Recommending GitHub Projects by Leveraging Developers’ Social Networks and Genetic Algorithm

by

Lance PoKai Wang

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science
with
Specialization in Data Science

Department of Computer Science
Carleton University
Ottawa, Ontario, Canada
2021

Copyright ©
2021- Lance PoKai Wang
The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the Thesis

Recommendating GitHub Projects by Leveraging Developers’
Social Networks and Genetic Algorithm

Submitted by Lance PoKai Wang
in partial fulfilment of the requirements for the degree of
Master of Computer Science

Dr. Olga Baysal, Thesis Supervisor

Dr. Robert Collier, Internal Examiner

Dr. Nafiseh Kahani, External Examiner

Dr. Michiel Smid, Chair of Defence

Dr. Michel Barbeau, Department Chair

Carleton University
2021
Abstract

In the uncommon and challenging year 2020, face-to-face collaboration is difficult. In particular, in the domain of software, online collaboration becomes more demanding than ever. As the leading platform of open-source software, GitHub, without a doubt, plays a vital role in hosting over 190 million of projects and fostering collaboration among developers. Developers in GitHub continuously seek new opportunities to contribute to new projects. So far, several researchers have used users’ historical activities, textual descriptions of projects or starred items to analyze or infer the interests or programming expertise of developers to offer possible project recommendations. While some research utilized social connections to obtain developers’ social importance, none of them have implemented such aspects for generating project recommendations. In this research, we use the latest GHTorrent dataset to construct and propose a GitHub project recommendation system by leveraging Parallel Genetic Algorithm and developers’ social networks. To the best of our knowledge, this is the first application of Genetic Algorithm in the GitHub project recommendation area.
I dedicate this thesis to my mother. Without her love and support, this work would not have been possible.
I, Lance PoKai Wang, would like to express my sincere gratitude to my fantastic supervisor, Professor Olga Baysal, for her continuous guidance, advice, and friendly discussions throughout my thesis. Her precise insights and constant effort provided me valuable feedback and support, which made this work successful.

I also thank Professor Frank Dehne for his advice and guidance on constructing parallel frameworks and also suggesting to work weights-distributed structure.

I am also grateful to Adam Spriggs. He has contributed to this work by implementing the baseline RepoPal, and his work has speed up the model evaluation.

I also thank my laboratory members, especially Norbert Eke and Khadija Osman, for providing insightful feedback for data collection and preprocessing.

I also wish to express my deepest appreciation to all the thesis committee members for their valuable time, insightful comments and encouragement.

Last but not least, I would like to thank my mother, LiChin, my brother, PoJun for their endless love and support in my entire life.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>x</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xi</td>
</tr>
<tr>
<td>Nomenclature</td>
<td>xiii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Open Source Software Contribution</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 GitHub</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 Collaborative Software Development</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Motivation</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Research Questions</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Thesis Contributions</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Thesis Organization</td>
<td>7</td>
</tr>
<tr>
<td><strong>2 Background and Related Work</strong></td>
<td>9</td>
</tr>
<tr>
<td>2.1 Mining GitHub</td>
<td>9</td>
</tr>
</tbody>
</table>
3.3.2.3 Expertise Similarity ........................................ 59
3.3.2.4 Social Importance ........................................ 59

3.4 Recommendation System Evaluation ................................ 59

3.4.1 Baseline: RepoPal ........................................ 59
3.4.2 Qualitative Evaluation ........................................ 60
  3.4.2.1 Survey Design ........................................ 60
  3.4.2.2 Survey Content ........................................ 62
  3.4.2.3 Participant Recruitment ................................ 63

3.5 Tools and Libraries Used ........................................ 64

4 Results .......................................................... 66

4.1 RQ1: How accurately can our framework recommend projects to GitHub developers? ........................................ 66
  4.1.1 Overall Developers’ Preferences ................................ 67
  4.1.2 Strengths and Limitations of Model B .......................... 67
  4.1.3 Strengths and Limitations of Model S .......................... 69
  4.1.4 Limitations of Both Models .................................. 70

4.2 RQ2: How effective is our recommendation approach compared to the baseline? ........................................ 72
  4.2.1 Usefulness of Models ........................................ 74
  4.2.2 Preliminary Effectiveness of Models ............................ 74
  4.2.3 Supporting Developers’ Project Preferences ...................... 75
  4.2.4 Supporting Developers’ Programming Expertise .................. 77
  4.2.5 Perceived Trustfulness of Models .............................. 78

4.3 RQ3: What is the run-time overhead of the GA-based recommendation? ........................................ 79

5 Discussion ...................................................... 84

5.1 Main Findings .................................................. 84
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The basic evolutionary computing metaphor linking natural evolution problem solving.</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>Sketch of the GA.</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>Crossover</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>Top 10 programming languages on GitHub.</td>
<td>38</td>
</tr>
<tr>
<td>5</td>
<td>Attributes of Java projects.</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>Attributes of developers.</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>Attributes of commits.</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>Project profile.</td>
<td>46</td>
</tr>
<tr>
<td>9</td>
<td>Developer profile.</td>
<td>47</td>
</tr>
<tr>
<td>10</td>
<td>ASN and ISN statistics.</td>
<td>51</td>
</tr>
<tr>
<td>11</td>
<td>Weight distribution.</td>
<td>58</td>
</tr>
<tr>
<td>12</td>
<td>Survey content: Model B.</td>
<td>62</td>
</tr>
<tr>
<td>13</td>
<td>Survey content: Model S.</td>
<td>63</td>
</tr>
<tr>
<td>14</td>
<td>Survey content: overview.</td>
<td>63</td>
</tr>
<tr>
<td>15</td>
<td>Survey responses: item rankings for Model S recommendation.</td>
<td>67</td>
</tr>
<tr>
<td>16</td>
<td>Survey responses: code reverse items have been processed.</td>
<td>73</td>
</tr>
<tr>
<td>17</td>
<td>Computational setups.</td>
<td>81</td>
</tr>
</tbody>
</table>
List of Figures

1. Research roadmap .................................................. 6
2. Architecture of RepoPal ........................................... 20
3. Genotype representation ........................................... 26
4. Example of uniform crossover .................................... 28
5. Parallel genetic algorithm’s structure ......................... 29
6. Top 10 GitHub languages and their distribution ............. 36
7. Word cloud: JavaScript ............................................ 36
8. Word cloud: HTML ............................................... 37
9. Word cloud: Java .................................................. 37
10. Commit distribution - before outlier elimination ............ 40
11. Commit distribution - after outlier elimination ............. 40
12. README vector for project ID 126282513 ....................... 46
13. README vector for project ID 107093997 ....................... 46
14. Example of an expertise vector ................................. 48
15. Example of a repo interest vector ............................... 48
16. Empirical study’s setup and overview .......................... 49
17. The concepts of ASN and ISN ................................. 50
18. Parallel Genetic Algorithm’s optimization structure ......... 52
19. Structure of a chromosome ...................................... 53
20. Simulation of the penalty formula .............................. 56
### Nomenclature

#### Abbreviations

This thesis uses some common abbreviations existing in the domain of computer science. The following table lists the abbreviations and their meaning:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>PGA</td>
<td>Parallel Genetic Algorithm</td>
</tr>
<tr>
<td>SGA</td>
<td>Simple Genetic Algorithm</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>SCM</td>
<td>Source Code Management</td>
</tr>
<tr>
<td>PRs</td>
<td>Pull Requests</td>
</tr>
<tr>
<td>EC</td>
<td>Evolutionary Computing</td>
</tr>
<tr>
<td>RS</td>
<td>Recommender Systems</td>
</tr>
<tr>
<td>RSSE</td>
<td>Recommendation Systems for Software Engineering</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Life Cycle</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>OO</td>
<td>Open-source</td>
</tr>
<tr>
<td>EA</td>
<td>Evolutionary Algorithm</td>
</tr>
<tr>
<td>SA</td>
<td>Simulated Annealing</td>
</tr>
<tr>
<td>CF</td>
<td>Collaborative Filtering</td>
</tr>
<tr>
<td>PCC</td>
<td>Pearson Correlation Coefficient</td>
</tr>
<tr>
<td>CSD</td>
<td>Crowd-sourced Software Development</td>
</tr>
<tr>
<td>CPR</td>
<td>Crossover Probability</td>
</tr>
<tr>
<td>GP</td>
<td>Global Parallelization</td>
</tr>
<tr>
<td>VSM</td>
<td>Vector Space Model</td>
</tr>
<tr>
<td>TD-IDF</td>
<td>Term Frequency-inverse Document Frequency</td>
</tr>
<tr>
<td>ASN</td>
<td>Actual Social Network</td>
</tr>
<tr>
<td>ISN</td>
<td>Inferred Social Network</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In this chapter, Section 1.1 introduces GitHub and its open source software (OSS) contribution ecosystem, Section 1.2 discusses our motivation behind this work and Section 1.3 explains our research questions. In Section 1.4, we outline the contributions of this study. Section 1.5 defines the overall structure of the thesis.

1.1 Open Source Software Contribution

Open source software (OSS) is one of the most significant organizational innovation of the 20th century [1], where developers from around the world voluntarily self-organize in virtual teams and coordinate successfully in the absence of price signals and without any pre-specified design rules or formal leadership [2–4].

In this section, we first introduce GitHub (github.com), a collaborative platform for code development. We then describe the ecosystem of GitHub and discuss why and how developers collaborate on projects via GitHub.

1.1.1 GitHub

Being launched in 2008, GitHub [3] is a hosting provider for software development. It offers the distributed version control, source code management (SCM) and other
functionalities. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project [5]. GitHub also offers various features, such as individual user profile, following-followed, activity traces for both users or projects [6]. According to Marlow [7] and Dabbish [8], these features are also utilized to form impressions and coordinate work.

Currently, GitHub hosts over 190 million projects (including at least 28 million public repositories), and more than 40 million developers are engaged in GitHub. In GitHub, small- or middle-scale teams or organizations develop their OSS projects, while large-scale enterprises (e.g., Facebook, Google, Microsoft, etc.) are also sharing their innovative projects with OSS developers.

As the number of developers and projects grows exponentially every year, the value of data mining, social analysis and software engineering in GitHub attracts more attention.

1.1.2 Collaborative Software Development

Nowadays, as the complexity and software size become more gigantic than ever, the growing complexity of software and its increasing diffusion into various application domains require higher quality standards in software development processes [9]. Therefore, in most cases, developers require contributions from the greater user community to develop and maintain their software projects.

In GitHub, collaboration can be manifested in different ways. For example, developers can not only directly contribute their code modification via pull requests (PRs) but also report bugs/issues to project owners. On the other hand, one project owner can assign developers to review PRs or perform code quality assessments. When it comes to collaborative software development in GitHub, developers tend to use profile and past activities to evaluate various metrics in order to find the people to work with
or projects to contribute to. According to Dabbish et al. [10], developers in GitHub make a surprisingly rich set of social inferences from the networked activity information in GitHub, such as inferring someone else’s technical goals and vision when they edit code or guessing which of several similar projects have the best chance of thriving in the long term. Users combine these inferences into practical strategies for coordinating work, advancing technical skills and managing their reputation.

1.2 Motivation

As the number of projects and users of GitHub continue growing, the complexity and diversity of social connection in GitHub increase dramatically as well. While searching for projects, developers often make decisions for choosing their next projects based on their current programming skills and personal project interests. In fact, previous research [11] defines that GitHub users can be separated into two groups: active and less-active. The active user is probably constantly looking for new interesting challenges and maybe even business opportunities, while the less-active user might be looking to grow their collaboration network or just to find a relevant project to work on, without too much effort. Nonetheless, among the sea of repositories, it is time-consuming and energy-wasting to browse projects one by one before the final decision is made. Therefore, recommender systems are commonly introduced and widely-used in GitHub. For example, in the past decade, many studies [12–17] proposed recommender systems for GitHub developers with different categories and purposes such as project recommendation, expert recommendation, PR reviewer recommendation, etc. We found that most research only focuses on historical activity and repository-textual data when it comes to recommending projects or experts. However, according to Dabbish et al. [10,18], social aspects also take a considerable place when developers make inferences of others’ overall abilities.
Previous studies [19] have pointed out that the real-time aspect of such recommender systems can be critical as new data is generated and the old is modified frequently. Furthermore, as projects and users’ scale is increasing rapidly, how recommender systems overcome scalability issues also needs to be considered. That is, to provide the best recommendation, search algorithms need to increase the searching area to find out all potential solutions; however, doing this may result in sizeable computing resource waste. Optimization of algorithms helps to determine the balance. Evolutionary Computing (EC) is the methodology that is widely used for optimization, while Genetic Algorithm is the most popular EC approach for optimization that has been used in different domains such as data science, engineering industry and smart city [21–24]. Genetic Algorithm is also frequently used to combine variables to analyze optimization problems [25].

Considering these facts, we summarize the motivation of this research. First, this research aims to design and propose a new recommender system that utilizes not only developers’ historical activities data, programming expertise, and project interests but also their social connections with others. Secondly, to better optimize the recommendations, this research uses Genetic Algorithm (GA) for solution (i.e., recommendations) finding. Finally, to increase the performance, offer real-time recommendations, and overcome scalability issues, instead of performing a simple GA, our framework utilizes parallel genetic algorithm (PGA) to reduce operation time and increase solution searching area.

1.3 Research Questions

The goal of this thesis is to provide software developers effective recommendations on the projects they can collaborate on based on their programming skills, interests, and
social network. The effectiveness of the recommendations is evaluated via a qualitative study with GitHub developers. While the performance and the applicability of the proposed parallel GA recommender system are assessed via quantitative metrics.

Figure 1 illustrates the road map that we followed in providing answers to the following research questions:

1. **RQ1: How accurately can our framework recommend projects to GitHub developers?** To the best of our knowledge, our proposed framework is the first to utilize Parallel Genetic Algorithm (PGA) for recommending GitHub projects for developers, and it is necessary to evaluate the accuracy of the framework. Therefore, in RQ1, we have sent a recruitment email to 990 GitHub developers to participate in our questionnaire that aims at seeking their perspectives and feedback. We have received 27 consents and obtained 16 valid responses.

2. **RQ2: How effective is our recommendation approach compared to the baseline?** As our proposed framework is a novel GitHub project recommendation approach, thus, we are interested in assessing its effectiveness compared to the existing recommender tools. Therefore, RQ2 aims at comparing our approach with RepoPal [26] by generating two different recommendation lists and analyzing the developers’ feedback on these recommendations. By conducting such comparison, we can determine which approach offers more helpful recommendations and gain insights into the usability of social networks in GitHub for recommendation frameworks.

3. **RQ3: What is the run-time overhead of the GA-based recommendation?** As the number of repositories and users in GitHub continues to grow, inevitably, the computational requirements (e.g., time) for enabling such recommendation tools will increase even more. Therefore, the scalability issue needs
to be tested and addressed. According to Wang et al. [20], historical usage data for recommendation decision-making often update frequently, which may lead to low recommendation efficiency and scalability, especially in the big data environment. Therefore, in RQ3, we measure and study the performance of the proposed framework and compare it with the performance of the baseline model (i.e., RepoPal). Furthermore, we implement our framework with different setups (i.e., configurations) to determine the balance between time costs and optimization levels.
1.4 Thesis Contributions

The key contributions of this research work are as follows:

1. Collection of the data about developers’ commit and social activity on GitHub

2. Leveraging both the developer’s social network and “inferred social network (ISN)”.

3. Building developers’ personal profiles that reflect their expertise and project interests.

4. Proposing a new PGA recommendation system by leveraging developers’ social network in GitHub.

5. Addressing the scalability issue of the recommendation system.

6. A qualitative evaluation of the recommendation system via a survey with GitHub developers.

7. Comparative analysis of the proposed framework against the baseline (RepoPal).

1.5 Thesis Organization

The structure of the remaining chapters in this thesis is as follows. Chapter 2 offers background and presents key concepts related to mining GitHub. It also introduces the utility of recommender systems in GitHub and Genetic Algorithm, as well as reviews relevant related work in this area. Chapter 3 presents the methodology and describes data collection methods and data analysis techniques used in this work. Chapter 4 reports the results and answers to our research questions, while Chapter 5...
discusses the implications of our work. And finally, Chapter 6 concludes this thesis by offering the summary and discussion of future directions for this line of research.
Chapter 2

Background and Related Work

In this chapter, we first provide background on mining GitHub activities and social data in Section 2.1. We then present existing recommender systems in both software engineering ecosystems and GitHub in Section 2.2 and Section 2.3 respectively. Finally, in Section 2.4 we present the brief history of evolutionary computing and related work on genetic algorithms.

2.1 Mining GitHub

The goal of this section is to describe the role that our results can play in the GitHub project recommendation domain. We start the item with the “value of GitHub activities” in Section 2.1.1. Then, we discuss the GitHub social connections (i.e., social activities) and several empirical findings from existing research in Section 2.1.2.

