The Pennsylvania State University
The Graduate School
College of Engineering

RECOMMENDATION SERVICES FOR OPEN SOURCE SOFTWARE COMMUNITY

A Dissertation in
Computer Science and Engineering
by
Mohammad Yacoub Allaho

© 2014 Mohammad Yacoub Allaho

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2014
The dissertation of Mohammad Yacoub Allaho was reviewed and approved* by the following:

Wang-Chien Lee  
Associate Professor of Computer Science and Engineering  
Dissertation Advisor, Chair of Committee  

C. Lee Giles  
Graduate School Professor of Computer Science and Engineering  
Professor of Information Sciences and Technology  

Piotr Berman  
Associate Professor of Computer Science and Engineering  

Zhenhui Li  
Assistant Professor of Information Sciences and Technology  

Lee Coraor  
Associate Professor of Computer Science and Engineering  
Graduate Program Chair  

*Signatures are on file in the Graduate School.
Abstract

Large collaborative online communities have appeared and grown rapidly in the presence of Web 2.0 technology, witnessed by the massive success of Open Source Software (OSS) projects such as the Apache projects and GNU/Linux. As members of OSS projects are mostly volunteers, their motivations usually involve personal goals/interests, such as sharpening their programming skills and gaining experience, following fellow peers, networking with the OSS community members, or simply supporting free open software projects. In contrast to commercial software with dedicated responsibility to software engineers, the amount of participation and commitment by the volunteering developers are crucial factors for the success of OSS projects. In fact, the failure of projects to attract active developers may prove to be the single important reason for the death of OSS projects. Therefore, recommendation services are crucially needed to automate the process of forming teams, finding the appropriate experts for a given task and suggesting open projects to potential developers.

Given the unique structure of the OSS community, the continuous growing number of OSS projects available online (over 4 millions) and the number of mixed-skilled developers (over 1 million), recommendation services become extremely challenging. In order to build successful recommendation services, we first investigate and study the OSS community structure and the social factors that affect the amount of participation and commitment in OSS projects. In particular, we conduct an extensive social link analysis on the Social Network (SN) and Affiliation Network (AN) of OSS developers. Moreover, we conduct both global and local networks analysis, where a global network analysis considers a developer’s connectivity in the whole OSS community network, whereas a local network analysis considers a developer’s connectivity within a team network that is affiliated to a project. The analysis demonstrates evident influence of the social factors on the developers’ overall participation and productivity. We further investigate the different types of affiliations that could associate developers into collaboration such as co-participating in the same project, belonging to the same company or institution or even being in the same city. We, also, study the effect of some project features on the developers’ selection of projects. The analysis results are used to build highly accurate recommendation services for OSS community.

We introduce three recommendation services for OSS community members. First, we consider the expert recommendation problem for members of the community. Given a query developer and a task in hand, the expert recommender service introduces experts to facilitate collaboration between newbie developers and experts. In this particular problem, we solve the issue of experts responsiveness to queries, where experts, usually, tend to ignore the requests to participate.
Another issue is data sparsity problem since most new members have few data history which is a challenge to recommender systems. To solve these challenges we propose a recommender system that uses the Degree of Knowledge and Social Relative Importance aspects in recommending experts who are knowledgeable on the skills required and highly responsive to queries.

Second, we consider the project recommendation problem for members of the community. Project recommendation is a unique problem since it involves recommending an item that requires a continuous action and commitment. This is not the case for consuming items such as books or software. We propose several models that use two main aspects that attract developers into joining a project. The first aspect is the project content and the second aspect is the peer affiliation influence where we identify three kinds of affiliation ties, namely, project, work and location affiliations.

The third recommender service considers recommending members to collaborate in a project as a team, which is know as the team formation problem in literature. Given a project/task with certain skills required and an upper bound of team size, the recommender service shall find a team that cover all the required skills and their Degree of Acquaintance (DoA) is maximized. We define the DoA in terms of the local clustering coefficient and the strength of social ties between the team members. OSS data analysis shows the importance of the connectivity and social tie strength to the overall productivity of the teams in open projects. This problem is NP-hard. We propose three algorithms to solve the defined problem. The results show the effectiveness of the proposed algorithms to find a well acquainted teams satisfying a given query. We further show that using the DoA, as a social closeness measure, produces more socially close teams than measures like graph diameter and density.

For the analysis and the evaluation of the proposed recommender services, we use datasets collected from Github.com and Openhub.net (previously Ohloh.net), which are two fast-growing and large-scale online OSS communities. The evaluation of the recommender services shows promising results of recommending experts, projects and teams to developers in the community.
# Table of Contents

List of Figures viii

List of Tables x

Acknowledgments xi

Chapter 1 Introduction 1

1.1 The Open Collaborative Projects Phenomenon ........................................... 1
1.2 Open Source Software Community Analysis ................................................ 1
1.3 Discovering and Analyzing the Trends and Behavior of Developers in Open Source Collaborative Projects ................................................................. 2
1.4 Recommending Expert Developers and Increasing their Responsiveness to Queries ................................................................. 2
1.5 Recommending Projects .................................................................................. 3
1.6 Open Source Software Project Staffing and Team Formation .......................... 4

Chapter 2 Analyzing the Social Networks of Contributors in Open Source Software Community 5

2.1 Introduction ....................................................................................................... 5
2.2 Related Works ................................................................................................... 8
2.3 Dataset ............................................................................................................. 8
2.4 Network Structure Analysis ............................................................................. 9
  2.4.1 Basic Measures of the Network Structure .................................................. 9
  2.4.2 Power-law Degree Distribution .................................................................. 10
  2.4.3 Node Degree Mixing Pattern ..................................................................... 10
    2.4.3.1 Degree correlation ............................................................................. 12
    2.4.3.2 Assortativity .................................................................................... 12
2.5 Social Networking Factors and Developer Contribution Analysis ................. 13
  2.5.1 Global Graph Analysis ............................................................................. 14
    2.5.1.1 Indegree & outdegree effect ............................................................. 14
    2.5.1.2 Betweenness and closeness centrality effect ................................... 15
  2.5.2 Local Graph Analysis ............................................................................... 17
    2.5.2.1 Degree of Connectivity Effect: Highly vs. Lowly Connected ........... 19
List of Figures

2.1 A community network example illustrating the difference between global and local network graphs. The OSS community network represents a global network graph, whereas a team network of project P represents a local network graph. Here nodes of the same color belong to one project. Note that it is possible for a node to belong to multiple projects and, therefore, can belong to multiple local networks as illustrated in the graph by the two tainted-color nodes belonging to P' and P''. 7

2.2 Github and Ohloh Out/Indegree Complementary Cumulative Distribution Functions (CCDF) for the OSS SN (x=Out/Indegree). All social networks show properties consistent with power-law networks. 11

2.3 Ohloh (left) and Github (right) degree complementary cumulative distribution functions (CCDF) for the OSS AN (x=degree). The networks does not show a good fit to power-low model. 11

2.4 Ohloh (left) and Github (right) knn functions show that, in OSS SN, high degrees tend to connect to lower degrees which indicate a mixture of degree connectivity. 12

2.5 Ohloh (left) and Github (right) knn functions show that, in OSS AN, nodes tend to connect with other nodes of similar degree. 13

2.6 Indegree and outdegree vs. number of participation for Ohloh ((a) and (b)) and Github ((c) and (d)). 15

2.7 Indegree and outdegree vs. total number of commits for Ohloh ((a) and (b)) and Github ((c) and (d)). 16

2.8 Indegree (left) and outdegree (right) vs. months-work for Ohloh. 16

2.9 Betweenness and closeness centrality vs. participations for Ohloh ((a) and (b)) and Github ((c) and (d)). 18

2.10 Betweenness (left) and closeness (right) centrality vs. months-work for Ohloh. 18

2.11 Effect of degree of connectivity on (a) average Commits/Month and (b) average Months-Work for projects of various ranges of team sizes. 20

2.12 Effect of tie strength on (a) average Commits/Month and (b) average Months-Work for projects of various ranges of team sizes. 21

3.1 Developer Features’ Complementary Cumulative Distribution Function (CCDF). 29

3.2 The Percentage of Hireable Developers for each Followers/Following Interval in (a) and for each Pub.Repos/D.Partc Interval in (b). 30

3.3 The Average Percentage of Developers with the Largest Similar Location (City) and Affiliation According to Each Team Size Interval. 30

3.4 OSS Project Feature Distributions. 33

4.1 (a) SPM model with one latent variable. (b) SCPM model with two latent variables. 42
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>The ER Framework</td>
</tr>
<tr>
<td>4.3</td>
<td>The average precision for various $k$ expert collaborators on combinations of DoK and SRI methods grouped by the SRI method</td>
</tr>
<tr>
<td>4.4</td>
<td>The average precision for various $k$ expert collaborators on multiple h.r. sets</td>
</tr>
<tr>
<td>4.5</td>
<td>The avg. Hits@100 for DoK models on multiple h.r. sets</td>
</tr>
<tr>
<td>4.6</td>
<td>The avg. resp.@20 for DoK models on multiple h.r. sets</td>
</tr>
<tr>
<td>4.7</td>
<td>The average nDCG@20 for DoK models on multiple h.r. sets</td>
</tr>
<tr>
<td>5.1</td>
<td>The Developer Affiliation and Preference Model (DAPM)</td>
</tr>
<tr>
<td>5.2</td>
<td>The result of each metric for developer Alice</td>
</tr>
<tr>
<td>5.3</td>
<td>The average precision for various $k$ retrieved projects on multiple h.r. sets</td>
</tr>
<tr>
<td>5.4</td>
<td>The average recall on multiple h.r. sets</td>
</tr>
<tr>
<td>5.5</td>
<td>The average nDCG@20 on multiple h.r. sets</td>
</tr>
<tr>
<td>5.6</td>
<td>The average suitability@20 on multiple h.r. sets</td>
</tr>
<tr>
<td>6.1</td>
<td>Different graph shapes: (a) line, (b) star, (c) highly connected graph</td>
</tr>
<tr>
<td>6.2</td>
<td>Graph $T$ representing a team formation with normalized edges’ weights and vertices' Clustering Coefficient (CC) values</td>
</tr>
<tr>
<td>6.3</td>
<td>Example of CC property</td>
</tr>
<tr>
<td>6.4</td>
<td>An illustration for Case 1. Figure (a) shows the team before adding the new member, and Figure (b) shows the team after adding the new member</td>
</tr>
<tr>
<td>6.5</td>
<td>An illustration for Case 2. Figure (a) shows the team before adding the new member, and Figure (b) shows the team after adding the new member</td>
</tr>
<tr>
<td>6.6</td>
<td>Graph $G$ representing a collaborative community graph with normalized edges’ weights and skills’ labels</td>
</tr>
<tr>
<td>6.7</td>
<td>Scalability Performance: (a) Execution Runs vs. Graph Size. (b) Execution Time vs. Graph Size. (c) Execution Runs vs. $\tau$. (d) Execution Time vs. $\tau$. (log scale)</td>
</tr>
<tr>
<td>6.8</td>
<td>Accuracy: (a) TDoA vs. Graph Size (log scale on x-axis). (d) TDoA vs. $\tau$.</td>
</tr>
<tr>
<td>6.9</td>
<td>The Team DoA comparison for STA, NFA, Diameter in (a), Density with changing $\tau$ in (b), and Density with changing $l$ in (c)</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Section</th>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Collected Data Statistics.</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>OSS Social Network Structure Measurements.</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Power-law coefficient estimates ((\alpha)) and corresponding goodness-of-fit metric ((D)).</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>Assortativity coefficients.</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>Collected Data Statistics.</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Developer Numeric Feature Statistics.</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>Developer Social Ties and Development Activity Correlations.</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>Programming Languages Statistics.</td>
<td>31</td>
</tr>
<tr>
<td>3.5</td>
<td>Project Numeric Feature Statistics.</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>Project Numeric Feature Correlations.</td>
<td>32</td>
</tr>
<tr>
<td>3.7</td>
<td>Project Feature Ranking.</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Collected Data Statistics.</td>
<td>49</td>
</tr>
<tr>
<td>4.2</td>
<td>Top-10 recommended experts for query (Q) comparing every DoK model.</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Top-10 recommended experts for query (Q) comparing every SRI method.</td>
<td>53</td>
</tr>
<tr>
<td>4.4</td>
<td>ER Evaluation on Hits, resp. and nDCG.</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>Project Attribute Nominal Values.</td>
<td>65</td>
</tr>
<tr>
<td>5.2</td>
<td>Collected Data Statistics.</td>
<td>71</td>
</tr>
<tr>
<td>5.3</td>
<td>Top-10 projects recommended for (Alice).</td>
<td>73</td>
</tr>
<tr>
<td>6.1</td>
<td>The developer-skill Matrix (A) for graph (G).</td>
<td>88</td>
</tr>
<tr>
<td>6.2</td>
<td>Snap shot of several feasible team formations for query (Q_{ex}).</td>
<td>88</td>
</tr>
</tbody>
</table>
Acknowledgments

I would like to acknowledge Dr. W.C. Lee for his sage advice, insightful criticisms and patient encouragement that aided my journey in writing and completing this dissertation. Also, I would like to acknowledge and thank my dissertation committee members, Dr. C. Lee Giles, Dr. P. Berman and Dr. Z. Li for their support and remarkable comments which greatly impacted the quality of the dissertation. Also, I would like to acknowledge Dr. D.N. Yang for his contribution in one of the works in this dissertation. Finally, I would like to acknowledge Kuwait University for providing me the financial support, as well as, facilitating numerous resources to aid scholarship students overcome academic and any daily life challenges.
Dedication

All thanks and praise be to Allah the cherisher and sustainer of the worlds, who bestowed me with life and health to complete this work.

This dissertation is dedicated to my father, mother and family for their unconditional love and support. It is also dedicated to my friends and colleagues for their support and encouragement. Finally, it is dedicated to every knowledge seeker who cherish research for the good of humanity.
Chapter 1

Introduction

1.1 The Open Collaborative Projects Phenomenon

Large collaborative online communities have become a phenomenon in the presence of Web 2.0 technology, witnessed by the massive success of Open Source Software (OSS) projects such as the Apache projects and GNU/Linux. As members of OSS projects are mostly volunteers [1], they usually work out of personal goals/interests, e.g., practicing existing skills and gaining experience, following fellow peers, networking with the OSS community members, or simply supporting free open software projects [2]. Consequently, the amount of participation and commitment by the volunteering developers are crucial factors in the cause of OSS projects success [3]. Therefore, providing recommendation services to OSS community is of great benefit. We believe that the OSS community is unique since it involves volunteers who dedicate their time and effort in contributing to open projects, which is unprecedented phenomenon. Factors like willingness to participate, preferences and social connectivity should all be considered in a recommender system. Also, with the continuous growing number of OSS projects available online (over 4 millions) and the number of mixed-skilled developers (over 1 million), recommendation services become extremely challenging. Moreover, considering new members to the OSS community in the recommender services is very challenging since new members usually have sparse data history. The following sections briefly introduce our work on the recommendation services for OSS community. The first step is to analyze thoroughly the OSS community structure and study the factors that affect developers’ participation and commitment as described next.

1.2 Open Source Software Community Analysis

In Chapter 2, we conduct an extensive statistical analysis on the social networks of contributors in OSS communities using datasets collected from two most fast-growing OSS social interaction sites, Github.com and Ohloh.net. Our goal is to analyze the connectivity structure of the social
networks of contributors and to investigate the effect of the different social ties structures on developers’ overall productivity to OSS projects. We, first, analyze the general structure of the social networks, e.g., graph distances and the degree distribution of the social networks. Our social network structure analysis confirms a power-law degree distribution and small-world network characteristics. However, the degree mixing pattern shows that high degree nodes tend to connect more with low degree nodes suggesting a collaboration between experts and newbie developers which makes it possible to improve this collaboration by an expert recommendation service. We further conduct the same analysis on affiliation networks and find that contributors tend to participate in projects of similar team sizes. This indicates the existence of certain project’s status features that can influence the developers’ selection of open projects. Second, we study the correlation between various social factors (e.g., closeness and betweenness centrality, clustering coefficient and tie strength) and the productivity of the contributors in terms of the amount of contribution and commitment to OSS projects. The analysis is conducted under the contexts of global and local networks, where a global network analysis considers a developer’s connectivity in the whole OSS community network, whereas a local network analysis considers a developer’s connectivity within a team network that is affiliated to a project. The analysis demonstrates evident influence of the social factors on the developers’ overall productivity.

1.3 Discovering and Analyzing the Trends and Behavior of Developers in Open Source Collaborative Projects

In Chapter 3, we conduct an extensive analysis on the developers of Open Source Software (OSS) projects. Our goal is to discover trends that govern the developers’ behavior in contributing to OSS projects. To achieve our goal, we define and analyze a set of developer and OSS project features. Moreover, we study the behavior of the developers on selecting OSS projects to participate in by analyzing the project features that dictate the developers’ selection. In addition, we study the difference between developers who seek a job and who do not seek a job in developing social ties. We, also, analyze the developers’ affiliation (e.g., corporate, university, institute, etc.) and location (e.g., city) statistics. It is found that a significant ratio of developers share the same affiliation and location in a team for a project that is being developed by remote collaborators. We use a dataset collected from Github.com, which is one of the most fast-growing and large-scale online OSS community. This study is substantial for recommender systems targeting the OSS community.

1.4 Recommending Expert Developers and Increasing their Responsiveness to Queries

In Chapter 4, we consider the experts recommendation problem for open collaborative projects in large-scale Open Source Software (OSS) communities. In large-scale online community, rec-
ommending expert collaborators to a project coordinator or lead developer has two prominent challenges: (i) the “cold shoulder” problem, which is the lack of interest from the experts to collaborate and share their skills, and (ii) the “cold start” problem, which is an issue with community members who has scarce data history. In this work, we consider the Degree of Knowledge (DoK) which imposes the knowledge of the skills factor, and the Social Relative Importance (SRI) which imposes the social distance factor to tackle the aforementioned challenges. We propose four DoK models and integrate them with three SRI methods under our proposed Expert Ranking (ER) framework to rank the candidate expert collaborators based on their likelihood of collaborating in response to a query formulated by the social network of a query initiator and certain required skills to a project/task. We evaluate our proposal using a dataset collected from Github.com. In addition, we test the models under different data scarcity levels. The experiment shows promising results of recommending expert collaborators that are highly responsive and motivated to contribute their skills.

1.5 Recommending Projects

In Chapter 5, we consider the problem of recommending open projects to developers in large-scale Open Source Software (OSS) communities. There exist thousands of OSS projects online, however, many fail due to lack of participation from developers. In an effort to increase the participation in OSS projects, we propose several recommendation models to direct developers to projects of their interest. There exist two main challenges. First, identifying the factors that developers act upon on selecting projects to join. Second, the data scarcity problem for new community members. Our solutions consider two main aspects in the recommendation, (i) the content of projects and (ii) the peer influence. The content of projects is based on project’s attributes that describe the status and activity levels (e.g., team size, popularity, complexity, skills required) of a project rather than just a software field and purpose description. The type of attributes are called dynamic attributes since they change with time, and they are more suitable for recommending items with activity indulging nature such as OSS development. The second aspect considers the social factor among developers and the fact that developers tend to follow collaborators of similar affiliations. We consider three types of affiliation similarities in the recommender models, namely, project co-participation, work and location affiliations. Based on these aforementioned factors, we propose four memory-based models using content and collaborative filtering methods and one probabilistic model. The probabilistic model combines the two recommender aspects in one model. We evaluate our proposal using a longitudinal dataset collected from Github.com over a span of one year. In addition, we test the models under different data scarcity levels. The experiment shows promising results of recommending projects to developers who tend to make real participations to the recommended projects.
1.6 Open Source Software Project Staffing and Team Formation

In Chapter 6, we consider the team formation problem in open collaborative projects existing in large community setting such as the OSS community. Given a query specifying a set of required skills for an open project and an upper bound of team size, the goal is to find a team that maximizes the Degree of Acquaintance (DoA) and covers all the required skills in the query. We define the DoA in terms of the team graph connectivity and edge weights, corresponding to the local Clustering Coefficient for each team member and the strength of social ties between the team members, respectively. Statistical analysis on historical data shows the importance of the connectivity and social tie strength to the overall productivity of the teams in open projects. We show that the problem defined is NP-hard, and present three algorithms, namely, PSTA, STA and NFA, to solve the problem. We experiment the algorithms on a dataset from the OSS community. The results show the effectiveness of the proposed algorithms to find a well acquainted teams satisfying a given query. We further show that using the DoA, as a social closeness measure, produces more socially close teams than measures like graph diameter and density.
Chapter 2

Analyzing the Social Networks of Contributors in Open Source Software Community

2.1 Introduction

The work in this chapter has been published in the proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), and in Lecture Notes in Social Networks (LNSN) series as a chapter in the book titled “Applications of Social Media and Social Network Analysis” by Springer publisher.

Large collaborative online communities have appeared and grown rapidly in the presence of Web 2.0 technology, witnessed by the massive success of Open Source Software (OSS) projects such as the Apache projects and GNU/Linux. As members of OSS projects are usually volunteers [1], their motivations usually involve personal goals/interests, e.g., sharpening their programming skills and gaining experience, following fellow peers, networking with the OSS community members, or simply supporting free open software projects [2]. In contrast to commercial software with dedicated responsibility to software engineers, the amount of participation and commitment by the volunteering developers are crucial factors for the success of OSS projects [3].

To investigate and better understand the success factors of OSS, several research studies have been reported in the literature projects [4, 5, 6, 7, 8, 9]. However, these studies are mainly focusing on success factors related to the user-developer interaction (e.g., user ratings and bug reporting), project related factors (e.g., programming language used, type of operating system and type of open source license). Recently, the importance of social factors in OSS community has drawn a growing amount of attention. Studies in [10] and [11] suggest that the rapport created from collaboration among individuals in open collaborative communities indeed plays a crucial role
leading to successes of OSS projects. However, the potential correlation between the productivity of OSS developers and the social ties among those developers in the community has not been quantitatively studied before. In this work, we aim to answer these research questions: What are the general structure and features of the OSS social network? What are the social ties patterns in OSS community network that affect the volunteering developers’ amount of contribution and commitment? Moreover, does the connectivity structure and strength of social ties in a team affect the amount of contribution and commitment in OSS projects?

In order to answer these critical research questions, we first analyze the general structure of the OSS social network and then investigate the correlation between various social ties structures and the amount of contribution and commitment of volunteering developers to OSS projects. In particular, we conduct an extensive social link analysis on the Social Network (SN) and Affiliation Network (AN) of OSS developers. We define the OSS SN as a directed graph where nodes represent developers and links represent a following/follower relationship just like the Twitter network, while defining the OSS AN as an undirected graph where nodes represent developers and links represent co-authorship in OSS projects. The SN and AN represent general but different social ties among the contributors of OSS communities, i.e., different kinds of link representation may draw different conclusions about the studied community. We collected our data from two most fast-growing OSS collaboration sites, Github.com and Ohloh.net. These two sites are unique since they provide many social interaction tools for developers, including the ability to form a network of followings/followers relationship which gives explicit information of acquaintances.

We emphasize here that most previous studies on the SNs of OSS typically construct the SN based on communication links which results in an implicit SN (i.e., the links may not represent true acquaintances for various reasons). Aiming to better understand the characteristics of OSS SNs, we perform a series of analysis on the social network structure, including measuring SNs distances (i.e., diameter and average path length), analyzing degree distributions, testing power-law model and examining degree mixing patterns between nodes (developers) in the OSS SN and AN.

Our social network structure analysis confirms a small-world characteristics for the OSS SNs. Also, the analysis shows that the degree distribution of the SNs follows a power law distribution that fits the scale-free network model. However, the degree distribution of the ANs does not fit the scale-free model which implies different characteristics of the two networks. Moreover, the degree mixing pattern analysis for the SNs shows that high degree nodes tend to connect more with low degree nodes suggesting collaboration between experts and newbie developers. On the other hand, the degree mixing pattern for the ANs shows an opposite trend.

In addition to the analysis of network structures, we also investigate the factors of social ties under the contexts of global and local network graphs to explore the social factors that affect the developers’ amount of contribution and commitment. A global network analysis studies the connectivity among individuals in the whole OSS community, whereas a local network analysis studies the connectivity among members in a team working on a particular project $P$, where a team is a subset of the community as illustrated in Figure 2.1. For the global network analysis, we
consider the betweenness and closeness centrality and for the local network analysis we consider the clustering patterns and ties strength. Note that nodes’ degree analysis is considered a global analysis if it considers all nodes in the community and a local analysis if it considers only team nodes. Both global and local network analysis demonstrate evident influence of the social factors on the developers’ overall productivity.

This work constitutes the first large-scale study that analyzes the social network of volunteering developers that is based on explicit acquaintances. Moreover, our work is the first to focus on investigating the correlations between social factors and developers’ productivity. We, further, quantify the developers’ productivity from an abstract meaning into measurable metrics in order to comprehend the amount of contribution and commitment each developer exerts in a project. In addition, we study the social factors in a community context and in a team level context to capture any social factor that might be related to either context. Our findings may then pave the road for future recommendation systems related to online open collaborative projects.

The rest of the chapter is organized as follows. Section 2.2 introduces several related works. Section 2.3 introduces the dataset collected and used in this study. Section 2.4 shows the analysis of network structure. The analysis includes network structure measures, degree distribution and degree mixing patterns analysis. Section 2.5 investigates the correlation between the various social ties’ structures and developers’ overall productivity globally, in the whole OSS community, and locally, in a team network. Section 2.6 discusses some threats to validity issues, and finally, Section 2.7 concludes the chapter.
2.2 Related Works

There exist several works that analyze the OSS social network. Both [17] and [18] study the structure of a collaborator communication network extracted from the e-mail network and find that small teams have a centralized structure, whereas larger teams have a modular structure. However, they do not study the correlation between network structure and developers productivity. In contrast to the above works, our datasets include social networks based on explicit acquaintances, which facilitate us to investigate the success factors of OSS projects. In another work, Subramaniam et al. [4] study other factors that may affect the development activity, such as developer interest, user interest, project activity (i.e., bug reports and user comments), project status (i.e., new project, development phase, maintenance phase) and project attributes (i.e., OSS license, operating system, programming language). On the contrary, we focus on the social tie factors of the contributors in the OSS projects. Moreover, Casaló et al. [5] collect information via questionnaires and study the correlation between developers commitment to their OSS project and the reputation of those projects. They conclude that OSS reputation have an indirect effect on the commitment of collaborators. In contrast to questionnaires, the contribution and commitment information in our study are collected from control version repositories that monitor the commits log for each developer.\footnote{The commits log is collected by Ohloh.net and Github.com.}

2.3 Dataset

We collect our data from two fast-growing OSS social interaction sites, Github.com and Ohloh.net. These two sites provide a variety of information regarding the social relationship and contributions of their volunteering developers. Unlike other OSS collaboration sites (e.g., sourceforge.net, freshmeat.net) Github and Ohloh provide facilities for contributors to explicitly recognize each other through a follow link in Github and Give kudo in Ohloh which creates a directed social network of contributors/developers. We collected data of over a million contributor and over two millions OSS project. The data were collected via APIs provided by Github and Ohloh.

The difference between Github and Ohloh is that Github hosts the actual project artifacts (i.e., source code, documentation) whereas Ohloh provides links to third party repositories. Ohloh provides more statistics about the OSS project such as program language statistics, skills possessed by the developers, commits ratio and the number of active months spent by developers on a project. On the other hand, Github provides more facilities for actual development process such as the ability to fork a project in a separate repository and report updates to the main trunk of a project. Since Github, actually, hosts the projects’ files, its number of contributors and projects are much more than Ohloh. We include two real datasets in our study to further confirm our observations and conclusions.

