UCL DEPARTMENT OF
COMPUTER SCIENCE

Research Note
RN/16/06

PREM: Prestige Network Enhanced Developer-Task Matching for Crowdsourced Software Development

August 9, 2016

Ke Mao[†], Qing Wang[‡], Yue Jia[†] and Mark Harman[†]
[†]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK
[‡]Institute of Software, Chinese Academy of Sciences, Beijing, China
k.mao@cs.ucl.ac.uk, wq@itechs.iscas.ac.cn, yue.jia@ucl.ac.uk, m.harman@ucl.ac.uk

Abstract

Many software organizations are turning to employ crowdsourcing to augment their software production. For current practice of crowdsourcing, it is common to see a mass number of tasks posted on software crowdsourcing platforms, with little guidance for task selection. Considering that crowd developers may vary greatly in expertise, inappropriate developer-task matching will harm the quality of the deliverables. It is also not time-efficient for developers to discover their most appropriate tasks from vast open call requests. We propose an approach called PREM, aiming to appropriately match between developers and tasks. PREM automatically learns from the developers' historical task data. In addition to task preference, PREM considers the competition nature of crowdsourcing by constructing developers' prestige network. This differs our approach from previous developer recommendation methods that are based on task and/or individual features. Experiments are conducted on 3 TopCoder datasets with 9,191 tasks in total. Our experimental results show that reasonable accuracies are achievable (63%, 46%, 36% for the 3 datasets respectively, when matching 5 developers to each task) and the constructed prestige network can help improve the matching results.

1. INTRODUCTION

Emerging social media and open innovation techniques have reshaped the way people collaborate and share information, which have also offered opportunities for evolutionary changes in software development paradigm [6]. Many successful software companies turn to employ decentralized software ecosystems such as open source and crowdsourcing communities to augment their software production. Crowdsourced Software Development (CSD) can be viewed as a specific mode of Global Software Development (GSD) [13]. It utilizes an open call format to attract geographically distributed online developers for accomplishing various types of software development tasks such as architecture, component design, component development, testing and bug fixing.

The crowdsourcing formulation widely in use was defined by Jeff Howe [14] in 2006. Although crowdsourcing is not specially proposed for software engineering domain, the trend of its application in software development keeps growing. A crowdsourcing industry report from Massolution [22] indicates the number of workers engaged in software development gained 151% in the year 2011. This increase is even more dramatic than the increase in crowdsourced micro tasks. Amazon Mechanical Turk (AMT)[1] is one of the most popular marketplace for crowdsourcing micro tasks such as photo tagging and logo designing. Crowdsourcing platforms that support software development include TopCoder, uTest, GetACoder, eLance, Freelancer, Tackcn, etc. Among them, TopCoder[2] has the world's largest community for crowdsourced software development. Its clients include lots of famous IT companies such as Google, Microsoft, Facebook and AOL. Compared with traditional software development, TopCoder's crowdsourced development exhibits the ability to deliver customer requested software assets with lower defect rate at an order of magnitude lower cost in less time [5, 17].

Current crowdsourcing approaches in practice adopt a pull methodology where tasks are posted on specialized online platforms. Although this enables simplicity and tasks accessibility, it provides little guidance in task selection and cannot guarantee that the tasks are performed by the best suitable developers: Usually there are lots of simultaneously competitive tasks posted on crowdsourcing platforms. For instance, illustrated as Figure 1, from February 10th 2014 to March 10th 2014, there are an average of 89 simultaneously active tasks available on TopCoder platform for a single day. According to a study [8] based on AMT platform, most crowd workers usually only view first few pages of recent tasks posted on the platform. Thus the appropriateness of the developer-task matching resulted by the pull methodology is questionable.

Considering that the crowd developers vary a lot in expertise, inappropriate developer-task matching can harm the quality of the deliverables. Also, developers need to spend much time in browsing vast amount of task description in order to choose their suitable ones. In this paper we propose a developer-task matching approach called PREM which enables ideal developer-task matching result to be actively pushed to developers.

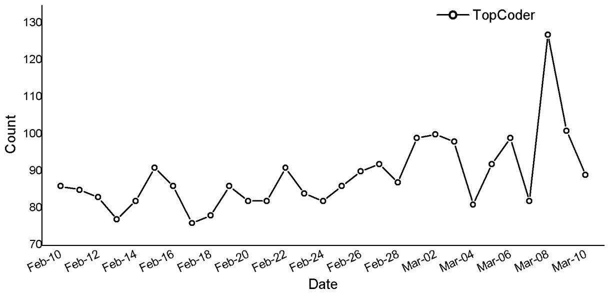

Figure 1: Daily available tasks on TopCoder.