2.1.1 Studying Developers’ Activities

Due to the improved and increasing capabilities of OSS development tools, the number of OSS developers, users and projects is increasing each year. Existing OSS platforms (e.g., GitHub, Altassian’s Bitbucket) involve and engage millions of developers with diverse expertise (i.e., skillsets), characteristics and roles, and a wide range of projects
that provide solutions to various problems for academia and industry.

Among these platforms, GitHub is the largest one, which hosts more than 190 million projects (or so-called repositories) to which over 40 million developers have contributed over the past decade. GitHub provides several features widely-used for developers to manage the projects, such as issues, commits and pull requests (PRs). These features also provide information on the activity of the projects and developers (e.g., commenting on a commit, opening an issue or submitting a PR). Developers not only can broadcast their activities and/or listen to the activities of others but they can also investigate and leverage activities occurring in a variety of projects in one place \[27\]. The knowledge obtained from this information can be used to leverage the metrics of developers and projects. According to Seker et al. \[28\] such metrics can then be used to solve various challenges in software engineering domain. Developers’ activities in GitHub offer the insights of collaboration, learning and reputation management in the community \[8\]. These activities are directly correlated to the reputation or performance of the developers \[29\].

2.1.2 Understanding Social Connections in GitHub

As developers continuously contribute, interact and collaborate on projects on GitHub, communities are formed throughout the projects’ growth. Investigating social coding on GitHub provides an opportunity for researchers to study the impact of programmer networking on software projects \[27\].

From 100,000 selected GitHub projects, Thung et al. \[27\] have extracted the information and analyzed both the project-project network and the developer-developer network. Though project-project network generally follows a powerlaw or long tail phenomenon, developer-developer network shows that social coding can improve collaboration among developers.

Alves et al. \[30\] have proposed a model of social coding collaboration network
to GitHub dataset and three new semantic properties to measure the strength of collaborations. Their key finding indicates that developers who contribute to many repositories do not reflect that they are influential developers (i.e., developers who have a strong social impact or a vast social network). The number of shared repositories is not a significant indicator of the strength of the collaboration.

Tsay et al. [31] have reported a study on open source software contribution in GitHub that focuses on the task of evaluating pull requests. They found that when project managers evaluate pull requests, the submitter who has both good technical contribution practices for a pull request and the strength of the social connection between the submitter and project manager is likely to have their pull request accepted.

Hu et al. [32] have analyzed the data to understand user social influence on the platform. To be precise, they proposed a “Following-Star-Fork-Activity” based approach using users’ follow relation, star and fork relation, and user activity to measure user influence in the GitHub developer’s social network. The authors then performed HITS, PageRank, and H-index to measure users’ influence from different perspectives. They used the classical voting algorithm, Borda count, to synthesize different results and obtain a comprehensive measure of each user’s influence.

Mo et al. [33] have propose an approach named GEMiner to identify experts for specific programming languages in GitHub. When evaluating the expertise of developers, GEMiner considers both the social influence and programming behaviour of developers. They also performed a Multi-Sources PageRank algorithm to model the social influence.

This previous research provides us insights that social connections in GitHub take an important role in project collaboration. To summarize, strong social connections not only bring better social influence but also leave a better impression and reputation for others.
2.2 Recommender Systems in Software Engineering

Recommender Systems (RS) are a subclass of information filtering systems that seek to predict the rating or preference a user would give to an item [34]. The central topic of research in the recommender systems field is to design algorithms that generate personalized ranked item lists. In the software engineering domain, recommender systems are widely-used to solve a broad range of problems.

Developers continually learn new technologies, libraries, and ideas. The systems they are working on have more complex code and rely on larger libraries. Nowadays, mastering a programming language is no longer sufficient to ensure software development proficiency. Developers must also learn to navigate large codebases and class libraries [35]. Recommendation systems for software engineering (RSSEs) are emerging to assist developers in software development life cycle (SDLC) with various activities [36]:

- **Requirement gathering and analysis phase**

  This phase is considered as one of the most critical phases of the software development life cycle since any requirement-related issue may lead to the failure of the software project [37]. The tasks related to eliciting, analyzing and specifying the functional and behavioural properties of the software are accomplished in this phase [38]. Castro-Herrera et al. [39] have proposed a recommender-system-based framework to facilitate the placement of stakeholders into cohesive and finely grained discussion forums. INTELLIREQ [40] provides a group decision environment during the decision process in requirement negotiation. For the small-sized projects, it is able to recommend which requirements should be implemented. Dumitru et al. [41] have developed a recommendation system that
models and recommends product features during the domain analysis process.

- **Design phase**
  
  In this phase, many RSSE applications mainly focus on helping designers decide which design pattern is appropriate for the project. Guéhéneuc and Mustapha [42] have proposed a recommender framework for the decision making of design patterns. The system analyzes the textual description of the design patterns and extracts significant words. Nahar and Sakib have introduced ADPR [43], a framework that recommends design patterns using anti-patterns, where anti-pattern detection is utilized for the recommendation of appropriate design patterns in the software design phase. Issaoui et al. [44] have proposed a design pattern recommender system that is guided by a set of semantic criteria and by the retrieval of the pattern intention that better fits the situation.

- **Implementation phase**

  RSSEs are commonly-used in this phase for supporting developers in a wide range of programming tasks. Common tasks are suggesting source code, finding reusable code, recommending related artifacts, resolving bugs [35]. Cordeiro et al. [45] have proposed a context-based recommendation to support problem-solving in software development. Rahman et al. [46] have developed an IDE-based web search solution that exploits three reliable web search engines and programming Q & A sites through their API endpoints. Their approach facilitates to search information via the web within IDE by considering both the problem content and problem context when developers face various programming errors and exceptions. CodeTube [47] extracts relevant fragments from software development video tutorials. It mixes several existing approaches and technologies like OCR and island parsing to analyze the complex unstructured
contents of the video tutorials and automatically complements the video fragments with relevant Stack Overflow discussions.

- **Testing phase**

Software testing can be an arduous and expensive activity, therefore many frameworks were introduced in order to not only aid developers and test managers to overcome testing issues but also optimizing resource usage. Miranda et al. [48] have introduced a recommender framework for allocating test cases to testers. This system not only can help test managers allocate test cases faster but also can provide recommendations to a new test manager regarding test cases and tester details including the history of previous allocations. Zanker and Nica [49] have proposed a knowledge-based recommendation framework to the domain of selecting appropriate effort estimation methods for software project management.

### 2.3 Recommender Systems for GitHub

Like RSSEs applications, a wide-range of recommender systems have been developed for GitHub to guide or help developers remove development barriers or make correct decisions. In this section, we will introduce and discuss existing research on RSs for GitHub. In Section 2.3.1 we introduce the works related to project recommendation, while Section 2.3.2 is related to expert recommendation applications. In Section 2.3.3 we discuss other relevant applications. Finally, being used as the baseline technique in the thesis, RepoPal [26], a framework that is used to detect similar Java repositories, is described and discussed in Section 2.3.4.
2.3.1 Project Recommendation

Joining a particular project can be based on various motivations among developers [16]. Contributors want to fix bugs, need a development of a particular feature, learn marketable skills or just promote themselves [50–52]. Therefore, project recommendation is the most widely-used RSSEs application in GitHub. It helps newcomers overcome the barriers of building their own reputation and portfolios and guide experienced developers to sharpen and extend their skillsets and social networks by making more contributions. In this section, we will cover some state-of-the-art project recommender frameworks and empirical insights about recommending GitHub projects.

When it comes to recommending projects in GitHub, different metrics will be performed based on different models or frameworks. Seker et al. [28] have applied collaborative filtering using item similarity logic model to evaluate different metrics. They found that commenting on any feature was as important as code contributions. Issue-related activities were also highly important in developer metrics. Zhang et al. [53] have proposed a user behaviour-based project recommendation framework. In their study, they found that fork, watch, pull-request and member are suitable for finding relevant projects, while the issue comment data is not. The basic idea of this approach is to measure the relevance between two projects by comparing how similar the user behaviour data on the two projects are. This approach brings another interesting insight that different user behaviour data are suitable for different recommendation purposes. Liu et al. [15] have proposed a learning-to-rank-based model which helps developers by recommending relevant projects for onboarding. They extracted nine features from projects that could be categorized into five categories: social tie, technical abilities, project growth, user profile and project time. All features were used for performing neural networks for predicting final grades for all candidate projects.
Xu et al. [17] have proposed an Apache Spark-based scalable software project recommendation on GitHub. This framework analyzes developers’ behaviours and extracts descriptions and source code from projects to build user-project matrix via TF-IDF and cosine similarity. Parallel computing is also applied for this framework in order to overcome the scalability issue of GitHub projects. Similar to [17], Sun et al. [54] have considered not only developer behaviours and project features, but user feedback to improve recommendations. The model also performs Simulated Annealing (SA) to automatically optimize parameter configurations to achieve higher accuracy. Sharma and Mahajan [55] have utilized collaborative filtering (CF) for recommending GitHub projects. Pearson correlation coefficient (PCC) has been performed for feature extraction from selected nine features of user activities. Cosine similarity is then used for computing and finding the top-k similar users and recommending their projects.

2.3.2 Recommending Experts

Finding proper people to conduct certain tasks in GitHub has equal importance compared to finding proper projects to join. In GitHub, experts recommendation is commonly used to find code reviewers, project co-workers or pull-request contributors. In this section, we highlight previous works on expert recommendation.

Yu et al. [12] have evaluated 84 code reviewer recommendation frameworks and proposed their own method by adding social networks aspect. They found that though traditional methods to bug tracking and code review are feasible for pull-request reviewer recommendations, the performance can be improved significantly by combining them with information extracted from prior social interactions between developers. These results prompt for novel tools to support process automation in social coding platforms that combine social (e.g., common interests among developers) and technical factors (e.g., developers’ expertise). Based on these findings, Yu et al. [13] have
proposed VSM (Vector Space Model) based PR reviewer recommendation by extracting textual contents (i.e., title and description) of PR and using TF-IDF approach to represent it as a weighted vector. Semantic similarity, expertise score, comment network and common interests are later used to recommend the most appropriate code reviewer. Rahman et al. [14] have proposed a code reviewer recommendation for PR named “CORRECT”. It captures the experience of potential reviewers with the external libraries (i.e., cross-project experience) and specialized technologies used in a pull request for reviewer recommendation. Ying et al. [56] have proposed a code reviewer recommendation structure by considering both expertise and authority. To speed up the performance, their approach employs LSI model to mine the semantic similarity to model expertise and calculate the number of PRs which reviewers have commented together to represent authority. Zanjani et al. [57] have proposed “cHRev”, an automatic recommendation model for peer reviewer recommendation in modern code review. Their findings demonstrate that analyzing specific information available from previously completed reviews (i.e., quantification of review comments and their recency) can add value for peer reviewer recommendations. Though they argue that the developer recommendation approach based on past source code commits was inadequate in effectively supporting this task, the general principles of frequency, workdays, and recency from such a developer recommendation approach are transformative.

DevRank [58] utilizes the data of different user behaviours such as “commit” and “follow” in order to rank the most influential developers effectively. PageRank [59] is then used to measure the influence of developers and find the most influential developers. Though the influence of a developer can be measured by the capacity to attract followers, the number of followers that the developer can obtain measures how successful the developer has been in spreading their influence. The experiment has shown that compared with the traditional PageRank and HITs [60], DevRank
achieves significant improvement.

Other than recommending code reviewers, the expert recommendation in GitHub can also be used to find the proper developers for software development. In crowdsourced software development (CSD) area, tasks require reliable development and sufficient participation to ensure a qualified asset that can be delivered to the client. Mao et al. [61] have proposed a developer recommendation framework that utilizes content-based recommendation techniques for developer recommendation in the emerging CSD context via mining registration historical data.

Expert recommendations can also be used to recommend bug fixers. Xia et al. [62] have proposed a developer recommender system for bug resolution which utilizes both bug report-based analysis (BR-based analysis) and developer-based analysis (D-based analysis) by using K-Nearest Neighbors to find out the nearest bug reports and possible developers.

### 2.3.3 Other Applications

Despite project and expert recommendation are being two popular areas that are widely researched, many recommendation frameworks have been developed for GitHub in supporting developers’ tasks.

Contributors make code changes and submit PRs to projects. In order to save project reviewers’ time, tags are widely used to attach additional information to PR and facilitate their organization. Jiang et al. [63] have provided insightful information that tags of PRs are useful for developers to track, search or classify PRs. But some developers found that it is difficult to choose the right tags and keep the consistency of tags. Therefore, they propose “FNNRec”, a tag recommendation for PRs that analyzes titles, descriptions, file paths and contributors, and utilizes feed-forward neural networks to compute probabilities and recommend tags.

Tesseract [64] analyzes information from code archives, communication records,
and bug repositories to capture the relations between code, developers, software bugs, and communication records for recommending issues/bugs to appropriate developers. The Tesseract search feature uses a synonyms thesaurus to include synonyms of queried terms, providing a larger set of related bugs to developers, and cosine document similarity is used to rank the resulting set of bugs.

2.3.4 RepoPal

As the baseline techniques used in this thesis, RepoPal [26], a similar GitHub repository detection framework, is described and introduced in this section.

RepoPal, proposed by Zhang et al., is used for detecting similar GitHub Java repositories. According to the findings of Mockus [65], large proportion of open-source (OO) projects on GitHub are similar as more than 50% of source code files are reused in more than one open-source project. Though GitHub provides a search engine to help developers find relevant repositories among the millions of projects, this search engine is certainly not an ideal tool as it only offers a simple text-based tool. Based on these reasons, there is an urgent need for developers to have a tool that can detect similar repositories on GitHub. Moreover, similar project detection can be used not only for recommender system development but also for code reuse, rapid prototyping, identifying alternative implementations and discovering code theft and plagiarism [66–68].

Figure 2 illustrates the architecture of RepoPal which consists of four components:

1. **Readme Relevance Calculator:** The hypothesis behind this is “repositories whose README files contain similar contents are likely to be similar with one another.” Readme Relevance computes the readme-based relevance score of two repositories by comparing their README files via Vector Space Model (SVM) and TF-IDF. SVM is commonly used to calculate similarities between textual
corpus and transform them into vectors of weights. Each pre-processed word corresponds to an element in the vector and its weight is calculated via TF-IDF weighting scheme. The formula of TF-IDF is explained in detail in Chapter 3. Given two repositories and their two representative vectors of weights, we can compute their readme-based relevance score, denoted as \textit{Relevance}(R_1, R_2), by taking the cosine similarity of their representative vectors as shown in equation 1:

\[
\text{Relevance}_r(R_1, R_2) = \frac{\sum_{t \in R_1 \cap R_2} w_{t,R_1} \times w_{t,R_2}}{\sqrt{\sum_{t \in R_1} w_{t,R_1}^2} \times \sqrt{\sum_{t \in R_2} w_{t,R_2}^2}}
\]  

(1)

2. \textbf{Stargazer Relevance Calculator:} The Stargazer Relevance Calculator component utilizes GitHub stars to rank repositories. This component is based on the heuristic that “repositories starred by users of similar interests are likely to be similar.” It is used to calculate stargazer-based relevance score between two repositories. It relies on the user similarity formula. Notations are: \( R \) denotes a GitHub repository, \( U \) denotes a single GitHub user, \( S(R) \) denotes all the users who starred \( R \), and \( S(R) \) denotes all the repositories a user starred. Given a user \( U_1 \) who starred repositories \( S(U_1) \), and a user \( U_2 \) who starred repositories
$S(U_2)$, the similarity score between two users is shown in equation 2:

$$Sim(U_1, U_2) = \frac{|S(U_1) \cap S(U_2)|}{|S(U_1) \cup S(U_2)|} \quad (2)$$

Given two repositories $R_1$ and $R_2$ which are starred by users $S(R_1)$ and $S(R_2)$, the stargazer-based relevance score can be calculated using the equation 3 shown below:

$$Relevance_s(R_1, R_2) = AVG_{U_i \in S(R_1), U_j \in S(R_2)} Sim(U_i, U_j) \quad (3)$$

The stargazer-based relevance score is computed as the average score of the user pairs’ similarity score. For two repositories, when the similarity scores of two repositories are higher, the relevance score is also higher.

