Fair Team Recommendations for Multidisciplinary Projects

Lucas Machado
Kostas Stefanidis
lucas.machado@tuni.fi
konstantinos.stefanidis@tuni.fi
Tampere University, Finland

ABSTRACT

The focus of this work is on the problem of team recommendations, in which teams have multidisciplinary requirements and team members' selection is based on the match of their skills and the requirements. When assembling multiple teams there is also a challenge of allocating the best members in a fair way between the teams. We formally define the problem and propose a brute force and a faster heuristic method as solutions to create team recommendations to multidisciplinary projects. Furthermore, to increase the fairness between the recommended teams, the *K-rounds* and *Pairs-rounds* methods are proposed as variations of the heuristic approach. Several different test scenarios are executed to analyze and compare the efficiency and efficacy of these methods, and it is found that the heuristic-based methods are able to provide the same levels of quality with immensely greater performance than the brute force approach.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Recommender Systems, Fairness, Group Formation, Team Recommendations

ACM Reference Format:
Lucas Machado and Kostas Stefanidis. 2019. Fair Team Recommendations for Multidisciplinary Projects. In *WI'19: IEEE/WIC/ACM International Conference on Web Intelligence, October 14–17, 2019, Thessaloniki, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/00000000000

1 INTRODUCTION

Recommender systems offer methods to process extensive amounts of data and derive valuable information from it [22]. They aim to anticipate the *preference* or *rating* that a user would give to some item, based on user activities and personal data. Recommender systems are currently widely used especially in digital media services so relevant items of information can be presented to users or to groups of users. In these recommender systems, it is common to group users based on similarities to provide recommendations of items to the group. In other cases, similar items are grouped in a package for recommendation to a user or group of similar users. Another use of recommender systems can be found on team formation problems.

Recommendations algorithms for team formation aim to assemble teams of individuals based on some specified criteria. For team formation, we need to extract and identify individual characteristics of the individuals, topics from documents and perform analysis and visualization of relation graphs [9]. The individuals are then grouped together based on how near they are from each other in a relations graph (implicit relation identification) or by expert finding [14]. While it is a difficult task to assemble the "best" team, due to the several different subjective factors that could define a team as best, a decision support system such as a recommender system may help on that [9]. Furthermore, most of the papers describe team formation in software development context [30], or based on users common interests and attributes [2, 10].

Thinking about a team as a package of items in which its members have skills that correspond to individual items attributes, a multidisciplinary team is a package of items with diversity in their attributes. Complex tasks often demand multidisciplinary teams, and an increase of a team output could be achieved through selecting members with specific skills to maximize that output. Forming multidisciplinary teams requires aligning people with different skills and backgrounds and should also consider people that are not similar as a possible good choice, while recommender systems are usually based on similarities as an indicator of good alignment [15]. In addition, the concept of diversity is positive in a multidisciplinary team context and may also trigger serendipity [21].

Other challenges may apply to this multidisciplinary team formation problem, for example when a team member is restricted to work for only one project. If several teams are being formed, all the best member candidates could be assigned to the first team, leaving the remaining less suitable candidates for the other projects. Thus, the *fairness* aspect of this team formation should be also taken into account, in a way that good members could be assigned to all teams.

In this work, our inspiration comes from a real world problem in which multidisciplinary teams need to be formed and allocated to work on different projects with requirements for members' skills and some constraints. We pay special attention to fairness when multiple teams need to be formed. That is, we focus on *"how to create team recommendations based on members' skills and projects requirements, while ensuring fairness?"*. Recommender systems could be very helpful in multidisciplinary team formation problems for projects, bringing several benefits. Particularly when there are thousands of candidates and dozens of teams to be formed, the amount of needed human labor can be significantly reduced due to systematic analysis of candidates. In addition, it is a difficult task to

WI'19, October 14–17, 2019, Thessaloniki, Greece

ACM ISBN 00000000000000. . . $00.00
https://doi.org/00000000000

maintain the fairness aspect between teams manually, a problem which that recommender system algorithm can solve.

Based on existing research, Section 2 explores the concepts of recommender systems, fairness and diversity. Section 3 formally defines the problem, while Section 4 proposes different methods for creating team recommendations. Section 5 experimentally evaluates the proposed methods. Finally, Section 6 concludes with a summary of the contributions of this work.

2 RELATED WORK

2.1 Recommender Systems

Being a subclass of information filtering systems, recommender systems are used to analyze large amounts of data and obtain valuable information from it [22]. Their goal is to suggest items or packages of items to users (individual recommendation) or groups of users (group recommendation). After being able to predict what an individual user would like, group recommendations are typically created by combining the individual models of the users in a group.

Group recommendations presents a research challenge in evaluating and improving their effectiveness. Many researchers tried to investigate these aspects (e.g., [1], [25]), and most of the literature focus on group formation and evolution, interfaces that give support to group recommenders and privacy concerns [6].

Fairness is another property that can be considered for recommender systems, especially when suggesting items to groups. Depending of the context of the recommender, fairness could be implemented in different ways. For instance when a package is suggested to a group of users, it would be fair if every user in the group is pleased by an enough amount of items of the package [4, 25]. In the context of multiple teams formation, fairness could be defined as if all the teams receive good members in a balanced way between them. No literature was found regarding fairness in teams recommendations.