Table 2.1 summarizes some general statistics of the collected data sets from Github and Ohloh. In Table 2.1, Contributions is the total number of distinct contributions in all OSS
Table 2.1. Collected Data Statistics.

<table>
<thead>
<tr>
<th></th>
<th>Ohloh</th>
<th>Github</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contributions</td>
<td>671,835</td>
<td>9,268,644</td>
</tr>
<tr>
<td>Contributors</td>
<td>297,046</td>
<td>1,034,996</td>
</tr>
<tr>
<td>Contributors with account</td>
<td>14,958</td>
<td>652,040</td>
</tr>
<tr>
<td>SN Edges</td>
<td>36,205</td>
<td>1,788,803</td>
</tr>
<tr>
<td>AN Edges</td>
<td>1,096,208</td>
<td>14,474,929</td>
</tr>
<tr>
<td>Projects</td>
<td>456,811</td>
<td>2,332,749</td>
</tr>
</tbody>
</table>

Table 2.2. OSS Social Network Structure Measurements.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Github</td>
<td>22</td>
<td>6.25</td>
<td>4.31</td>
<td>0.112</td>
<td>0.19</td>
</tr>
<tr>
<td>Ohloh</td>
<td>23</td>
<td>7.11</td>
<td>3.21</td>
<td>0.25</td>
<td>0.15</td>
</tr>
</tbody>
</table>

projects, whereas contributors, in the second row, is the distinct number of contributors in each community. For example, if a community has two contributors A and B, and contributor A participates in 2 projects while contributor B participates in 3 projects, then this community has 2 contributors and total of 5 contributions. Furthermore, the contributors with account are the contributors that are registered in the site with a profile information.

2.4 Network Structure Analysis

In this section we study the general graph distance measures, the node degree distribution and the mixing pattern of degrees in both SN and AN.

2.4.1 Basic Measures of the Network Structure

We first measure the SNs graph distances to comprehend the general structure of the networks. Table 2.2 shows the diameter and average path length of Github and Ohloh SNs. Recall that the graph diameter is the longest distance of the shortest distances between every pair of nodes in the network, while the average path length is the average length of those shortest paths. The average path length found in both Github and Ohloh confirms the result by Stanley Milgram [12] which states that any two people are on average separated by six intermediate connections, known as the “six degrees of separation” in the “small world” phenomenon. Also, the diameter of the OSS SN is small compared to non-social networks (e.g., the Web network has a diameter of 905 [13]), which is a feature of most social graphs. Additionally, Table 2.2 includes the average degree, clustering coefficient (CC) and closeness centrality of the OSS SNs. Besides, the average betweenness centrality for Github and Ohloh are $3.44 \times 10^{-6}$ and $1.58 \times 10^{-4}$, respectively (refer to Section 2.5 for measurement definitions). The structure measures of these two SNs show great similarity. In the next section, we test the OSS SN and AN against the scale-free network model.
2.4.2 Power-law Degree Distribution

Node degree analysis is essential to reveal the connectivity trends between nodes (developers) in a graph. Thus, we first investigate whether the OSS SN and AN follow a scale-free network model or not. A scale-free network has a degree distribution following a power-law model where the exponent $\alpha$ is typically $2 < \alpha < 3$ in most cases. Figure 2.2 shows the outdegree and indegree Complementary Cumulative Distribution Function (CCDF) of the OSS SN for Github and Ohloh, respectively. Basically, the plots show the probability of any node having a degree greater than or equals the given degree $x$. Both outdegree and indegree distributions are consistent with the fitted power-low distribution model. To confirm this result, Table 2.3 shows the maximum likelihood estimate of the scaling exponent $\alpha$ and the Kolmogorov-Smirnov goodness-of-fit statistic $D$, where a value close to zero indicates that the data is closely estimated by the given model. Both networks follow a power-law distribution and confirm properties of scale-free networks where few nodes have high degrees and the majority have low degrees.

On the other hand, the OSS AN does not show a good fit to the scale-free model. Figure 2.3 shows a sharp drop on the CCDF on high degrees showing that at some point of high degree the probability of a node having higher degree drops dramatically indicating big gaps of degrees among large degree nodes. This is because the degree of a node in affiliation networks may increase in high order. For example, once a node $x$ joins a project all the nodes participating in that project become connected to that node causing a large increase in the node degree of $x$. The maximum likelihood estimate of the scaling exponent $\alpha$ for Ohloh AN is 1.51 and 1.72 for Github AN.

2.4.3 Node Degree Mixing Pattern

In this section, we investigate the mixing pattern of degrees between connected nodes. In other words, we would like to know whether similar degree nodes tend to connect to each other or is it a mixture and how much is that mixture? This study is important since it reveals important insights to the structure of the network. Connectivity between similar degree nodes indicates a centralized network where high-degree nodes are connected together. This kind of network is more susceptible to diffuse information fast. On the other hand, networks where high-degree nodes tend to connect to low-degree nodes show the opposite behavior but it is more robust than centralized networks since removing a high degree node will not dramatically affect the network connectivity.

### Table 2.3. Power-law coefficient estimates ($\alpha$) and corresponding goodness-of-fit metric ($D$).

<table>
<thead>
<tr>
<th></th>
<th>Indegree</th>
<th></th>
<th>Outdegree</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$D$</td>
<td>$\alpha$</td>
<td>$D$</td>
</tr>
<tr>
<td>Github</td>
<td>2.31</td>
<td>0.0258</td>
<td>2.82</td>
<td>0.0433</td>
</tr>
<tr>
<td>Ohloh</td>
<td>2.77</td>
<td>0.0281</td>
<td>3.04</td>
<td>0.0231</td>
</tr>
</tbody>
</table>
Figure 2.2. Github and Ohloh Out/Indegree Complementary Cumulative Distribution Functions (CCDF) for the OSS SN ($x=$Out/Indegree). All social networks show properties consistent with power-law networks.

Figure 2.3. Ohloh (left) and Github (right) degree complementary cumulative distribution functions (CCDF) for the OSS AN ($x=$degree). The networks does not show a good fit to power-law model.
2.4.3.1 Degree correlation

First we study the Joint Degree Distribution (JDD), also referred to as the mixing pattern \[14\] and the 2K distribution \[15\]. JDD gives the probability of a node of degree \( k_1 \) to be connected with a node of degree \( k_2 \). JDD is approximated by the degree correlation function \( k_{nn} \). For an undirected graph, \( k_{nn} \) is a mapping between every degree and the average degree of all neighboring nodes (connected) to nodes of that degree. On the other hand, \( k_{nn} \) for a directed graph is a mapping between every outdegree and the average indegree of all neighboring nodes (connected) to nodes of that outdegree. Figure 2.4 shows the OSS SN \( k_{nn} \) plots. Both Ohloh (Figure 2.4(a)) and Github (Figure 2.4(b)) show that high-degree nodes tend to connect to lower-degree nodes (note that the slope of the \( k_{nn} \) distribution is negative), which indicates that OSS SN have a mixture of degree connectivity. This mixing between different degree nodes shows that there are interactions between experts and newbie developers which is very positive in an open collaborative environment.

On the other hand, the OSS AN \( k_{nn} \) distribution shows an opposite trend where similar degree nodes tend to connect with each other as shown in Figure 2.5 for both Ohloh and Github. We note that if every node had the same number of project affiliations as its neighbors, then Figure 2.5 would be symmetric on the diagonal. Since it is not, then this result shows that developers who participate in large team size projects tend to participate in other large team size projects. Also, the same trend is true for developers who participate in small or medium team size projects. In the next section, we further investigate the mixing pattern by introducing the assortativity coefficient measure to quantify the degree of mixing.

2.4.3.2 Assortativity

Assortativity is a measure of the likelihood for nodes to connect to other nodes with similar degrees. The assortativity coefficient \( r \) ranges between -1 and 1. Positive assortativity coefficient means that nodes tend to connect to other nodes with similar degree, while negative assortativity
cohbn means that nodes tend to connect to other nodes with different degrees. Table 2.4 shows the assortativity coefficient for OSS SN and AN for both Ohloh and Github. The OSS SN negative assortativity coefficients indicate that high-degree nodes tend to connect to low-degree nodes, while the OSS AN positive assortativity coefficients indicate that similar degree nodes tend to connect together, which confirms the observation of degree correlation discussed earlier. The small negative value of the assortativity coefficient for the OSS SN indicates that the degree mixture is not high between nodes.

Figure 2.5. Ohloh (left) and Github (right) $k_{nn}$ functions show that, in OSS AN, nodes tend to connect with other nodes of similar degree.

<table>
<thead>
<tr>
<th></th>
<th>Ohloh</th>
<th>Github</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social Net.</td>
<td>-0.0177</td>
<td>-0.0465</td>
</tr>
<tr>
<td>Affil. Net.</td>
<td>0.5811</td>
<td>0.4817</td>
</tr>
</tbody>
</table>

2.5 Social Networking Factors and Developer Contribution Analysis

To discover the social factors that affect the contributors’ participation and contribution in OSS projects, we conduct an extensive analysis to investigate the correlations between various social connectivity structures and the amount of contribution exerted by the contributors. We first conduct a global graph analysis, where we analyze the developers’ connectivity in the whole community graph. In particular, we study the effect of node (developer) indegree, outdegree, betweenness and closeness centrality on the developer’s contribution and commitment. Since the social network of contributors in OSS community is based on teams then we, further, conduct a local graph analysis, where we analyze the developers’ connectivity in a team level graph. In particular, we study the effect of nodes’ different tie patterns and tie strength on developers’ amount of contribution and commitment. In the next two sections, we elaborate in more details every network analysis and every social factor and contribution measure.
2.5.1 Global Graph Analysis

We analyze the correlation between node indegree/outdegree, betweenness and closeness centrality against developers’ contribution and commitment. The amount of contribution for each developer is estimated by two metrics: (i) the number of participations in distinct OSS projects, and (ii) the total number of commits to all OSS projects, where a commit is an update submitted to a project by a developer. Furthermore, a developer commitment is estimated by the number of months where a developer submitted at least one commit denoted by $\text{months-work}$. This metric is collected from Ohloh only since Github does not provide the number of active months for developers. We note that $\text{months-work}$ is not simply the number of months between first and last commit for a developer. In particular, the number of idle months are not counted. Note that this information is hard to monitor and collect for a large number of developers, unless it is provided by the social platform provider.

2.5.1.1 Indegree & outdegree effect

Figure 2.6 shows the in/outdegree correlation with the number of participations for Ohloh and Github data. In particular, for each degree we take the average participation of nodes with that degree. Both indegree and outdegree show positive correlation with the number of participations indicating that developers with high indegree or outdegree have more tendency to participate in more OSS projects.

Figure 2.7 shows the in/outdegree correlation with the total number of commits in OSS projects for Ohloh and Github datasets. In particular, for each degree we take the average commits of nodes with that degree. Similar to participations, both indegree and outdegree show positive correlation with the total commits indicating that developers with high indegree or outdegree have more tendency to commit more work in OSS projects.

In Figures 2.6 and 2.7, we notice a bulky head for the correlation points distribution, which suggests a wider range of participation and total commits values for nodes with high in/outdegree. This is more obvious in Github dataset than Ohloh dataset since the amount of data in Github is much more than the amount of data in Ohloh. Since we are aggregating nodes based on in/outdegree and the node degree follows a power-low distribution, where low degree nodes are much more than high degree nodes, then the data points aggregated for low-degree nodes are much more than high-degree nodes which makes the results for low degree nodes more consistent than high degree nodes.

Finally, Figure 2.8 shows the in/outdegree correlation with the total number of months-work in OSS projects for Ohloh data. For each degree we take the average months-work of nodes with that degree. Similar to participations and total commits, both indegree and outdegree show positive correlation with the total months-work indicating that developers with high indegree or outdegree have more tendency to stay longer time committing to OSS projects.
2.5.1.2 Betweenness and closeness centrality effect

In this section we analyze the effect of betweenness and closeness centrality on a developer amount of contribution and commitment. Recall that betweenness centrality for a node in a graph measures how many times this node acts as a bridge along the shortest path between two other nodes, and closeness centrality is the inverse of the sum of all distances between that node and all reachable nodes in the graph. In other words, a node with high closeness centrality has short path length, on average, to every other node and therefore can spread information fast. Therefore, a node with high betweenness centrality acts as a gate keeper, whereas a node with high closeness centrality acts as a hub in a social network.

Figure 2.9 shows how the number of participations correlating to the betweenness and closeness centrality of nodes in the Ohloh and Github datasets. In particular, for each participation amount we take the average betweenness and closeness of nodes with the same number of total participations. Both the betweenness and closeness show positive correlation with the number of participations indicating that developers with high betweenness or closeness have more tendency to participate in more OSS projects.

In Figure 2.9, we also notice a bulky head for the correlation points distribution which suggests
Figure 2.7. Indegree and outdegree vs. total number of commits for Ohloh ((a) and (b)) and Github ((c) and (d)).

Figure 2.8. Indegree (left) and outdegree (right) vs. months-work for Ohloh.
a wider range of betweenness and closeness centrality values for nodes of high participation. Similar to the observation in previous section, since we are aggregating nodes based on the total number of participations and the participation distribution follows a power-law distribution where few developers have too many participations while the majority have few participations [16], then the data points aggregated for the low participation nodes are much more than the high participation nodes. This makes the results for the low participation nodes more consistent than the high participation nodes. Analysis on the Ohloh dataset (Figures 2.9(a) and 2.9(b)) shows more clear evidence of the positive correlation between participations and the betweenness and closeness centrality.

Finally, Figure 2.10 shows the number of months-work correlation with nodes’ betweenness and closeness centrality for Ohloh data. In particular, for each months-work we take the average betweenness and closeness of nodes having the same number of months-work. Both betweenness and closeness show positive correlation with the number of months-work indicating that developers with high betweenness or closeness have more tendency to stay longer time committing to OSS projects. However, the node closeness effect seems more than the node betweenness effect.

Interestingly, we did not find a clear evidence of a correlation between either betweenness or closeness centrality and the total number of commits. We argue that developers with high betweenness or closeness, usually, have administrative roles in OSS projects where it imposes high number of social ties. Following this argument, administrative contributors have a higher number of participations and a smaller number of commits than non-administrative contributors. Also, based on Figure 2.10, we argue that administrative contributors stay longer time committed to their projects than non-administrative contributors.

2.5.2 Local Graph Analysis

In this section, we investigate the impact of social connectivity and the ties strength between contributors in a team on the amount of contribution and commitment observed on those team members. The connectivity of contributors in a team is measured using the Local Clustering Coefficient (CC), whereas the ties strength is represented by the frequency of co-participation between the contributors. Moreover, The amount of contribution from a developer in a project is measured by the average number of commits made per month (denoted by commits/month), and the commitment by a developer to a project is measured by the number of active months where a developer submitted at least one commit to a particular project (denoted by months-work). The amount of contribution shows the average amount of work produced by a developer per month in a project, whereas the commitment shows the period of time that a developer stayed committed and active in a project. Note that these metrics are similar to the metrics used in the global network analysis. However, the metrics in the local network analysis estimates the contribution and commitment of developers corresponding to a particular project instead of all participated-in projects as in the global network analysis.

In details, the social factors are measured as follows: For each OSS project team, we extract
Figure 2.9. Betweenness and closeness centrality vs. participations for Ohloh ((a) and (b)) and Github ((c) and (d)).

Figure 2.10. Betweenness (left) and closeness (right) centrality vs. months-work for Ohloh.
the team subgraph from the OSS community social graph and, first, compute the connectivity for each developer \(i\) in a team \(T\) with the local Cluster Coefficient \((CC)\), which represents the connectivity ratio between the acquaintances of developer \(i\) as in Eq. (2.1).

\[
CC_T(i) = \frac{2(k_{N_i})}{|N_i|(|N_i| - 1)}
\]  

(2.1)

where \(N_i\) is the set of \(i\)'s neighbors (acquaintances) in \(T\), and \(k_{N_i}\) is the number of links connecting these neighbors (the vertices in \(N_i\)). Second, we compute a link weight between developers \(i\) and \(j\) (denoted by \(w_{i,j}\)) by calculating the frequency of co-participation between \(i\) and \(j\) and normalize it by dividing by the maximum link weight in the overall social graph, such that \(w_{i,j} = [0, 1]\).

We observe the effects of social factors in various project team sizes. In particular, we group projects into the following team size ranges: \([5,10), [10,25), [25,50), [50,75), [75,100)\) and \([100\) and above\). In this study, we analyze over 1300 OSS projects of different team sizes and topics. There are many more projects in the dataset, but we only consider the projects with team size of more than four developers to exhibit the different connectivity patterns.

### 2.5.2.1 Degree of Connectivity Effect: Highly vs. Lowly Connected

This study analyzes the connected contributors and divides them into two subcategories, Highly Connected (HC) and Lowly Connected (LC) besides the Not-Connected (NC) contributors. The HC contributors are those with \(CC > 0\), i.e., some or all of a contributor’s acquaintances are connected, forming one or several cliques pattern. The LC contributors are those with \(CC = 0\), i.e., a contributor’s acquaintances are not connected, forming a star, line or circle shape pattern. Finally, the NC contributors are those with no acquaintances. To proceed, we select the projects that show all the three types of connectivity patterns (HC, LC and NC) in order to conduct the comparison.

Figure 2.11 analyzes all projects with team sizes of more than four contributors according to the team size sets mentioned above. Figure 2.11(a) shows that the average commits per month is larger for HC contributors than those LC ones in most team sizes. Similarly, Figure 2.11(b) shows that the average months-work is larger for HC contributors than those LC ones in most team sizes. Moreover, both figures show that the NC contributors make the lowest contribution and commitment. These results show a strong statistical evidence that, in general, HC contributors contribute more and are more committed to open projects than LC contributors in various team size ranges.

### 2.5.2.2 Ties Strength Effect: Strong Ties vs. Weak Ties

Next, we study the effect of tie strength on the amount of contribution and commitment, under different team size ranges. Since we measure the strength of a tie between two contributors by the frequency of their co-participations in projects, then one co-participation is counted as one unit of link weight between two developers. Accordingly, we consider a contributor belonging to the Weak Tie set if she has no more than two units of link weights. Contributors with more than
two units of link weights are in the *Strong Tie set*. Finally, contributors with no acquaintances are in the *No Tie set*. To proceed, we select the projects that show all the three types of tie weights (Strong, Weak and No Ties) in order to conduct the comparison.

Figure 2.12 analyzes all projects with team sizes of more than four contributors according to the team size sets mentioned above. Figure 2.12(a) shows that the average commits per month is larger for the Strong Ties contributors than those Weak Ties ones in most team sizes. Also, Figure 2.12(b) shows that the average months-work is larger for the Strong Ties contributors than those Weak Ties ones in all team sizes. These results show a strong statistical evidence that, in general, contributors connected with Strong Ties contribute more and are more committed to open projects than the Weak Ties and No Ties contributors.

### 2.5.2.3 Discussion

In Figure 2.11 and Figure 2.12, most very large teams (those with one hundred and above members) demonstrate different trends, where LC and Weak Tie contributors show more contribution

---

2We decided to have a threshold of two units of link weights to distinguish the weak ties from the strong ties because participants follow power low distribution [16] and most OSS projects have one to two participants.
Figure 2.12. Effect of tie strength on (a) average Commits/Month and (b) average Months-Work for projects of various ranges of team sizes.

and commitment than HC and Strong Tie contributors. We argue that, in these large teams, contributors with high connectivity and strong social ties may have more administrative and controlling roles. On the other hand, contributors with low connectivity and weaker social ties may commit more time to contribute to the project since they may join big projects to gain experience and form connections with reputable developers. Moreover, Figure 2.12 shows that the difference between Weak Ties and No Ties increases as the team size increases. This suggests that Weak Ties become important as the team size increases. We intend to investigate these different trends more in the future.

2.6 Threats to Validity

Some of the contributors in OSS projects are anonymous, which are contributors with no personal account and therefore difficult to trace their identity. In our analysis, we only consider contributors with accounts to link their contribution with their social attributes. Moreover, we define the team size by the known contributors only and ignore the anonymous ones because their social ties are not known. This may cause a threat to validity since anonymous contributors could add
weight to any connectivity pattern. Another threat is that we only capture explicit acquaintances among the developers, where there might be acquaintances that are not declared. However, since we conducted the analysis on large number of OSS developers and projects without selection criteria the threats to validity is minimized. Moreover, the results show consistency on both Ohloh and Github data.

2.7 Conclusion

We conducted an extensive statistical social network analysis on more than a million of online volunteering developers. Our study included a global network analysis (i.e., OSS community level) and a local network analysis (i.e., team network level). The analysis shows that the OSS SNs and ANs follow a power-law distribution and the SNs exhibit small world properties. Also, the OSS SN degree mixing pattern shows that nodes with different degrees are connected together, suggesting interactions between experts and newbie developers. On the other hand, the OSS AN degree mixing pattern shows that developers who participate in projects of certain team size tend to participate to other projects of similar team size. Second, we show positive correlations between indegree, outdegree, betweenness and closeness centrality and the developers’ contribution and commitment in OSS projects. Finally, the local network analysis revealed a strong statistical evidence that, in general, highly connected and strongly tied contributors are more productive than the low connected, weakly tied and not connected contributors. These results lead into better understanding of the OSS social factors that influence team productivity and commitment essential to the success of OSS projects. Moreover, this study helps building recommendation systems, for online open collaborative projects, that are capable of recommending experts and teams of high productivity and expertise.
Trends and Behavior of Developers in Open Collaborative Software Projects

3.1 Introduction

The work in this chapter has been published in the proceedings of the 2014 IEEE International Conference on Behavioral, Economic, and Socio-Cultural Computing (BESC), and it has been awarded the best paper in the conference.

Large collaborative online communities have appeared and grown rapidly in the presence of Web 2.0 technology, witnessed by the massive success of Open Source Software (OSS) projects such as the Apache projects and GNU/Linux. Several research studies investigate the success factors of OSS in the literature [4, 5, 6, 7, 8, 9]. However, these studies are mainly focusing on success factors related to the user-developer interaction (e.g., user ratings and bug reporting) and the project related factors (e.g., programming languages, operating systems and the open source license used). In addition, other studies investigate the social ties between the collaborators and its effect on OSS success. Studies in [10] and [11] suggest that the rapport created from collaboration among individuals in open collaborative communities indeed plays a crucial role leading to the success of OSS projects.

In contrast to commercial software with dedicated responsibility to software engineers, the amount of participation and commitment by the volunteering developers are crucial factors for the success of OSS projects [3]. As the members of OSS projects are usually volunteers [1], it is crucial to investigate their motivations for getting involved in open projects. A study by Guido et al. [2] suggests that the motivations usually involve personal goals/interests, e.g., sharpening the programming skills and gaining experience, following fellow peers, networking with the OSS
community members, or simply supporting free open software projects. We realize that many developer’s and OSS project’s features are not explored and investigated yet.

In this work, we analyze a unique set of developer and project features. Our main purpose is to explore unique trends that can uncover the economics of OSS production and most importantly what drives the developers to participate in OSS projects and what projects they select. We introduce and analyze rich features related to the volunteering developers and the OSS projects. We use statistical and data mining tools to investigate the behavior of the volunteering developers in selecting OSS projects to participate in.

The set of developer features includes several categories. The first category involves the social activity level by finding the number of followers and followings of a developer. The second category involves a developer activity level by finding the number of public repositories a developer has and the number of project participations. The third category involves personal information such as a developer’s affiliation, location and whether he/she is looking for a job or not.

The set of OSS project features includes several categories as well. The first category involves the technical information features such as the main programming language used, the complexity of the software and whether the project has a wiki documentation or not. The second category involves the popularity of a project among the developers and the level of activity of a project. A project popularity features include the number of watchers (i.e., how many people following the development process of the project) and the team size. A project level of activity features include the number of open issues and the number of branch repositories created by developers as a work space related to a main project. The third category involves general information such as the owner type of the project, i.e., a user or organization, and the period of time a project has been active.

It is found that developers who seek a job are more socially active by forming a social network in an effort to connect to professionals and project managers. Also, in a contrary of the environment of OSS development that assumes remote developers, a significant ratio of developers share the same affiliation and location in a team on average, which indicates the existence of a core developers that belong to the same affiliation and sometimes from the same city in some projects. Moreover, we discover certain project features that new developers consider when selecting a project, such as a project complexity and popularity, whereas more experienced developers consider other features such as the primary skill required and the size of a team. Finally, we conduct our analysis on a real dataset collected from GitHub.com, which is a fast-growing online OSS community that includes over a two million open project and over a million contributor.

In this work, we have made a number of contributions:

- We conduct the analysis on a large dataset consisting of more than two million open project and more than a million contributor.
- We study and analyze 6 developer features and 9 project features extensively.
- We discover and analyze several trends in the OSS development process and developers’ behavior.
• We identify and rank the project features most responsible for the developers' behavior in selecting a project to participate in.

The rest of the chapter is organized as follows. Section 3.2 reviews some related works. Section 3.3 introduce the dataset. Section 3.4 discusses the developer features and trends analysis. Section 3.5 discusses the project features and the developers' preferences on selecting projects. Section 3.6 discusses and analyze the findings, and finally Section 3.7 concludes the chapter.

3.2 Related Works

There exist several works that study and analyze the interests and motivations of developers. Subramaniam et al. [4] study the factors that may affect the development activity, such as developer interest, user interest, project activity (i.e., bug reports and user comments), project status (i.e., new project, development phase, maintenance phase) and project attributes (i.e., OSS license, operating system, programming language). Lee et al. [19] study the motivations of developers to choose between the commercial and the OSS project and found that highly skillful developers find participating in OSS projects more rewarding than commercial software projects. Also, Roberts et al. [20] find that developers' status and reputation is one of the important motivations to contribute to OSS projects. Moreover, Casaló et al. [5] collect information via questionnaires and study the correlation between developers commitment to their OSS project and the reputation of those projects. They conclude that OSS reputation have an indirect effect on the commitment of collaborators. Also, Guido et al. [2] suggest that the motivations usually involve personal goals/interests such as sharpening the programming skills and gaining experience, networking with the OSS community members, or simply supporting free open software projects. Krishna et al. [21] study the OSS developers and projects growth rate and distributions. Their work concludes that the failure of projects to attract more developers may prove to be the single important reason for the death of OSS projects. This conclusion suggests the importance of studying the OSS features attracting the developers to participate in the OSS projects which is the main purpose of our work.

3.3 Dataset

We conduct the analysis on a dataset collected from Github.com which is one of the most fast-growing OSS online repository. Github.com, hosting over two million OSS projects and over one million contributors, is unique since it provides a variety of interaction tools between the contributors and tools that allow to track the developers’ activities such as forking projects and hosting personal public repositories. Table 3.1 summarizes the data set collected from Github.com. Participations in the first row is the total number of developer-project records in the data set. In the analysis, we only consider the collaborators with account in Github.com since they are identifiable by a unique ID, whereas anonymous contributors are hard to track.
Table 3.1. Collected Data Statistics.

<table>
<thead>
<tr>
<th></th>
<th>Github</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participations</td>
<td>9,268,644</td>
</tr>
<tr>
<td>Contributors</td>
<td>1,034,996</td>
</tr>
<tr>
<td>Contributors with account</td>
<td>652,040</td>
</tr>
<tr>
<td>Social Ties</td>
<td>1,788,803</td>
</tr>
<tr>
<td>Projects</td>
<td>2,332,749</td>
</tr>
</tbody>
</table>

and know their participation history. The social ties are computed from the followers/following links between developers.