3. **Time Relevance Calculator:** It is based on the hypothesis that “repositories starred together within a short period of time by the same user are likely to be similar.” Before computing the time relevance score, new notations are required to be introduced: given the fact that user $U$ starred repositories $R_i$ and $R_j$, we can use $D(U, R_i, R_j)$ to represent the difference in time at which user $U$ starred two repositories rounded up to the nearest number of hours. The time relevance score between repositories $R_1$ (starred by users $S(R_1)$) and $R_2$ (starred by users $S(R_2)$) can be computed via the following (Equation 4):

$$Relevance_t(R_1, R_2) = AVG_{U_i \in S(R_1) \cap S(R_2)} \frac{1}{|D(U_i, R_1, R_2)|} \quad (4)$$

For two repositories, when the time differences are smaller, the relevance score is higher. If the intersection is empty, the time-based relevance score is set to 0.
4. **Composer:** The composer component computes the overall relevance score of all three relevance metrics from Equation 1, Equation 3 and Equation 4 for two repositories \( R_1 \) and \( R_2 \) as follows:

\[
Relevance(R_1, R_2) = Relevance_r(R_1, R_2) \times Relevance_s(R_1, R_2) \times Relevance_t(R_1, R_2)
\]

In this thesis, we use RepoPal as a baseline and state-of-the-art technique when we compare it with our new GA-based recommender system. RepoPal can be considered as an activity-based recommendation while our new technique is a social-based recommender system.

### 2.4 Evolutionary Computing

This section provides the necessary background and explains the basics of evolutionary computing (EC). We begin with a brief history of the field of evolutionary computing. Next, we offer introduction to some of the biological processes that have served as inspirations to researchers. We then discuss the genetic algorithm (GA) and parallel genetic algorithms (PGAs) in Section 2.4.1 and Section 2.4.2 in detail. We then offer examples of applications where GA and PGAs have been successfully applied in software engineering domain in Section 2.4.3.

Evolutionary computing is one of the research areas in computer science. As the name suggests, it has a special flavour of computing that was inspired by natural evolution, and this concept was adopted in many algorithms that are called Evolutionary Algorithms (EA). Some computer scientists have referred to natural evolution as a
source of inspiration: the power of evolution in nature is evident in the diverse species that make up our world, each tailored to survive well in its own niche.

Considering natural evolution simply as follows. A given environment is filled with a population of individuals that strive for survival and reproduction. The environment determines the fitness of these individuals which relates to how well they succeed in achieving their goals. That is, it decides their chances of survival and of multiplying. Meanwhile, in the context of a random trial-and-error-style problem-solving process, we have a set of candidate individuals (i.e., solutions). Their quality (i.e., how well they solve the problem) determines their chance of being kept and used as seeds for building further candidate solutions. The relations between these two scenarios are shown in Table 1.

Table 1: The basic evolutionary computing metaphor linking natural evolution problem solving.

<table>
<thead>
<tr>
<th>Evolution</th>
<th>Problem Solving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>↔</td>
</tr>
<tr>
<td>Individual</td>
<td>↔</td>
</tr>
<tr>
<td>Fitness</td>
<td>↔</td>
</tr>
<tr>
<td>↔ Problem</td>
<td></td>
</tr>
<tr>
<td>↔ Candidate solution</td>
<td></td>
</tr>
<tr>
<td>↔ Quality</td>
<td></td>
</tr>
</tbody>
</table>

The idea of applying Darwinian principles to automated problem-solving dates back to the 1940s. In 1948, Turing proposed “genetical or evolutionary search”, and by 1962 Bremermann had actually executed computer experiments on “optimization through evolution and recombination”. During the 1960s, Fogel, Owens, and Walsh introduced evolutionary programming, while Holland called his method a genetic algorithm (GA).
2.4.1 Genetic Algorithm

Genetic Algorithm (GA) is the most popular class of EAs, and most GA computations it can be reproduced in other EAs and population-based techniques \[74\]. GA is widely used for solving and optimizing problems such as drug development, filling your fridge with healthy food at low cost, combat a fire, program your computer with a minimum number of bugs. GA was initially conceived by Holland as a means of studying adaptive behaviour, as suggested by the title of the book describing his early research: *Adaptation in Natural and Artificial Systems* \[72\]. The characteristics of GA are (1) binary representation, (2) fitness proportionate selection, (3) a low probability of mutation, and (4) an emphasis on genetically inspired recombination as a means of generating new candidate solutions. It is summarized in Table 2.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Bit-strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recombination</td>
<td>1-Point crossover</td>
</tr>
<tr>
<td>Mutation</td>
<td>Bit flip</td>
</tr>
<tr>
<td>Parent selection</td>
<td>Fitness proportional, implemented by Roulette Wheel</td>
</tr>
<tr>
<td>Survival selection</td>
<td>Generational</td>
</tr>
</tbody>
</table>

The main genetic operators, which are described in the following paragraphs, are (a) genotype representation (b) selection (c) crossover and (d) mutation. Many other operators have been also proposed and used. In this section, we will only describe Simple Genetic Algorithm (SGA) by introducing the three steps mentioned above.

1. **Genotype Representation:** The first yet the most important decision to make is deciding on how to represent solutions (i.e., chromosomes). In fact, improper representation can lead to poor performance of the GA \[73\]. There are several commonly used genotype representations.
- **Binary representation** is the simplest and most widely used representation in GAs. In this type of representation, the genotype consists of bit strings.

For some problems when the solution space consists of boolean decision variables – yes (true) or no (false), the binary representation is natural. As shown in Figure 3(a), if there are \( n \) items, we can represent a solution by a binary string of \( n \) elements, where the \( x \)th element defines whether the item \( x \) is picked (1) or not (0).

- **Real valued representation** is used when researchers want to define the genes using continuous rather than discrete variables. The precision of these real valued or floating point numbers is however limited to the computer. Figure 3(b) presents an example that each gene receives a real value via fitness function and this can be used to compare to other genes.

- **Integer representation** is used when researchers face problems that binary space 'yes' or 'no' is insufficient. This approach may bring more encode spaces. Figure 3(c) shows a example when genes need to use integer to present four directions (i.e., East(1), South(2), West(3) and North(4)).

- **Permutation representation** is used in many real-world problems. By using this approach, the solution is represented by an order of elements. Figure 3(d) presents a classic travelling salesman problem (TSP). The salesman has to take a tour of all the cities, visiting each city exactly once and come back to the starting city. The total distance of the tour has to be minimized. Using permutation representation makes sense for this problem.

2. **Selection:** In the GA framework, selection is used to produce the intermediate population, which consists of copies of current generation chromosomes.
The number of each chromosome’s copies depends on its fitness score. The most commonly-used selection approaches are (a) Roulette Wheel Approach, (b) Tournament Approach and (c) Elitist Approach.

- **Roulette Wheel Approach:** This process is based on the idea of a roulette wheel with PS unequal slots, with sizes proportional to the fitness of the respective chromosomes. Then, each time the roulette wheel is “spinned”, the selection probability \( p(i) \) of a chromosome \( i \) is equal to:

\[
p(i) = \frac{FV(i)}{SFV}
\]  

where \( SFV \) is the sum of the fitness values of all chromosomes of current generation. Roulette wheel spinning is iterating for PS times to select PS chromosome copies. The individuals with higher fitness scores have more copies on the wheel, while low-score individuals have fewer copies; however, even the worst chromosome has a tiny chance to “survive”, namely to be copied to the intermediate population.

- **Tournament Approach:** The idea of the tournament approach is based on contests between KK individuals, where KK is a pre-defined number. The following iteration is repeated PS times: KK chromosomes of the
current generation are randomly selected, and their fitness values are compared to each other. A copy of the chromosome with the best fitness value is added to the intermediate population. The value of KK usually ranges between 3 and 5. Increasing KK value would reduce the “survival” probability of weak solutions, namely it increases selective pressure [75].

• **Elitist Approach:** The above two selection approaches do not guarantee a copy of the best chromosome will be preserved or included in the next iteration. To overcome this issue, the elitist approach is introduced to include an additional procedure of passing at least one copy of the best chromosome (i.e., best solution) to the next generation. While some research tends to pass two copies to the next generation, and one to freely participate in the evolutionary process.

3. **Crossover:** Crossover is a gene recombination process. It applies to a pair of parental chromosomes. Each parent chromosome can participate in this “reproduction process” with a “crossover probability (CRP, usually larger than 0.5)”.

Crossover can be performed in different formats. The basic one is the single-point crossover (as shown in Table 3(a)). The operator cuts each parent chromosome into two pieces, then parents exchange their pieces in order to produce the offspring.

Direct extensions of the basic crossover process are two or multiple-point crossover (as shown in Table 3(b)). As their name implies, parent chromosomes are cut into two or more pieces, which they exchange, in order to produce the offspring [75]. In addition, *uniform crossover* is another commonly used approach that does not divide the chromosome into segments, rather treats each gene separately. By doing this, each gene has a greater chain to perform crossover, and
due to each gene being treated separately, the diversity of chromosomes can be
increased. Figure 4 provides an example of uniform crossover; we can observe
that each gene is considered individually and does not affect the previous or
next ones.

Table 3: Crossover: (a) one-point crossover, (b) two-point crossover (slashes indicate
crossover points).

<table>
<thead>
<tr>
<th></th>
<th>(a) Single-point Crossover</th>
<th>(b) Two-point Crossover</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent A</td>
<td>11000101110/101111000</td>
<td>11/000101110101/111000</td>
</tr>
<tr>
<td>Parent B</td>
<td>01110011101/001101101</td>
<td>01/110011101001/101101</td>
</tr>
<tr>
<td>Offspring A</td>
<td>11000101110/001101101</td>
<td>11/110011101001/111000</td>
</tr>
<tr>
<td>Offspring B</td>
<td>01110011110/101111000</td>
<td>01/000101110101/101101</td>
</tr>
</tbody>
</table>

Figure 4: Example of uniform crossover.

4. **Mutation:** Mutation applies to offsprings. It brings new genetic materials
with two meanings: (a) extending searching areas of the solution space, and (b)
local refinement of good solutions. However, as mutation happens randomly, the
evolution process could be better or worse. That is, the quality of solutions could
be fluctuating. Application to binary GA is fairly simple: the gene selected for
mutation (i.e., mutation point) is changed from 0 to 1 and vice versa.

### 2.4.2 Parallel Genetic Algorithm

GA is well suited for applying parallelization because of its population-based ap-
proach, where all solution candidates can be dealt with in parallel. According to [74],
PGAs can be applied into the following six domains: implementation, hardware, software, API, applications and problem domain.

The implementation of PGAs is mainly three kinds of models (as shown in Figure 5): global parallelization, island model, and cellular model. The following sections explain these in detail.

1. **Global Parallelization**

Global parallelization (GP) is the most straightforward implementation of a PGA. In Figure 5(a), black circles indicate individuals (i.e., solutions). In GP, intermediate solutions are managed by a centralized population, and each solution is evaluated in parallel. “Generational” and “Steady-state” are two standard classifications of GP. The generational approach generates a new population after all slave nodes complete their tasks. However, the steady-state approach generates a new solution immediately after one slave node completes its task and returns results. Master-slave parallel architecture is generally deployed to implement GP, where the master node manages the population and sends evaluation tasks to slave nodes. The slave nodes perform tasks given by the master node and return the results.

2. **Island Model**
Figure 5(b) illustrates the structure of the parallel island (or multi-population) model. The parallel island model is a PGA that manages several sub-populations in separate islands and executes the GA procedure in each island in parallel over a different set of solutions. Unlike the global parallelization, which manages a centralized population, the island model manages a set of decentralized populations.

To implement it, each island is assigned to one computing element (i.e., CPU cores or Nvidia CUDAs). At every migration step, a form of solution information is exchanged (migrated) to another CPU following some sort of communication strategy, e.g., shared memory or message passing, according to migration topology [74].

3. Cellular Model

The parallel cellular (or fine-grain) (Figure 5(c)) model arranges solutions on a grid structure, and the replacements are executed to solutions within a defined neighbourhood. Cells can be updated synchronously or asynchronously [76]. Synchronous updating updates all solutions simultaneously, while asynchronous updates cells at a time in some order. Furthermore, several updating strategies for asynchronous updating exist such as fixed-line sweep, fixed random sweep, new random sweep, and uniform choice [77].

2.4.3 Genetic Algorithm in Software Engineering

In this section, we describe GA applications in software engineering.

RevRec [23] is a peer code reviewers recommendation framework that utilizes GA to evaluate the appropriate peer reviewer to be invited or assigned to review code changes based on their expertise and collaboration history in the past reviews. According to the authors, the combination of expertise and social-technical relationships
based recommendation is more accurate and effective in practice.

Shirvani and Gorji [78] have proposed a genetic-based algorithm for combining cloud services that ensure multiple clouds work efficiently. This framework is able to provide an overview of the weaknesses of other available methods in terms of computational complexity in the automated selection of web services and makes it possible to fulfill the demands of the composition of web services in a more optimal way.

Azar et al. [79] have proposed a GA-based approach for combining and/or adapting existing software quality predictive models (experts) to a particular context. For each given task, this approach is able to assign the appropriate expert to deal with.

Uguz [21] has proposed a two-stage feature selection approach to reduce the high dimensionality of a feature space composing of a large number of terms, remove redundant and irrelevant features from the feature space and thereby improve the performance of text categorization. Each term within the text is ranked depending on their importance for the classification using the IG method classification in the first stage. In the second stage, GA and PCA are performed for feature selection and feature extraction, which are applied separately on the terms, then are ranked in decreasing order of importance, and a dimension reduction is carried out.

Gangavarapu and Patil [22] have proposed a penalty-based filter–wrapper hybrid greedy ensemble approach to facilitate optimal feature selection approach. This approach greedily selects the features from the subspaces obtained from the predetermined base selection methods. Performance-dependent penalty parameters were used to penalize the base feature subspaces essential to achieve the optimal ensembling of those subspaces. Furthermore, the authors leverage effective heuristic search strategies including the greedy parameter-wise optimization and the GA to obtain optimal values of the penalty parameters.

Seng et al. [80] have proposed a single-objective optimization-based approach using
GA to improve software quality via suggesting a list of refactorings. The search process utilizes a single fitness function to maximize a weighted sum of several quality metrics. The used metrics are coupling, cohesion, complexity and stability, which are mainly related to various class-level properties. These conditions serve at preserving the program behaviour.

Antoniol et al. [81] have developed a GA-based refactoring process to reduce the size and minimize the coupling of libraries by incorporating dynamic information into clustering and GA. The authors also optimized libraries with respect to a given user profile. However, different profiles would have lead to different libraries, which makes this mechanism appealing for generating customized software configurations.
Chapter 3

Methodology

This chapter describes the approach we adopted to construct a PGA-based GitHub repository recommender system. In Section 3.1, we describe the datasets we collected and used in this thesis, while Section 3.2 presents the data pre-processing, pre-calculation, profile building, and analysis techniques. Section 3.3 introduces our proposed framework, and Section 3.4 describes the techniques used to evaluate the proposed framework with the baseline and the design and setup of a qualitative study via a survey with actual developers. Finally, Section 3.5 reports on the software and tools used in this work.

3.1 Data Mining

In this work, we extracted and collected the GitHub user and project data from GHTorrent (ghtorrent.org) and gathered the commit data via GitHub API (api.github.com).

This research aims to propose a recommender system by analyzing user’s historical activities and social connections. Thus, we opted for mining the data from GHTorrent. Unlike GitHub API, GHTorrent provides “backup” data publicly, while the former offers “live” data. Though it is possible to mine all data via GitHub
GitHub enforces a query limit (i.e., 5,000 queries per hour). It does offer paid membership for researchers or enterprises (i.e., 25,000 queries per hour), however, it is still insufficient for researchers to collect the data using API. Considering that we need to collect around 1.7 million Java repositories and 705 thousand user data, it is infeasible to collect such dataset using GitHub API. Therefore, we choose to mine user and repository data from GHTorrent within the time range from July 2017 to June 2019 to collect the most accurate past data of GitHub. The reason for narrowing down the time range to two years is that developers’ interest may evolve over time; we wanted to avoid the waste of storage and computing resources for collecting the complete history of GitHub that goes back to 2008 (i.e., the year GitHub was founded). While collecting user commits, GHTorrent cannot provide a “tree structure” of commit data (i.e., we can not collect file modifications or commit’s author); therefore, we use GitHub API to collect the data about 1.5 million commits.

3.1.1 User and Project Selection

In this section, we introduce our user and project selection criteria and explain the reasons behind this selection.

Project Selection: When it comes to choosing repositories from GHTorrent, it is essential to understand which language is widely used for development. The number of projects for the specific language can be used as an indicator for evaluating the popularity of that language.

Table 4 and Figure 6 show GitHub’s top 10 languages and their number of projects developed in these languages. Almost 3 million repositories (19.9%) use JavaScript in their development, while 1.8 million projects (12.4%) are developed in HTML. It seems that limiting our dataset to JavaScript would be ideal as it can potentially provide us with the most repositories and related developers. To further validate
whether JavaScript or HTML could be the better language choice for our study, we perform TF-IDF-based analysis for project descriptions of JavaScript, HTML and Java projects. We use project description as it serves as a brief summary of the project, and project owners tend to use the shortest yet richest sentences to describe their project. Therefore, project descriptions can offer us the most valuable corpus to perform the analysis. Even if only trivial information is provided, TF-IDF is able to detect most important terms while removing less meaningful ones.