Most of the works implement fairness as an improvement over aggregation methods, either for group recommendations or for group recommendations to individuals (e.g. [7], [17], etc.). Due to the possible different preferences within users in a group, achieving fairness could improve overall satisfaction but also reduce it for a few members. Commonly it is implemented with a penalty factor to the amount of variation between the predicted ratings.

2.2 Group formation

The employment of soft-computing and smart methods for selecting personnel has been extensively researched. Several studies support selection of individuals with the use of computer intelligence techniques and information systems (e.g. [26], [27]). Mohanty et al. [19] presents an important review on this field. The attempts to solve personnel selection problems with technology are frequent (e.g., [3]). For example for selecting teams of software engineers many techniques are used, such as fuzzy logic (e.g., [26]), semantics (e.g., [28]) and rough sets (e.g., [12]). Many authors also address the issue of combining these intelligent systems approaches aiming to improve the selection of members' performance (e.g., [16], [20]).

The use of those personnel selection algorithms comes naturally to form teams. When used with recommender systems, team formation can help in decision taking problems by indicating the best combination of members in regard to specific criteria. For instance, Colomo-Palacios et al. [8] describe how to use rough sets and fuzzy logic to form and recommend Scrum teams, based on team members' roles. This described system is able to help project managers in assembling the best team for Scrum projects, based on available staff and each project required competences.

Constraints can also be applied to recommender systems, as described by Stefanidis and Pitoura [23]. In their paper, a problem of team formation for recommending an item when the team members are affected by constraints is presented. A greedy algorithm is depicted in which the team is built by incrementally selecting users by how much score they add to the team and by how well they satisfy the set of imposed constraints. The novel aspect is considering group consensus not only towards an item recommended for the team, but also regarding other group members.

As with the literature described above, most of the research of team formation is focused on the software development context. In addition, Minto and Murphy [18] suggest an approach to form and recommend emergent teams based on how software artifacts are changed by developers. Yilmaz et al. [30] depict a team recommender based on personality of team members, in which a machine-based classified predicts the performance of the possible teams. Lappas et al. [13] also propose a team recommender in which individuals are grouped by skill requirements, but also uses a communication cost indicator to measure effectiveness. Therefore, despite following the same general procedure, the problem of recommending teams has a tendency to differ in how to aggregate the members for a team, which depends largely on the application domain and its particularities. Moreover, there is still space for more research on team recommender systems for more general contexts. Those other contexts may have different constraints and methods to assess fairness, efficacy and to form the team itself than recommender systems in the software development context.

3 PROBLEM DEFINITION

3.1 Motivating example

Assume that there are several projects that aim to create products and satisfy needs. Each one of them has different needs of skilled people based on their requirements, restrictions, context and goals. For example, a project on developing a new website for a company would require individuals with skills of front-end development, design of interfaces, and user experience. However, a project aimed at creating a device for measuring heart rate would need individuals with expertise in health sciences, engineering and ergonomics.

The individuals that could work on a project possess different sets of skills. The ability or expertise to do something well is defined as a *skill*. A *project* is a collaborative effort to reach a goal, which is carefully designed and planned [24] and that requires a team of people with specific skills for that.

This work investigates a team formation problem in the context of a platform in which several different projects are available to receive applications from interested individuals (applicants) to work on them. For all projects, a team should be formed by matching the project requirements with applicants' skills. Furthermore, each project has a determined number of team members required (for example, 6 members). Figure 1 shows the result of the recommender

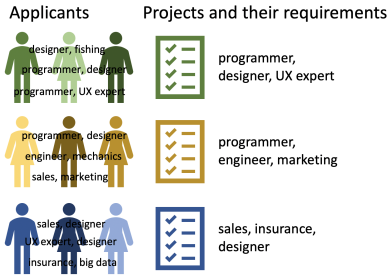

Figure 1: Example of suggested teams of three applicants each, and their skills, for a set of three projects and their required skills.

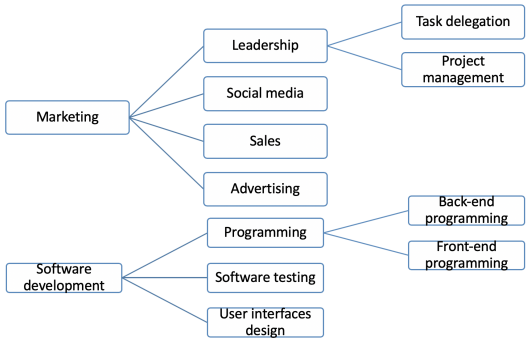

Figure 2: Example of a set of skills in a hierarchy

system, in which teams of applicants are suggested for the projects based on their skills. In this hypothetical situation, all the projects have team members that satisfy their requirements.

The teams should be formed in a way that maximizes matches between project requirements and applicants' skills. A perfectly maximized but unrealistic team formation would be when for every applicant in the team, the applicant possesses all of the project required skills. However, a given applicant cannot belong to more than one team, which poses a restriction since forming a team limits the available choices of applicants for other teams. Therefore, when forming all the teams, some fairness is required in such a way that all teams are similarly good and choices are not made purely on finding the best possible applicants for a project, leaving other projects with the remaining less suitable applicants.

3.2 Model

Skills, which might be attributes of applicants or project requirements, are represented by textual tags and relate to each other in a graph relationship. Figure 2 shows an example of a set of skills and their relations. In this example, skills are represented by nodes which are connected by edges.