3.4 Developers’ Behavior and Analysis

In this work, we define a developer behavior as the actions of a developer related to (i) activity level and (ii) selection of OSS projects. In order to study the behavior of a developer, we introduce and analyze several developer features. The features belong to three categories. The first category involves the social activity level by counting the number of followers and followings of a developer, the second category involves the collaboration activity level and the third category involves personal information.

Social Ties.

(1) Followers count: The number of followers is an indication of the social status and a recognition by the OSS community members. Developers with high number of followers are usually project managers or people with highly recognized contributions. Figure 3.1(a) shows the Followers Complementary Cumulative Distribution Function (CCDF). Basically, the CCDF plot shows the probability (y-axis) of any data point (developer) to have a value (e.g. followers count) greater than or equal the given value at x-axis. A proper power-law model is shown as the fitted data in Figure 3.1. To test the dataset feature against the power-law model we find the estimate of the scaling exponent $\alpha$ and the Kolmogorov-Smirnov goodness-of-fit test $D$ as in [22], where a value close to zero indicates that the data is closely estimated by the power-law model. A power low distribution indicates that few developers have high number of followers while the majority have few followers.

(2) Following count: The number of people a developer follow is an indication of social activeness. Figure 3.1(b) shows that Following follows a power-law distribution as well. Developers with high following count can demonstrate different characters and roles. Some may be apprentices who follow expert developers and others may be experts who create a professional network by following other fellow developers. Table 3.2 shows basic statistics of Followers and Following features.

The OSS community is a good environment for developers who look for a job. Github.com gives the option to a developer to declare if he/she is hireable or not in their profile. We want to analyze the correlation between being hireable and social ties. We first divide the developers according to their number of Followers/Following ties into four groups. Then, we calculate
the percentage of hireable developers for each group interval as displayed in Figure 3.2(a). In general, as social ties increase, the percentage of people looking for a job increases. Moreover, the percentage of hireable developers is higher with the increase of Following count than the Followers count, especially when Following count is > 100 with almost double the percentage of hireable developers. This suggests that developers who have many following ties are more likely interested in finding a job than developers who have many followers. This is reasonable since developers who look for a job may want to connect to professionals and project managers to hire them.

**Development Activity.**

(3) Public Repositories (Pub.Repos.) count: Github.com allows developers to create public repositories to host projects or parts of a project. The number of Pub.Repos. a developer has indicates his/her activeness in the OSS developing process. Figure 3.1(c) shows the CCDF function for Pub.Repo. The high $\alpha$ value and the extended plot to the right indicates that developers with high Pub.Repos. have very close values.

(4) Developer Participations (D.Partc.): this is the count of projects a developer participates in. A higher number of participations indicates a more active developer in the community. Figure 3.1(d) shows that the probability of a developer to have many project participations decrease faster than power-law model which means developers with > 100 participations have large participation disparity. The large standard deviation of D.Partc. (in Table 3.2) also confirms the large disparity in D.Partc. among the developers.

Table 3.3 shows the correlations between the social ties and the development activity given by Pearson coefficients, where the value is between -1 and 1 indicating a negative or positive dependency between two variables, respectively. All coefficients in Table 3.3 show positive correlation, however, the magnitudes are different. The correlation coefficients between Followers-D.Partc. and Following-D.Partc. are close to zero which indicates a weak correlation between the social ties and the number of project participations. On the other hand, the correlation coefficients between Followers-Pub.Repos. and Following-Pub.Repos. have higher magnitude which indicates a higher correlation between the social ties and the number of public repositories. Moreover, the correlation between Following-Pub.Repos. is more than Followers-Pub.Repos. This suggests that developers who have high following ties are more likely to be active in terms of OSS developing process.

Moreover, we study the percentage of hireable developers when development activity increases. We first divide the developers according to their number of Pub.Repos. and D.Partc. count into four groups. Then, we calculate the percentage of hireable developers for each group interval as displayed in Figure 3.2(b). In general, as the developers’ participation and public repositories increase, the percentage of people looking for a job increases. Also, we notice that when D.Partc. and Pub.Repos. are low (the first two intervals), the percentage of hireable developers is higher with the increase of D.Partc. than the Pub.Repos., and when D.Partc. and Pub.Repos. are high (the last two intervals), the percentage of hireable developers is higher with the increase of Pub.Repos. than the D.Parcc., especially when it is > 50 with almost double the percentage of
Table 3.2. Developer Numeric Feature Statistics.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>min.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>max.</td>
<td>9008</td>
<td>1865</td>
<td>5246</td>
<td>13738</td>
</tr>
<tr>
<td>avg.</td>
<td>6.14</td>
<td>4.69</td>
<td>7.74</td>
<td>11.19</td>
</tr>
<tr>
<td>std. dev.</td>
<td>49.80</td>
<td>17.17</td>
<td>20.86</td>
<td>68.64</td>
</tr>
</tbody>
</table>

Table 3.3. Developer Social Ties and Development Activity Correlations.

<table>
<thead>
<tr>
<th></th>
<th>Correl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Followers</td>
<td>0.16</td>
</tr>
<tr>
<td>Pub.Repos.</td>
<td>0.07</td>
</tr>
<tr>
<td>D.Partc.</td>
<td>0.25</td>
</tr>
<tr>
<td>Following</td>
<td>0.06</td>
</tr>
</tbody>
</table>

hireable developers. This is in accordance with the previous finding, where it is reasonable for the developers who are looking for a job to be more active in OSS development as a way to be recognized in the community, especially in having more public repositories than many project participations.

**Developer Affiliation and Location.**

Since OSS development is mainly based on online collaboration between remote developers, it is interesting to know whether OSS projects consist of developers from same affiliations and locations or not? Moreover, the information about a developer affiliation and location can help improve recommender systems to suggest projects to developers in an effort to increase the participation rate in OSS projects.

(5) **Affiliation:** There exist many affiliations in OSS community. Some of them are corporates and others are universities. First, we identify the set of affiliations involved in each project and find the largest one in terms of number of members. The largest affiliation involved in a project is the dominant affiliation. Then, we find the percentage of developers belonging to the dominant affiliation in each project. Figure 3.3 shows the average percentage of developers belonging to the dominant affiliation for each set of projects according to their team size interval. For small team size projects, \([2, 5]\) members, more than 50% of the developers are from the same affiliation. Then, surprisingly, as the team size increases, almost third of the team belongs to the same affiliation, even for large teams of \(> 40\) developers. This results suggest that OSS projects have core developers from one affiliation.

(6) **Location:** In addition to the affiliation membership, we analyze the location of developers. We identify a location by the city name. Similar to affiliation, we identify the location where each developer belongs to in each project and find the dominant location. Figure 3.3 shows that in small team size project, more than half the developers belong to the same city. However, unlike affiliation, as the team size increases the percentage of developers belonging to the same city decreases until it reaches around 18% for teams of \(> 40\) members. This suggests that the core developers can belong to the same affiliation but can be from different locations at the same time. We also learn that small teams are more probable to have developers from the same affiliation.
3.5 Developers’ Preference on Selecting OSS Projects

It is of high paramount to know the project features that are of most interest to developers. The question we are trying to answer is the following: What are the important project features that make a developer decide whether to participate or not in a project? This study involves individual preference analysis because a set of features interesting to a developer may not be interesting to another. To achieve our goal, we first define and analyze a set of project features that can affect the decision of developers. The features belong to three categories. The first category involves the technical information, the second category involves the popularity of a project among the developers and the level of activity of a project and the third category involves information about the owner type and a project’s age.

The second step is to define the developer level of interest in a project. A simple way would be defining a binary value to indicate whether a developer participates in a project or not. However, binary value does not show the level of interest to the selected project. Therefore, we define a categorical variable with three levels of developer interest as Not Interested (NI), Interested (IT)
and Strongly Interested (SI). We use the amount of commits\(^1\) as an indication of a developer interest in a project as follows: NI if commits = 1, IT if commits from [2, 40] and SI (if commits > 40). The rational is that a developer contributing more in a project is more interested in this project and vice versa.

There are two kind of feature selection approaches in literature, the first selects a subset of features and the second ranks the features according to some feature evaluator. In this study, we are mostly interested on ranking the features based on their importance to the developer. More precisely, we calculate the Information Gain for each feature and rank the features based on their Information Gain value. We detail the feature ranking process in a later section. Next, we introduce and analyze the set of project features.

### 3.5.1 Project Features

**Technical Information.**

\(^{(1)}\) The primary programming language (Lang.): Usually a software is written in several pro-

\(^{1}\)A commit is the update procedure a developer performs to upload his/her edits to a project’s files.
### Table 3.4. Programming Languages Statistics.

<table>
<thead>
<tr>
<th>Prog. Language</th>
<th>Freq.</th>
<th>Relative %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruby</td>
<td>279,640</td>
<td>18.49</td>
</tr>
<tr>
<td>JavaScript</td>
<td>266,130</td>
<td>17.59</td>
</tr>
<tr>
<td>Java</td>
<td>183,804</td>
<td>12.15</td>
</tr>
<tr>
<td>Python</td>
<td>150,762</td>
<td>9.97</td>
</tr>
<tr>
<td>PHP</td>
<td>144,707</td>
<td>9.57</td>
</tr>
<tr>
<td>C</td>
<td>80,143</td>
<td>5.30</td>
</tr>
<tr>
<td>Others</td>
<td>407,357</td>
<td>26.93</td>
</tr>
</tbody>
</table>

programming languages but there is only one main language that identifies the core structure of the software. The main programming language is usually the main skill to ask for when recruiting developers to work in a commercial software project and therefore it is a crucial feature for selecting a project to participate in. Github.com reports 92 programming languages. Table 3.4 shows the frequency and relative percentage of the top six programming languages used in the OSS projects. The top six languages are used in 74% of the OSS projects.

(2) **Complexity of the OSS:** Complexity is an important feature in selecting a project and it is widely ignored in previous work. Newbie developers may want to start with relatively easy projects to train their skills while experts may want to participate in large complex software to improve their status. There are many metrics to measure a software complexity, however, the main and simplest metric is the code size [23]. We use the code size (C.size), measured by line of code, as a main metric to indicate the level of software size and complexity.

(3) **Project documentation:** This feature is crucial for large projects because new contributors usually rely on good documentations to understand the project, especially with a wiki page to introduce the project and its documentation. We use a binary feature indicating whether a project has a wiki page or not called (hasWiki). A project wiki can attract more developers to participate and therefore can be an important feature that developers look for on their selection process.

**Popularity and Activity Information.**

Popularity features give an indication of how much popular an OSS project is among the OSS community developers. On the other hand, activity features give an indication of how much active an OSS project is. The following are the popularity features.

(4) **Number of watchers (Watchers):** Github.com allows developers to add projects to a watch list to track the projects’ development progress and activities. The number of watchers gives an indication of how many developer are interested in a project. Figure 3.4(a) shows the distribution of watchers which follows a power-law distribution.

(5) **Team size (T.size):** is the number of developers contributing to a project. The team size is also an indicator of a project’s popularity and can be a feature that attracts developers who are looking for professional networking. Figure 3.4(b) shows the distribution of team size which follows a power-law distribution with a wide range of large team sizes.
Table 3.5. Project Numeric Feature Statistics.

<table>
<thead>
<tr>
<th></th>
<th>T.size</th>
<th>Forks</th>
<th>OI</th>
<th>C.size</th>
<th>Watchers</th>
</tr>
</thead>
<tbody>
<tr>
<td>min.</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>max.</td>
<td>3342</td>
<td>5106</td>
<td>1540</td>
<td>6029900</td>
<td>19258</td>
</tr>
<tr>
<td>avg.</td>
<td>11.51</td>
<td>4.73</td>
<td>1.87</td>
<td>8214.54</td>
<td>21.86</td>
</tr>
<tr>
<td>std. dev.</td>
<td>107.36</td>
<td>35.59</td>
<td>12.42</td>
<td>65681.09</td>
<td>169.23</td>
</tr>
</tbody>
</table>

Table 3.6. Project Numeric Feature Correlations.

<table>
<thead>
<tr>
<th></th>
<th>T.size-C.size</th>
<th>OI-Forks</th>
<th>Watchers-Forks</th>
<th>Watchers-OI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correl.</td>
<td>0.27</td>
<td>0.40</td>
<td>0.85</td>
<td>0.38</td>
</tr>
</tbody>
</table>

The following are the activity features. (6) **Number of Open Issues (OI):** An OI is a case related to a software whether it is a bug report or technical issue in a project. The number of OI is an indication of a project level of activity. A project with more number of open issues can attract a certain type of developers who seek to involve in solving technical issues. Figure 3.4(c) shows the distribution of OI which follows a power-law distribution.

(7) **Number of forks (Forks):** Github.com allows a developer to fork (branch) a main trunk repository for a project to create a personal repository as a workspace. A project with many forks indicates that many developers are interested and working on this project. Figure 3.4(d) shows the distribution of forks which follows a power-law distribution.

**Owner and Age Information.**

(8) **Owner type (OwnerT.):** Github.com allows a user or an organization to create a project. A developer may be interested to collaborate in projects owned by organizations or users.

(9) **Age of the project (Recency):** We divide a project age into three categories, old (2 years or older), recent (6 months or older) and new (up to 6 months old). A project age feature can also be a selection criteria where some developers prefer old stable projects to work in, other developers may prefer new projects.

Table 3.5 shows the basic statistics of the project numeric features. The analysis considers projects with more than one collaborator. C.size has a very high standard deviation (std. dev.) indicating a large difference between projects’ complexity. Also, T.size and Watchers has high std. dev. indicating big gaps in T.size and Watchers between the projects. Forks and OI have the least std. dev. indicating close Forks and OI values between the projects. Table 3.6 introduce important correlations found between some of the numeric features based on Pearson correlation. The positive correlation between T.size and C.size indicates that large teams involve in large complex projects. Moreover, Table 3.6 shows a strong positive correlation between the number of Watchers, OI and Forks. We speculate that as the number of OI increases, the number of Watchers and Forks increase as well. Note that we do not focus on causality study in this work which identifies which feature triggers the change of another feature.
3.5.2 Project Feature Preference Analysis

As mentioned above, the feature selection analysis has to be done on individual level due to the diverse preference from person to another. However, we are interested in finding general trends of developers’ preferences instead of individual ones. On the other hand, considering all the developers as one group will not show the different trends. Therefore, we group the developers into four sets based on their number of project participations as follows: Group with participations \([2, 10]\), \([11, 50]\), \([51, 200]\) and \(>200\). For each group, the average of each feature preference is taken to be the trend of this group. More detail on this process in the next section.

We choose the number of participations to divide the groups for the following reasons. First, developers with few participations are usually new collaborators (more than 70% are new), and developers with many participations are usually earlier (old) developers with more experience (more than 80% are old). Therefore, we can analyze the developers’ preferences based on their recency to the community since a newbie developer may have different preferences in selecting projects than the expert developers. Second, it is more reasonable to group developers over a close range of participation count because the developers with a certain level of activity may share similar preferences and trends.

We adopt the feature selection method where it ranks the features based on the Information
Gain value of each feature. To apply the method on the dataset, we prepare the dataset so that each record in the database contains the project features and the developer’s classes of preference as Not Interested (NI), Interested (IT) or Strongly Interested (SI).

**Project Feature Ranking**

We use the Information Gain (IG) as a feature evaluator. IG ratio is a value from 0 to 1 that specifies how important a feature is in classifying an item. If every value in a feature results in a unique classification of an item, then IG equals 1. On the other hand, IG decreases when the values of a feature do not result in a unique classification for an item. In our case, a project feature has high IG if it can uniquely classify whether a developer is NI, IT or SI in a project. This indicates that this feature is important for deciding the preference of a developer in a project, which also means that the developer, indirectly, relies on this feature in his/her decision of contribution to that project.

For each group of developers, we apply the feature selection ranking method on each individual in the group. The result is the average of ranking over every developer in the group. The feature ranking results and the GI values are shown in Table 3.7 for each group. Table 3.7(a) shows the features ranking of developers with participations [2, 10]. We see that the top three important features are C.size, Watchers and T.size. This suggests that the developers of this group look for project complexity (by C.size information) as their top priority in addition to the project popularity (by Watchers and T.size information). This is reasonable since most of the developers in this group are new developers and they may search for a project to participate in based on its complexity level and popularity.

Table 3.7(b) shows the features ranking of developers with participations [11, 50]. In this group, Lang., project Recency and T.size are the top three important features on selecting a project. We argue that once a developer gains some experience in OSS development he/she would develop expertise in certain programming languages. Therefore, the developer selection would be based on the primary programming language of a project where the developer has some expertise in. Then comes a project Recency as a second priority, where a developer may look for

<table>
<thead>
<tr>
<th>Rank</th>
<th>Feature</th>
<th>IG</th>
<th>Rank</th>
<th>Feature</th>
<th>IG</th>
<th>Rank</th>
<th>Feature</th>
<th>IG</th>
<th>Rank</th>
<th>Feature</th>
<th>IG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C.size</td>
<td>0.721</td>
<td>1</td>
<td>Lang.</td>
<td>0.301</td>
<td>1</td>
<td>T.size</td>
<td>0.214</td>
<td>1</td>
<td>T.size</td>
<td>0.206</td>
</tr>
<tr>
<td>2</td>
<td>Watchers</td>
<td>0.638</td>
<td>2</td>
<td>Recency</td>
<td>0.193</td>
<td>2</td>
<td>Lang.</td>
<td>0.170</td>
<td>2</td>
<td>Lang.</td>
<td>0.149</td>
</tr>
<tr>
<td>3</td>
<td>T.size</td>
<td>0.602</td>
<td>3</td>
<td>T.size</td>
<td>0.133</td>
<td>3</td>
<td>C.size</td>
<td>0.088</td>
<td>3</td>
<td>C.size</td>
<td>0.113</td>
</tr>
<tr>
<td>4</td>
<td>Forks</td>
<td>0.575</td>
<td>4</td>
<td>OwnerT.</td>
<td>0.108</td>
<td>4</td>
<td>Recency</td>
<td>0.084</td>
<td>4</td>
<td>Recency</td>
<td>0.045</td>
</tr>
<tr>
<td>5</td>
<td>Lang.</td>
<td>0.475</td>
<td>5</td>
<td>hasWiki</td>
<td>0.084</td>
<td>5</td>
<td>OwnerT.</td>
<td>0.0530</td>
<td>5</td>
<td>OwnerT.</td>
<td>0.029</td>
</tr>
<tr>
<td>6</td>
<td>OI</td>
<td>0.451</td>
<td>6</td>
<td>C.size</td>
<td>0.073</td>
<td>6</td>
<td>Forks</td>
<td>0.047</td>
<td>6</td>
<td>hasWiki</td>
<td>0.028</td>
</tr>
<tr>
<td>7</td>
<td>Recency</td>
<td>0.417</td>
<td>7</td>
<td>Watchers</td>
<td>0.055</td>
<td>7</td>
<td>Watchers</td>
<td>0.044</td>
<td>7</td>
<td>OI</td>
<td>0.027</td>
</tr>
<tr>
<td>8</td>
<td>OwnerT.</td>
<td>0.241</td>
<td>8</td>
<td>Forks</td>
<td>0.0349</td>
<td>8</td>
<td>hasWiki</td>
<td>0.038</td>
<td>8</td>
<td>OwnerT.</td>
<td>0.026</td>
</tr>
<tr>
<td>9</td>
<td>hasWiki</td>
<td>0.142</td>
<td>9</td>
<td>OI</td>
<td>0.048</td>
<td>9</td>
<td>OI</td>
<td>0.026</td>
<td>9</td>
<td>Forks</td>
<td>0.024</td>
</tr>
</tbody>
</table>
new or old and stable project to participate in. T.size is still in rank 3 as in the previous group. Tables 3.7(c) and 3.7(d) show the features ranking of developers with participations \([51, 200]\) and \(>200\), respectively. The ranking of the top four features are the same for the two groups. It shows that for developers with high project participations, T.size is the most important feature on project selection, then comes the programming language and then the complexity of the project. This indicates that those developers prefer a certain team size to work with as their first priority. It is reasonable to control the team size preference if the developer participates in many projects. For example, a developer may prefer large team sizes to have minimum responsibility and be able to participate in more projects.

In general, the team size appears on the top three important features in all the groups which means it is a key feature for project selection. Moreover, contrary to our believe, the programming language is not a top priority for beginners but it is a priority for more experienced ones. This gives the impression that new developers are seeking to learn new skills, whereas experienced developers seek to improve the existing skills.

Regarding the rest of the features, developers with few participations (group \([2, 10]\)) are more interested in OI and Forks than the developers with higher participations. This suggests that developers with few participations care more about open issues and the number of forks in the project to gain experience. Also, ownerT. and hasWiki are more important for the developers with high participations than the developers with low participations. This suggests that the developers with high participations follow certain type of a project owner and certain standard of documentation.

3.6 Discussion of the Findings and its Usage

As discussed in the introduction, the participation of developers in the OSS projects is essential to guarantee the continuous and success of these projects. The purpose of our analysis is to pave the way for developing recommender systems that are capable of accurately recommending projects to developer to increase the participation rate. Newbie developers can be recommended popular projects with the type of complexity they are seeking, whereas developers with more experience can be recommended project with skills that match their skills of expertise and of a team size range preferred to them. Moreover, the affiliation and location of the majority in a project team can be used to attract more developers of the same affiliation or location to participate. Moreover, social ties can play crucial part on attracting more developers to participate as it has been found in previous works. Therefore, we can recommend projects with popular contributors to the developers who seek a job since the trend shows that they have high following ties. Many features can be exploited to the favor of an accurate recommender system.
3.7 Conclusion

We conducted an extensive analysis on the developers of Open Source Software (OSS) projects of more than a million developer and over two millions project. Our study investigates the trends and behavior of developers in the OSS community and the development process. We study and analyze 6 developer features belonging to three categories. The first category involves the social activity level, the second category involves the collaboration activity level and the third category involves personal information. Moreover, we study and analyze 9 project features belonging to three categories as well. The first category includes technical information, the second category includes the popularity and activity level information and the third category includes the owner type and recency of the project.

Furthermore, it is found that developers who seek a job are more socially active than others. Also, surprisingly, a significant ratio of developers share the same affiliation and location with team members, which indicates the existence of a core developers that belong to the same affiliation and sometimes from the same city in some projects. Moreover, we discover certain project features that new developers consider when selecting a project, such as a project complexity and popularity, whereas more experienced developers consider other features such as the primary skill required and the size of a team.

The feature selection analysis conducted in this work finds the significant features that affect a developer decision on selecting a project to participate in. In the future work, we intend to analyze the values of these features that a developer prefers. In another word, we want to discover the kind of projects that a certain type of developers are interested in. Moreover, we intend to exploit the findings of this work in building a recommender system for the OSS community to increase the participation of the developers and assist the projects with low participation to find the suitable developers.
Increasing the Responsiveness of Recommended Expert Collaborators for Online Open Projects

4.1 Introduction

The work in this chapter has been published in the proceedings of the 2014 ACM international Conference on Information and Knowledge Management (CIKM).

With continuous growth of online social communities, engaging experts of various specialties to collaborate online becomes more and more feasible nowadays. In Open Source Software (OSS) communities, skills and strong motivations from volunteers are both crucial elements leading to success of collaborative software projects [4, 5, 10, 24, 11]. With growing complexity of software systems and size of online collaborative projects, finding relevant expertise for specific tasks becomes very challenging. Thus, there is a need to develop expert recommendation systems for OSS communities.

Two main challenges arise in developing such expert recommendation systems. The first is to find a suitable yet motivated expert and the second is related to data scarcity. For the first challenge, there exist several works to find experts who are the most knowledgeable about the given task, e.g. [25, 26, 27]. However, these works may not be sufficient for recommending experts in large online community as they mostly focus on finding experts in offline corporate projects. Under the context of OSS development, a significant issue, known as the “cold shoulder” problem [10], is that an expert may simply ignore requests/invitations from project coordinators. To mitigate this problem, most previous works, e.g. [24, 11, 28] suggest decreasing the social distance to the project coordinator who issues the request. The idea is to increase the chance of responses by finding socially close experts based on their social distance to the requester. The
challenge in this matter is how to define this social closeness notion in a massive network of collaborators with the ability of accommodating disconnected or new members.

We adopt the notion of Relative Importance in [29] where two nodes in a graph are related by the paths that connect them and the amount of importance is given according to the path(s) distance, i.e., the longer the path is the less importance conveyed by that path. To accommodate the nature of the open collaborative networks that may have disconnected subgraphs, we adapt the methods in [29] to propose Social Relative Importance (SRI) such that disconnected nodes are still considered with penalty weights. We consider three SRI approaches, including two weighted-path methods, i.e., Shortest Path (SP) and k-Short Node Disjoint Paths (kSNDP), and a Markov-Chain based approach, i.e., Random Walk with Restart (RWR).

Given the unique environment of open collaborative projects that requires motivations and skills, the SRI approaches may not be sufficient for predicting the likelihood of an expert willing to collaborate, since socially close developers do not necessarily have the required skills and knowledge to work on the requested task. To address this problem, we introduce the notion of the Degree of Knowledge (DoK) that anticipates the expert’s ability and willingness to collaborate in a given task. The first DoK model is based on the amount of accumulated knowledge on programming skills over previously participated in projects called Relative Accumulative Knowledge Score (RAKS). The second model uses a content-based filtering approach [30], where a developer’s profile of previously participated projects is compared with the query project, and the developer’s preference is given by the RAKS score. We call the model the Project Content-Based Filtering (PCBF) model.

In addition to the cold shoulder problem, the challenge of data scarcity is prominent in OSS communities since the developer-project participation follows a power-law distribution [16], i.e., the majority of developers contribute only to a few projects yet there are a few developers contributing to many projects. The developers who participate in few projects does not mean that they are not experts since some of them do have a great amount of contribution in those projects. In recommender systems, the problem of a user or an item having limited history data is known as the “cold-start” problem, which degrades the performance of recommender systems. To mitigate this problem, we introduce two probabilistic generative models, namely, Skill Preference Model (SPM) and Skill and Contribution Preference Model (SCPM). These two models, following the concept of Aspect model, introduced by Hofmann et al. [31], represents an individual’s preferences as a convex combination of preference factors. The first model uses a latent variable to capture the developers’ preference on selecting projects with certain skills and contribution, while the second model uses two latent variables to separately capture the developers’ skill preferences and the developers’ contribution level.

Accordingly, we propose the Expert Ranking (ER) framework which combines the factors of DoK and SRI, to rank the candidate expert collaborators based on their amount of knowledge and interest to collaborate with the query initiator on a given task. Finally, we evaluate the proposed models on a real dataset collected from GitHub.com, a fast-growing online OSS community. The experiment shows promising results for recommending expert collaborators to make real
collaborations to projects.

In this work, we have made a number of contributions:

- We develop an expert recommender system for online open collaborative projects by recommending experts with high chance of responding to queries and collaborating in a task.

- To address the cold shoulder problem, we design a number of models, including RAKS which quantifies the abstract meaning of knowledge on certain skills, PCBF which is capable of measuring the amount of knowledge and interest of an individual to participate in a given project, SPM and SCPM. Further, we adapt a number of SRI methods, SP, kSNDP and RWR, suitable for open collaborative networks.

- To address the cold start problem, we design two probabilistic generative models, SPM and SCPM, capable of recommending experts with limited data history.

- To rank the candidate expert collaborators, we propose the ER framework which integrates the factors of DoK and SRI.

