Tampere University

Lucas Machado

Fair Team Recommendations for Multidisciplinary Projects

Faculty of Information Technology and Communication Sciences (ITC)
Master's thesis
May 2019

Abstract

Lucas Machado: Fair Team Recommendations for Multidisciplinary Projects
Master's thesis
Tampere University
Master's Degree Programme in Software Development
May 2019

With the ever increasing amount of data in the world, it becomes harder to find useful and desired information. Recommender systems, which offer a way to analyze that data and suggest relevant information, are already common nowadays and a important part of several systems and services. While recommender systems are often used for suggesting items for users, there are not many studies about using them for problems such as team formation. This thesis focus on exploring a variation of that problem, in which teams have multidisciplinary requirements and members' selection is based on the match of their skills and the requirements. In addition, when assembling multiple teams there is a challenge of allocating the best members in a fair way between the teams.

With the studied concepts from the literature, this thesis suggests 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. The K-rounds method is able to generate substantially more fair team recommendations, while keeping the same levels of quality and performance as other methods. The Pairs-rounds method presents slightly better recommendations quality-wise than the K-rounds method, but its recommendations are less fair to a small degree. The proposed methods perform well enough for use in real scenarios.

Keywords: Recommender systems, fairness, group formation, team recommendation.

Contents

1 Introduction

Availability and quantity of data is increasing day after day in our digital and connected world. According to Guo (2017), it is expected that from 2016 to 2025 the global data amount will grow by a factor of 10, more than doubling every two years. As the amount of data grows, new challenges emerge about organizing and extracting useful knowledge from it. Knowledge obtained from raw data can be utilized to get insights, to detect patterns and even for helping in decision making processes.

Recommender systems, which are a subclass of information filtering systems, are a tool to process extensive amounts of data and derive valuable information from it (Ricci et al. 2011). They work by trying to antecipate the *preference* or *rating* that a user would give to some item (Ricci et al. 2011), 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 the opposite direction, often 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, despite that there are only a few research papers published about.

Recommender systems' algorithms for team formation aim to assemble teams of individuals based on some specified criteria. For team formation, those algorithms need to extract and identify individual characteristics of the individuals, topics from documents and perform analysis and visualization of relation graphs (Datta et al. 2014). 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 (Lin et al. 2009). 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 (Datta et al. 2014). Furthermore, most of the papers describe team formation in software development context (Yilmaz et al. 2015; Colomo-Palacios et al. 2012), or based on users common interests and attributes (Dorn et al. 2011; Al-Adrousy et al. 2015; Awal et al. 2014).

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 (Lykourentzou et al. 2017). In addition, the concept of diversity is positive in a multidisciplinary team context and may also trigger serendipity (Ratcheva 2007).

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.

Inspired by 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, a gap was found in literature. There is no research available concerning multidisciplinary team formation based on skills and project requirements, especially regarding fairness when multiple teams need to be formed. For that reason, this thesis proposes the exploration of the research question *"How to create team recommendations based on members' skills and projects requirements, regarding 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 maintain the fairness aspect between teams manually, a problem which that recommender system algorithm could easily solve.

Based on existing research, Chapter 2 explores the concepts of recommender systems, fairness and diversity. State of art literature is reviewed over recommender systems main topics to provide a background for the research question.

Chapter 3 gives a motivating problem and formally defines it. The motivating problem illustrates the research question and helps to understand the real world problem that inspired this thesis. The formal definition specifies how candidates could be chosen as team members, since it is not possible to use traditional similarity criterion as in most recommender systems methods.

Chapter 4 proposes different methods for creating team recommendations. Since a brute force algorithm would be inefficient, a heuristic method is presented. Furthermore, two other variant methods are described to improve the fairness aspect of the recommendations.

Chapter 5 describes and analyzes an experimental evaluation of the proposed methods. To evaluate the efficiency and effectiveness of the methods, a set of test

scenarios is designed and executed. The findings are compared and analyzed in regard to quality and fairness of the team recommendations.

Finally, Chapter 6 evaluates the contributions of this thesis. The objectives of this work are evaluated, and its implications are discussed. Besides the contributions, the limitations as well as possibilities for further research are presented.

2 Related work

Based on existing literature, this chapter explores the different types of recommender systems, the properties of fairness and diversity applied to them, how recommenders are evaluated, and group formation methods.

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 (Ricci et al. 2011). Their goal is to suggest items or packages of items to users (individual recommendation) or groups of users (group recommendation).

Consider that in a recommendation system the set of items to be rated is I, and the set of users is U. An item $i \in I$ could be rated with a score by a user $u \in U$. The score is defined as $rating(u, i)$ in the interval $[0.0, 1.0]$, and the set of all recorded ratings is R. Often, the amount of items in the set I is vast, whereas only a few of them are rated by the users. Furthermore, $I(u)$ is the subset of rated items by a user $u \in U$, while similarly $U(i)$ is the subset of users which gave a rating to an item $i \in I$ (Ntoutsi, Stefanidis, Rausch, et al. 2014).

Recommender systems try to estimate a relevance score for items which were not rated by the users, denoted by $relevance(u, i)$, for $u \in U$ and $i \in I$ (Kyriakidi et al. 2017). There is extensive research literature in regard to the approaches for estimating the relevance score between items and users. Typically, the methods for recommending are differentiated within: content-based, which suggest items based on the similarity to items previously well rated by the user (e.g., M. Pazzani et al. 1997; Mooney et al. 2000), collaborative filtering, that recommend items based on items which other users with similar preferences like (e.g., Konstan et al. 1997; Breese et al. 1998) and knowledge-based ones (e.g., Shoham et al. 1997).

A recommender can also create suggestions of one or more items for a group of users. Those group recommender systems work by aggregating data from individual users. Several strategies for implementing the aggregation are used, for instance the Least Misery Strategy, which uses the minimum of the ratings to minimize the misery for group members (Ricci et al. 2011).

2.1.1 Collaborative Filtering

The methods used for collaborative filtering in recommender systems are based on gathering and analyzing huge data quantities about the preferences, actions and behaviour of a user, and then based on the similarity of that user with other users,

trying to anticipate what they would like. The basic idea is that people who gave the same rating for some item in the past are likely to give the same rating for new items. For example, a person might ask friends for a movie recommendation, and if the recommendation comes from friends who share similar preferences and interests, it is more likely to be trusted than recommendations from people with other interests.

In general, given a user $u \in U$ and a set of items I, the steps that a collaborative filtering recommender system needs to take in order to generate a list of suggestions for the user u are:

- Find similar users with u.
  - Utilize an appropriate similarity function $S(u, u')\forall u' \in U, u \neq u'$.
- Predict relevances score for items not rated by u.
  - Based on the similar users to u, apply a relevance function $relevance(u, i)$, where $i \notin I(u)$, in order to produce an estimated score for that item.
- Recommend the top-k items.
  - Rank the items based on the predicted score found in the previous step and report back the k items with the highest scores.

An important benefit found in the collaborative filtering method is that it does not depend on the items being computer analyzable. For that reason, it is able to precisely suggest intricate items without the need of "comprehending" the item itself. Nonetheless, the algorithms for collaborative filtering suffer from three main problems: scalability, cold-start and sparsity. In several real scenarios in which recommendations are made by these systems, the amount of users and items involved are in the order of millions. As a result, to execute the algorithms and calculate the recommendations there is a need of a very large amount of computational power. Furthermore, in order to make precise recommendations a large number of existing data is required by these systems, thus resulting in the cold start problem (Rubens et al. 2015; Elahi et al. 2016). Finally in most cases the number of items is very large. Even the users that are more active will only have given ratings to a very small subset of items from the entire dataset. Hence, there would be only a few ratings even for the most popular items.

2.1.2 Content-based Recommendations

The content-based filtering approach creates recommendations of items based on comparing the content of the items and a profile of a user. A collection of terms

or keywords represent the content of a item, which typically are words occurring within a document. The same terms, assembled with the analysis of the items' content which the user rated or interacted with, are used to represent the user profile. Therefore, content-based algorithms try to recommend similar items to the ones already rated by the user. More specifically, several possible items are compared with the items which the user rated in the past, and then the best matches are suggested (Brusilovsky et al. 2007; Aggarwal 2016). This developed the drawback of overspecialization. The systems often tend to propose new items that are very similar to those the user has already seen, and so the system suffers from a diversity problem, where the user is not offered the opportunity to explore new items. A key feature of the content-based recommender systems is the classification learning algorithm. These algorithms work by learning a function which models the interests of each user. The function anticipates if the user will have interest in a new item, given the user model and a the item. These predictions can take a probabilistic form by estimating the probability that the user will like an item, or they can take a numeric form in which the algorithm will directly compute a numeric value that represents the item's relevance to the user, such as the level of interest. Traditional algorithms of machine learning are used for many of these algorithms. Some of the algorithms that may be used by content-based recommender systems include: Nearest Neighbor Methods, Decision Trees, Rocchio's Algorithm, Rule Induction and Naive Bayes (M. J. Pazzani et al. 2007).

The general steps required by a content-based recommender system are summarized as follows:

- Generate a profile based on user u.
  - If the profile is not given, generate a user profile that share the same attributes as the description of the items and as values of these attributes the values of items that the user has already viewed.
- Apply a classification learning algorithm.
  - Utilize one of the previously mentioned algorithms to predict a score for any new item i such that $i \notin I(u)$.
  - Alternatively, a similarity function such as cosine correlation may be used.
- Recommend the top-k items.
  - Rank the items based on the predicted score found in the previous step and report back the k with the highest scores.

A key issue of content-based filtering is if the system is capable to learn preferences of users from users' behaviour within one source of content and then use them with other types of content. For instance, it is useful to recommend news articles based on news' browsing, however it would be much more useful when discussions, videos, products, music etc. from different sources could be recommended based on news browsing (Ntoutsi and Stefanidis 2016). Furthermore, keywords alone may not be sufficient to judge the quality and relevance of an a item. Finally the content-based algorithm suffer from the cold start problem, since they require a training dataset in order to extract the user profile from the items that the user has already seen.