Skills are not only attributes of applicants, but also requirements for the projects. Let P be a set of projects, in which each project p, $p \in P$, is described by a set of required skills $p = \{r_1, \ldots, r_n\}$ that the team members of the project must possess, in which r_i is a textual tag representing a skill. It can be assumed that all projects have the same amount of required skills.

Example 3.1 (Projects p_1 and p_2 and their sets of required skills).
p_1 = {programming, sales, user interfaces, design}
p_2 = {programming, user experience, marketing, advertising}

To determine if an applicant a has the skill r that is needed in the project p, or if the applicant has any skills related to r within its skill set, the function $scoreAR(a, r)$ is used. The function returns a score of how well an applicant has skills that relate to a specific required skill of a project. The *similarity* function returns a score in the interval $[0, 1]$ depending on how related the skills are.

$$scoreAR(a, r) = \sum_{\forall s_i \in a} similarity(s_i, r) \quad (1)$$

Knowing how well an applicant's skills suit one project requirement, it is possible to calculate how well an applicant fits in a project in consideration with all of the project's required skills. The function $scoreAP(a, p)$ is used to calculate the matches between the set of skills of an applicant a and all the required skills of a project p, based on the function $scoreAR(a, r)$.

$$scoreAP(a, p) = \sum_{\forall r_i \in p} scoreAR(a, r_i) \quad (2)$$

Furthermore, with the information of how well applicants could fit within a project, the function $scoreTP(t, p)$ denotes how well a team of k applicants t, $t \subset A$, matches with all the required skills of a project p. By using this function it is possible to compare how well different team formations fit to a project, according to Definition 3.2 below. It is assumed that all teams are formed with the same amount of applicants k.

$$scoreTP(t, p) = \sum_{\forall a_i \in t} scoreAP(a_i, p) \quad (3)$$

Definition 3.2. Given a project $p = \{r_1, \ldots, r_n\}$, and a set of applicants $A = \{a_1, \ldots, a_m\}$, where each applicant a_i is associated with a set of skills $\{s_{i_1}, \ldots, s_{i_x}\}$, the best team of k applicants for the project p is the team T^* for which:

$$T^* = argmax_{|T|=k} scoreTP(T, p),$$

such that, $\forall r_j \in p$, $\exists a_i \in T$, with $s_{i_y} = r_j$; and there are at least k applicants in the set A.

Therefore, to form q teams of k members, the set A must contain at least $q \times k$ applicants.

The model presented above differs from the methods in research literature in the sense that teams are not formed based on similarities between its members as in content-based methods, neither on past ratings as in collaborative filtering approaches. The concept of ratings itself is not used in a traditional way, but replaced by the relation between required skills of projects and applicants' skills, which for this problem would be better called *scores*. Since the relations between applicants' skills — items' attributes — are not taken into consideration, content-based filtering is not suitable. Furthermore, recommended teams are also not calculated based on historic data of past formed teams. Projects are often unique and rarely the assumption that there are two or more projects with the same requirements can be made. Hence, collaborative filtering would also be unfit for this problem, since the sparsity problem would be taken to an extreme in which the subset of rated items for a user would consist of at most one item.

Nonetheless, the model seems to fit better within knowledge-based approaches. It could be related to constraint-based knowledge-based systems, as the projects possess requirements for the desired applicants' skills. Furthermore, knowledge-based methods have as their strengths the ability to work well with sparsity, complex and specific problems, which is the case presented by this work.

Moreover, the concept of group recommendations to individuals is used. The formation of a team involves calculating the score of applicants for a project, then combining them into the team, based on the requirements (criteria) set by the projects.

3.3 Fairness-aware Team Formation

It is not sufficient to find the best team for a project using the Definition 3.2, since the assignment of applicants to a specific team makes them unavailable to other teams. Therefore, the property of fairness needs to be applied when suggesting multiple teams to multiple projects, so that there is a balance between the teams, as specified by Definition 3.3 below.

Definition 3.3. **Fairness to teams:** Let $\mathcal{T}$ be a set of n teams $(T_1, \ldots, T_n)$, assigned to a set $\mathcal{P}$ of projects $(p_1, \ldots, p_n)$. Given a set $\mathcal{TS}$ including all pairs of teams $(T_i, T_j) \in \mathcal{T}$, to ensure fairness in group formation, minimize:

$$\sum_{(T_i, T_j) \in \mathcal{T}} |scoreTP(T_i, p_i) - scoreTP(T_j, p_j)|$$

The implication of applying Definition 3.3 is that the best possible team is not always going to be chosen for some projects. However, by choosing teams with slightly lower scores (*scoreTP*) for some projects, it is possible to choose teams with greater scores for others, thus minimizing the differences and increasing the fairness between them. Furthermore, the implementation of fairness in this work is novel relating to the research literature, as most of the approaches are implemented by reducing the variation between predicted ratings, i.e. by increasing similarity between items. On the other way, this work proposes to achieve fairness through the way in which teams are formed, considering the presented restrictions of the context.

4 METHODS

4.1 Brute force algorithm

Assume that $combinations(k, L)$ is a function that calculates the binomial coefficient (generates a list of all possible combinations) of k elements from the set of elements L. In our context k is the amount of applicants in a team, and L is the set of all applicants. Moreover, the function $scoreAP(a, p)$ calculates how well a given applicant a is suited to a given project p, based on the model in Section 3. Algorithm 1 below uses the $combinations(k, L)$ and $scoreAP(a, p)$ functions to implement a brute force method to generate the best teams recommendations:

For every project p in the set of projects P, all the possible team combinations T of k members are generated from the set of available applicants A. Then for every possible team t in T, its score relating to the project p is calculated with the function $scoreTP(t, p)$. The team with the maximum score is chosen as the best team for that project and its members are removed from the set of available applicants A. This team formation process is repeated for all projects.