Specifically, we propose to consider the competition nature of the emerging crowdsourcing platform when suggesting the matching result, because the competition relationship among developers can affect their task selection behaviour. For example on TopCoder platform, it is observed that highly skilled developers face tougher competition, thus they register particular task early, which deter the entry of their opponents in the same task [4]. The feature of dealing with crowd developers' competition relationship differs our approach from previous developer recommendation or task triage methods in classical software development paradigm. The major contributions of this paper are:

- PREM, a novel developer-task matching approach enhanced by developers' prestige network. Our approach automatically learns from the developer's historical tasks and considers the competition nature of crowdsourcing (e.g., the "cheap talk" phenomenon [10]) to better match between available tasks and developers.
- CSD task features and similarity measures are proposed for learning task preference. Prestige network construction and corresponding refinement algorithms are designed to consider the competition relationship among developers.
- An empirical evaluation of our method conducted on 3 datasets, corresponding to development, assembly and bug fixing contexts separately. The datasets contain as many as 9,191 successful tasks crowdsourced on TopCoder, the world's largest crowdsourcing platform for software development. The results show with the reasonable accuracy and the effectiveness of prestige network based enhancement.
- An implementation of our approach, CrowdDev platform. The platform is publicly available via the Internet to support global crowd developers.

The rest of this paper is organized as follows: We begin with Section II, describe CSD process, and introduce the competition in CSD as background information. Section III introduces our PREM approach to developer-task matching for CSD tasks. The evaluation of our work is presented in Section IV. This is followed by a discussion in Section V. Section VI presents related work. Finally Section VII concludes and presents directions for future work.

[1] AMT Website: http://www.mturk.com/

[2] TopCoder Website: http://www.topcoder.com/

2. BACKGROUND

Prior to presenting our approach of developer-task matching, we introduce the background knowledge in CSD process. Since TopCoder is currently the world's largest CSD platform which supports a global crowd of more than 600,000 developers. The platform is famous for its CSD methodology. It is based on this platform shall we introduce the process.

The overall process of CSD used by TopCoder platform is achieved by the open competition format for each development phase, utilizing TopCoder's global talent pool of developers. The development process commences with a requirement phase. During this phase, the project manager, who comes from the crowd or the platform and is responsible for managing following phases and communicating with the client companies to identify their project goals, task plan and estimated budget. Then the requirements specification are defined and are passed as the input to the next phase. The subsequent architecture phase decomposes the application into a set of components. (TopCoder's CSD is also the practice of component-based software engineering). The component design activity produces a full set of design documentation such as UML diagrams and component specifications.

These specified components design are then implemented in the subsequent development phase. The component development activity may incorporate pre-built reusable components. The finished components are combined together in an assembly phase and are further certified by system level testing activities. Assembly tasks require online developers build the application by assembling the winners' component implementations according to the architecture design. Finally, the fully functioning solution is deployed into the customer's quality assurance environment in the deployment phase. After a period of user acceptance testing, all developed assets are delivered to the client. For further maintenance activities, TopCoder provides "Bug Hunt" and "Bug Races" tasks for discovering and fixing bugs.

The essential of crowdsourcing is the "wisdom of the crowd". The above introduced CSD process relies on highly interactive and social communities to finish the CSD tasks. For the crowd developers, on one hand they collaborate on the higher-level process, for example when a developer implements a component, he may inquire the designer about the UML diagrams and may discuss the application framework with the architect via the community forum. On the other hand, they also compete against each other on a specified task. Because the open call format is usually achieved by a series of contests. This competition nature of crowdsourcing is one of the key features that distinguish CSD from OSS development. An previous study [19] on CSD shows a higher competition level can be beneficial in improving software quality.

As for the TopCoder platform, it has a reputation system to help developers strategically choose their opponents. For instance, lower rated developer's entry may be deterred by the higher rated developer [4]. If we consider the reputation from a social network perspective, we can draw a prestige network from the developers' participation and winning history. One such real prestige network we extract from the TopCoder platform is given in Figure 3. Later we will show how to construct this network and how to utilize it in developer-task matching.

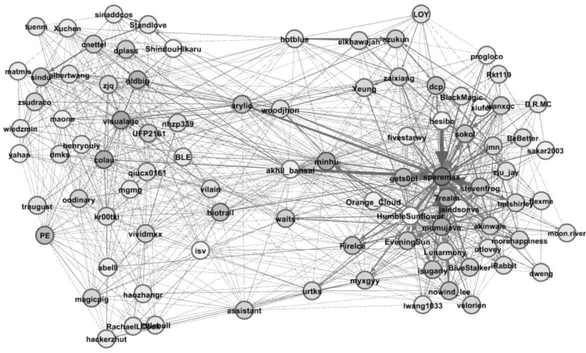

Figure 3: A real developer prestige network.

