Improving group project matchings of TU Delft's Project Forum

Author: Philippe Andreevic Louchtch

Date: 11 April 2025

Improving group project matchings of TU Delft's Project Forum

By

P. A. Louchtch

In partial fulfilment of the requirements for the degree of:

Master of Science
in Computer Science

at the Delft University of Technology,
to be defended publicly on Friday, 25 April 2025 at 11:00 in 4.W510 Van Wijngaarden

Supervisor:	Prof. M. de Weerdt	
Thesis committee:	G. Migut,	TU Delft

Contents

Abstract

This thesis investigates improving the group project matching algorithm of TU Delft's Project Forum platform. We formalize the matching problem as a many-to-one, one-sided matching with group formation, where students have preferences over project topics and may wish to pregroup with peers.

We implement and evaluate two mechanisms: Chiarandini – a mechanism reimplemented from the literature and adapted to our grouping setting – and our novel Fair mechanism which incorporates additional fairness considerations for pregrouping. Both use mixed integer linear programming. Leximin is selected as the objective function. These two mechanisms are compared to our previous mechanism, BEPSys, which matches greedily determined groups of students to projects using an optimal algorithm.

Our evaluation uses historical instances from TU Delft and SDU as well as synthetic instances that we generate. Results show that both new mechanisms significantly outperform the prior BEPSys algorithm, both in terms of quality of results and runtime. Our novel Fair mechanism successfully allows pregrouping without disadvantaging solo students, although sometimes at significant cost to pregrouping students. The leximin objective, implemented using an ordered weighted averaging function, is shown to not work optimally in some tested instances.

1 Introduction

The Delft University of Technology (TU Delft), like many comparable universities, has courses where students need to work on some topic of choice in groups, the so-called group project courses. The topics have a limited capacity, so not everyone can get their most desired project topic. Naturally, for smaller courses the students and the teachers can go through the process of group creation and project assignment manually: (1) students first form groups and announce their formation to the course administration, (2) the topics are then somehow assigned to those groups, such as a first-come first-serve principle or they may even submit a shortlist of their top picks to the course administration for consideration. Cumbersome but manageable for small classes. However, when there are many students participating in mandatory courses, such as the case with the Computer Science undergraduate program, automation is required to ensure an efficient, streamlined, orderly and not least, fair and optimal matching process.

In this thesis we are investigating how to improve the group project matching algorithm of *Project Forum* (formerly *BEPSys*), TU Delft's platform for managing group project courses. The current matching mechanism, referred to as BEPSys, has several limitations that affect both the quality of matchings and the satisfaction of students, which creates additional work for the course administration in the form of complaints and manual corrections. Our goal is to develop and evaluate alternative mechanisms that can produce better matchings within maintaining acceptable runtimes, even for large problem instances.

The matching problem we face is complex: we must assign students to project topics while simultaneously forming groups that satisfy size constraints. Students have preferences over which project topics they want to work on, and some students also have preferences about whom they want to work with. This creates a many-to-one, one-sided matching problem with a group formation component. Our objective is to optimize student welfare in a fair manner - ensuring that no student is disproportionately disadvantaged for the marginal improvement of others.

The importance of fairness cannot be overstated. A matching where 95% of students are extremely satisfied but 5% are deeply unhappy may result in excessive work for course administration, not to mention potential loss of motivation for those students. From our experience and anecdotal evidence, students typically fall into two categories: those with strong interest in a small subset of project topics, and those who care more about working with preferred colleagues. A good matching mechanism should accommodate both types of students. An additional requirement is to institute an additional notion of fairness between the two types of students.

Typical problem instances at TU Delft involve upwards of 150 students, 40 projects, and group size bounds between 4 and 5 students. The current BEPSys algorithm handles this by first greedily grouping students based on similarity of preferences, then assigning these groups to projects, see Appendix B (AIDM-B: Improving BEPSys) for a more formal description of the BEPSys mechanism. While functional, this approach often produces suboptimal matchings, especially for students who prioritize project topics over grouping preferences.

Through literature review and preliminary analysis, we have identified a promising mechanism developed by Chiarandini et al. [12] and developed a novel variant that incorporates additional fairness considerations for pregrouping. Our evaluation aims to compare these mechanisms against the current BEPSys algorithm and provide recommendations for which mechanism should be used going forward.

Our evaluation shows that both mechanisms significantly outperform the BEPSys mechanism in both quality of outcome and runtime. Our novel Fair mechanism successfully prioritizes students without grouping preferences. However, it may undesirably dissatisfy pregrouped students in some instances. The implementation of our objective function, although otherwise desirable, is shown to be defective in larger scenarios where it fails to find the fully optimal solution.

The remainder of this thesis is structured as follows: Section 2 reviews related work in matching theory and group formation. Section 3 formally defines our problem model. Section 4 translates our problem model into a *mixed integer linear program*. Section 5 presents our experimental evaluation using both historical and synthetic data. Finally, Sections 6 and 7 discuss our findings and outline directions for future work.

2 Related Work

In this chapter we present and discuss related work, namely the prior mechanism BEPSys and literature review.

2.1 Prior mechanism: BEPSys

In this section we briefly detail the BEPSys mechanism. Please refer to our prior work in Appendix B – AIDM Project: Improving BEPSys for a more detailed treatment.

The BEPSys mechanism accepts instances with the following inputs: students, project topics, the common capacity of each project topic (number of groups), the group size bounds (each group must have at least ℓ and at most u students), the totally ordered and complete project topic preferences of each student, grouping preferences of students (a set of at most $u - 1$ preferred students).

The mechanism's procedure consists of two steps: (1) students are partitioned into groups and (2) the groups are matched to projects.

The grouping procedure of the mechanism consists of three sequential steps.

1. The initial step of the grouping determines mutual cliques from the grouping preferences.
2. Additional students are added to the cliques based on their grouping preference. Preference is given to students whose grouping score is highest for the non-full groups. Grouping score is determined by the number of virtual directed edges there are between them and the group, where a virtual directed edge is represented by the student in the students' grouping preference. For example, a group consisting of a mutual clique with students s_1 and s_2 will have a score of s_2 for student s_3 if exactly one of the students s_1 or s_2 included student s_3 in their grouping preference and student s_3 included exactly one of the students s_1 or s_2.
3. Students without grouping preferences are added as tentative groups of size one. Each group's project preference is determined by aggregating the members' project preferences into a single aggregated project preference using Borda counts. The groups are then iteratively combined with a greedy method using the project preference likeness score. The score is based on the mean squared error measure. The process continues until all groups are valid according to the group size bounds, if this is not possible some groups are disbanded and the process continues until the termination condition is met.

The formed groups are then matched to project topics by treating it as an optimal minimum-cost flow and finding the optimal solution using an efficient, optimal method.

2.2 Literature review

Our problem involves grouping and matching students with projects. Students must express preferences over the project topics and also able to indicate with whom they wish to be grouped together. The goal is to find matchings that are optimal in terms of student satisfaction but not at the expense of some students, so fairness is an important consideration as well.

The field of matching literature is quite broad [19] and typically involves two sides that need to be matched together, e.g. students and project topics. A further distinction between matching problems is who has preferences over who. In one-sided markets, only one side has preferences over the other. In a two-sided matching problem, each side has preferences over the other. For the one-sided markets, the typical focus is on optimality of the matching based on some criterion. For the two-sided markets, the desired solution concept is *stability* – for which many weaker, stronger, and setting specific variations exist. Typically, stability implies that the matching outcome must be such that no two unpaired participants (e.g. a student and a supervisor) prefer each other over their assigned pairing and thus may *deviate* from the matching. Although two-sidedness exists in our setting to some extent, our centralized matching scheme is dictatorial in the sense that it is not possible to unilaterally deviate from the matching outcome. Furthermore, in our setting it is undesirable for project topics, or by extension their supervisors, to express preferences over students because each student should be given an equal chance in doing a project they desire.

Monte and Tumennasan [20] proposed an algorithm for a similar problem model as ours where their solution concept is *strategy-proofness.* The algorithm is based on the well-studied serial dictatorship mechanism (also known as immediate acceptance), called Serial Dictatorship with Project Closures (SDPC). Although their model does not consider project topics to have capacity for multiple groups, there is a reduction available by assigning students to project topic *slots* instead of directly to the topics. In [18] Kamiyama proposes a modification for the SDPC mechanism to handle preferences where not all projects are deemed *acceptable,* that is an agent prefers to remain unmatched rather than being matched to a project that is *unacceptable.* These algorithms do not include grouping preferences, nor are they compatible with our desired optimization criterion. Nevertheless, we have extended the SDPC mechanism with the concept of slots and included this algorithm in our evaluation.

Soft lower quotas, where the college or course is either completely unmatched or meet the lower quotas, are a feature of the Hungarian college admissions problem (two sided, stable) studied by Biro et al in [7]. Although this problem also encompasses *common* quotas – quotas over a set of colleges, thus the sum of some local quotas of colleges in the set is higher than and must not exceed the common quota. Additionally, in a later paper, an *integer programming* approach is considered [2]. Building upon their experience with solving two-sided, stable matching problems with quotas using integer programming, in another paper, Ágoston et al [3] consider the problem of assigning students to company projects with hard lower and upper quotas with additional lower and upper quotas on specific *types* (nationality, gender, etc) – so called *distributional constraints*. While in this thesis we do not consider distributional constraints, they may become a requirement in the future.

A related matching problem is the aptly named *student-project allocation* (SPA) problem. It involves students that need to undertake *individual* projects. Topics are offered by supervisors, whereby a supervisor may offer more topics than they can actually supervise so as to offer more choice to the students. The problem consists of finding an assignment under some criteria – for example student welfare or stability in case of two-sided matchings – such that the supervisor capacities are not violated. The supervisor capacities are also an important factor to consider, dynamically distributing supervisor capacity over the topics can result in better matching outcomes for the students and/or fairer workload distribution over the supervisors.

Grouping preferences models

For the problem of allocating neighboring plots of land to (pairs) of friends, Elkind et al [15] presents some variations of the strategy-proof serial dictatorship mechanism. Burkett et al [11] considered the problem of allocating rooms (with capacity of two) to students, where students agree on a common ranking of the rooms. Yet in another, similar, setting, Nicolò et al [21] considered the problem of matching agents in pairs to undertake a project whereby students have preferences over both their peers, and projects. An interesting feature is that the preferences are dichotomous, i.e., partition objects into two equivalence classes; fellow students are classified either as *friend* or *outsider* and projects as either *good* or *bad*.

Dutta and Massó [14], and Revilla [22] consider the stability of matching in the setting of two-sided, many-to-one matching problem of assigning *workers* to *firms* whereby the *workers*' preferences include both the *firm* and potential *colleagues* – other workers matched to the same firm. Because the generalized ordinal preference model over all possible combinations of firms and fellow workers is difficult to reason over, the authors consider practical and realistic restrictions on the preferences – characterizations of types of preferences a worker could reasonably hold. Dutta and Massó present a characterization whereby the workers mainly care about one or the other type of outcome, being matched to the most preferred firm, or with most preferred colleague(s). A similar trade-off exists in our problem: does the student prefer to maximize their satisfaction with the topic obtained or prefer to work in a group with preferred colleagues? Such characterization (restriction) on preferences is called *worker-lexicographic* (*$\mathcal{W}$-lexicographic*), that is when an agent's preferences are dominated by care about who their co-workers in an outcome are. The converse restriction *firm-lexicographic* (*$\mathcal{F}$-lexicographic*) is when the agent's preferences are mostly determined by their preference over the firms. Furthermore, when considering worker-lexicographic preferences, the authors consider two cases: when workers form *couples* and prioritize being matched *together,* and the case where their preferences over each other are drawn from a common ranking. In a related work, Revilla builds on the work of Dutta and Massó for the scenario of *groups*, most importantly, he considers practical restrictions on preferences when workers have certain sets of workers they want to work with, be it regardless of the firm or on the contrary, a preference for a firm is conditional on certain set of colleagues (*$\mathcal{F}$-essentiality).* Interestingly, for these models Revilla assumes the preferences to be *separable* (not to be confused with the synonymous concept presented by Dutta and Massó) whereby the preferences can be decomposed into preferences over the two attributes. This is similar to the preference model of BEPSys, where the students submit two semi-independent – some expectations exist but no guarantee is given – preferences: one over project topics and another for their preferred groupmates.

Hogan [17] in her thesis on two-sided, many-to-one, stable matching of (groups) of women to sororities, presents an idea of group *offer numbers* with which a group indicates at least how many members must remain together in the final matching. The two-sided, many-to-one, stable matching problem with groups and *offer numbers* is considered by Shimada et al in [23]. They present and evaluate a multi-objective optimization problem formulated as a *mixed integer linear program*, where they try optimizing on stability, welfare (simple sum of utilities) and a global group offer number.

Group formation as a problem is investigated in the field of group formation games, usually as hedonic coalition games – *hedonic* because the agents are considered to care about their own benefit. One such paper is that of Aziz et al [6], therein coalition formation games that are *B-hedonic* or *W-hedonic* is considered. Put simply, a game is *B-hedonic* if the agent compares coalitions (groups of agents) by comparing the most preferred peers within those groups. And analogously, *W-hedonic* if the agent compares coalitions by comparing the least preferred peers between the groups. Of the two types of games, it is tractable to both compute and verify pareto-optimal *W-hedonic* games.

The preferences models presented are interesting and provide an avenue for further research, in this work we will limit ourselves to the preference model of BEPSys as that gives us a set of historical instances to evaluate on.

Branzei et al [10] present an approach to dealing with separate preferences by defining a numerical relation between them and including externalities such as peer effects as additive utilities in a utilitarian model. Bodine [8] extracts peer preferences from social networks and also captures this in an additive, utilitarian model.

Fair and optimal matching of students to projects with pregrouping

To the best of our knowledge, our exact problem has not been considered but the work of Chiarandini et al [12] comes the closest. There, the authors are tasked with solving a very similar problem setting where students are allowed to express ordinal preferences only over the offered project topics, such that the students' *welfare* (satisfaction) is optimized but fairly. Grouping wishes are handled exogenously by allowing such students to pre-form groups and participate as undividable units in the matching, rather than as individual students. The students' satisfaction is assumed to depend on how they value their own outcome with respect to project topic and how their outcome relates to the outcomes of other students. The authors, following the example of Anwar and Bahaj [4] and the NP-hardness proof of Arulselvan [5], formulate the problem as a *mixed integer linear program* and furthermore, present and evaluate a range of objective functions from the literature. Among the evaluated objective functions, most important for us is the *profile*[1]-based optimization approach based on finding the outcome that is $leximin$ ordered before any other – put simply, minimizing the number of students with the rank r, then $r-1$ and so on.

The solution and model of Chiarandini et al suffices for our use-case of project-based courses, however, in our setting it is not desirable to always allow students to pre-group with friends – only if this is not detrimental to the matching outcome of students who care about their project topic preferences the most and/or do not or cannot form groups in advance. The model of Chiarandini does not consider satisfying grouping wishes to be part of the overall satisfaction of the outcome, which we consider to be an additional level of fairness – a group of friends should, in general, be less dissatisfied with a less desired project than an individual student with only preferences over the project. Furthermore, as is also noted by the authors, students that are grouped ahead of the matching may have an unintended advantage in the matching scheme but their experiments with past datasets did not confirm this possibility. In our setting we consider obtaining a desired group composition to be part of satisfying preferences and balance with satisfying project topic preferences of agents who prioritize having their project topic preferences satisfied.

Conclusion

We based our thesis on the foundation created by Chiarandini et al which we will modify and extend to support more flexible pregrouping variants and to include a notion of fairness between the students who create groups ahead of time and those that do not.

[1] A profile of the matching outcome is a list-based, histogram-like representation of the outcome. The index represents how high the assigned alternative is ranked by the agent, and its value indicate the number of agents that have obtained an assignment that they rank as such.

3 Problem Definition

Having explored the literature, in this section we formally formulate our problem definition. Only once we properly define the problem can we proceed with solving it.

We start by formulating the base of the problem definition, excluding the optimization aspect of our problem. This base specifies the problem in terms of its input, the problem instances, and its output, the valid solutions. In the second subsection we gradually develop our optimization problem and additional problem instance components.

3.1 Formal definition of the base problem

The essence of our problem is matching students to project topics to form project groups. There are some limitations: any group must adhere to group size bounds – upper and lower – to be valid and each project topic supports a limited number of groups – its capacity – modeled as *topic group slots*.

Formally, we define our problem as follows. This formulation does not include optimization yet and thus no preference model is included.

$$\begin{aligned} \text{students } & S = \{s_1, \dots, s_n\} \\ \text{topics } & P = \{p_1, \dots, p_m\} \end{aligned}$$

$$\begin{aligned} \text{group capacity of topic } c&: P \to \mathbb{N} \\ \text{group size lower bound } \ell&: P \to \mathbb{N} \\ \text{group size upper bound } u&: P \to \mathbb{N} \end{aligned}$$

$$\text{topic group slots } T = \cup_{p_i \in P} \left\{t_{p_i,1}, \dots, t_{p_i,c(p_i)}\right\}$$

$$\begin{aligned} &\text{The matching } \mu \subset S \times T, \text{s.t:} \\ &1.\ \forall s \in S, \quad |\mu(s)| = 1 \\ &2.\ \forall t \in T, \quad \ell \leq |\mu(t)| \leq u \\ &3.\ \mathrm{M} \ni \mu \end{aligned}$$

This model is based on that of Chiarandini et al [12] but with some differences. Other than notation, our model differs by how we handle pregrouping. Our model models students as single individuals, whereas the original Chiarandini model opts for atomic units of students with a cardinality representing the number of students in that pregrouped unit. Modeling students individually permits more flexibility in our model for defining different pregrouping models. More on this in the following sections.

Clarifications

We tried to keep the model definition brief and concise, but we want to add the following additional clarifications.

- Our problem definition assumes problem instances to have sufficient capacity to match all students: $\sum_{p \in P} c(p) \cdot u \geq |S|$
- The groups *slots* for topics are a virtual concept to add group capacities to project topics. It also does double duty as a placeholder for the eventual group, because the model does not assign students to groups and groups to topics. Instead, students are assigned to groups that are bound to a topic, the topic group slot.
- Furthermore, because it is more intuitive, we sometimes directly refer to assigning topics to students rather than their slots; the slots are an implementation detail and there is no difference between the slots of a topic. Thus, when we later refer to project topics directly by p or P rather than their corresponding set of topic group slots t or T, imagine a mapping between them.

3.2 The optimization problem

We want the solutions to be not merely valid but for them to be as optimal and fair as possible. For this thesis and the current problem domain, we define optimality mean the students' satisfaction for the benefit of their educational obtainment and motivation as well as reducing the course administration workload caused by bad matchings.

To formulate the optimization problem, we need to define what the students want, their preferences, and how to satisfy those preferences optimally. These aspects we will analyze and formulate in this section.

3.2.1 The preference model

We begin with the preferences model; what the students can express preferences over and their representation.

3.2.1.1 The broad preference model

We start with the most general and concise formulation. In this model, we assume each student has preference over all possible, valid matching outcomes. It allows the students to express not only which project topic they prefer to work on but also with whom, and even which groups and topics other students get. Formally, under this preferences model, each student s declares a total ordering $\succcurlyeq$ over all possible, valid matchings.

Although very expressive, it is too broad and thus not tractable or even practical to elicit for any serious problem instance. Furthermore, from the perspective of fairness, it is undesirable to allow students directly influence the matching outcome of others. But, even if we restrict the preference to outcomes involving themselves only, i.e.: their individual outcomes of project topic and group composition, it is still too impractical for our average problem instance.

3.2.1.2 The split preference model

A further restriction of the model is achieved by splitting the preference model into two. One for the project topics and another for the grouping. Doing so removes the combinatorial explosion at the expense of expressiveness and fidelity of the preferences. The tradeoff between the two aspects of the outcome (topic, group) is then left to the objective function rather than the individual student.

3.2.1.3 The topic preferences with pregrouping model

We restrict the above model even more by limiting the group preferences to a single group composition, thereby allowing the student to express their desire to be paired with a specific set of students (respecting the bounds on group size).

Here too a question of fairness arises when students can directly influence the outcomes of other students. To resolve it, we limit the preference model once more to only allow mutual groupings. That is a submitted group composition preference will only be valid if it is mutual, i.e., forms a clique. This we call pregrouping, as these students group in advance of the matching.

Formally, we define this model as follows. Each student has two preferences, one is a *total* and *complete* ordering over project topics, the other is a simple subset of students that he/she desires to pregroup with.

$$\begin{aligned}
&\forall s \in S: \\
&\cdot\ \exists \succ_s \in \Pi \quad \text{s.t}\ \forall p_1 \neq p_2 \in P:\quad p_1 \succ_s p_2 \vee p_2 \succ_s p_1 \\
&\cdot\ \exists \gamma_s \subset S \quad \text{s.t.}\quad \gamma_s = \emptyset \ \vee\ (2 \leq |\gamma_s| \leq \mathcal{u} \ \wedge\ s \in \gamma_s) \ \leftrightarrow\ \gamma_s \in \hat{G}
\end{aligned}$$

$$\text{project topic preferences}\quad \Pi = \{\succ_{s_1}, \dots, \succ_{s_n}\}$$

$$\begin{aligned}
&\text{set of pregrouping wishes } \hat{G} \subset \mathcal{P}(\mathrm{S}) \text{ across all students},\quad \text{s.t.}: \\
&\cdot\ \forall \hat{g} \in \hat{G}:\ 2 \leq \hat{g} \leq \mathcal{u} \\
&\cdot\ \cup_{\hat{g}\in\hat{G}} \subseteq S \ \wedge\ \cap_{\hat{g}\in\hat{G}} = \emptyset \\
&\cdot\ \forall \gamma_{s_1} \neq \gamma_{s_2}:\quad \gamma_{s_1} = \gamma_{s_2} \ \vee\ \gamma_{s_1} \cap \gamma_{s_2} = \emptyset \\
&\cdot\ \forall \hat{g} \in \hat{G}:\ \cap_{s\in\hat{g}}\, \gamma_s = \hat{g} = \cup_{s\in\hat{g}}\, \gamma_s
\end{aligned}$$

Clarifications

- Each student defines a total, complete ordering over projects (no ties). We write $p_1 \succ_s p_2$ to indicate that student s prefers project topic p_1 over p_2. More precisely, the student prefers any project topic slot associated with p_1 over those associated with p_2 whilst being completely indifferent between the topic slots associated with any given project topic, formally: $\{t_{p_1,1}, \dots, \mathrm{t}_{p_1,\mathrm{c}(\mathrm{p}_1)}\} \succ_s \{t_{p_2,1}, \dots, \mathrm{t}_{p_2,\mathrm{c}(\mathrm{p}_2)}\}$
- Π is the set of all project topic preferences in the problem instance.
- Each student (s) may express their pregrouping wish (γ_s), which is either empty (no pregrouping wish) or a set of no more than $\mathcal{u}$ (including s self) students. This pregrouping wish is completely mutual (each included student has identical pregrouping wish). All the pregrouping wishes are contained in the set $\hat{G}$.
- Even though the above formal definition allows for γ_s to be of all sizes between 2 and $\mathcal{u}$, in practice we disallow the $\mathcal{u} - 1$ size (submax) as we deem it undesirable for a single student to end up in a group of friends and feel not integrated.

- Although the final problem instance only contains valid pregroupings, students submit their preferences individually. Their individual input is processed, validated, pruned, and filtered at the start of the matching. Students who were included in the group preference but are not mutual are pruned out of the pregrouping input. If multiple cliques are possible, the largest is chosen. Ambiguities are resolved non-deterministically.

This preference model is similar to the previously used BEPSys mechanism. It is compatible with past problem instances and simplifies integration with existing signup systems.

3.2.2 The objective

Now with the preferences model defined, we move on to formulating the optimality criterion.

We must contend with the split preferences model: how the preferences relate to each other and how, taken together, they relate to those of others. Our objective function will have to make a tradeoff between the two and inform the students of it, so that they take it into consideration when forming their preferences.

We formulate the objective in parts, first by considering only the project topics, then the grouping model and finally, combine the two together.

3.2.2.1 Topic

A big part of students' satisfaction with the matching outcome is the allocated topic. Although we set out to make the students as happy as possible, it is beneficial to consider the problem from the other perspective: minimizing the number of *unhappy* students. We do not want to optimize the outcome for the majority at the expense of a very dissatisfied minority, but neither do we want an overly compromised outcome. The difficulty lies in finding a balance between the two.

Based on this, we start by minimizing the worst assignment. This is known as the $\mathrm{minimax}$ objective. Formally, if we encode the distribution of obtained ranks in a matching as a list denoted as $\sigma: \mathbb{N} \to \mathbb{N}$, then the objective is:

$$\min \max \{ i \in \mathbb{N}^{+} : \sigma(i) > 0 \}$$

The σ is a rank profile and is represented by a list whose values are the number of students having an assignment they rank as the index of that value, like an array in programming languages. For example, the rank profile $\sigma = (4,2,1,0,1)$ represents an outcome where 4 students have obtained their 1st ranked topic, 2 students their 2nd, 1 their 3rd and 1 their 5th.