We plot the result in Figures 7, 8 and 9 to represent the keywords (i.e., applications, techniques, frameworks, etc.) inside JavaScript, HTML and Java projects, respectively. From Figure 7 we can see that the range of applications in JavaScript is fairly limited. For example, the most common applications are jQuery, Rails, AngularJS and Vue. HTML, as shown in Figure 8 on the other hand presents a poorer result as it does not have any specific or well-known application.

Last but not least, while Java does not account for the highest number of projects in GitHub, the application range is the widest compared to the other two languages. For example, as shown in Figure 9 Spring Boot, Apache, Android, Google, VPN and even HTML are very common repositories and topics for Java.

Based on these findings, Java would be the best target language in this research compared to JavaScript and HTML as it provides the widest range of applications and industry usage.

**User Selection:** Similar to the project selection, we need to scale down the GitHub data and narrow down our user population. To accomplish this, we first build Java repository pool from all 1.7 million Java repositories with four constraints:

1. **With Java language:** Our proposed approach can only recommend Java projects, since only Java projects are included in our dataset.

2. **Two-year range:** GitHub was founded back in 2008 and includes 13 years of
Figure 6: Top 10 GitHub languages and their distribution.

Figure 7: Word cloud: JavaScript.
Figure 8: Word cloud: HTML.

Figure 9: Word cloud: Java.
Table 4: Top 10 programming languages on GitHub.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Language</th>
<th>Number of Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JavaScript</td>
<td>2,989,270</td>
</tr>
<tr>
<td>2</td>
<td>HTML</td>
<td>1,871,587</td>
</tr>
<tr>
<td>3</td>
<td>Java</td>
<td>1,735,903</td>
</tr>
<tr>
<td>4</td>
<td>Python</td>
<td>1,601,680</td>
</tr>
<tr>
<td>5</td>
<td>Ruby</td>
<td>913,594</td>
</tr>
<tr>
<td>6</td>
<td>PHP</td>
<td>641,642</td>
</tr>
<tr>
<td>7</td>
<td>CSS</td>
<td>639,750</td>
</tr>
<tr>
<td>8</td>
<td>C++</td>
<td>620,864</td>
</tr>
<tr>
<td>9</td>
<td>C#</td>
<td>509,890</td>
</tr>
<tr>
<td>10</td>
<td>Jupyter Notebook</td>
<td>409,550</td>
</tr>
</tbody>
</table>

user activity data. Therefore, it is possible to restore and analyze all data since the platform was launched. However, as developers’ interests may evolve and change rapidly, it would be a resource waste to obtain all historical data. Since the latest version of GHTorrent was released in June 2019, we only include and consider the two-year range of data, that is, only user activities that occurred from July 2017 to June 2019 are considered and included in the repository pool.

3. **Non-solo projects:** GitHub hosts a diversity of projects, from solo projects to the scale of thousands of developers per project. As the purpose of the user selection step is to extract the appropriate users that have contributed to multi-developer projects, it is necessary that we eliminate developers who contributed to solo or so-called toy projects. That is, the projects which have only one contributor are excluded from the project pool.

4. **With some popularity:** As the social aspect is the key element of our proposed recommender system, it is important that selected users at least have
some social connections. Therefore, all projects in the dataset must have “a
certain level of popularity”. To quantify this attribute, we set the limit that all
projects must have 5 watchers to exist in the project pool.

After preliminary repository pool selection, we can guarantee that the projects
in the repository pool have a minimum quality as we narrow down the time range
to eliminate trivial (i.e., outdated) commit data, exclude solo projects and ensure all
projects have at least five watchers. Finally, there are 16,895 projects in the repository
pool. Next, we perform the following user selection approach:

1. **Outlier elimination:** We first analyze the number of commits that projects
have during the two-year range. In Figure [10] we can see that while the average
number of commits is 292, the median (Q2) is 49. This trend indicates that out-
liers are skewing our data. We then investigate the projects with an “extreme”
number of commits and found that those projects are mainly related to bot or
business issue-tracking projects, which we perceive as bringing minimal value
for developers when recommending GitHub projects since developers would not
want to join these projects. Therefore, outlier elimination is necessary prior to
user selection process. Thus, we remove the top 5% of projects (i.e., top 5% of
projects with the highest commit number) and perform a similar analysis for
the remaining projects. Figure [11] illustrates better results. The average num-
ber of commits is now 122, and the median (Q2) is 43. By eliminating outliers,
we are able to remove bot-like projects of little value for developers.

2. **Random project selection and user extraction:** From the cleaned reposi-
tory pool, we randomly select 1,000 projects and all their users. Finally, we
have obtained 9,697 developers as our target users.
3.1.2 Data Collection

Our research’s scope is limited to 1.7 million Java repositories and focuses on generating recommendations for 9,697 target developers.

1. **Data collection of Java projects:** For Java projects, we collect all 1,735,903 projects and their attributes (as shown in Table 5). We need to note that the attribute ‘readme_text’ is the original textual data extracted from GitHub pages. That is, it is un-preprocessed data and is further mentioned in Section 3.2.1.
Table 5: Attributes of Java projects.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>(Int) Unique ID of a repository</td>
</tr>
<tr>
<td>url</td>
<td>(String) GitHub page of a repository</td>
</tr>
<tr>
<td>owner_id</td>
<td>(Int) The ID of a repository owner</td>
</tr>
<tr>
<td>name</td>
<td>(String) The name of the project</td>
</tr>
<tr>
<td>description</td>
<td>(String) Textual description of the repository</td>
</tr>
<tr>
<td>language</td>
<td>(String) The main language used for project development</td>
</tr>
<tr>
<td>create_at</td>
<td>(DateTime) The time the repository was created at</td>
</tr>
<tr>
<td>delete</td>
<td>(Boolean) If the repository is deleted or not</td>
</tr>
<tr>
<td>readme_text</td>
<td>(String) The textual content of repository’s README file</td>
</tr>
<tr>
<td>watchers</td>
<td>(Array) An array that contains the watcher (who stars the repository) list of the repository, including user ID and timestamp</td>
</tr>
<tr>
<td>topics</td>
<td>A list that contains all pre-defined topics of the repository</td>
</tr>
<tr>
<td>update_at</td>
<td>(DateTime) The last time the repo was updated at</td>
</tr>
</tbody>
</table>

2. **Data collection of target developers:** For 9,697 target developers, we retrieve their data via both GHTorrent and GitHub API. All collected attributes are shown in Table 5.

3. **Data collection of other Java users** As the social network is a critical aspect that we leverage in the proposed framework, we also collect the data about all users from 1.7M Java projects, as shown in Table 6.

### 3.1.2.1 Commit Features

For 9,697 target developers, we collect their Java-related commits from July 2017 to June 2019. Attributes are listed in Table 7.

For our target developers, we build their expertise vectors based on their commit records. More precisely, we investigate file extensions of their commits to construct such vectors as indicators for representing developers’ skill sets. Detailed construction steps are introduced in Section 3.2.1.
### Table 6: Attributes of developers.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>(Int) Unique number for a developer</td>
</tr>
<tr>
<td>login</td>
<td>(String) Unique string for a developer; also used as log-in credentials</td>
</tr>
<tr>
<td>company</td>
<td>(String) The business organization that developer works for</td>
</tr>
<tr>
<td>create_at</td>
<td>(DateTime) The time of the developer profile was created</td>
</tr>
<tr>
<td>type</td>
<td>(String) The type of developer; for this work, all target developers are “USR”</td>
</tr>
<tr>
<td>fake</td>
<td>(Boolean) If the developer account is fake or not.</td>
</tr>
<tr>
<td>deleted</td>
<td>(Boolean) If the developer account is deleted or not.</td>
</tr>
<tr>
<td>email</td>
<td>(String) The email address of a developer; used for sending out the survey</td>
</tr>
<tr>
<td>social_relation</td>
<td>(Array) The list of ’following-followed’ relations of a developer (as described in Section 3.1.2.2)</td>
</tr>
<tr>
<td>repo_participation</td>
<td>(Array) The list of projects that developer has contributed to</td>
</tr>
</tbody>
</table>

### Table 7: Attributes of commits.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha</td>
<td>(String) Unique string of a commit</td>
</tr>
<tr>
<td>create_at</td>
<td>(DateTime) The time the commit was created at</td>
</tr>
<tr>
<td>author</td>
<td>(Array) The list that contains the information about the original author of the commit</td>
</tr>
<tr>
<td>committer</td>
<td>(Array) The list that contains the information of the submitter of the commit</td>
</tr>
<tr>
<td>files</td>
<td>(Array) The list that contains the information about the files related to this commit</td>
</tr>
</tbody>
</table>

#### 3.1.2.2 Social Network Features

In this research, we defined the social networks of developers via “actual social network” and “inferred social network”. All social connections contain the following or followed user information and are stored in developer’s attribute “social_relation” (as shown in Table 6).

1. **Actual social network:** For developers who have actual “following-followed” relations, we collect such records of their social network. These records can be mined from GHTorrent with precise timestamps and user information and therefore can be used for extracting the social network data.
2. **Inferred social network**: However, we found that in some cases the developer has not developed any social connections with others on GitHub. In this situation, we extract the “inferred social network” by including users from the projects that the developer has participated in (i.e., contributed to) and define this as their “inferred” social network.

It is worth noting that although it is better to maximize the social connections by applying both techniques. However, we argue that for those developers who have existing social networks we assume that they make their own choices in who they want to follow, whatever their reason might be. Therefore, we do not infer social networks for these developers. Further details and examples for both ASN and ISN approaches are introduced in Section 3.3.1.

### 3.2 Profile Building

In this section, we introduce feature extraction and data preprocessing in Section 3.2.1, while profile building for both developers and projects is discussed in Section 3.2.2.

#### 3.2.1 Data Preprocessing and Text-based Metrics Extraction

TF-IDF (term frequency-inverse document frequency) \([82,83]\) evaluates how relevant (i.e., important) a word is in a document. TF-IDF value is obtained by multiplying both TF and IDF metrics; the former represents the times that a word appears in a document, while the latter indicates the inverse document frequency of the word across a set of documents. TF-IDF is widely used for extracting keywords of documents.

Vector Space Model (VSM) \([84]\): VSM is based on the notion of similarity. The model assumes that the relevance of a document to query is roughly equal to the
document-query similarity. Each dimension corresponds to a separate term. If a term occurs in the document, its value in the vector is non-zero. Several different ways of computing these values, also known as (term) weights, have been developed.

As discussed in Section 3.1.2 we collect README files of all 1.7M publicly available projects. We perform the following steps to extract TF-IDF metrics of the repositories:

1. We first convert all texts of the document into lowercase. This is a mandatory step as programming languages consider textual data as case-sensitive.

2. Stop words such as ’me’, ’we’, ’is’, etc. are removed as these words do not provide any additional value to the document vector.

3. Single characters are not valuable for knowing the importance of the document, so it is always good to remove the single characters.

4. As the last preprocessing step, stemming is applied to convert words to their stem.

5. Finally, we perform TF-IDF to generate word dictionary that contains all keywords and their values.

Similar to repositories, we retrieve all Java commits of the target developers to build their expertise vectors. However, instead of using TF-IDF, we perform VSM to do the calculation. The steps are as follows:

1. Query all commits submitted by the developer with file extensions. Here, we obtain a list of file extensions.

2. We eliminate useless extensions such as ’.md’, ’.gitignore’, and other non-programming-related extensions.
3. VSM is applied to generate word vector and their weights.

4. Finally, standardization is performed to make sure that keys with values in different ranges have the same importance.

Last but not least, as we already obtain the records of the repositories that developers have contributed to, it is essential to precalculate the repository interest to increase the performance of the frameworks. For this attribute, we simply sum up the repository README vectors of all the projects developers participated in. Then, we normalize the final value of the sum vector to make it comparable to all other vectors.

### 3.2.2 Profile Building

After data preprocessing is complete, we next build profiles for project and developers.

1. **Project Profile:** As shown in Table 8, we include all necessary attributes of the projects to generate recommendations. Comparing to Table 5, we remove some attributes such as name, description, language, etc., as those attributes do not add value for either generating recommendations nor gathering information.

   It is important to emphasize that “README_vector” in the Table 8 plays a critical role in our parallel GA model as it serves as a part of the “interest vector” of the developer’s profile. Figures 12 and 13 provide examples of the data that the README_vector consists of. TF-IDF analyzes all README documents, eliminates unimportant terms, and uses the remaining terms to shape the project. From Figure 12, we can observe that project with ID ’126282513’ may relate to a Spring Boot web application and deploy cloud computing. While for project with ID ’107093997’, shown in Figure 13, is likely to be an Android library developed in Android Studio IDE. Each example also show that the length of README vectors varies depending on the project.
Table 8: Project profile.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Unique number for a repository</td>
</tr>
<tr>
<td>README_vector</td>
<td>A dictionary that contains all keywords and their weights of the repository</td>
</tr>
<tr>
<td>users</td>
<td>A list that contains all users who at least have contributed to this repository once</td>
</tr>
<tr>
<td>topics</td>
<td>A list that contains all topics of the repository pre-defined by the project owner</td>
</tr>
<tr>
<td>watchers</td>
<td>A list that contains the watchers (who star the repository) list of the repository, including user ID and timestamp</td>
</tr>
</tbody>
</table>

Figure 12: README vector for project ID 126282513.

Figure 13: README vector for project ID 107093997.

2. Developer Profile: Table 9 reports all attributes of developer profiles. From the table, it is necessary to mention that we consider 'following' and 'followed' as two different relations in GitHub, while the former indicates the people that the developer admires or appreciates, while the latter points out that the developer is admired by others. In this research, we consider both relations.

(a) **Expertise vector** is calculated from the developer’s commit data. We extract all the file extensions, remove trivial ones and perform VSM to construct such vectors. Expertise vector is helpful when developer expertise similarity is required in our fitness function. For example, Figure 14
Table 9: Developer profile.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>ID of the user</td>
</tr>
<tr>
<td>repo_number</td>
<td>Number of repositories that the developer has contributed to</td>
</tr>
<tr>
<td>expertise_vector</td>
<td>A dictionary that contains all file extensions and their weights of the user</td>
</tr>
<tr>
<td>repo_interest_vector</td>
<td>A dictionary that contains all contributed repository keywords and their weights of the user</td>
</tr>
<tr>
<td>repos</td>
<td>A list that contains the repositories that the user has already contributed to</td>
</tr>
<tr>
<td>social_relation_following</td>
<td>A list that contains users who are followed by the user</td>
</tr>
<tr>
<td>social_relation_followed</td>
<td>A list that contains users who are following the user</td>
</tr>
</tbody>
</table>

presents a developer’s expertise vector. In this example, we can see that in the past two years this developer have mainly contributed to Java and XML files, with smaller commits to Gradle files.

(b) **Repo_interest_vector** is an accumulated README vector of the developer’s past projects. We assume that a developer has more interest in the project if they had a higher number of contributions to this project. On the other hand, if the developer has contributed to other projects only a few times, while these projects are a part of developer’s repo interest vector, the proportion is smaller. Also, different projects may have overlap areas (it could be a specific tool, IDE, library or scientific theory), our approach can potentially point out these pieces of information. An example of the repo interest vector is shown in Figure 15. The length of the repo interest vector can vary depending on developers.
3.3 Recommendation Framework

In this section, we introduce our proposed GitHub project recommendation structure. Figure 16 illustrates the overall research roadmap. Social analysis is described in Section 3.3.1, while the detailed Parallel Genetic Algorithm (PGA) is introduced in Section 3.3.2. Details on PGA’s weights distribution, interest similarity, expertise similarity and social importance are explained in Section 3.3.2.1, Section 3.3.2.2, Section 3.3.2.3 and Section 3.3.2.4 respectively.

3.3.1 Social Analysis

As mentioned in Section 3.1.2.2 and Section 3.2.2 our framework relies on the social relations between developers to generate recommendations. Social analysis aims at determining all potential recommendation candidates for developers by analyzing
Figure 16: Empirical study’s setup and overview.

their social relations. To do this, we first take into account the type of social network that the developer belongs to by examining whether they have any actual social connections or not. If social network exists in our database, we then consider this user to belong to Actual Social Network (ASN), otherwise Inferred Social Network (ISN).

For ASN developer, we search for their following people and organize a repository list that they contributed to as the recommendation candidates. For ISN developer, on the other hand, recommendation candidates are found by searching for the users from the projects that the developer has contributed to. The recommendation candidates are stored on our database and are further used in the PGA optimization stage.