The asymptotic computational complexity of Algorithm 1 is factorial as denoted by $O(k \times \binom{A}{k} \times P)$, or $O(n!)$.

Algorithm 1: Brute force recommender method

```
Input: A set of applicants A, a set of projects P, and the team size k
bestTeams = [];
foreach p_i in P do
    possibleTeams = combinations(k, A);
    teamsScores = [];
    foreach t_j in possibleTeams do
        scoreTP = 0;
        foreach a_n in t_j do
            scoreTP = scoreTP + scoreAP(a_n, p_i);
        end
        put {t_j : scoreTP} in teamsScores;
    end
    bestTeamForProject = max(teamsScores :: scoreTP);
    put {p_i : bestTeamForProject} in bestTeams;
    remove members in bestTeamForProject from A;
end
Result: Set of bestTeams
```

4.2 Heuristic algorithm

It is noticeable, however, that the brute force approach defined by Algorithm 1 is very computationally expensive with its factorial asymptotic complexity, due to the calculation of all possible team combinations. Therefore, a heuristic which could be applied to minimize the computations while keeping the recommendation efficacy is proposed.

Instead of generating all possible team combinations, Algorithm 2 first calculates $scoreAP(a, p)$ between every applicant a in the set A and every project p in the set of projects P. These values are stored as a set in the *projectApplicantsScores* variable. Then the applicant a who had the best calculated (maximum) *scoreAP* for a given project p is chosen as a member of that project team and is removed from the set of available applicants A. This previous step is repeated k times until the project p team has all its k members chosen. This team formation process is then repeated for all other projects of the set P.

Furthermore, Algorithm 2 could be optimized to improve the fairness according to Definition 3.3. That optimization is specified in the novel variants Algorithm 3 and Algorithm 4 below, which implement a *k-rounds choosing* method to generate more fair teams recommendations.

Instead of choosing all the k best applicants as members of a project team and then repeating the process for other project teams, Algorithm 3 chooses only one applicant a who had the best calculated *scoreAP* for every project p as a member of that project team, and also removes it from the set of available applicants A. This procedure happens in k rounds to add the *k-nth-member* until all the teams have k members.

Similarly to Algorithm 3, Algorithm 4 implements a variation *pairs-rounds choosing* method to form teams. Based on the calculations of $scoreAP(a, p)$ between every applicant a in the set A and every project p in the set of projects P, there are $k/2$ rounds in which the pair of applicants a_1 and a_2 who had the best values of

Algorithm 2: Heuristic recommender method

Input: A set of applicants A, a set of projects P, and the team size k

$bestTeams$ = [];
$projectApplicantsScores$ = [];
foreach p_i *in* P **do**
  foreach a_n *in* A **do**
    put $(p_i, a_n, scoreAP(a_n, p_i))$ in $projectApplicantsScores$;
  end
end
foreach p_i *in* P **do**
  for $m = 1$ **to** k **do**
    $bestApplicant$ = $max(projectApplicantsScores[p_i] :: scoreAP)$;
    put $bestApplicant$ in $bestTeams[p_i]$;
    remove $bestApplicant$ from $projectApplicantsScores$;
  end
end

Result: Set of $bestTeams$

Algorithm 3: *K-rounds choosing* recommender method

Input: A set of applicants A, a set of projects P, and the team size k

$bestTeams$ = [];
$projectApplicantsScores$ = [];
foreach p_i *in* P **do**
  foreach a_n *in* A **do**
    put $(p_i, a_n, scoreAP(a_n, p_i))$ in $projectApplicantsScores$;
  end
end
for $m = 1$ **to** k **do**
  foreach p_i *in* P **do**
    $bestApplicant$ = $max(projectApplicantsScores[p_i] :: scoreAP)$;
    put $bestApplicant$ in $bestTeams[p_i]$;
    remove $bestApplicant$ from $projectApplicantsScores$;
  end
end

Result: Set of $bestTeams$

scoreAP for p are assigned as team members and removed from the set of available applicants A. Again, this process is repeated until all the projects have teams of k members. If k is an odd number, then during the last round only one team member will be assigned.

It is expected that by selecting team members one by one or in pairs between the projects, the fairness between teams is increased.

In contrast to Algorithm 1, the asymptotic computational complexities of Algorithms 2, 3 and 4 are linear as defined by $O(A \times P + k \times P)$, or $O(n)$.

5 EXPERIMENTAL EVALUATION

5.1 Dataset

A preprocessed dataset derived from the DBLP dataset is used for testing. It was created by Wang et al. [29] and consists of a CSV file of 7428 lines. Each line corresponds to a researcher from DBLP and contains the person's name and a varying number of skill tags related to that person. Each researcher has at least one skill and there are 4480 unique skills among all people, with some skills appearing more than once for the same person.

Due to the nature of DBLP, the skills associated with the researchers correspond to keywords used in those researchers publications. Truthfully, it may not represent the same definition of skills used in this work. However, since the skills derived from the DBLP dataset represent an area of knowledge or expertise in which a person published research, it could be considered as a sufficient approximation. In addition, a best suited dataset was not found publicly.