3. OUR APPROACH

In this section we explain the key ideas behind our PREM approach. We present the framework of PREM , describe the two components of PREM (i.e., Content Based Initial Matching and Prestige Network Based Enhancement). We show two types of developers in the prestige network whose competition relationship are considered in order to refine the developer-task matching result.

The framework of our PREM approach is shown in Figure 2, where the hollow arrows indicate the learning process based on historical data, while the solid arrows illustrate the matching process for new arriving tasks. PREM for developer-task matching is achieved via both task content and developers' prestige network extracted from the CSD websites. Here the content refers to a group of features describing a CSD Task, such as required techniques, payment, title, description, posted data and delivering deadline, etc. The learner tries to capture the characteristics (e.g., expertise, expected payment and task duration, etc.) of each developer from their historical activities and match the most suitable developers to new arriving tasks. This initial result is further refined by the process of prestige network based enhancement.

3.1 Content Based Initial Matching (CBIM)

The fundamental assumption of the content base matching method is that developers will prefer or be familiar with the tasks which are "similar" to the tasks they have performed in the past. We employ multi-class, single-label classification techniques to automate the process of identifying developers who may be suitable with the new arriving tasks. Here we treat developer as class, each task in historical records is labelled with the first place winner. The matching approach consists of the following steps:

The first step is data filtering. We filter those historical tasks (i.e., training set) with incomplete information (e.g., missing winner label or lack of task description). To exclude irrelevance empirical knowledge, we remove the tasks which development type do not match current application domain

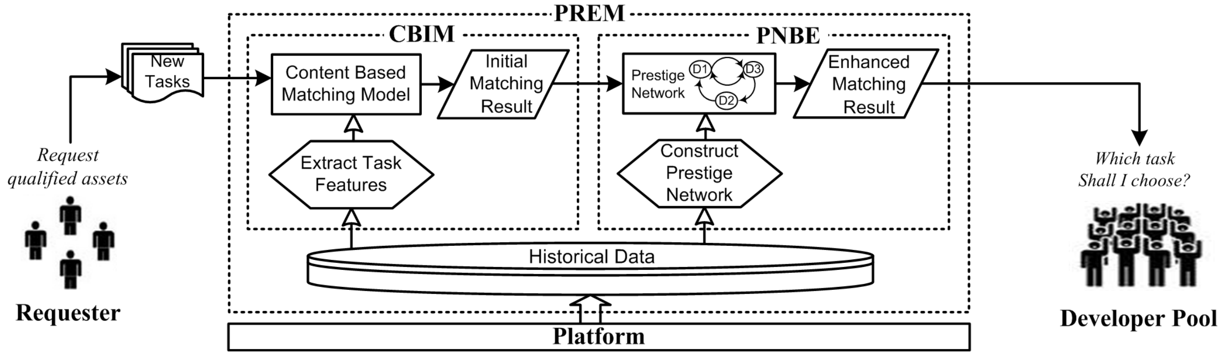

Figure 2: The framework of PREM.

Table 1: Features of a Task

Feature	Format	Description
Date	Numeric	Post date of the task.
PL	Text	Programming Language used.
Title	Text	Title of the posted task.
Tech	Text	Techniques used.
Description	Text	Detailed task description.
Duration	Numeric	Time allocated to the task.
Payment	Numeric	Reward of the task.

(e.g., development, assembly, bug fixing, etc.). Also, we treat a developer won at least 5 tasks as a skilled developer in this paper, and filter out those unskilled developers. Then the filtered data are passed to the subsequent step for feature extraction.

3.1.1 Task Feature Extraction

Each CSD task on TopCoder platform is described by a variety of information. For feature extraction in this paper we extract text features including title, description, programming language and techniques. Numeric features include task post date, allocated task duration and payment. The detailed feature description is presented in Table 1. In order to unify numeric and text features, we convert each text feature into word vector format, keeping only meaningful and descriptive tokens processed by tokenization and stop word (e.g., 'a', 'the', 'and', 'of', 'is', 'this', etc.) removal. To be specific, suppose there are m terms after tokenization and stop word removal for a text feature, the corresponding vector of this feature in task t would be $v_t = (w_{t,1}, w_{t,2}, ..., w_{t,m})$, where $w_{i,j}$ stands for the weight for each term $term_j$, which is calculated by Term Frequency-Inverse Document Frequency (TF-IDF) according to Equation 1.

$$w_{i,j} = \big(1 + log(tf_j)\big) * log\frac{|T|}{df_i} \tag{1}$$

Here tf_j is the number of times $term_j$ appears in task description i, $|T|$ is the total number of tasks and df_i equals the number of tasks containing $term_j$. The extracted text feature vectors together with the numeric features are then used for model building.