The concepts of ASN and ISN is further illustrated in Figure 17. Figure 17(a) presents an example of ASN. Target developer U1 is following developers U2 and U3 on GitHub. Meanwhile, all of them have contributed to some projects (e.g., U1 has participated on projects P1, P2 and P3). Since the “following” relations with developer U2 and U3 are real (i.e., direct), we define such connections as “actual” social network (ASN) in our work. Therefore, the ASN approach defines a social network list of developer U1, that is \{U2, U3\}. The approach then combines all the
projects that developers in the social network participated in. In this example, the recommendation candidates from the developer’s ASN are \{P_4, P_5, P_6, P_7, P_8\}:

\{(U_2 \cup U_3) - U_1\}

Figure 17(b) illustrates an example of ISN. Target developer U_4 does not have any direct following relations on GitHub. However, our data indicate that U_4 has participated on projects P_6 and P_9. Also, P_6 has other contributors such as U_2 and U_3, and P_9 has contributors U_5 and U_6. In this case, the ISN approach automatically generates “inferred” social connections to U_2, U_3, U_5 and U_6 for the developer U_4. Then we search for all the recommendation candidates for this ISN. In this example, the recommendation candidates are projects within the ISN including \{P_2, P_4, P_5, P_7, P_8, P_{10}, P_{11}, P_{12}\}:

\{(U_2 \cup U_3 \cup U_5 \cup U_6) - U_{14}\}

Finally, Table 10 presents the statistics for both ASN and ISN in our dataset. For both types of developers, the minimum of recommendation candidates is zero,
Table 10: ASN and ISN statistics.

<table>
<thead>
<tr>
<th>Social Connection Type</th>
<th>Number of Developers</th>
<th>Candidate Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
</tr>
<tr>
<td>ASN</td>
<td>5,492</td>
<td>0</td>
</tr>
<tr>
<td>ISN</td>
<td>4,205</td>
<td>0</td>
</tr>
</tbody>
</table>

while the maximum values are quite different. This shows that ISN has the ability to extend the potential recommendation candidates if cases when developer has no ASN connections.

### 3.3.2 Parallel Genetic Algorithm Optimization

Once recommendation candidates are found, PGA is performed for finding the best recommendation combination. Figure 18 shows the flow of the PGA optimization process. This research leverages the Global Parallelization pattern when one master node controls the progress of the PGA, while several slave nodes perform the tasks of genetic algorithm.

Before diving into the details of our PGA framework, we introduce the basic configurations of our GA model itself. To the best of our knowledge, our model is the first PGA framework that is used for GitHub project recommendation problem; therefore, we investigate previous works inside and outside of software engineering domain \[21\] \[24\] \[78\] \[81\] \[85\] \[92\] and use the basic GA setups and configurations in this work. We mention several ideas on improving our GA configuration (e.g., tune up of the parameters) as future extensions in Section 6.2.

1. **Chromosome:** While previous research \[23\] combine all candidates into a chromosome for recommending code reviewers, it is unlikely doable for our research as developers may have thousands of candidates, which may cause severe computing issues. Therefore, we narrow down the length of chromosomes...
Figure 18: Parallel Genetic Algorithm’s optimization structure.
to 10 to reduce computational barriers. In addition, our chromosome structure follows multiple representation formats, which are real-valued representation and binary representation. Each chromosome randomly picks 10 projects from the repository candidate pool to organize itself. By doing this, we can guarantee the diversity of chromosomes is well-maintained. Meanwhile, as shown in Figure 19, genes (i.e., fragments of the chromosome) keep the information of:

(a) **Repo_id**: A unique repository ID that indicates a repository candidate.

(b) **From_id**: A unique developer ID that can indicate where the repository is originated from.

(c) **Flag**: A binary encoding that indicates whether the repository is accepted or rejected.

(d) **Individual fitness score**: The fitness value of the repository candidate; only “accepted” projects have this value.

![Chromosome Structure](image)

**Figure 19**: Structure of a chromosome.

2. **Population Size**: As shown in Figure 18, slave nodes are responsible for generating the initial population, and each slave node populates 1,000 chromosomes at the beginning. Since the length of chromosome is set to 10, we argue that initial population of 1,000 chromosomes, in theory, should be able to fully
cover the highest number of candidates that our target developer has (i.e., 9,925 projects for ISN as shown in Table 10).

3. **Generations**: Choosing an appropriate generation limit is critical when constructing GA framework. Though our proposed framework sets 100 as the generation limit, we also compared the performance of different generation thresholds to investigate the performance and the average fitness scores, which we discuss in Chapter 4.

4. **Number of recommendations**: In this research, we design our recommender system to generate five projects as final recommendations. However, we can not achieve this by using only the fitness function. Therefore, besides the fitness function, we also introduce a penalty formula to allow the system to generate exactly five recommendations.

5. **Fitness Function**: Our proposed PGA follows the following fitness function: given the target user has interest vector $u_i$, expertise vector $u_e$ and chromosome candidates $CAs$, fitness function can be defined as follows:

$$RepScore_{(ui,ue,CAs)} = \sum_{c \in CAs}[\alpha(SIM_{u_i,i' \in c}) + \beta(SIM_{u_e,e' \in c})] \times k \quad (7)$$

where $\alpha$ represents the weight of project interest, while $\beta$ represents the weight of expertise, which is introduced in next section; $i'$ and $e'$ indicates the README vector of project $c$ and the expertise vector of social connector of project $c$; $k$ represent the social importance of social connector of project $c$; and:

$$\alpha + \beta = 1 \quad (8)$$
the number of the recommended projects is \textit{RepoCount}.

Last but not least, fitness score can be defined as follow:

\[
    FitnessScore = \frac{RepoScore}{RepoCount} \quad (9)
\]

the solution with the highest score is considered as the best solution.

\textbf{Penalty Formula:} Due to the nature of GA, it is difficult to enforce the generation of exact five recommendations based on the fitness function. Therefore, we introduce the penalty formula that can assist in achieving the goal. Given the number of recommendations is greater than five, the penalty formula is to be applied:

\[
    PF(s, n) = \frac{s}{\sqrt{n}} \quad (10)
\]

Figure 20 illustrates the results of the simulation we ran for the penalty formula. As shown, given the original fitness score is 100, when the number of recommendations is greater than five, the score is penalized with loss of 60% or more of the original score.

Finally, Figure 21 presents an example of how our fitness function and penalty formula work. Here, three chromosomes are now being evaluated by our fitness function. The figure shows that chromosomes A and B only recommend five projects as final recommendations, while chromosome C has six recommended projects. After fitness evaluation, we obtain the fitness scores of three chromosomes as 0.706, 0.618, and 0.776, respectively. Since chromosomes A and B only recommend five projects, the penalty formula is not applied. Chromosome C, on the other hand, the fitness score is penalized by the penalty formula as it recommends more than five projects. Finally, the model receives the final
scores of three chromosomes, and chromosome A outperforms chromosomes B and C.

6. **Selection:** To introduce a general and basic GA model, we apply the Roulette Wheel approach as the selection approach in this work. The individuals with higher fitness scores have more copies on the wheel, while low-score individuals have fewer copies. By doing this, higher-score chromosomes have higher chances to triumph, and lower-score ones are likely to be ignored and loosing the chance to reproduce.

7. **Crossover and Mutation Rates:** In this research, we follow the classical configurations \([85 \, 89]\) of both crossover rate and mutation rate which are 80% and 1%, respectively. In addition, we follow the one-point crossover and mutation approach. While there are advanced and complex approaches, we use the basic configuration for our approach since the goal is to propose the first
Figure 21: Workflow of fitness function and penalty formula.

GA-based GitHub project recommendation framework.

Figure 22 presents an example of how one-point (or so-called single-point) crossover works. There are two parental chromosomes with the same length at the beginning. The GA model randomly decides the point to perform crossover (i.e., crossover point). After the crossover point is decided, both chromosomes are separated into two pieces from the crossover point. The tails (i.e., green and purple colour fragments) of its two parents are swapped to get new off-springs.

Figure 22: Example of one-point crossover.

Figure 23 presents an example of how one-point mutation works in our work. From the example, the GA model randomly decides the point to perform mutation (i.e., mutation point). The gene which is selected to mutate changes its
Table 11: Weight distribution.

<table>
<thead>
<tr>
<th>Node</th>
<th>$\alpha$</th>
<th>$\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Node 2</td>
<td>0.1</td>
<td>0.9</td>
</tr>
<tr>
<td>Node 3</td>
<td>0.2</td>
<td>0.8</td>
</tr>
<tr>
<td>Node 4</td>
<td>0.3</td>
<td>0.7</td>
</tr>
<tr>
<td>Node 5</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>Node 6</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Node 7</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>Node 8</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>Node 9</td>
<td>0.8</td>
<td>0.2</td>
</tr>
<tr>
<td>Node 10</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td>Node 11</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

binary code. In the example, the original binary code for project P7 is “0” (rejected), the mutation changes its value to “1” (accepted). On the other hand, if the original value is “1” (accepted), it is modified to “0” (rejected) as well.

Figure 23: Example of one-point mutation.

3.3.2.1 Weights Distribution

Main node dispatches 11 nodes with specified weight of $\alpha$ and $\beta$ (shown in Table 11). Each node then runs 100 generations and returns its best solution.
3.3.2.2 Interest Similarity

When computing the similarity of interest, we retrieve the target developer’s repository interest vector and the README vector of the current repository and perform cosine similarity \[93,94\] to obtain the value.

3.3.2.3 Expertise Similarity

This value comes from calculating the cosine similarity between the expertise vectors of the current project’s developer and the target developer (i.e., \fbox{\textit{from}\_id} in Figure 19).

3.3.2.4 Social Importance

In this research, social importance is an indicator that quantifies the weight of developers, which can be considered as the number of followers. In the fitness function, the notation \(k\) represent the social importance of social connector of project \(c\).

3.4 Recommendation System Evaluation

In this section, we report on how we evaluate our proposed framework. Comparison with the baseline (i.e., RepoPal) is discussed in Section 3.4.1 while Section 3.4.2 discusses qualitative evaluation.

3.4.1 Baseline: RepoPal

To decide on the baseline for comparing our approach with, we choose the most relevant recommender system that exits to date, RepoPal \[26\]. RepoPal is an activity-based recommender. We followed the approach described in the original paper and
implemented a RepoPal framework. For our evaluation, we generate five recommendations for each model (RepoPal and our proposed recommender) and evaluate these recommendations with actual GitHub developers via a qualitative study.

3.4.2 Qualitative Evaluation

Evaluation of the recommender systems is a challenging task [95–101], since we do not have the ground truth labels available to assess the recommendation’s accuracy. Therefore, we design and develop a survey for the target developers in order to collect their opinions and feedback on the two recommendation systems. Based on the participants’ ratings, we can evaluate the accuracy of the recommendations. Additionally, developers’ feedback can help us better understand strengths and limitations of the developed framework.

3.4.2.1 Survey Design

To recruit participants for our study in evaluating recommendation systems, we first send out a recruitment email describing our research study and asking developer’s consent and willingness to participate in it. We then send out the survey to only those developers who have agreed to participate in our study.

In the survey, we use a Likert scale [102] for evaluating different aspects of two recommendation models. Likert scale is applied as one of the most fundamental and frequently used psychometric tools in education and science research. This approach scales the responses to questions by using five-level or three-level Likert items.

When designing the survey, we stick to the following guidelines [103, 104]:

1. **Renaming models:** From the two models, we know which model is based on social network or activity-based. However, the survey participants should be not able to guess or perceive the details of models in order to produce the most
objective feedback. Therefore, many survey designers rename the models to obscure participants. The most commonly used are Model A, B, C,... or Model 1, 2, 3, ... These approaches efficiently keep participants in the dark about knowing any details of models. But it raised other issues. For some users, when they see there are two models A and B (or 1 and 2), they tend to guess one of them is better than the other one without carefully evaluating the contents of the models. That is, sequence-based renaming can potentially introduce bias in participants’ responses.

Therefore, instead of following the sequence, we rename our models by giving them meanings that only we can comprehend. For example, the baseline model RepoPal is renamed as Model B, while Model S is our proposed framework since it is a social-based model.

2. **Switching orders:** Though we eliminate the renaming bias by adding a non-sequence internal meaning to our models, it is possible that participants are affected by other biases. For example, survey participants may consider the model that appears first (or last) to outperform the other model. Unfortunately, this bias can not be determined. But we can attempt to reduce it. To do so, we divide our surveys into two groups, the first group starts with the evaluation of the Model B, while the second half starts with Model S.

3. **Reverse coded items:** Reverse coding is a common validation technique for survey items to rephrase a “positive” item in a “negative” way. When done properly, this can be used to check if respondents are giving consistent answers. In our survey, we create one reverse-coded item to test whether all responses are valid and consistent.
Table 12: Survey content: Model B.

<table>
<thead>
<tr>
<th>No.</th>
<th>Question</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B-1)</td>
<td>I find the recommendations helpful.</td>
<td>Multiple choice</td>
<td>Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(B-2)</td>
<td>I will try out at least one of the recommendations.</td>
<td>Multiple choice</td>
<td>Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(B-3)</td>
<td>I perceive the recommendations reflect my project-selection preferences.</td>
<td>Multiple choice</td>
<td>Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(B-4)</td>
<td>I perceive the recommendations reflect my programming expertise (i.e., programming skills).</td>
<td>Multiple choice</td>
<td>Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(B-5)</td>
<td>I will use this model for future project recommendations.</td>
<td>Multiple choice</td>
<td>Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(B-6)</td>
<td>Do you have any additional feedback or comments about Model B?</td>
<td>Long-answer Text</td>
<td>(Optional) Participants may leave any feedback in this field.</td>
</tr>
</tbody>
</table>

3.4.2.2 Survey Content

We separate the survey into three sections (the sample of survey questionnaire is included as Appendix C):

1. **Model B:** As discussed in the previous section, Model B contains recommendations generated by RepoPal and lists six questions, which are shown in Table 12:

2. **Model S:** On the other hand, Model S offers recommendations generated by our social-based model and lists eleven questions shown in Table 13. It is worthwhile noticing that, unlike RepoPal, recommendations from our model are not rank-based. Our model provides best recommendation combination for the developer as this combination outperforms others throughout the evolution process. Therefore, as a part of the survey, we ask participants to rank all recommended projects one-by-one by giving them scores (i.e., numbered questions S-6.1 to S-6.5 in Table 13).

3. **Overview:** Last but not least, in the final part of the survey, we are interested to learn what model the participant prefers. Additionally, we are interested in learning what aspects the participant find useful in recommender systems in...
Table 13: Survey content: Model S.

<table>
<thead>
<tr>
<th>No.</th>
<th>Question</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S-1)</td>
<td>I find the recommendations helpful.</td>
<td>Multiple choice</td>
<td>Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(S-2)</td>
<td>I will try out at least one of the recommendations.</td>
<td>Multiple choice</td>
<td>Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(S-3)</td>
<td>I perceive the recommendations DO NOT reflect my project-selection preferences.</td>
<td>Multiple choice</td>
<td>(Code reversed question) Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(S-4)</td>
<td>I perceive the recommendations reflect my programming expertise (i.e., programming skills).</td>
<td>Multiple choice</td>
<td>Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(S-5)</td>
<td>I will use this model for future project recommendations.</td>
<td>Multiple choice</td>
<td>Agree, Neutral or Disagree</td>
</tr>
<tr>
<td>(S-6.1)</td>
<td>How would you rank project P1</td>
<td>Linear scale</td>
<td>1 (Highly Dislike) to 10 (Highly Like)</td>
</tr>
<tr>
<td>(S-6.2)</td>
<td>How would you rank project P2</td>
<td>Linear scale</td>
<td>1 (Highly Dislike) to 10 (Highly Like)</td>
</tr>
<tr>
<td>(S-6.3)</td>
<td>How would you rank project P3</td>
<td>Linear scale</td>
<td>1 (Highly Dislike) to 10 (Highly Like)</td>
</tr>
<tr>
<td>(S-6.4)</td>
<td>How would you rank project P4</td>
<td>Linear scale</td>
<td>1 (Highly Dislike) to 10 (Highly Like)</td>
</tr>
<tr>
<td>(S-6.5)</td>
<td>How would you rank project P5</td>
<td>Linear scale</td>
<td>1 (Highly Dislike) to 10 (Highly Like)</td>
</tr>
<tr>
<td>(S-7)</td>
<td>Do you have any additional feedback or comments about Model S?</td>
<td>Long-answer Text</td>
<td>(Optional) Participants may leave any feedback in this field</td>
</tr>
</tbody>
</table>

Table 14: Survey content: overview.