Minimizing the worst outcome within the matching is not enough as no criterion is given regarding the other students. We chose to build upon the idea of the $\mathrm{minimax}$ criterion further and expanded to the above lying ranks of the outcome. Intuitively, we minimize the number of students from the worst outcome up. This is referred to as the *generous maximum matchings* [19]. Formally, having minimized the worst possible worst rank in the matching, say k, minimize the number of students with k, subject to that minimize those with $k-1$, $k-2$ and so on.

3.2.2.2 Grouping

Let us consider only the grouping wishes. Our grouping preferences model allows students to submit grouping wishes, which we can fulfill if fully mutual. We can guarantee satisfaction of mutual grouping wishes. This is the *hard pregrouping* variant.

$$\text{hard pregrouping - } \mu \textbf{ is} \text{ consistent with } \hat{G}\text{:} \quad \forall \hat{g} \in \hat{G}, \exists t \in T \ \text{ s.t. } \ \hat{g} \subseteq \mu(t)$$

But it might not always be desirable to guarantee this, either due to feasibility of the instance or a large negative effect on the quality of outcome. In that case, we may drop the guarantee and soften the above condition to *not every*. We call this variant *soft pregrouping*.

$$\text{soft pregrouping - } \mu \textbf{ should} \text{ be consistent with } \hat{G}\text{: } \exists \hat{G}' \subseteq \hat{G} \ s.t. \forall \hat{g} \in \hat{G}', \exists t \in T \ \text{ s.t. } \ \hat{g} \subseteq \mu(t)$$

3.2.2.3 Satisfying both preferences

To combine our insights on separate satisfaction of preferences, we need to consider the context and domain of our problem. Based on domain expert's knowledge, we assume two types of students. Topic-minded and group-minded. That is, those who wish to obtain their most desired project topic, and those who wish to at least obtain their desired group composition. It is for that latter type that pregrouping is made possible. Therefore, the students who indicate a preference for pregrouping are considered to be of that latter type.

Although the educational goals of some types of courses include soft skills related to group work such as cooperation and team building, and these are thought to better achieved when working together with new people, we think that nonetheless it is better to accommodate and allow pregrouping. Students that have strong grouping wishes will try to achieve their desired grouping through strategic behavior, such as picking specific project topics and aligning topic preferences as well as attempting to trade places after matching, causing additional workload on the staff. Thus, it is better to allow, accommodate and manage this use-case.

There are many ways both preferences can be satisfied. The choice of which depends on the context of the matching problem. A couple of variants are presented below:

- **Simple** - *satisfy both independently. Lock in the pregroupings and then form groups whilst assigning topics. This is the chosen approach of Chiarandini et al for the Southern Denmark University group project matching.*
- **Fair** (discount) – *pregrouping contributes to the overall satisfaction, compensating the topic allocation. The objective function can be more relaxed assigning topics to pregroupings.*
- **Fair** (dynamic) – *pregrouping should not be at the expense of topic-minded students. This is our chosen variant as it best fits our matching setting, where pregrouping is a bonus.*
- **Conditional** – *students have conditional pregrouping wishes, the pregrouping should only be kept together if can be matched to specific topics, otherwise drop the pregrouping. These topics can be simply the top x of topic preferences, then the pregrouping is only maintained if the group can obtain one of their shared top x (say top 5) choice. Alternatively, these groups should submit an additional topic preferences as part of the pregrouping signup step, thus if pregrouping & topic preference cannot be granted, the solo-topic preferences are used instead.*

Of these variants we choose the simple and fair (dynamic). Both variants require an objective function for the topic allocation. For this function, we use the iterative minimax to minimize the number of students with the least good outcomes. The chosen variants can be implemented in addition and without changes to project topic-centric objective functions.

4 Mechanisms: MILP formulations

In the previous section we developed and defined our problem formally. In this section we translate our problem model into a mixed integer linear program (MILP) formulation that can be implemented and solved by a solver such as Gurobi [16]. Just like our problem definition, our model is built up modularly and consists of the base model, the project preferences (optimization) and the pregrouping models.

We present two mechanisms. Both are constructed from the pieces we outline below. The first mechanism is our modification of the Chiarandini mechanism [12] with a more flexible pregrouping model. The second model is our novel mechanism constructed from the modified Chiarandini mechanism, implementing the *fair (dynamic)* variant mentioned in section 3.2.2.3 which we name the "Fair" mechanism.

The rest of this section is structured as follows. First, we introduce the base model, then the optimization part followed by the grouping variants: *hard*, *soft,* and *conditional*. Then we conclude by defining the procedure and additional constraints for *fair* mechanism.

4.1 The base model

The formal problem model presented in the previous chapter is sufficient to produce a simple MILP model. Just as there, the basis is devoid of optimization concerns and merely defines valid outcomes. It does not include (pre)grouping or preferences and is a good foundation for modular objectives and constraints, which are presented later.

Let $\mathcal{X}$ be the basis model, defining valid outcomes:

$$\mathcal{X} = \begin{cases} \sum\limits_{t \in T} x_{st} = 1 & \forall s \in S \\ \sum\limits_{(s,t) \in S \times T} x_{st} \geq \ell \, y_t & \forall t \in T \\ \sum\limits_{(s,t) \in S \times T} x_{st} \leq u \, y_t & \forall t \in T \\ x_{st} \in \{0,1\} & \forall (s,t) \in S \times T \\ y_t \in \{0,1\} & \forall t \in T \end{cases}$$

The model consists of two decision variables: x_{st} and y_t. Where x_{st} determines the assignment of a student s to a project topic slot t and $\sum_{t \in T} x_{st} = 1$ constraint enforces that each student is matched to exactly one project topic slot. The decision variable y_t controls the opening of a project topic slot t – a project topic slot is said to be *open* if it has (or will have) a valid number of students assigned to it. This decision variable is necessary to define other constraints.

Then there are the two constraints to enforce the lower and upper group size bounds, presented here in combined form: $\ell \, y_t \leq \sum_{(s,t) \in S \times T} x_{st} \leq u \, y_t$. Note that the constraint is only effective for *open* project topic slots.

This concludes our definition of the base model. Turning this base model into an optimization model is achieved by adding an objective function, z:

$$\begin{array}{rl} min & z \\ s.t. & \mathcal{X} \end{array}$$

4.2 Objective function

Earlier, in our problem definition, we discussed our choice of objective function. We opted for the *generous maximum matchings* objective, a profile-based optimization method. In this subsection we extend our base model to an optimization model with preferences and chosen objective function.

4.2.1 Profile-based optimization

Profile-based optimization considers the *profile* of the outcome, rather than a scalar value, along with a definition for ordering over the profiles. This method allows for more precision than a single scalar value, e.g. the AUPCR metric (see experimental analysis), as such metrics inherently encode trade-offs in the aggregated values which are not necessarily desirable. An outcome with a better score may not always be an improvement.

4.2.1.1 The profile

In general, a profile of the matching outcome is an overview of the outcome whereby, per rank, the number of students having obtained a project they rank correspondingly, is presented. Visually, it is a distribution of the number of students over the ranks. More relevantly, mathematically it is a vector, or list, of the number of students having obtained a project they rank corresponding to the index in the profile vector, in their project topic preferences.

For example, the rank profile $\sigma = (4,2,1,0,1)$ represents an outcome where 4 students have obtained their 1st ranked topic, 2 students their 2nd, 1 their 3rd and 1 their 5th.

4.2.1.2 Comparing profiles – the $\mathrm{leximin}$ order

To optimize over rank-profiles, we need to define what constitutes a better profile. A comparison that exactly implements our chosen objective function is the $\mathrm{leximin}$ order, so called because it picks the profile that is the smallest when ordered lexicographically. We are using the *distribution approach* for profile-based optimization as set out by Chiarandini et al [12].

As a brief aside, the name is technically incorrect when applied to the distributional rank profile representation. For example, given two rank profiles $\sigma_1 = (4,3,2,1)$ and $\sigma_2 = (1,2,3,4)$ where σ_1 is clearly better but lexicographically σ_2 is considered before it and thus the winner. Contrast this with the (non-distributional) sorted vector representation, the previous two profiles then become $\sigma_1 = (1,1,1,1,2,2,2,3,4)$ and $\sigma_2 = (1,2,2,3,3,3,4,4,4,4)$. Now indeed, σ_1 comes lexicographically before σ_2. Nevertheless, for the sake of consistency we will keep referring to this ordering as $\mathrm{leximin}$ even when applied to the distributional variant of the profile.

The formal definition is as follows:

$$\begin{aligned} \sigma_1 <_{\mathrm{leximin}} \sigma_2 \quad & \text{iff} \quad \exists i \geq 1, \text{such that } \forall j > i\colon\ \sigma_1(j) = \sigma_2(j) \wedge \sigma_1(i) < \sigma_2(i) \\ \sigma_1 =_{\mathrm{leximin}} \sigma_2 \quad & \text{iff} \quad \forall j\colon\ \sigma_1(j) = \sigma_2(j) \end{aligned}$$

In words, the profiles are compared pairwise from the worst to best rank. A profile is leximin before another if at some rank, being pairwise equal for all preceding (worse) ranks, it has less students assigned than in the other. Note that this comparison assumes profiles with the same number of students.

As an example, consider two profiles:

$$\sigma_1 = (1, 1, 1, 2, 6, 9)$$

$$\sigma_2 = (0, 2, 1, 3, 5, 9)$$

According to the $leximin$ order, profile σ_2 is better and is determined to be so at the second comparison, that of rank 5. Note that less students in σ_2 have obtained their most preferred allocation.

4.2.1.3 Efficient LP formulation – ordered weighted function (OWA)

The $\mathrm{leximin}$ ordering approach can be implemented efficiently by means of a weighting scheme for the ranks, the so-called *ordered weighted averaging (OWA)* [24] via [12]. The scheme constructs weights for a given instance such that by minimizing the sum of weighted ranks, the solution is $\mathrm{leximin}$ optimal. We present the scheme briefly and refer the interested reader to the source material.

The objective is the sum of weighted ranks:

$$\min \sum_{i=1}^{\Delta} F_i$$

Where for each rank i:

$$F_i = \begin{cases} \frac{i}{8} \cdot w_i \cdot \sigma(i) & \mathrm{i} \leq 8, \\ 10000i & \text{otherwise} \end{cases}$$

$$w_i = \begin{cases} \dfrac{\mathrm{B}^{n-1}}{(1+\mathrm{B})^{\mathrm{n}-1}} & i = 1 \\ \dfrac{\mathrm{B}^{n-i}}{(1+\mathrm{B})^{\mathrm{n}+1-\mathrm{i}}} & \text{otherwise} \end{cases}$$

and:

$$\mathrm{B} = \frac{1}{8} - 0.001$$

The weighting scheme has one downside however, it does not scale well to many ranks. For ranks larger than 8 it will encounter numerical problems, such as stability and overflows, hence the scheme is applied only till rank 8.

4.3 Grouping variants

In this subsection we expand our model with pregrouping, this constitutes our extension of the original Chiarandini model. In our problem definition section, we have defined three pregrouping variants: *hard*, *soft*, and *conditional*. For the latter two we have two implementation versions, one as a constraint and another as an objective function. Below we present both. We make use of the constraint variant due to our choice of objective function.

4.3.1 *Hard* pregrouping

In the *hard* pregrouping variant, the pregroupings are guaranteed. Its constraint form is a simple linking of the individual members' decision variables. Remember, each student can submit at most one pregrouping preference, the pregrouping is mutually agreed, consists of at most $\mathcal{u}$ members and none of the groups in $\hat{G}$ overlap. In other words, linking the decision variables will not *directly* make the problem instance infeasible. Only in tiny instances can this constraint lead to infeasible matchings.

Formally, the hard grouping constraint is defined as follows:

$$\begin{aligned} &\forall \hat{g} \in \hat{G},\ \ t \in T: \\ &x_{s_i t} = x_{s_j t} \qquad \forall s_i, s_j \in \hat{g} \end{aligned}$$

However, that is not efficient to implement. As an optimization, we can reduce the number of necessary constraints by exploiting transitivity. By picking some single student s_1 within the group $\hat{g}$ and link all other members to his/her decision variables we can reduce the number of required constraints:

$$\begin{aligned} &\forall \hat{g} \in \hat{G},\ \ t \in T: \\ &x_{s_1,t} = x_{s_j,t} \qquad \forall s_j \in \hat{g} \setminus \{s_1\} \end{aligned}$$

4.3.2 *Soft* grouping

The *soft* variant does not guarantee pregroupings, (some) pregrouping(s) are dropped if the problem instance is infeasible otherwise. Although *hard* pregrouping is almost always feasible, it may be desirable to drop pregroupings if that is better for the overall outcome, such as done by the *fairness* mechanism. In the context of our work, this variant is necessary for the *fairness* mechanism.

The constraint for the soft variant is more complex. We use binary decision variables $v_{\hat{g}} \in \{0,1\}$ to indicate the dropping ($v$iolation) of each pregrouping $\hat{g} \in \hat{G}$.

The constraint itself is as follows:

$$\begin{aligned} &\forall \hat{g} \in \hat{G}: \\ &(v_{\hat{g}} = 0) \rightarrow \left(x_{s_1 t} = x_{s_j t}\right) \qquad \forall t \in T,\ \ s_j \in \hat{g} \setminus \{s_1\} \end{aligned}$$

Ensuring a lower bound of satisfied pregroupings is accomplished with an ϵ-constraint. Counting violations towards a penalty on the objective function is also possible. The epsilon constraint is especially useful when the full precision of the objective function representation is needed, such as with the *OWA* objective function.

Objective function version, where z is some objective function:

$$\min z + \sum_{g \in \hat{G}} 1000\, v_{\hat{g}}$$

The ϵ-constraint, to be used in addition to the main constraint:

$$\sum_{\hat{g} \in \hat{G}} v_{\hat{g}} \leq \epsilon$$

Implementation notes

The conditional constraint is not a normal linear programming constraint. Fortunately, the Gurobi solver allows expressing such constraints through the `globalIndicatorConstraint` API function, which, given a decision variable and a constraint, generates and adds the necessary constraints to achieve equivalent effect.

4.3.3 Conditional grouping

Pregrouping is for students who prioritize their desired grouping over the best possible topic allocation. However, the preceding variants merely lock the students into their desired group compositions and could cause them to incur an unbounded loss in allocated project topic satisfaction. This is even more likely with our fairness procedure, described below, where pregroupings have a lower priority in topic allocation.

Not all students may wish to remain together regardless of an undesirable topic, for them we propose a conditional grouping variant. It allows the students to submit a pregrouping preference that is conditional on some topics. For this implementation it is simply a limit in the form of a rank, say k, up to which they find project topics acceptable to remain as a group. If the *pregroup* cannot obtain their shared top-k project topics, the pregrouping is dropped. The hope is then that individual students of that group can obtain a better project topic outcome on their own than they would as a group.

The implementation closely resembles that of the *soft-grouping* scheme. For conditional variant, the conditional linking constraint is applied only to the top-k choices of the group and pregrouping decision variable $v_{\hat{g}}$ must also imply assignment of one of the top-k choices if it is 0. In other words, if the group is together, it must be allocated one of their top-k choices.

The constraint:

$$\begin{aligned}
&\forall \hat{g} \in \hat{G}: \\
&\quad (v_g = 0) \rightarrow \left(x_{s_1 t} = x_{s_j t}\right) \qquad && \forall t \in \mathrm{top}(\succ_{s_1}, k)\,,\ \ \forall s_j \in \hat{g} \setminus \{s_1\} \\
&\quad (v_g = 0) \rightarrow \left(\Sigma_{t \,\in\, \mathrm{top}(\succ_s, k)}\, x_{st} = 1\right) && \forall s \in \hat{g}
\end{aligned}$$

Where $\mathrm{top}(\succ_s, k)$ represents a function that given the topic preferences $\succ_s$ of some student s, gives a set of the top k ranked alternatives.

Analogously to the *soft-grouping* scheme, the objective function penalty:

$$\min z + \sum_{\hat{g} \in \hat{G}} 1000\, v_{\hat{g}}$$

And the ϵ-constraint:

$$\sum_{\hat{g} \in \hat{G}} v_{\hat{g}} \leq \epsilon$$

4.4 The Chiarandini mechanism

With the base model, grouping scheme and objective function defined, we can now properly present the full models of the mechanisms.

Because we have based our model on that of the original Chiarandini mechanism model and modularized it, constructing the Chiarandini mechanism is a simple exercise in combining the base model, the objective function and one of, if any, the grouping variants. The hard pregrouping variant simulates the grouping model of the original Chiarandini mechanism.

4.5 The Fairness mechanism

Before we present the Fairness mechanism, some context. As a reminder, our chosen criterion for fairness between the two types of students is that *pregrouping* may not be at any expense of the *solo* students. This criterion is applicable to courses where the learning goals include interpersonal development, which is best achieved by working with new people.

Earlier we have roughly divided the students into two types, those that prefer obtaining their best topic pick, which we will call *solo students*, and those who prefer obtaining a specific group composition, e.g., to work with friends, the *pregrouping* students. Under our preference model, we assume only the students who consider themselves to be of the latter type to submit a pregrouping preference. The categorization of students into these two types is the basis of this mechanism.

The procedure of the Fairness mechanism consists of two rounds. One initial round with all students being treated equally – without regard to or even considering their grouping wishes, simulating a scenario without pregrouping to determine the baseline outcome for the *solo* students. And a second, final, proper round with the grouping wishes handled according to the chosen grouping variant and the project topics are allocated such that the final topic allocation to the *solo* students is *at least as good* as the baseline outcome. The baseline is recorded as a rank profile, it is anonymous and not bound to individuals.

The *at least as good* criterion for the baseline is *equality* or *pareto better.* Put simply, an outcome is pareto better if at least one student is better off without anyone else becoming worse off. In our case we compare profiles of ranks, so we do not compare outcomes of individuals but consider the outcomes in terms of number of students assigned to ranks only. Returning to one of our previous examples:

$$\sigma_1 = (1, 1, 1, 2, 6, 9)$$

$$\sigma_2 = (0, 2, 1, 3, 5, 9)$$

$$\sigma_3 = (1, 2, 2, 2, 5, 8)$$

neither σ_1 or σ_2 is pareto better than the other. While σ_2 has one less student with rank 5 it also has one student less with a top ranked allocation. However, σ_3 is pareto *better* than σ_1, that is $\sigma_3 >_{\text{pareto}} \sigma_1$. The number of students with ranks 5 and 6 is decreased in favor of ranks 2 and 3.

4.5.1 The procedure

Both rounds make use of the same objective function z. The choice of objective function is not essential to the procedure itself.

The procedure is as follows:

1) Let $\mu_1 = M_{r1}(S, P, c, \ell, u, \Pi, z)$ be the initial matching determined solely by considering the project topic preferences (π^T) of the students.

2) Let $\sigma_{\text{baseline}} = profile(\mu_1,\ S_{\text{solo}})$ be *the baseline*, the profile of *solo* students in matching μ_1. Where $S_{\text{solo}} = \{s \in S \mid \forall \hat{g} \in \hat{G} : s \notin \hat{g}\}$.

3) Let $\mu_2 = M^{\Gamma}_{r2}(S, P, c, \ell, u, \Pi, z, \Gamma, \hat{G}, \sigma_{\text{baseline}})$ be the final matching such that both:

    a. The outcome for the *single* students of μ_2 is equal to or pareto better than of μ_1:
    $$profile(\mu_2, S_{\text{solo}}) \geq_{\text{pareto}} profile(\mu_1, S_{\text{solo}})$$

    b. The grouping of students in μ_2 is consistent with $\hat{G}$ under the pregrouping variant Γ. For example, for the hard grouping variant the following should hold:
    $$\hat{G} \subseteq G(\mu_2)$$

4) Done (if feasible)

The second round consists of solving the instance with the pregrouping and the pareto (with respect to the baseline from the first round) constraints applied. The pareto constraint is outlined directly below in subsection 4.5.1.1. For the pregrouping constraints, please refer to subsection 4.3.

4.5.1.1 The pareto constraint

Conceptually this constraint is simple. It states that in the final matching, for any arbitrary rank r, the number of students having obtained r or better, must be at least par the baseline.

Let $worst(\sigma) = \max\{\, i \in \mathbb{N}^{+} : \sigma(i) > 0\}$ indicate the worst rank present in σ. We define the constraint using a cumulative sum per rank as follows:

$$\forall i \in \{\, 1, \ldots, worst(\sigma_{baseline})\,\}:$$
$$\sum_{j=1}^{i} profile(\mu_2, S_{solo})(j) \geq \sum_{j=1}^{i} profile(\mu_1, S_{solo})(j)$$

5 Experimental evaluation

In this section we empirically evaluate and compare the performance of the Chiarandini, our Fair and the (prior) BEPSys mechanisms in terms of their outcome quality and runtime performance. We structure this section by first stating our overarching research question and evaluation goals, then narrowing into specific evaluation questions. In the subsequent subsections, we present the metrics for our evaluation, describe our experimental setup, and analyze the results.

5.1 Evaluation Goals

Our main research question is: *how can we improve the group project matching algorithm of Project Forum to improve match quality while maintaining acceptable runtime performance for large instances?*

Through our literature review and analysis, we have identified a promising mechanism (Chiarandini) and developed a novel variant (Fair) that incorporates additional fairness considerations for pregrouping. This evaluation aims to:

- Compare the new mechanisms against the (prior) BEPSys algorithm
- Compare the Chiarandini and Fair mechanisms to identify their respective strengths, weaknesses, and trade-offs
- Provide recommendations for which mechanism should be implemented in future deployments

To answer these questions, we utilize two complementary sources of data:

- *Historical instances* from TU Delft and SDU (made available by Chiarandini et al.). These represent real-world scenarios with authentic preference structures and constraints. Results on these instances should provide strong indicators of how the mechanisms will perform in the future.
- *Synthetic instances* generated through our problem generator. These allow us to systematically vary parameters, explore corner cases, and conduct sensitivity analyses that would be impossible with only historical data.

This approach allows us to evaluate both practical performance and theoretical properties of the mechanisms.

For each data source type, we formulate specific questions that guide our experimental design:

Historical Instances:
1. How do the new mechanisms compare to the prior BEPSys algorithm in terms of match quality and runtime performance?
2. How does our Fair mechanism compare to Chiarandini in our setting, and does it achieve its design goals regarding fairness?
3. What are the effects of relaxing pregrouping size restrictions on outcome fairness and optimality?

Synthetic Instances:
1. How do our mechanisms scale with increasing instance sizes?
2. What is the effect of over-provisioning instances with project topics?
3. How do group size bounds impact mechanism performance and outcome quality?
4. What influence does pregrouping have on match outcomes and mechanism performance?

Each of these questions will be addressed through experiments detailed in the following, corresponding sections.

5.2 Objective function

To evaluate our mechanisms, we must distinguish between two related but distinct concepts:

- The objective function used within the mechanisms themselves to find optimal matchings
- The evaluation metrics we use to compare the quality of matchings produced by different mechanisms

For our mechanisms, we selected the *leximin* ordering as our objective function (detailed in Sections 3.2.2 and 4.2.1). This choice was based on both theoretical properties and preliminary experimental evaluation using TU Delft instances. The leximin ordering prioritizes minimizing the number of students with the worst ranks, then the second-worst ranks, and so on - effectively implementing a "generous" approach that prevents any student from receiving disproportionately poor outcomes.

It's important to note that we are not evaluating alternative objective functions as part of this study. Our focus is on comparing the mechanisms using this objective function. However, the experimental framework we've developed can readily be adapted to evaluate alternative objective functions.

5.3 Metrics

In this section, we present the metrics we use and have considered using to evaluate and compare the matchings produced by our mechanisms.

While runtimes are straightforward to measure, quantifying outcome quality is more complex (see sections relating to the objective: 3.2.2 and 4.2.1). An ideal metric is intuitive[2], sensitive to small variations and incorporates the desired criteria of optimality and fairness – for which there is no single clear way to represent it. Although the objective function meets the latter, it fails the former. For our evaluation we require metrics that are easy to understand, compare and able to show small differences. Since no single metric perfectly captures all these aspects, we use several complementary approaches. These are outlined below.

Our analysis is divided into two parts: project satisfaction metrics and pregrouping satisfaction metrics.

5.3.1 Project satisfaction

In our ordinal preference model, a student's satisfaction is determined by the rank position of their assigned project. To enable comparison of the results, we require functions that aggregate these ranks into a presentable format. The following subsections outline the potential metrics and approaches we have identified.

5.3.1.1 Worst Rank

The worst obtained rank in a matching outcome serves as a simple indicator of the least satisfactory allocation. While not representative of the overall matching quality, it provides a basic fairness measure and complements well with more comprehensive metrics, such as the sum of ranks.

Formally the metric is defined as follows (we will refer to it by the latter form):

$$\max\{\, i \in \mathbb{N}^+ : \sigma_\mu(i) > 0 \,\} = \text{worst}(\sigma_\mu)$$

5.3.1.2 Sum of Ranks

The sum of ranks is a simple, intuitive metric calculated by adding up all obtained ranks in a matching. Solutions with a smaller sum of ranks are considered better.