- We evaluate our models on a real historical data set from Github.com which consists of more than a million contributors and more than two millions open projects. Further, we evaluated the proposed models on four levels of history data availability.

The rest of the chapter is organized as follows. Section 4.2 reviews some related works. Section 4.3 introduces some preliminaries and defines the expert collaborator recommendation problem. Section 4.4 introduces the proposed models. Section 4.5 details the experiment methodology and evaluates the proposed models, and finally Section 4.6 concludes the chapter.

### 4.2 Related Works

In this section we present some works on experts recommendation proposed in the literature. Surian et al. [32] consider finding the most compatible developers to a query initiator by considering whether a candidate and the initiator have collaborated before, worked in similar projects, and shared similar skills. This approach focus on the compatibility between the developers but ignores the compatibility between the developer and the task in hand. As a result, it may not recommend new experts. The works by Martín-Vicente et al. [33] and Massa et al. [34] consider building a trust network for recommendation either by inferred action, i.e., approving similar items, as in [33], or by explicit trust action as in [34]. In our work, the trust network is realized by the explicit social (acquaintances) and collaboration links. Fritz et al. [27] developed a model to find experts for specific software components by authorship and the amount of interaction with the component information. This model is suitable to find experts within the project’s team members and is not suitable for open collaborative community consisting of thousands of projects. On the other hand, Ma et al. [26] argue that searching for experts based on their software components’ usage expertise is comparable to implementation expertise which makes
the search possible outside the team members. Hu et al. [35] propose to combine developers’ skills information with the collaboration network. In our work, we combine the collaboration network with the social network, i.e., the network represents social links and the link weights are derived from previous collaborations if exist. Finally, the works in [36, 37] target the field of finding experts on scientific papers via bibliographic network and propose a Bayesian model to estimate an expert’s knowledge on some topic. In our work, in addition to modeling the amount of expertise, we model the likelihood of an expert to participate in a task with specific skills by adding the amount of contribution on each skill to the model. In contrast to previous works on experts recommendation, we focus on increasing the likelihood of experts responding to a query by considering their participation history, knowledge and social ties.

4.3 Expert Collaborator Recommendation Problem

In this section, we introduce some preliminaries and describe the problem formulation for expert collaborator recommendation in the OSS communities.

Let \( G(X, E) \) be an undirected social graph representing an OSS community, where \( X = \{x_1, ..., x_n\} \) is the set of active developers in the community and \( E \) represents the acquaintances relationships among the developers. Here each edge \( e \) is associated with a weight \( w \). Edge weights can be derived from any relation type. In this work, edge weight is derived from the frequency of collaborations. Note that, if two nodes are socially linked but never collaborated before then \( w = 0 \), otherwise \( w > 0 \). Also, let \( S = \{s_1, ..., s_m\} \) be the set of all skills and \( Y = \{Y_1, ..., Y_l\} \) be the set of all projects. Every project \( Y_k \in Y \) consists of a set of developers and requires a set of skills. Thus, we also use \( Y_k \) to denote the set of skills in a project \( k \), i.e., \( Y_k \subseteq S \).

Given a developer \( x_i \), let \( y_i = \{y_1, ..., y_w\} \) be a vector of projects that \( x_i \) participates in and let \( x_i = \{s_1, ..., s_l\} \) be a vector of skills possessed by developer \( x_i \). Together, \( y_i \) and \( x_i \) represent a Developer Profile (DP). Likewise, let \( s_j = \{x_1, ..., x_h\} \) be a vector of developers who possess skill \( s_j \), which represents a Skill Profile (SP). Vectors \( x \), \( s \) and \( y \) are derived from a 3-dimensional, developer-skill-project, matrix \( M \) of size \( n \times m \times l \) where a matrix entry \( m_{i,j,k} > 0 \) denotes the contribution of developer \( i \) to project \( k \) with skill \( j \).

**Problem Formulation.** Given an expert collaborator query \( Q \equiv (x_Q, Y_Q) \) where \( x_Q \in G \) is the query issuer (a developer who issues the query) and \( Y_Q = \{s_1, ..., s_r\} \) is a set of required skills for the project/task. In responding to \( Q \), an expert collaborator recommender finds and ranks top-\( n \) developers who are most likely to collaborate with \( x_Q \) on project/task \( Y_Q \).

4.4 Proposed Solutions

To answer the expert collaborator query, we consider two main aspects to find and rank experts who are most likely to collaborate with a query issuer in a given task. The first aspect considers the Degree of Knowledge (DoK) and the second aspect considers the Social Relative Importance (SRI). We propose four models under the DoK aspect and enhance the models with three SRI
approaches. The two aspects are used in the Expert Ranking (ER) framework to rank the experts. In the following sections, we introduce in detail the proposed solutions.

4.4.1 Degree of Knowledge

The Degree of Knowledge (DoK) aspect concerns about the amount of knowledge a developer has towards a certain set of skills and the likelihood of contributing in projects with such skills needed. In this section, we introduce in detail four alternative models for the DoK aspect: (1) Relative Accumulative Knowledge Score (RAKS) based on the accumulated knowledge on programming skills over previously participated in projects; (2) Project Content-Based Filtering (PCBF) which compares previous projects with the query project where the similarity is weighted by a developer’s preference given by the RAKS score; (3) Skill Preference Model (SPM), based on a probabilistic generative model that captures the developers’ preferences on selecting certain skills with certain amount of contribution; (4) Skill and Contribution Preference Model (SCPM), based on a probabilistic generative model. Here, SCPM captures developers’ preference on selecting skills and their contribution level separately, allowing for more precise prediction.

Instead of considering all the developers in the OSS as candidates for a given query \( Q \equiv (x_Q, Y_Q) \), we first use \( Y_Q \) as a filter to eliminate those who do not have the required skills. Thus, a set of developers \( X_Q = \bigcup_{s_j \in Y_Q} s_j \) where each developer in \( X_Q \) possesses at least one skill in \( Y_Q \) are extracted.\(^1\) In the following sections, we detail the DoK models.

4.4.1.1 Relative Accumulative Knowledge Score

The Relative Accumulative Knowledge Score (RAKS) measures the amount of knowledge a developer has on certain skills. Given a developer \( x_i \in X_Q \), RAKS is defined based on Eq. (4.1).

\[
RAKS(x_i, Y_Q) = \kappa \left( \mu_i \sum_{s_j \in Y_Q} \sum_{Y_k \in Y_i} cont_{s_j,Y_k} \right)
\]  

(4.1)

where \( \mu_i = \left( \frac{|x_i \cap Y_Q|}{|Y_Q|} \right) \) is the ratio of known skills to \( x_i \) in \( Y_Q \), \( cont_{s_j,Y_k} \) is the frequency of contributions or commits of developer \( x_i \) using skill \( s_j \) in project \( Y_k \), and \( \kappa \) is a normalizing factor such that \( RAKS(x_i, Y_Q) = [0,1] \). The RAKS score considers the amount of contribution, measured by the number of commits\(^2\), by a developer \( i \) using a skill \( j \) in different projects as a way to measure developer \( i \)'s expertise to skill \( j \). Also, the \( \mu \) factor in Eq. (4.1) assists to distinguish between a developer who knows all the required skills in \( Y_Q \) and another who knows a fraction of the required skills.

\(^1\)Note that we do not solve the skills coverage problem in this work since experts are ranked as individuals and not as a team.

\(^2\)A commit is the update procedure a developer performs to upload his/her edits to a project’s files.
4.4.1.2 Project Content-Based Filtering

The Project Content-Based Filtering (PCBF) aims to improve over RAKS by considering both the amount of knowledge and the likelihood of a developer $x_i$ to participate in $Y_Q$. For a developer $x_i \in X_Q$, PCBF computes the expected value for $x_i$ to collaborate in $Y_Q$ based on Eq. (4.2):

$$PCBF(x_i, Y_Q) = \kappa \sum_{Y_k \in Y_i} \text{sim}(Y_Q, Y_k) \times RAKS(x_i, Y_k)$$  \hspace{1cm} (4.2)

where $\text{sim}(Y_Q, Y_k)$ is the similarity measure between $Y_Q$ and a previously participated in project $Y_k$ by developer $x_i$, $RAKS(x_i, Y_k)$ measures the RAKS score for $Y_k$ similar to Eq. (4.1), and $\kappa$ is a normalizing factor such that $PCBF(x_i, Y_Q) = [0, 1]$.

In a user-item recommendation scenario, the content-based filtering model measures the expected rate of a user $i$ on selecting a given item $j$ based on the history of user $i$’s ratings for previously selected items that are similar to item $j$. Likewise, Eq. (4.2) measures the expected rate of developer $i$ to collaborate in project $Y_Q$ based on the history of developer $i$’s previous collaborations on similar projects to $Y_Q$ as well as $i$’s knowledge on each of these previous projects. The similarity measure is essential to the success or failure of a content-based filtering approach. In PCBF, we use skills as a main attribute to measure similarities between projects.
4.4.1.3 Skill Preference Model (SPM)

The Skill Preference Model (SPM) is designed to predict the probability of an expert to participate in a query project $Y_Q$. SPM is based on the notion of aspect model which models individual preferences as a convex combination of preference factors. In OSS community, a history record $\langle x, y, s, c \rangle$ consists of a developer $x$ participating in project $y$ using skill $s$ with amount of contribution (commits count) $c$ has a great amount of information that can be used for advanced predictions. A latent variable in the aspect model is associated with each observation (record) which allows classifying each skill and contribution selection by $x$ in its appropriate class. This creates a great amount of flexibility on classifying individuals’ preferences and behavior. In addition, probabilistic generative models tend to work well with scarce data since it can classify records based on the trained distributions of the model’s parameters.

Given a query $Q \equiv (x_Q, Y_Q)$, we want to estimate for each $x_i \in X_Q$ the probability for $x_i$ to participate in $Y_Q$ (i.e., $P(Y_Q|x_i)$ or simply $P(y|x)$). Note that $P(y|x)$ can be computed as

$$P(y|x) = \frac{P(x, y)}{P(x)} \propto P(x, y) \quad (4.3)$$

Assuming one latent variable $Z = \{z_1, ..., z_k\}$, we design the aspect model as in Figure 4.1(a). The model shows the important fact that $x$, $s$, $y$ and $c$ are independently conditioned on $Z$. The probability model can be simply written as

$$P(y|x) = \sum_{z \in Z} \sum_{s \in S_y} \sum_{c \in C_x} P(x, y, z, s, c) \quad (4.4)$$

where $S_y$ is the set of skills associated with $y$ and $C_x$ is the set of contribution levels associated with $x$. Also, the joint probability distribution over all factors is

$$P(x, y, z, s, c) = P(x)P(z|x)P(s|z)P(c|z)P(y|z) \quad (4.5)$$

where $P(z|x)$ is the probability of $x$ falling in class $z$, and $P(s|z)$, $P(c|z)$ and $P(y|z)$ are class-conditional multinomial distributions giving the probabilities of class $z$ selecting $s$, $y$ and $c$. Overall, the SPM model with one latent variable gives the probability of developer $x$ selecting project $y$ with skill $s$ and amount of contribution $c$. $P(x) = 1/|n|$ where $n$ is the total number of developer in the community and hence it is the same for every $x$ which we ignore in later equations. Comparing to a user-item recommendation scenario, one can think of $s$ as the content of an item and $c$ as the rating given to the item. Hence, SPM models the developer’s preference on selecting projects with certain skills and how much contribution is exerted in such project with these skills.

To train SPM model, we use the Expectation Maximization (EM) algorithm to learn the model parameters from the set of developer-project-skills-contributions history $H$, i.e., $\langle x, y, s, c \rangle \in H$.3

---

3The amount of contribution in this work is converted to a nominal value representing different levels of contributions (e.g., low, med and high). One can interpret it as ordinal value as well.
Note that a project may require multiple skills in which \( s \) may be a set of skills \( S \). Our model parameter learning algorithm is based on the idea of maximizing the log-likelihood of \( \mathcal{L}(\theta) \).

\[
\mathcal{L}(\theta) = \sum_{\langle x, y, s, c \rangle \in H} \log(P(x, y, s, c|\theta))
\]

(4.6)

where \( \theta \) denotes the model parameters, i.e., \( P(z|x), P(s|z), P(c|z) \) and \( P(y|z) \). The EM algorithm iterates between the E-step and M-step. In the E-step, the algorithm calculates the posterior probability of every latent variable \( z \in Z \), based on the current estimates of the parameters. More specifically, we calculate

\[
P(z|x, y, s, c) \propto \frac{P(z|x)P(s|z)P(c|z)P(y|z)}{\sum_{z \in Z} P(z|x)P(s|z)P(c|z)P(y|z)}
\]

(4.7)

In the M-step, model parameters are computed to maximize the expected log-likelihood in the E-step as below.

\[
P(z|x) = \frac{\sum_{\langle x, y', s', c' \rangle \in H \ z \in Z} P(z|x, y', s', c')}{\sum_{\langle x, y', s', c' \rangle \in H \ z' \in Z} P(z|x, y', s', c')}
\]

\[
P(y|z) = \frac{\sum_{\langle x', y', s', c' \rangle \in H \ z \in Z} P(z|x', y', s', c')}{\sum_{\langle x', y', s', c' \rangle \in H \ z' \in Z} P(z|x', y', s', c')}
\]

(4.8)

\[
P(s|z) = \frac{\sum_{\langle x', y', s', c' \rangle \in H \ z \in Z} P(z|x', y', s', c')}{\sum_{\langle x', y', s', c' \rangle \in H \ z' \in Z} P(z|x', y', s', c')}
\]

\[
P(c|z) = \frac{\sum_{\langle x', y', s', c' \rangle \in H \ z \in Z} P(z|x', y', s', c')}{\sum_{\langle x', y', s', c' \rangle \in H \ z' \in Z} P(z|x', y', s', c')}
\]

where \( \sum_{z \in Z} P(z|x) \), \( \sum_{y \in Y} P(y|z) \), \( \sum_{s \in S} P(s|z) \) and \( \sum_{c \in C} P(c|z) \) are all 1. Note in Eq. (4.8) the variable with a prime means counting every value of this variable. Iterating between the E-step and M-step, the EM algorithm improves the model parameters on each iteration until they converge to a local log-likelihood maximum.

4.4.1.4 Skill and Contribution Preference Model (SCPM)

A developer has different contribution efforts (levels) using certain skills in each project participation. Using one latent variable may not be able to precisely capture the contribution level of a
developer since the contribution information $c$ is mixed with the skills information $s$. To distinguish between a developer’s preference on selecting certain skills and his/her contribution level, we introduce a second version of SPM that uses two latent variables as depicted in Figure 4.1(b).

Having two latent variable $Z_s = \{z_{s_1}, ..., z_{s_k}\}$ representing developers’ skill preference and $Z_c = \{z_{c_1}, ..., z_{c_l}\}$ representing developers’ contribution level. Then we notice that $x$, $s$ and $y$ are independently conditioned on $Z_s$, while $x$, $c$ and $y$ are independently conditioned on $Z_c$. The probability model can thus be written as

$$P(y|x) = \sum_{z_s \in Z_s} \sum_{z_c \in Z_c} \sum_{s \in S} \sum_{c \in C_x} P(x, y, z_s, z_c, s, c)$$

and the joint probability distribution over all factors is

$$P(x, y, z_s, z_c, s, c) = P(x)P(z_s|x)P(z_c|x)P(s|z_s)P(c|z_c)P(y|z_s, z_c)$$  (4.10)

where $P(z_s|x)$ is the probability of $x$ falling in skill preference class $z_s$ and $P(z_c|x)$ is the probability of $x$ falling in contribution level class $z_c$. $P(y|z_s, z_c)$ is the joint probability of both $z_s$ and $z_c$ selecting project $y$. Eq. (4.10) calculates the joint probability of developer $x$ falling in class $z_s$ to select project $y$ with skill $s$, and $x$ falling in class $z_c$ to contribute at $c$ level in project $y$.

Again here $P(x) = 1/|n|$ where $n$ is the total number of developer in the community and hence it is the same for every $x$ which we ignore in later equations. This refined model is expected to raise the accuracy of prediction.

The EM algorithm is used here as well. However, SCPM is more challenging since it has two latent variables. In this case the E-step contains two posterior probabilities, one for each latent variable as follows.

$$P(z_s|x, y, s, c, z_c) \propto \frac{P(z_s|x)P(z_c|x)P(s|z_s)P(c|z_c)P(y|z_s, z_c)}{\sum_{z_s \in Z_s} P(z_s|x)P(z_c|x)P(s|z_s)P(c|z_c)P(y|z_s, z_c)}$$

$$P(z_c|x, y, s, c) \propto \frac{P(z_s|x)P(z_c|x)P(s|z_s)P(c|z_c)P(y|z_s, z_c)}{\sum_{z_c \in Z_c} P(z_s|x)P(z_c|x)P(s|z_s)P(c|z_c)P(y|z_s, z_c)}$$  (4.11)

In the M-step, model parameters are computed to maximize the expected log-likelihood in the E-step as below.
disconnected from the query issuer. In the following, we introduce three adapted approaches to
consider the disconnected nodes to accommodate new members or members that are
importance recognizes the importance of a target node relative to a source node.

Given a network \( G \) with experts who are willing to contribute their skills and knowledge.

### 4.4.2 Social Relative Importance

Different from global importance models, e.g., centrality measures and \textit{PageRank}, that measures the importance of a node in the entire network, in this work we consider the \textit{Social Relative Importance} (SRI) between two nodes, i.e., a candidate developer relative to the query issuer. Given a network \( G(X, E) \), the relative importance between a source node \( i \in G \) and a target node \( j \in G \) can be defined by the path distance between \( i \) and \( j \), in general. In this sense, relative importance recognizes the importance of a target node relative to a source node.

We adopt the approaches in [29] to estimate the relative importance and modify the approaches to consider the disconnected nodes to accommodate new members or members that are disconnected from the query issuer. In the following, we introduce three adapted approaches to
realize SRI, in which two are weighted-path methods, i.e., Shortest Path (SP) and k-Short Node Disjoint Paths (kSNDP), and the third is a Markov-Chain based approach, i.e., Random Walk with Restart (RWR).

4.4.2.1 Shortest Path

The SP is the simplest form of measuring SRI between two nodes. Given a social graph $G$ as defined in Section 4.3, an edge weight $w$ may equal zero if the two end nodes never collaborated before but are socially acquainted. Since we cannot use zero edge weights into weighted-path methods, we offset every edge weight by a very small value $\epsilon$ that is less than the minimum edge weight in $G$. Eq. (4.13) shows the SP model for nodes $i$ and $j$.

$$SP_{i,j} = \begin{cases} \text{mindist}\{P(i,j)\} & \text{if } P(i,j) \neq \emptyset \\ \lambda & \text{if } P(i,j) = \emptyset \end{cases} \quad (4.13)$$

where $P$ is the set of paths between $i$ and $j$ and $\lambda$ is a very large value as a penalty if no path exist. The $\text{mindist}$ function outputs the minimum distance value of the shortest path in $P$.

4.4.2.2 k-Short Node Disjoint Paths

In addition to SP, we consider the kSNDP defined in [29]. Instead of considering a single shortest path, kSNDP accounts for multiple node-disjoint paths between $i$ and $j$ of length $\leq k$ nodes. Node-disjoint means that no node is re-visited in another path. A target node connected to a source node through multiple unique paths is assumed to have high importance value. We refer the reader to [29] for further details on calculating kSNDP. Similar to the SP approach, if a target node is disconnected from the source node, then a penalty value $\lambda$ is set instead.
4.4.2.3 Random Walk with Restart

Random Walk with Restart, a.k.a. personalized PageRank or PageRank with priors [29], is based on Markov chains. The original idea derived from the random walker model which gives, globally, the probability of a random walker in a graph to end on some node after a number of time-steps transiting from one node to another neighboring node with a probability $\alpha$ to randomly jump to any node in the graph. These probabilities represent the fraction of time that the random walker spends at any single node. RWR adds to the random walker model prior probabilities to start at certain node(s) $R$ where those probabilities sum to one, and a probability $\beta$ to restart the random walk from $R$ at each time-step. After convergence, the probability of a node $j$ relative to a source node $i \in R$ gives the likelihood of $i$ to visit $j$, which is a stochastic notion of relative closeness in a graph. In this work, the query issuer represents the re-starting node $r$ where $R = \{r\}$. We use the resulting scores, biased towards $r$, as the relative importance value after convergence. Originally, RWR works in a connected graph only. Therefore, if a target node is disconnected from $r$, then a penalty value $\lambda$ is set instead.

4.4.3 Expert Ranking Framework

Given the various realizations of SRI and DoK, we propose a multiplicative framework, called Expert Ranking (ER) to rank experts as in Figure 4.2. Given a query $Q \equiv (x_Q, Y_Q)$, the ER framework obtains the OSS data records, $(x, y, s, c)$, and social graph $G$, then filters the records based on the skills required in $Y_Q$ and processes $G$ according to each SRI method as discussed in Section 4.4.2. The ER framework, then, integrates the outputs from DoK and SRI based on the ER scoring function, $ER_{Q,i}$, in Eq. (4.14) for a query $Q$ and a candidate expert $i$.

$$ER_{Q,i} = \begin{cases} 
DoK_{Y_Q,i} \times SRI_{x_Q,i} & \text{if } SRI \in \{kSNDP, RWR\} \\
DoK_{Y_Q,i} \times (SRI_{x_Q,i})^{-1} & \text{if } SRI \in \{SP\}
\end{cases}$$ (4.14)

where $DoK \in \{RAKS, PCBF, SPM, SCPM\}$. Basically, one DoK model is used with one SRI approach on each ER evaluation, e.g., $ER = RAKS \times kSNDP$. In the case of $SRI \in$
Table 4.1. Collected Data Statistics.

<table>
<thead>
<tr>
<th></th>
<th>Github</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participations</td>
<td>9,268,644</td>
</tr>
<tr>
<td>Contributors</td>
<td>1,034,996</td>
</tr>
<tr>
<td>Contributors with account</td>
<td>652,040</td>
</tr>
<tr>
<td>SN Edges</td>
<td>1,788,803</td>
</tr>
<tr>
<td>Projects</td>
<td>2,332,749</td>
</tr>
</tbody>
</table>

$kSNDP, RWR\}$, ER multiplies the values of the two models since a larger value of kSNDP (multiple short paths) and RWR (probability value) is better than a smaller value. On the other hand, in the case of $SRI \in \{SP\}$, ER multiplies DoK by the reciprocal of SRI since a smaller value of SP (shorter path means closer distance) is better than a bigger value. The ER score is distinct for each SRI and DoK model combination. Therefore, we emphasize that our purpose of the ER is to rank the candidate experts and not to define a unique measuring score.

We opted to use the multiplication over addition in the ranking function for the following reasons: First, it is simpler and more effective to penalize the ranking score, in the case of disconnected experts from $x_Q$, than addition which needs a trained balancing weight factor between SRI and DoK. Second, the absolute value differences between SRI and DoK are of less interest and importance than the percentage changes between them which makes a multiplication a better choice. Finally, the multiplication performs better than addition in the initial experiments. However, we do not show those results since this is not the focus of this work.

4.5 Experiment and Evaluation

We evaluate the proposed models on a historical data set collected from Github.com which is one of the most fast-growing OSS online repository. Github.com, hosting over two million OSS projects and over one million contributors, is unique since it provides a variety of interaction tools between the contributors. One of these tools allows contributors to explicitly recognize each other through a follow link that creates a directed social network of contributors. Table 4.1 summarizes the data set collected from Github.com. Participations in row one is the total number of developer-project records. In the experiment, we consider only the collaborators with account in Github.com since they are identifiable by a unique ID, whereas anonymous contributors are hard to track and know their participation history.

4.5.1 Experiment Methodology

To setup the experiment, we conduct a series of processing steps on the Github.com dataset. First, a social graph is constructed from direct follower/following links between the contributors. However, we consider these edges as undirected in the experiment. We emphasize that the graph edge is based on a social tie and the edge weight is based on collaborations between two endpoints.
(i.e., developers). This means if two individuals are acquaintances but did not collaborate, then their edge weight is set to zero, on the other hand, if two individuals collaborated in a project but are not acquaintances, then there is no edge between them. We do not use an affiliation graph since the OSS community is a large Web-based community in which some developers work in isolation and do not actually know each other. Therefore a social network would resemble true acquaintance among contributors which may increase a recommender system’s accuracy. Besides the social graph of collaborators, projects’ meta data, e.g., skills used, collaborators, and their amount of contribution (number of commits), is stored in a database.

Having the graph of collaborators and the projects’ database, we randomly select one project at a time, and from the set of collaborators in this project we choose one developer as a query issuer $x_Q$ and mark off (remove) his collaborators from the social graph and the projects’ database. The required skills of the marked-off project become $Y_Q$. In the social graph, the edge weights between $x_Q$ and his/her acquaintances in $Y_Q$ are decremented by one. Therefore, if the marked-off project is the only collaboration, then the edge weight is set to zero. Note that the social links are not necessarily existing within members of the same project. On the contrary, many links exist with developers outside the project team. In this case, the SRI approaches consider distant collaborators in the solution.

The marked-off collaborators and their associated projects become the test set with their associated queries of $x_Q$ and $Y_Q$ each. Then the experiment is conducted as follows: For each query, $Q \equiv (x_Q,Y_Q)$, we match the top-20 developers recommended by ER with the original team members marked off and consider each match as a hit, which means that the recommender system was able to predict a collaborator that indeed made a collaboration with the query initiator $x_Q$. In addition to finding matches, we evaluate the relevancy and responsiveness of a matched contributor to $Q$ by her/his percentage of contribution in the marked-off project $Y_Q$, i.e., developers who contributed more in the marked-off project are more relevant and responsive to $Q$ than developers who contributed less. We discuss the evaluation metrics in detail in Section 4.5.2.

To train the probabilistic models, SPM and SCPM, we remove all the test (marked-off) projects’ teams from the history data. Furthermore, we make sure that the selected test projects’ teams do not have overlapping contributors to insure that each test is unique and not repeated. Also, we select teams of sizes between 40 to 50 members to have fair evaluation between the queries since small teams of 2-5 members may be difficult to find matches in top-20 while large teams of $> 100$ members may be easy to find matches in top-20. Moreover, in SPM and SCPM, we want to compute $P(y|x)$, where $y$ is marked-off (i.e., not included in the training set). Therefore, in the experiment, we use a similar project to $y$ in terms of skills required to be the project query $Y_Q$. In a non-test (operating) setting, $Y_Q$ will be the task/project associated with $x_Q$.

Finally, to evaluate the strength and weakness of each model on different data availability, we divide the test sets, the marked-off projects, based on the developers’ number of participation records in each team. We divide the test set into four sets: $[2, 10)$, $[10, 50)$, $[50, 200)$ and $\geq 200$. For example, projects in $[2, 10)$ include teams of developers with participation records between
2 to 9 project participations. Set \([2, 10)\) resembles the most scarce data set, and set \(\geq 200\) resembles the most abundant dataset. The next section presents the evaluation and discussion.

### 4.5.2 Evaluation

We want to evaluate the different combinations of DoK models and SRI methods based on (i) how many expert candidates can they find? (ii) how responsive are those candidates to \(Q\)? and (iii) how well are they ranked in the top-20? We use the \emph{precision} metric for (i), the expert’s percentage of contribution, i.e., \emph{responsiveness} (resp.), in \(Y_Q\) for (ii) and the \emph{normalized Discounted Cumulative Gain} (nDCG) \([38]\) metric for (iii). Recall that \emph{precision} is the fraction of relevant records over the total number of records retrieved. As mentioned in the previous section, we consider a developer as relevant if she/he is a member of the original marked-off project \(Y_Q\). We emphasize that finding the members of the original team is not the purpose of the recommender system proposed, but only a method to evaluate its effectiveness in recommending developers who tend to make actual collaborations in the future.