<table>
<thead>
<tr>
<th>No.</th>
<th>Question</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O-1)</td>
<td>Which recommendations you found useful?</td>
<td>Multiple choice</td>
<td>Model B, Model S, Both Models or None</td>
</tr>
<tr>
<td>(O-2)</td>
<td>Which aspects of recommendation you find useful?</td>
<td>Checkboxes</td>
<td>Agree, Activities, Social Connections, Expertise, or Interests</td>
</tr>
</tbody>
</table>

3.4.2.3 Participant Recruitment

The qualitative study conducted as a part of this research has obtained approval from the Carleton University Office of Research Ethics, the clearance certificate is included as Appendix A. For our 9,697 target developers, we were able to find and obtain 3,327 email addresses. We have sent the first survey to random 1,000 emails of the developers in our dataset to obtain their consent. The consent questionnaire
is included as Appendix B. We receive 29 responses, out of which 27 have agreed to participate in the research survey, and 2 developers have declined the invitation. Furthermore, ten email addresses have bounced back. Therefore, the response rate is 5.5%. Among 27 developers who agreed to participate in the study, 18 have submitted their survey responses survey, while we two responses were deemed as invalid due to the inconsistent answers of reverse coded item (i.e., question S-3 in Table[13]). Finally, 16 out of 27 respondents have completed and submitted the survey, i.e., a submission rate of 59.2%. Detailed response analysis and findings are discussed in the next chapter.

3.5 Tools and Libraries Used

The following commercial or OSS software, libraries and services are used throughout this research for data collection, data storage, development, or data analysis.

Pycharm Professional [105]: We use Pycharm environment 2020.2.2 with Python 3.7 64-bit to conduct all experiments of this thesis.

Gensim [106]: Gensim is a popular framework for topic modeling. In this research, we utilize it to build TF-IDF vectors.

GitHub API [107]: We use GitHub API to retrieve commit data, README files and email addresses of developers.

Google BigQuery [108]: We use Google BigQuery to query and manage the GHTorrent SQL dataset.

MongoDB [109]: As a local data management tool, MongoDB provide a flexible structure for organizing data. It also offers practical Python library called Pymongo [110] for operating queries between Python and MongoDB server.

CSV: Comma-separated values file stores values in different formats such as numbers, strings, etc. in a structured tabular form. Python’s Pandas [111] package works
very efficiently with CSV files.

**WordCloud** [112]: WordCloud is a technique to show which words are the most frequent among the given text. It provides Python library for visualize our data.

**Qualtrics** [113]: Qualtrics offers a cloud-based subscription software platform for experience management. In this research, we use Qualtrics for conducting surveys and data analysis.

**Plot Likert** [114]: Plot Likert is a Python library which aims at visualizing Likert scale results. In this research, we use this library to present the visualized survey outcomes in Chapter 4.
Chapter 4

Results

In this chapter, we present the results for the research questions described in Section 1.3. In each section, the results provide the answer for each research question of this work.

4.1 RQ1: How accurately can our framework recommend projects to GitHub developers?

To answer our first research question, we need to investigate the item rankings from developers. Table 15 lists item rankings of our proposed model’s recommendations from all 16 valid survey responses we received, while Figure 25 summarizes the visualized Likert scale results of Table 15. From Table 15, we can observe that some rankings of recommendations are marked as N/A, which means that when we distributed the survey, these repositories were no longer available (GitHub projects may be deleted by their owner anytime) thus making our participants unable to evaluate them. Therefore, we removed such no-longer-existing projects, and as compensation, we also remove the same number of projects from Model B and vice versa. Based on the results in Table 15, we present our findings and summary for RQ1.
Table 15: Survey responses: item rankings for Model S recommendation.

<table>
<thead>
<tr>
<th>Respondent</th>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
<th>Item 4</th>
<th>Item 5</th>
<th>Average</th>
<th>Preferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>6.4</td>
<td>S</td>
</tr>
<tr>
<td>P2</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>6.6</td>
<td>B</td>
</tr>
<tr>
<td>P3</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td>2</td>
<td>4.6</td>
<td>N</td>
</tr>
<tr>
<td>P4</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>N/A</td>
<td>6.5</td>
<td>S</td>
</tr>
<tr>
<td>P5</td>
<td>5</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>4.8</td>
<td>N</td>
</tr>
<tr>
<td>P6</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>6.6</td>
<td>N</td>
</tr>
<tr>
<td>P7</td>
<td>7</td>
<td>9</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
<td>6.3</td>
<td>S</td>
</tr>
<tr>
<td>P8</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>S</td>
</tr>
<tr>
<td>P9</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>S</td>
</tr>
<tr>
<td>P10</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>N</td>
</tr>
<tr>
<td>P11</td>
<td>10</td>
<td>9</td>
<td>3</td>
<td>N/A</td>
<td>N/A</td>
<td>7.3</td>
<td>S</td>
</tr>
<tr>
<td>P12</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>S</td>
</tr>
<tr>
<td>P13</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>N/A</td>
<td>N/A</td>
<td>6</td>
<td>N</td>
</tr>
<tr>
<td>P14</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>N/A</td>
<td>6</td>
<td>6.5</td>
<td>S</td>
</tr>
<tr>
<td>P15</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>S</td>
</tr>
<tr>
<td>P16</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3.8</td>
<td>B</td>
</tr>
</tbody>
</table>

4.1.1 Overall Developers’ Preferences

As we can see, most of the participants find Model S as their preferred model for recommending GitHub projects (i.e., the column “O-2” in Table 15 codes “B”, “S” and “N” represent Model B, Model S or None, respectively). More precisely, 9 out of 16 responses (56%) point out that they find our model to outperform the baseline; on the other hand, only two participants prefer RepoPal as it matches their needs for GitHub project recommendation better. Last but not least, five developers find both models to be limited. We take a deeper look at these three groups of responses.

4.1.2 Strengths and Limitations of Model B

As model B (i.e., RepoPal) is a recommendation system that evaluates the interests of developers and textual similarity between projects, it highly relies on the latest
information about GitHub. Any outdated data would have a negative effect on project recommendations. To address this issue, as discussed in Section 3.1, we consider only two years of the developers’ data such as their activity. That is, from July 2017 to June 2019.

Furthermore, Figure 24 summarizes the key aspects that our participants consider helpful when it comes to GitHub project recommender systems. We can observe that 14 out of 16 (87.5%) developers believe that their “Expertise” (i.e., skillsets) makes a decisive impact on the quality of recommendations. Beyond that, over half of the participants consider “Interests” (68.8%) and ”Activity” (56.3%) to play essential roles in such recommender systems, which are precisely the aspects that RepoPal leverages. However, it is not clear why only 12.5% of developers prefer RepoPal.

Our first finding suggests that while developers consider their expertise, interests and activity to lead to accurate recommendations, the benefits maybe diminished by outdated data. In this research, while collecting most recent data, we are relying on the GHTorrent as our data source. Leveraging GitHub API for data collection would be our preferred choice, however, due to the limitations discussed in Section 3.1, mining GitHub via its API is a very expensive task.

![Figure 24: Recommendation aspects that developers find most useful.](image)
4.1.3 Strengths and Limitations of Model S

According to Figure 24, only 43.8% of developers believe that social connections have a positive impact on recommendation. However, as shown in Table 15, 56.3% of survey participants prefer our model as a better recommendation system compared to RepoPal (12.5%).

Respondent P1 mentions the following about RepoPal: “...all of the projects were in a different language, making it hard for me to use...” Though we only select Java projects in this research, some projects may contain files written in other programming languages. Since RepoPal does not consider actual files and their extensions within the project, non-Java projects are likely to be recommended to developers. On the other hand, for our model, P1 writes: “...Much better, the java test utils that were recommended seems like a great project, and I would use it.” This feedback echoes our assumption. Since our model considers not only the primary language of projects, it is more likely to recommend Java repositories.

To assess the benefits of social networks in our framework, we study the feedback from respondent P9 as they mention: “This is a more useful model. At the time the survey is considering, all of the recommended repositories were implementations of the open-source software our team was working on. I would likely look through the code of some of the recommendations.” This hints us that developer P9 could correctly “detect” that the recommendations of Model S are based on his social connections. P9 also mentions: “our team was in contact with 4 of the five people running the recommended projects.” When we traced back his records in our database, we found that our model generated recommendations by applying Inferred Social Network (ISN) technique. That is, the recommendation list is based on the repositories that P9’s co-contributors have been engaged in. This demonstrates that ISN works well in the cases of insufficient actual social networks and validates that
our PGA framework can recommend appropriate projects among many repository candidates.

**Limitations of Model S.** While we have received critical praise for Model S from the developers, a number of further improvements are required. In fact, according to Figure [25] while it seems that the majority of responses were satisfied with our recommendations, the ratings with the score lower than 6 attract our attention as we receive feedback highlighting common issues.

Respondents P10 and P12 mention: “...*The other recommendation has been dead for years.*” While P12 and P15 say: “this model contains archived project though it is interesting.” These two comments raised an issue about our project selection approach. As described in Chapter [3] when selecting repositories we eliminated all ‘deleted’ and ‘fake’ projects. Furthermore, we also check whether projects are available publicly by performing the web-crawler technique. However, for those ‘archived’ repositories, neither web-crawler nor GitHub API was able to detect such projects. Fortunately, the latest GitHub API does provide this attribute in the query results at the moment; so we could extend our data preprocessing to make sure we eliminate ‘archived’ projects.

While we tried our best to eliminate unavailable projects before generating recommendations, it is still possible for the projects to be deleted or set to “private” anytime in GitHub. In fact, P6 is further reflecting on this point: “*The dataset is outdated, i.e. it contains a 404.*” These 404 errors mean that projects are not available to access (unreachable), and therefore, our recommendations may not be useful to the developers.

### 4.1.4 Limitations of Both Models

One of the surprising results was the fact that 31.3% of developers found both models being of little to no use. To identify potential issues, we carefully investigate the
feedback of these developers. We determine two main issues that might explain their opinion.

Participant P5 mentions that: “4 out of the five recommendations corresponds to forks of some of my organization repositories belonging to colleagues of mine.” In addition, P10 also points out: “Repositories proposed have very little sense since they are basically just forks of the same project...” These two developers essentially talk about the same problem - forking. Although forked repositories can be used as recommendations, not all developers find them meaningful or useful. From the above two comments, we speculate that these developers are seeking creative and original projects rather than those forked from the existing ones.

For the second issue is related to developer habits and behaviour. As P6 mentions: “…model B seems to work with either the artifact dependencies or the personal dependencies (users committed to the same code base). If I need to look through others code, a keyword / search term approach would be more helpful to me.” Based on this feedback, we infer that this developer is likely in favour of Model B; however, due to their own habits, the developer believes Model B would not be needed most of the time as they prefer keyword searching instead. One take-away from this finding for researchers would be to develop recommender systems that can be fine-tuned by the users based on their preferences, habits, etc.
Answer to RQ1: The survey results demonstrate that most of the participants (56.3%) are satisfied with our model’s recommendations compared to the baseline’s (RepoPal). Key limitation of both models is an outdated data. Our recommendation system can be further improved by eliminating forked and archived projects. While only 44% of developers value recommendations based on their social network, we believe that collaboration and social connections are critical in GitHub, and thus these aspects can be leveraged for recommending GitHub projects.

4.2 RQ2: How effective is our recommendation approach compared to the baseline?

To answer this research question, we further investigate the survey responses. Table 16 presents the survey answers of all 16 participants about their opinions of Model B and Model S. To get more insights from the responses, we go through all survey questions one by one and discuss different aspects of both models.
Table 16: Survey responses; code reverse items have been processed.

<table>
<thead>
<tr>
<th>Respondent</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
<th>B-5</th>
<th>Preferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>S</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N</td>
</tr>
<tr>
<td>P4</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>S</td>
</tr>
<tr>
<td>P5</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>N</td>
</tr>
<tr>
<td>P6</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>N</td>
</tr>
<tr>
<td>P7</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>S</td>
</tr>
<tr>
<td>P8</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>S</td>
</tr>
<tr>
<td>P9</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>S</td>
</tr>
<tr>
<td>P10</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>N</td>
</tr>
<tr>
<td>P11</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>S</td>
</tr>
<tr>
<td>P12</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>S</td>
</tr>
<tr>
<td>P13</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>N</td>
</tr>
<tr>
<td>P14</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>S</td>
</tr>
<tr>
<td>P15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>S</td>
</tr>
<tr>
<td>P16</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Respondent</th>
<th>S-1</th>
<th>S-2</th>
<th>S-3*</th>
<th>S-4</th>
<th>S-5</th>
<th>Preferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>N</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>P5</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>N</td>
</tr>
<tr>
<td>P6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>P7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>S</td>
</tr>
<tr>
<td>P8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>S</td>
</tr>
<tr>
<td>P9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>P10</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>N</td>
</tr>
<tr>
<td>P11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>S</td>
</tr>
<tr>
<td>P12</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>S</td>
</tr>
<tr>
<td>P13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N</td>
</tr>
<tr>
<td>P14</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>S</td>
</tr>
<tr>
<td>P15</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>S</td>
</tr>
<tr>
<td>P16</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B</td>
</tr>
</tbody>
</table>
4.2.1 Usefulness of Models

Figure 26 presents the results related to the usefulness of Model B. We can see that 56.2% of developers are dissatisfied with the recommendations offered by Model B, and only 6.2% of participants (i.e., one developer) are satisfied with them. According to column ‘Preferred’ in Table 16 for those developer who prefer Model S (or so called ‘Model S developers’), it shows a similar trend as 67% of participants do not find the recommendations of RepoPal useful, while 33% of respondents remain neutral. Even for Model B developers (i.e., those who prefer Model B), only one respondent remains neutral while the rest do not find the recommendations useful (‘disagree’). As explained in Section 4.1, we refer this limitation to outdated data. Due to the time gap between the GHTorrent dataset and the “live” data in GitHub, developers’ interests might have shifted, while RepoPal is unable to update the data. However, as our model provides more possible candidates using social connections, developers’ current interests may be correctly detected and recommended.

As shown in Figure 27 while 18.8% of participants do not find our model useful, 50% of developers believe that our model helps guide them in finding their future projects to contribute to. For developers who prefer Model S (as shown in Table 16), 72.3% of developers are satisfied with our GitHub project recommendations, while the rest stand neutral.

4.2.2 Preliminary Effectiveness of Models

Beyond the usefulness of models, we further study developer perceptions and explore the preliminary effectiveness of both models’ recommendations. In particular, we evaluate developer willingness to try out one or more offered recommendations.

According to Figure 28, it seems that nearly one-third of participants would like to try at least one recommendation of RepoPal, while 56.2% of participants express
that they would not try out any recommendations for RepoPal. When it comes to Model S, as shown in Figure 29, 63% of participants are willing to try out at least one recommendation, and only 18.8% of developers are not planning on using Model S’ recommendations.

Though the outdated data may affect the developer perception of RepoPal, the sparsity of starred projects in GitHub may also negatively affect the recommendations. While we were exploring the dataset, we found that only 32.4% of target developers have at least one starred project, not to mention that fewer developers have more than one starred projects. Though RepoPal can calculate the similarity of developers via starred repositories, the lack of starred projects can reduce the accuracy of final recommendations. On the other hand, our model utilizes social connections to expand possible recommendations; with the aid of parallel computing and weights balancing, our models can find more meaningful recommendations.

### 4.2.3 Supporting Developers’ Project Preferences

We now assess models’ capacity in supporting developers’ project selection preferences, that is how well each model provides recommendations that are reflective of developers’ interests.
According to Figure 30, we see that 50% of developers find that RepoPal is unable to address their project selection preferences.

As shown in Figure 24 in Section 4.1, only 43.8% of developers perceive social connections as helpful for recommending projects. This finding suggests that most of the developers perceive that our model does not support their preferences for project selection. Indeed, Figure 31 shows that only 37.5% (7 developers) of all participants agree that our model considers their project selection preferences. More precisely, two developers disagree with this, and 7 remain neutral. While our model does consider developers’ past activity data, we use it for one of the two parameters for optimizing recommendations. The very beginning step (i.e., candidate finding) was conducted by analyzing their social connections. However, if we consider only those developers who are in favour of Model S (i.e., Model S developers), we can observe that over one half of them (55.6%) finds our model to be supportive of their preferences for project selection, while one-third (33%) remain neutral. This trend shows that even though, only about one third of developers believe our model considers their project preferences, while those who favour Model S agree that it does reflect their preferences.
4.2.4 Supporting Developers’ Programming Expertise

Questions B-4 and S-4 in Table 16 aim to determine the models’ ability to support developers’ programming skills and expertise.

RepoPal is essentially an interest-based recommendation system, and programming expertise is not considered. Hence, we can expect that participants’ responses are likely to consist of disagreement or neutral perception. Unsurprisingly, in Figure 32, we see that ‘disagree’ and ‘neutral’ take 75% of responses, only one-quarter of developers believe that RepoPal recommendations reflect their programming expertise.