2.1.3 Knowledge-based Recommendations

Recently, knowledge-based recommender systems have received a lot of attention. These systems require not only information (knowledge) about the users and the suggested items, but they also necessitate the domain knowledge about how these suggested items respond to the users needs and preferences (Frikha et al. 2017). In more detail, a knowledge-based recommender system requires the following knowledge; information on the users and their corresponding contextual parameters, information about the items and their features, and finally, the knowledge models, meaning, the knowledge about the matching between the item and the users needs.

Two main categories of knowledge-based systems are the constraint-based and case-based recommender systems (Aggarwal 2016). The recommendation procedure is similar for both methods; the users specify their requirements and items that satisfy them are tried to be identified by the system. If no items are found, then the users need to change their requirements. The difference being that in constraint-based systems the users explicitly define their requirements as a set of recommendation rules that the system tries to satisfy, while in the case-based systems, different similarity measures are used based on the requirements of the users, such as maximize or minimize certain properties, e.g., "more RAM memory" and "lowest price" respectively.

Knowledge-based approaches answer to the sparsity problem that collaborative filtering suffers from. Additionally, often a user wants to define a requirement for the suggested items explicitly, a feature that is supported in knowledge-based systems but not for example in collaborative filtering or content-based ones. At the same time however, this is also one of the major drawbacks of this approach. The process to acquire this knowledge explicitly is time consuming.

2.1.4 Group recommender systems

In many cases, it would be interesting to recommend items to a group of users instead of to a single user. For example, a group of friends might want to watch a movie together or have a playlist of songs for a party, and recommendations could be given based on all group members' models. Creating recommendations to groups is more complex than suggesting items to individuals.

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 (Ricci et al. 2011). Two approaches can be utilized for generating group recommendations (Baltrunas et al. 2010; Jameson et al. 2007). In the first one a joint profile of a group of users is created, and recommendations are made based on a created artificial user that serves as the entire group (McCarthy et al. 1998; Yu et al. 2006). Thus, the recommendation represents to some extent the group preferences mediated in this artificial user. The general steps for this approach are defined below:

- Based on a group of users G, $G \subset U$, create an artificial user v
  - Several techniques could be used to create the artificial user v. While there is not a well founded method, averaging of group members' attributes is commonly used.
- Apply a recommendation algorithm to the created user v.
  - Depending on the context, collaborative filtering, content-based filtering, or knowledge-based approaches can be used to predict the score of items for the user v.
- Recommend the top-k items.
  - Rank the items based on the predicted score found in the previous step and report back the k items with the highest scores.

The other approach creates individual and ranked recommendations for all the users in a group, then an aggregation algorithm combines them into a single list for the group. In a problem of recommending recipes, Berkovsky et al. (2010) compared both the approaches and found that the performance of the first one is slightly better. The generic steps for the second approach are described as follows:

- Apply a recommendation algorithm to all the users of the group G, $G \subset U$.

  - Depending on the context, collaborative filtering, content-based filtering, or knowledge-based approaches can be used to predict the score of items for the users.
  - For every user $u \in G$, a list of recommendations is created in the same way as individual recommendations.

- Aggregate the recommendations into a single list.
  - An algorithm aggregates all the lists of recommendations generated in the last step into a single list.
  - Different methods could be used for combining the lists, such as Optimal Aggregation, Kendall tau Distance, Average Aggregation, Least Misery Aggregation and Borda Count Aggregation (Dwork et al. 2001; Meena et al. 2013; Ricci et al. 2011).

- Recommend the top-k items.
  - Rank the items from the single list created in the previous step by their predicted scores and report back the k items with the highest scores.

Group recommendations presents a research challenge in evaluating and improving their effectiveness. Many researchers tried to investigate these aspects (e.g. Amer-Yahia et al. 2009, Ntoutsi, Stefanidis, Nørvåg, et al. 2012, Stratigi et al. 2018, etc.), and most of the literature focus on group formation and evolution, interfaces that give support to group recommenders and privacy concerns (Chen et al. 2016). Ntoutsi, Stefanidis, Nørvåg, et al. (2012) for example, suggests a model that uses recommendations for items that users similar to the members of a group liked in the past. Users are separated into clusters by their similarity, and then recommendations are based on the cluster members.

One user study that evaluated the advantages of group recommendations is PolyLens. PolyLens is group recommender system extended from the MovieLens recommender (O'Connor et al. 2005). In that study, users were allowed to compose groups while the system analyzed how the groups had an influence on the way users used MovieLens. Using the least misery heuristic, group recommendations were created by combining recommendations of individual group members. Different criteria were used to evaluate user satisfaction, such as the easiness of creating groups and adding members to it, the usefulness of group recommendations and overall satisfaction. In addition to other findings, the study came to the conclusion that when in a group, users prefer group recommendations.

Other works try to incorporate the property of fairness in group recommenders. The model proposed by Stratigi et al. (2018) uses the semantic distance between

users to balance the recommendations. Different methods can be used to improve the fairness based on the recommender system domain.

Group recommendations can also be used to recommend to individual users, since aggregating recommendations to a group is similar to aggregating multi-criteria recommendations (Ricci et al. 2011). For instance when recommending news, the criteria topic, location and how recent the items are could be used within a group recommender. However, this criteria should not be treated in the same way and different weights must be assigned to them. In that way, by using the criteria it is possible to better predict the final user overall satisfaction.

2.1.5 Diversity in Recommender Systems

In some situations it may not be useful for a user to receive recommendations of similar items. For example when exploring travel destinations, it could be better to receive a set of suggestions for different locations rather then suggestions for different hotels in the same location. The property of diversity in a recommender system deals with that issue. Usually diversity is described as being the opposite of similarity. One of the most used approaches to measure diversity is based on calculating item-item similarity by their contents (Smyth et al. 2001; Ricci et al. 2011).

In research literature many definitions for diversity are found. According to Drosou et al. (2010), they could be mostly categorized in: (i) content-based, in which items are selected by the dissimilarity in their attributes in relation to others items (for example when information does not overlap) (e.g., Zhang et al. 2008; Stefanidis, Drosou, et al. 2010); (ii) novelty-based, in which items are selected based on whether they contain new information that were not shown to the user before (e.g., Clarke et al. 2008); and (iii) semantic-based, when items are selected by belonging to distinct topics and categories (e.g., Agrawal et al. 2009)

Generally, selecting diverse items is defined as selecting k items within a set, such that within the k items the diversity is maximized (Kyriakidi et al. 2017). In the content-based method, the main activity is selecting items that present dissimilarity in relation to each other by not containing information that overlaps. The dissimilarity could be calculated by the contents of the items for example using a Jaccard-like definition of distance (Kyriakidi et al. 2017). However, oftentimes items are poorly described or do not have enough content, thus not being effective for dissimilarity calculation. In those cases, other methods could be used, such as the *ratings-based approach* as described by Kyriakidi et al. (2017). That approach takes advantage of the ratings given by a set of users for particular items. The idea is to define the similarity between two data items by how large is the set of users who gave ratings to both items.

Moreover, there could be a trade-off between diversity and other recommender systems properties such as accuracy. Some approaches try to include diversity by relaxing the possible items to a trust-region, thus maintaining the accuracy, or using user preferences to obtain diverse items that are still relevant to the user (e.g. Zhang et al. 2008, Stefanidis, Drosou, et al. 2010).

2.1.6 Fairness in Recommender Systems

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 (Serbos et al. 2017; Stratigi et al. 2018; Yao et al. 2017; Burke 2017). 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. Christensen et al. 2011, Masthoff 2015, Quijano-Sanchez et al. 2013, 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.

For example, Stratigi et al. (2018) proposes including fairness in a novel measure of similarity for creating group recommendations in the health domain. The goal is to recommend health documents to groups of patients, using an aggregation method based on the semantic distance between their health problems. Since the patients usually have a variety of health problems, fairness is needed within these group recommendations. It is implemented in such a way that, in a set of recommendations there will be at least a few interesting items for each patient, regardless of not all the recommended items being interesting for the patient. Typically, if at least a few recommended items are good enough, the user is able to tolerate being suggested other non-interesting items.

2.1.7 Evaluation of Recommender Systems Efficacy

There are three methods to measure how effective the suggestions generated by a recommender system are. The effectiveness can be measured through user studies, A/B tests (online evaluation), and offline evaluations (Beel, Genzmehr, et al. 2013).

User studies are conducted in such a way that a small amount of users (dozens or a few hundred) act as judges by evaluating which recommendations are best between the results of distinct recommendation methods.

A/B tests works similarly to user studies but are applied in large scale and the users are not focused on the task of evaluation. Different recommendation methods results are presented to thousands of users and the success of those methods are measured implicitly with click-through or conversion rates. A/B tests are typically applied to real products and services with active user bases.

Offline evaluations are done by using previous data of users' ratings. Based on this historic information, recommendations can be evaluated.

Different metrics are used to evaluate recommender systems output. The most often used are the root mean squared error and the mean squared error (Candillier et al. 2007). In addition, the quality of a method for recommending items is also assessed by information retrieval metrics as discounted cumulative gain and precision and recall (Candillier et al. 2007). There are other factors regarded as important in the evaluation such as coverage, novelty and diversity (Lathia et al. 2010). Nonetheless, there is a lot of criticism towards many of the traditional evaluation measures (Turpin et al. 2001).

A recommendation method effectiveness is then measured by how well that method is able to anticipate the ratings that would be given by users in the recommender dataset. However, whereas a rating is a clear representation of a user liking an item, not all domains provide the concept of rating items, such as in the domain of team formation, in which users commonly do not rate a team. In those situations, implicit measurements of effectiveness could be used with offline evaluations. For instance, a recommender system might be presumed effective if the teams are built with as many members as possible from a members reference list.