Recall that the nDCG evaluates relevant records retrieved based on its relevancy level and its position on the rank, i.e., a system ranking highly relevant records in high positions is considered better than a system ranking highly relevant records in low positions. Under the context of this work, relevancy is estimated by the amount of contribution a developer exerts in a project, i.e., the number of commits. To measure nDCG, we develop a multi-level relevancy model based on the percentage of contribution for a developer in a project as follows: \(\geq 50\%\) is highly relevant, \([20\%, 50\%)\) is moderate relevant, \([5\%, 20\%)\) is relevant, and \(\leq 5\%\) is least relevant.

We emphasize that the nDCG is different from \emph{precision} as the latter only counts the number of relevant records in the retrieved set, while, nDCG evaluates the relevancy and ranking position of these relevant records. Moreover, \emph{resp.} evaluates the quality of the relevant records in terms of the amount of relevancy regardless of ranking. Finally, the parameters of the models are as follows: for kSNDP, the maximum path length is \(k = 6\) and the decay coefficient is \(2\). For RWR, the restart probability \(\beta = 0.3\). For SPM, \(|Z| = 15\), and for SCPM \(|Z_s| = 10\) and \(|Z_c| = 6\), where the number of classes initialized by a clustering technique. In the following sections we present the results and discuss and analyze the findings.

#### 4.5.2.1 Evaluation of Expert Recommendation on an Individual Case

In this section, we show the impact of our proposed models on recommending experts to a candidate individual and task/project query. Later in the next sections, we show the aggregate results of multiple queries. Here we randomly select a query developer with a project in hand and recommend experts to him/her using every proposed model and evaluate the quality of each model results. A random project with ID \((3976679)\), noted by \(Y_Q\), and a developer with ID \((21979)\), noted by \(x_Q\), are selected where they both represent \(Q\).

Table 4.2 shows the top-10 recommended experts for query \(Q\) using each DoK model with SRI being the shortest path (SP) method. We select SP as the base line method to control the social
Table 4.2. Top-10 recommended experts for query Q comparing every DoK model.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>n</td>
<td>n</td>
<td></td>
<td>D</td>
<td>y</td>
<td>2.24</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>n</td>
<td>n</td>
<td></td>
<td>A</td>
<td>y</td>
<td>9.78</td>
<td>A</td>
<td>y</td>
<td>9.78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>n</td>
<td>n</td>
<td></td>
<td>B</td>
<td>y</td>
<td>0.20</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>y</td>
<td>9.78</td>
<td>n</td>
<td></td>
<td></td>
<td>E</td>
<td>y</td>
<td>0.05</td>
<td>n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>n</td>
<td>A</td>
<td>y</td>
<td>9.78</td>
<td>F</td>
<td>y</td>
<td>0.05</td>
<td>B</td>
<td>y</td>
<td>0.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>n</td>
<td>C</td>
<td>y</td>
<td>25.71</td>
<td>G</td>
<td>y</td>
<td>0.10</td>
<td>H</td>
<td>y</td>
<td>0.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>y</td>
<td>0.20</td>
<td>n</td>
<td>n</td>
<td></td>
<td>D</td>
<td>y</td>
<td>2.24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>n</td>
<td>B</td>
<td>y</td>
<td>0.20</td>
<td>u</td>
<td></td>
<td>I</td>
<td>y</td>
<td>4.07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>J</td>
<td>y</td>
<td>6.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>K</td>
<td>y</td>
<td>27.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

factor and be able to compare the different DoK models. For each DoK model, the table shows the recommended expert symbol ID (given by a capital letter for ease of comparison), whether the expert is a hit or not and the percentage of contribution the expert exerted on the query project in the case of a hit. RAKS found 2 experts, A and B, in which A has conducted 9.78% of the total contribution of the query project $Q$ and B has conducted 0.2%. PCBF found 3 experts including A and B in addition to expert C which has conducted 25.71% of the total contribution. The higher the contribution percentage is the higher the match is and the probability of an expert to respond to a query. Therefore, we consider PCBF better than RAKS since it found more hits and it found a highly responsive expert in the top-10 ranks. SPM found 6 hits all in top 6 ranks, however, it did not find C expert in the top-10 ranks. Meanwhile, SCPM found 7 experts with relatively high responsiveness as experts K, A, J and I are all in top-10 ranks. Therefore, SPM has high precision and better ranking but moderate responsiveness, however, SCPM has high precision and high responsiveness in this individual case example.

Table 4.3 shows the top-10 recommended experts for query Q using each SRI method with RAKS being the DoK model. We select RAKS as the base line model to control the DoK factor and be able to compare the different SRI methods. For each SRI method, the table shows the recommended expert symbol ID, whether the expert is a hit or not and the percentage of contribution the expert exerted on the query project in the case of a hit. The table shows that RWR method finds more match experts than SP and kSNDP and also ranks the hits higher. The two experiments above show only an individual case to demonstrate the effectiveness of each DoK model and SRI method in recommending experts to a query developer. In the next sections, we conduct the experiment on multiple queries and discuss the findings.

4.5.2.2 ER Evaluation

Based on the ER framework, one can use different combinations of DoK and SRI schemes to make expert recommendations. In order to evaluate the ER on each DoK model with different SRI methods, we evaluate all the combinations grouped by the SRI method. The first experiment evaluates the DoK models without the SRI factor, i.e., DoK-only. The second, third and fourth
Table 4.3. Top-10 recommended experts for query Q comparing every SRI method.

<table>
<thead>
<tr>
<th>rank #</th>
<th>Expert ID</th>
<th>hit?</th>
<th>cont. %</th>
<th>Expert ID</th>
<th>hit?</th>
<th>cont. %</th>
<th>Expert ID</th>
<th>hit?</th>
<th>cont. %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>n</td>
<td>n</td>
<td>A</td>
<td>9.78</td>
<td>y</td>
<td>9.78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>n</td>
<td>n</td>
<td>B</td>
<td>0.20</td>
<td>y</td>
<td>J</td>
<td>6.42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>y</td>
<td>9.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>n</td>
<td>A</td>
<td>y</td>
<td>9.78</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>y</td>
<td>0.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>n</td>
<td>n</td>
<td>D</td>
<td>2.24</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>n</td>
<td>B</td>
<td>y</td>
<td>0.20</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

experiments evaluate the DoK models with SP, kSNDP and RWR, respectively. Figure 4.3 and Table 4.4 shows the results of each DoK model and SRI method combination. Table 4.4 shows the Hits@100, resp.@20 and nDCG@20, where the bold face result resembles the maximum result for its group, while, the highlighted result resembles the maximum value overall.

DoK-only: Figure 4.3(a) shows the average precision of the top 20 expert collaborators retrieved. With no SRI factor, we can see that the overall precision values are very low (below 0.1). Also, we notice that the memory-based models, RAKS and PCBF, have better precision, Hits@100, resp.@20 and nDCG@20 than SPM and SCPM. The reason might be that, without a social factor, SPM and SCPM find experts that are not responsive or willing to participate with the query issuer. As a result, the memory-based models will perform better with no social factor since it depends mainly on history data.

DoK-{SP}: In Figure 4.3(b), we notice that the precision values are all above 0.1, which is a great improve when using a social factor. Comparing the DoK models, we notice the superiority of SPM and SCPM over RAKS and PCBF in all metrics. SPM and SCPM seem to have similar precision, however, SCPM has better Hits@100, resp.@20 and nDCG@20. This means SPM and SCPM find similar number of hits on top-20 but SCPM finds more responsive developers and rank them higher than SPM.

DoK-{kSNDP}: Figure 4.3(c) and Table 4.4 shows that SPM and SCPM perform better than RAKS and PCBF on all metrics except resp.@20 where RAKS and PCBF have higher values. This shows the advantage of RAKS and PCBF on retrieving highly responsive developers with the kSNDP method, while SPM and SCPM retrieve more hits and rank them higher.

DoK-{RWR}: In Figure 4.3(d), we notice that the RAKS and PCBF precision got slightly higher, while the SPM and SCPM precision got lower, and overall close precision performance to all. Also, RAKS has the highest resp.@20 and nDCG@20 while SCPM has the highest Hits@100. Using RWR seems to boost the memory-based DoK models while a bit halting the probabilistic based DoK models. The reason might be that RWR finds socially close developers that also have high centrality in the social network which prevent them from being highly responsive. This explains the fact that probabilistic models find more hits but less responsive developers.
Figure 4.3. The average precision for various $k$ expert collaborators on combinations of DoK and SRI methods grouped by the SRI method.

Moreover, memory-based models, RAKS and PCBF, finds more responsive developers since they count more on history data.

Overall, SCPM-SP has the best $\text{Hits}@100$ and $\text{nDCG}@20$, where RAKS-RWR has the best $\text{resp.}@20$. There is no winner between the SRI methods but it is definite that SRI methods improve the DoK models in general. Also, the probabilistic models seem to perform the best with SP, while, the memory based models perform the best with RWR for the aformentioned reasons. Moreover, we notice that SPM has the lowest $\text{resp.}@20$ in all combinations because SPM models skill preference and contribution level combined. With the separation of preferences, skill and contribution level, we see that SCPM can find more responsive developers than SPM.

4.5.2.3 ER with Multiple History Record Levels Evaluation

This evaluation investigates the ER performance on different history records (h.r.) levels. We control the social factor by using one SRI method and compare all DoK models on multiple h.r. levels. We choose the base line $SRI = \{SP\}$, then $ER = \{RAKS, PCBF, SPM, SCPM\} \times \{SP\}$.

**Precision:** Figure 4.4 shows the average precision of the top 20 expert collaborators found corresponding to different sets of h.r. $[2, 10), [10, 50), [50, 200)$ and $\geq 200$. Figure 4.4(a) shows clearly the superiority of the probabilistic models SPM and SCPM over the memory based models RAKS and PCBF which could not retrieve any developer on top 20. This particular result shows...
Table 4.4. ER Evaluation on Hits, resp. and nDCG.

<table>
<thead>
<tr>
<th>ER model</th>
<th>Hits@100</th>
<th>resp.@20</th>
<th>nDCG@20</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAKS</td>
<td>4.2593</td>
<td>28.7922</td>
<td>0.0853</td>
</tr>
<tr>
<td>PCBF</td>
<td>5.8889</td>
<td>23.7754</td>
<td>0.0747</td>
</tr>
<tr>
<td>SPM</td>
<td>1.7037</td>
<td>2.8053</td>
<td>0.0185</td>
</tr>
<tr>
<td>SCPM</td>
<td>1.3333</td>
<td>4.1888</td>
<td>0.0101</td>
</tr>
<tr>
<td>RAKS-SP</td>
<td>6.1852</td>
<td>34.6882</td>
<td>0.1525</td>
</tr>
<tr>
<td>PCBF-SP</td>
<td>7.037</td>
<td>30.1184</td>
<td>0.1284</td>
</tr>
<tr>
<td>SPM-SP</td>
<td>10.1852</td>
<td>25.945</td>
<td>0.223</td>
</tr>
<tr>
<td>SCPM-SP</td>
<td>11.0741</td>
<td>38.0679</td>
<td>0.2647</td>
</tr>
<tr>
<td>RAKS-kSNDP</td>
<td>5.3704</td>
<td>37.6289</td>
<td>0.1199</td>
</tr>
<tr>
<td>PCBF-kSNDP</td>
<td>6.0741</td>
<td>30.0497</td>
<td>0.094</td>
</tr>
<tr>
<td>SPM-kSNDP</td>
<td>7.3704</td>
<td>17.047</td>
<td>0.1496</td>
</tr>
<tr>
<td>SCPM-kSNDP</td>
<td>7.4815</td>
<td>23.7152</td>
<td>0.1611</td>
</tr>
<tr>
<td>RAKS-RWR</td>
<td>8.1111</td>
<td>44.5444</td>
<td>0.2183</td>
</tr>
<tr>
<td>PCBF-RWR</td>
<td>7.8148</td>
<td>40.7941</td>
<td>0.1873</td>
</tr>
<tr>
<td>SPM-RWR</td>
<td>9.4815</td>
<td>26.7395</td>
<td>0.1796</td>
</tr>
<tr>
<td>SCPM-RWR</td>
<td>9.4074</td>
<td>31.6185</td>
<td>0.2003</td>
</tr>
</tbody>
</table>

the power of SPM and SCPM in recommending experts with very limited data history.

In Figures 4.4(b) and 4.4(c), we still notice the superiority of SPM and SCPM over RAKS and PCBF. However, Figure 4.4(b) shows that RAKS is performing better than PCBF which still suffers from data scarcity and it improves as data becomes abundant as shown in Figures 4.4(c) and 4.4(d). In particular, Figure 4.4(d) shows that memory based models, RAKS and PCBF, performs better than probabilistic models, SPM and SCPM, when data is abundant. Comparing SPM and SCPM, we notice similar performance, with similar number of hits on top 20.

Figure 4.5 shows that SCPM retrieves more experts than SPM in all test sets except set [10, 50) where they appear to be identical. Moreover, SPM and SCPM have more hits than PCBF and RAKS in all test sets except the ≥ 200 set. Also, PCBF outperforms RAKS on sets [50, 200) and ≥ 200 of abundant data history.

Responsiveness: Next, we evaluate the responsiveness of the relevant experts to Q. Figure 4.6 shows resp. @20 for each data history record set. In scarce data set [2, 10), SPM and SCPM outperform RAKS and PCBF with the superiority to SCPM. However, when data becomes more abundant, RAKS and PCBF performs better where PCBF has slight advantage over RAKS in sets [50, 200) and ≥ 200. The results suggest that SPM can find candidates as much as SCPM, as suggested by the precision evaluation, but SCPM finds more responsive candidates than SPM.
(a) For h.r. between [2, 10)

(b) For h.r. between [10, 50)

(c) For h.r. between [50, 200)

(d) For h.r. ≥ 200

Figure 4.4. The average precision for various k expert collaborators on multiple h.r. sets.

Figure 4.5. The avg. Hits@100 for DoK models on multiple h.r. sets.

Figure 4.6. The avg. resp.@20 for DoK models on multiple h.r. sets.
Ranking: Finally, we evaluate the quality of ranking by the nDCG metric. Figure 4.7 shows nDCG@20 for each data history record set. We see the difference between SPM and SCPM very clear, where SCPM has better quality on ranking more relevant developers in higher positions than SPM in all test sets. Also, RAKS has better quality than PCBF in all test sets except the $\geq 200$ set, and RAKS and PCBF perform better than SPM and SCPM in the very abundant data set of $\geq 200$.

Discussion: Dividing the data set according to the history records availability gave new insights on the strengths and weaknesses of well known recommender models such as memory-based and probabilistic models. The previous evaluation shows that the probabilistic models, SPM and SCPM, have unmatched superior performance over the memory-based models, RAKS and PCBF, when data is scarce. The design of SPM and SCPM allows high prediction power, even when data is very scarce, because a developer’s preference is classified according to the learned parameters from the training dataset. On the other hand, the memory-based models, RAKS and PCBF, perform better when data is very abundant because the models by design increase the prediction power as it sees more data which makes it more accurate than the probabilistic models in this case.

We also notice another unpredicted results where the test set $\geq 200$ has the lowest performance in all evaluations. We argue that most of the developers who have too many participations ($\geq 200$) tend to have low contribution in these projects. This is logical since a volunteering developer may not have enough time to contribute much in each project (assuming a simultaneous participation to multiple projects at one time). The less amount of contribution by a developer in a project is interpreted by the DoK models as lack of experience or/and a lack of interest from that developer in the project, which reduces the prediction accuracy.

Comparing SPM to SCPM, the evaluation shows that SCPM finds more responsive experts than SPM and has higher nDCG than SPM. This is due to the separation of the latent variable $Z$ into $Z_c$ and $Z_s$, which makes the distinction between a developer’s contribution level and skills preference, respectively. Given this advantage, SCPM can more accurately recommend higher
contributing (responsive) developers than SPM.

4.6 Conclusion

In this work, we target two main problems in recommending experts in online open projects. The first problem is related to the responsiveness of experts in collaborating in open projects, and the second problem is related to data scarcity. We mitigate those problems in two aspects, i.e., DoK and SRI. The DoK involves the amount of knowledge or expertise of candidate experts to a given task. The SRI involves the social factor between the candidate experts and a query issuer. We consider three approaches, i.e., SP, kSNDP and RWR, to estimate the SRI and proposed four models, i.e., RAKS, PCBF, SPM and SCPM to estimate the DoK. Moreover, we introduced the ER framework to incorporate the DoK and SRI to rank the candidate experts. We conduct extensive experiments on a real historical data set from Github.com of more than a million contributors and more than two millions open projects. We evaluate the precision, responsiveness and goodness of ranking of each DoK model and SRI approach combination. We also evaluate the DoK models on different data scarcity level. The results show the effectiveness of the ER in recommending experts that are highly responsive and motivated to contribute their skills. In the future, we aim to investigate the possibility of defining a metric score to quantify the expert’s interest to participate instead of the ER score for ranking. We believe that the proposed models to recommend experts can increase the experts’ involvement in open projects and, as a consequence, increase its likelihood of success.
Chapter 5

Recommending Projects for Developers in Open Source Software Community

5.1 Introduction

Large collaborative online communities have appeared and grown rapidly in the presence of Web 2.0 technology, witnessed by the massive success of Open Source Software (OSS) projects such as the Apache projects and GNU/Linux. In contrast to commercial software with dedicated responsibility to software engineers, members of OSS projects are usually volunteers [1] with motivations that cover a broad spectrum of personal goals and interests, e.g., sharpening their programming skills and gaining experience, following fellow peers, networking with the OSS community members, or simply supporting free open software projects [2]. The amount of participation and commitment by the volunteering developers are crucial factors for the success of OSS projects [3]. As a matter of fact, there exist over two million OSS projects available online, however only few are successful. Krishna et al. [21] conclude that the failure of projects to attract more developers may prove to be the single important reason for the death of OSS projects. This suggests the importance of attracting the developers to participate in OSS projects.

In order to know what attract the developers to participate in OSS projects, several works study the motivations of developers. Lee et al. [19] study the motivations of developers to choose between the commercial and the OSS project and found that highly skillful developers find participating in OSS projects more rewarding than commercial software projects. Also, Roberts et al. [20] find that developers’ status and reputation is one of the important motivations for them to contribute to OSS projects. Moreover, Casaló et al. [5] collect information via questionnaires and study the correlation between developers commitment to their OSS project
and the reputation of those projects. They conclude that OSS reputation have an indirect effect on the commitment of collaborators. Besides motivations, in Chapter 3 it is found that developers tend to select projects with certain features (e.g., programming language, complexity, popularity, team size, etc) based on their participation experience. Also, it has been found that around 40% of developers in a team are associated via work affiliation, and around 30% belong to the same city (location). We exploit these findings in our proposed solutions for recommending projects to developers in OSS communities.

In this work, we propose several models to recommend OSS projects to developers in an effort to increase the developers’ participation by guiding them to the projects of their interest. There exist several works in literature that introduce recommender systems for software artifacts, such as recommending software components [39, 40], source code [41, 42, 43], software functions [44], software libraries [45]. However, there are no recommender systems for OSS projects capable of directing developers to the projects of their interest.

In our proposed solutions, we consider two aspects of OSS projects (i) the projects’ content and (ii) the peer influence. For the first aspect, projects’ content, we realize that projects are different than any typical consuming item (e.g., books, movies, software). A consuming item has fixed content and features that is meant to be used by a consumer, whereas a project is a dynamic continues event that involves participation of many people. Therefore, describing a project using static features may not give the right description of the project status. This is similar to a scenario of inviting person $x$ to party $y$. From $x$’s perspective, it is not enough to know the type of party $y$ (e.g., birthday or social gathering party). Other information, such as the place, the time, how many are going and who is going may be the key that triggers $x$ to join party $y$. Similarly, in recommending open projects to developers, it may not be sufficient to match the field of the OSS, programming languages and packages to use. Therefore, we take into account a number of dynamic factors (e.g., team size, popularity, complexity, etc) in our project recommender models.

The second aspect considers the peer (or social) influence between the developers. We consider three types of peer influence, channeled via project collaboration, work affiliation and location affiliation. In OSS community, developers spread over a wide range of companies and institutes, as well as different locations. As mentioned above, developers tend to follow other developers with similar affiliations. Moreover, in [24, 11, 10, 46], it is found that developers tend to collaborate with other developers whom they have collaborated with before. The developers’ collaboration history and work and location affiliation are all considered in our proposed recommender models. Here we note that, in this work, the term “affiliation” has a broader meaning than just a work affiliation, but rather we use it to mean association to a common entity as we explain next.

In this chapter, we propose five project recommender models, including four memory-based models and one probabilistic model. In the memory-based models consist of one content-based model and three Collaborative Filtering (CF) models. The Project Content-Based Filtering (PCBF) utilizes the dynamic project features to match the query developer’s previous projects with the potential projects based on the developer’s level of contribution on his/her previously
participated in projects. The CF models are proposed based on distinct similarity functions. The similarity measure between the developers is the essence of the CF model, therefore, we exploit every possible similarity trait between developers in order to improve the recommender system. The first similarity function is based on the common project collaboration between developers and is called the Project CF based on Project Affiliation (PCF-PA). The second is based on the similar work affiliations (e.g., working for the same company or institute) and is called the PCF based on Work Affiliation (PCF-WA). The third is based on the similar location affiliation, i.e., city, and is called the PCF based on Location Affiliation (PCF-LA). Finally, we propose a probabilistic model that combines all the aforementioned factors, namely the content preference and the affiliated peer influence. The probabilistic model, called the Developer Affiliation and Preference Model (DAPM), is proven to be superior over the memory-based models for recommending OSS projects to new members to the OSS community.

We evaluate the proposed models on a real longitudinal dataset collected from GitHub.com, a fast-growing online OSS community. The evaluation is conducted on two datasets, one is collected on December of year 2011 and the second on December of year 2012, where the 2011 dataset is used to train the models and the 2012 dataset is used as a test data. The developers new project participations (joining) in the 2012 dataset is the ground truth for our recommender models performance evaluation. The experiment shows promising results for recommending OSS projects to developers who do make real collaborations to projects.

In this work, we have made a number of contributions:

- We introduce and utilize the dynamic features of open projects into a content-based recommender model that recommends projects with similar development processing and status level to a query developer.
- We introduce and exploit three developers’ similarity aspects for collaborative recommendation in three different collaborative filtering models.
- We introduce a probabilistic model that combines the developers’ project content preferences and affiliation influence aspects in one model.
- We evaluate our models on a real longitudinal dataset from Github.com which consists of more than a million contributors and more than two millions open projects. Further, we evaluate the proposed models on four levels of history data availability.

The rest of the chapter is organized as follows. Section 5.2 reviews some related works. Section 5.3 introduces some preliminaries and defines the project recommendation problem. Section 5.4 introduces the proposed models. Section 5.5 details the experiment methodology and evaluates the proposed models. Finally Section 5.6 concludes the chapter.
5.2 Related Works

There exist a variety of recommender systems for software development that involves many process phases of a software. In this section we mention some of these recommender systems. Ichii et al. [39] introduce a recommender system for software components based on a CF model that uses a browsing history of components as a similarity measure between developers. Utilizing similarity based on browsing similar components is not a reliable technique since merely browsing does not mean likeness of a component in many cases. McCarey et al. [40] build a recommender system for software components based on CF model. However they consider the components being used by a developer as users such that recommendation is based on similarity between components and not actual developers. In contrast, our work considers similarity between the developers instead of their software artifacts archive. Besides software component recommendation, there exist several works on source code recommendation [41, 42, 43], software functions recommendation [44], and software libraries recommendation [45]. However, most previous works are for software artifacts and not for project recommendation. Software artifacts can be considered as typical consuming items which are different than a project developing event. The most similar technical problem as ours is recommending events to people as in [47]. However, open projects are events that continue for a longer time and require more commitment. Hence it is a more challenging problem.

5.3 Project Recommendation Problem

In this section, we introduce some preliminaries and describe the problem formulation for project recommendation in the OSS communities.

Let $G(X, E)$ be an undirected social graph representing the connection among developers in an OSS community, where $X = \{x_1, ..., x_n\}$ is the set of active developers in the community and $E$ represents the affiliation relationships among the developers. Here each $e \in E$ is a hyper edge representing a type of affiliation relationship of three types (i) collaboration, (ii) work or (iii) location affiliation. Also, let $Y = \{y_1, ..., y_m\}$ be the set of all projects and $A = \{a_1, ..., a_k\}$ be the set of all project’s attributes. Every project $y \in Y$ consists of a set of developers. Thus, we use $X_y$ to denote the set of developers in a project $y$, i.e., $X_y \subseteq X$. Moreover, $A_y$ denotes the set of attribute values for a project $y$. Also, given a developer $x$, $Y_x = \{y_1, ..., y_k\}$ denotes the set of projects that $x$ participates in. $F_x$ is the set of collaborators with $x$. Finally, $W = \{w_1, ..., w_k\}$ is the set of work affiliations and $L = \{l_1, ..., l_k\}$ is the set of location affiliations in OSS community. $W_y$ or $L_y$ donates the set of work or location affiliations in project $y$.

**Problem Formulation.** Given a query developer $x_q \in G$ and a set of OSS projects $Y = \{y_1, ..., y_n\}$, a project recommender finds top-$n$ OSS projects $Y_{top-n}$, where $Y_{top-n} \not\subseteq Y_{x_q}$, to $x_q$ that fits $x_q$’s interests and ranks the projects based on the likelihood of $x_q$ to participate in the project.
5.4 The Proposed Solutions

We introduce five models to solve the project recommendation problem. The first solution applies a content based filtering approach where similar projects to the projects of the query developer is recommended. The second solution uses similar mind developers to the query developer to rate the projects and recommend them to the query developer. The second solution has three variations based on different types of similarity models between developers. The first variation uses a similarity model based on the number of collaborations between developers. The second variation uses the work affiliation similarity, and the third variation uses the location affiliation similarity. Finally, the fifth solution uses a probabilistic model that identifies a developer’s preferences on selecting projects of specific features and considers a developer’s affiliation influence on selecting a project with certain collaborators. The following sections detail each solution.

5.4.1 Project Content-Based Filtering with Dynamic Features

A direct way to recommend OSS projects to a developer is to find similar projects to the ones a developer participated in. The content similarity function is the core part of the Content-Based Filtering (CBF) approach. OSS projects contain many informations detailing a project’s field, purpose, skills or activities. Hence the difficulty is to know which content information produce the most accurate prediction for recommendation. The OSS project is not like a consuming product item (e.g., books, movies, software, etc.), rather it is an activity indulging product. Therefore, it is important to find the activity characteristics of a project to match it with those of a developer’s type of project activities.

A project’s activities can be described using what we call the Dynamic Project Features (DPF). They are called dynamic because their value changes, e.g., team size, popularity, complexity, etc. This is in contrast to static features such as a project topic and programming languages used. We define three main categories of attributes to capture the dynamic nature of a project. The attributes are (i) the popularity level, measured by the number of followers or “watchers” to a project and the team size, (ii) the activity level, measured by the number of open issues and forks\(^1\), and (iii) the complexity level, measured by the programming code size (by line-of-code count). The dynamic features are selected based on our analytical findings in Chapter 3. Those features capture the developers’ preference on the type of projects they participate in. For example, many developers look for popular projects to participate in for broader social connectivity, while some other developers tend to involve in projects with high or low activities depends on the pace of their responses and activeness. Also, the size of the code gives a hint on how large and complex a project is. Moreover, developer may search for certain complexity to improve their experience.