On the other hand, as we can see from Figure 33, while ‘neutral’ option takes the most significant proportion of the responses, we see no participants that find our model unable to reflect their programming skillsets. Since we build the expertise vectors based on commit activity and use them in the optimization stage, the final recommendations are adjusted (or affected) by these values. Therefore, for developer who prefer Model S over Model B, about 67% of them believe that our model does consider their expertise, while the rest of them (33%) take a neutral stand.
4.2.5 Perceived Trustfulness of Models

Last but not least, questions B-5 and S-5 of Table 16 evaluate developers’ confidence in trusting and using each model. Based on the responses mentioned in Table 16, we can observe that most developers prefer our model. However, “using it in the future” implies some degree of trust. That is, unlike trying out one or more recommendations as a one-time thing, developers will use recommendation system in the future only if they believe the recommendations are useful and helpful in reducing the barrier of finding relevant projects to join. When answering this question, participants have already evaluated all aspects of the two models (i.e., usefulness, effectiveness, project selection preferences, and programming expertise). Thus, developers are now able to make their final decision on whether they will use these two models in the future.

Figure 34 illustrates that only 12.5% of participants stand neutral or plan on using RepoPal as their GitHub project recommendation system in the future. The rest of them (75%) state that they will not consider RepoPal for future use. Again, as discussed in Section 4.1, the key limitation we believe is the outdated data.

Figure 35 demonstrates that 18.8% of participants claim that they will not consider our model in the future, while 56.3% of developers express their willingness to use
Answer to RQ2: To evaluate our model against the baseline (RepoPal), we design and conduct a survey with actual GitHub developers and seek their feedback on the recommendations generated by both models. Based on their feedback, we found that 50% of developers find our model helpful compared to 6.3% for RepoPal. Furthermore, 63% of developers are willing to try out at least one of our model’s recommendations. Moreover, majority of the developers agree that our model reflects their programming expertise and plan on using our recommender system in the future. However, only 3 out of 16 developers believe that our model support their project preferences. Overall, based on the qualitative study, our recommender system demonstrates promising potential.

4.3 RQ3: What is the run-time overhead of the GA-based recommendation?

RQ1 and RQ2 answer the questions about the accuracy and the effectiveness of our model. While these two indicators are critical for recommender systems, it is also essential to evaluate the system performance. Hence, in this research question, we aim at assessing the performance of our proposed model compared to RepoPal.

1. Pre-calculations: In this research, since we mainly use offline data (i.e., fixed data that remains unmodified throughout the study), it is possible and time-saving to pre-calculate the parameters required for the PGA computing and RepoPal. As introduced in Chapter 3.2, we calculate in advance and build developer and repository profiles that are used later in PGA computing. For RepoPal, we pre-calculated the similarity of developers and repositories.

2. Computational setups: We conduct five different setups, i.e., four for our
model and one for RepoPal. The specifications of the five setups are shown in Table 17.

- **Standard**: This research mainly uses this setup for generating recommendations and conducting surveys. This setup follows Global Parallelization (GP) pattern which consists of one master node and 11 slave nodes. Master node is responsible for distributing weights (i.e., $\alpha$ and $\beta$) to slave nodes, while slave nodes perform GA procedures.

- **Single Standard**: Unlike the 'standard' setup which distributes 1,100 iterations into 11 slave nodes, 'single standard' performs Simple Genetic Algorithm (SGA) where one node (i.e., thread) performs all 1,100 generations and outputs the final solution. However, as our fitness function requires $\alpha$ and $\beta$, these two weights are randomly decided in each generation.

- **Heavy**: Theoretically, the fitness score (i.e., accuracy) increases with the number of generations. However, due to GA’s convergence issue, the cost with respect to time may not be able to exchange equal fitness score improvement. The 'heavy' setup aims to testify whether it can provide more accurate results without paying extra time under the same conditions. Similar to 'standard' setup that follows GP structure but increases generations to 500 for each slave node.

- **Single Heavy**: The 'single heavy' setup performs SGA with 5,500 generations and weights $\alpha$ and $\beta$ are randomly decided in each generation.

- **RepoPal**: We followed the original paper of RepoPal [26] and implemented this model. As shown in Table 17, due to the nature of RepoPal, it does not have 'generations', weights of '$\alpha$ and $\beta'$ and 'fitness score'. Therefore, we mark those three attributes as 'N/A'. Furthermore, similar
to the original paper, we do not implement parallel computing for RepoPal. As a compensation, 'Single Standard' and 'Single Heavy' setups can demonstrate comparative results.

Table 17: Computational setups.

<table>
<thead>
<tr>
<th></th>
<th>Single Standard</th>
<th>Standard</th>
<th>Heavy</th>
<th>Single Heavy</th>
<th>RepoPal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generations</td>
<td>1100 * 1 node</td>
<td>100 * 11 nodes</td>
<td>500 * 11 nodes</td>
<td>5500 * 1 node</td>
<td>N/A</td>
</tr>
<tr>
<td>(\alpha) and (\beta)</td>
<td>Random</td>
<td>Specified</td>
<td>Specified</td>
<td>Random</td>
<td>N/A</td>
</tr>
<tr>
<td>Avg. Fitness Score</td>
<td>0.68 - 0.73</td>
<td>0.76 - 0.78</td>
<td>0.8 - 0.83</td>
<td>0.77 - 0.85</td>
<td>N/A</td>
</tr>
<tr>
<td>Avg. Time Spent</td>
<td>0.8(s)</td>
<td>0.5(s)</td>
<td>1.6(s)</td>
<td>3.2(s)</td>
<td>43.7(s)</td>
</tr>
</tbody>
</table>

3. Discussion: We have performed five setups ten times to reduce the deviation of computer performance and reported their performance. According to Table 17 with the same fitness function, the 'heavy' setup obtains the highest average fitness score. However, if we set 'standard' as the baseline, we found that given 220% more time in the 'heavy' setup, the average fitness score only grows between 5.23% to 6.41%, which indicates this setup encounters a convergence issue. Therefore, we argue that in implementation-wise it is trivial to increase generation to 500 under the same fitness function as it only returns minimal growth of the fitness score.

Given the same number of total iterations (i.e., 'standard' vs. 'single standard' and 'heavy' vs. 'single heavy'), the average operation time increases, while the average fitness score drops slightly. We summarize two possible reasons. First, while deviation may be reduced via multiple operations, they might still exist and affect the performance results. Another possible reason is the nature of our fitness function. When the central node manages weights \(\alpha\) and \(\beta\), the searching areas are fully controlled and covered. On the other hand, 'single standard' and 'single heavy' setups assign weights randomly; hence the searching areas may overlap or be missed. Due to this phenomena, compared to multiple nodes
setups (i.e., 'Standard' and 'Heavy'), the average fitness score of single node setups (i.e., 'Single Standard' and 'Single Heavy') becomes more unstable.

The average operation time of RepoPal is of serious concern. To better understand RepoPal’s poor performance, we need to compare the dataset used in this work with the one in the original paper. According to the original study [26], the authors also leverage the data from GHTorrent, however, the scale of the dataset makes a difference. In our study, we include all Java repositories as all of them might potentially become recommendation candidates. That is, over 1.7 million Java repositories need to be processed in each RepoPal calculation. RepoPal sounds promising in the original paper because it only mines 1,000 projects for the study. However, this number is not representative of the GitHub data and its scale. On the other hand, by using parallel computing and random chromosome construction, our model can efficiently deal with the scale. In addition, star sparsity is also of a concern. The original study selects GitHub projects having at least 20 stars. That is, the projects under study are relatively popular. However, our dataset contains small or medium projects, which are unlikely to have that many stars. When RepoPal is running, it spends considerable time calculating the importance of such no-star projects resulting in a heavy computational barrier while generating no results. However, our model controls the size of recommendation candidates by analyzing social connections. This approach finds a balance between a large number of projects hosted in GitHub and exploring all possible candidates.
**Answer to RQ3:** To evaluate our model’s performance, we have designed and performed five setups and compared it with RepoPal. ‘Standard’ setup, that we adopted, takes the least operation time and returns the most acceptable fitness score. Although increasing generations results in positive growth to fitness score, the change is limited. While performing the same number of generations (i.e., iterations), Simple GA has lower average fitness scores and longer operation time compared to Global Parallel GA. Additionally, average fitness scores of SGA are more unstable as it can not fully cover entire searching area and manage weights of $\alpha$ and $\beta$. Last but not least, RepoPal takes the longest time generating recommendations, mainly due to the scale of the data used in this work, i.e., 1.7 million projects. The sparsity of stars (project popularity) may also contribute to RepoPal’s poor performance.
Chapter 5

Discussion

In this chapter, we discuss the results of our investigations. In Section 5.1 we review the key findings of this work. Then, in Section 5.2 we discuss the implications of our work for GitHub and academic community. Finally, we describe threats to validity in Section 5.3.

5.1 Main Findings

Our work presents a new GitHub project recommender system by leveraging Parallel Genetic Algorithm computing. Through the responses of survey participants, we have received not only the feedback on our technique but also empirical evaluation of our system by actual GitHub developers. We have started with the concept of collaborative culture that GitHub has adopted (Section 1.1.2) and proposed a social-based PGA GitHub project recommender system (Section 3.3). In Chapter 4 we have answered the research questions stated in Section 1.3 evaluating the accuracy, effectiveness and performance of our model against the baseline.

In this work, we have extracted the latest GitHub (offline) data from GHTorrent and chose Java language for mining projects and their developers. According to Table 4 and Figure 6, both JavaScript and HTML are most used languages in the dataset,
however, based on the keyword analysis (as shown in Figure 7, Figure 8, and Figure 9) we have selected Java as it is used in more domains and applications in GitHub. Moreover, the previous research [115] in 2019 also stated that Java is one of the most popular programming languages.

As described in Section 3.1.2.1 and Section 3.2.1, this work builds developer expertise vectors by analyzing the file extensions of their commits and utilizes TF-IDF to materialize them. Therefore, we expect the developers to be familiar with (or at least have base knowledge of) Java language. Based on this assumption, all target developers in this research should be Java developers to some degree. Surprisingly, one survey participant P3 states that: "...all those are java project and I don’t know java." However, while we investigate the expertise vector of P3, we found that there are only the weights for '.java' and '.xml' available in the vector. The developer’s commit data has also proved that the developer modified only those two types of files in their commits. It seems that P3’s perception of their Java expertise may be incorrect (i.e., “perception is not reality”). The purpose of the expertise vector in this research is to provide information about developers’ programming interests, and after checking the commit files, we found that P3 does know how to modify Java files as the commit data suggests. Therefore, based on our definition of expertise, developer P3 can certainly modify Java code and thus can be considered a Java developer (perhaps an occasional one). In fact, our approach of determining expertise is more precise than previous research [17, 26, 116] that only considers the primary language (PR) of the project or the project’s README file to infer developers’ expertise. Considering only these two attributes (PR and README), the extracted expertise may be incorrect or missing. Hence, we argue that compared to the README-based or PR-based expertise, our commit-based TF-IDF expertise vectors provide a more accurate and complete picture of the developer’s skillset.

In this research, we consider Actual Social Network (ASN) and Inferred Social
Network (ISN) for generating recommendation candidates. While it is ideal to rely on actual social network of developers, the reality is that one half of the GitHub developers do not have any social network, and nearly 25% of them only have one or two connections (as shown in Figure 1). In fact, the survey results shown in Figure 24 in Section 4.1 provide further evidence that most GitHub developers do not consider their social connection to be useful for recommending future projects. As previous research [117,118] suggests, GitHub developers create social connections with others because they are friends or they are influential in the real-world, rarely because they are professional in certain GitHub projects. Therefore, based on these evidence, leveraging ISN is necessary.

This work leverages interest vector and expertise vector to support out framework in providing the best recommendation combination. Previous works [28,54,55] mainly utilize fork, watch or pull-request data to represent the developer’s interests to recommend projects. On the other hand, as discussed early on, previous research [117,118] tend to use README information or primary language as the baseline of developer’s expertise. However, it is impractical and inappropriate to set fixed weight values
for all developers. Therefore, we leverage parallel computing for our GA framework by distributing weights and making sure all search areas are covered. According to the survey responses, our model outperforms in terms of accuracy, developer preference and performance compared to the traditional approach (i.e., the RepoPal [26] baseline).

Regarding our model, it is worth mentioning that as GitHub hosts over 190 million repositories (as of January 2020), the scalability issue for any GitHub project recommender systems is critical and inevitable. As mentioned in Section 3.3, instead of containing all candidates in chromosomes, we have decided to limit the length of chromosomes to 10 candidates, while slave nodes randomly decide on the candidates of chromosomes. This is because active developers (i.e., those who participate in a number of projects) may have thousands of candidates via ISN, which may slow down the performance as each chromosome of these developers can be quite lengthy. By doing this, we can make sure that our model’s recommendations are generated in a relatively short time. To decide on the correct number of generations and threads, we have designed four different setups and recorded their performance. As discussed in Section 4.3, we found that though a large generation number leads to a better fitness score, the operation time costs become higher. Secondly, to compute our fitness function with the same dataset, parallel computing is required for making it cost efficient and stabilizing the fitness score.

To compare the accuracy and effectiveness of our model against the baseline (i.e., RepoPal), we have conducted a survey inviting 490 front-line developers and asked them to share their feedback and opinion about the recommendations of both models. According to the results listed in Sections 4.1 and 4.2, our model outperforms RepoPal as our framework can offer recommendations based on both the interests and expertise of developers. Moreover, even though only 43.8% of developers find their social connections useful for finding appropriate GitHub projects, over half of the
participants expressed their willingness to use our framework in the future compared
with 12.5% for RepoPal.

Finally, from the survey feedback, we found that participants tend to dislike forked
projects when evaluating the recommendations as they believe such projects bring no
added value and less interesting. We also noticed that although many participants can
detect that our model’s recommendations are based on their social connections, few of
them argue that these recommended projects are from their colleagues, which affects
their willingness to participate in such projects. Finally, during the data collection
phase we did not eliminate ‘archived’ projects, and therefore some feedback suggested
that while developers found the recommendations interesting, they were not able to
join projects as those projects have been inactive for years.

5.2 Implications

Nowadays, the number of GitHub projects increases rapidly, and developers are com-
monly seeking new challenges. We believe that the findings of our research would be
embraced by two categories of the stakeholders: 1) research community in particular
researchers interested in developing recommender systems for GitHub, and 2) GitHub
developers. We next describe the implications of our work for these two groups.

5.2.1 Implications for Research Community

As more and more developers and organizations develop their projects on GitHub,
the number of repositories in GitHub has reached a new level. In the year 2020 alone,
over 60 million new repositories were created in GitHub. Our key motivation for this
work is to help new developers mitigate onboarding challenges or assist experience
developers in finding new projects to join. These challenges do still exist, while only
a few solutions have been previously offered. 119 123
When recommending new GitHub projects, we provide an alternative solution by considering developers’ social connections. To resolve the lack of social network, we have proposed the Inferred Social Network technique by creating a network of all the peers from the projects that the developer have participated in. We leverage social network feature for finding possible projects to recommend.

Secondly, inspired by the previous works [23][74][124], we have leveraged a GP-based Parallel Genetic Algorithm architecture for project recommendation. In this research, we have designed the weight of developers’ interests and expertise, and used the master node for distributing them. Each slave node only needs to search a specific area but do so more efficiently and at scale. The length of chromosome is also narrowed down to 10 to overcome computational barriers.

5.2.2 Implications for GitHub Community

Being the largest open-source software hosting platform, GitHub has the greatest number of developers with different interests, expertise, and communities worldwide. We believe this work can bring some inspiration to developers, project owners, and GitHub management.

1. Developers: Our survey results suggest that most developers do not perceive social network component as useful to be leveraged for recommending new projects to join. However, our findings demonstrate that even inferred social connections can help explore new and creative projects. Therefore, for GitHub developers, we recommend seeking/making social connections as they can not only improve personal portfolios but bring more opportunities.

2. Project Owners: During our empirical exploratory work, we found that many intermediate or small-scale projects tend to leave README descriptions unclear or empty. The purpose of such documentation is to offer GitHub users
clear projects’ content, purposes, vision, etc. Many software engineering researchers mine and utilize such information for collecting different attributes about the projects. Any unclear or empty description can lead projects to become less visible or even be eliminated from the collected data. Therefore, we advise project owners to maintain detailed and high quality README descriptions for their GitHub projects.

3. GitHub Management: The key challenge for developing a reliable recommender system is the quality of the data source. It would be ideal for GitHub to offer data mining and collection capabilities for conducting research that relies on the GitHub data. While GitHub offers APIs for mining their data, API query limits do create a barrier for researcher. Even using the offered paid membership makes any data science research insufficient.