Algorithm 4: *Pairs-rounds* recommender method

Input: A set of applicants A, a set of projects P, and the team size k

$bestTeams$ = [];
$projectApplicantsScores$ = [];
foreach p_i *in* P **do**
  foreach a_n *in* A **do**
    put $(p_i, a_n, scoreAP(a_n, p_i))$ in $projectApplicantsScores$;
  end
end
for $m = 1$ **to** k **do**
  foreach p_i *in* P **do**
    $bestApplicant$ = $max(projectApplicantsScores[p_i] :: scoreAP)$;
    put $bestApplicant$ in $bestTeams[p_i]$;
    remove $bestApplicant$ from $projectApplicantsScores$;
    $secondBestApplicant$ = $max(projectApplicantsScores[p_i] :: scoreAP)$;
    put $secondBestApplicant$ in $bestTeams[p_i]$;
    remove $secondBestApplicant$ from $projectApplicantsScores$;
  end
end

Result: Set of $bestTeams$

To assemble the hierarchy relationship between the skills, the Wordnet database was used. Wordnet is a large lexical database in English language, in which words are grouped by their cognitive synonyms (synsets) [11]. There are about 117 000 synsets in Wordnet and each one of them convey a different concept. They are connected by lexical and conceptual-semantic relationships, and with those connections it is possible to estimate how similar in meaning one word is to another. Therefore, use of Wordnet suits for computing similarity between the skills derived from the preprocessed DBLP dataset.

5.2 Measurements

In our experiments, we compare how well the proposed heuristics perform timewise, since the presented brute force approach has a factorial asymptotic complexity ($O(n!)$) while the heuristic methods have linear ($O(n)$). For verifiying that, the execution time of these algorithms is measured.

Since the *scoreTP* value is an indicator of how well a team fits into a project requirements, the analysis of the success of the recommendations is focused on it. The sum of all the *scoreTP* values in a set of recommended teams indicate how successful the recommendation method is, relative to its parameters (amount of projects, amount of required skills by project and amount of members in each team). Therefore, this measurement is taken into account as it conveys s better quality and quantity of matches between applicants' skills and project requirements.

Furthermore, based on the *scoreTP* values, a *fairness-deviation* indicator is proposed to measure the fairness digression between recommended teams. Assuming that an absolute fair set of teams would be a set in which all of the teams have the same *scoreTP*,

the fairness-deviation indicates how much in average the teams deviated from this absolute fair situation. The fairness-deviation between a set of recommended teams $\mathcal{T}$ is defined by

$$\text{fairness-deviation}(\mathcal{T}) = \frac{\sum_{\forall t_i \in \mathcal{T}} \mid t_i - \text{mean}(\mathcal{T}) \mid}{\text{len}(\mathcal{T})}.$$

where mean($\mathcal{T}$) is the arithmetic mean of the *scoreTP* values of the teams in the set $\mathcal{T}$, and len($\mathcal{T}$) is the count (length) of teams in the set.

5.3 Methods

A Python script was crafted to implement the experiments and output the results. The complete code is available in a GitHub repository[1].

After loading the preprocessed DBLP dataset, by joining all the skills of all the 7428 researchers — from now on referred as applicants — and eliminating duplicates, a set of 4480 unique skills was formed. Since computing the similarity between skills could be a resource intensive procedure due to the task of finding nodes and paths within the skills' graph, a similarity matrix between all the skills in the dataset is precalculated and saved to speed up the overall algorithm.

NLTK (Natural Language Toolkit) — a software library to work with human language data — provides an API[2] to access Wordnet dataset and exposes a function *path_similarity* that was used as the similarity function for synsets (skills). It returns a value in the interval [0, 1], from no link to identity between the synsets. Due to the fact that a word may convey different meanings, and that it is not feasible to determine individually the correct synset for every skill in the dataset, it is decided that the first found synset is used. If no synsets are found for a skill in Wordnet, its similarity to all other skills is considered to be zero, except the similarity to itself that is then considered one.

Several test scenarios were created to analyze different aspects of the algorithms. A test scenario in this context is a team formation task in which all the algorithms are executed to create team recommendations to a common randomly generated set of projects. Test scenarios were organized into 4 *test groups*, which may contain one or more of them. Each test group has a different goal regarding analyzing algorithms efficiency or effectiveness. Algorithm 1 (brute force) was executed only in test group 4.

Test scenarios receive input, execute algorithms and report measurements. Each test scenario receives as input the amount of projects p and the set of skills that could be used as project requirements, then a set of p projects is created with 20 randomly sampled skills from the given set. This set of created projects together with the parameter k of how many members each team should have are subsequently used as input to the algorithms. The results returned by the algorithms are measured and recorded. All the test scenarios in test groups 1, 2 and 3 use the full set of 7428 applicants as input to the algorithms, while the test scenarios in test group 4 use a sampled subset of 100 applicants from the full set.

The objective of test groups 1 and 2 is to analyze how much time the algorithms need to create the team recommendations. For test group 1 all the parameters were fixed and only the amount of projects p changed. A total of 10 test scenarios were tested with values of p ranging between 3 and 30, in intervals of 3 ($\{3, 6, 9, \ldots, 30\}$). Similarly for test group 2, all the parameters were also fixed except for the amount k of team members. The values of k tested were again the range between 3 and 30, in intervals of 3 ($\{3, 6, 9, \ldots, 30\}$), for a total of 10 test scenarios.