Formally the metric is defined as follows:

$$\sum_i i \cdot \sigma(i)$$

While lacking inherent fairness properties and cannot be used for different sized cohorts, it works well for quick “at a glance” comparisons especially when used alongside other metrics.

5.3.1.3 Objective function

The problems of the sum of ranks metric can be solved using an appropriate weighting scheme, at the expense of intuitiveness. In fact, our objective function is such a scheme (Section 4.2.1.3).

Formally it is defined as follows, where w_i is the weight of rank i (see the abovementioned section):

$$\sum_i w_i \cdot \sigma(i)$$

Although perfect for ranking outcomes and incorporates the desired criteria of optimality and fairness, the weighing scheme can make the resulting value impossible to interpret. For evaluation purposes, we are not only

[2] ease of relating (the magnitude of) change to the actual outcome

interested in knowing if an outcome is better, but also by how much. The values produced by the objective function are not intuitive as it is impossible to relate differences in value to the differences in outcome. Sensitivity is also an issue as the produced values are enormous.

5.3.1.4 *AUPCR* (Area under the Profile Curve Ratio)

Introduced by Diebold et al. [13], it can be described as a measure of how concentrated the ranks are towards the 1s rank and is similar to the *Gini coefficient*. Maximum concentration at rank 1 results in an AUPCR of 1.0, whereas the opposite type of distribution will be near 0.0.

Formally it is defined as follows:

$$\text{Total Area} = |S| \cdot |P|$$

$$\text{Area under the Profile Curve (AUPC)} = \sum_{i=1}^{|P|} \sum_{j=1}^{i} \sigma_\mu(j)$$

$$\text{Area under the Profile Curve Ratio} = \frac{\text{AUPC}}{\text{TA}}$$

Notice that project topic slots are irrelevant to its definition.

This metric has similar shortcomings as the weighted sum of ranks. More critically however, it lacks sensitivity to small changes in the outcomes such as those between the Chiarandini and Fair mechanisms.

The following modification, termed *relative AUPCR*, attempts to improve sensitivity for use in such comparisons. The key idea is to use value that is smaller than $|P|$. Doing so decreases the AUPC more than it does the *total area,* thus making the metric more sensitive to changes in the profile.

Instead of $|P|$, we use the maximum *worst rank* among the outcomes we wish to compare. Formally, given a set of matchings $R = \{\mu_1, \ldots, \mu_n\}$ for which we want to calculate the *relative* AUPCRs, we use $r = \max\left(\bigcup_{i=1}^{n} \text{worst}(\sigma_{\mu_i})\right)$ instead of $|P|$ when calculating the value of AUPCR for each of these matching $\forall \mu_x \in R$. See the modified AUPC formula below.

$$\text{Area under the Profile Curve (relative)} = \sum_{i=1}^{r} \sum_{j=1}^{i} \sigma_{\mu_x}(j)$$

5.3.1.5 *Rank profiles*

The rank profile is the complete distribution of ranks in a matching outcome, showing how many students have received their 1st, 2nd, 3rd, etc. choice. It can be presented visually or textually and provides the most detailed view of the outcome. However, it is cumbersome to use for comparing many outcomes.

By discarding some of the fidelity, the distribution can be visualized as a boxplot. Our boxplots visualize the median value (line), the quartiles (box edges), the IQR (inter-quartile range) as whiskers as well as outlier values.

There are also alternative criteria for ranking solutions that are not directly based on ordinal rankings. Of which we mention two: popularity and envy.

5.3.1.6 Popularity

The popularity criterion [1] compares solutions by the number of agents preferring one over the other. An agent prefers a solution in which it is better off. However, we have found this criterion to be unfit for our use case as it is not compatible with profile-based optimization. That is, when two matching outcomes have identical rank profiles, the popularity criterion may differ. We illustrate this with the following example:

> Imagine two different matching outcomes μ_1 and μ_2 for a single, identical problem instance. These two outcomes differ only in the allocation of agents s_1, s_2 and s_3, such that the full rank profiles are identical:
>
> $$profile(\mu_1) = profile(\mu_2)$$
>
> The alternatives assigned to three agents for each of the matching outcomes are ranked by them as follows:
>
> $$ranks(\mu_1, \{a_1, a_2, a_3\}) = \{2, 3, 4\}$$
> $$ranks(\mu_2, \{a_1, a_2, a_3\}) = \{3, 4, 2\}$$

Under the popularity criterion, matching outcome μ_1 wins as both agents a_1 and a_2 are better off and both prefer it over μ_2 , in which only a_3 is better off (by two ranks, $4 \rightarrow 2$), even though the profiles of these outcomes are identical.

The popularity criterion is promising but is incompatible with the profile-based approach that we have chosen.

5.3.1.7 Envy

A fairness criterion known from *envy-free* matchings [9]. When an agent prefers to some alternative that has been assigned to another agent over its own, it is then called *envious* or *having* envy. Envy-based metrics measure the amount of envy in the solution, e.g., number of agents that *have* envy. Such metric attempts to approximate the agents' perception of the fairness and thus quality of a solution. The formal definition is omitted.

We find this criterion to be only of limited use in our setting. Although fairness is a large motivator for our work, we do not think representing it as envy is beneficial. This is because we are also concerned with rank optimality of matching outcomes, whereas envy is always present for an agent as long as its more preferred alternative is allocated to some other agent.

5.3.1.8 Gini coefficient

Another fairness criterion, most often used as an indicator of wealth distribution. In our case that is the spread of the rank frequencies (the rank profile). More specifically, it indicates how close or tightly distributed the values are, with tighter distribution (less spread out) considered to be better. It is similar to the AUPCR metric, which is tailored towards rank distributions.

5.3.1.9 Conclusion

Ultimately, there is no perfect metric. We will use the metrics that best fit the given context. Specifically, we use the following metrics, in the following cases:

- For quick comparisons in aggregate: AUPCR metric
- For quick judgements*: sum of ranks* complemented with the *worst rank*
- For deeper, side-by-side comparisons: rank profiles in written or visual form (boxplot, bar chart)

5.3.2 Pregrouping satisfaction

Beyond project satisfaction, we must also measure how well our mechanisms satisfy students' pregrouping wishes. For this, we use a simple binary metric.

5.3.2.1 Togetherness

A student is considered *together* if their entire pregrouping wish is satisfied. Otherwise, they are *not together*. We measure the proportion of pregrouping students who are together.

We chose not to include partial satisfaction of pregrouping wishes in our metrics because it's unclear how students value partially satisfied pregroupings, and partial satisfaction is difficult to quantify in a meaningful way.

In our analysis, we stratify results by student type (solo vs. pregrouping) and pregrouping satisfaction status (together vs. not together) to provide a view of how different mechanisms balance these objectives.

5.4 Experimental setup

5.4.1 Source code

The experiments are defined programmatically in the codebase. The code for generating the plots too.

The codebase of the thesis, including the code for experiments, reports, and plots can be found at the URL below. Note that the code is contained in the "thesis-experiments" branch.

`https://github.com/PhilipeLouchtch/aidm-optimal-groups/tree/thesis-experiments`

The experiments are defined in code and are included in the repository. The code for the experiments is in the `nl.tudelft.aidm.optimalgroups.experiment` package. The experiments for the historical and synthetic problem instances are in their corresponding sub-packages, found under `src/main/java`:

- Historical: `nl.tudelft.aidm.optimalgroups.experiment.paper.historical`
- Synthetic: `nl.tudelft.aidm.optimalgroups.experiment.paper.synthetic`

The plotting code is written in R and is found under src/main/r.

The references to the specific experiment code and plotting functions used, is contained in the relevant parts in the evaluation subsections.

5.4.2 Result collection details

Runtimes are counted from the start of the mechanism execution. In case of MILP-based mechanisms, the runtime includes MILP model construction, initialization, interfacing with the solver, execution of the solver, and collecting/extracting the results.

Before each experiment the Java VM is warmed up with a small batch of synthetic instances that are solved on all relevant mechanisms. To ensure consistent results, the Gurobi solver was configured to execute in a single-threaded mode. All instances are solved 3 times to detect and eliminate runtime outliers. However, no such outliers were found.

Results are written to a file. The granularity of the exported results depends on the experiment. For instance, the rank profile may be decomposed by student type and/or pregrouping status, i.e., solo, pregrouping satisfied, pregrouping unsatisfied. The results can also contain other metrics such as togetherness and the number of pregrouping students.

The results for the first to experiments with historical problem instances are collected via the `HistoricalExpsRunner`.

The experiments with the synthetic problem instances are more intricate because there are multiple experiments, each with its own set of parameters, including the problem instance generation step. See the `group` and `nogroup` sub-packages for the experiments involving problem instances with and without pregroupings, respectfully.

5.4.3 Software and hardware environment

The results were collected on a desktop PC equipped with an AMD 5600X CPU running the Windows 10 22H1 operating system with OpenJDK 17 as the Java SDK and Gurobi 9.1.1 as the solver.

5.5 Evaluation based on Historical Instance

In this part we present the results of our experiments with the set of historical problem instances. We first detail the problem instances, give an overview of the research questions and then the experiments together with their results.

5.5.1 The instances

The set of historical problem instances consists of a set of instances sourced from the Delft University of Technology (TUD) and Southern Denmark University (SDU). Although the problem settings of the two universities are similar, they have differences which make direct comparison of outcomes between instances of the two universities difficult. These differences are presented below and further down we present the individual instances broken down by their parameters.

5.5.1.1 Differences between TUD and SDU instances

The technical differences between the two sets are presented in the table below. But first there are two additional points that we want to add.

The first point is that each TUD instance is for a single course, e.g. object-oriented programming or research project, within one general field of study, e.g. computer science. Whereas the SDU instances include courses or project topics of multiple fields and/or disciplines, here students submit a partial preference profile over any alternatives they are eligible for. This difference we categorize as “eligibility” in the table below.

The second point is a historical background on the evolution of (pre)grouping preferences in TU Delft instances. In the early TU Delft instances, the grouping preferences were applied broader than pregrouping, later shifting towards just pregrouping. This is because the BEPSys grouping algorithm also supports bilateral and unilateral preferences, even if the latter is highly unlikely to be met. The algorithm was never precisely detailed to the students, but the communication to the students did explain the algorithm in general terms. This led students to try including friends and acquaintances into their preferences, even if unilateral, to secure a better outcome. Later, the problem model was narrowed to pregrouping only, but only in communication to the students and the ability to submit non-unanimous peer preferences, or for pregroupings of alternative sizes has remained possible.

At this point we want to emphasize that our problem definition deals with mutual pregroupings only, so our implementation only extracts mutual cliques from grouping preferences. Yet do remember that the students’ decisions were informed by their understanding of the algorithm, problem model and strategic behavior and are encoded in the instances.

As for the technical differences between the problem models, we have summarized them in the table below. Additional clarification is provided below the table.

	TUD	**SDU**
Slots	*Global, per instance* [1]	*Per topic*
Group size bounds	*Global, per instance* [1]	*Per topic*
Eligibility	*All*	*Depends on student/project*
Project topic preferences	*Complete*	*Partial (top 8)*
Pregrouping [2]	*Up to $u - 1$ friends (clique). Pregrouping variant: configurable.*	*Hard pregrouping of 2 or 3 students.*
Misc.	*Students without topic preferences are interpreted to be fully indifferent. They are not counted towards metrics or plots.*	*Instances 2015 and 2016 have no pregrouping due to a business choice to disallow pregrouping*

1) By “Global, per instance” we mean that all project topics within an instance have the same number of slots or group size bounds. This is not a shared resource, so in the case of group slots, assigning a group to a topic does not decrease the number of available slots of another topic. This differs from the SDU instances, where each project topic defines its own value, so one topic can support more groups than another and/or of differently sized groups.
2) The students in the TUD instances are allowed to submit a set of up to $u - 1$ students with whom they would like to work with. These do not need to constitute cliques, and when not a clique, are to be satisfied in a best-effort manner by the BEPSys algorithm. For the other mechanisms we only consider the cliques and handle them according to the chosen pregrouping variant. In the SDU instances, students wishing to pregroup can only do so in groups of 2 to 3 students. These groups are atomic and have a singular project topic preference list.

Both problem settings are fully supported by our implementation and our results for the SDU problem instances should be directly comparable with those of their original publication.

5.5.1.2 Instance summaries

Below are table summaries of the two sets of historical problem instances. Because the SDU project topics define their own group size bounds, we present the bounds as a distribution in the SDU summary table. *The group bounds* indicate the allowed group sizes in the format of $[\ell\text{-}u]$ with the dash character as a separator. For the TUD instances the outlying group bounds are bolded, those of SDU are varied and therefore are listed in order of prevalence and furthermore are prefixed with the number of occurrences.

We omit the number of slots in favor of the *pressure* metric. The *pressure* of an instance is a ratio of students to project topic capacity; with 1.0 indicating there to be just enough project topic and slots offered for each student to be successfully matched. Lower numbers indicate overprovisioning of project topic capacity.

The last column, *pre-groupings frequencies,* contains a |-separated list of the number of students with a pregrouping preference of the corresponding size (see header). For example, if there are exactly 5 pairs of students who wish to be matched together in pair, then under ‘2:’ 10 students are written. Another example, for CE11 there are 12 students who wish to work in pairs (constituting 6 pairs), 6 students in triples (2 groups of 3 students), 16 students in groups of 4 (4 groups), 40 in groups of 5 (8 groups) and 24 in groups of 6 (4 groups).

id	Group bounds	$\lvert S \rvert$	$\lvert P \rvert$	Pressure	Pre-groupings frequencies 2: 3: 4: 5: 6:
CE3	**[3-3]**	108	36	1.000	10| 6
CE4	[4-5]	160	7	0.914	8|15|16|60
CE10	[4-5]	149	42	0.714	6|18|40|30
CE11	**[6-6]**	252	9	0.934	12| 6|16|40|24
CE14	[4-5]	50	12	0.834	6| 0| 0| 0
CE17	[5-6]	45	5	0.534	10| 3| 4| 0| 0
CE18	[4-5]	111	32	0.719	12|12|12|25
CE23	[4-5]	51	14	0.786	12| 0| 0| 5
CE39	[4-5]	287	73	0.795	8| 3| 0|65
CE42	[4-5]	39	14	0.571	2| 0| 0| 0
CE45	[4-5]	356	84	0.857	22| 0| 0|15

Table 1: Technical University of Delft (TUD) problem instances

id	Group bounds (distribution)	$\lvert S \rvert$	$\lvert P \rvert$	Pressure	Pre-groupings distribution 2: 3:
2008	33: [3-5] 14: [3-4] 2: [3-6] 1: [3-3] [4-5] [3-10]	200	52	0.570	38|12
2009	27: [3-5] 12: [3-4] 5: [3-3] 1: [4-5]	129	45	0.594	24|15
2010	34: [3-5] 12: [3-4] 4: [3-3] 1: [4-5] [3-6]	193	52	0.672	38|24
2011	48: [3-5] 16: [3-4] 4: [3-3] 1: [3-6]	259	69	0.676	48|24
2012	59: [3-5] 15: [3-4] 4: [3-3] 1: [4-5] [3-6]	300	81	0.628	58|36
2013	70: [3-5] 13: [3-4] 9: [3-3] 3: [3-6]	355	95	0.659	56|51
2014	74: [3-5] 12: [3-4] 9: [3-3] 2: [3-6] 1: [3-7]	371	98	0.595	74|66
2015	69: [3-5] 11: [3-4] 8: [3-6] 7: [3-3]	422	96	0.595	-
2016	67: [3-5] 15: [3-6] 13: [3-4] 5: [3-3]	416	100	0.525	-

Table 2: University of Southern Denmark (SDU) problem instances

5.5.2 The experiments

In the evaluation section's introduction, we have outlined our research questions for the evaluation. This subsection aims to answer those that are defined for the evaluation with historical instances. We first compare the Chiarandini mechanism with that of BEPSys. Then, we compare the Chiarandini mechanism with Fair in the setting of max-size pregroupings only. Finally, we evaluate if relaxing pregrouping rules is beneficial.

5.5.2.1 Chiarandini vs BEPSys

In this experiment we compare the performance of the BEPSys mechanism with that of Chiarandini. We omit the Fair mechanism from this comparison as its outcomes are similar to those of Chiarandini. With this experiment we aim to answer the first question: "*How do the new mechanisms compare to the prior BEPSys algorithm*?". Our preliminary results have shown us the difference between BEPSys and the new mechanisms to be considerable and the difference between the latter to be relatively small. Therefore, for this experiment we compare BEPSys only against the Chiarandini mechanism.

5.5.2.1.1 Setup

Only TU Delft instances were used because the BEPSys mechanism does not support the more flexible SDU problem model.

Only pregroupings of size u are considered valid. Invalid pregroupings are ignored and their constituents are classified as *solo* students instead.

5.5.2.1.2 Presentation

We have carefully selected four instances whose results represent the full range of relative mechanism performance. The results of each selected instance are presented as boxplots of ranks for each of the types of students. The plots are ordered from best (left) to worst relative performance.

5.5.2.1.3 Results

The matchings obtained with the Chiarandini mechanism generally demonstrate significant improvement over those found by the BEPSys mechanism (Figure 1). The two left-most results are the exception rather than the norm, whereas the two rightmost results (CE10 and CE45) are more typical. Even in the best cases, BEPSys produced matchings whose rank profiles are not as optimal as those produced by the Chiarandini mechanism. However, when it comes to *pregrouping* students, the median assigned rank is lower than that of the Chiarandini mechanism. Nevertheless, this outcome is undesirable as the optimization goal of our problem setting is to prioritize *solo* students, while pregrouping is an additional service to the students who value doing so.

The performance of BEPSys is explained by its algorithm, which consists of two sequential steps. First, the groups are created using a greedy algorithm. Then, the groups are assigned to projects using an optimal algorithm but based on the groups' aggregated topic preference profile. The problem lies in the grouping algorithm, it handles forming pregroups from cliques well and pregrouping students usually align their individual topic preferences before signup. The opposite is true for the solo students, the greedy grouping by similarity of project topic preference is not very good and the problem is made worse by aggregating the grouped individuals' topic preferences into a single project topic preference of that group. The outcome is usually good for pre-formed groups, but not so for the rest of the students. Larger instances, such as CE45, exacerbate this problem.

At this point we conclude our evaluation with the BEPSys algorithm.

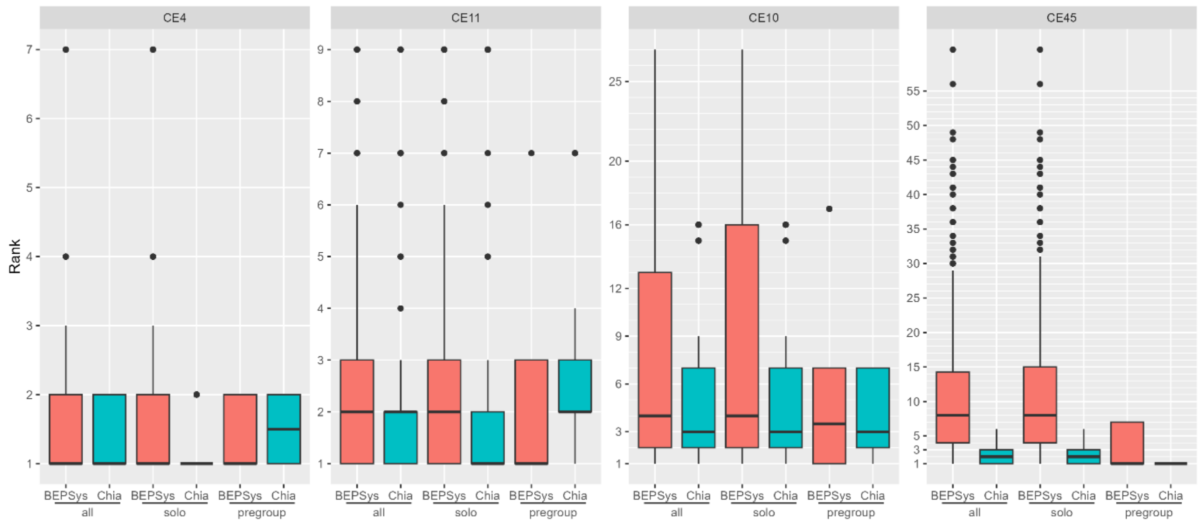


Figure 1: Choice instance results for the BEPSys and the Chiarandini mechanisms. The plot shows the ranks per student type as a boxplot, with dots indicating outliers. Lower is better. The results are grouped by the type of student (all, solo, pregrouped) to make comparing the two mechanism easier.

5.5.2.2 Chiarandini vs the Fair mechanism

In the previous experiment we have compared the BEPSys mechanism to that of Chiarandini and have determined that the Chiarandini mechanism to be superior. For this experiment we compare the Chiarandini mechanism with the Fair mechanism.

As our Fair mechanism is based on that of Chiarandini, the results are similar. By design, the Fair mechanism should prioritize *solo* students over *pregrouping*. Comparing the two mechanisms against each other allows for a granular comparison. Our analysis focuses on how this how this difference in design translates in the trade off in outcome quality for these two types of students.

5.5.2.2.1 Setup

All TUD and SDU instances are included. With respect to the grouping preferences, for TUD instances only pregroupings of size u considered whereas for SDU instances the grouping preferences are kept as is.

5.5.2.2.2 Presentation

The results of this experiment are presented as tables. The results are stratified by problem instance, student type (solo/pregrouping) and mechanism. The results are grouped in tables first by source (TU Delft, SDU), then by student type.

The tables are structured as follows. The rows contain the results for the given problem instance and student type. There are three columns with metrics. The first two contain the metrics for the given mechanism while the third contains the difference between the profiles of outcomes of the two mechanisms, from the perspective of the fair mechanism: profile delta = $profile(\mu_{Fair}) - profile(\mu_{Chiarandini})$. For example, for instance CE4 (Table 3), the Fair mechanism assigned 5 solo students to their 1st ranked project topic less and 5 solo students their 2nd ranked project topic more than the Chiarandini mechanism.

The format of the metrics in the first two columns is presented as follows: "sum of ranks (difference: this - other) [worst rank]". When an outcome is pareto dominated, we leave the dominated outcome empty while the dominant mechanism's metrics are shown in **boldface**, like is seen for CE4 (solo students). When both outcomes are identical, we show the equality sign: "=". Furthermore, we leave out the worst rank metric when identical in both outcomes, as is the case for all instances shown in Table 3.

All (valid) pregroupings were satisfied, and therefore the pregrouping satisfaction metric is excluded from the tables to save space.

5.5.2.2.3 Results - TU Delft instances

TU Delft instances - *solo* students			
CE	**Fair**	**Chiarandini**	**Profile delta**
3	410 (-2)	412 (+2)	[+2 -2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 +1 0 0 +1 0 -1 0]
4		**121 (-5)**	[-5 +5]
10	**577 (-98)**		[+7 +7 -6 +3 -3 -1 -2 -1 0 +1 0 0 0 0 0 -5]
11	388 (-54)	442 (+54)	[-2 +13 -1 0 -3 -1 0 0 -6]
14	**74 (-2)**		[+2 -2 0]
17	**50 (-2)**		[+2 -2]
18	**147 (-17)**		[+12 -7 -5]
23	**71 (-8)**		[+8 -8 0]
39	**458 (-28)**		[+14 -1 -12 -1]
42	=	=	
45	693 (+3)	690 (-3)	[-6 +9 -3 0 0 0]

Table 3: Results for Chiarandini and Fair mechanisms for the solo students in TU Delft instances. See 5.5.2.2.2 for the specifics of presentation and format.

TU Delft instances - *pregrouping* students			
CE	**Fair**	**Chiarandini**	**Profile delta**
3	=	=	
4	**85 (-5)**		[+5 -5]
10	175 (+56) [17]	119 (-56) [7]	[+5 -5 -1 -4 0 +5 -5 0 0 0 0 0 0 0 0 0 +5]
11	90 (+15) [9]	75 (-15) [7]	[+5 -6 0 -1 0 0 -4 0 +6]
14	=	=	
17	=	=	
18	[7]	**53 (-22) [3]**	[0 -10 +5 0 +4 0 +1]
23	[7]	**5 (-12) [1]**	[-5 +2 +2 0 0 0 +1]
39	[4]	**115 (-65) [3]**	[-20 -5 +5 +20]
42	=	=	
45	[4]	**15 (-25) [1]**	[-10 0 +5 +5]

Table 4: Results for Chiarandini and Fair mechanisms for the pregrouped students in TU Delft instances. See 5.5.2.2.2 for the specifics of presentation and format.

At first glance and judging by the sum of ranks metric, most of the results are as expected: under the Fair mechanism, solo students obtain a better topic allocation at an expense to the pregrouping students. However, closer examination also reveals outliers. We summarize and break down the results below, classified by outcome type.