GHTorrent, a mirror of the GitHub data, has been the most appropriate choice for our research. However, GHTorrent has known issues with its data quality, as well as not being updated as often as needed (most recent version is dated to June 2019).

5.3 Threats to Validity

In this research, we identified several threats to the validity of our work, and we now describe them and report the measures we took to curb them.

1. Quality of GHTorrent Our research relies on the GHTorrent dataset which offers most recent version as of June 2019. That is, any data newer than June 30th, 2019 is not included in this research. However, this suggest a limitation that our recommendations may miss developers’ current interests and expertise. As discussed in Section 4.1, the data quality and the potential time gap
may affect our survey participants’ trust and interests in both recommendation models.

2. **Data preprocessing:** Any data preprocessing technique is subject to some information loss (as we perform TF-IDF and VSM for building profiles). We build several models by applying different options of preprocessing algorithms to make sure our data is as complete as possible.

3. **Quality of our PGA model:** As the quality of recommendations depends on the fitness function we proposed, there is some room for improved function in the future. Furthermore, due to GA’s nature, it is possible to have more or less than five recommendations. For tackle this problem, we proposed a weight-distributed fitness function and adjusted the penalty function many times to make sure our model generates five recommendations.

4. **Human bias:** There is a possibility of human error in selecting the right answers in the survey. Participants may also return invalid responses even though we already set a code reversed question in the survey. To mitigate this, we manually checked all questionnaires to ensure all responses are valid.

5. **Project availability:** GitHub projects can be created, modified, become private or deleted at any time. Therefore, while selecting projects and sending recommendations to our survey participants, we checked whether all projects were available and unarchived. However, as discussed in Chapter 4, several participants reported that some of their recommendations were unavailable or already archived. It is unfeasible for researchers to mitigate this issue as any project can become private, archived or deleted anytime (e.g., only hours later after recommendations are generated).
Chapter 6

Conclusions

We conclude this thesis with the thesis statement. A list of our contributions is presented in Section 6.1 and our suggestions for future directions are discussed in Section 6.2.

**Thesis Statement:** Onboarding projects is a long-live challenge for open-source software developers. Developers may generally evaluate their own interests and expertise when making decisions on what project they can contribute to. However, the social aspect might be the one they often ignore. Our research proposes a promising solution for recommending GitHub projects by leveraging social connections and matching developers’ programming expertise and their interests to the right projects.

6.1 Summary of Contributions

This work proposed a new GP-based Parallel Genetic Algorithm for recommending GitHub projects and made the following contributions.

To the best of our knowledge, while Genetic Algorithm has been used for cataloguing repositories, predicting configurations, recommending topics or code reviewers in GitHub, this work is the first study that leverages GA to recommend GitHub projects.
First, this research proposed Inferred Social Network (INS) for searching for potential recommendation candidates if Actual Social Network (ASN) of developers is insufficient. This approach allows us to extend and explore more possible projects for developers.

Second, instead of considering only developers' interests or expertise data, in this work, we collect, organize, and build developers' interest and expertise profiles by using TF-IDF and VSM techniques.

Third, we proposed a weight-distributed fitness function managed by the central node. This approach can maximize the searching areas while performing the GA technique and ensuring that all sub-nodes (i.e., slave nodes) search their own areas extensively by providing sufficient generations without trading additional operation time or overlaps with other sub-nodes.

Fourth, our model overcomes scalability issues compared with other recommendation frameworks by shortening the length and randomly picking candidates to form chromosomes.

Fifth, according to the survey responses, our model earns preference from more developers compared to the baseline, and over 56% of developers are interested in using our model in the future.

Finally, while developers find little value in adding social aspects to recommender systems, our research demonstrates that such connections can have a positive impact on GitHub project recommendations.

6.2 Future Work

During the last stage of this work, we identified several directions for future improvement. In this study, we relied on the offline data as our primary dataset, and we found that while the dataset is only nearly 1.5 years old, the recommendations based
on the dated dataset negatively impact the satisfaction of our survey participants. For future studies, we suggest that “live” data at a smaller scale can improve the quality of recommendations.

This research utilizes the weights of interests and expertise to evaluate the quality of recommendation combinations. However, we have only extracted the two most popular developers’ attributes. In the future, we plan to include more aspects or attributes from the developer data to reflect a more complete developer profile. Similarly, project profiles can be improved by extracting more aspects than just README descriptions and watchers of the projects. In addition, while the README description can provide fine-grained data of the project, many middle or small-scale projects may leave their README descriptions empty or unclear. In the future, it is possible to mine the source code of the projects to obtain a more complete picture of projects. In fact, previous research by DéjàVu \cite{125} is already analyzing the project source code for detecting code duplications. Code2vec \cite{126} is also a trending technique that analyzes the source code and leverages neural network for predicting a function’s name inside the code.

Besides, due to our the current genotype representation, the current version requires developers to have at least ten recommendation candidates for our model to work. Furthermore, we found that a small percentage of developers could not satisfy this precondition even when leveraging the Inferred Social Network technique. To maximize the number of developers who can benefit from our model, we can extend the searching depth of ISN. However, we need to have a better understanding to what extent inferred relations can accurately represent developers’ interests.

For GA’s aspect, we can use different genotype representations. In fact, considering the purpose of our model is to decide five projects from chromosome as recommendations, it is possible to use permutation representation as it can sort projects and return a priority list of recommended projects. By doing this, the issue with
duplicated projects in one chromosome would no longer exist. In addition, different
crossover and mutation formats can be applied to our model as well. For example,
the uniformed crossover approach may further increase the diversity of chromosomes
compared to a single-point crossover. Mutation approach can be improved as well.
Instead of changing the status of accepted or rejected, replacing the gene with other
project candidates could be a better solution for increasing diversity and reducing the
premature convergence problem. Last but not least, trying different sets of genera-
tions, initial population, termination conditions are also possible to further improve
our current model.

While this research utilizes the weight-distributed approach to maximize the
searching area, it may result in unnecessary computational resources in the trivial
searching areas. A better solution to overcome this issue can be a multi-objective op-
timization approach. We can narrow down multiple objectives into a single objective
to simplify the real-world problems.

Finally, according to the survey participants’ feedback, ’forked’ projects are per-
ceived as inappropriate recommendations as they lack new ideas and originality.
Moreover, since ‘archived’ projects are not open for any new contributions, they
should be removed from the recommendation list.
List of References


Appendix A

Research Ethics Clearance
CERTIFICATION OF INSTITUTIONAL ETHICS CLEARANCE

The following research has been granted clearance by the Carleton University Research Ethics Board-B (CUREB-B). CUREB-B is constituted and operates in compliance with the Tri-Council Policy Statement: Ethical Conduct for Research Involving Humans (TCPS2).

Ethics Clearance ID: Project # 115243

Project Team Members: Mr. PoKai (Lance) Wang (Primary Investigator)
Dr. Olga Baysal (Research Supervisor)

Study Title: Evaluating the performance of the social-based recommender framework

Funding Source: (If applicable):

Effective: March 02, 2021                      Expires: March 31, 2022

This certification is subject to the following conditions:

1. Clearance is granted only for the research and purposes described in the application.

2. Any modification to the approved research must be submitted to CUREB-B via a Change to Protocol Form. All changes must be cleared prior to the continuance of the research.

3. An Annual Status Report for the renewal or closure of ethics clearance must be submitted and cleared by the renewal date listed above. Failure to submit the Annual Status Report will result in the closure of the file. If funding is associated, funds will be frozen.

4. During the course of the study, if you encounter an adverse event, material incidental finding, protocol deviation or other unanticipated problem, you must complete and submit a Report of Adverse Events and Unanticipated Problems Form.

5. It is the responsibility of the student to notify their supervisor of any adverse events, changes to their application, or requests to renew/close the protocol.

6. Failure to conduct the research in accordance with the principles of the Tri-Council Policy Statement: Ethical Conduct for Research Involving Humans 2nd edition and the Carleton University Policies and Procedures for the Ethical Conduct of Research may result in the suspension or termination of the research project.
IMPORTANT: Special requirements for COVID-19:

If this study involves in-person research interactions with human participants, whether on- or off-campus, the following rules apply:

1. Upon receiving clearance from CUREB, please seek the approval of the relevant Dean for your research. Provide a copy of your CUREB clearance to the Dean for their records. See Principles and Procedures for On-campus Research at Carleton University and note that this document applies both to on- and off-campus research that involves human participants. Please contact your Dean’s Office for more information about obtaining their approval.

2. Provide a copy of the Dean’s approval to the Office of Research Ethics prior to starting any in-person research activities.

3. If the Dean’s approval requires any significant change(s) to any element of the study, you must notify the Office of Research Ethics of such change(s).

Upon reasonable request, it is the policy of CUREB, for cleared protocols, to release the name of the PI, the title of the project, and the date of clearance and any renewal(s).

Please email the Research Compliance Coordinators at ethics@carleton.ca if you have any questions.

CLEARED BY: Bernadette Campbell, PhD, Chair, CUREB-B

Date: March 02, 2021

Natasha Artemeva, Co-Chair, PhD, Vice Chair, CUREB-B
Appendix B

Consent Questionnaire
Introduction

Hi GitHub developer,

Based on your contributions and activities in GitHub, I would like to invite you to participate in this survey. Before you participate in the survey, we would like to receive your consent.

Participation
You are invited to participate in this research project. After you read this document and agree to participate in the project, you will receive and be asked to complete two questionnaires about GitHub project recommendation. Completing the questionnaires will take approximately 10 to 15 minutes.
Your participation is voluntary, meaning you may decline if you wish. You can cease participating at any time by leaving any remaining questions blank. You will not have to provide a reason to justify your decision. Once you have finished completing the questionnaire and submitted it, you will no longer be able to withdraw from the experiment.

Privacy
The data collected for this project will be confidential to the extent permitted by law. This information will be kept by the lead researcher on the project at Qualtrics servers for six months. In order to preserve your identity and the confidentiality of your information, the information gathered will be coded. The data may be published in reports, articles or be the subject of scientific discussion. However, it will not be possible to identify you in any of these cases. Your response will be stored in Qualtrics servers which are located in North America region and will be destroyed within six months.

Ethical Considerations
This research has been cleared by Carleton University Research Ethics Board (CUREB-B Clearance #115243). Should you have any ethical concerns with the study, please contact the REB Chair, Carleton University Research Ethics Board-B, by email: ethics@carleton.ca.
Contact Person
If you have questions about the project or your role in it or wish to withdraw from the questionnaire, please contact (Lance Po-Kai Wang, project lead, carletonu.thesis30@gmail.com)

Q1. Consent
(Click to select)

☐ I WOULD like to participate in the survey, please send me my recommendations
☐ I don't want to participate in the survey
Appendix C

Survey Questionnaire
Carleton University - SCS Survey

Dear GitHub user @JohnDoe123,

Based on your contributions and activities in GitHub, I would like to invite you to participate in this survey. The survey will take approximately 10-15 minutes to complete. Your participation in this study is entirely voluntary.

Based on your GitHub activities related to Java projects from July 2017 to June 2019, we developed two recommender systems to suggest other Java projects you might want to join and contribute to. For each recommendation list, the system offers up to five GitHub projects that might suit you. You can click the name of any project to visit the GitHub repository to get more details. It shall help you evaluate the recommendations.

----------------------------------

***Participation
You are invited to participate in this research project. After you read this document and agree to participate in the project, you will be asked to complete two questionnaires about GitHub project recommendation. Completing the questionnaires will take approximately 10 to 15 minutes.

Your participation is voluntary, meaning you may decline if you wish. You can cease participating at any time by leaving any remaining questions blank. You will not have to provide a reason to justify your decision. Once you have finished completing the questionnaire and submitted it, you will no longer be able to withdraw from the experiment.

----------------------------------

***Privacy
The data collected for this project will be confidential to the extent permitted by law. This information will be kept by the lead researcher on the project at Qualtrics servers for six months.

In order to preserve your identity and the confidentiality of your information, the information gathered will be coded. The data may be published in reports, articles or be the subject of scientific discussion. However, it will not be possible to identify you in any of these cases. Your response will be stored in Qualtrics servers which are located in North America region and will be destroyed within six months.

----------------------------------

***Ethical Considerations
This research has been cleared by Carleton University Research Ethics Board (CUREB-B Clearance #115243). Should you have any ethical concerns with the study, please contact the REB Chair, Carleton University Research Ethics Board-B, by email: ethics@carleton.ca.

----------------------------------

***Contact Person
If you have questions about the project or your role in it or wish to withdraw from the questionnaire, please contact (Lance, project lead, carletonu.thesis30@gmail.com)

*Required
1. Consent *

Tick all that apply.

☐ I have read and understood the project description
☐ I AGREE to participate by completing the questionnaires
☐ I AGREE my data will be stored and protected by Qualtrics Servers in North America region, but may be disclosed via a court order or data breach

This is your recommendations of Model B:

(1) tomlooman / ActionRoguelike (https://github.com/tomlooman/ActionRoguelike)
(2) Marak / faker.js (https://github.com/Marak/faker.js)
(3) minio / minio (https://github.com/minio/minio)
(4) google / tf-quant-finance (https://github.com/google/tf-quant-finance)
(5) popey / sosumi-snap (https://github.com/popey/sosumi-snap)

Based on your opinion, please answer the following questions regarding recommendations of Model B:

2. I find the recommendations helpful. (B-1) *

Mark only one oval.

☐ Agree
☐ Neutral
☐ Disagree

3. I will try out at least one of the recommendations. (B-2) *

Mark only one oval.

☐ Agree
☐ Neutral
☐ Disagree
4. I perceive the recommendations reflect my programming expertise (i.e., programming skills). (B-4) *

Mark only one oval.

☐ Agree
☐ Neutral
☐ Disagree

5. I will use this model for future project recommendations. (B-5) *

Mark only one oval.

☐ Agree
☐ Neutral
☐ Disagree

6. Do you have any additional feedback or comments about Model B? (B-6)

__________________________________________________________________________
__________________________________________________________________________
__________________________________________________________________________

This is your recommendations of Model B:

(P1) Marak / faker.js (https://github.com/Marak/faker.js)
(P2) google / tf-quant-finance (https://github.com/google/tf-quant-finance)
(P3) JetBrains / kotlin (https://github.com/JetBrains/kotlin)
(P4) vueuse / vueuse (https://github.com/vueuse/vueuse)
(P5) minio / minio (https://github.com/minio/minio)

Based on your opinion, please answer the following questions regarding recommendations of Model B:
7. I find the recommendations helpful. (S-1) *

Mark only one oval.

☐ Agree  
☐ Neutral  
☐ Disagree

8. I will try out at least one of the recommendations. (S-2) *

Mark only one oval.

☐ Agree  
☐ Neutral  
☐ Disagree

9. I perceive the recommendations DO NOT reflect my project-selection preferences. (S-3) *

Mark only one oval.

☐ Agree  
☐ Neutral  
☐ Disagree

10. I perceive the recommendations reflect my programming expertise (i.e., programming skills). (S-4) *

Mark only one oval.

☐ Agree  
☐ Neutral  
☐ Disagree
11. I will use this model for future project recommendations. (S-5) *

Mark only one oval.

☐ Agree
☐ Neutral
☐ Disagree

12. How would you rank project <P1> (S-6.1) *

https://github.com/Marak/faker.js/

Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Highly Dislike | Highly Like

13. How would you rank project <P2> (S-6.2) *

https://github.com/google/tf-quant-finance/

Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Highly Dislike | Highly Like

14. How would you rank project <P3> (S-6.3) *

https://github.com/JetBrains/kotlin/

Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Highly Dislike | Highly Like
15. How would you rank project <P4> (S-6.4) *
   https://github.com/vueuse/vueuse/

   Mark only one oval.
   
<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<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>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Highly Dislike</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Highly Like</td>
</tr>
</tbody>
</table>

16. How would you rank project <P5> (S-6.5) *
   https://github.com/minio/minio/

   Mark only one oval.
   
<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<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>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Highly Dislike</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Highly Like</td>
</tr>
</tbody>
</table>

17. Do you have any additional feedback or comments about Model S? (S-6)

   ________________________________
   ________________________________
   ________________________________
   ________________________________
   ________________________________

Recommendation Aspects

18. Which recommendations you found useful? *

   Mark only one oval.
   
   - [ ] Model B
   - [ ] Model S
   - [ ] Both models
   - [x] None of them
19. Which aspects of recommendation you find useful? *

Tick all that apply.

- [ ]Past activities (e.g., prs, commits)
- [ ]My social connections
- [ ]My expertise (i.e., programming languages)
- [ ]My interests (e.g., projects I follow)

This content is neither created nor endorsed by Google.