We also add two static project features including (i) the primary programming language used, and (ii) the owner of the project type whether a user or organization. The owner type can find whether the developer likes to collaborate in projects owned by individuals or organizations.

\(^1\)A fork is a copy of a project repository used for experimenting purposes.
There are two ways to assign weights to the attributes. The first method is to assign equal weights to all attributes considering equal importance for each one. The second method assigns different weight to every attribute depending on its importance. We adopt the Goodall model [48] to measure the similarity between projects. The Goodall model assigns higher weight to a matched attribute if its value is infrequent than if the value is frequent, e.g., consider the programming languages Java and Smalltalk where Java is considered more popular language that Smalltalk, then a match on Smalltalk would weight more than a match on Java since the former is more descriptive value than the latter. Since this is not the main focus of this work, we refer the reader to [49] for details on the Goodall similarity measure.

The Project Content-Based Filtering (PCBF) model computes the expected contribution of a query developer $x_q$ in project $y \in Y$ as in Eq. (5.1).

$$E(\text{cont}_{x_q,y}) = \kappa \sum_{\forall y_j \in Y_{x_q}} \text{sim}(y,y_j) \times \text{cont}_{x_q,y_j}$$  \hspace{1cm} (5.1)

where $\text{sim}(y,y_j)$ is the similarity function for $y$ and a previously participated project $y_j$ by developer $x_q$ based on the attributes mentioned above, $\text{cont}_{x_q,y_j}$ is the contribution of $x_q$ in $y_j$, and $\kappa$ is a normalizing factor such that $E(\text{cont}_{x_q,y}) = [0, 1]$. A developer contribution to a project is measured by the number of commits to a project. A high number of commits indicates a high interest from a developer to the project. On the other hand, a low number of commits indicates a low interest from a developer to the project. $\text{cont}_{x_q,y_j}$ is discretized to a six level of contributions based on the number of commits a developer submits to $y_j$ as shown in Table 5.1.

**Dynamic Feature Values and Similarity Function:** The similarity function is computed as in Eq. (5.2).

$$\text{sim}(y,y_j) = \sum_{v_{a_k} \in V_{a_k}} w_k \times v_{a_k}$$  \hspace{1cm} (5.2)

where $v_{a_k} \in V_{a_k}$ is an attribute value and $V_{a_k}$ is the set of attribute values of attribute $a_k$. $v_{a_k}$ is based on the probability of a value occurring in $V_{a_k}$ distribution of value occurrence. A less frequent value is given more weight than more frequent ones. $w_k$ is the attribute weight in the function such that $\sum w = 1$ and $\text{sim}(y,y_j) = [0, 1]$.

The dynamic and static attributes contain real and nominal values. In order to facilitate the attribute matching process we convert the attributes of real values to be nominal values such that each range of real values fall in one category. Table 5.1 presents each attribute and its nominal values with each category followed by the range of real values representing the category (e.g., team size of 2 to 10 members is under a small team size category). The distribution of real values over the nominal categories are based on power-low distribution since all the dynamic attributes follow a power-law distribution as presented in Chapter 3.

---

2A commit is the update procedure a developer performs to upload his/her edits to a project’s files.
Table 5.1. Project Attribute Nominal Values.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Nominal value [range]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Watchers</td>
<td>not popular [0], least popular [1], moderate [2, 20],</td>
</tr>
<tr>
<td></td>
<td>popular [21, 100], very popular &gt;100</td>
</tr>
<tr>
<td>Team Size</td>
<td>very small [1], small [2, 10], moderate [11, 100],</td>
</tr>
<tr>
<td></td>
<td>large [101, 1000], very large &gt;1000</td>
</tr>
<tr>
<td>Open Issues</td>
<td>not active [0], least active [1, 10], moderate [11, 100],</td>
</tr>
<tr>
<td></td>
<td>active [101, 1000], very active &gt;1000</td>
</tr>
<tr>
<td>Forks</td>
<td>not active [0], least active [1, 10], moderate [11, 100],</td>
</tr>
<tr>
<td></td>
<td>active [101, 1000], very active &gt;1000</td>
</tr>
<tr>
<td>Code Size</td>
<td>not known [0], small [1, 100], moderate [101, 1000],</td>
</tr>
<tr>
<td></td>
<td>large [1001, 10000], very large &gt;10000</td>
</tr>
<tr>
<td>Contribution (Commits)</td>
<td>not active [1], least active [2, 10], moderate [11, 40],</td>
</tr>
<tr>
<td></td>
<td>least moderate [41, 100], active [101, 1000], very active</td>
</tr>
<tr>
<td></td>
<td>&gt;1000</td>
</tr>
</tbody>
</table>

Static Features

<table>
<thead>
<tr>
<th>Prog. Lang.</th>
<th>93 programming language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner Type</td>
<td>User, Organization</td>
</tr>
</tbody>
</table>

5.4.2 Project Collaborative Filtering Recommender

Collaborative Filtering (CF) has proven to be very successful in recommender systems. Unlike the content-based filtering approach, CF can broaden the scope of project types being recommended to include projects that a query developer might be interested in and has not been tried before. Given a user/item scenario, the essence of CF is to find similar minded people to the query user and to consider the items selected by the similar minded people for recommendation to the query user considering the rating given to those items as a measure of preference. CF assumes that the similar minded people have similar taste on selecting items and evaluating them. In OSS community, the items are open projects and users are active developers. Therefore, similarity between the developers may not be based on personal taste only but may involve certain affiliations as well. In the following sections, we introduce three types of affiliations that define the similarity between developers, namely (i) project affiliation, work affiliation and location affiliation.

5.4.2.1 Project Collaborative Filtering with Project Affiliation

Developers participating in similar projects can indicate similar taste on selecting projects. We define project affiliation between developers as the number of project collaborations between them. Two developers are considered collaborators if they co-participated in the same project. The more projects they co-participate in, the more similar they are, and vice versa.

The Project Collaborative Filtering with Project Affiliation similarity (PCF-PA) model computes the expected contribution of a query developer $x_q$ in project $y \in Y$ as in Eq. (5.3).
\[ E(\text{cont}_{x_q,y}) = \kappa \sum_{\forall x_i \in F_{x_q}} \text{sim}(x_q, x_i) \times \text{cont}_{x_i,y} \]  

(5.3)

where \( \text{sim}(x_q, x_i) \) is the similarity function between \( x_q \) and a collaborator developer \( x_i \in F_{x_q} \) where \( F_{x_q} \) is the set of collaborators with \( x_q \). \( \text{cont}_{x_i,y} \) is the contribution of \( x_i \) in \( y \), and \( \kappa \) is a normalizing factor such that \( E(\text{cont}_{x_q,y}) = [0,1] \). \( \text{cont}_{x_i,y} \) is discretized to a six level of contributions based on the number of commits a developer \( x_i \) submits to \( y \). The similarity function is computed as in Eq. (5.4).

\[ \text{sim}(x_q, x) = |Y_{x_q} \cap Y_x| \]  

(5.4)

which is the cardinality of the common projects between \( x_q \) and \( x \).

### 5.4.2.2 Project Collaborative Filtering with Work Affiliation

The OSS community includes many work affiliations. According to the results in Chapter 3, on average \%53 to \%35 (depending on the team size) of developers in a project belong to one company or institute. This gives an indication that developers from the same work entity do often participate in the same projects. Therefore, we use the work affiliation as a similarity measure between developers. We define work affiliation similarity between developers as the number of common work affiliations between the developers. Two developers are similar if they belong to the same affiliation. The more common affiliations they belong to, the more similar they are, and vice versa.

The Project Collaborative Filtering with Work Affiliation similarity (PCF-WA) model computes the expected contribution of a query developer \( x_q \) in project \( y \in Y \) as in Eq. (5.5).

\[ E(\text{cont}_{x_q,y}) = \kappa \sum_{\forall x_i \in X_{W_{x_q}}} \text{sim}(x_q, x_i) \times \text{cont}_{x_i,y} \]  

(5.5)

where \( \text{sim}(x_q, x_i) \) is the similarity function between \( x_q \) and a collaborator developer \( x_i \in X_{W_{x_q}} \) where \( X_{W_{x_q}} \) is the set of developers having the same work affiliations as \( x_q \). \( \text{cont}_{x_i,y} \) and \( \kappa \) are similar to Eq. (5.3). The similarity function is computed as in Eq. (5.6).

\[ \text{sim}(x_q, x) = |W_{x_q} \cap W_x| \]  

(5.6)

where \( W \) is the set of work affiliations for \( x_q \) and \( x \). Then \( \text{sim}(x_q, x) \) is the cardinality of the common work entities between \( x_q \) and \( x \).

### 5.4.2.3 Project Collaborative Filtering with Location Affiliation

The OSS developers are from all over the world. Most of the development communication and updates is done remotely. However, according to the results in Chapter 3, on average \%57 to \%17 (depending on the team size) of developers in a project are from the same locations/regions. This
Figure 5.1. The Developer Affiliation and Preference Model (DAPM).

gives an indication that developers from the same location/region do often participate in the same projects. Therefore, we use the location affiliation as a similarity measure between developers. We define location affiliation similarity between developers as a binary value to indicate whether the two developers belong to the same location or not. Two developers are similar if they belong to the same location. In this work, we limit the location scope to be a city where a developer declares he/she is located in.

The Project Collaborative Filtering with Location Affiliation similarity (PCF-LA) model computes the expected contribution of a query developer $x_q$ in project $y \in Y$ as in Eq. (5.7).

$$E(\text{cont}_{x_q,y}) = \kappa \sum_{x_i \in X_{l_{x_q}}} \text{sim}(x_q, x_i) \times \text{cont}_{x_i,y}$$

(5.7)

where $\text{sim}(x_q, x_i)$ is the similarity function between $x_q$ and a collaborator developer $x_i \in X_{l_{x_q}}$ where $X_{l_{x_q}}$ is the set of developers having the same location affiliation $l$ as $x_q$. $\text{cont}_{x_i,y}$ and $\kappa$ are similar to Eq. (5.3). The similarity function is computed as in Eq. (5.8).

$$\text{sim}(x_q, x) = l_{x_q} \land l$$

(5.8)

where $l$ is the location for $x_q$ and $x$. Then $\text{sim}(x_q, x)$ is a binary value indicating whether $x_q$ and $x$ belong to the same location or not.

5.4.3 The Developer Affiliation Influence and Project Preference Probabilistic Model

We realize that there exist two important aspects that govern a developer’s selection of projects. The first aspect concerns a developer preferences on selecting projects with certain attributes, and the second aspect concerns the influence of different affiliation ties (i.e., project, work or location affiliation) between collaborators on attracting a developer to join a project. These two aspects are addressed separately in the memory-based models introduced in the previous sections. The memory-based models usually suffer from the cold-start problem which are well
known challenge for new joining members with limited background history since memory-based models depend mainly on the data availability. Meanwhile, probabilistic models often mitigate the shortcoming of memory-based approaches by learning the parameters of the models to be used on the prediction. Moreover, we need a recommender model capable of combining all the important factors that influence a developer’s selection of a project. In this section, we introduce a probabilistic model that combines the developers’ project preferences and peers’ affiliation influence in one model which we call the Developer Affiliation and Preference Model (DAPM).

We adopt a probabilistic model that is based on the notion of aspect model that models individual preferences as a convex combination of preference factors. A latent variable in the aspect model is associated with each observation (record) which allows classifying each selection by a developer in its appropriate class. This creates a great amount of flexibility on classifying individuals’ preferences. In addition, probabilistic generative models tend to work well with scarce data since it can classify records based on the learned distributions of the model’s parameters. In an OSS community, a history record \(\langle x, y, A_y, f, w, l \rangle \in H\), where \(H\) is the set of data records history, consists of a developer \(x\) participating in project \(y\) with attributes \(A_y\), where the attributes \(A_y = \{a_1, ..., a_m\}\) are similar to the attributes discussed in Section 5.4.1, and collaborating with “friend” \(f\) (\(f\) is a collaborator with \(x\) in \(y\)) whose work affiliation is \(w\) and location affiliation is \(l\). Figure 5.1 shows the DAPM graphical model. The right hand side considers the developers’ preferences on joining projects with certain attributes, and the left hand side considers the different affiliation ties influence of collaborators on developers to join a project. The main goals of the model are to (i) differentiate developers’ preferences on selecting certain project attributes by introducing the latent variable \(z\), (ii) identifying developers’ likelihood of joining friends with certain work and location affiliation, and (iii) combining the preference and affiliation aspects in one prediction model.

Given a query developer \(x_q\) and a set of OSS projects \(Y\), we want to estimate for each \(y_j \in Y\) the probability for \(x_q\) to participate in \(y_j\) (i.e., \(P(y_j|x_q)\) or simply \(P(y|x)\)). Note that \(P(y|x)\) can be computed as

\[
P(y|x) = \frac{P(x, y)}{P(x)} \propto P(x, y)
\] (5.9)

In Figure 5.1, \(Z = \{z_1, ..., z_k\}\) is a latent variable where \(z\) represents a preference class of \(x\). \(y \in Y\) is an open project and \(A_y\) is the set of attributes of project \(y\). \(f\) represents a collaborator of \(x\) such that \(f \in F_x\) is the set of collaborators with \(x\). \(w \in W\) and \(l \in L\) is the work and location affiliations, respectively independent of \(f\). The model assumes that \(x, y\) and \(A_y\) are independently conditioned on \(z\) and \(f, w, l\) and \(z\) are independently conditioned on \(x\). The probability model can be simply written as

\[
P(x, y) = \sum_{z \in Z} \sum_{w \in W_y} \sum_{l \in L_y} \sum_{f \in F_x \cap X_y} P(x, z, y, A_y, f, w, l)
\] (5.10)

where \(W_y\) is the set of work affiliations of developers in \(y\), \(L_y\) is the set of location affiliations of
developers in $y$ and $f$ belongs to $F_x$ which is the set of $x$’s collaborators (friends) in $y$ ($X_y$ is the set of developers in $y$). Then the joint probability distribution over all factors is

$$P(x, z, y, A_y, f, w, l) = P(x)P(z|x)P(y|z)P(A|z)P(f|x)P(w|x)P(l|x)$$ \hspace{1cm} (5.11)

where $P(z|x)$ is the probability of $x$ falling in class $z$, and $P(y|z)$ and $P(A|z)$ are class-conditional multinomial distributions giving the probabilities of class $z$ selecting $y$ with attributes $A = \{a_1, ..., a_m\}$. Note that $P(A|z) = P(a_1|z), ..., P(a_m|z)$ where $m$ is the number of attributes. Also, $P(f|x)$ is the probability of $x$ selecting $f$ as a collaborator, and $P(w|x)$ and $P(l|x)$ are probabilities of $x$ following a collaborator with work affiliation $w$ and location affiliation $l$, respectively. $P(x) = 1/|n|$ where $n$ is the total number of developer in the community and hence it is the same for every $x$ which we ignore in later equations. We note that only parameters $P(z|x)$, $P(y|z)$ and $P(A|z)$ are need to be learned while parameters $P(f|x)$, $P(w|x)$ and $P(l|x)$ are computed directly from the dataset $H$.

Overall, the DAPM model gives the probability of developer $x$ selecting project $y$ with attributes $A$ and following collaborator $f$ with work and location affiliations $w$ and $l$, respectively. Comparing this solution to a user-item recommendation scenario, one can think of $A$ as a content of an item and $f$ as a friend’s influence with similarity with $x$ on $w$ and $l$. Hence, DAPM models the developer’s preference on selecting projects with certain attributes and following collaborators with certain affiliations.

To learn the $P(z|x)$, $P(y|z)$ and $P(A|z)$ parameter (presented in the right side in Figure 5.1), we use the Expectation Maximization (EM) algorithm to learn the model parameters from the set $(x, y, A_y, f, w, l) \in H$. However, we only use the records $(x, y, A_y)$ to train parameters $P(z|x)$, $P(y|z)$ and $P(A|z)$ and use the records $(x, f, w, l)$ to learn parameters $P(f|x)$, $P(w|x)$ and $P(l|x)$.

Our model parameter learning algorithm is based on the idea of maximizing the log-likelihood of $L(\theta)$.

$$L(\theta) = \sum_{(x,y) \in H} \log(P(x,y|\theta))$$ \hspace{1cm} (5.12)

where $\theta$ denotes the learned model parameters, i.e., $P(z|x)$, $P(y|z)$ and $P(A|z)$. The EM algorithm iterates between the E-step and M-step. In the E-step, the algorithm calculates the posterior probabilities of every latent variable $z \in Z$ based on the current estimates of the parameters as in Eq.(5.13).

$$P(z|x, y, A) \propto \frac{P(z|x)P(y|z)P(A|z)}{\sum_{z \in Z} P(z|x)P(y|z)P(A|z)}$$ \hspace{1cm} (5.13)

In the M-step, model parameters are computed to maximize the expected log-likelihood in the E-step as below.
\[ P(z|x) = \frac{\sum_{\langle x,y',A' \rangle \in H} \sum_{z \in Z} P(z|x',y',A')} {\sum_{\langle x,y',A' \rangle \in H} \sum_{z' \in Z} P(z|x',y',A')} \]

\[ P(y|z) = \frac{\sum_{\langle x',y',A' \rangle \in H} \sum_{z \in Z} P(z|x',y',A')} {\sum_{\langle x',y',A' \rangle \in H} \sum_{z \in Z} P(z|x',y',A')} \]  

\[ P(A|z) = \frac{\sum_{\langle x',y',A' \rangle \in H} \sum_{z \in Z} P(z|x',y',A')} {\sum_{\langle x',y',A' \rangle \in H} \sum_{z \in Z} P(z|x',y',A')} \]

where \( \sum_{z \in Z} P(z|x) \), \( \sum_{y \in Y} P(y|z) \) and \( \sum_{a_i \in A_i} P(a_i|z) \) are all 1. Note in Eq. (5.14) the variables with a prime means counting every value of this variable. Iterating between the E-step and M-step, the EM algorithm improves the model parameters on each iteration until they converge to a local log-likelihood maximum.

For parameters \( P(f|x) \), \( P(w|x) \) and \( P(l|x) \), we learn them directly from the data records \( H \) as below.

\[ P(f|x) = \frac{|h|_{\forall \langle x,y',f,w',l' \rangle \in H}}{|h|_{\forall \langle x,y',f',w',l' \rangle \in H}} \]

\[ P(w|x) = \frac{|h|_{\langle x,f',w',l' \rangle \in H}}{|h|_{\langle x,f',w',l' \rangle \in H}} \]

\[ P(l|x) = \frac{|h|_{\langle x,f',w,l \rangle \in H}}{|h|_{\langle x,f',w,l' \rangle \in H}} \]

where \( h \in H \) and \( |h| \) is the number of records belonging to the subscript constraints. Also, \( \sum_{f \in F} P(f|x) \), \( \sum_{w \in W} P(w|x) \) and \( \sum_{l \in L} P(l|x) \) are all 1. Finally, having the DAPM model parameters, we can compute \( P(y|x) \).

### 5.5 Experiment and Evaluation

We evaluate the proposed models on a longitudinal data set collected from Github.com on a span of one year from December of 2011 to December of 2012. Github.com is one of the most fast-growing OSS online repository. Github.com, hosting over two million OSS project and over one million contributor, is unique since it provides a variety of project’s and developer’s information. Table 5.2 summarizes the data set collected from Github.com. The statistics belong to the latest dataset. “Project Participations” in the table is the total number of developer-project records.

In the experiment, we consider only the collaborators with account in Github.com since they are identifiable by a unique ID, whereas anonymous contributors are hard to track and know their
### Table 5.2. Collected Data Statistics

<table>
<thead>
<tr>
<th>Data</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developers</td>
<td>1,034,996</td>
</tr>
<tr>
<td>Developers with account</td>
<td>652,040</td>
</tr>
<tr>
<td>Collaboration Edges</td>
<td>37,543,480</td>
</tr>
<tr>
<td>Projects</td>
<td>2,332,749</td>
</tr>
<tr>
<td>Project Participations</td>
<td>9,268,644</td>
</tr>
<tr>
<td>Work Affiliation</td>
<td>31,019</td>
</tr>
<tr>
<td>Location Affiliation</td>
<td>6,474</td>
</tr>
</tbody>
</table>

participation history.

#### 5.5.1 Experiment Methodology

The main goal of the proposed models is to recommend potential projects to developers. Hence, the experiments aim to test the models’ recommendation accuracy and quality. In order to achieve the goal, we use a longitudinal dataset of one year span. We tag the earlier dataset as $D_1$ and the later dataset as $D_2$. First, we collect the developers that exist in both datasets to make sure that developers participating in projects in $D_1$ also exist and participate in projects in $D_2$. This set of developers is $X$. Second, we randomly select developers as queries where each $x_q \in X$ has at least 5 new project participations in $D_2$. Finally, we train the models using information in $D_1$ and we test the models using $D_2$. Given $x_q$, let $Y_{x_q}$ be the set of projects that $x_q$ has joined in $D_2$, then the experiment is to test whether the models can recover any or every $y \in Y_{x_q}$ or not. Every recommended $y \in Y_{x_q}$ to $x_q$ is considered a hit. We emphasize that the experiment tests the models against real developers’ selection of projects over a span of one year.

In an effort to conduct a complete evaluation, we add three variations of the proposed models in Section 5.4. The first variation combines the three types of affiliations (i.e., project, work and location) linearly in a CF model which we call Project CF based on all affiliations or (PCF-all). The second variation considers testing DAPM model with the project’s content preference factor without the peer affiliation factors in which we call the Developer Preference Model (DPM). Finally, the third variation tests DAPM model with the peer affiliation factors without the project’s content preference factor in which we call the Developer Affiliations Model (DAM). DPM represents the right side of Figure 5.1 and it is the probabilistic model version of PCBF, while DAM represents the left side of Figure 5.1 and it is the probabilistic model version of PCF-all.

We evaluate the quality of the recommendation results of each model based on (i) how many hits can they find? (ii) how many hits can they find from the set $Y_{x_q}$? (iii) how well are the hits ranked in the top-20? and (iv) how good (suitable) are the found hits to the query developers? We use the precision metric for (i), the recall metric for (ii), the normalized Discounted Cumulative Gain (nDCG) [38] metric for (iii) and the average amount of contribution that $x_q$ exerted in each hit project found for (iv). The precision is the fraction of relevant records over the total
number of records retrieved. As mentioned in the previous section, we consider a project \( y \) as relevant (hit) if it is in \( Y_{x_q} \). The \textit{recall} is the fraction of relevant records over the total number of relevant records, which in our case is \( Y_{x_q} \). We emphasize that finding the projects in \( Y_{x_q} \) is not the purpose of the recommender system proposed, but only a method to evaluate its effectiveness in recommending projects that \( x_q \) indeed joined (participated in) in the future.

The nDCG evaluates relevant records retrieved based on its relevancy level and its position on the rank, i.e., a system ranking highly relevant records in high positions is considered better than a system ranking highly relevant records in low positions. Under the context of this work, relevancy is estimated by the amount of contribution a developer exerts in a hit project, i.e., the number of commits. Meaning, a project found in which \( x_q \) contributed more is considered more relevant than a project found in which \( x_q \) contributed less. To measure nDCG, we develop a multi-level relevancy model based on the percentage of contribution for a developer in a project as follows: \( \geq 50\% \) is highly relevant, \( [20\%, 50\%) \) is moderate relevant, \( [5\%, 20\%) \) is relevant, and \( \leq 5\% \) is least relevant. We emphasize that the nDCG is different from \textit{precision} as the latter only counts the number of relevant records in the retrieved set, while, nDCG evaluates the relevancy and ranking position of these relevant records. Moreover, a model with high nDCG means that it can find high relevant projects to \( x_q \) in high ranks than a model with lower nDCG.

For evaluating question (iv), we use the percentage amount of contribution that \( x_q \) exerted in each hit project found. A high contribution percentage means a high suitability and hence high responsiveness from the developer to select the hit project. The rational behind this metric is to evaluate the capability of a model to retrieve high suitable projects to \( x_q \). We call this metric \textit{suitability}. We emphasize, that \textit{suitability} is different than nDCG since \textit{suitability} measures the amount of relevancy regardless of ranking while nDCG measures the ranking but not focusing on the amount of relevancy.

Finally, to evaluate the strength and weakness of each model on different data availability (sparsity) scenarios, we divide the query developers set \( X_Q \), where \( \forall x_q \in X_Q \), based on the developers’ number of participation in projects. We divide \( X_Q \) into four sets: \( [2, 10) \), \( [10, 50) \), \( [50, 200) \) and \( \geq 200 \). For example, developers in \( [2, 10) \) has participation records, \( \langle x, y \rangle \), between 2 to 9 project participations. Set \( [2, 10) \) resembles the most scarce data set, and set \( \geq 200 \) resembles the most abundant dataset. For DAPM, we train the model with \( |Z| = 6 \) since we consider 6 levels of developer preferences based on the levels of contributions as in Table 5.1. In the following section, we present the results and discuss and analyze the findings.

### 5.5.2 Evaluation of Project Recommendation on an Individual Case

In this section, we show the impact of our proposed models on recommending projects to a candidate individual. Later in the next section, we show the aggregate results of multiple query developers. Here we randomly select a query developer and recommend OSS projects to him/her using every proposed model and evaluate the quality of each model results. A random developer with ID (873785) is selected and noted by \textit{Alice}, in which she is associated with a work affiliation
Table 5.3. Top-10 projects recommended for Alice.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>y</td>
<td>0.065</td>
<td>A</td>
<td>y</td>
<td>0.065</td>
<td>n</td>
<td>n</td>
<td></td>
<td>C</td>
<td>y</td>
<td>0.303</td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>B</td>
<td>y</td>
<td>2.0</td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>y</td>
<td>0.065</td>
<td>B</td>
<td>y</td>
<td>2.0</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>y</td>
<td>0.065</td>
<td>B</td>
<td>y</td>
<td>2.0</td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td>n</td>
<td>n</td>
<td></td>
<td></td>
<td>n</td>
<td></td>
</tr>
</tbody>
</table>

with ID (11306) and a location with ID (1496).

Table 5.3 shows the top-10 projects recommended for Alice by each recommender model. For each model, the table shows the recommended project ID (given by a capital letter for ease of comparison), whether it is a hit or not and the percentage of contribution of Alice (the query developer) in this project if it is a hit. PCBF found one project, A, in low rank in which Alice had only 0.065% of the total contribution of this project. A very low contribution means a very low suitability or relevancy to the query issuer. PCF-PA found 2 projects, A and B, where project A is ranked first. Project B has contribution of 2% from Alice which is considered high, however, it is ranked low. PCF-WA could not find project B in top-10 results and PCF-LA could not find any hit in top-10. DAPM, with the advantage of combining the project content and the peer affiliation influence factors, was able to find 3 match projects in top-4 ranks where it introduced a new project hit in the first rank which has contribution of 0.303% from Alice. Also, DAPM ranked project B in 4th instead of 9th rank as in PCF-PA. This individual case project recommendation example shows that DAPM can (i) find more hits, (ii) rank hits higher and (iii) find hits (projects) that are suitable to a query developer and rank them high. In addition, Figure 5.2 shows the result of each metric when using every project recommender model. It is obvious that DAPM is superior in all metrics. In the next sections, we conduct the experiment on multiple queries (query developers) and discuss the findings.

