	7.75	4.56	3.38	3.25	6.5	4.38	5.63	7.22
Experience obj. value	0.08	0.32	0.13	0.57	0.32	0.32	0.06	0.07	0.13	0.11
Combined obj. value	2.94	2.27	3.94	2.56	1.85	1.78	3.28	2.22	2.88	3.66

Table A.15: NSGA-II Pareto Solution 2 (priority objective value: 5.26, experience objective value: 0.21, combined objective value: 2.73)

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
#Priority 1	0	0	0	2	2	2	0	0	2	0
#Priority 2	2	0	1	2	1	1	0	0	1	0
#Priority 3	0	0	0	1	0	1	2	1	1	2
#Priority 4	0	1	0	1	4	0	1	3	0	0
#Priority 5	2	3	1	0	0	1	0	0	2	1
#Priority 6	1	0	2	0	0	3	0	2	0	1
#Priority 7	2	0	0	1	0	0	1	1	0	1
#Priority 8	1	0	0	1	1	0	1	1	1	1
#Priority 9	0	2	2	0	0	1	3	0	1	2
#Priority 10	1	2	2	0	0	1	1	0	0	1

Table A.16: NSGA-II, Pareto Solution 3: Priority distribution.

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Priority obj. value	5.78	7.13	7.13	3.5	3.5	4.9	6.89	5.25	4.25	6.67
Experience obj. value	0.02	0.32	0.32	0.07	0.13	0.08	0.13	0.08	0.07	0.24
Combined obj. value	2.90	3.72	3.72	1.78	1.81	2.49	3.51	2.66	2.16	3.45

Table A.17: NSGA-II Pareto Solution 3 (priority objective value: 5.50, experience objective value: 0.14, combined objective value: 2.82)

Team (anonymized)	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
Expert students	0	4	1	0	2	1	0	0	1	2
Advanced students	4	0	2	5	2	2	1	2	2	1
Intermediate students	5	5	2	4	2	3	4	6	4	2
Novice students	0	1	3	0	2	3	3	0	1	3
Broken constraints	0	0	0	0	0	0	0	0	0	0

Table A.18: Tabu Search: Instructor rating distribution and number of broken constraints.

Team (anonymized)	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Expert students	1	1	2	2	0	0	2	0	2	1
Advanced students	6	3	1	2	1	3	2	1	0	2
Intermediate students	3	4	4	4	4	3	1	6	4	4
Novice students	0	1	1	1	3	2	3	1	2	2
Broken constraints	0	0	0	0	0	0	0	0	0	0

Table A.19: NSGA-II Pareto Solution 1: Instructor rating distribution and number of broken constraints.

Team (anonymized)	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Expert students	0	1	1	4	0	0	2	0	2	1
Advanced students	6	1	3	2	2	1	1	2	1	2
Intermediate students	2	4	3	1	4	6	3	6	4	4
Novice students	2	3	1	2	2	1	2	0	1	2
Broken constraints	0	0	0	0	0	0	0	0	0	0

Table A.20: NSGA-II Pareto Solution 2: Instructor rating distribution and number of broken constraints.

Team (anonymized)	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Expert students	2	0	0	1	1	2	2	1	0	2
Advanced students	1	1	3	1	4	3	1	3	2	2
Intermediate students	4	6	2	5	1	2	5	2	6	4
Novice students	2	1	3	1	2	3	1	2	0	1
Broken constraints	0	0	0	0	0	0	0	0	0	0

Table A.21: NSGA-II Pareto Solution 3: Instructor rating distribution and number of broken constraints.

Team (anonymized)	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Simplex solution	1.63	1	2.75	1.5	1.33	1.1	2.75	1	1.38	1.38
MOPSO solution	4.7	4.8	6.78	4.13	3.25	3.88	5.63	3.88	1.88	5.25

Table A.22: Priority objective values of the assignment solutions computed by Simplex (priority obj. value: 1.58) and MOPSO (priority obj. value: 4.41).

List of Figures

Bibliography

[1] Hazem Ahmed and Janice Glasgow. Swarm intelligence: Concepts, models and applications. 02 2012.

[2] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach2. The goal question metric approach. 1994.

[3] Bernd Bruegge and Allen H Dutoit. *Object Oriented Software Engineering Using UML, Patterns, and Java.* Prentice Hall, 2009.

[4] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Springer, NY 10013, USA,, 2007.

[5] J. L. Cohon and D. H. Marks. A review and evaluation of multiobjective programming techniques. *Water Resources Research*, pages 208–220, 1975.

[6] G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Trans. Evol. Comp*, 6(2):182–197, April 2002.

[8] Dora Dzvonyar. *TEMPO: A Framework for Team Composition and Management in Project-Based Organizations.* Dissertation, Technische Universität München, München, 2019.

[9] Dora Dzvonyar, Lukas Alperowitz, Dominik Henze, and Bernd Bruegge. Team composition in software engineering project courses. *Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials*, pages 16–23, 2018.

[10] Dora Dzvonyar, Dominic Henze, Lukas Alperowitz, and Bernd Bruegge. Algorithmically supported team composition for software engineering project courses. *IEEE Global Engineering Education Conference*, pages 1753–1760, 2018.

[11] Jose Esgario, Iago Egias da Silva, and Renato Krohling. Application of genetic algorithms to the multiple team formation problem. 03 2019.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman Publishing Co., Inc., USA, 1995.

[13] Fred Glover and Manuel Laguna. *Tabu search I*, volume 1. 01 1999.

[14] Frederick Herzberg. Motivation-hygiene theory. Organizational Behavior I: Essential Theories of Motivation and Leadership, John B. Miner:61–75, 2005.

[15] Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. Dynamic weighted aggregation for evolutionary multi-objective optimization: Why does it work and how? 04 2001.

[16] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, New Jersey.

[17] F. S. Lobato and V. Steffen. *Multi-Objective Optimization Problem.* Springer, 2017.

[18] Richard Lowry. The friedman test for 3 or more correlated samples.

[19] Seyedali Mirjalili and Jin Song Dong. *Multi-Objective Optimization using Artificial Intelligence Techniques.* Springer, Cham, Switzerland, 2019.

[20] A. Osyczka. Multicriteria optimization for engineering design. *Design Optimization*, pages 193–227, 1985.

[21] Colin Reeves. *Genetic Algorithms*, volume 146, pages 109–139. 09 2010.

[22] Gonsalves Tad and Itoh Kiyoshi. Multi-objective optimization for software development projects. *Lecture Notes in Engineering and Computer Science*, 2180, 03 2010.

[23] Lianying Zhang and Xiang Zhang. Multi-objective team formation optimization for new product development. *Computers Industrial Engineering*, 64(3):804– 811, 2013.

[24] Stanley Zionts and Jyrki Wallenius. An interactive programming method for solving the multiple criteria problem. *Management Science*, 22:652–663, 1976.