Test group 3 tests aimed at comparing the fairness-related results between the algorithms and had randomly generated input parameters inside some constraints. From the range of numbers between 5 and 35 a sample of 10 numbers was selected to be used as p values (amount of projects). Then from the range between 3 and 12 a sample of 5 numbers was used as k values (amount of members in each team). These range values were chosen trying to represent the extreme situations in a real scenario, as Agile teams, for example, usually have less than 10 members [5]. All the possible combinations between the 10 sampled p and 5 sampled k values are used to generate a total of 50 test scenarios. Those 50 scenarios are executed receiving as input only the 200 most frequent skills as possible project requirements. Likewise, another 50 test scenarios are created and executed with another samples of p and k values following the same constraints, however using the 200 less frequent out of the 2000 most frequent skills to create project requirements. The behaviour of the algorithms during the extreme situations of overfitting and underfitting of data can be analyzed with this variation between the most and less frequent skills.

Test group 4 has only one test scenario and uses a reduced amount of data with small values for parameters, so it allows to conceptually test the brute force algorithm and compare it with the heuristic methods. A random sample of 100 applicants is used, and the task involves only 4 projects ($p = 4$) with 4 team members each ($k = 4$).

5.4 Efficiency

Calculating the best teams with the brute force method (Algorithm 1) is not feasible due to its factorial asymptotic complexity. In practice, given that in the dataset there are 7428 applicants and the total possible teams is given by $\binom{N}{k}$, for example for a team of only 6 members there are $\binom{7428}{6}$, or $2.3282073138 \times 10^{20}$ possible teams. With some tests in a common 3,1 GHz Intel (I5-7267U) processor, on average 10^4 teams are calculated per second, which gives that for calculating all possible choices for a single team to obtain the best one would take approximately 737 million years.

Table 1 below shows the results of test group 4 test scenario with all the algorithms, including the brute force method. Since the test scenario used a sampled subset of 100 applicants from the original full set, and only had to form 4 teams of 4 members each, the amount of calculations was reasonable enough to be executed. The teams formed by brute force and heuristic approach were identical since the core idea is the same: To choose the best possible team for a project, either by comparing with all other teams possibilities or by choosing the best k applicants. However, the time taken to execute the algorithms is very different, with the brute force method needing over 60,000 times the amount of time that the heuristic method needed. For those reasons, the brute force algorithm was not executed in other test scenarios.

[1] https://github.com/machadolucas/Team-Recommender
[2] http://www.nltk.org/howto/wordnet.html

Table 1: Results of test scenario from test group 4

Method	Time (s)	scoreTP sum	f-deviation
Brute force	891.737	156.643	16.270
Heuristic	0.014	156.643	16.270
K-rounds	0.012	160.642	5.219
Pairs-rounds	0.012	159.507	7.969

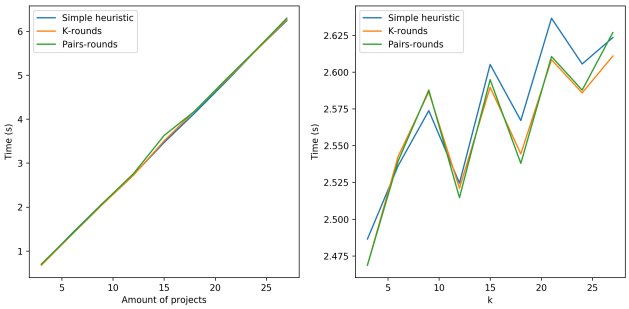

Figure 3: Time consumed by algorithms

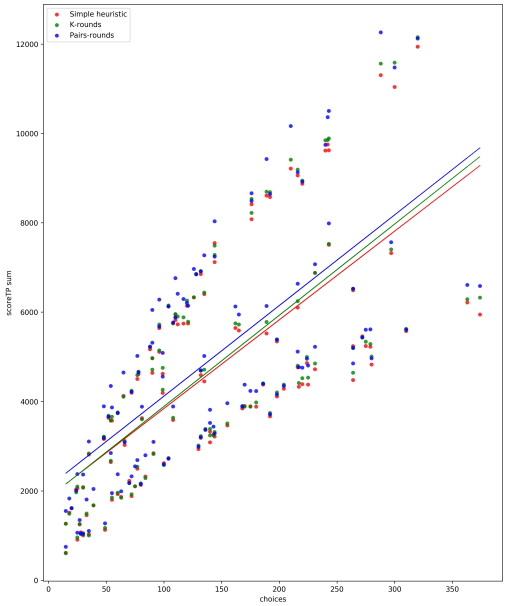

Figure 4: Sum of scoreTP values over amount of choices made, for different algorithms

The time spent to execute the Algorithms 2 (simple heuristic method), 3 (*k-rounds* method) and 4 (*pairs-rounds* method) with test groups 1 and 2 is shown in Figure 3. The graph on the left, created with the test scenarios of group 1, shows very clearly the linear nature of the algorithms, with almost perfect convergence as the amount of projects increase for all the heuristic methods. That corroborates the calculated asymptotic complexity presented in Section 4.

Nevertheless, on the right side of the Figure 4 the increase of k value results in differently shaped lines with small variations. The scale of these variations should be taken into account though, since they are very small and a trend line would still show linearity. This graph was created with test scenarios from test group 2.