5.5.3 Aggregate Results and Evaluation

This evaluation investigates the performance of the models on different history records (h.r.) levels. We evaluate the precision, recall, nDCG and suitability metrics for 8 models where 5 are memory-based models (i.e., PCBF, PCF-PA, PCF-WA, PCF-LA and PCF-all) and 3 are probabilistic models (i.e., DPM, DAM and DAPM).

Precision: Figure 5.3 presents the average precision of the top 20 projects found corresponding to different sets of h.r. [2, 10), [10, 50), [50, 200) and ≥ 200. Figure 5.3(a) shows clearly the superiority of DAPM over the memory-based models and DAM and DPM. Also, we notice that DPM performs better than PCBF and DAM performs better than PCF-all. This result shows the advantage of the probabilistic models over memory-based models in the case of very limited
data history. Also, it shows the advantage of combining the project content and the developers’ affiliations aspects in one probabilistic model. Comparing the PCF models, we notice that PCF-LA has the least precision while PCF-PA and PCF-WA exchange performance superiority and PCF-all is in the middle of PCF-PA and PCF-WA. This result shows that the project and work affiliations are more important than location affiliation in the case of new or low active members (developers with few history records).

In Figure 5.3(b), we still notice the superiority of DAPM over other models. Moreover, the PCF models improve the previous results with the superiority of PCF-PA and PCF-all over PCF-WA this time. At this stage of developer activeness (with h.r. [10, 50]) work affiliation has less importance than project affiliation.

In Figure 5.3(c), DAM takes the lead over DAPM which shows the importance of affiliation ties of project content preference. Also, this is visible by comparing the results between DAPM and PCF-PA and PCF-LA. Moreover, surprisingly, the location affiliation in PCF-LA has very high precision compared to the previous two results of less data records. This shows the shift of developers’ concern from work affiliation to location affiliation when they become more active or be a senior in the community. Finally, we notice the DPM has comparable precision with
PCF-WA and PCF-all which shows the impact of project content preference on active/senior developers.

Finally, Figure 5.3(d) shows the superiority of PCF-PA over probabilistic models due to the very abundant data records. We also, notice the improvement of DPM which has much higher precision than PCF-WA and PCF-LA. Again this shows that developers with more experience tend to select projects with similar features.

Recall: Next, we evaluate the recall performance. In Figure 5.4, DAPM has the highest recall only in the rarest data records set, while PCF-all has the highest recall in the rest of the sets. Also, we notice that PCF-WA has better recall than PCF-PA in h.r. [2, 10) set which shows that work affiliation ties are crucial in the early stages of new members. Overall, we notice that models that use affiliation ties (i.e., PCF variations, DAM and DAPM) have higher recall than models that use project content only (i.e., PCBF and DPM). Hence, the social aspect (the affiliation ties) is capable of retrieving more matches than the project content aspect.
Figure 5.4. The average recall on multiple h.r. sets.

Figure 5.5. The average nDCG@20 on multiple h.r. sets.

**Ranking:** Next, we evaluate the quality of ranking by the nDCG metric. Figure 5.5 shows nDCG@20 for each data history record set. In h.r. [2, 10) set, DAPM has unmatched superiority over memory-based models, which means DAPM can find highly relevant projects on high ranks for new members with very few data records. On the other hand, in ≥ 200 set, PCF-PA has higher performance than probabilistic models. Also, we notice the improvement of DPM in the ≥ 200 set over PCF-WA and PCF-LA, which indicates the increasing importance of a project’s features preference for senior members.

**Suitability:** Finally, we evaluate the quality of retrieved projects. Figure 5.6 shows the unmatched superiority of DAPM in h.r. [2, 10) set. This means that DAPM is highly capable of recommending high suitable (relevant) projects to new members with very few data records. DAPM has close performance with PCF-LA in h.r. [10, 50) set and has highest performance again in h.r. [50, 200) set. In h.r. ≥ 200 set, PCF-all and PCF-PA have the advantage over probabilistic models. Moreover, we notice the advantage of combining the project content aspect
with the developers’ affiliations aspect in one model which gave a high advantage to DAPM over DAM in all sets except in the h.r. $\geq 200$ set.

**Discussion:** Dividing the data set according to the history records availability gave new insights on the strengths and weaknesses of well known recommender models such as memory-based and probabilistic models. The previous evaluation shows that the probabilistic model, DAPM, has superior performance over the memory-based models when data is scarce. The design of DAPM allows high prediction accuracy, even when data is very scarce, because of the learned parameters from the training dataset. On the other hand, the memory-based models, PCF variations, perform better when data is very abundant because the models by design increase the prediction power as it sees more data which makes it more accurate than the probabilistic models in this case.

Comparing the projects’ content and the affiliation ties aspects, the models using the affiliation ties (PCF, DAM and DAPM) show better performance than the models using projects’ content (PCBF and DPM) in general. This shows the profound effect of social ties in attracting developers to participate in new projects. Moreover, we observe that the impact of projects’ content aspect increases as the developer has more participations. This is true because as the developers have more experience on OSS development process and participation, they become more selective on the type of projects they tend to participate in. Furthermore, projects’ content information (i.e., DPM) does not perform well alone, however, when it is combined with the developers’ affiliation information (as in DAPM), we notice clearly the boost that it causes to DAM performance overall. This shows the advantage of considering the two aspects together.

Comparing the different types of affiliations, overall, the project affiliation has the best performance. This is because OSS projects is the main media of collaboration between developers. However, the impact of work and location affiliations are also visible and combining the three types of affiliations, as in DAM and PCF-all, produces models with high accuracy performance. Moreover, comparing work and location affiliations, work affiliation ties seems to be more impor-
tant than location affiliation ties for new members, in h.r. [2, 10) and [10, 50) sets. However, the location ties gain more impact on developers with project participations of more than 50. The importance of work affiliation ties in the early stages of a developer is reasonable since new members would join projects with members of the same work affiliation to seek close collaboration as a start. On the other hand, the increase impact of the location affiliation ties as a developer has more project participations could be related to the fact that developers establish ties with other developers of certain location over time, however, this phenomenon needs more investigation.

5.6 Conclusion

In this work, we target the problem of recommending OSS projects to active developers in the OSS community. The work is motivated by the fact that many OSS projects tend to fail due to the lack of participation from developers. Hence, our goal is to increase the developers’ participation in the OSS projects. There exist two main challenges in recommending OSS projects. First, identifying the factors that developers act upon on selecting projects to join, and second, the data scarcity problem for new community members. Our solutions consider two main aspects in the recommendation, (i) the content of projects and (ii) the developers’ affiliations. The content of projects are dynamic attributes (e.g., team size, popularity, complexity, skills required). For the second aspect, we consider three types of affiliation ties in the recommender models, namely, project co-participation, work and location affiliation ties. We propose four memory-based models using content and collaborative filtering methods (i.e., PCBF, PCF-PA, PCF-WA and PCF-LA) and one probabilistic model (DAPM). We evaluate our proposal using a longitudinal dataset collected from Github.com over a span of one year. In addition, we test the models under different data scarcity levels. The results show the advantage of DAPM over memory-based models in the case of new or less active members with few data records. Meanwhile, the PCF models have better performance in the case of very abundant data records. Moreover, combining the two aspects of projects’ content and developers’ affiliation in DAPM shows great performance boost in retrieving more and high relevant projects to developers with few data records. In the future, we intend to improve the models to be capable of recommending projects to a group of active developers instead of an individual developer. Finally, We believe that the proposed models to recommend projects can increase the developers’ involvement in open projects and, as a consequence, increase its likelihood of success.
Chapter 6

Recommending Teams for Open Collaborative Projects Based on the Degree of Acquaintance

6.1 Introduction

The work in this chapter has been published in the proceedings of the 2013 International Conference on Database Systems for Advanced Applications (DASFAA).

Large collaborative online communities have become a phenomena in the presence of Web 2.0 technology, witnessed by the massive success of Open Source Software (OSS) projects such as the Apache projects and GNU/Linux. As members of OSS projects are mostly volunteers [1], they usually work out of personal goals/interests, e.g., practicing existing skills and gaining experience, following fellow peers, networking with the OSS community members, or simply supporting free open software projects [2]. Consequently, the amount of participation and commitment by the volunteering developers are crucial factors in the cause of OSS projects success [3].

Research on software engineering reveals a number of factors that assist in increasing the developers participation in OSS projects, including the computer language required, the operating system used, or the type of license for open software [4, 5]. For example, in [4], it is observed that OSS projects requiring popular computer languages (e.g., Java and C variants) attract more participants since many developers are experienced in these languages. This finding suggests that developers tend to participate in a project if they possess the skill(s) required for that project. Recently, the importance of social factors, such as for a participant to join fellow peers or build a professional network with fellow developers in the community, for a project to ensure the rapport between group members, are noticed. Backstrom et al. [10] study the group formation and the membership growth in large open communities and find that the probability of an individual
joining a community group increases as the number of friends inside the community and the internal connectedness of friends in the community increase. Hinds et al. [24] suggest that, when forming a new team, people tend to join others whom they have established work ties before. Moreover, Hahn et al. [11] point out that existing ties and relationships in OSS communities affect the formation of new project teams. It has been pointed out that prior collaborative ties among developers increases the probability of developers to join a new project where prior ties with the initiators exist. These studies suggest that existing ties (acquaintances) between individuals in open collaborative communities are crucial for those individuals to connect and join a new emerging project to form a team with rapport. These findings give a new insight to the effective factors in forming a successful team in open collaborative projects, which is our motivation to conduct this work. In this work, we study the problem of team formation in the volunteer-based community of open collaborative projects.

There exist several works that address the team formation problem in literature. However, they are not suitable for the open source projects that are based on volunteers. An early work by Barreto et al. [28] defines the staffing of software projects as a constraints satisfaction problem. The work considers only skills matching but does not consider the social ties between members. Other recent works optimize the team utility in terms of the team communication cost, which is mostly measured by different graph distance measures in a connected graph. The first work involving the communication cost is the work of Lappas et al. [50], where they use the diameter and the spanning tree of the team graph to measure the team communication cost. Li et al. [51] enhance the Steiner Tree algorithm used in [50] to solve a generalized team formation problem that assigns different number of experts for each required skill as a constraint. Both [50] and [51] tend to add mediator members, i.e., members that do not possess the skills required, to the team in order to minimize the communication cost. Nevertheless, it is worth noting that mediator members, in most cases, are not involved in OSS projects because these mediators will not feel the obligation to participate in open projects if they do not possess the required skills [4]. Another issue is that OSS projects usually consist of volunteers that sometimes form several subgraphs or even individual members that work on their own [17]. Therefore, it is difficult to optimize the communication cost in OSS projects. Recently, Gajewar et al. [52] introduce a measure of team communication quality based on a graph density which is claimed to be more robust than the diameter and the spanning tree measure, and thus more suitable for modular graph structures. However, the density measure ignores the aspect of graph structure and connectivity between the team members. Recall that a graph density is the total edges weights
divided by the total number of vertices in a graph. In Figure 6.1, assuming all edge weights in graphs (a), (b) and (c) equal one. The density of graphs (a) and (b) is 0.75, regardless of their different graph structures. However, in terms of social ties, each vertex has different tie structure, e.g., vertex 1 is connected to one vertex in graph (a), however, it is connected to all the vertices in graph (b) which indicates a different social influence for vertex 1 in the two graphs, (a) and (b). Moreover, when considering the diameter measure, both graphs (b) and (c) have diameter 2, while it is obvious that graph (c) is more socially tight than graph (b). As in Section 2.5.2, we show a statistical evidence that each team member’s connectivity and tie strength are important in increasing the team productivity. Therefore, we need a more fine-grained measure to find a well socially tight team with high participation and productivity outcome.

**Our Contributions.** In light of the above observations, this work proposes a novel concept, which is called the Degree of Acquaintance (DoA), which seamlessly integrates the connectivity, measured using the local *Clustering Coefficient* for each member, and the strength of ties specified by the frequency of co-participation. Thus, given an emerging OSS project that requires members with certain skills, our goal is to create a team that covers all the required skills and maximizes the two social factors which are defined by the DoA of the team. In this work, we have made a number of contributions in achieving our goal:

- Our key contribution is to account for the *connectivity* (local clustering coefficient) and *ties strength* for users (in a social network) in the team formation problem. We define the Degree of Acquaintance based on these two factors (refer to Section 6.3). This is fundamentally different than the other approaches that uses aggregated graph metrics such as graph Diameter or graph Density.

- We formulate the problem of team formation based on DoA and prove that it is NP-hard and hard to approximate problem (refer to Section 6.4).

- We propose three new algorithms to solve the DoA team formation problem. The first one, *Partial Selective Tree search Algorithm* (PSTA), and the second one, *Selective Tree search Algorithm* (STA), are based on a BFS tree search, where PSTA produces the optimal solution but is less scalable while STA is more scalable but does not guarantee optimality. The third algorithm, *Neighbor-First search Algorithm* (NFA), is an efficient and scalable greedy approach (refer to Section 6.5).

- We evaluate the scalability and performance of the proposed algorithms and two existing approaches (i.e., Diameter and Density based) using real data from *ohloh.net* that includes hundreds of thousands of developers and over a million relationships. Experimental results show that the proposed algorithms can find the teams with high DoA in much higher magnitude than the existing approaches (refer to Section 6.6).

The rest of the chapter is organized as follows. The following section (Section 6.2) introduces some related works involving the team formation problem. Then, Section 6.3 defines the DoA,
and Section 6.4 defines the DoA based team formation problem and discusses the DoA properties used in our algorithms. Section 6.5 discusses in details the proposed algorithms, and Section 6.6 evaluates the proposed algorithms. Finally, Section 6.7 concludes the chapter.

6.2 Related Works

Barreto et al. [28] consider the staffing problem for software projects and define the problem as a constraints satisfaction problem. The constraints are variables that are project dependent such as pursuing specific budget or a due date. [28] does not consider social aspects. Wi et al. [53] transform the team formation problem into integer programming maximization problem where three objectives are observed, the knowledge on the areas of expertise, the familiarity between individuals, and the team size. Unlike our defined Degree of Acquaintance, the familiarity metric in [53] is considered between two individuals only and not for an individual in a team. Also, the problem defined does not consider the connectivity among the members since it does not study the social graph. Dorn et al. [54] define the team formation problem as trade-off between skill coverage and team distance, given a query with weighted skills indicating variant importance levels for each skill. Simulated Annealing approach is used to find an approximate solution.

The work of Lappas et al. [50] is the first to consider the team formation problem with the consideration of communication cost. Given a query of required skills, the goal is to form a team that covers all the required skills and at the same time minimizes the communication cost in the team. The communication cost is evaluated by, first, the graph diameter and , second, the Minimum Spanning Tree (MST) of the graph. Gajewar et al. [52] solve the team formation problem when the query requires a minimum number of individuals for each skill in a task. The cost function used is based on the graph density which is claimed to be more robust than the diameter and the MST cost functions since adding or deleting any vertex in a graph does not affect the density value as much as it affects the diameter or MST of the graph. In [51], Cheng-Te et al. complement the work in [50] by considering multiple experts required for each task in a query and adapt the Enhanced Steiner algorithm to suit the generalized problem definition. To further enhance the scalability issue, [51] proposes the group-based team formation in which individuals are aggregated into groups according to their related skills and then the Enhanced Steiner algorithm is applied on the aggregated graph to find a suitable subgraph that satisfies the query.

Yang et al. [55] solves a group formation problem given several social constraints. An initiator issues a query to form a group of individuals with a minimum social distance to the initiator and acquaintance limit among the group members. In another version of the problem, the availability time slots constraint is added to the query. Another work by Anagnostopoulos et al. [56] study the team formation problem under the assumption of online incoming stream of tasks. The goal is to find a team that covers the tasks and at the same time balance the tasks overload on the team members to maximize the team utility, which is a trade-off between work load and team size.
6.3 The Degree of Acquaintance

The social network of an OSS community is modeled as an undirected graph $G(X,E)$, where $X = \{x_1, ..., x_n\}$ is the set of $n$ vertices that represent all the active developers in the community, and $E$ is the set of weighted edges that represent the relationships between developers. We use an $n \times n$ matrix $M$ to present the social network. Also, let $w_{ij}$ in $M$ denotes the edge weight between individuals $x_i$ and $x_j$. $^1$

Let $S = \{s_1, ..., s_m\}$ be a universe of $m$ skills. We define an $n \times m$ developer-skill matrix $A$, where the rows consist of $n$ developers and the columns consist of $m$ skills. Each element of $A$, denoted as $a_{i,j}$, is a binary value, where $a_{i,j} = 1$ indicates that developer $x_i$ possesses skill $s_j$; and $a_{i,j} = 0$, otherwise. Each row $i$ in $A$, denoted by $x_i$, is a vector of skills possessed by developer $x_i$. Also, each column $j$ in $A$, denoted by $s_j$, is a vector of developers who possess skill $s_j$. We refer to $x_i$ as the developer profile of $x_i$ and $s_j$ the skill profile of $s_j$. Next, we define the Degree of Acquaintance.

6.3.1 Definition of DoA

Let $G$ be a collaborative community. The Degree of Acquaintance (DoA) for an individual $x_i$ in a team $T$, where $T \subseteq G$, consists of two factors: (i) the total weights of edges incident to $x_i$, and (ii) the connectivity among $x_i$’s neighbors. While the first factor is easy to understand, we exploit the local Clustering Coefficient (CC) of vertex $x_i$ in graph $T$, defined in Eq. (2.1) in Section 2.5.2, to capture the second factor. Formally, let $w_{i,j}$ denote the edge weight between vertex $x_i$ and $x_j$, the DoA for an individual $x_i$ in a team $T$ is the linear combination of the total weights and the CC factors as defined in Eq. (6.1).

$$\text{DoA}_T(x_i) = \alpha \left( \sum_{j \in N_i} w_{ij} \right) + (1 - \alpha) CC(x_i)$$  \hspace{1cm} (6.1)

where $\alpha = [0, 1]$ is a control parameter to balance the two factors. A proper value for $\alpha$ would depend on the nature of the team desired in terms of connectedness or ties strength. In our experiment, we set $\alpha = 0.5$. The DoA is a team structure-dependent metric which does change when adding or removing vertices from $T$.

The DoA of a team $T$ is defined as the summation of $\text{DoA}_T(x_i)$ for every vertex $x_i \in T$ as in Eq. (6.2).

$$\text{DoA}(T) = \sum_{\forall x_i \in T} \text{DoA}_T(x_i)$$  \hspace{1cm} (6.2)

To eliminate the naming confusion between Eq. (6.1) and Eq. (6.2), we refer to Eq. (6.1) as the Individual DoA ($\text{IDoA}$), and Eq. (6.2) as the Team DoA ($\text{TDoA}$). To illustrate how to compute $^1$In this work, $w_{ij}$ is the collaboration counts between individuals $x_i$ and $x_j$ normalized by dividing by the maximum weight in $G$. 


Figure 6.2. Graph $T$ representing a team formation with normalized edges’ weights and vertices’ Clustering Coefficient (CC) values.

In Section 2.5.2, we present a statistical evidence that demonstrates the importance of the two DoA factors on the contribution and commitment exerted by developers in the OSS projects. These statistical results show an obvious effect of connectivity and ties strength between developers on the amount of contribution and commitment in OSS projects. Therefore, taking into account these two factors in the Team DoA would improve the teams’ productivity. Accordingly, we formulate the team formation problem and define a query in the next section.

6.4 The DoA Based Team Formation Problem

In this section, we formulate the team formation problem based on the notion of DoA, and present several properties useful to our proposed algorithms.

6.4.1 Problem Formulation

Definition. Degree of Acquaintance based Team Formation (DoA-TF): Given a social graph $G(X,E)$ and a developer-skill matrix $A$, a DoA based team formation query $Q \equiv \{S_q, \tau\}$,
where $S_q$ is a set of skills required ($|S_q| = 1$) and $\tau$ is the maximum number of developers allowed in a team, finds the set of developers to form a team $T$ that covers all the skills required by $Q$ such that $|T| \leq \tau$ and that $\text{DoA}(T)$ is maximized.

**Proposition 1.** The DoA-TF problem is NP-complete.

*Proof.* We consider a special case of the DoA-TF problem, where $\alpha = 0$ and every vertex in $G_F$ covers all required skills. In this case, only the Clustering Coefficient is considered. Then we prove the proposition by a reduction from the $k$-clique problem, a well known NP-complete problem. An instance of the $k$-clique problem consists of a graph $\hat{G}(\hat{X}, \hat{E})$ and $k$, where $\hat{X}$ is the set of vertices in $\hat{G}$, $\hat{E}$ is the set of edges in $\hat{G}$, and $k$ is a positive integer. A decision problem version of the $k$-clique asks whether there exists a clique of size $k$ in $\hat{G}$ or not.

We transform an instance of the $k$-clique problem to an instance of the DoA-TF special case problem by a direct mapping from $\hat{G}(\hat{X}, \hat{E})$ to $G_F(X_F, E_F)$. Having $\tau = k$, the solution for DoA-TF is a clique of size $\tau$ in $G_F$. Therefore, solving the DoA-TF special case problem instance can obtain the solution to the $k$-clique instance. The proposition follows.

Note that finding the maximum clique problem is both NP-hard and hard to approximate (not approximable within $|X|^{(1-\epsilon)}$ for any $\epsilon > 0$) [57]. Consequently, the general DoA-TF problem is NP-hard and hard to approximate, which makes the problem very challenging.

### 6.4.2 DoA Properties

We aim to have the Team DoA serving as the objective function for team formation. One may think that adding more members to a team would increase the objective function. However, Eq. (6.1) is not monotonic because the CC is not monotonic. As illustrated in Figure 6.3(a), $CC(x_3) = 1$. By adding $x_4$ and $e_{3,4}$ (see Figure 6.3(b)), $CC(x_3)$ decreases since not all of its neighbors are connected. Nevertheless, the first term in Eq. (6.1), $\sum_{ij \in N_i} w_{ij}$, is monotonic. Therefore, if the second term in Eq. (6.1), $CC(x_i)$, does not decrease when adding candidate members to $T$ during query processing, we can ensure the monotonicity of Eq. (6.1). Consequently, Eq. (6.2) becomes monotonic as well.
As monotonicity is important to assure the optimization of the objective function in any optimization algorithm, we introduce two cases where adding acquainted members does not change the clustering coefficient of the members in the team. The first case is illustrated in Figure 6.4. In Case (1), assume all members in some team know each other (i.e., their CC equals 1). If a new member joins the team, where everyone in the team knows the new member, their clustering coefficient remains 1.

The second case is illustrated in Figure 6.5. In Case (2), assume each member in some team knows at least one member in the team, but there is no mutual acquaintance among the members (i.e., their CC equals 0). If a new member joins the team, and the new member is acquainted to only one member in the team, then the CC remains 0 for each member in the team.

Our strategy to process the DoA-TF query, in Section 6.5, is to carefully examine the structure of the community graph in order to select individuals and construct a team subgraph in consecutive steps. As a result, when we add a vertex $x$ to $T$, where $x$ is part of a full-clustering structure (Case (1)), the objective function increases. Likewise, when we add a vertex $x$ to $T$, where $x$ is part of a zero-clustering structure (Case (2)), the objective function increases as well.

For a graph $G(X, E)$ with undirected weighted edges and a subgraph $T(\hat{X}, \hat{E})$, where $T \subseteq G$, such that $\hat{X} \subseteq X$ and $\hat{E} \subseteq E$, suppose we want to add a vertex $x_i \in X$ to $T$ with $k$ existing vertices in $T$ that are neighbors to $x_i$, where $k \leq |N_i|$ ($N_i$ is the set of neighboring vertices to
Figure 6.6. Graph G representing a collaborative community graph with normalized edges’ weights and skills’ labels.

$x_i$ in $G$). When adding $x_i$ to $T$, edges $e_{i,j_1}, e_{i,j_2}, \ldots, e_{i,j_k}$ are included in $\hat{E}$, then the objective function for $T$ must be increased according to the following two lemmas:

**Lemma 1.** If $CC_G(x_i) = 1$ and $CC_G(x_j) = 1, \forall x_j \in N_i$, then adding $x_i$ to $T$ increases the Individual DoA values for $x_i$ and its $k$ neighboring vertices in $T$, and as a result increases the objective function value.

*Proof.* Since $CC = 1$ for $x_i$ and $\forall x_j \in N_i$, then edges $e_{i,j_1}, \ldots, e_{i,j_k} \in \hat{E}$ are part of a full-clustering subgraph. Therefore, when adding $x_i$ to $T$, only the edge weights dominate the objective function monotonically.

**Lemma 2.** If $CC_G(x_i) = 0$ and $CC_G(x_j) = 0, \forall x_j \in N_i$, then adding $x_i$ to $T$ increases the Individual DoA values for $x_i$ and its $k$ neighboring vertices in $T$, and as a result increases the objective function value.

*Proof.* Since $CC = 0$ for $x_i$ and $\forall x_j \in N_i$, then edges $e_{i,j_1}, \ldots, e_{i,j_k} \in \hat{E}$ are part of a zero-clustering subgraph. Therefore, when adding $x_i$ to $T$, only the edge weights dominate the objective function monotonically.

### 6.5 Team Formation Algorithms

A straightforward approach to solve the DoA-TF problem is to find every team following the constraints and select the one with the maximum $TDoA$ value. Next, we present an example of finding the optimal solution for some query $Q$ by using the brute-force approach.
The brute-force approach is computation intensive and requires $O(2^n)$ time to find the optimal solution. With a vast OSS community, with over a million developer, this straightforward approach is not efficient. To address this issue, we propose three algorithms for DoA-TF.
6.5.1 Partial-Selective Tree Search Algorithm

The Partial-Selective Tree search Algorithm (PSTA) takes a tree-search approach to find the optimal team. The algorithm starts with finding the feasible subgraph $G_F \subseteq G$ of developers that possess at least one required skill in $Q$. Then the developers who possess the rare skill (i.e., the skill with the least number of developers) are considered as seeds to grow search trees of team solutions. Therefore, the whole search space have multiple trees. Notice that a tree grows into lower level branches by adding a candidate team member one at a time. Thus, each node on a tree contains a partial team solution, denoted by $T_{d,b} \subseteq G_F$, where $d$ is the depth level of the node, $b$ is the branch count in level $d$. As mentioned, PSTA adds one vertex to each node (i.e., team) in the current level to create the child nodes in the next level. Therefore, the number of vertices (team members) in a node at level $d$ equals $d$, $|T_{d,b}| = d$. In other words, the maximum level of each search tree is $\tau$. PSTA algorithm is shown in Alg. 1.

We discuss PSTA in much details since it has main concepts that are used in the two other algorithms. The major steps in PSTA are (i) selecting seed vertices for level 1, where each seed vertex creates the first member of each team permutation, (ii) Candidates set selection, where at each node a set of candidate vertices are chosen, (iii) child node creation, where at each node PSTA selects vertices from the candidates set to create a child node. Next we detail the PSTA algorithm.

First, lines 1-2 find $G_F$ and calculates the IDoA for each vertex in $G_F$. Then, line 3 finds the rarest skill profile, $s_{rare}$, in $S_q$, i.e., the rarest skill has the lowest number of developers possessing that skill. We choose the seeds from $s_{rare}$ to start the tree search in order to minimize the number of search trees. We detail the seeds selection procedure later in this section and proceed to candidates selection procedure next.