- Expected
    a. **CE 10**: The solo students are considerably better off under the Fair mechanism at the expense of pregrouped students. Two pregroups are also better off yet one group has sunk to a very low rank. This is a good test case and motivation for a *conditional pregrouping* variant. Overall, the improvement in the sum of ranks to solo students is greater than the loss incurred by the pregrouped students.
    b. **CE 11**: Also shows a significant improvement to the solo students with the Fair mechanism, with an even better sum of ranks tradeoff between the solo and the pregrouped students compared to CE10.
    c. **CE 18, 23 and 39**: The solo students improve with the Fair mechanism but contrary to CE10 and 11, at a greater cost to the pregrouping students than the improvement to the solo students.
    d. **CE42**: This is a small instance without valid pregroupings, and both mechanisms find identical rank profiles.
- Notable and/or unexpected
    a. **CE 4**: The result is opposite our expectation: solo students are pareto better off with Chiarandini but pregrouped under Fair. We discuss this result in depth below.
    b. **CE 14, 17**: The solo students are pareto better off with the Fair mechanism, yet these instances don't contain valid pregroupings so both mechanisms are expected find solutions with identical rank profiles. We suspect numerical issues in the objective function. We discuss this outcome further below.
    c. **CE45**: Chiarandini outperforms the Fair mechanism. This is a large instance but there are only 3 groups of size u. We suspect one of the two: (1) the solo students got unlucky during the baseline matching of the Fair mechanism, and (2) the numerical stability issue of the objective function which we already is suspect to be the cause of the CE 14 and 17 outcomes. The latter is more likely for large instances, such as this. Both factors could contribute to this outcome.

In-depth: CE14 and 17
Because there is no pregrouping we can reason directly about the students based on their project topic preferences and the objective function, therefore Chiarandini should have found the same solution. Randomness in the solver is a potential explanation but that would imply the solver to be not optimal, so this is an unlikely explanation. A more likely culprit is the implementation of our objective function. The OWA objective function is known to have a numerical stability issue for larger instances with many students and topics [12]. Similar outcomes under different conditions have been noted in the live system with newer course editions than what we managed to include in this work. However, these two datasets are not large.

In-depth: CE4
The difference between the results of the two mechanisms is one set of five solo students trading places with a pregroup of five, going from their top topic choice to their second best and vice versa. To understand what is going on we need to remember that the Fair mechanism outcome is constructed relative to its initial baseline outcome that doesn't consider pregrouping preferences, and not relative to the outcome as found with Chiarandini with pregrouping. Then, we must consider the difference between the two final outcomes, wherein five solo students improve one rank and one pregroup reciprocally drops down one rank. We suspect that in both outcomes the differences concern the same students, one set of 5 solo students and one pregrouping. We also suspect both had identical top two topic preferences. Therefore, it seems to us most likely that the pregroup in question was *lucky* during the baseline matching of the Fair mechanism at least part of the students therein obtaining their first pick, or alternatively, the set of solo students being less lucky and obtaining a worse outcome in the baseline than they had gotten with the Chiarandini mechanism. During the second round with the grouping constraints on, the most optimal choice was to give the pregrouping their first choice.

We conclude that the Fair mechanism does not always result in better outcomes for the solo students than the Chiarandini mechanism. Perhaps allowing instances to contain pregroups of smaller sizes will provide the mechanism with additional combinatorial freedom leading to more desirable outcomes.

Lastly, we note that the runtimes of the Chiarandini and Fair mechanisms for solving the TU Delft instances with max-sized pregrouping are in seconds. See Figure 2 below.

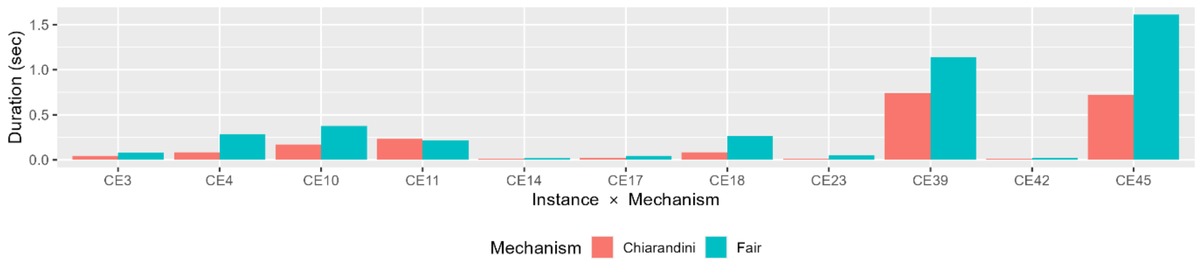


Figure 2: The Chiarandini and Fair mechanisms runtimes of solving the historical TU Delft instances

5.5.2.2.4 Results - SDU Instances

The results below were obtained adhering to all the specifics of the SDU setting, these are: restrictions, per-project group size bounds and group slots, and pregroups of two and three students.

SDU instances - *solo* students			
Edition	**Fair**	**Chiarandini**	**Profile delta**
2008	**254 (-9)**		[+5 -2 -2 -1]
2009	=	=	[-2 +4 -2]
2010	**207 (-5)**		[+4 -3 -1]
2011	385 (-15) [5]	400 (+15) [6]	[-3 +10 -2 -1 -2 -2]
2012	**439 (-1)**		[0 +1 -1 0 0]
2013	**493 (-31) [6]**	524 [7]	[+16 -12 0 0 -2 -1 -1]
2014	**472 (-6)**		[+2 -1 0 +1 -2 0]
2015	**674 (-4)**		[+4 -4 0]
2016	=	=	=

Table 5: Results for Chiarandini and Fair mechanisms for solo students in SDU instances. See 5.5.2.2.2 for the specifics of presentation and format.

SDU instances - *pregrouping* students			
Edition	**Fair**	**Chiarandini**	**Profile delta**
2008	[4]	**90 (-4) [3]**	[-2 +2 -2 +2]
2009	[3]	**51 (-2) [2]**	[0 -2 +2]
2010		**89 (-6)**	[-4 +2 +2]
2011		**149 (-15)**	[0 -6 +3 0 0 +3]
2012		**210 (-2)**	[0 -2 +2 0 0]
2013	[7]	**221 (-37) [6]**	[-10 +2 +4 -3 +1 +4 +2]
2014		**281 (-16)**	[-6 +4 -2 0 +4]
2015	Pregrouping disabled		
2016	Pregrouping disabled		

Table 6: Results for Chiarandini and Fair mechanisms for the pregrouping students in SDU instances. See 5.5.2.2.2 for the specifics of presentation and format.

Note the absence of results for the pregrouping students for instances 2015 and 2016. From 2015 on, the University of Southern Denmark has decided to disallow pregrouping after considering them possibly disadvantageous towards the *solo* students.

Compared to the results obtained for the TU Delft instances, here our Fair mechanism shows a more consistent performance. In nearly every instance the solo students are prioritized and obtain a better matching compared to Chiarandini at the expense of the pregroup students.

There are two noteworthy outcomes. The 2009 instance outcome is interesting, the Chiarandini mechanism spread four *solo* students from rank 2 over ranks 1 and 3 and managed to improve two *pregrouping* students by moving them from rank 3 to rank 2. The 2015 instance results shows the same unexpected behavior that we have seen with the TUD instances; the Fair mechanism manages to find a pareto improvement over Chiarandini although the results are expected to be the identical.

5.5.2.3 Relaxing pregrouping size restrictions

In the preceding experiments, we have limited the pregrouping to be only of size u for the TUD instances. This is a recent change to the pregrouping setting. We do not know if this restriction is beneficial for either the *solo* or *pregrouping* students. However, looking at the problem setting of the SDU and the results it produces, we wonder if relaxing this restriction is beneficial.

With this experiment we aim to investigate the effect that relaxing the pregrouping size restrictions has on the project satisfaction of the *solo* students. Secondarily, we also consider whether the tradeoff between the solo and the classes of pregrouping students is worth it, the latter based on their project satisfaction and togetherness.

To do so, we define three pregrouping *scenarios* and apply them to a selected set of historical TUD instances which contain a sufficiently large and varied set of pregrouping preferences. We include both the Chiarandini and the Fair mechanisms, not driven by the intention to compare them directly but to see how they behave under the scenarios and allow for a per-scenario comparison.

We look at three successively more relaxed, pregrouping *scenarios*:

- Max-only: only pregroupings of the maximum group size, u
- Except: all pregroupings *except* those of the *sub-max* size, $u - 1$
- Any: all pregroupings are allowed, $\leq u$

Furthermore, we classify the students by their *true* pregrouping class as follows:

- Solo:	students that are not pregrouping
- Maximal:	pregroups of size u
- Sub-max:	pregroups of size $u - 1$
- Small:	pregroups of size $2 \leq \lvert g \rvert < u - 1$

5.5.2.3.1 Setup

Pregrouping

The pregrouping scenario determines which pregroupings are valid, the invalid pregrouping wishes are ignored during matching. However, in contrast to the previous experiments, their constituents are still referred to and stratified by their true pregrouping class.

Pregrouping constraint

The *soft* pregrouping constraint was utilized for all mechanisms in this experiment to ensure satisfiability of the instance, therefore not all pregroups may be *together.*

Problem instances

For this experiment, only a subset of the TUD instances meets our requirement. The earlier instances, CE 4, 10, 11 and 18, have sufficient variety and amount of pregroupings to use in this experiment, these constitute our main problem set for this experiment. The grouping preferences in the later instances, CE23, 39, and 45, are mostly *maximal* but do contain a few pair grouping wishes. We include the plots for this the second set of instances at the very bottom of this experiment for completeness and without comment.

5.5.2.3.2 Presentation

The results are presented as plots of the relative AUPCR values, grouped by mechanism, scenario and pregrouping class. The color of the dots indicates the instance. In addition to the dots, the plots are overlayed with box-and-whiskers plots to aid in visual comparisons between scenarios or mechanisms. Furthermore, we have grouped the results by mechanism to make comparing the scenarios easier.

The AUPCR results are presented separately, a plot dedicated for each subset. We also have included the results on togetherness of students, in the interest of economy of space these are presented in a single plot.

Due to space constraints, we abbreviate the *except* scenario as “exc” on the x-axis.

5.5.2.3.3 Results – project satisfaction

In general, we again clearly see the difference in outcomes between the Chiarandini and the Fair mechanism. Reiterating the results of the previous subsection, we see the Fair mechanism giving preferential treatment to the *solo* students, whilst under the Chiarandini mechanism the pre-grouped students are for some problem instances better and under other worse off than the *solo* students.

However, with this experiment we are interested in observing how the scenarios under the given mechanisms impact the classes of students. We list our observations on the main instance set below, with a conclusion to follow.

1) The effect of relaxing the pregrouping restriction on the *solo* students is mixed, regardless of the mechanism
2) Under the Chiarandini mechanism, the *maximal* class is progressively getting better off as we relax the pregrouping restrictions, that is, include more pregroups into the matching
3) Under the Fair mechanism, and in contrast to the Chiarandini mechanism, we see a substantial decrease in project satisfaction for the first scenario where that class is enabled for pregrouping. The clearest example is the *small* class under the *except* scenario compared to the more/less restrictive scenarios left and right of it in the plot below. This indicates a pattern for the Fair mechanism, corresponding with its design. The students are well off while they are considered *solo* by the mechanism, but less so when participating as a pregroup (and competing for desired projects), however, this effect is smoothed out when there are more pregroups participating in the matching.
4) Some pregroups are together naturally, without needing to be enforced through the mechanism.
5) Relaxing the pregrouping restriction, reduces the togetherness of pregroups

Based only on project satisfaction, there is no clear winning scenario; some instances improve, other get worse. If we consider the impact on the togetherness of students (second plot), the picture becomes clearer. Relaxing the pregrouping restriction does not lead to a consistent improvement to the *solo* students. Nor does it really help the pregrouping students, their togetherness drops and in the case of the Fair mechanism so does their project satisfaction.

Note that the above is true for the tested instances, but not necessarily for future instances future instances. Communicating this effect clearly to the students can alleviate this problem by allowing them to make a more informed decision about how they want to participate in the matching, if they want to participate in the matching solo or if their desire to pregroup outweigh the expected loss of project satisfaction. An alternative solution lies in a more advanced pregrouping constraint, such as conditional pregrouping.

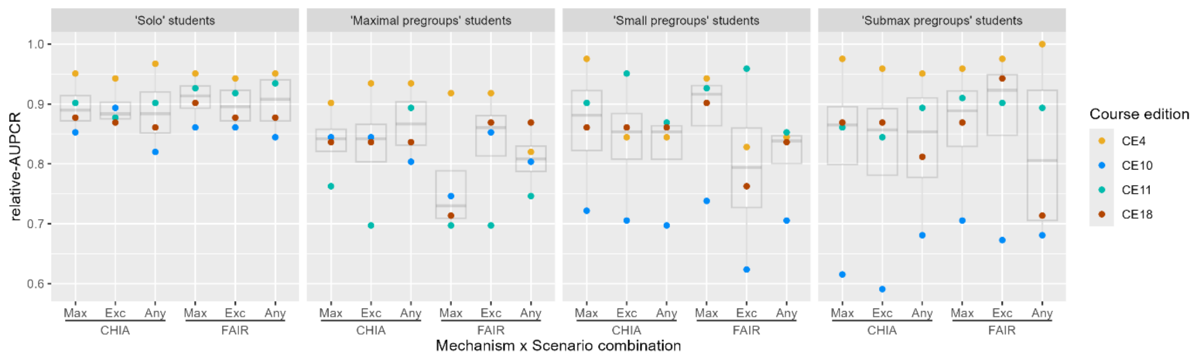


Figure 3: project satisfaction (aggregated as rAUPCR) per grouping type of student (solo, maximal, small and submax), mechanism (Chiarandini, Fair), grouping scenario (maximal, excluding, any) and instances of the main set of instances (CEs 4, 10, 11 and 18)

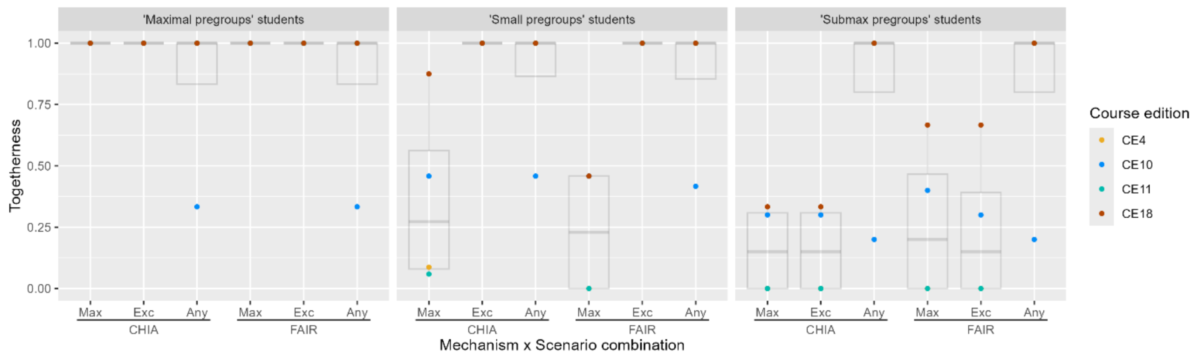


Figure 4: grouping satisfaction of students (togetherness, normalized) per grouping type of student (maximal, small and submax), mechanism (Chiarandini, Fair), grouping scenario (maximal, excluding, any) and instances of the main set of instances (CEs 4, 10, 11 and 18)

Results of the remaining instances

For completeness, Figure 5 and Figure 6 below present the results for the remaining instances which contain mostly max-size pregroupings.

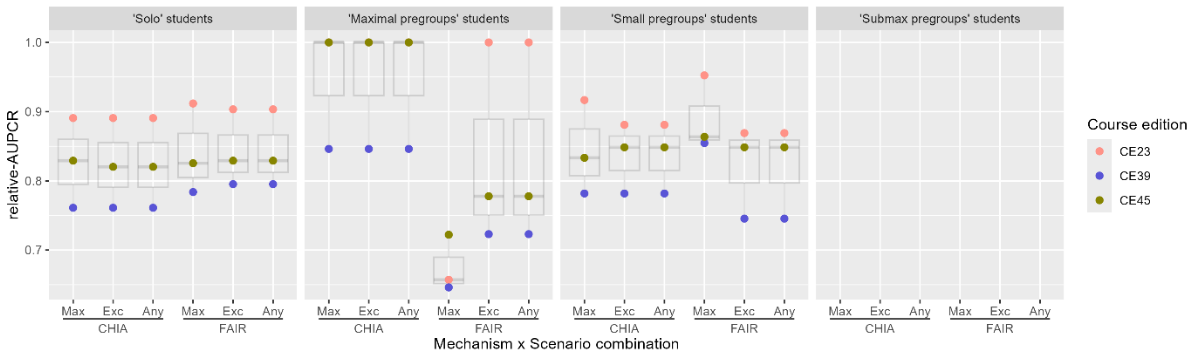


Figure 5: project satisfaction (aggregated as rAUPCR, within the instance) per grouping type of student (solo, maximal, small and submax), mechanism (Chiarandini, Fair), grouping scenario (maximal, excluding, any) and instances of the remaining set of instances (CEs 23, 39 and 45) whose pregroupings are limited to pregroups of max-size and some pairs

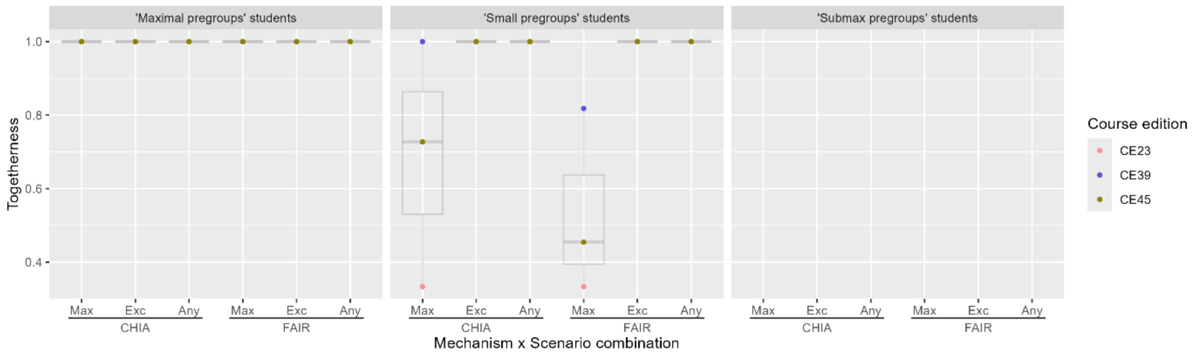


Figure 6: grouping satisfaction of students (togetherness, normalized) per grouping type of student (maximal, small and submax), mechanism (Chiarandini, Fair), grouping scenario (maximal, excluding, any) and instances of the remaining set of instances (CEs 23, 39 and 45) whose pregroupings are limited to pregroups of max-size and some pairs

5.6 Evaluation based on Synthetic Instances

In this part we evaluate our mechanisms using synthetic datasets that we generate at runtime.

In the first experiments we evaluate how the more basic parameters of an instance – such the instance size, provisioning of topics and/or group slots, group size bounds – influence the outcomes and how they are handled by the mechanisms. After that, we conclude with an experiment on the effect of pregrouping for the Chiarandini and Fair mechanisms.

5.6.1 The instances

We have created simple yet for our purposes sufficient problem instances generator for the intended evaluation. To generate problem instances, we use the parameters as defined in our problem model such as the number of project topic, slots and students and most crucially, their preferences.

The problem instances are generated by creating the requested number of students, project topics with the corresponding number of group slots and generating the project topic and peer preferences.

Generating problem instances consists of supplying the problem model parameters and generating agents' preferences to match them. The project topic preferences are generated by drawing the topics according to a distribution. The idea is that students can be modeled as having a preference *type*, and that type represents a distribution of desirability of project topics. Naturally, not every student is the same; their preferences vary, due to personal taste or even strategic reasons. This ordering and variation we model as a distribution. We conceived the following distributions, which we usually refer to as preference types, and presented ordered from most to least unanimous desirability:

- **unanimous**: no variation, a simple randomly predefined ordering of projects.
- **nearly unanimous**: mostly the same but now includes some variation; the resulting preferences between agents differ a little bit.
- **mostly unanimous**: the variation is more pronounced than the previous type.
- **3-peaked (realistic)**: actually, a composite distribution. There are three types of students, each type corresponding to a different distribution of the *mostly unanimous* type. The proportion of students per type is 45%, 35% and 20%.
- **random**: the project topic preferences are completely random, maximum variation.

To generate a project topic preference profile for a student, the generator uses the given distribution to draw the topics. The 3-peaked distribution is different in that it is composed of three types; we have set those types to have a chance of being drawn of 40, 35 and 20%. Generating the students' preference from a 3-peaked distribution is as follows. First the student's type is drawn, the type represents the actual project topic distribution and the student's distribution is drawn from it in the same manner as with the other distributions. See Figure 14 in Appendix A for a visualization of the resulting distribution of topic desirability of the generated preferences.

We have also implemented the generation of pregroupings; however, this will be described later in the experiment with synthetic pregroupings.

To describe the ratio of students to project topics and slots, we define a composite *pressure* metric. It represents how well provisioned the instance is, or to a lesser extent, describes the amount of competition there is between students for topics. More specifically, it is a ratio between the minimal number of groups that the students can be grouped in to and the actual capacity of the instance: $\text{pressure} = \frac{min\_grps}{|P| * C}$, where C is the common group slots capacity.

For values to be used in the experiments for the problem model parameters, we defined a set of scenarios for each of them:

- number of students $|S|$: *small* (50), *medium* (200) and *large* (600).
- pressure: *loose* (0.5), *medium* (0.75) and *tight* (1.0).
- group bounds: the common TUD setting of $\ell = 4,\ u = 5$, unless stated otherwise

From each set of parameters 5 instances are generated.

5.6.2 The experiments

We begin with simple experiments *without* pre-groupings to investigate the effect of the problem instance size, the number of project group slots, the distribution of project preferences and the group size bounds. Then we move to a more complicated setting of pregrouping with an experiment to evaluate the effect of pregrouping under various scenarios on the outcomes for all participating students.

The reason why we kept the pregrouping to one experiment is because adding pregrouping to the mix significantly complicates an experiment due to a combinatorial explosion; what proportion of students to dedicate to pregrouping, how are their preferences distributed compared to the *solo* students, and how do they relate to the other parameters of the problem instance? We attempted to investigate these questions in that experiment.

As already presented in the introduction, the following is the questions we aim to answer in this part of the evaluation:

- How do our mechanisms scale with larger instances sizes?
- What is the effect of over-provisioning instances with project topics?
- How does the group size bounds impact our mechanisms or outcomes in general?
- What influence does pregrouping have?

To this end we have run the following set of experiments, referred to by the parameter or aspect it is varying:

- Instance *size* and *pressure* – investigate the effect of the number of students, *pressure* and the interaction between project topics and slots.
- *Group size bounds* – investigate the effect of different group size bounds, that is smaller or bigger groups and the range between the bounds.
- *Pregrouping* – investigate the effect of pregrouping: in proportion of pregrouping/solo and difference in preferences between the types of students.

Some experiments include an adaptation of the so-called Serial Dictatorship mechanism for our setting. During the work on this thesis, we have explored multiple approaches towards solving our problem. The serial dictatorship mechanism was extended to a similar problem model before and is found in the literature as the Serial Dictatorship with Project Closures [20]. We have further modified it to support project slots of our problem model. We have included its results to serve as a kind of reference point for both BEPSys and the new MILP based mechanisms. This mechanism does not find complete matchings, it is left out of the experiments where this occurred or is otherwise unable to solve. In the interest of time and space we will not detail it any further.

5.6.2.1 Instance size and pressure

Problem instance size is determined multiple factors: the number of students, the number of project topics and the group capacity each topic has (the slots). As mentioned before, the latter two can be taken and expressed in relation to the number of students, as *pressure.*

In this experiment we look at the effect of the number of students and pressure, also included are all the project preferences distributions.

An additional question we found interesting is how the desirability of the projects relates to the runtime and outcome quality for the tested mechanisms. Given our scale of unanimity of desirability of project topics, we hypothesize unanimous preferences to be the easiest to solve, because the matching decisions are trivial. Followed by the least unanimous, the random preference model because everyone can get what they want easily due to low competition for the same topics. And the hardest to solve preference model we expect to be the somewhat-less-unanimous models due to high competition and a lot of subtle yet significant allocation choices.

5.6.2.1.1 Setup

We have generated and solved 5 instances with every combination of parameters. The rank profiles and runtimes along with the used parameters and the generated instance sequence number were recorded and plotted.

Each instance is solved 3 times to detect and prevent outliers. No such outliers were detected, the presented results consist of the second run only.

5.6.2.1.2 Presentation

We present the results for the runtime and outcome quality separately below in plots. Plots for both the runtime and quality outcomes have a similar layout. Each plot is like a table of smaller plots, each subplot shows the how the mechanisms perform for the given project preference model, size and pressure setting. The pressure setting is determined by the pressure scenario (rows) and the number of slots (nested x-axis), so as the number of slots increases the number of projects decreases. The columns represent the used project preference model, ordered from most to least unanimous. The rows represent the size in terms of students and pressure.