Therefore, the time spent to generate the recommendations prove that the heuristic algorithms are efficient and suitable for use in real-world scenarios, particularly considering that other optimizations in data structure and architecture, as well as in the code could be implemented to further reduce the processing time.

5.5 Effectiveness

User studies, A/B tests (online evaluation), and offline evaluations, as suggested by literature, are not appropriated methods to evaluate this work's recommended teams efficacy. Since these traditional efficacy evaluation methods rely on the existence of previous datasets as reference for comparison, or in users judgment or behaviour towards the recommendations, and the problem explored by this work is quite unique, it is not possible to use them for evaluation. For that reason, the recommendations are evaluated by how well the team members of the team recommendations adhere to the required skills of the projects, represented by the sum of *scoreTP* values. In addition, they are also evaluated by how fair the teams are in the context of a set of recommended teams $\mathcal{T}$. The fairness-deviation indicator is used for that.

Figure 4 shows the sum of all the *scoreTP* values over the amount of choices made in a set of recommended teams $\mathcal{T}$. The amount of choices made refer to the amount of team members in each team multiplied by the amount of project teams ($k \times \text{len}(\mathcal{T})$). By applying linear regression to this data (as indicated by the lines in the figure), it is possible to notice that the *Pairs-rounds* method achieves slightly better results in overall than the other methods, while the *K-rounds* method seems to improve when the amount of choices made increase. However, the results of the three algorithms are very similar and in practice their differences could be considered negligible. This figure was created with the results of test groups 1, 2 and 3.

Due to the two variations of skill sets used to generate project requirements in test group 3, it is also possible to notice two linear areas of concentrated points in Figure 4. More frequent skills in project requirements have a bigger chance to find more similarities with applicants' skills, thus increasing the overall value of *scoreTP*.

Figure 5 shows the fairness-deviation values for all the test executions, over the *amount of choices made*. With linear regression analysis on this data (as indicated by the lines in the figure), it is observed that the *K-rounds* and *Pairs-rounds* methods produce significantly more fair results than the simple heuristic method without fairness optimization. Figure 5 was also created from the results of test groups 1, 2 and 3.

Based on this data, *k-rounds* and *pairs-rounds* methods clearly show an large improvement in fairness measurements when comparing to the original simple heuristic method.

As can be noticed in the results of Table 1, the brute force method could be completely dismissed in a practical implementation, since the team recommendations created by it are exactly the same as the ones created by the simple heuristic, but at a far more expensive

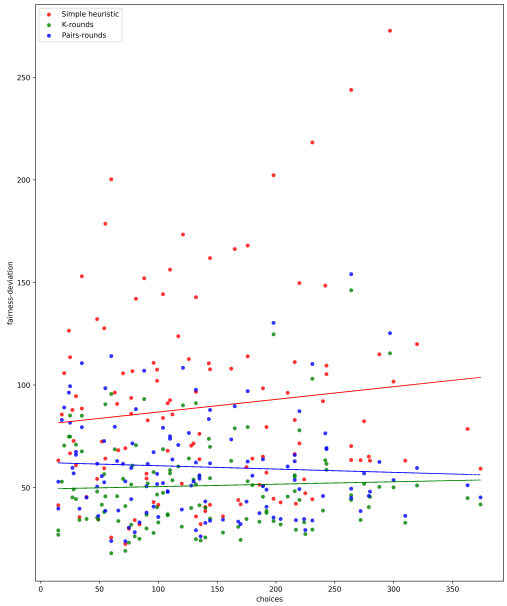

Figure 5: Fairness-deviation over amount of choices made, for different algorithms

computational cost. Furthermore, the brute force and heuristic methods not only produced less fair results than the k-rounds and pairs-rounds methods but also had a lower sum of the *scoreTP* values. This occurred because first a full team was formed and its members were made unavailable for other teams. Such situation can happen in brute force and simple heuristic methods, and for that reason, the order in each teams are formed is important in these approaches and the sum of *scoreTP* values may not be maximized. That further supports the claim that k-rounds and pairs-rounds methods have a better efficacy in general when compared to simple heuristic method, and also explains their slightly better results in Figure 4.

6 CONCLUSIONS

The focus of this work is on the problem of fair team recommendations. We formally define the problem and propose algorithms as solutions to create team recommendations to multidisciplinary projects. Both efficiency and efficacy evaluation show that the proposed heuristic-based methods are able to create team recommendations for multidisciplinary projects in a very successful way. The recommendations have the same level of quality as the brute force approach, but are calculated tens of thousands times faster. The suggested variations of the heuristic also prove to create results with more fairness.

REFERENCES

[1] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawlat, Gautam Das, and Cong Yu. 2009. Group recommendation. *Proceedings of the VLDB Endowment* 2, 1 (8 2009), 754–765.

[2] Gaganmeet Kaur Awal and K. K. Bharadwaj. 2014. Team formation in social networks based on collective intelligence - an evolutionary approach. *Applied Intelligence* 41, 2 (9 2014), 627–648.

[3] Ana Barcus and Gilberto Montibeller. 2008. Supporting the allocation of software development work in distributed teams with multi-criteria decision analysis. *Omega* 36, 3 (6 2008), 464–475.