Candidates Selection: At each node solution $T_{d,b}$, there exist a set of candidate developers/vertices, denoted by $X_c$, from which the selection is drawn to create a child node solution $T_{d+1,b}$ in the next level of $d + 1$. The routine of selecting members in $X_c$ is shown in Alg. 2. Basically, if the vertices in $T_{d,b}$ do not cover the query skills, then $T_{d,b}$ is not a solution and $X_c$ should contain the vertices possessing the uncovered yet skills. On the other hand, if the vertices in $T_{d,b}$ cover all query skills, then $X_c$ will contain every vertex in $X_F$ that is not in $T_{d,b}$.

In details, Alg. 2, first obtains the team profile denoted by $T_p$, which is the union of all user profiles in the current team solution $T$ (line 1 of Alg. 2). Afterward, the routine identifies the set of missing skills denoted by $\Gamma$ in line 2 of Alg. 2. If $\Gamma$ is empty, then the current team formation covers all the required skills, otherwise it is not a valid solution yet.

For a valid team, we compute the $TDoA$, $DoA(T)$, and compare it with $DoA(T^*)$, where $T$ is the current node team set, and $T^*$ is the team set with the maximum $TDoA$ so far. Initially $T^*$ is empty. $T$ becomes $T^*$ if it has a higher $TDoA$ value than $T^*$. Then, the candidates set, $X_c$, includes every vertex in $G_F$ that is not included in the current team set yet according to lines

---

2In this work, a node is related to a tree, while a vertex is related to a network graph.
3-8 of Alg. 2. In contrast, for an invalid team, the candidates set, $X_c$, contains every developer (vertex) that possess at least one of the missing skills in $\Gamma$, as in line 10 of Alg. 2

**Child Nodes Creation:** The crux of PSTA is to divide the vertices in $X_c$ into monotonic and non-monotonic candidate sets. In details, $X_c$ is divided into $N$ and $H$ sets, where $N$ is the set of neighboring vertices to $T_{d,b}$ and $H$ is the set of non-neighboring vertices. Moreover, $N$ is divided into $N_1$, $N_0$ and $N_p$, where $CC_{G_F}(x_i) = 1$ for $\forall x_i \in N_1$, $CC_{G_F}(x_i) = 0$ for $\forall x_i \in N_0$, and $0 < CC_{G_F}(x_i) < 1$ for $\forall x_i \in N_p$. Vertices in $N_1$ follow Case (1) and vertices in $N_0$ follow Case (2) and thus are monotonic, while vertices in $N_p$ are not. Therefore, selecting vertices from $N_1$ and $N_0$ monotonically guarantees to optimize the solution and otherwise for $N_p$. PSTA selects the vertex with the highest total adjacent link weight from $N_1$ and $N_0$ (lines 11 and 14).ootnote{In lines 11, 14 and 17 of Alg. 1, the index $b_c$ represents the branch count of the child nodes.} This forward-pruning process reduces the search space tremendously. On the other hand, a selection from $N_p$ does not guarantee to optimize the next level solution, therefore, PSTA creates a child solution from each vertex in $N_p$ (line 17). Finally, $H$ is treated as a separate graph since none of the vertices in $H$ are connected to the current node solution. Therefore, PSTA selects seeds from $H$ (lines 18 to 20) similar to the seeds selection from $s_{rare}$. Next we detail the process of seeds selection.

**Seeds Selection:** $s_{rare}$ is divided into three sets (line 4), i.e., $seeds_1$, $seeds_0$ and $seeds_p$, corresponding to $N_1$, $N_0$ and $N_p$ respectively. Then the seeds selection routine is used; shown in Alg. 3. Having the seed sets, PSTA selects the vertex with the highest $IDoA$ in $G_F$ from each $seeds_1$ and $seeds_0$ if $\tau \geq |N_{\text{selected}}|$ to guarantee that $IDoA$ is obtainable after $\tau$ steps (levels). Otherwise, each vertex in $seeds_1$ and $seeds_0$ grows as a separate search tree (lines 1-5 for $seeds_0$ and lines 6-10 for $seeds_1$). Finally, each vertex in $seeds_p$ creates a child node (line 11). In the case of the $d = 1$ (i.e., level 1), then each selected seed is the first member in each node solution, hence $T_{d,b} = \Phi$ in Alg. 3. The same routine is used for $H$, however, the seed vertex selected is added to the current solution of $T_{d,b}$.

The algorithm stops after processing all the nodes at level $\tau$ and the solution would be the team that satisfies the constraints and having the maximum $TDoA$.

**Proposition 2.** The PSTA algorithm finds the optimal solution for a given query $Q$.

**Proof.** According to Lemma 1, we conclude that any selected vertex $x \in N_1$ added to solution $T$ does optimize the objective function of $T \cup x$. Also, according to Lemma 2, we conclude that any selected vertex from set $x \in N_0$ added to solution $T$ does optimize the objective function of $T \cup x$. Since the selection process is monotonic and each vertex from set $N_p$ is added to solution $T$, the PSTA algorithm assures finding the optimal solution.

In the above proof, the set $H$ is ignored in the argument because it creates a subgraph that follows the same procedure of creating solutions for the whole graph. The time complexity of
Algorithm 1: The Partial-Selective Tree Search Algorithm (PSTA)

Input : $G(X,E)$; matrix $A$; query $Q$; $\alpha$.
Output: $T^* \subseteq G$; DoA($T^*$).

1. **Init.:** $G_F(X_F,E_F) \leftarrow \bigcup_{j \in S_q} s_j$; $T^* = \emptyset$; // Calculate IDoA

2. $s_{rarc} \leftarrow \arg \min_{j \in S_q} |s_j|$;

3. seeds$_0$, seeds$_1$ and seeds$_p$ $\leftarrow s_{rarc}$;

4. SeedsSelection(seeds$_0$, seeds$_1$, seeds$_p$, $G_F$);

5. **for** $d = 1; d \leq \tau; d += 1$ **do**

6. **for** $b = 1; b \leq \text{BreadthSize}; b += 1$ **do**

7. Get $T^*$, DoA($T^*$) and $X_c$ for $T_{d,b}$ by Alg. 2;

8. $N_1$, $N_0$, $N_p$, $H \leftarrow X_c$;

9. **if** $N_1 \neq \emptyset$ **then**

10. $x_{selected} \leftarrow \arg \max_{i \in N_1, j \in T_{d,b}} w_{i,j}$;

11. $T_{d+1,b} += T_{d,b} \cup x_{selected}$;

12. **if** $N_0 \neq \emptyset$ **then**

13. $x_{selected} \leftarrow \arg \max_{i \in N_0, j \in T_{d,b}} w_{i,j}$;

14. $T_{d+1,b} += T_{d,b} \cup x_{selected}$;

15. **if** $N_p \neq \emptyset$ **then**

16. **foreach** $x_i \in N_p$ do $T_{d+1,b} += T_{d,b} \cup x_i$;

17. **if** $H \neq \emptyset$ **then**

18. seeds$_0$, seeds$_1$ and seeds$_p$ $\leftarrow H$;

19. SeedsSelection(seeds$_0$, seeds$_1$, seeds$_p$, $G_F$);

PSTA is $O(n^\tau)$. The complexity comes close to the upper bound if at each node the candidate vertices are in set $N_p$. However, in reality the time complexity is much smaller than the upper bound. Yet, PSTA is not scalable to large graphs. Therefore, we developed the complete Selective Tree search Algorithm (STA) to mitigate the scalability issue and bring it to a practical level.

6.5.2 Selective Tree Search Algorithm

The Selective Tree search Algorithm (STA), shown in Algorithm 4, is similar to the PSTA algorithm except that it uses the forward-pruning on each selection set at each node. The seed selection is as follows. For sets seeds$_0$, seeds$_1$ and seeds$_p$, STA always selects the vertex with the maximum IDoA from each set, as in lines 4-9 of Alg. 4. Hence, the maximum fan-out of the root is three. At each node, from level one to level $\tau - 1$, STA selects the vertex with the maximum IDoA from sets $N_p$ and $H$, as in lines 20-25 of Alg. 4. The selection heuristic for $N_0$ and $N_1$ is the same as in PSTA. Thus, the tree fan-out is reduced tremendously. STA is scalable but does not guarantee an optimal solution. STA may still not scale to very large graphs and queries with large team size upper bounds. To further address the scalability issue, we introduce a polynomial-time algorithm in the following section.
Algorithm 2: Finding Candidate set $X_c$, $T^*$ and $DoA(T^*)$ Routine

**Input**: $T; T^*; G_F(X_F, E_F); matrix A; S_q$.

**Output**: $T^*$; $DoA(T^*)$; $X_c$.

1. $T_p \leftarrow \bigcup_{i \in T} x_i$; // get team profile
2. $\Gamma \leftarrow S_q - T_p$; // get uncovered skills
3. if $\Gamma = \Phi$ then
   4. $DoA(T)$; // Calculate Team DoA
   5. if $DoA(T) > DoA(T^*)$ then
      6. $T^* \leftarrow T$;
      7. $DoA(T^*) \leftarrow DoA(T)$;
     8. $X_c \leftarrow X_F - T$;
   9. else
      10. $X_c \leftarrow \bigcup_{j \in \Gamma} s_j$;

Algorithm 3: PSTA SeedsSelection Routine

**Input**: seeds$_0$; seeds$_1$; seeds$_p$; $G_F$; $T_{d,b}$

**Output**: Level $d + 1$ nodes

1. $x_{selected} \leftarrow \arg \max_{i \in \text{seeds}_0} DoA_{G_F}(i)$;
2. if $(\tau - |T_{d,b}|) \geq |N_{x_{selected}}|$ then
   3. $T_{d+1,b_{c++}} \leftarrow T_{d,b} \cup x_{selected}$;
   4. else
      5. foreach $x_i \in \text{seeds}_0$ do $T_{d+1,b_{c++}} \leftarrow T_{d,b} \cup x_i$;
   6. $x_{selected} \leftarrow \arg \max_{i \in \text{seeds}_1} DoA_{G_F}(i)$;
    7. if $(\tau - |T_{d,b}|) \geq |N_{x_{selected}}|$ then
       8. $T_{d+1,b_{c++}} \leftarrow T_{d,b} \cup x_{selected}$;
    9. else
       10. foreach $x_i \in \text{seeds}_1$ do $T_{d+1,b_{c++}} \leftarrow T_{d,b} \cup x_i$;
       11. foreach $x_i \in \text{seeds}_p$ do $T_{d+1,b_{c++}} \leftarrow T_{d,b} \cup x_i$;

6.5.3 Neighbor-First Search Algorithm

We developed a greedy algorithm, called Neighbor-First search Algorithm (NFA), which uses $IDO_A$ as a heuristic for vertex selection. The NFA algorithm, shown in Algorithm 5, starts by finding the feasible graph $G_F$. Then it selects a seed vertex from $G_F$ with the maximum $IDO_A$ and adds the seed vertex into the team solution set $T_t$ (lines 2-3), where $T_t$ is the team set at step $t$. With $T_t$ containing the seed vertex, the algorithm proceeds by entering a while loop. At the start of each iteration (line 5), NFA finds the candidates set $X_c$, which contains the developers possessing the uncovered skills and $G_F - T_t$ if all skills are covered by $T_t$. Moreover, it calculates the objective function if $T_t$ is valid.

Having $X_c$, NFA proceeds by finding the set of vertices neighboring to the vertices in $T_t$ (line 6), denoted by $N_t \subseteq X_c$. Afterwards, if $N_t$ is not empty, NFA selects the vertex with maximum $IDO_A$ from $N_t$ to the current solution $T_t$. On the other hand, if there are no vertices neighboring to $T_t$, then the algorithm selects the vertex with the maximum $IDO_A$ from $X_c$ to $T_t$ (lines 7-11).
Algorithm 4: The Selective Tree Search Algorithm (STA)

Input: $G(X,E)$; matrix $A$; query $Q$; $\alpha$.
Output: $T^* \subseteq G$; $DoA(T^*)$.

1. Init.: $G_F(X_F,E_F) \leftarrow \bigcup_{j \in S_q} s_j$; $T^* = \Phi$; $d = 1$; $b = 0$;
2. $s_{rare} \leftarrow \arg\min_{j \in S_q} |s_j|$;
3. seeds$_0$, seeds$_1$ and seeds$_p \leftarrow s_{rare}$;
4. $x_0 \leftarrow \arg\max_{i \in \text{seeds}_0} DoA_{G_F}(i)$;
5. $T_{d,b} \leftarrow x_0$;
6. $x_1 \leftarrow \arg\max_{i \in \text{seeds}_1} DoA_{G_F}(i)$;
7. $T_{d,b} \leftarrow x_1$;
8. $x_p \leftarrow \arg\max_{i \in \text{seeds}_p} DoA_{G_F}(i)$;
9. $T_{d,b} \leftarrow x_p$;
10. for $d = 1; d \leq \tau; d++$ do
11.     for $b = 1; b \leq \text{BreadthSize}; b++$ do
12.         Get $T^*$, $DoA(T^*)$ and $X_c$ for $T_{d,b}$ by Alg. 2;
13.         $N_1, N_0, N_p, H \leftarrow X_c$;
14.         if $N_1 \neq \Phi$ then
15.             $x_{selected} \leftarrow \arg\max_{i \in N_1, j \in T_{d,b}} w_{i,j}$;
16.             $T_{d+1,b} \leftarrow T_{d,b} \cup x_{selected}$;
17.         if $N_0 \neq \Phi$ then
18.             $x_{selected} \leftarrow \arg\max_{i \in N_0, j \in T_{d,b}} w_{i,j}$;
19.             $T_{d+1,b} \leftarrow T_{d,b} \cup x_{selected}$;
20.         if $N_p \neq \Phi$ then
21.             $x_{selected} \leftarrow \arg\max_{i \in N_p} DoA_{G_F}(i)$;
22.             $T_{d+1,b} \leftarrow T_{d,b} \cup x_{selected}$;
23.         if $H \neq \Phi$ then
24.             $x_{selected} \leftarrow \arg\max_{i \in H} DoA_{G_F}(i)$;
25.             $T_{d+1,b} \leftarrow T_{d,b} \cup x_{selected}$;

The algorithm stops when the team size reaches $\tau$, and the result is the team formation $T^*$ with the maximum $TDoA$ obtained. Again the team solution may not be necessary of $\tau$ members as $\tau$ is only the upper bound of team size.

The time complexity of the NFA algorithm includes the following three parts. First, $G_F$ creation takes $O(nl)$. Second, finding neighbors to $T_t$ (line 6 of Alg. 5) takes $|T_t| \times |X_c|$ times, which in worst case is $(\frac{n}{2} \times \frac{n}{2}) = O(n^2/4)$. Finally, the whole NFA algorithm iterates for $\tau$ times which gives a total time complexity of $O(\tau(n^2/4) + nl)$. Considering $n$ as the largest factor in the running time, it becomes approximately $O(n^2)$. In the next section, we discuss the experiment and the performance evaluation of the proposed algorithms.
Algorithm 5: The Neighbor-First Algorithm (NFA)

**Input**: $G(X, E)$; matrix $A$; query $Q$; $\alpha$.

**Output**: $T^* \subseteq G$; $DoA(T^*)$.

1. **Init.**: $G_F(X_F, E_F) \leftarrow \bigcup_{j \in S_q} s_j$; $T_1 \leftarrow \Phi$; $t = 1$;
2. $x_{seed} \leftarrow \arg \max_{i \in X_F} DoA_{G_F}(i)$;
3. $T_1 \leftarrow x_{seed}$;
4. **while** $|T_t| \leq \tau$ **do**
5. Get $T^*$, $DoA(T^*)$ and $X_{c_t}$ for $T_t$;
6. $N_t \leftarrow \{i | i \in X_{c_t} \land e_{i,j} \in T_t \not= \Phi\}$;
7. **if** $N_t \not= \Phi$ **then**
8. $x_{selected} \leftarrow \arg \max_{i \in N_t} DoA_{G_F}(i)$;
9. **else**
10. $x_{selected} \leftarrow \arg \max_{i \in X_{c_t}} DoA_{G_F}(i)$;
11. $T_{t+1} \leftarrow T_t \cup x_{selected}$;

6.6 Performance Evaluation

In this section we evaluate the proposed algorithms NFA, PSTA, and STA in addition to the Brute-Force Approach (BFA). Also, we compare the proposed algorithms with two existing approaches that form teams based on graph Diameter [50] and Density [52].

6.6.1 Dataset

The real dataset is collected from Ohloh.net, a fast growing OSS social site. It is an online community web service that provides a platform for developers and users to interact. There are over 600,000 developers in Ohloh.net with over 1,096,000 relationships and 83 different skills, mainly related to computer languages. Ohloh.net is unique because, first, it hosts OSS projects from multiple version control repositories (e.g. Subversion, CVS, Git, Mercurial, etc.) which increases the number of OSS projects hosted and includes various software topics. Second, and more importantly, it provides social ties information for the developers, such as the recognition and approval network, where developers are allowed to explicitly express approval and recognition to each other based on previous collaboration. In the dataset, we only consider the developers who have announced the skills they possess and contributed to OSS projects. Finally, we realize that many previous work use the DPLB dataset but we opted not to use it since it does not represent the sentiment of open projects environment which is the focus of this work.

6.6.2 Scalability and Accuracy Evaluation

**Scalability Evaluation**: First, we evaluate the scalability of the proposed algorithms in terms of the number of Execution Runs (ER) and the Execution Time (ET) with different $\tau$ and $G_F$ sizes, i.e., $\tau = [3, 20]$ and $|G_F| = [26, 7748]$, respectively. The size of $G_F$ depends on the number of skills specified in a query, thus the increment of the graph size is not equally distanced in
the experiment. The ER represents the number of iterations for BFA and NFA, while it is the number of nodes in the search tree for the PSTA and STA. Figure 6.7(a) shows the ER scalability as $G_F$ increases. BFA and PSTA ER increase exponentially, however, PSTA ER is smaller than BFA. NFA ER is constant since it, always, iterates $\tau$ times. STA ER is higher than NFA but does not increase exponentially. Figure 6.7(b) shows the ET as $G_F$ increases. Again the BFA and PSTA ET increase exponentially, while the STA and NFA ET scale well on large graphs.

Furthermore, we evaluate the scalability when $\tau$ increases. Figure 6.7(c) shows how the ER scales as $\tau$ increases. PSTA and STA converge as $\tau$ increases because as the tree search explores more levels, the choices become limited and fewer nodes (runs) are created. BFA ER increases exponentially, and NFA ER increases linearly. Figure 6.7(d) shows how the ET scales as $\tau$ increases. Again it follows the same trend as the ER in Figure 6.7(c).

**Accuracy Evaluation:** Figure 6.8 evaluates the accuracy, which is the difference between the Team DoA value and the optimal one for different graph sizes and $\tau$. Figure 6.8(a) plots the $TDoA$ value for each algorithm with different $G_F$ sizes. BFA and PSTA $TDoA$ results are identical since they output the optimal solution. The first seven runs compare the STA and NFA results to the optimal solution; beyond run seven the computation becomes excessive for BFA and PSTA while the graph size increases. Meanwhile, the other runs only compare the difference between STA and NFA, where higher $TDoA$ value is preferred. Figure 6.8(b) shows the $TDoA$
value for each algorithm as $\tau$ increases. From Figure 6.8(b), the BFA and PSTA TDoA results are identical since they output the optimal solution. We calculate the Mean Absolute Error (MAE) for both STA and NFA, where STA algorithm’s MAE equals 0.0797, and NFA algorithm’s MAE equals 0.1506.

### 6.6.3 Comparison with the Graph Diameter and Density Approaches

We compare the proposed algorithms in terms of Team DoA with two prominent team formation approaches of different objectives. The first approach finds the team with the smallest graph diameter in an effort to minimize the communication cost in a team. This approach is implemented by the RarestFirst algorithm in [50] denoted by Diameter. The second approach finds the team with the maximum density in [52] and denoted by Density.

**Experiment Setup:** The Diameter approach is a minimization problem, where it treats edge weights as distances, i.e., the higher the weight is the farther the distance is, and vice versa. In order to apply the Diameter approach on our dataset, we take the reciprocal of each edge weight, thus, a high edge weight indicates a closer distance, and vice versa. Also, we assign a high weight between not connected vertices as a penalty. On the other hand, the Density approach is a maximization problem, thus, we can apply it directly on our dataset.

Since the RarestFirst algorithm iterates every skill set and selects one member from each set that minimizes the diameter, the algorithm often results in a team cardinality identical to the number of required skills. In contrast, our proposed algorithms have an upper bound for the team size. Therefore, if $\tau > l$, the solution of the RarestFirst algorithm will always have fewer members than our algorithms. Also, if $\tau < l$, the solution of the RarestFirst algorithm will always have more members than our algorithms. Therefore, on each experiment we assign $\tau = l$ as in Figure 6.9(a). On the other hand, the Density approach does not have this restriction.

**Results:** Figure 6.9(a) compares TDoA results for STA, NFA and Diameter as $l$ and $\tau$ increase. It shows that Diameter outperforms NFA in small teams with $\tau \leq 4$ but its performance starts
degrading tremendously on finding larger teams. Similarly, Figure 6.9(b) shows TDoA results for STA, NFA and Density as \( \tau \) increases for constant \( l \). It shows that Density performs similar to NFA in small teams (\( \tau = 5 \)), but its performance starts degrading tremendously on finding larger teams. Moreover, Figure 6.9(c) shows TDoA results for STA, NFA and Density as \( l \) increases for constant \( \tau \). Figure 6.9(c) shows that STA and NFA, consistently, outperform Density. These results show that our proposed approaches outperform the Diameter and Density approaches in finding a well acquainted team members with high connectivity and tie weights.

6.7 Conclusion

In this work, we defined a new DoA based team formation problem and proved that it is NP-hard. We proposed three algorithms, namely PSTA, STA and NFA, to address the problem. We evaluated the proposed algorithms on a real dataset collected from OSS community that consist of over 600,000 developer and over 1,096,000 relationships. The PSTA is proved to find the optimal solution, and STA and NFA demonstrated scalable performance. Also, the proposed algorithms outperform the Density and Diameter approaches in maximizing the Team DoA.
Chapter 7

Conclusion and Future Directions

In this report, a dissertation proposal on recommendation services for Open Source Software community is introduced. We first investigated and studied the OSS community structure and the social factors that affect the amount of participation and commitment in OSS projects. In particular, we conducted an extensive social link analysis on the Social Network (SN) and Affiliation Network (AN) of OSS developers. Moreover, we conducted both global and local networks analysis, where a global network analysis considers a developer’s connectivity in the whole OSS community network, whereas a local network analysis considers a developer’s connectivity within a team network that is affiliated to a project. The analysis demonstrates evident influence of the social factors on the developers’ overall participation and productivity. The data was collected from two most fast-growing OSS collaboration sites, Github.com and Ohloh.net.

Further, we investigated the trends and behavior of developers in the OSS community and the development process. We study and analyze 6 developer features belonging to three categories. Moreover, we study and analyze 9 project features belonging to three categories as well. It is found that a significant ratio of developers share the same affiliation and location with team members, which indicates the existence of a core developers that belong to the same affiliation and sometimes from the same city in some projects. Moreover, we discover certain project features that new developers consider when selecting a project, such as a project complexity and popularity, whereas more experienced developers consider other features such as the primary skill required and the size of a team.

In the expert recommendation service, we targeted two main problems in recommending experts in online open projects. The first problem is related to the responsiveness of experts in collaborating in open projects, and the second problem is related to data scarcity. We mitigate those problems in two aspects, i.e., DoK and SRI. The DoK involves the amount of knowledge or expertise of candidate experts to a given task. The SRI involves the social factor between the candidate experts and a query issuer. We considered three approaches, i.e., SP, kSNDP and RWR, to estimate the SRI and proposed four models, i.e., RAKS, PCBF, SPM and SCPM to estimate the DoK. Moreover, we introduced the ER framework to incorporate the DoK and SRI.
to rank the candidate experts. We conducted extensive experiments on a real historical data set from Github.com of more than a million contributors and more than two million open projects. We evaluated the precision, responsiveness and goodness of ranking of each DoK model and SRI approach combination. We also evaluated the DoK models on different data scarcity level. The results show the effectiveness of the ER in recommending experts that are highly responsive and motivated to contribute their skills.

In the project recommendation service, our goal is to increase the developers’ participation in the OSS projects. There exist two main challenges in recommending OSS projects. First, identifying the factors that developers act upon on selecting projects to join, and second, the data sparsity problem for new community members. Our solutions consider two main aspects in the recommendation, (i) the content of projects and (ii) the developers’ affiliations. The content of projects are dynamic attributes (e.g., team size, popularity, complexity, skills required). For the second aspect, we consider three types of affiliation ties in the recommender models, namely, project co-participation, work and location affiliation ties. We proposed four memory-based models using content and collaborative filtering methods (i.e., PCBF, PCF-PA, PCF-WA and PCF-LA) and one probabilistic model (DAPM). We evaluated our proposal using a longitudinal dataset collected from Github.com over a span of one year. In addition, we test the models under different data scarcity levels. The results show the advantage of DAPM over memory-based models in the case of new or less active members with few data records. Meanwhile, the PCF models have better performance in the case of very abundant data records. Moreover, combining the two aspects of projects’ content and developers’ affiliation in DAPM shows great performance boost in retrieving more and high relevant projects to developers with few data records.

Finally, in the team recommendation service, we defined a new Degree of Acquaintance based team formation problem and proved that it is NP-hard. We proposed three algorithms, namely PSTA, STA and NFA, to address the problem. We were able to recommend teams that cover a set of given skills required for an OSS project and show strong social closeness in terms of ties strength and connectivity among the team members. We evaluated the proposed algorithms on a real dataset collected from OSS community that consist of over 600,000 developer and over 1,096,000 relationships. The PSTA is proved to find the optimal solution, and STA and NFA demonstrated scalable performance. Also, the proposed algorithms outperform the Density and Diameter approaches in maximizing the Team DoA. In the next section, we discuss the future possible directions of this work.

Future Research Directions

There are three main directions that can be considered to continue this work. The first direction is to consider recommending reusable source code (e.g., software component, libraries, packages, code snippets) to developers. Moreover, source code content can be used in the experts and projects recommendation as well. In this case, all OSS projects repositories shall be accessed to
retrieve their source code for indexing and analysis.

The second direction is to consider OSS projects clustering problem. Clustering the projects would facilitate excellent information for recommender services. Knowing the cluster where a project belongs too, we can identify, in more accuracy, the developers who are interested in such collection of projects. In our work, we cluster projects based on dynamic features, however, this field can be highly enriched with more project information such as source code, packages used or even wiki pages if they exist for a project.

The third direction is to further investigate the social ties between developers from different sources such as forums, bug report discussions and wiki pages editors. This type of analysis will discover new type of expertise that can be, later, used for experts recommendation. The field of recommender services for open source software community has many topics and challenges. Proceeding to investigate a research in this area can lead to many research opportunities and achievements.
Bibliography


Vita
Mohammad Yacoub Allaho

EDUCATION

• Doctor of Philosophy, Computer Science and Engineering, December 2014, The Pennsylvania State University, University Park, Pennsylvania, USA.

• Master of Science, Computer Engineering, August 2005, Old Dominion University, Norfolk, Virginia, USA.

• Bachelor of Science, Computer Engineering, December 2003, Old Dominion University, Norfolk, Virginia, USA.

SELECTED PUBLICATIONS


WORK EXPERIENCE

• 2006-2007
  Kuwait University, College of Computing Sciences and Engineering, Computer Engineering department, KUWAIT
  Job Title: Teaching Assistant

• 2005-2006
  International Turnkey Systems - ITS, KUWAIT
  Job Title: System Engineer