Several researchers however, are critical towards this type of offline evaluations (Jannach et al. 2013; Turpin et al. 2001; Beel, Genzmehr, et al. 2013). For example, it has been demonstrated that there is a low correlation between results of A/B tests and user studies and offline evaluation results (Turpin et al. 2001; Beel and Langer 2015). Moreover, a commonly used dataset in offline evaluation has been brought wrong conclusions in algorithms' evaluation due to containing duplicated data (Basaran et al. 2017). Oftentimes, the assessed user-satisfaction of a recommender also does not correlate to the results of alleged offline evaluations (Beel, Genzmehr, et al. 2013). Therefore, results of offline evaluation need to be interpreted carefully and critically.

2.2 Group formation

The employment of soft-computing and smart methods for selecting personnel has been extensively researched. In contrast with Malinowski et al. (2008) claims, several studies support selection of individuals with the use of computer intelligence techniques and information systems (e.g. Strnad et al. 2010, Toroslu et al. 2007, Celik et al. 2009).

Mohanty et al. (2010) presents an important review on this field. The attempts to solve personnel selection problems with technology are frequent (e.g. Barreto et al. 2008, Barcus et al. 2008). For example for selecting teams of software engineers many techniques are used, such as fuzzy logic (e.g. Strnad et al. 2010, J. Wang et al. 2003), semantics (e.g. García-Crespo et al. 2009, Valencia-García et al. 2010) and rough sets (e.g. Imai et al. 2011). Many authors also address the issue of combining these intelligent systems approaches aiming to improve the selection of members' performance (e.g. Zhong et al. 2001, Mahmoud 2011, Nowicki 2010, Li et al. 2011).

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. (2012) describe how to use rough sets and fuzzy logic to form and recommend Scrum teams, based on team members' roles. The recommendations are based on hybrid techniques and are described by the following steps:

- Labeling of competences: Each work package (a project for a Scrum team) receives labels of the required competences, that are also weighted by their relative importance for the project.
- Fuzzy transformation: The weights of the labels are transformed into linguistic values with the use of fuzzy methods. Thus, the matching between staff and project required competencies is made easier.
- Rough set categorization: Based on a set of earlier assessments, a rough set method is used to determine the competence level of each individual within the required competencies of the project.
- Matching and recommendation process: Each candidate is matched with the competences, then the system recommends teams for the project based on different criteria such as minor gap, ranked teams, best teams, etc.

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.

A recommender system for team formation in which extreme situations are taken into consideration is described by Al-Adrousy et al. (2015). Such system is designed for a Mobile Ad-hoc Network context, in which challenges as intermittent connectivity, limited coverage and limited computing power are present. Therefore, the recommender can not assume the same conditions of most other recommender systems, such as time stability of how users exist in the network to be selected. The aim is to build teams of skilled members in short intervals of time for ad-hoc projects, which for instance could be about writing codes, testing or creating websites, or designing specifications. Another possible use could be for exchanging materials (books, code or articles).

Besides restrictions of the domain, constraints can also be applied to recommender systems, as described by Stefanidis and Pitoura (2013). 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 et al. (2007) suggest an approach to form and recommend emergent teams based on how software artifacts are changed by developers. Yilmaz et al. (2015) 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. (2009) 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

This chapter provides a motivating example to illustrate how the concepts of recommender systems and fairness could be used together to solve a real-world problem. Furthermore, the problem being investigated is formally defined, while referencing the literature.

3.1 Motivating example

Assume that there are several projects that aim to create products and satisfy needs. Each one of these projects 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 back-end development, front-end development, design and prototyping 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. Figure 3.1 shows nine individuals with their skill sets and 3 projects with their required 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 (Stevenson 2010) and that requires a team of people with specific skills for that.

This thesis 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 3.2 shows the result of the recommender 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

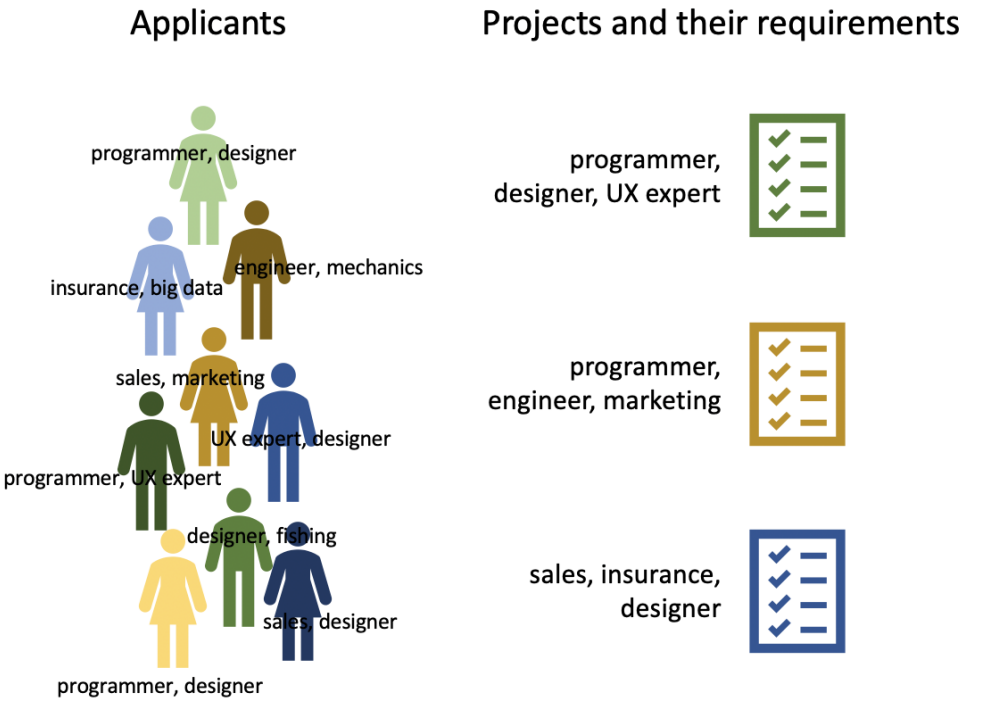

Figure 3.1 *Example of a set of individuals and their skills, and a set of projects and their required skills*

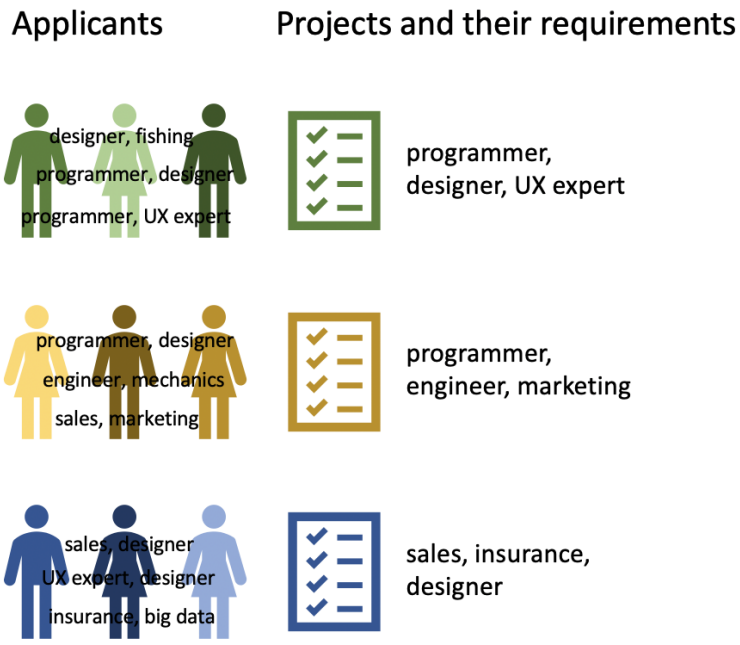

Figure 3.2 *Example of suggested teams of three applicants each, and their skills, for a set of three projects and their required skills.*

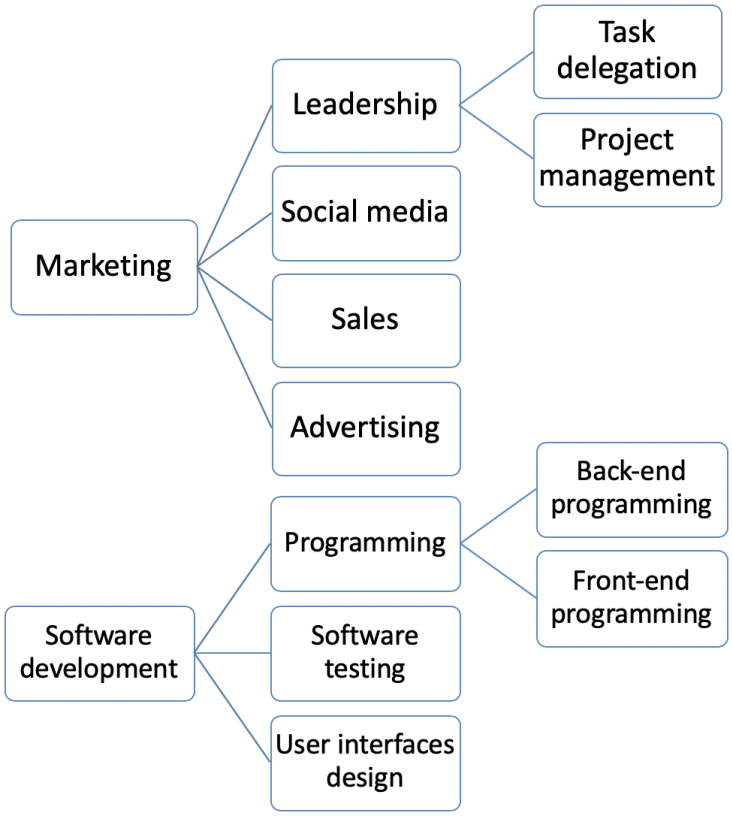

Figure 3.3 *Example of a set of skills in a hierarchy*

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 hierarchy relationship. Figure 3.3 shows an example of a set of skills and their relations. In this example, skills are represented by nodes which are connected by edges and follow an order from the most abstract (root) skill to more specialized (branches and leaves) skills. The skill "Software development" is related to the skill "Programming", which is related to the skill "Back-end programming".