[4] Robin Burke. 2017. Multisided Fairness for Recommendation. *The Computing Research Repository (CoRR)* abs/1707.0 (2017). http://arxiv.org/abs/1707.00093

[5] Denise Canty. 2015. *Agile for Project Managers.* Auerbach Publications. 135 pages. https://doi.org/10.1201/b18052

[6] Jinpeng Chen, Yu Liu, and Deyi Li. 2016. Dynamic Group Recommendation with Modified Collaborative Filtering and Temporal Factor. *The International Arab Journal of Information Technology* (2016).

[7] Ingrid A. Christensen and Silvia Schiaffino. 2011. Entertainment recommender systems for group of users. *Expert Systems with Applications* (5 2011).

[8] Ricardo Colomo-Palacios, Israel González-Carrasco, JosÃĺ Luis López-Cuadrado, and ÃĄngel García-Crespo. 2012. ReSySTER: A hybrid recommender system for Scrum team roles based on fuzzy and rough sets. *International Journal of Applied Mathematics and Computer Science* 22, 4 (1 2012), 801–816.

[9] Anwitaman Datta, Jackson Tan Teck Yong, and Stefano Braghin. 2014. The zen of multidisciplinary team recommendation. *Journal of the Association for Information Science and Technology* 65, 12 (12 2014), 2518–2533.

[10] Christoph Dorn, Florian Skopik, Daniel Schall, and Schahram Dustdar. 2011. Interaction mining and skill-dependent recommendations for multi-objective team composition. *Data & Knowledge Engineering* 70, 10 (10 2011), 866–891.

[11] Christiane Fellbaum (Ed.). 1998. *WordNet: an electronic lexical database.* MIT Press.

[12] Shinya Imai and Junzo Watada. 2011. A Rough Sets Approach to Human Resource Development in IT Corporations. In *International Journal of Simulation: Systems, Science and Technology.* 249–273. https://doi.org/10.1007/978-3-642-19820-5_13

[13] Theodoros Lappas, Kun Liu, and Evimaria Terzi. 2009. Finding a team of experts in social networks. *KDD* (2009).

[14] Ching Yung Lin, Nan Cao, Shi Xia Liu, Spiros Papadimitriou, Jimeng Sun, and Xifeng Yan. 2009. SmallBlue: Social network analysis for expertise search and collective intelligence. In *ICDE.*

[15] Ioanna Lykourentzou, Robert E. Kraut, and Steven P. Dow. 2017. Team Dating Leads to Better Online Ad Hoc Collaborations. In *CSCW.*

[16] Tarek Mahmoud. 2011. Adaptive control scheme based on the least squares support vector machine network. *International Journal of Applied Mathematics and Computer Science* 21, 4 (12 2011), 685–696.

[17] Judith Masthoff. 2015. Group Recommender Systems: Aggregation, Satisfaction and Group Attributes. In *Recommender Systems Handbook.* Springer US, Boston, MA, 743–776. https://doi.org/10.1007/978-1-4899-7637-6_22

[18] Shawn Minto and Gail C. Murphy. 2007. Recommending Emergent Teams. In *MSR.*

[19] Ramakanta Mohanty, Vadlamani Ravi, and Manas Ranjan Patra. 2010. The application of intelligent and soft-computing techniques to software engineering problems: a review. *International Journal of Information and Decision Sciences* 2, 3 (2010), 233.

[20] Robert Nowicki. 2010. On classification with missing data using rough-neuro-fuzzy systems. *International Journal of Applied Mathematics and Computer Science* 20, 1 (3 2010), 55–67.

[21] Violina Ratcheva. 2007. Redefining multidisciplinary team boundaries in resolving heterogeneous knowledge dilemmas. *International Journal of Intelligent Enterprise* 1, 1 (2007), 81.

[22] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to Recommender Systems Handbook. In *Recommender Systems Handbook.* 1–35. https://doi.org/10.1007/978-0-387-85820-3_1

[23] Kostas Stefanidis and Evaggelia Pitoura. 2013. Finding the Right Set of Users: Generalized Constraints for Group Recommendations. *CoRR 2013* (2 2013). http://arxiv.org/abs/1302.6580

[24] A Stevenson. 2010. *Oxford Dictionary of English* (3rd edition ed.). Oxford University Press. https://doi.org/10.1093/acref/9780199571123.001.0001

[25] Maria Stratigi, Haridimos Kondylakis, and Kostas Stefanidis. [n. d.]. FairGRecs: Fair Group Recommendations by Exploiting Personal Health Information. In *DEXA.*

[26] D. Strnad and N. Guid. 2010. A fuzzy-genetic decision support system for project team formation. *Applied Soft Computing* 10, 4 (9 2010), 1178–1187.

[27] Ismail H. Toroslu and Yilmaz Arslanoglu. 2007. Genetic algorithm for the personnel assignment problem with multiple objectives. *Information Sciences* 177, 3 (2 2007), 787–803.

[28] R. Valencia-García, F. García-Sánchez, D. Castellanos-Nieves, J.T. Fernández-Breis, and A. Toval. 2010. Exploitation of social semantic technology for software development team configuration. *IET Software* 4, 6 (2010), 373.

[29] Xinyu Wang, Zhou Zhao, and Wilfred Ng. 2015. A Comparative Study of Team Formation in Social Networks. In *DASFAA.*

[30] Murat Yilmaz, Ali Al-Taei, Rory V O'Connor, and Rory V. O'Connor. 2015. A Machine-Based Personality Oriented Team Recommender for Software Development Organizations. In *Systems, Software and Services Process Improvement.*