The runtimes plot indicates the runtimes for solving the generated instances in seconds.

The outcome quality plot aims to only roughly indicate the quality of the outcome and to that end it presents two metrics: the AUPCR (left, boxplots) and the worst rank (right, dots as black stars). The Fair mechanism was to save space because its matchings are identical to Chiarandini when pregrouping is disabled. Only instances of 200 students were included.

5.6.2.1.3 Results – runtime

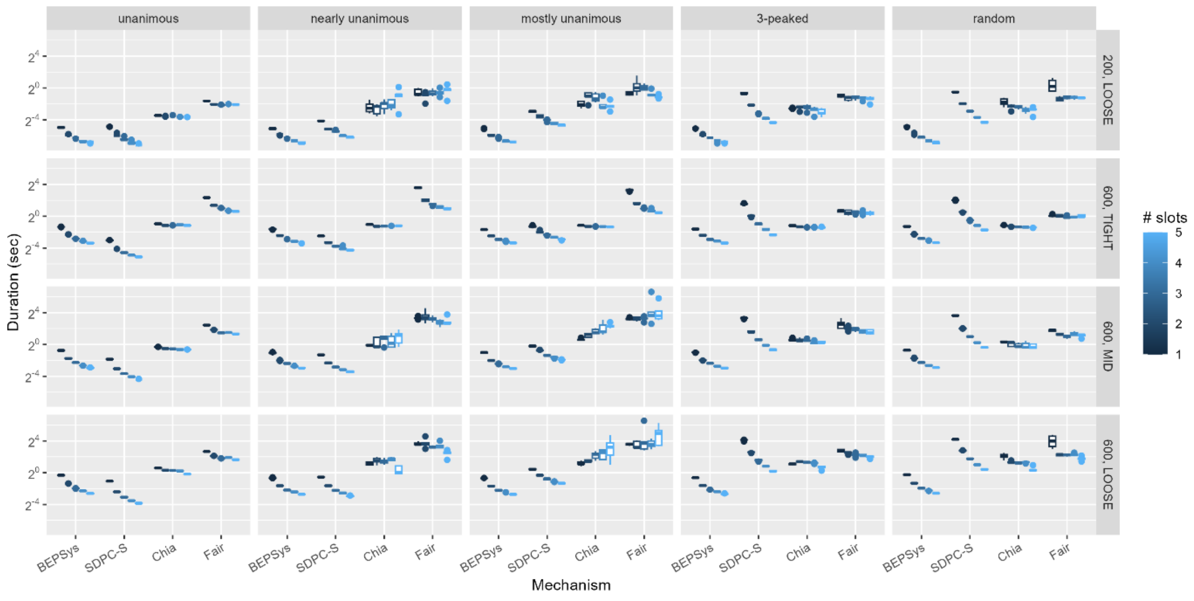


Figure 7: Runtimes of solving synthetic problem instances using different preferences generators (columns) and varying the problem instance size, (over)provisioning of topics/slots with pressure (loose, mid, tight). The number of topics in the instance is determined by the pressure and number of slots,

1) Provisioning an instance with more slots rather than a wider selection of project topics, lowers the computational complexity of the instance in the mid and tight pressure-scenarios (middle two rows), except for the *mostly unanimous* setting.
2) The *mostly unanimous* preference model is most computationally demanding for the MILP based mechanisms, with the Fair mechanism being most heavily impacted (more than double of Chiarandini)
3) Some outlier instances were even more demanding for the Fair mechanism, the results are consistent over multiple runs for the same generated instance.
4) SDCP-S struggles with the least-unanimous preference models of *3-peaked* and *random.*
5) The reworked BEPSys mechanism has solved all instances quickly with excellent runtime scaling.

The runtime results for Fair over Chiarandini are more than just double. This is noteworthy because the Fair mechanism solves the Chiarandini model twice: first without pregrouping and a second time with additional constraints. In this experiment, these additional constraints should be irrelevant due to the absence of pregrouping. Nonetheless, it seems the additional constraints add significant computational difficulty. Additionally, there is also some overhead added from the application creating and submitting the mixed integer program to the solver, a hint of this can be seen in the results for the Fair mechanism with the 600-tight unanimous instances compared to those for Chiarandini, but we have not investigated this further.

The re-implementation and algorithmic improvement to the BEPSys mechanism managed to solve all instances quicky and without problems. During our initial requirements elicitation, a major pain point we have heard is that the old BEPSys algorithm is slow and can sometimes get stuck in infinite loops. Over the course of our work, we have experienced these issues and have incrementally identified and improved upon these points in our re-implementation, but our starting point was a pure re-implementation. This way we have improved the grouping algorithm and replaced min-flow matching with a Google OR-Tools implementation. It is this re-implemented and improved version that we have included in our evaluation because we did not want it to freeze or stall our experiments. Regarding the outcome quality, the improved implementation does not differ significantly for the original.

5.6.2.1.4 Quality

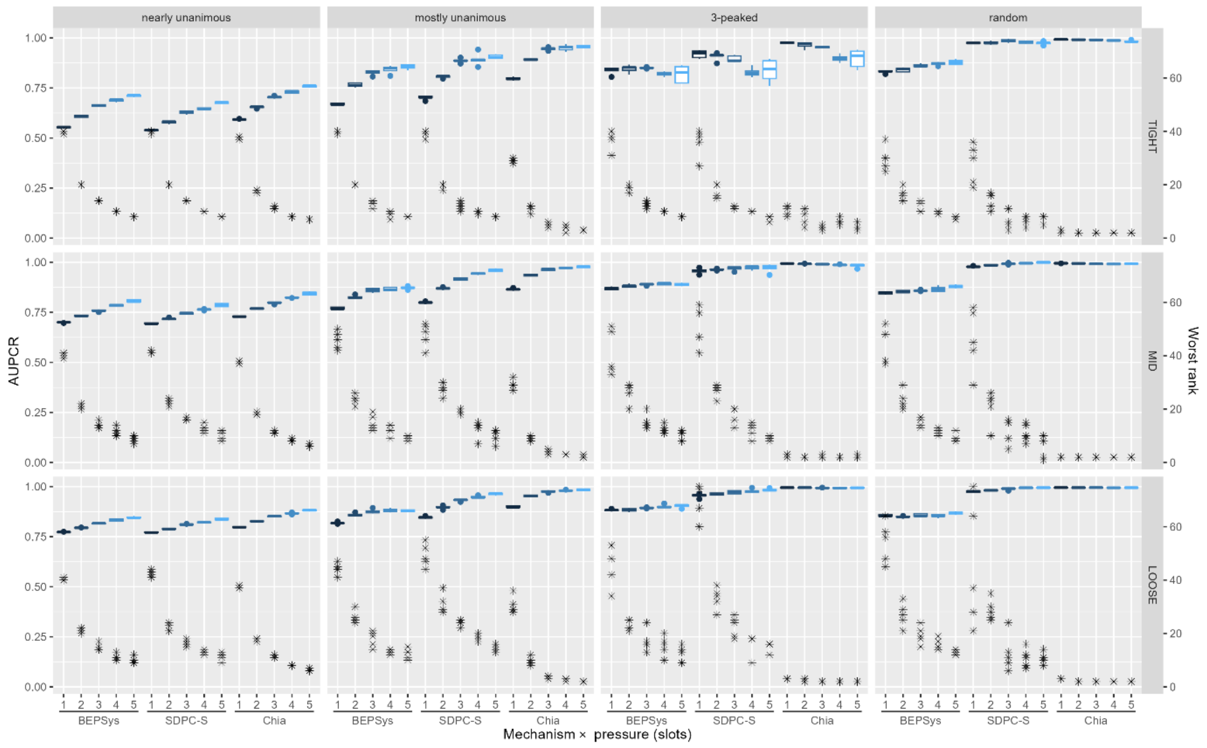


Figure 8: Quality metrics for the size experiment with the BEPSys, SDPC-S and Chiarandini mechanisms. Shows two metrics. Left: AUPCR, shown as lines/boxplot. Right: worst rank, shown as asterisks. Instances of ***200*** *students with three pressure scenarios: tight, middle and loose. The number of topics in the instance is determined by the pressure and number of slots.*

1) The Chiarandini mechanism has outperformed all other tested mechanisms. The differences are more pronounced in the bigger instances, less pressure and/or more students. Even more pronounced is the difference in worst-rank, which the OWA objective function minimizes.
2) Judging by the AUPCR metric, the SDPC-S mechanism manages to find better solutions. However, in terms of worst-rank it is often scores worse. This observation is in line with the typical serial dictatorship mechanism result – a long-tailed power law distribution.

5.6.2.2 Group size bounds

With this experiment we have explored the effect of the group size bounds by varying the lower and upper bounds and the range between them.

5.6.2.2.1 Setup

The setup is similar to the previous experiment. However, the SDPC-S mechanism was left out as it did not manage to find complete matchings in this experiment. Furthermore, we include only the Chiarandini mechanism for the *quality* analysis because its results are identical to the Fair mechanism due to lack of pregrouping and including the BEPSys mechanism we considered to be of lesser relevance. Each project topic is set to have a single slot of capacity.

5.6.2.2.2 Presentation

The runtimes and outcome quality plots have a similar structure. We again use the table-like, faceted arrangement, this time with the group lower bound as columns and rows. Each subplot shows the performance of the given mechanism for that specific group size bound.

For conciseness. The runtime plots contain all preferences models and show only the 200 and 600 student size instances.

The quality outcome plot is more advanced and also breaks down the results by the preferences model because it matters for the outcome quality analysis. For this experiment, each of the quality metrics are presented separately.

5.6.2.2.3 Runtime

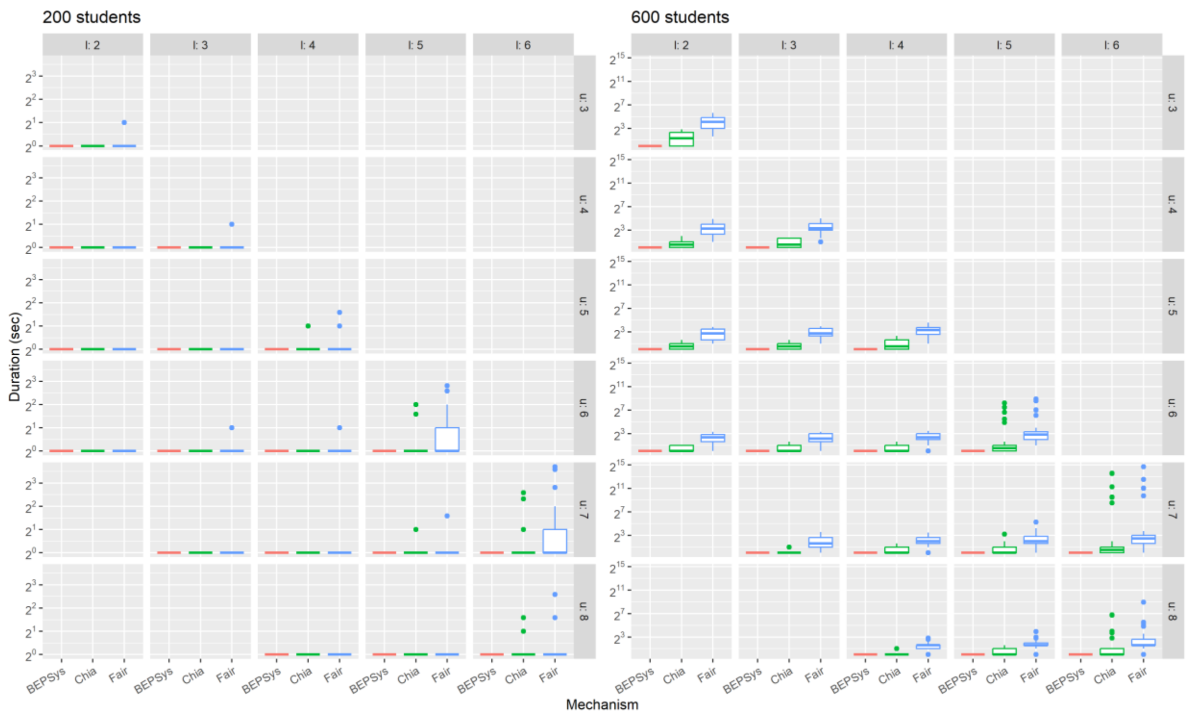


Figure 9: Runtimes of the group size bounds experiment. The group lower bound is varied over the columns. The group upper bound is varied over the rows. The results for each combination of group size bound and mechanism include all pressures and project preference types. The project topics in every instance have 1 slot.

Observations from Figure 9:

1) Widening the range lowers the difficulty of instances. This is seen while comparing results horizontally to the left or vertically down.
2) Increasing the group size bounds while keeping the range constant increases the required runtime. This is seen while viewing the results diagonally down to the right.

5.6.2.2.4 Quality

Figure 10 visualizes the quality metrics of the matchings found by the Chiarandini mechanism in the 200 students scenario with the realistic “3-peaked” project topics preference type with pressure set to ‘tight’ and al

Observations from Figure 10:

1) Bigger group size bounds show better outcome quality.
2) Wider group size bounds also show better outcome quality.

From this observation we realize that we have missed a critical detail for this experiment. Namely that if we vary the group size bounds, we decrease the student capacity of each topic and therefore less students may get their desired projects. This has not been compensated for in the experiment setup.

However, we do note that making the group size bounds wider is beneficial to the overall outcome satisfaction. We assume that a lower lower-bound makes it easier to assign students to a more niche, alternative project topic rather than finding a lower ranked but more commonly popular alternative.

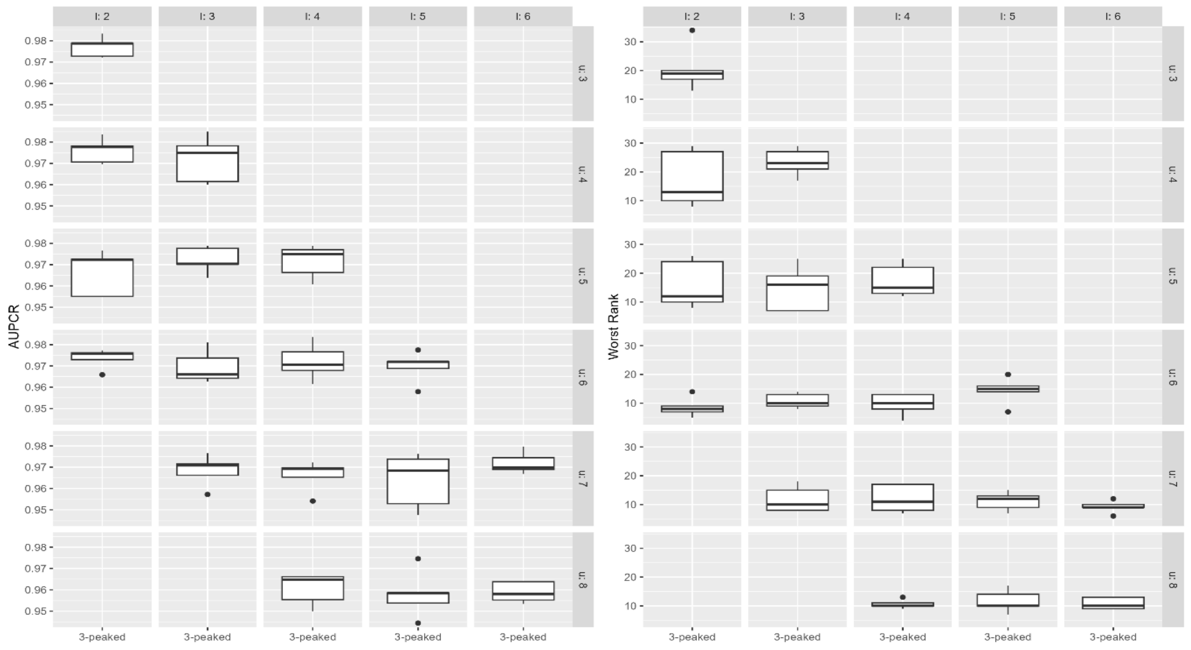


Figure 10: The quality metrics for the Chiarandini mechanism in group size bounds experiment. These results are filtered to contain only instances of 200 students, "tight" pressure and the "3-peaked" preference type. The project topics in every instance have 1 slot. The group lower bound – indicated by "l:" – is varied over the columns. The group upper bound – indicated by "u:" – is varied over the rows. Two metrics are presented: (left) The AUPCR and (right) the worst rank.

5.6.2.3 Pregrouping

In this final experiment we will be evaluating the effect of pregrouping on the Chiarandini and Fair mechanisms. Adding pregrouping to our generator leads us to three additional instance parameters: (1) the ratio of pregrouping students to solo students, (2) the project preference type of the pregrouping students and (3) the size of each pregrouping group.

The latter parameter allows us to vary how the pregrouping students rank the projects relative to the solo students. We suspect that the pregrouping students' preferences differ from the solo students, be it due to strategic behavior (picking a more niche topic that is still interesting to the group) or simply diverging preferences. Nevertheless, we suspect the preferences of the pregrouping students can have significant impact on the outcome. Due to time and space constraints, we have not evaluated the effect of pregrouping sizes with synthetic instances like we did in Section 5.5.2.3, all visualized results are with *max-sized* pregroups only.

5.6.2.3.1 Setup & presentation

The following charts present the rank profiles of the students stratified by the type of student: solo, pregrouped (together) and unsatisfied (that is, not together). A color is assigned to each type. The solo students are represented by the green color, the students that are *together* in yellow, and those that are *not together*, by purple.

The visualized result is an average of 5 synthetic problem instances in an attempt to approximate an expected outcome for that setting. Therefore, the y-axis indicates the *proportion* of students rather than indicating the absolute number of students. Because of the stochastic nature of the synthetic problem instances, comparisons and analysis should be kept at a general level, not on minute differences.

Due to space limitations, we limit the outcome quality results to instances with 200 students.

The *3-peaked* distribution is used for the solo students' project topic preferences. However, the preferences for the pregrouping students are determined by three scenarios:

- **Identical**: the pre-grouping students' project preferences are drawn from the same distribution as the solo students
- **Different**: the pre-grouping students' project preferences are drawn from a different distribution than those of the solo students. The distribution is of the same type for both types of students (3-peaked).

- **Mix**: the pre-grouping students' project preferences are drawn (per group) from either one or the other distribution (50%-50%), where one is the same distribution that is used for the solo students and the other being the exclusively pre-grouped students' distribution.

The outcome quality plots are of the table of subplots type. The pregrouping preference scenario is represented by the columns. The proportion of pregrouping is represented by the rows and consists of the proportions: 10 %, 30 %, 60%, and 90% pre-grouping students. This the first row with proportion 10% involves 180 solo students and 20 pre-grouped students (= 4 groups of 5 students each). The third row, 60%, involves instances of 80 solo students and 120 pre-grouped students (= 24 groups of 5).

We present two outcome quality plots, each relating to a project pressure scenario. The first, Figure 11, contains the results for the *medium* project pressure (0,75) scenario, the second, Figure 12, the *tight* (1,0) scenario. The number of slots for this experiment is kept constant at 1, therefore pressure determines the number of offered project topics only.

The plots were influenced by so-called split-violin plots, but all available packages for R for such plots make use of estimators and smoothing functions which we were not comfortable with using. Therefore, we implemented a discrete version made up of two bar-plots that oppose each other over the x-axis. The upward plots (+, positive values) represent the Fair mechanism results, the downward plots (-, negative values) represent the results for obtained with the Chiarandini mechanism. The negative sign is an artifact of visualization and should be ignored when interpreting the results.

The mechanisms were set to the *soft* pregrouping constraints to ensure feasibility.

5.6.2.3.2 Result - quality

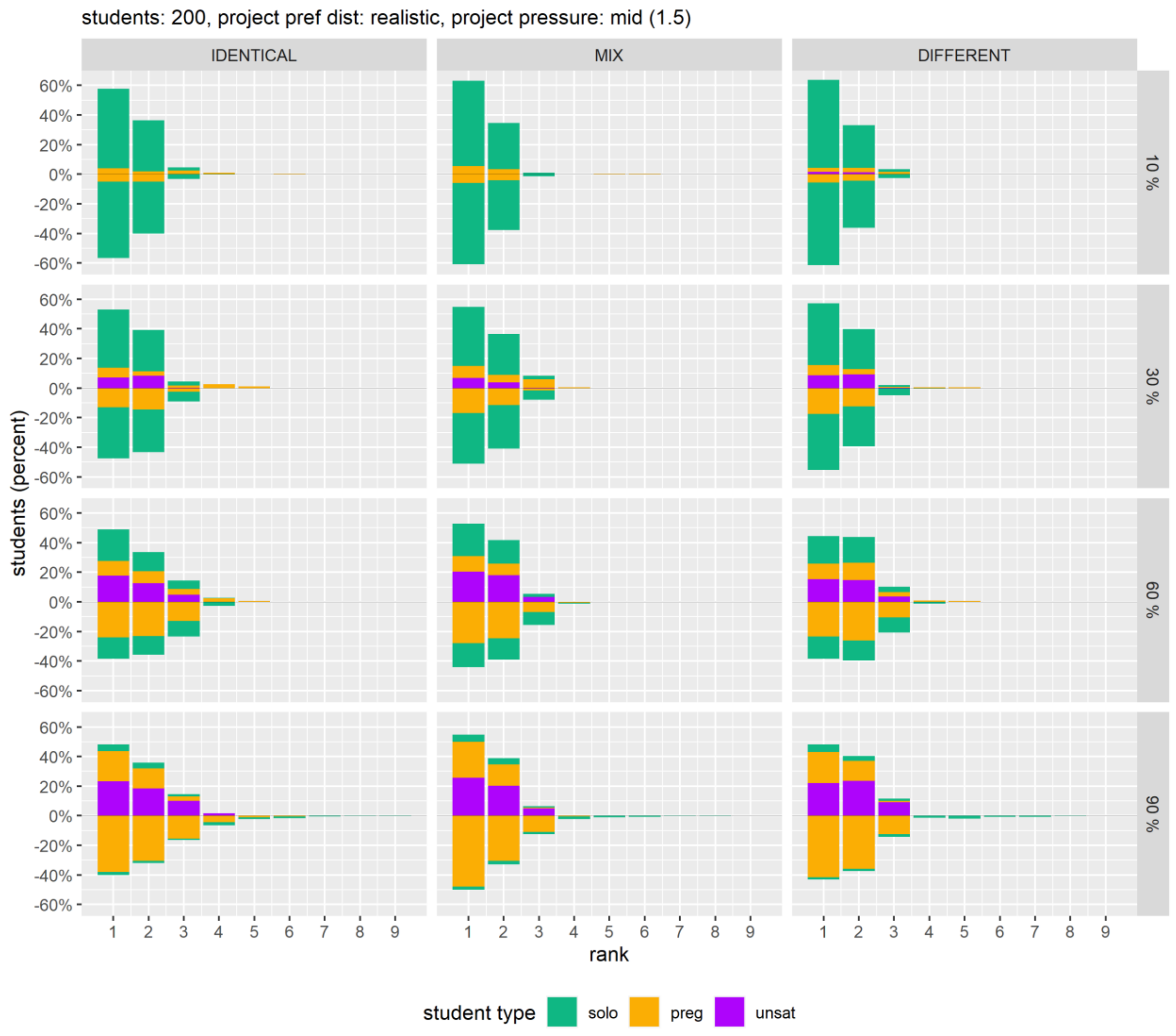


Figure 11: Pregrouping outcome quality. Instances with 200 students, "tight"' pressure, all pregroupings are of max size, single slot topics. The proportion of pregrouping students is indicated on the right (rows). The project

preference type of pregrouping students is indicated at the top (columns). The (+) upward plots (positive values) represent the Fair mechanism results, the (-) downward plots (negative values) represent the Chiarandini results. The negative sign is an artifact of visualization and must be ignored when interpreting the results.