The similarity between two skills s_1 and s_2 can then be calculated by the shortest path distance between their corresponding nodes in the hierarchy. The distance between two nodes $dist(s_1, s_2)$ is the graph geodesic distance considering non-weighted edges. Therefore, any existing edge between two nodes represents a distance of 1. To find the shortest path distance, where the skill $s_{lca} = LCA(s_1, s_2)$ is the least common ancestor of s_1 and s_2, we compute the sum of distances from s_{lca} to s_1 and from s_{lca} to s_2. Accordingly, the $similarity(s_1, s_2)$ function is defined as follows:

$$similarity(s_1, s_2) = \begin{cases} 0, & \text{if no relation} \\ 1, & \text{if } s_1 = s_2 \\ \dfrac{1}{dist(s_1, s_{lca}) + dist(s_2, s_{lca})}, & \text{otherwise} \end{cases}$$

Therefore, the similarity between the skills "Software testing" and "Back-end programming" in Figure 3.3 is $\frac{1}{3}$, while the similarity between the skills "Back-end programming" and "Front-end programming" is greater at the value of $\frac{1}{2}$. The similarity of the skills "Programming" and "Advertising" is 0, since they are not in the same hierarchy and it is not possible to find a common ancestor skill.

Different sets of skills are attributes of different applicants. Let A be a set of applicants to the projects, in which each applicant a, $a \in A$, possess a set of skills $\{s_1, \ldots, s_n\}$, in which s_i is a textual tag representing a skill.

Example 3.2.1 (Applicants a_1 and a_2 and their sets of skills)
$a_1 = \{$*programming, user experience, photography*$\}$
$a_2 = \{$*visual design, marketing, sales, cooking*$\}$

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 3.2.2 (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.

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

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 $scoreAR(a, r)$.

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

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 3.2.3 below. It is assumed that all teams are formed with the same amount of applicants k.

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

Definition 3.2.3
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 problem presents itself similarly to a packages-to-group recommendation, in which packages (teams) of items (applicants) are recommended to groups of projects. 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 thesis.

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 3.2.3, 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 3.3.1 below.

Definition 3.3.1
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 3.3.1 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 thesis 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 thesis proposes to achieve fairness through the way in which teams are formed, considering the presented restrictions of the context.

4 Methods

This chapter proposes different methods for creating team recommendations. Section 4.1 defines a brute force method, while Section 4.2 specifies a heuristic approach and further optimizations to increase fairness in the recommendations.

4.1 Brute force algorithm

Based on the functions of Chapter 3, Algorithm 1 implements the function $scoreAP(a, p)$ which calculates how well a given applicant a is suited to a given project p.

Algorithm 1: Function $scoreAP(a, p)$

Input: An applicant a and a project p

```
scoreAP = 0;
foreach r_i in p.requirements do
    /* calculates scoreAR between applicant a and individual
       project requirement r_i                                  */
    scoreAR = 0;
    foreach s_j in a.skills do
        scoreAR = scoreAR + similarity(s_j, r_i);
    end
    scoreAP = scoreAP + scoreAR;
end
```

Output: scoreAP

Result: How well an applicant a skills match with a project p requirements

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, as shown in Example 4.1.1 below. In our context k is the amount of applicants in a team, and L is the set of all applicants. Algorithm 2 below uses the $combinations(k, L)$ function and Algorithm 1 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 2 is factorial as denoted by $O(k \times \binom{A}{k} \times P)$, or $O(n!)$.

Example 4.1.1 (Use of the function $combinations(k, L)$)
Let $k = 2$ *and* $L = \{1, 2, 3\}$,
$combinations(k, L) = [\{1, 2\}, \{1, 3\}, \{2, 3\}]$

Algorithm 2: Brute force method to generate best team recommendations

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 $bestTeams$;
  `/* After the best team for a project is determined, its members are removed from the set of applicants` A `*/`
  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 2 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.

Algorithm 3 below also uses the function from Algorithm 1, but in place of generating all possible team combinations, it 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.

Algorithm 3: Heuristic method to generate team recommendations

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 $bestTeams$

Furthermore, Algorithm 3 could be optimized to improve the fairness according to Definition 3.3.1. That optimization is specified in the novel variants Algorithm 4 and Algorithm 5 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 4 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.

Algorithm 4: *K-rounds choosing* method to generate team recommendations

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 $bestTeams[p_i]$;
        /* After $bestApplicant$ is chosen for a project p_i, it should be unavailable for other projects. Therefore, it is removed */
        remove $bestApplicant$ from $projectApplicantsScores$;
    end
end

Result: Set of $bestTeams$

Similarly to Algorithm 4, Algorithm 5 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 $scoreAP$ 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 2, the asymptotic computational complexities of Algorithms 3, 4 and 5 are linear as defined by $O(A \times P + k \times P)$, or $O(n)$.

Algorithm 5: *Pairs-rounds* method to generate team recommendations

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 $bestTeams[p_i]$;
        /* After $bestApplicant$ is chosen for a project p_i, it should be unavailable for other projects. Therefore, it is removed */
        remove $bestApplicant$ from $projectApplicantsScores$;
        $secondBestApplicant =$
         $max(projectApplicantsScores[p_i] :: scoreAP)$;
        put $secondBestApplicant$ in $bestTeams[p_i]$;
        /* $secondBestApplicant$ should be also unavailable for other projects and it is removed */
        remove $secondBestApplicant$ from $projectApplicantsScores$;
    end
end

Result: Set of $bestTeams$

5 Experimental Evaluation

This chapter describes an experiment using the methods from Chapter 4. The objective of the experiment is to test the feasibility, efficiency and efficacy of the proposed algorithms for team recommendations creation. It is expected that the heuristic-based methods perform significantly faster than the brute force approach. In addition, the k-rounds and pairs-rounds variants are expected to show better fairness than the previous methods.

Several test scenarios are created and executed through the implemented algorithms of the methods. Section 5.1 describes the data that was used in the experiment. Section 5.2 presents two measurements that are collected and analyzed during the experiments. The quality of the recommendations is measured by the sum of their scores, while the fairness aspect is measured by a novel fairness-deviation indicator. Section 5.3 details how the algorithms were implemented and the test scenarios were created and executed. Finally, Section 5.4 and Section 5.5 evaluate the expectations and summarize the findings of this thesis.

5.1 Dataset

A preprocessed dataset derived from the DBLP dataset[1] is used for testing. The DBLP dataset is an online bibliography database for publications on computer science (Ley 1997), created by Trier University. The preprocessed dataset was created by X. Wang et al. (2015) and consists of a CSV (comma-separated values) file of 7428 lines. Each line corresponds to a researcher from DBLP dataset 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. Example 5.1.1 below illustrates a line from the dataset representing a researcher named *Alpa Jain* and seven skills related to that person.

Example 5.1.1 (A line from the preprocessed dataset)

```
Alpa Jain,text,extraction,queries,information,query,sql,databases
```

Due to the nature of the DBLP dataset, the skills associated with the researchers correspond to keywords used in those researcher's scientific publications. Truthfully, it may not represent the same definition of skills used in this work (the ability or expertise to do something well). However, since the skills derived from the DBLP dataset represent an area of knowledge or expertise in which a person published

[1] `https://dblp.uni-trier.de/`

research, it could be considered as a sufficient approximation. In addition, a best suited dataset was not found publicly.

To assemble the hierarchy relationship between the skills, the Wordnet[2] database was used. Wordnet is a large lexical database in English language, in which words are grouped by their cognitive synonyms (synsets) (Fellbaum 1998; Fellbaum 2005). 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 well as a way to compute similarity between the skills derived from the preprocessed DBLP dataset, since the skills are words without any connections between them.

5.2 Measurements

In our experimental evaluation, we compare how well the proposed heuristic methods perform timewise, studying different parameters, especially since the presented brute force approach has a factorial asymptotic complexity ($O(n!)$) and the heuristic methods have linear asymptotic complexity ($O(n)$). For that reason, and for verifying the linearity of those methods, 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 $\text{mean}(\mathcal{T})$ is the arithmetic mean of the $scoreTP$ values of the teams in the set $\mathcal{T}$, and $\text{len}(\mathcal{T})$ is the count (length) of teams in the set.

[2] `https://wordnet.princeton.edu/`

Table 5.1 below illustrates an example of mean, sum and fairness-deviation values calculated with $scoreTP$ values of hypothetical sets $\mathcal{T}$ of teams. The four combinations of teams' $scoreTP$ values in the first column are all different. Despite that all the sets $\mathcal{T}$ have the same mean (50) and sum (150) values, the fairness-deviation values are different. While the first row depicts a set of perfectly fair teams, the following rows represent distinct levels of deviation from absolute fairness. The fairness-deviation value grows as the difference between the $scoreTP$ values increases within a set of teams.

Table 5.1 *Example of fairness-deviation calculation for sets of teams*

$scoreTP$ values for $\mathcal{T}$	mean($\mathcal{T}$)	sum($\mathcal{T}$)	fairness-deviation($\mathcal{T}$)
{50, 50, 50}	50	150	0.00
{45, 50, 55}	50	150	3.33
{35, 50, 65}	50	150	10.00
{12, 42, 96}	50	150	61.33

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[3].

The script takes advantage of the fact that the datasets used are not from live environments of online services – and therefore, they do not change – to precompute some steps. 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' hierarchy relationship, a similarity matrix between all the skills is calculated and saved to speed up the overall algorithm.

NLTK (Natural Language Toolkit)[4] – a software library to work with human language data – provides an API[5] to access Wordnet dataset and exposes a function *path_similarity* that can be used to calculate the similarity between two synsets. It returns a value between zero and one, with one representing identity (the two synsets are exactly the same), and zero representing no link between the synsets. Due to the fact that a word may convey different meanings – and therefore, be represented by several different synsets within Wordnet –, 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

[3] `https://github.com/machadolucas/Team-Recommender`
[4] `http://www.nltk.org`
[5] `http://www.nltk.org/howto/wordnet.html`

similarity to all other skills is considered to be zero, except the similarity to itself that is then considered one. The *path_similarity* function was then used to calculate the similarity between all the unique skills extracted from our dataset, producing a 4480×4480 matrix. Once filled, the similarity matrix is saved as a binary file that can be quickly loaded for further executions of the script.

Regardless that this precomputation was done with a static dataset, a precomputed similarity matrix could be used also for dynamic data with the appropriate changes in the code. For every new skill that would be added to the skill set, it would be needed to calculate only the similarity between the new and the previous skills since the similarity value works bidirectionally. Then those new values would be added to the matrix.

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 are organized into *test groups*, which may contain one or more of them. Each test group has a different goal regarding analyzing algorithms efficiency or effectiveness. Table 5.2 below summarizes the different test groups, the amount of test scenarios and the purpose for each group.

Table 5.2 *Test groups, amount of scenarios in each, and their purposes.*

Test group	**Amount of scenarios**	**Purpose**
1: Varying amount of projects	10	Analyze time performance
2: Varying amount of team members	10	Analyze time performance
3: Main test with random parameters	100	Compare heuristic algorithms
4: Very small dataset for brute force analysis	1	Analyze brute force algorithm

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 few teams have less than 3 or more than 12 members: Agile teams, for example, usually have less than 10 members (Canty 2015). All the possible combinations between the 10 sampled p and 5 sampled k values are used to generate a total of 50 test scenarios. As an alternative to the full set of 4480 unique skills, 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, this second set of 50 test scenarios uses 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. Instead of using the full set of 7428 applicants, a sample of 100 applicants is extracted and used. Furthermore, the task consists of creating recommendations for only 4 projects ($p = 4$) with 4 team members each ($k = 4$). The whole skill set is used to create project requirements, since it should not affect the performance of the algorithms.

The algorithms executed in each test scenario are different depending on the test group. For each test scenario in test groups 1, 2 and 3, Algorithms 3 (heuristic method), 4 (*K-rounds* method) and 5 (*Pairs-rounds* method) are executed. In test group 4, Algorithm 2 (brute force) is executed in addition to the others.

Finally, measurements collected during all the tests are printed by the script on computer terminal, and used for creation of the relevant figures (graphs). Example 5.3.1 below illustrates the summarized output of a test scenario from test group 3.

Example 5.3.1 (Example of terminal output for a test scenario)

```
---------- Test input: ----------
Applicants: 7428 Projects: 20 k: 8

Heuristic results: Took: 7.646s
Scores: Max:1058.281 Min:214.749 Range:843.532 Mean:365.291
Sum:7305.829 f-deviation:112.729

K-rounds heuristic results: Took: 8.541s
Scores: Max:564.448 Min:273.220 Range:291.228 Mean:364.754
Sum:7295.083 f-deviation:50.836

Pair-rounds heuristic results: Took: 8.530s
Scores: Max:672.251 Min:260.797 Range:411.454 Mean:364.807
Sum:7296.139 f-deviation:63.049
```

5.4 Efficiency

Calculating the best teams with the brute force method (Algorithm 2) 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 $\binom{N}{k}$ (in which N is the amount of applicants available and k is the amount of members in the team), 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 $2.3282073138 \times 10^{16}$ seconds, or 737 million years.

Table 5.3 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 ($\binom{100}{4} + \binom{96}{4} + \binom{92}{4} + \binom{88}{4}$) 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 that reason, the brute force algorithm was not executed in other test scenarios.

The time spent to execute the Algorithms 3 (simple heuristic method), 4 (*k-rounds* method) and 5 (*pairs-rounds* method) with test groups 1 and 2 is shown in

Table 5.3 *Results of test scenario from test group 4*

Method	Time (s)	scoreTP sum	fairness-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

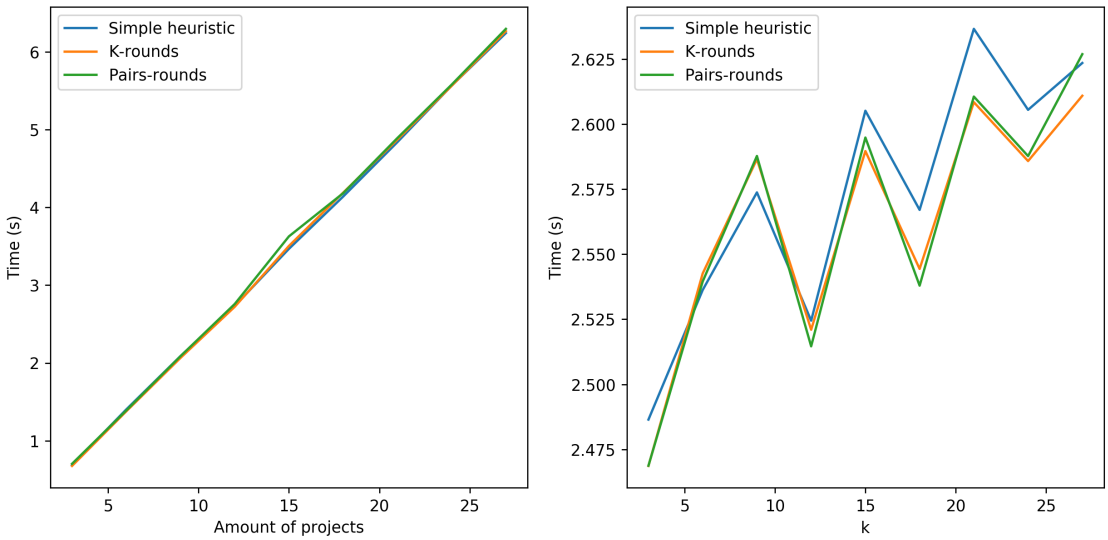

Figure 5.1 *Time consumed by algorithms*

Figure 5.1. 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 Chapter 4.

Nevertheless, on the right side of the Figure 5.2 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.

Furthermore, 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. User studies would require that a small amount of users judge the quality of the recommendations, but with our DBLP derived dataset and random

generated project requirements it would be a rather difficult and imprecise task. For the execution of A/B tests, a large system for recommending teams with our dataset would be needed. That system would have to be into regular use by thousands of users. Offline evaluation is also not possible because there is not historical datasets with information of users' ratings of team recommendations with the used dataset. Therefore, 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 5.2 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 ($k \times \text{len}(\mathcal{T})$). A greater result of this indicator points out a possible better overall investment of desired human potential on the projects, since the value refers to a better quality and quantity of matches between applicants' skills and project requirements. 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 project requirements used for test scenarios in test group 3, in Figure 5.2 is also possible to notice two linear areas of concentrated points. The first area is over the trend lines and represent the tests executed with projects using requirements from the 200 most frequent skills. More frequent skills in project requirements have a bigger chance to find more similarities with applicants' skills, thus increasing the overall value of $scoreTP$. The second area of concentrated points under the trend lines represent the tests executed with the 200 less frequent out of the 2000 most frequent skills, that in contrast to the project requirements from the tests of the first area, have smaller chances of finding similarities with applicants' skills. However, the relations between the algorithms' are consistent regardless of the skills used as project requirements.

Figure 5.3 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

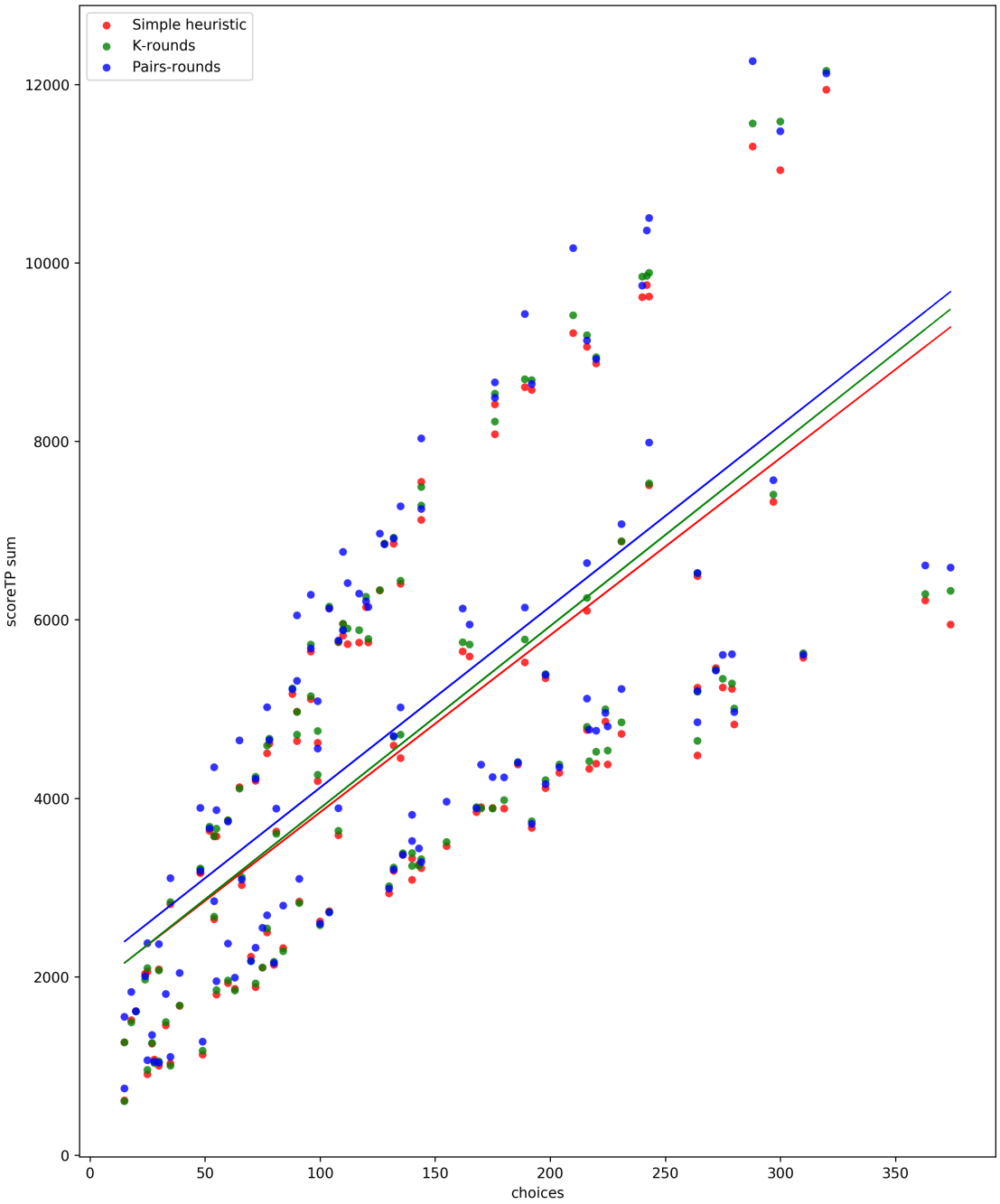

Figure 5.2 *Sum of scoreTP values over amount of choices made, for different algorithms*

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. It is also interesting to notice how the *Pairs-rounds* method has a tendency to converge to the same fairness levels as the *K-rounds* method, as the amount of choices made increase. Figure 5.3 was also created from the results of test groups 1, 2 and 3.