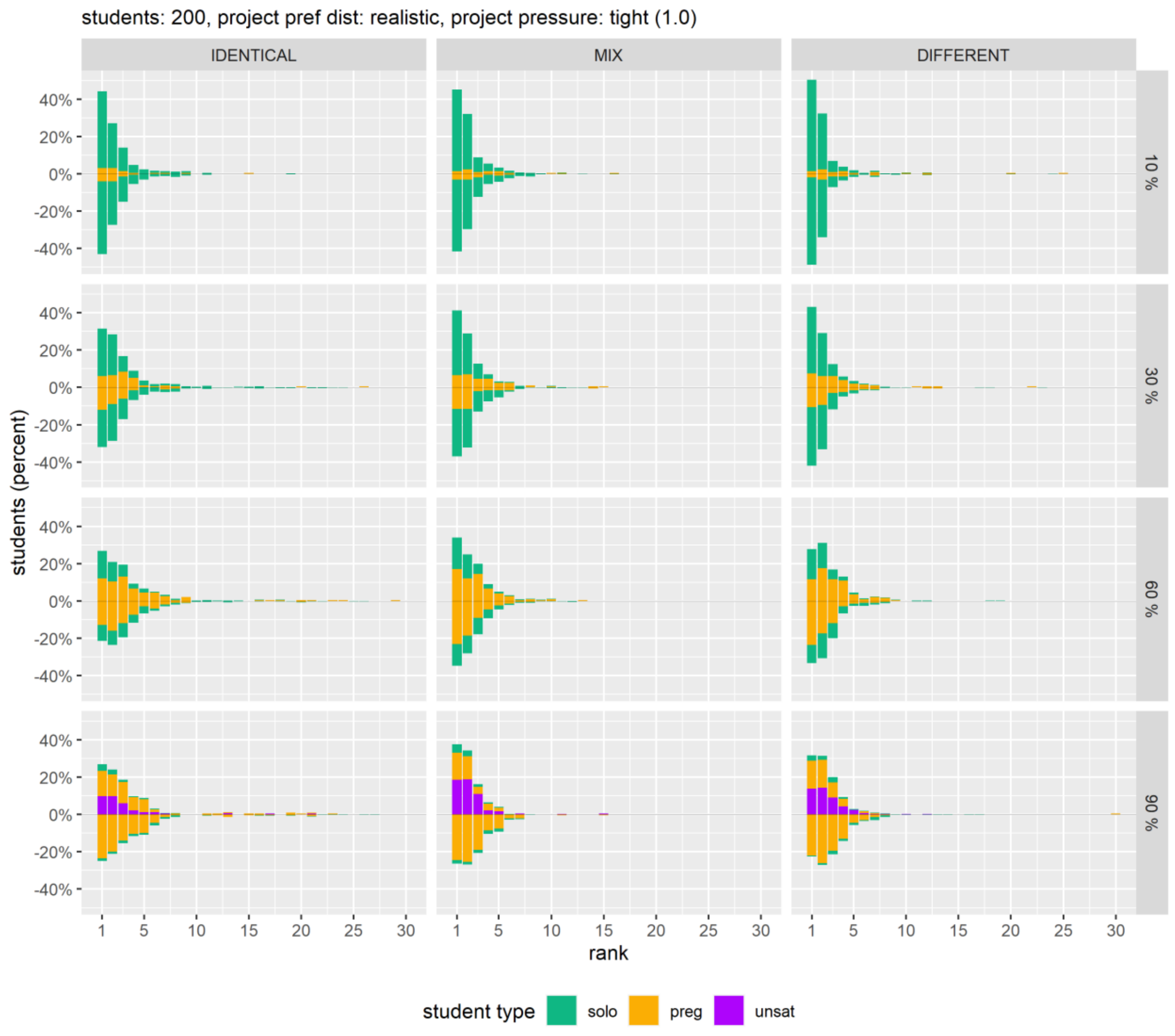


Figure 12: Pregrouping outcome quality. Instances with 200 students, "tight"' pressure, all pregroupings are of max size, single slot topics. The proportion of pregrouping students is indicated on the right (rows). The project preference type of pregrouping students is indicated at the top (columns). The (+) upward plots (positive values) represent the Fair mechanism results, the (-) downward plots (negative values) represent the Chiarandini results. The negative sign is an artifact of visualization and must be ignored when interpreting the results.

Observations:

1) The Fair mechanism does not manage to satisfy all pregroupings, whereas the Chiarandini does not seem to have a problem doing so. This is more pronounced in the medium pressure scenario over the tight one, where at already 10% pregrouping proportion it no longer can keep all pregroupings together.
2) Chiarandini is not always better for the pregrouping students, see for example the "tight, 90%, mix" scenario where there are less students and pregroupings assigned from the 3rd rank on. We think this can be explained due to large amount of pregrouping students being broken up.
3) It does make a difference for pregrouping students to draw their topic preferences from a different distribution. This can be observed when comparing the results horizontally. The distribution of ranks is getting denser towards the 1st rank when going from *IDENTICAL* to *MIX* to *DIFFERENT.* This does not hold for the 90% pregrouping scenario, there we can see that *MIX* of project topic preferences does better as the pregroups are split over two preference distributions so in that scenario there is the least amount of contention.
4) Both tight and mid scenarios are reminiscent of the results seen for the historical instances.

5.6.2.3.3 Result - runtime

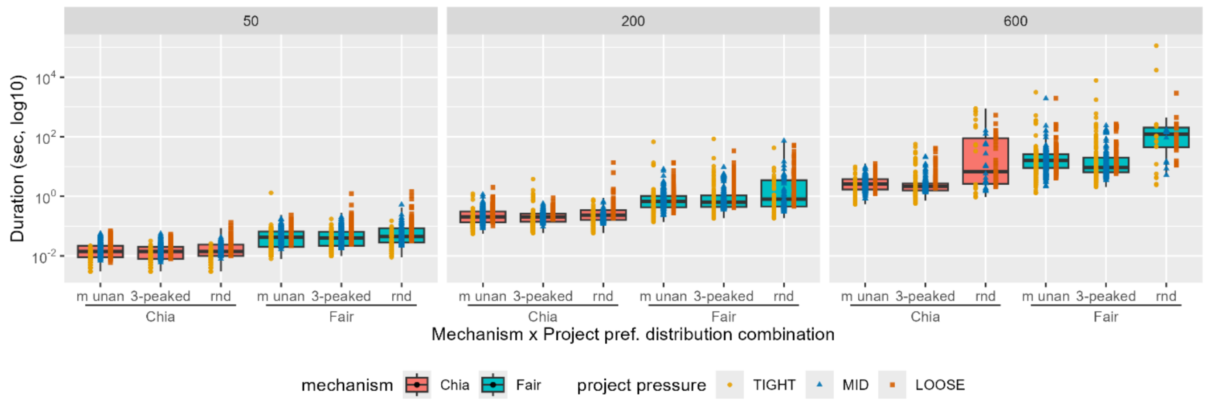


Figure 13: All runtime results of the synthetic pregrouping instances. Includes all pregrouping proportions (10, 30, 60, 90%), pregrouping students' preference types (same, mix, different), pressures (low, mid, high encoded respectively as squares, triangles and circles) and project preference types (mostly unanimous, 3-peaked/realistic and random).

Figure 13 visualizes the runtimes of instances solved during this experiment, including results that we have omitted from the quality analysis. To save space, we kept the stratification of the results to a minimum.

Observations and discussion:

1) Larger instances are clearly more difficult.
2) Fair mechanism is more computationally intensive than Chiarandini.
3) The effect of pressure on difficulty is not so clear. Although small *tight* instances are solved faster on average than mid and loose, *tight* instances can get significantly more difficult to solve than mid or loose instances as the instance size is increased. This is especially true for the Fair mechanism, as we see from the results of 3-peaked and random instances of size 600 students. A closer analysis of these results – not included here – revealed the runtimes to grow corresponding to the pregrouping proportion in the instance. This effect on the runtimes drops off at the 90% proportion, suggesting heterogenic instances to be more difficult than homogenic ones.
4) Our Fair mechanism required more than a day to solve the largest and most difficult instances. Specifically, instances with 600 students, tight pressure and random preferences. To put it in perspective, historical instances for TU Delft of size 250, 290 and 356 (CE11, 39 and 45, see Table 1) are solved between 0.25 and 1.5 seconds (see Figure 2). Comparing these results suggests that the synthetic instances on average are more difficult to solve than the historical samples that we have used in our evaluation. Judging by the synthetic results for the large instances with 600 students, we should expect runtimes with magnitude of minutes to solve with the Fair mechanism – assuming similar parameters as our historical instances. These findings come with a caveat: for the evaluation we have limited the Gurobi solver to using a single thread for solving. We expect that by removing this limit, the runtimes will decrease proportionally.

6 Discussion

Our experimental evaluation has provided insights into the performance of our old and new matching mechanisms. In this section we discuss the key findings and their implications.

Mechanism Performance

The Chiarandini and Fair mechanisms consistently outperform the prior BEPSys mechanism in terms of matching quality. This improvement is most beneficial to solo students. The BEPSys algorithm's two-step approach (first grouping, then assigning) creates inherent limitations that the combined optimization approach of the MILP-based mechanisms overcomes.

Between Chiarandini and Fair, the results align with their design goals. The Fair mechanism generally provides better outcomes for solo students at some expense to pregrouping students, though this trade-off varies across instances.

The leximin ordering objective function proved to produce excellent solutions. However, we observed some numerical stability issues with the OWA implementation of our chosen objective function, particularly in larger instances. This suggests that an alternative implementation of this objective or different ones should be explored.

Runtime Performance

Our new mechanisms have good runtime performance characteristics:

- Historical instances are solved within seconds.
- Synthetic evaluation has shown the mechanisms to be able to scale to larger instances but solving them may require a day of runtime. However, synthetic instances are computationally more demanding than similar historical instances. Furthermore, the Gurobi solver was configured to run with a single thread for evaluation purposes. Setting this back to default is likely to result in significantly shorter runtimes, although we have not evaluated this.
- The Fair mechanism is computationally more intensive than Chiarandini but is within acceptable limits.

The low runtime opens possibilities for expanding the problem model or implementing more interactive matching systems. The runtimes are sufficiently low that even large instances can be solved quickly enough in practice.

Pregrouping Considerations

Our investigation into pregrouping revealed several important insights:

- Relaxing pregrouping size restrictions does not consistently improve outcomes for solo students
- Pregrouping students often face a trade-off between togetherness and project satisfaction
- The Fair mechanism effectively prevents pregrouping from disadvantaging solo students, but sometimes at significant cost to pregrouping students

These findings suggest that conditional pregrouping might be a worthwhile addition to our problem model.

Strategic Behavior Considerations

Our work revealed several opportunities for strategic behavior within the matching system:

- Post-matching trading: Students were allowed to trade project topics after the matching was made public. This may incentivize students to strategically include highly desirable projects (beyond their actual preferences) in their preference lists (after their genuine top choices), hoping to trade for their truly desired project later. We recommend disallowing trading to prevent this behavior.
- Preference stuffing: In instances where project topics are overprovisioned, students who can identify unpopular projects can strategically fill their preference lists with these projects (after their genuine top choices) to increase their chances of receiving their most preferred topic. This is particularly problematic with rank-optimizing objective functions like leximin. We recommend limiting preference list length and avoiding severe overprovisioning.
- Inconsistent pregrouping preferences: Project Forum allows pregrouping students to submit different individual preference profiles, which we observed in historical data. This enables strategic behavior within groups, where some students might intentionally try to sway the group's project assignment toward their personal preferences. We recommend creating a specialized pregrouping signup flow that allows for and better explains the consequences of different preference submission options (aggregated, individual, conditional) so that students can make better informed choices.

Limitations

Our evaluation has some limitations worth noting:

- The synthetic instance generator, while useful, cannot fully capture the complexity of real-world problem instances
- The binary togetherness metric for pregrouping satisfaction may not fully capture the nuanced preferences students have regarding partial satisfaction of pregrouping wishes
- We have ex-post limited pregrouping in the older historical TU Delft problem instances to only allow pregroups of maximum size. These results may not completely represent real instances where students were only allowed to pregroup in maximum sized groups.

Despite these limitations, our evaluation provides strong evidence that both new mechanisms are significant improvements over the prior mechanism, with the Fair mechanism adding fairness between solo and pregrouping students for relevant settings.

7 Conclusion and future work

This thesis has investigated how to improve the group project matching algorithm of Project Forum at TU Delft. We identified two promising mechanisms – Chiarandini and our novel Fair variant – and evaluated them against the prior BEPSys algorithm using both historical and synthetic instances.

Our experimental results demonstrate that both MILP-based mechanisms significantly outperform the BEPSys mechanism in terms of matching quality and runtime performance (if compared to the original BEPSys). The Fair mechanism achieves its design goal of prioritizing solo students, although this comes at the expense of pregrouping students' project satisfaction and sometimes their togetherness.

The choice between the Chiarandini and Fair mechanisms depends on the specific priorities of the course. If the educational goals emphasize working with new people and developing interpersonal skills, the Fair mechanism may be preferable as it ensures solo students are not disadvantaged by pregrouping and discourages students from doing so. If maximizing overall satisfaction is the priority, the Chiarandini mechanism may be more appropriate, though it potentially allows pregrouping students to gain an advantage.

7.1 Future work

Other requirements can be handled by extending the problem model or changing the objective function. We have classified these directions as future work below:

7.1.1 Explore changes to objective functions

One finding during our evaluation is that the OWA implementation of the leximin objective does not guarantee optimal solutions. Alternative, correct implementation of the *generous maximum matchings,* such as an iterative search should be explored such as described in [12] in the paragraph preceding the OWA-based approach.

The iterative approach requires the solving of $worst(\sigma_\mu) - 1 \leq |P| - 1$ problems. We do not expect this to be an issue as the mechanisms are able to solve historical instances within a second, so if it takes just as long to solve each iteration then the resulting runtime is still acceptable. However, should the increase in runtime prove to be unacceptable then there are alternative objective functions that can be considered and evaluated. In order of similarity:

- Exponential weighting scheme
- AUPCR maximization
- Combined worst-rank and sum of ranks minimization.

7.1.2 Changes to the problem model

Other than the objective function, the problem definition can be extended with additional considerations and optimization criteria. We list some in order of demand.

Supervisor capacities

Supervisors can offer multiple project topics but their supervisory capacity is limited. This creates a challenge for the supervisors to estimate the desirability of the topics they offer ahead of time. If the topics prove too popular, the supervision for those topics may not be available and some topic slots may need to be scrapped after matching, creating unnecessary work for the course administration and affected supervisors. Furthermore, the TU Delft problem setting considers a single global group capacity setting, making offering each additional project topic a significant commitment for the supervisors and their research group. An easy first step is to use the more flexible SDU model with individual project topic slot capacities, this will put the responsibility for estimating demand on the supervisors or the course administration but potentially with better outcomes as a result.

A more advanced change is to model the supervisory capacities that their offered project topics share, so that no more groups can be assigned a project topic than the supervisory staff for those topics can handle in total. This offers additional possibilities; supervisors can offer more project topics than they have capacity for and/or offer more group slots for each topic. For example, supervisor A has a capacity for c_A= 3 and offers topic slots $t^A_{p_1,1}, t^A_{p_1,2}, t^A_{p_2,1}, t^A_{p_2,1}, t^A_{p_3,1}$ – meaning two slots for project topic 1 and 2 and one slot for project topic 3 – the resulting matching should match at most $c_A = 3$ groups to the topics offered by that supervisor.

Grouping preference model

Our grouping preference model is straightforward: students may submit pregrouping requests, though we cannot guarantee that these requests will be met. Alternative approaches are possible. For example, students can indicate who they prefer to work with and who they wish to avoid instead of, or in addition to, pregrouping. This may be desirable in some settings but needs to be determined. The actual desired pregrouping preference model for these settings also needs to be determined.

Partially ordered project topic preferences

Anecdotally we know students usually have a strong preference ordering between their very top choices, say first 3, and get more indifferent for the remaining topics. Some students may even be (nearly) indifferent between some topics for a high ranks as well. Given that we are optimizing for ranks, allowing ties into project topic preferences seems like a nice low hanging fruit.

Distributional constraints

An additional wish we have heard mentioned by the course administration is to allow so-called distributional constraints. These constraints help enforce distribution of students according to some defined property such as nationality, culture, native language, gender, experience, skills. Heterogenous groups may aid in achieving interpersonal educational goals.

Conditional pregrouping

Although we have introduced the conditional pregrouping variant in section 4.3.3 and have identified its potential usefulness in our experimental evaluation, we have not implemented or evaluated it. Nor is the variant fully fleshed out:

- We are uncertain about the fairness implications of it – do we consider the students pregrouped even if disbanded?
- How do we model their preferences? Do they have different project topic preferences for when pregrouped and when *not together*? Or do we assume the same and assume they prefer to be pregrouped for top-k, and otherwise try to obtain the top-k as *solo*?

This work we leave open for the next graduate student.

Student-proposed project topics

An interesting domain feature is that the students may approach supervisors and propose their own custom project topics that they can then undertake in a pregroup, if the proposal is accepted by the supervisor. The requirement for pregrouping stems from the inability of the matching system to handle such projects.

An extension to the matching system can make this process more accessible. The idea is that the proposed and accepted topic can only be opened if the proposing student(s) match it as well as sufficient additional number of students to meet the lower group bound. The proposing students would naturally be forced to rank the project topic as their top pick, but that is not a guarantee that enough additional students can be matched to it in an optimal matching. If the project is not open, the proposing students need to be matched to some other topic.

7.1.3 Other

An interactive matching system

It is hard to determine the perfect mechanism or objective function ahead of time for all possible instances. Sometimes a different mechanism or objective function leads to a matching that simply *looks* more desirable from the point of view of the course administration, that is more fair and/or optimal trade-off based on human judgement and intuition.

Because the proposed mechanisms have a sufficiently fast runtime, it is possible to solve an instance with various mechanisms, objective functions and configurable parameters (if any) and present the solution for selection by the course administration. One idea is to let the selection of the final outcome be anonymous, so that it is unknown which student is matched to what nor which mechanism or objective function generated it. An interesting experiment is to let the students themselves vote for their favored outcome.

8 Bibliography

[1] David J. Abraham, Robert W. Irving, Telikepalli Kavitha, and Kurt Mehlhorn. 2007. Popular Matchings. *SIAM J. Comput.* 37, 4 (January 2007), 1030–1045. https://doi.org/10.1137/06067328X

[2] Kolos Csaba Ágoston, Péter Biró, and Iain McBride. 2016. Integer programming methods for special college admissions problems. *Journal of Combinatorial Optimization* 32, 4 (January 2016), 1371–1399. https://doi.org/10.1007/s10878-016-0085-x

[3] Kolos Csaba Ágoston, Péter Biró, and Richárd Szántó. 2018. *Stable project allocation under distributional constraints*. https://doi.org/10.1016/j.orp.2018.01.003

[4] A. A. Anwar and A. S. Bahaj. 2003. Student project allocation using integer programming. *IEEE Trans. Educ.* 46, 3 (January 2003), 359–367. https://doi.org/10.1109/TE.2003.811038

[5] Ashwin Arulselvan, Ágnes Cseh, Martin Groß, David F. Manlove, and Jannik Matuschke. 2018. Matchings with Lower Quotas: Algorithms and Complexity. *Algorithmica* 80, 1 (January 2018), 185–208. https://doi.org/10.1007/s00453-016-0252-6

[6] Haris Aziz, Felix Brandt, and Paul Harrenstein. 2013. Pareto optimality in coalition formation. *Games and Economic Behavior* 82, (November 2013), 562–581. https://doi.org/10.1016/j.geb.2013.08.006

[7] Péter Biró, Tamás Fleiner, Robert W. Irving, and David F. Manlove. 2010. The College Admissions problem with lower and common quotas. *Theoretical Computer Science* 411, 34 (January 2010), 3136–3153. https://doi.org/10.1016/j.tcs.2010.05.005

[8] Elizabeth Bodine-Baron, Christina Lee, Anthony Chong, Babak Hassibi, and Adam Wierman. 2011. Peer Effects and Stability in Matching Markets. In *Algorithmic game theory: 4th International Symposium, SAGT 2011, Amalfi, Italy, October 17-19 2011 : proceedings / Giuseppe Persiano (ed.)*, Giuseppe Persiano (ed.). Springer, Heidelberg, 117–129.

[9] Steven J. Brams and Alan D. Taylor. 1996. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press.

[10] S. Branzei, Tomasz Michalak, Talal Rahwan, K. Larson, and N. R. Jennings. 2013. Matchings with externalities and attitudes. May 2013. 295–302. Retrieved June 2, 2021 from https://eprints.soton.ac.uk/346853/

[11] Justin Burkett, Francis X. Flanagan, and Amanda L. Griffith. 2018. Allocating group housing. *Soc Choice Welf* 50, 4 (April 2018), 581–596. https://doi.org/10.1007/s00355-017-1097-x

[12] Marco Chiarandini, Rolf Fagerberg, and Stefano Gualandi. 2019. Handling preferences in student-project allocation. *Ann Oper Res* 275, 1 (January 2019), 39–78. https://doi.org/10.1007/s10479-017-2710-1

[13] Franz Diebold and Martin Bichler. 2017. Matching with indifferences: A comparison of algorithms in the context of course allocation. *European Journal of Operational Research* 260, 1 (July 2017), 268–282. https://doi.org/10.1016/j.ejor.2016.12.011

[14] Bhaskar Dutta and Jordi Massó. 1997. Stability of Matchings When Individuals Have Preferences over Colleagues. *Journal of Economic Theory* 75, 2 (August 1997), 464–475. https://doi.org/10.1006/jeth.1997.2291

[15] Edith Elkind, Neel Patel, Alan Tsang, and Yair Zick. 2020. Keeping Your Friends Close: Land Allocation with Friends. *arXiv:2003.03558 [cs]* (March 2020). Retrieved June 30, 2021 from http://arxiv.org/abs/2003.03558

[16] Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. Retrieved from https://www.gurobi.com

[17] Virginia Hogan. Matching In Groups: A Theoretical and Empirical Study. 68.

[18] Naoyuki Kamiyama. 2013. A note on the serial dictatorship with project closures. *Operations Research Letters* 41, 5 (January 2013), 559–561. https://doi.org/10.1016/j.orl.2013.07.006

[19] D. Manlove. 2013. *Algorithmics Of Matching Under Preferences*. World Scientific Publishing Company. Retrieved from https://books.google.nl/books?id=jPO6CgAAQBAJ

[20] Daniel Monte and Norovsambuu Tumennasan. 2013. Matching with quorums. *Economics Letters* 120, 1 (January 2013), 14–17. https://doi.org/10.1016/j.econlet.2013.03.007

[21] Antonio Nicolò, Arunava Sen, and Sonal Yadav. 2019. Matching with partners and projects. *Journal of Economic Theory* 184, (November 2019), 104942. https://doi.org/10.1016/j.jet.2019.104942

[22] Pablo Revilla. 2007. Many-to-One Matching When Colleagues Matter. *SSRN Journal* (2007). https://doi.org/10.2139/ssrn.1014549

[23] Natsumi Shimada, Natsuki Yamazaki, and Yuichi Takano. 2020. Multi-objective Optimization Models for Many-to-one Matching Problems. *Journal of Information Processing* 28, 0 (2020), 406–412. https://doi.org/10.2197/ipsjjip.28.406

[24] Ronald R. Yager. 1997. On the analytic representation of the Leximin ordering and its application to flexible constraint propagation. *European Journal of Operational Research* 102, 1 (October 1997), 176–192. https://doi.org/10.1016/S0377-2217(96)00217-2

9 Appendix A - Preferences generators visualization

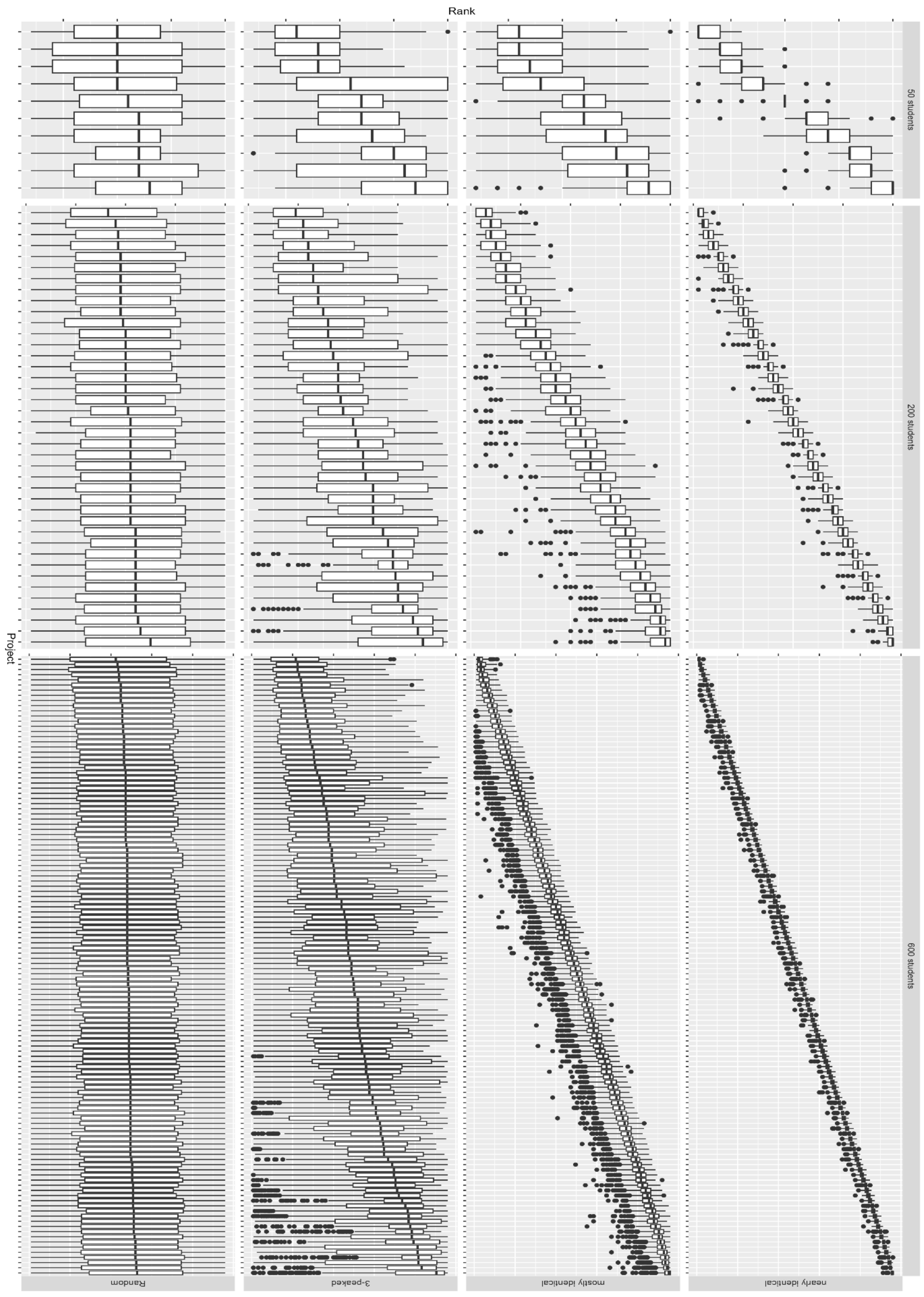


Figure 14: Generated project preferences by type and scale. The visualization plots how the project topic is ranked across all generated project preferences. The topics are sorted by their median rank, this can be seen as a proxy for its desirability.

10 Appendix B – AIDM Project: Improving BEPSys

CS4210-B: Making Optimal Project Groups

Hidde Bolijn (4750799) Philippe Louchtch (4697685)
Max de Krieger (4705483)

June 24, 2019

1 Introduction and background

The Technical University of Delft (TU Delft) is growing. Each year, more and more students start their studies. Some study programmes have special courses where students do group work on large projects. Such courses can have a large menu of topics (or projects) to choose from. The students must also form groups. For the Computer Science department, this is already automated through an application called *BEPSys* (Bachelor end-project system). However, other departments also have a need for such a tool. There exists a large body of existing research on matching, social choice, game theory and social welfare optimization.

Before the use of an automated student group project matching tool is expanded throughout the rest of the TU Delft university, the performance and functioning of the current tool, BEPSys, must be investigated and evaluated.

The goal of our research is summarized as follows:

- Investigate performance and shortcomings of BEPSys tool
- Propose, implement and evaluate improvements to BEPSys tool
- Propose, implement and evaluate alternative or novel techniques for solving the TU Delft student group projects problem
- Identify missing parts of existing theory for the TU Delft student group project problem
- Build a framework for continued future research, experimentation and deployment for the TU Delft group projects problem

The unique aspect about the problem we face is the fact that students are able to give multi-dimensional preferences. In reality this means that students want to be grouped with their friends while also desiring a project ranked high in their project preference profile.

2 Problem definition

Formally, the TU Delft student group projects problem in the context of our research is defined as follows:

- Students S [1]
- Projects P, where each project $p \in P$ has room (slots) for one or more groups
- Minimum group size m_{min} and maximum group size m_{max}
- Each student $s \in S$, has:
    - A linear ranking of projects (*project preference profile*)
    - A set of no more than $m - 1$ preferred group partners (*group preference* or *peer preferences*)

[1] also referred to as Agents in the code

The goal is to find a mapping $\mu : S \rightarrow P$, maximizing the sum of student satisfaction, subject to group size constraints and the project slot constraints. Student satisfaction is function of the rank of the project assigned and the group he or she ends up in. This is highly subjective, however, BEPSys maintainers have indicated that the student's *peer satisfaction* has been found more important than the assigned project satisfaction.

2.1 Definitions

Before continuing, we would like to give a concise summary of some definitions used by us.

Rank the rank or index of some item in a linear list. In context of our project usually denoting the position of the assigned item in the agent's preference profile over the items.

Peer preferences A student can indicate with which other students he or she would like to be grouped with, the preferred partners. This is not a linear preference profile. Merely a set of students, but not larger than m_{max}, with whom the student wants to be grouped with.

Friends The peer preferences of a student. Also referred to as friends list.

Project The object with which an agent needs to be matched with

Project slot The projects have room for multiple groups. We model this as project slots. The agent then has an equal preference between all slots for a given project in its project preference profile

Lonely (student) A student without peer preferences

Peer satisfaction Percentage of peer preference honored of a student.

3 Related work

Some of the well known matching mechanisms are *stable-matching*, *deferred acceptance*, *top trading cycle* and *(randomized) serial dictatorship*. An overview of most mechanisms and properties of matching mechanisms can be found in [4]. This work provides the fundamental basis of the problem we are dealing with, namely assigning a finite amount of resources to agents with a preference profile.

The research problem that comes the closest to ours, is the allocation of courses to students. This is a simpler version where each student has a preference over courses and each student is to be linked to at most one course. A summary and ranking on various metrics of matching mechanisms on this simpler problem is given in [1]. The difference between this work and our is the basis of the problem. In [1], agents are not grouped but directly assigned to a resource, and agents are indifferent to who else is taking a certain course.

However, the TU Delft student group project problem is multi-dimensional. That is, students not only have preferences over the projects but also preferences with whom they would like to work together with.

4 Techniques implemented/investigated

The students, in general, strongly prefer working in a good, well functioning group over having just their top project preference satisfied. A simple yet intuitive approach is a two-step procedure. That is reduce the multi-objective problem into subsequent, single objective matching problems:

1. Form groups based on group preferences and project preference compatibility [*group forming*]

2. Calculate group's project preference through aggregating the project preferences of its members [*preference aggregation*]

3. Assign the formed groups to projects [*project matching*]

In the following subsections we will introduce and outline the implemented variations of group-forming, preference-aggregation and project-matching algorithms. Lastly, we will outline two novel, non-two-step methods.

4.1 Forming groups

In this section, all implemented group forming methods will be discussed.

4.1.1 BEPSys group forming algorithm

The BEPSys group forming algorithm consists of three main steps:

Form groups from cliques - Students that have the exact same friend list (excluding themselves of course) are joined together in a *tentative* group (a group that does not necessarily obey the minimum and maximum group size yet).

Best match ungrouped - From all the students that are not part of tentative groups yet, construct multiple *possible* groups based on their friends list. Pick the "best" groups (according to how many people submitted each other as friends) from the possible groups and add them as a tentative group.

Merge groups - Calculate how many groups can be of maximum size, then merge all tentative groups to *final* groups, based on the similarity of the project preferences of tentative groups. If there are still tentative groups while having reached the calculated amount of maximum sized groups, force them to merge into smaller groups. If the resulting set of final groups is not conform to the group size constraints, destroy the last two final groups and try to merge them back until the resulting set of final groups does fit the group size constraints.

This algorithm is based on the expert knowledge that students prefer to end up in a good or fun group (with their friends) over having their top project preferences honored. Hence, it optimizes for peer satisfaction and attempts to mix in lonely or merge small friends-only groups that are most compatible.

4.1.2 Improved BEPSys algorithm

The last step of the BEPSys algorithm is rather shaky, in the sense that it is not guaranteed that a solution will be given as the algorithm can end up in an infinite loop, endlessly trying to break and merge the last few groups to create a set of groups conforming to the group size constraints. This lead us to introduce the improved BEPSys algorithm.

Form groups from cliques - The same as in the first step of BEPSys, with the additional constraint that the created clique can not be bigger than the maximum group size. If this is the case, no clique is created at all.

Best match ungrouped - The same as in the second step of BEPSys.

Calculate plausible set - Calculate a plausible set of groups that conforms to the amount of students and size constraints. If this is not possible, an error is returned.

Merge groups - Merge all groups as in the BEPSys algorithm, except the calculated set is used as a strict guideline. If a group merges and is considered final, the algorithm will take into account how many of that group size have been formed. If the size of the group that wants to merge is unavailable, it will refrain from merging.

The key to the improved BEPSys is determining a set which keeps the size constraints in mind. Because the algorithm adheres to the calculated set, either it will know if it is not possible, or a solution will be given.

The way a plausible set is calculated is by dividing the students into as many large groups as possible. If there is a remainder, the groups are evenly reduced to bump the remainder up to the minimum group size. If this fails, the maximum group size is reduced by 1 and the algorithm repeats. A more detailed pseudocode is given below.

Pseudocode

Algorithm 1 Calculate a plausible set

```
$S \leftarrow$ #students
$m_{max} \leftarrow$ maximum group size
$m_{min} \leftarrow$ minimum group size
$G \leftarrow \emptyset$
while $true$ do
    $groups_{max} \leftarrow \lfloor \frac{S}{m_{max}} \rfloor$            ▷ Create as many groups of max size possible
    $remainder \leftarrow S \mod m_{max}$
    if $remainder = 0$ then
        $G[m_{max}] \leftarrow groups_{max}$                              ▷ All groups are max size
        return $G$
    else if $m_{max} = m_{min}$ then                          ▷ If there is remainder while the size is fixed
        return $\emptyset$                                             ▷ Matching not possible
    else if $remainder \geq m_{min}$ then                    ▷ If remainder conforms to size constraints
        $G[m_{max}] \leftarrow groups_{max}$
        $G[remainder] \leftarrow 1$
        return $G$
    else
        $\Delta_{min} \leftarrow m_{min} - remainder$                        ▷ Distance towards min group size
        if $groups_{max} \geq \Delta_{min}$ then         ▷ Enough groups to bump remainder up to min size
            $groups_{max} \leftarrow groups_{max} - \Delta_{min}$
            $G[m_{max}] \leftarrow groups_{max}$
            $G[m_{max} - 1] \leftarrow \Delta_{min}$
            $G[m_{min}] \leftarrow G[m_{min}] + 1$
            return $G$
        else
            $m_{max} \leftarrow m_{max} - 1$
        end if
    end if
end while
```

4.1.3 Combined preference grouping

In an attempt to reduce our multi-dimensional preference problem to an easier, single-objective problem, we implemented a *combined preference grouping* method where the peers preferences and project preferences of students are aggregated into one single combined preference profile. The outcome of this algorithm is a grouping of students and thus says nothing about the preference aggregation or project assignment.

First, a weight must be attached to the peer preferences and project preferences that sum up to 1. In this project we chose 0.5 for peer preferences and 0.5 for project preferences. The combined preference profile of a student is computed using Borda count (see section 4.2.1 for a detailed explanation). The aggregated project preferences of peers get a half point for each alternative it is ranked above, and the project preferences of the student get a half point for each alternative it is ranked above. In the case where a student did not give any peer preferences, the combined preference profile is the same as the project preference profile.

A concrete example of this computation is given here. Let students $S = \{a, b, c\}$ and projects $P = \{1, 2, 3, 4\}$. The preference profiles are as follows:

a: $1 \succ 2 \succ 3 \succ 4$
b: $4 \succ 1 \succ 3 \succ 2$
c: $4 \succ 1 \succ 3 \succ 2$

Let us say that student a has declared student b and c as his peers. The Borda count of the combined preference profile of student a is then as follows:

Project 1: $0.5 * 3$ (from student a) $+0.25 * 2$ (from student b) $+0.25 * 2$ (from student c) $= 2.5$
Project 2: $0.5 * 2$ (from student a) $+0.25 * 0$ (from student b) $+0.25 * 0$ (from student c) $= 1$
Project 3: $0.5 * 1$ (from student a) $+0.25 * 1$ (from student b) $+0.25 * 1$ (from student c) $= 1$
Project 4: $0.5 * 0$ (from student a) $+0.25 * 3$ (from student b) $+0.25 * 3$ (from student c) $= 1.5$

Ties are broken randomly so the final combined preference profile of student a is $1 \succ 4 \succ 2 \succ 3$ or $1 \succ 4 \succ 3 \succ 2$.

Once all the combined preference profiles of students are computed, the Kendall tau distance (as explained in [2]) from each student's profile to every other student's profile is computed.

The final step of this algorithm is a serial dictatorship, where a student picks other students of which the combined preference profile differs the least, until the desired group size is reached. We chose to not do this serial dictatorship in a random order, but instead ordered the students by amount of peers they defined. The reason behind this simple heuristic is to make sure that students that have not given any peer preferences, do not "steal" students that have.

Pseudocode

Algorithm 2 Combined Preferences

```
$S \leftarrow$ students
$P \leftarrow$ projects
$Pref \leftarrow$ student preference profile          ▷ $Pref_{s_p}$ is the rank of project $p$ for student $s$
$Diff \leftarrow$ empty 2-dimensional matrix                      ▷ Used for differences
$G \leftarrow$ empty collection of groups
Compute combined preferences as described above
for $1 \leq i \leq |S|$ do
    for $1 \leq j \leq |S|$ do
        if $i = j$ then
            continue
        end if
        $diffsum \leftarrow 0$
        for $1 \leq p \leq |P|$ do
            $diffsum \leftarrow diffsum + |Pref_{i_p} - Pref_{j_p}|$        ▷ Add rank difference of project p
        end for
        $Diff_{i_j} \leftarrow diffsum$
    end for
end for
Sort students by amount of given peer preferences
for $1 \leq i \leq |S|$ do
    if $S_i \in G_x$ for some $x$ then                      ▷ If student is already grouped
        continue
    end if

    Add $S_i$ to new group $G_i$
    Sort $Diff_i$ in ascending order

    for $1 \leq m \leq$ available group size do     ▷ Separate algorithm for available group sizes
        if $S_i \in G_x$ for some $x$ then                  ▷ If student is already grouped
            continue
        end if
        Add student of $Diff_{i_m}$ to group $G_i$              ▷ Add student with least difference
    end for
end for
return G
```

4.2 Aggregating preferences

A group is often composed of students that do not have identical preference profiles. In a "make groups first, then match groups to projects" approach, the project preferences of all members need to be aggregated in order to match the group to a project.

We investigated two techniques for aggregating project preferences. This problem is researched in the voting mechanisms field and thus we took two mechanisms from this field that enjoy different properties.

Ideally, we want a voting rule that strikes a good balance between prioritizing most preferred options as well as not marginalizing agents whose preferences are wildly different from the rest of the group. Such as the case when a *lonely* student is grouped with those who are friends among themselves. In the general case this is difficult to achieve. What is ethical? How do we prioritize and give weight? For this reason, Condorset consistency is not necessarily desired here. Strategic behaviour is difficult due to other factors making it difficult to predict with whom an agent will be grouped with.

However, one key property that is important to us is tractability.

Nevertheless, we have implemented two simple yet somewhat contrasting methods.

4.2.1 Borda count

Aggregating preferences using Borda count is the most intuitive method. It gives an alternative a point for each alternative it is ranked above by a voter. Computing this for our problem, is simply summing up the ranks for each project given by each student and finally sorting this list in a descending order of points. Ties in summed up ranks are broken randomly.

4.2.2 Copeland's method

Copeland's method is another voting mechanism that is explained in great detail in [3]. In short, the method orders alternatives based on pairwise victories minus pairwise defeats. One nice property of the Copeland's method is that, if there exists a Condorcet winner (also explained in [3]), this alternative will be ranked number one. We use this method to not only get the highest ranked alternative but instead form a total ordering over all alternatives. Ties in pairwise victories minus defeats are broken randomly.

4.3 Project matching

The two project matching algorithms we implemented will be described in the subsections below.

4.3.1 Minimum-cost flow problem

Each project has a set amount of slots, that is, groups that can have this project assigned. From the group members, a group preference profile is computed as explained in section 4.2. Hence, a group is like a single agent with a linear preference profile.

This is modeled as a Minimum-cost Flow Problem (max-flow with min-cost search setting) between the groups and project slots. For each group, a directed, weighted edge is created between it and the project slots of each project in its preference profile. The edges in MaxFlow graphs have a *capacity* property, this property signifies how much *flow* can be directed through the edge (pipe). Furthermore, the pipe can have *cost* associated with it representing the cost of using each unit of capacity of said pipe. The goal of the MinCost MaxFlow problem is to maximize the flow from source to sink whilst minimizing the costs for doing so. of an edge between the group and project slot is the rank of the project in the preference profile.

4.3.2 Randomized serial dictatorship

Randomized Serial Dictatorship in an easy to implement matching mechanism where, in a random order, agents get to pick their most preferred available resource. In our case, we randomize the order of the formed groups and let them pick their most preferred project that still has one or more open slots.

4.4 Matching students to projects directly

During our experiments we have noticed the group-first algorithms work very well at maximizing *peer satisfaction.* However, the *lonely* students have little to be satisfied about. They have no peer preferences and often end up in projects that they rank low.

We know that in general, students significantly prefer good groups over having most preferred project assigned. But for lonely students, the notion of being *good group* is unmeasurable to us and any algorithm without more data. We can however, prioritize trying to accommodate the project preferences of these students. We do this by matching students directly to project first using MinCost MaxFlow. The matching algorithm adheres to maximum project capacity constraints but not to group sizes constraints. Turning this matching into a final, valid group matching is not trivial. A naive attempt at doing so is what we call *Iterative Least-Preferred Project Pruning* and is introduced below.

4.4.1 Iterative least-preferred project pruning (ILPPP)

In order to obtain better upper bounds on problem instances, we have experimented with matching students directly to project first. When the problem instances have a large choice of projects, the resulting matching when seen as a distribution of students per project, is a long tailed one. There are a couple of very popular projects, often due to friends having aligned preference profiles, and many other projects have only one or two students assigned.

Intuitively, it makes sense to re-assign the students with (almost) unique assigned project to some other preferred project they have in common.

As the name suggests, the algorithm iteratively prunes a *least-preferred* project[2], re-matches students to the remaining process and repeats this process until there are no projects to which less than m_{min} students are assigned.

However some projects are tied in terms of least preferred and we must prune one. Therefore, we branch at this point and try removing each of the tied least-preferred projects in turn.

The following is a brief pseudo-code outline of the algorithm. It has been simplified for clarity, the implementation is a parallel branch-and-bound algorithm with dynamic programming.

[2]a project whose removal has the least impact on some metric, in our case AUPCR of the ranks of the assigned projects to students

Algorithm 3 Iterative Least Preferred Project Pruning

```
function ILPPP MATCHINGS($projects$, $students$, $m_{min}$, $m_{max}$)
    $matchings \leftarrow$ match $students$ to $projects$ with MinCostMaxFlow
    $metric \leftarrow aupcrStudents(matchings)$
    if $metric \leq bestSoFar$ then
        return $\varnothing$
    end if

    if no project in $matchings$ exists with $\leq m_{min}$ students assigned then
        if $formGroups(matchings, m_{min}, m_{max}) \neq \varnothing$ then          ▷ without remainder
            return $(metric, matchings)$
        end if
    end if

    $solutions \leftarrow \emptyset$
    for all $lp \in equallyLeastPopularProjects(matchings)$ do
        $solution \leftarrow ILPPP(projects \setminus lp, students, m_{min}, m_{max})$
        $solutions \leftarrow solutions \cup solution$
    end for
    return $best(solutions)$
end function

function EQUALLYLEASTPOPULARPROJECTS($matchings$)
    $projects \leftarrow projectsIn(matchings)$
    $agents \leftarrow agentsIn(matchings)$
    $results \leftarrow \emptyset$
    for all $p \in projects$ do
        $m \leftarrow$ match $students$ to $projects \setminus p$ with MinCostMaxFlow
        $results \leftarrow results \cup (aupcrStudents(m), p)$
    end for
    $grouped \leftarrow$ group $results$ by AUPCR values
    $eqLeastPreferred \leftarrow$ projects in $grouped$ with highest AUPCR value
end function
```

Students to project matching algorithm (by solving it as a Minimum-cost Flow Problem) By matching students to projects directly, we can obtain an upper bound on an optimal assignment.

Ideas for improvement:

- Some students are *indifferent* with respect to the assignment project and can be used as wildcards during group creation. When algo fails to create groups from a cluster, these wildcard-students can be assigned a different project (if the remainder is wildcard, move them to other project; if other projects have wildcards, move them into project needing more students to form groups)
- Form groups as soon as able and prune these students from search as well as update project capacities

5 Experimental results

In this section we discuss the results obtained. We compare all combinations of the group forming, preference aggregation and project assignment algorithms. Next to this, we compare these results to the overarching method implemented.

All experiments are done on historical data obtained from BEPSys. This data is divided in three different *course editions*, simply meaning different courses BEPSys have been used for.

5.1 The metrics

There is no single perfect metric to evaluate our results with. However, we can plot the distribution of "rank" the student/group has. For both the students (the rank of the project assigned to the team he/she is part of, in his/her linear preference profile) and group (the rank of the assigned project in the aggregated preference profile created from its members).

Intuitively, we wanted to measure the "skewness" of the distribution of ranks of the assigned projects. Where the more heavily "right skewed" a distribution is better the matching. A single metric that models this is the *Area Under Profile Curve Ratio* (often abbreviated to AUPCR), as explained in [1].

Ideally, the profile curve would be a straight horizontal line which occurs when every student gets their rank one choice. The difference between the ideal horizontal line and the actual profile curve is captured by the AUPCR, defined to be the ratio of the integral under the curve and the total area.

5.2 Tables and visualization

In tables 1, 2 and 3 below, the results of all experiments can be found. Highlighted in boldface is the largest obtained value(s) for the corresponding course edition.