Based on this data, *k-rounds* (Algorithm 4) and *pairs-rounds* (Algorithm 5) methods clearly show an large improvement in fairness measurements when comparing to the original simple heuristic method (Algorithm 3).

As can be noticed in the results of Table 5.3, the brute force method (Algorithm 2) 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 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 5.2.

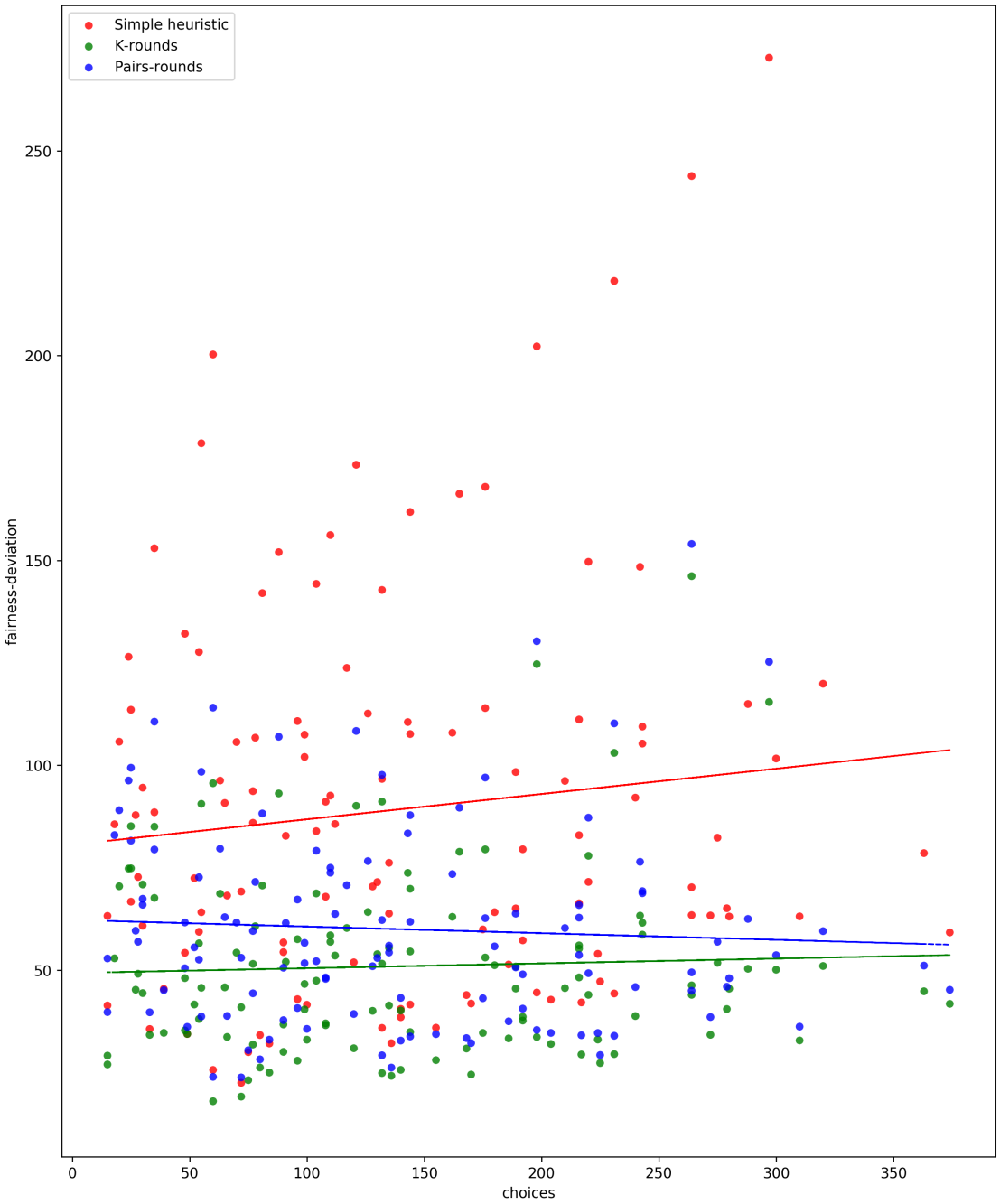

Figure 5.3 *Fairness-deviation over amount of choices made, for different algorithms*

6 Conclusions

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. As expected, the recommendations have at least the same level of quality as the brute force approach, but are calculated tens of thousands times faster.

The suggested variations of the heuristic method also prove to create results with more fairness, as expected. K-rounds method is able to generate significantly more fair team recommendations, while maintaining the same levels of quality and performance as other methods. Pairs-rounds method shows slightly better overall quality in the recommendations than the K-rounds method, however performing slightly worse regarding fairness. Both of these approaches present a solid answer for achieving more fair team recommendations.

For simplicity, during the experiments we assumed fixed values for some variables. Despite having several test scenarios with varying amounts of projects, different amount of team members for each project and even different sets of applicants, the amount of required skills per project was constant. Moreover, within a test scenario, the amount of members per team was the same for all the projects. In real scenarios those values may differ, which may lead to misunderstandings with the presented measurements. For example, teams with more members or projects with more requirements have a tendency to have bigger $scoreTP$ values, since there is more possible matches between applicants skills and project requirements. Thus, some teams could have bigger scores just because of having different parameters, invalidating the fairness-deviation indicator. To overcome that limitation, the calculation of a team $scoreTP$ could for instance be tweaked to be use the weighted average of the members' $scoreAP$ values, relative to the amount of members and requirements in the project.

In another possible scenario in which not all project requirements are equally important and there is the need to emphasize some of them, some slight adjustments could be done in the scores' calculation. The projects requirements could have a weight factor, for example. The weight factor would be multiplied with the similarity calculation between skills to compose the $scoreAP$, thus increasing the score for applicants with the most desired requirements.

Furthermore, despite the great benefit that this work has of using the extensive Wordnet database to calculate similarity between skills, it also presents some limitations. Foremost, Wordnet only has synsets in English, and therefore, it would not work with other languages' terms. In addition, we only computed similarity between single words as Wordnet does not support compound words neither using

them for the computation. Forasmuch as in a real scenario skills could be often described in a compound word format, using Wordnet may not be fitting. Thus, better results could be achieved in the future if other multilingual and compound supporting lexical databases are available to be used.

Moreover, due to the lack of a publicly accessible and recognized dataset of applicants and their skills, the use of DBLP dataset may not exactly reflect the team formation problem. To overcome current restrictions related to the limited available data for the entities involved in a recommender systems, the enormous amount of diverse data on the Web that are created and collected without interruption can be used (Efthymiou, Papadakis, Stefanidis, et al. 2019; Efthymiou, Papadakis, Papastefanatos, et al. 2017; Christophides et al. 2015; Efthymiou, Stefanidis, et al. 2015). Several sources (ontologies, social media, etc.) could be used to enrich data at different levels (information and preferences about applicants, information about projects). Traditional problems in recommender systems such as data sparsity and cold-start could be tackled with that wealthier data input, thus generating better recommendations.

The affirmation that there is not enough data to describe an entity is not true anymore; perhaps the recommender does not have enough data, but there is plenty in the Web. The current challenge is how to gather and filter useful data, removing the noise and using it together with already existing data in the recommender, in a way that the user experience and quality of recommendations are increased.

Finally, all the proposed heuristic methods in this thesis perform well enough for use in real scenarios, considering the probable adaptations needed for specific problems. The k-rounds method could be efficiently used particularly for any team formation problem in which fairness between teams would be beneficial. For example in educational environments for assembling teams of students based on their traits (Val et al. 2014), or when teams are needed to be formed quickly to respond to short-term specific activities (that could be interpreted as projects) based on members' expertise, such as in emergency services (Nair et al. 2002). Another interesting application would be in crowd-sourcing platforms – similar to what inspired this thesis –, in which individuals from a set of enrolled people with specific interests could be combined as teams to work on projects, as described by W. Wang et al. (2017). Further expanding beyond team formation problems, any problem in which fairness in resources distribution is needed could take advantage of the fairness improving methods proposed.

References

Al-Adrousy, Waleed M., Hesham A. Ali, and Taher T. Hamza (2015). "A recommender system for team formation in MANET". In: *Journal of King Saud University - Computer and Information Sciences* 27.2, pp. 147–159. DOI: 10.1016/j.jksuci.2014.06.014.

Aggarwal, Charu C. (2016). *Recommender Systems*. Cham: Springer International Publishing. DOI: 10.1007/978-3-319-29659-3.

Agrawal, Rakesh et al. (2009). "Diversifying search results". In: *Proceedings of the Second ACM International Conference on Web Search and Data Mining - WSDM '09*. New York, New York, USA: ACM Press, p. 5. DOI: 10.1145/1498759.1498766.

Amer-Yahia, Sihem et al. (2009). "Group recommendation". In: *Proceedings of the VLDB Endowment* 2.1, pp. 754–765. DOI: 10.14778/1687627.1687713.

Awal, Gaganmeet Kaur and K. K. Bharadwaj (2014). "Team formation in social networks based on collective intelligence – an evolutionary approach". In: *Applied Intelligence* 41.2, pp. 627–648. DOI: 10.1007/s10489-014-0528-y.

Baltrunas, Linas, Tadas Makcinskas, and Francesco Ricci (2010). "Group recommendations with rank aggregation and collaborative filtering". In: *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10*. New York, New York, USA: ACM Press, p. 119. DOI: 10.1145/1864708.1864733.

Barcus, Ana and Gilberto Montibeller (2008). "Supporting the allocation of software development work in distributed teams with multi-criteria decision analysis". In: *Omega* 36.3, pp. 464–475. DOI: 10.1016/j.omega.2006.04.013.

Barreto, Ahilton, Márcio de O. Barros, and Cláudia M.L. Werner (2008). "Staffing a software project: A constraint satisfaction and optimization-based approach". In: *Computers & Operations Research* 35.10, pp. 3073–3089. DOI: 10.1016/j.cor.2007.01.010.

Basaran, Daniel, Eirini Ntoutsi, and Arthur Zimek (2017). "Redundancies in Data and their Effect on the Evaluation of Recommendation Systems: A Case Study on the Amazon Reviews Datasets". In: *Proceedings of the 2017 SIAM International Conference on Data Mining*. Philadelphia, PA: Society for Industrial and Applied Mathematics, pp. 390–398. DOI: 10.1137/1.9781611974973.44.

Beel, Joeran, Marcel Genzmehr, et al. (2013). "A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation". In: *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation - RepSys '13*. New York, New York, USA: ACM Press, pp. 7–14. DOI: 10.1145/2532508.2532511.

Beel, Joeran and Stefan Langer (2015). "A Comparison of Offline Evaluations, Online Evaluations, and User Studies in the Context of Research-Paper Recommender Systems". In: *Research and Advanced Technology for Digital Libraries*, pp. 153–168. DOI: 10.1007/978-3-319-24592-8_12.

Berkovsky, Shlomo and Jill Freyne (2010). "Group-based recipe recommendations". In: *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10*. New York, New York, USA: ACM Press, p. 111. DOI: 10.1145/1864708.1864732.

Breese, John, David Heckerman, and Car Kadie (1998). "Empirical analysis of predictive algorithms for collaborative filtering". In: *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 43–52. DOI: 10.1111/j.1553-2712.2011.01172.x.

Brusilovsky, Peter, Alfred Kobsa, and Wolfgang Nejdl, eds. (2007). *The Adaptive Web: Methods and Strategies of Web Personalization*. Berlin, Heidelberg: Springer-Verlag.

Burke, Robin (2017). "Multisided Fairness for Recommendation". In: *The Computing Research Repository (CoRR)* abs/1707.0.

Candillier, Laurent, Frank Meyer, and Marc Boullé (2007). "Comparing State-of-the-Art Collaborative Filtering Systems". In: *Machine Learning and Data Mining in Pattern Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 548–562. DOI: 10.1007/978-3-540-73499-4_41.

Canty, Denise (2015). *Agile for Project Managers*. Auerbach Publications, p. 135. DOI: 10.1201/b18052.

Celik, Metin, I. Deha Er, and Y. Ilker Topcu (2009). "Computer-based systematic execution model on human resources management in maritime transportation industry: The case of master selection for embarking on board merchant ships". In: *Expert Systems with Applications* 36.2, pp. 1048–1060. DOI: 10.1016/j.eswa.2007.11.004.

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

Christensen, Ingrid A. and Silvia Schiaffino (2011). "Entertainment recommender systems for group of users". In: *Expert Systems with Applications*. DOI: 10.1016/j.eswa.2011.04.221.

Christophides, Vassilis, Vasilis Efthymiou, and Kostas Stefanidis (2015). *Entity Resolution in the Web of Data*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers.

Clarke, Charles L.A. et al. (2008). "Novelty and diversity in information retrieval evaluation". In: *Proceedings of the 31st annual international ACM SIGIR con-*

ference on Research and development in information retrieval - SIGIR '08. New York, New York, USA: ACM Press, p. 659. DOI: 10.1145/1390334.1390446.

Colomo-Palacios, Ricardo et al. (2012). "ReSySTER: A hybrid recommender system for Scrum team roles based on fuzzy and rough sets". In: *International Journal of Applied Mathematics and Computer Science* 22.4, pp. 801–816. DOI: 10.2478/v10006-012-0059-9.

Datta, Anwitaman, Jackson Tan Teck Yong, and Stefano Braghin (2014). "The zen of multidisciplinary team recommendation". In: *Journal of the Association for Information Science and Technology* 65.12, pp. 2518–2533. DOI: 10.1002/asi.23139.

Dorn, Christoph et al. (2011). "Interaction mining and skill-dependent recommendations for multi-objective team composition". In: *Data & Knowledge Engineering* 70.10, pp. 866–891. DOI: 10.1016/j.datak.2011.06.004.

Drosou, Marina and Evaggelia Pitoura (2010). "Search result diversification". In: *SIGMOD record* 39.1, pp. 41–47.

Dwork, Cynthia et al. (2001). "Rank aggregation methods for the Web". In: *Proceedings of the tenth international conference on World Wide Web - WWW '01*. New York, New York, USA: ACM Press, pp. 613–622. DOI: 10.1145/371920.372165.

Efthymiou, Vasilis, George Papadakis, George Papastefanatos, et al. (2017). "Parallel meta-blocking for scaling entity resolution over big heterogeneous data". In: *Information Systems* 65, pp. 137–157. DOI: 10.1016/j.is.2016.12.001.

Efthymiou, Vasilis, George Papadakis, Kostas Stefanidis, et al. (2019). "MinoanER: Schema-Agnostic, Non-Iterative, Massively Parallel Resolution of Web Entities". In: *Advances in Database Technology - 22nd International Conference on Extending Database Technology, {EDBT} 2019, Lisbon, Portugal, March 26-29, 2019*, pp. 373–384.

Efthymiou, Vasilis, Kostas Stefanidis, and Vassilis Christophides (2015). "Big data entity resolution: From highly to somehow similar entity descriptions in the Web". In: *2015 {IEEE} International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*, pp. 401–410.

Elahi, Mehdi, Francesco Ricci, and Neil Rubens (2016). "A survey of active learning in collaborative filtering recommender systems". In: *Computer Science Review* 20.C, pp. 29–50. DOI: 10.1016/j.cosrev.2016.05.002.

Fellbaum, Christiane, ed. (1998). *WordNet: an electronic lexical database*. MIT Press.

Fellbaum, Christiane (2005). "WordNet and Wordnets". In: *Encyclopedia of Language and Linguistics*. Ed. by Alex Barber. Elsevier, pp. 2–665.

Frikha, Mohamed, Mohamed Mhiri, and Faiez Gargouri (2017). "Using Social Interaction Between Friends in Knowledge-Based Personalized Recommendation".

In: *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. Vol. 2017-Octob. IEEE, pp. 1454–1461. DOI: 10.1109/AICCSA.2017.206.

García-Crespo, Ángel et al. (2009). "BMR: Benchmarking Metrics Recommender for Personnel issues in Software Development Projects". In: *International Journal of Computational Intelligence Systems* 2.3, pp. 256–266. DOI: 10.1080/18756891.2009.9727658.

Guo, Huadong (2017). "Big Earth data: A new frontier in Earth and information sciences". In: *Big Earth Data* 1.1-2, pp. 4–20. DOI: 10.1080/20964471.2017.1403062.

Imai, Shinya and Junzo Watada (2011). "A Rough Sets Approach to Human Resource Development in IT Corporations". In: *International Journal of Simulation: Systems, Science and Technology*, pp. 249–273. DOI: 10.1007/978-3-642-19820-5_13.

Jameson, Anthony and Barry Smyth (2007). "Recommendation to Groups". In: *The Adaptive Web*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 596–627. DOI: 10.1007/978-3-540-72079-9_20.

Jannach, Dietmar et al. (2013). "What Recommenders Recommend – An Analysis of Accuracy, Popularity, and Sales Diversity Effects". In: *User Modeling, Adaptation, and Personalization*. June, pp. 25–37. DOI: 10.1007/978-3-642-38844-6_3.

Konstan, Joseph A. et al. (1997). "GroupLens: applying collaborative filtering to Usenet news". In: *Communications of the ACM* 40.3, pp. 77–87. DOI: 10.1145/245108.245126.

Kyriakidi, Marialena, Kostas Stefanidis, and Yannis Ioannidis (2017). "On Achieving Diversity in Recommender Systems". In: *Proceedings of the ExploreDB'17 on - ExploreDB'17*, pp. 1–6. DOI: 10.1145/3077331.3077341.

Lappas, Theodoros, Kun Liu, and Evimaria Terzi (2009). "Finding a team of experts in social networks". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, p. 467. DOI: 10.1145/1557019.1557074.

Lathia, Neal et al. (2010). "Temporal diversity in recommender systems". In: *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '10*. New York, New York, USA: ACM Press, p. 210. DOI: 10.1145/1835449.1835486.

Ley, Michael (1997). "Die Trierer Informatik-Bibliographie {DBLP}". In: *Informatik '97, Informatik als Innovationsmotor, 27. Jahrestagung der Gesellschaft für In-*

formatik, Aachen, 24.-26. September 1997, pp. 257–266. DOI: 10.1007/978-3-642-60831-5_34.

Li, Chunshien and Tai-Wei Chiang (2011). "Function Approximation with Complex Neuro-Fuzzy System Using Complex Fuzzy Sets – A New Approach". In: *New Generation Computing* 29.3, pp. 261–276. DOI: 10.1007/s00354-011-0302-1.

Lin, Ching Yung et al. (2009). "SmallBlue: Social network analysis for expertise search and collective intelligence". In: *Proceedings - International Conference on Data Engineering*, pp. 1483–1486. DOI: 10.1109/ICDE.2009.140.