	Group AUPCR	Student AUPCR	Average group satisfaction
BEPSysFixed_Borda_ILPPP	0.998	**0.984**	0.494
BEPSysFixed_Borda_MaxFlow	0.999	0.951	0.374
BEPSysFixed_Borda_RSD	0.999	0.951	0.374
BEPSysFixed_Copeland_ILPPP	0.992	**0.984**	0.494
BEPSysFixed_Copeland_MaxFlow	**1.0**	0.943	0.374
BEPSysFixed_Copeland_RSD	**1.0**	0.943	0.374
CombinedPreferences_Borda_MaxFlow	**1.0**	0.947	**0.548**
CombinedPreferences_Borda_RSD	**1.0**	0.947	**0.548**
CombinedPreferences_Copeland_MaxFlow	0.998	0.924	**0.548**
CombinedPreferences_Copeland_RSD	0.998	0.924	**0.548**

Table 1: All results for course edition 3

	Group AUPCR	Student AUPCR	Average group satisfaction
BEPSys_Borda_MaxFlow	0.956	0.910	**1.0**
BEPSys_Borda_RSD	0.912	0.873	**1.0**
BEPSys_Copeland_MaxFlow	0.960	0.912	**1.0**
BEPSys_Copeland_RSD	0.927	0.881	**1.0**
BEPSysFixed_Borda_ILPPP	0.978	**0.964**	0.862
BEPSysFixed_Borda_MaxFlow	0.956	0.910	**1.0**
BEPSysFixed_Borda_RSD	0.921	0.879	**1.0**
BEPSysFixed_Copeland_ILPPP	**0.967**	**0.964**	0.862
BEPSysFixed_Copeland_MaxFlow	0.960	0.923	**1.0**
BEPSysFixed_Copeland_RSD	0.930	0.896	**1.0**
CombinedPreferences_Borda_MaxFlow	0.965	0.927	0.829
CombinedPreferences_Borda_RSD	0.939	0.903	0.829
CombinedPreferences_Copeland_MaxFlow	0.961	0.923	0.829
CombinedPreferences_Copeland_RSD	0.937	0.896	0.829

Table 2: All results for course edition 4

	Group AUPCR	Student AUPCR	Average group satisfaction
BEPSysFixed_Borda_ILPPP	0.994	**0.980**	0.849
BEPSysFixed_Borda_MaxFlow	0.998	0.955	**0.936**
BEPSysFixed_Borda_RSD	0.997	0.954	**0.936**
BEPSysFixed_Copeland_ILPPP	0.990	**0.980**	0.870
BEPSysFixed_Copeland_MaxFlow	0.999	0.942	**0.936**
BEPSysFixed_Copeland_RSD	0.999	0.941	**0.936**
CombinedPreferences_Borda_MaxFlow	0.999	0.961	0.811
CombinedPreferences_Borda_RSD	0.999	0.961	0.811
CombinedPreferences_Copeland_MaxFlow	**1.0**	0.940	0.811
CombinedPreferences_Copeland_RSD	**1.0**	0.940	0.811

Table 3: All results for course edition 10

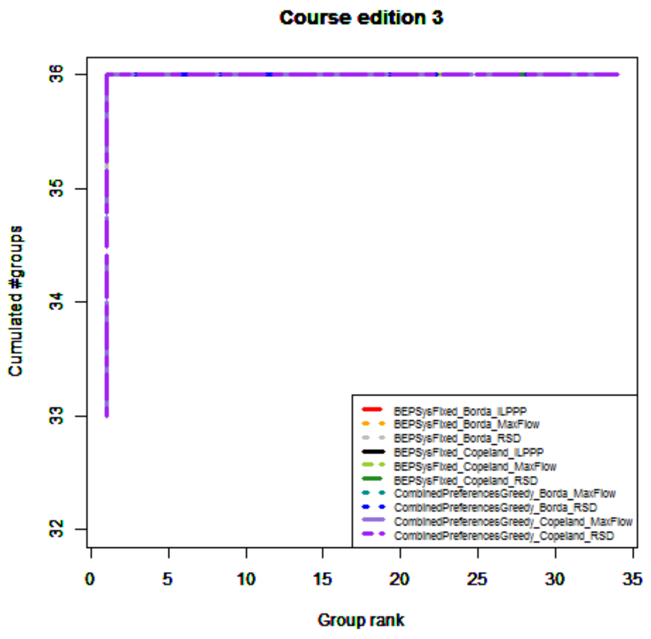


Figure 1: Group project ranks for course edition 3, with the cumulative amount of groups on the y-axis

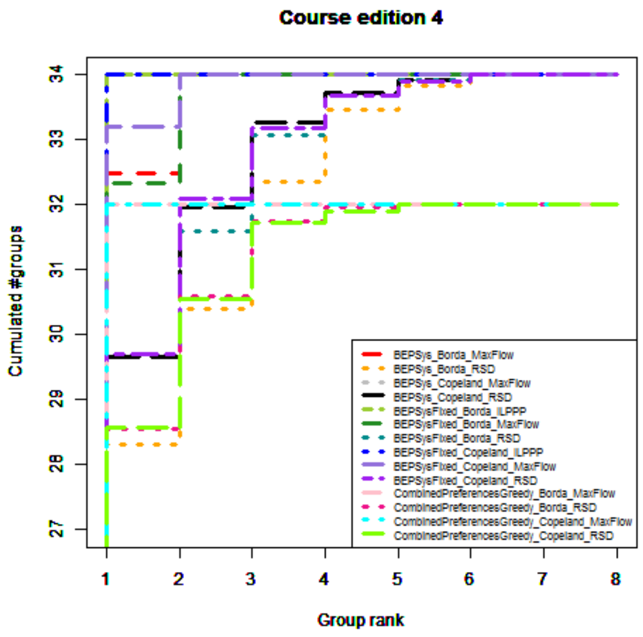


Figure 2: Group project ranks for course edition 4, with the cumulative amount of groups on the y-axis

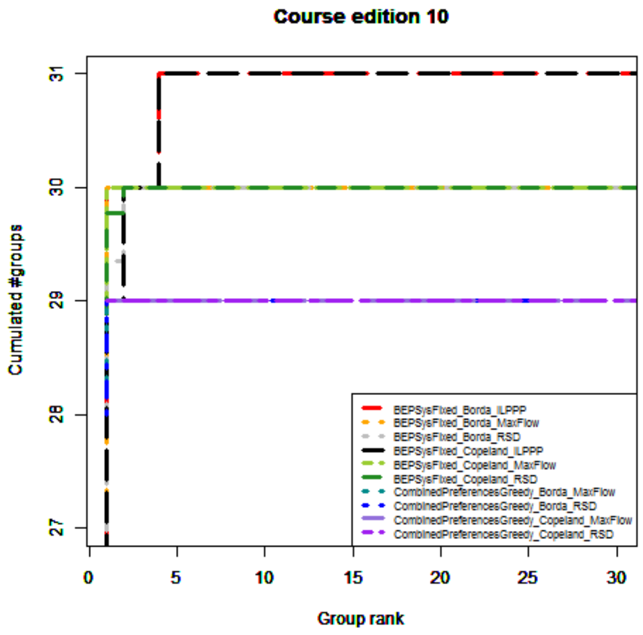


Figure 3: Group project ranks for course edition 10, with the cumulative amount of groups on the y-axis

In figures 1, 2 and 3 above, a visualization of the group rank is shown for all three evaluated course editions. Note that the y-axis does not start from zero in order to better highlight the differences. From the tables, as well as from the visualization can be observed that in almost all cases, the max-flow algorithm performs better with respect to the group project rank.

The results look near optimal on this problem, but it should be kept in mind that this is of course an artificial problem. It is more interesting to see the project rank of individual students, instead of the project rank of groups.

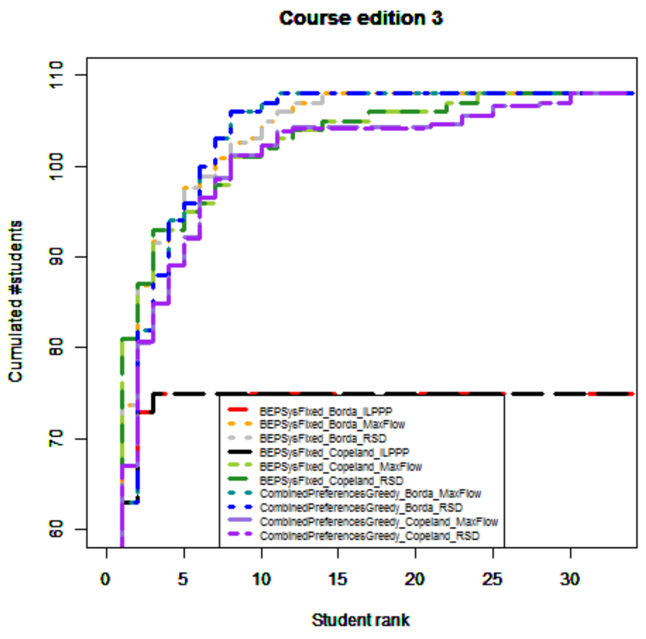


Figure 4: Student project ranks for course edition 3, with the cumulative amount of students on the y-axis

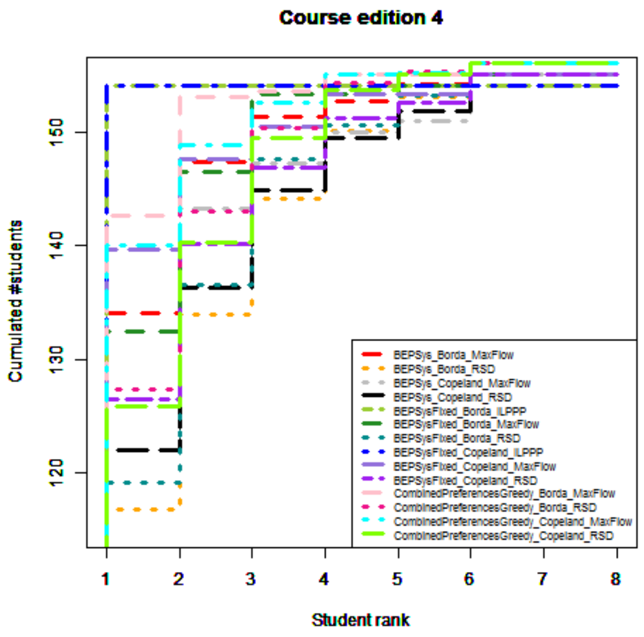


Figure 5: Student project ranks for course edition 4, with the cumulative amount of students on the y-axis

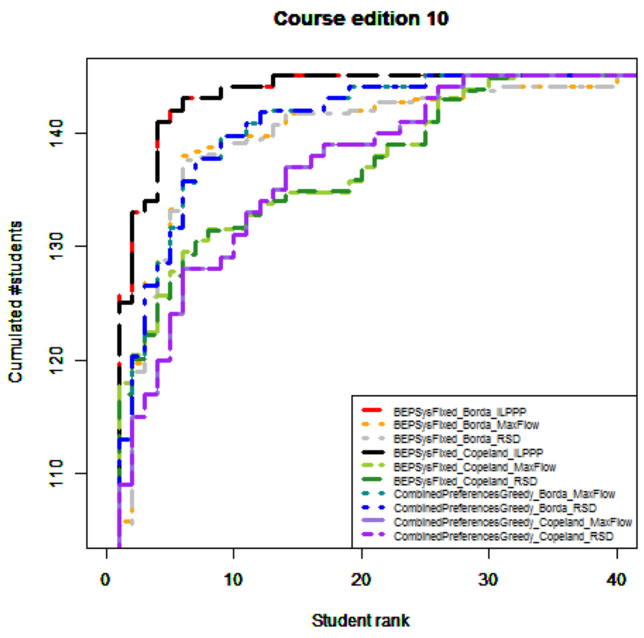


Figure 6: Student project ranks for course edition 10, with the cumulative amount of students on the y-axis

In figures 4, 5 and 6 above, the profile curve concerning the project rank of individual students is shown, comparing the max-flow and RSD algorithm. The group forming is always done via the combined preference grouping and preference aggregation using Borda count.

It is immediately visible that this distribution is much more scattered than the project rank of groups. Again, max-flow seems to slightly outperform RSD but definitely not in all cases.

6 Conclusion

As part of our research goal, we wanted to investigate the performance and possible shortcomings of the current system in place, BEPSys. In conclusion, BEPSys implements the three phase procedure we discussed with group forming, preference aggregation and project assignment. After evaluating the results we conclude that the preference aggregation through Borda count and the project assignment using a max-flow algorithm are performing very well. The BEPSys group forming mechanism, on the other hand, leaves room for improvement. Through case analysis and experimental results we found out that it often fails to produce a grouping, leading to an infinite loop. We produced an improvement to the BEPSys group forming algorithm, which was part of our second research goal.

Next to improving BEPSys, we also implemented a new group forming method, combined preference grouping, based on the premise that the problem is easier if we combine peer- and project preferences. For the preference aggregation method, we implemented Copeland's method as counterpart of Borda count. Also, we implemented randomized serial dictatorship for project assignment. Lastly, we implemented ILPPP as an experiment to better integrate the three phases of the approach.

Our results show that there is no single optimal combination of mechanisms. However, we can conclude that in most cases, when optimizing for student AUPCR, using Borda count for preference aggregation is more preferable than Copeland's method. Also, our introduced techniques with ILPPP and combined preference grouping tend to increase student AUPCR but pay the price in group satisfaction.

In conclusion, we provided a collection of algorithms and concepts to solve the TU Delft student group projects problem. Every combination of provided mechanisms lead to a different solution, optimized towards different goals.

7 Discussion & future work

Although we implemented multiple mechanisms for this problem and obtained results on real world data, it is necessary to say that we did not find an all-encompassing solution. The heart of the issue is the multi-dimensional preferences that is not (yet) well researched or it is hard to give any guarantees.

Another point of criticism towards this work is the data from which we obtained results. It is all historical data from the BEPSys system, but no experiments were run on generated synthetic data. In conclusion, the amount of test data was small, possibly too small to make any general statements about best practises.

As future work, we feel that there is room for improvement when implementing ILPPP with combined preferences.

References

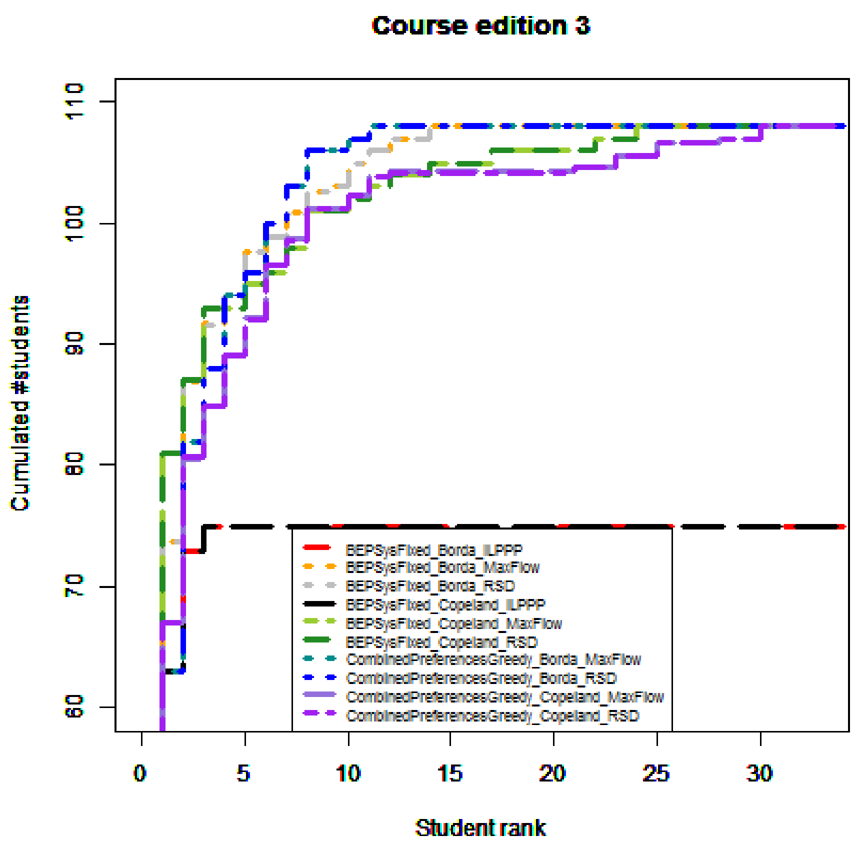
[1] Franz Diebold and Martin Bichler. Matching with indifferences: A comparison of algorithms in the context of course allocation. *European Journal of Operational Research*, 260(1):268 – 282, 2017.

[2] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[3] Donald G. Saari and Vincent R. Merlin. The copeland method. *Economic Theory*, 8(1):51–76, Feb 1996.

[4] Tayfun Snmez and M. Utku nver. Chapter 17 - matching, allocation, and exchange of discrete resources. volume 1 of *Handbook of Social Economics*, pages 781 – 852. North-Holland, 2011.

Appendix A. All results



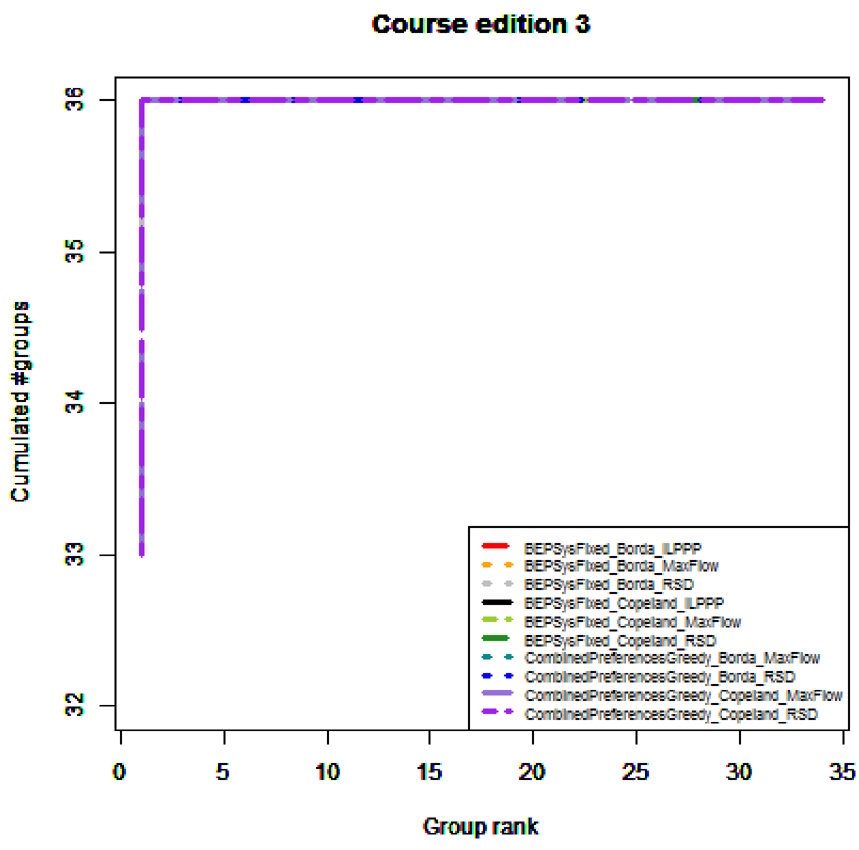
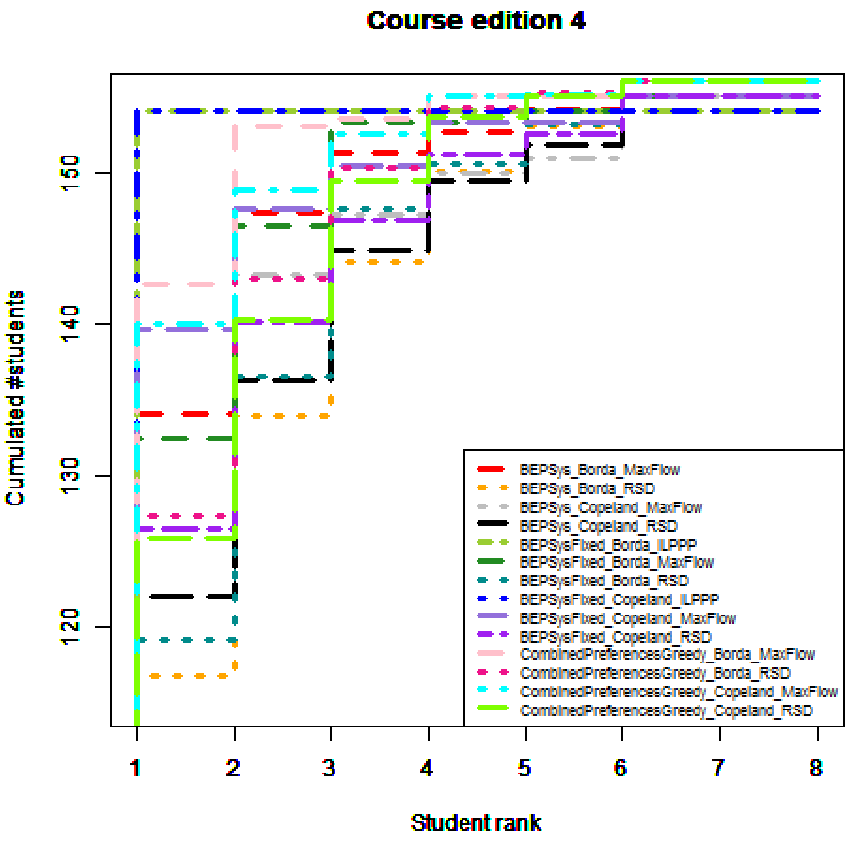


Figure 7: Student and group ranks for course edition 3



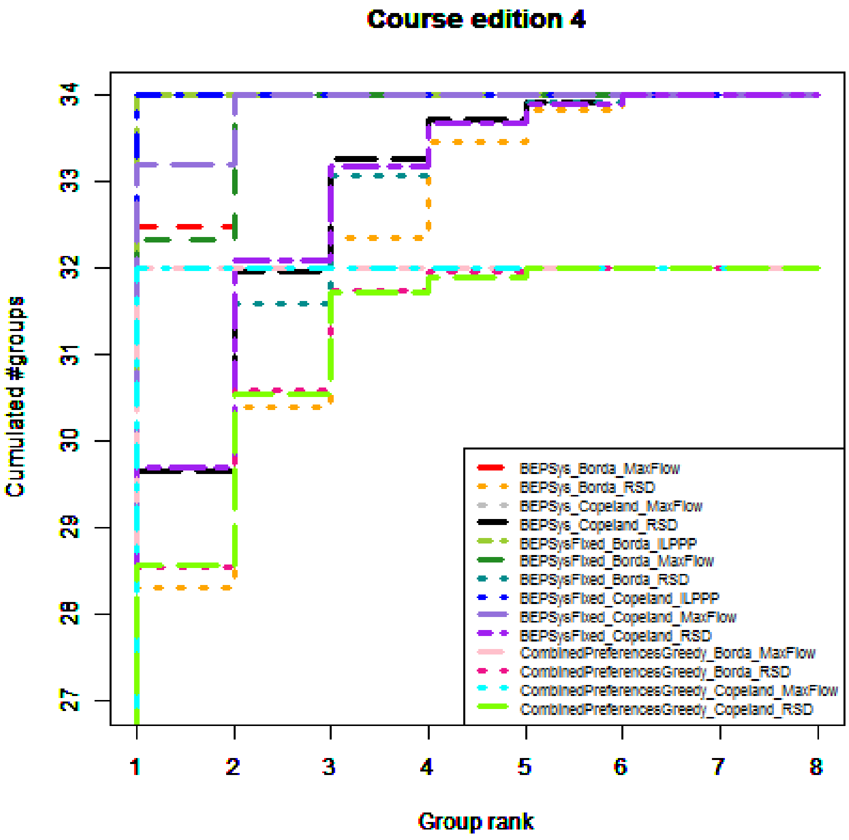
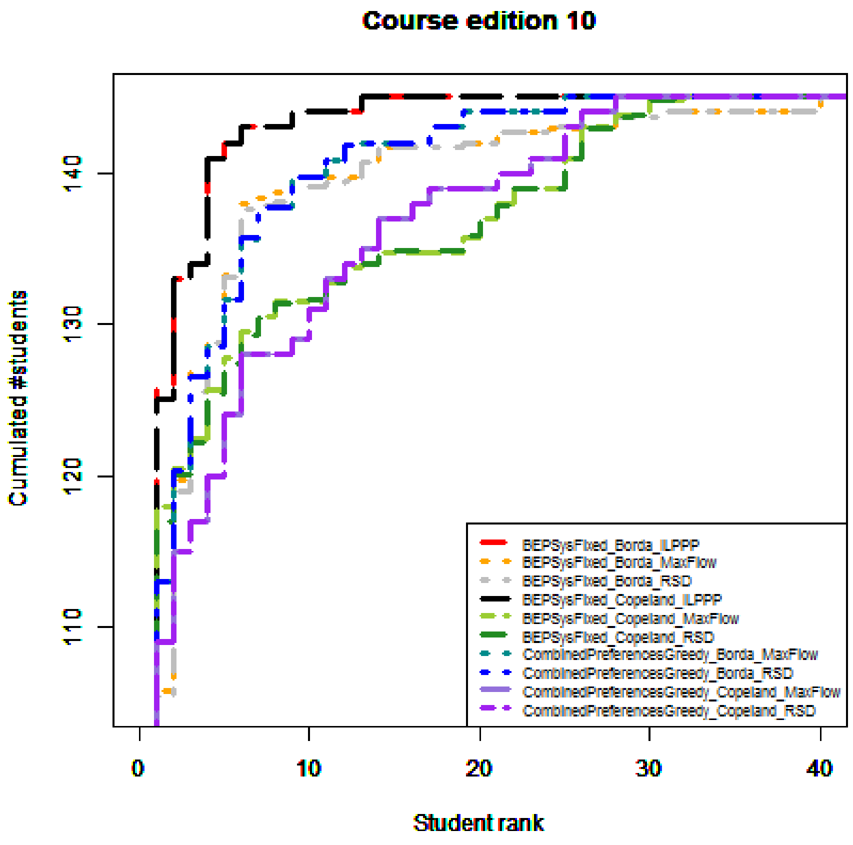


Figure 8: Student and group ranks for course edition 4



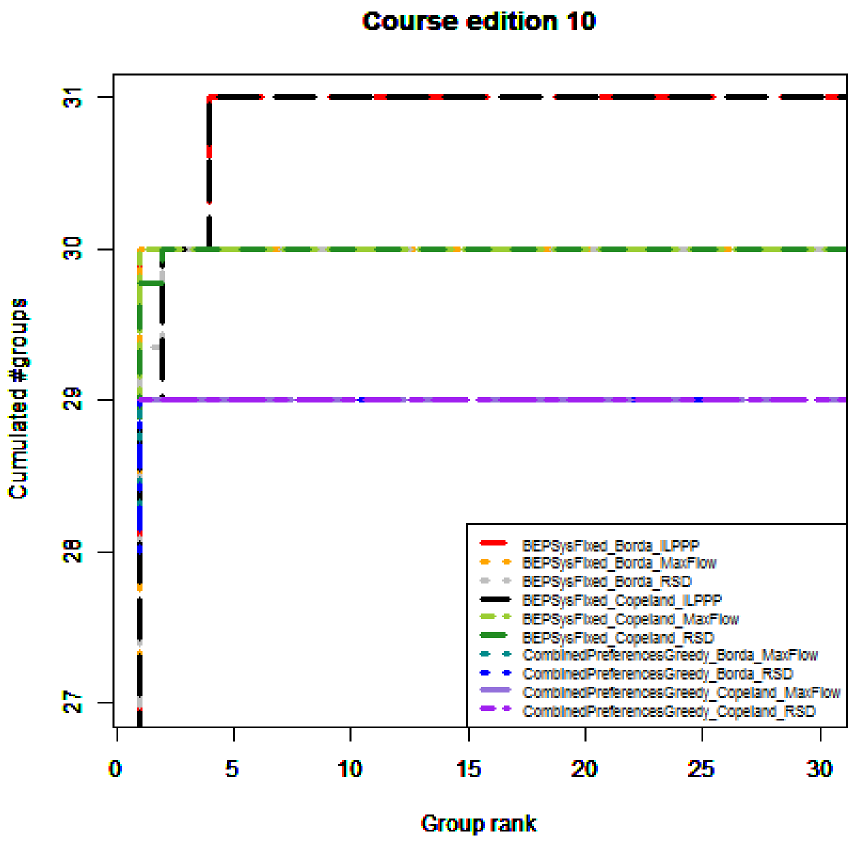


Figure 9: Student and group ranks for course edition 10