Lykourentzou, Ioanna, Robert E. Kraut, and Steven P. Dow (2017). "Team Dating Leads to Better Online Ad Hoc Collaborations". In: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing - CSCW '17*. New York, New York, USA: ACM Press, pp. 2330–2343. DOI: 10.1145/2998181.2998322.

Mahmoud, Tarek (2011). "Adaptive control scheme based on the least squares support vector machine network". In: *International Journal of Applied Mathematics and Computer Science* 21.4, pp. 685–696. DOI: 10.2478/v10006-011-0054-6.

Malinowski, Jochen, Tim Weitzel, and Tobias Keim (2008). "Decision support for team staffing: An automated relational recommendation approach". In: *Decision Support Systems* 45.3, pp. 429–447. DOI: 10.1016/j.dss.2007.05.005.

Masthoff, Judith (2015). "Group Recommender Systems: Aggregation, Satisfaction and Group Attributes". In: *Recommender Systems Handbook*. Boston, MA: Springer US, pp. 743–776. DOI: 10.1007/978-1-4899-7637-6_22.

McCarthy, Joseph F. and Theodore D. Anagnost (1998). "MusicFX: An Arbiter of Group Preferences for Computer Supported Collaborative Workouts". In: *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'98)*. DOI: 10.1145/289444.289511.

Meena, Ritu and Kamal K. Bharadwaj (2013). "Group Recommender System Based on Rank Aggregation – An Evolutionary Approach". In: *Mining Intelligence and Knowledge Exploration*, pp. 663–676. DOI: 10.1007/978-3-319-03844-5_65.

Minto, Shawn and Gail C. Murphy (2007). "Recommending Emergent Teams". In: *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*. IEEE, pp. 5–5. DOI: 10.1109/MSR.2007.27.

Mohanty, Ramakanta, Vadlamani Ravi, and Manas Ranjan Patra (2010). "The application of intelligent and soft-computing techniques to software engineering problems: a review". In: *International Journal of Information and Decision Sciences* 2.3, p. 233. DOI: 10.1504/IJIDS.2010.033450.

Mooney, Raymond J and Loriene Roy (2000). "Content-based book recommending using learning for text categorization". In: *Proceedings of the fifth ACM conference on Digital libraries*. ACM, pp. 195–204.

Nair, Ranjit, Milind Tambe, and Stacy Marsella (2002). "Team formation for reformation". In: *Proceedings of the AAAI spring symposium on intelligent distributed and embedded systems*, pp. 52–56.

Nowicki, Robert (2010). "On classification with missing data using rough-neuro-fuzzy systems". In: *International Journal of Applied Mathematics and Computer Science* 20.1, pp. 55–67. DOI: 10.2478/v10006-010-0004-8.

Ntoutsi, Eirini and Kostas Stefanidis (2016). "Recommendations beyond the ratings matrix". In: *Proceedings of the Workshop on Data-Driven Innovation on the Web - DDI '16*. New York, New York, USA: ACM Press, pp. 1–5. DOI: 10.1145/2911187.2914580.

Ntoutsi, Eirini, Kostas Stefanidis, Kjetil Nørvåg, et al. (2012). "Fast group recommendations by applying user clustering". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7532 LNCS, pp. 126–140. DOI: 10.1007/978-3-642-34002-4_10.

Ntoutsi, Eirini, Kostas Stefanidis, Katharina Rausch, et al. (2014). "'Strength Lies in Differences'". In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management - CIKM '14*. Vol. 9. 5. New York, New York, USA: ACM Press, pp. 729–738. DOI: 10.1145/2661829.2662026.

O'Connor, Mark et al. (2005). "PolyLens: A Recommender System for Groups of Users". In: *ECSCW 2001*. Dordrecht: Kluwer Academic Publishers, pp. 199–218. DOI: 10.1007/0-306-48019-0_11.

Pazzani, Michael J. and Daniel Billsus (2007). "Content-Based Recommendation Systems". In: *The Adaptive Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 325–341. DOI: 10.1007/978-3-540-72079-9_10.

Pazzani, Michael and Daniel Billsus (1997). "Learning and Revising User Profiles: The Identification of Interesting Web Sites". In: *Machine Learning* 27.3, pp. 313–331. DOI: 10.1023/A:1007369909943.

Quijano-Sanchez, Lara et al. (2013). "Social factors in group recommender systems". In: *ACM Transactions on Intelligent Systems and Technology* 4.1, pp. 1–30. DOI: 10.1145/2414425.2414433.

Ratcheva, Violina (2007). "Redefining multidisciplinary team boundaries in resolving heterogeneous knowledge dilemmas". In: *International Journal of Intelligent Enterprise* 1.1, p. 81. DOI: 10.1504/IJIE.2007.013810.

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2011). "Introduction to Recommender Systems Handbook". In: *Recommender Systems Handbook*, pp. 1–35. DOI: 10.1007/978-0-387-85820-3_1.

Rubens, Neil et al. (2015). "Active learning in recommender systems". In: *Recommender Systems Handbook, Second Edition*. Boston, MA: Springer US, pp. 809–846. DOI: 10.1007/978-1-4899-7637-6_24.

Serbos, Dimitris et al. (2017). "Fairness in Package-to-Group Recommendations". In: *Proceedings of the 26th International Conference on World Wide Web - WWW '17*. New York, New York, USA: ACM Press, pp. 371–379. DOI: 10.1145/3038912.3052612.

Shoham, Yoav and Marko Balabanovic (1997). "Fab: Content-based, Collaborative Recommendation". In: *Communications of the ACM* 40.3, pp. 66–72. DOI: https://doi.org/10.1145/245108.245124.

Smyth, Barry and Paul McClave (2001). "Similarity vs. Diversity". In: *Case-Based Reasoning Research and Development*. Ed. by David W. Aha and Ian Watson. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 347–361. DOI: 10.1007/3-540-44593-5_25.

Stefanidis, Kostas, Marina Drosou, and Evaggelia Pitoura (2010). "PerK: personalized keyword search in relational databases through preferences". In: *Proceedings of the 13th International Conference on Extending Database Technology*, pp. 585–596. DOI: http://doi.acm.org/10.1145/1739041.1739111.

Stefanidis, Kostas and Evaggelia Pitoura (2013). "Finding the Right Set of Users: Generalized Constraints for Group Recommendations". In: *CoRR 2013*.

Stevenson, A (2010). *Oxford Dictionary of English*. 3rd edition. Oxford University Press. DOI: 10.1093/acref/9780199571123.001.0001.

Stratigi, Maria, Haridimos Kondylakis, and Kostas Stefanidis (2018). "FairGRecs: Fair group recommendations by exploiting personal health information". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11030 LNCS, pp. 147–155. DOI: 10.1007/978-3-319-98812-2_11.

Strnad, D. and N. Guid (2010). "A fuzzy-genetic decision support system for project team formation". In: *Applied Soft Computing* 10.4, pp. 1178–1187. DOI: 10.1016/j.asoc.2009.08.032.

Toroslu, Ismail H. and Yilmaz Arslanoglu (2007). "Genetic algorithm for the personnel assignment problem with multiple objectives". In: *Information Sciences* 177.3, pp. 787–803. DOI: 10.1016/j.ins.2006.07.032.

Turpin, Andrew H and William Hersh (2001). "Why batch and user evaluations do not give the same results". In: *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '01*. New York, New York, USA: ACM Press, pp. 225–231. DOI: 10.1145/383952.383992.

Val, Elena del et al. (2014). "A Team Formation Tool for Educational Environments". In: *Trends in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection*. Ed. by Javier Bajo Perez et al. Springer International Publishing, pp. 173–181. DOI: 10.1007/978-3-319-07476-4_21.

Valencia-García, R. et al. (2010). "Exploitation of social semantic technology for software development team configuration". In: *IET Software* 4.6, p. 373. DOI: 10.1049/iet-sen.2010.0043.

Wang, Juite and Yung-I Lin (2003). "A fuzzy multicriteria group decision making approach to select configuration items for software development". In: *Fuzzy Sets and Systems* 134.3, pp. 343–363. DOI: 10.1016/S0165-0114(02)00283-X.

Wang, Wanyuan et al. (2017). "Toward Efficient Team Formation for Crowdsourcing in Noncooperative Social Networks". In: *IEEE Transactions on Cybernetics* 47.12, pp. 4208–4222. DOI: 10.1109/TCYB.2016.2602498.

Wang, Xinyu, Zhou Zhao, and Wilfred Ng (2015). "A Comparative Study of Team Formation in Social Networks". In: *Database Systems for Advanced Applications*. Ed. by Matthias Renz et al. Cham: Springer International Publishing, pp. 389–404. DOI: 10.1007/978-3-319-18120-2_23.

Yao, Sirui and Bert Huang (2017). "Beyond Parity: Fairness Objectives for Collaborative Filtering". In: *Advances in Neural Information Processing Systems 30*. Ed. by I Guyon et al. Curran Associates, Inc., pp. 2921–2930.

Yilmaz, Murat et al. (2015). "A Machine-Based Personality Oriented Team Recommender for Software Development Organizations". In: *Systems, Software and Services Process Improvement*. Ed. by Rory V. O'Connor et al. Vol. 543. Communications in Computer and Information Science. Cham: Springer International Publishing, pp. 75–86. DOI: 10.1007/978-3-319-24647-5_7.

Yu, Zhiwen et al. (2006). "TV program recommendation for multiple viewers based on user profile merging". In: *User Modeling and User-Adapted Interaction*. DOI: 10.1007/s11257-006-9005-6.

Zhang, Mi and Neil Hurley (2008). "Avoiding monotony". In: *Proceedings of the 2008 ACM conference on Recommender systems - RecSys '08*. New York, New York, USA: ACM Press, p. 123. DOI: 10.1145/1454008.1454030.

Zhong, Ning and Andrzej Skowron (2001). "A rough set-based knowledge discovery process". In: *International Journal of Applied Mathematics and Computer Science* 11, pp. 603–619.