3.1.2 Model Building

In this step we firstly train a classification model on prepared historical tasks. Various of supervised learning algorithms can be adopted. Note that for each new arriving task, the algorithm should be able to generate a distribution of probabilities for all developer labels, such algorithms can be C4.5 Decision Tree [25], Random Forest [20], Naïve Bayes [18], K-Nearest Neighbour [9], etc. Note that in order to use the K-Nearest Neighbour algorithm, we need to define the similarity measure. In this paper we define the similarity Sim between two task t_i and t_j according to the post date distance, difference in task programming language ,matched number of techniques, allocated duration distance, payment difference and the text matching degree (i.e. cosine similarity of two vectors) in title and description. Sim is define according to Equation 2.

$$\begin{aligned} Sim(t_i, t_j) = w_1 Dis(F_{1,i}, F_{1,j}) + w_2 Dis(F_{2,i}, F_{2,j}) \\ + w_3 Dis(F_{3,i}, F_{3,j}) + ... + w_n Dis(F_{n,i}, F_{n,j}) \end{aligned} \tag{2}$$

Where w stands for the weight assigned to the corresponding feature (in this paper we use equal weights i.e. 1.0 for all features), Dis indicates the distance function which vary among different features. The definition of distance measures in this paper is shown in Table 2.

Lastly, we apply the trained model on new arriving tasks, the ranked top N developers from the generated probability distribution are matched to each corresponding task.

The output of CBIM is a list of candidate developers $D =$

Table 2: Feature Distance Measures

Feature	Distance Measure
Date	$(Date_i - Date_j)/Date_{MaxDiff}$
PL	$PL_i == PL_j ?1 : 0$
Title	$\frac{Tit_x \cdot Tit_y}{\|Tit_x\|\|Tit_y\|}$
Tech	$Match(Tech_i, Tech_j)/NumberOfTechs_{Max}$
Description	$\frac{Des_x \cdot Des_y}{\|Des_x\|\|Des_y\|}$
Duration	$(Duration_i - Duration_j)/Duration_{Max}$
Payment	$(Payment_i - Payment_j)/Payment_{Max}$

```
Algorithm 1: Constructing the Prestige Network
Input: The historical tasks set $T$, the registrants set $R_i$
       and the winners set $W_i$ for each task $t_i \in T$
Output: The prestige network $G$
Node set $V \leftarrow \emptyset$, edge set $E \leftarrow \emptyset$
for each task $t_i \in T$ do
    for each winner $w_j \in W_i$ do
        add node $w_j$ to $V$ if $w_j \notin V$
        for each registrant $r_k \in R_i$ do
            add node $r_k$ to $V$ if $r_k \notin V$
            if $r_k \neq w_j$ then
                add edge $(r_k, w_j)$ to $E$
Prestige network $G \leftarrow (V, E)$
return $G$
```

$\{d_1, d_2, ..., d_n\}$, where d_1 to d_n are sorted descendingly according to their probabilities of being the most suitable developers, assessing from the task content perspective.

3.2 Prestige Network Based Enhancement

The initial matching result can be regarded as the seed and is used as the input for further refinement by Prestige Network Based Enhancement (PNBE). The basis idea of PNBE is to consider the competition nature of crowdsourcing and better match between developers and tasks. PNBE contains the steps as follows:

3.2.1 Prestige Network Construction

In this paper we define the prestige network as a social network connotating the completion nature among developers. If one developer defeated another developer by winning a task, there is a directed edge drawn from the loser node to the winner node. The process for constructing the prestige network is described as Algorithm 1.

The input of this algorithm include a set of historical tasks, and corresponding registrants and winners for each task in the set. The algorithm iterates through each task, adds directed edges from each registrant to each winner in the task. Duplicate edges are allowed. Finally the constructed prestige network represented by a graph is returned.

Motivation: The use of prestige network is initially motivated by attempting to capture the "cheap talk" strategic behaviour observed in Archak's study [4]. The basic idea is that given the initial matched developers $D = \{d_1, d_2, ..., d_n\}$. Based on the content features, d_i is more suitable compared with the developers ranked after d_i. For d_{i+1} to d_n those who are deterred by d_i should be removed from the matching list. This aims to incorporate both developers' task content preference and their strategic behaviour in choosing the task: For example, developer A and B are both high skilled developers and share the similar task preference. For a particular task t, when developer A is slightly more suitable than B and B has been defeated by A before, B may move to another task in order to reduce the risk in winning the prize, despite his preference on the task content of t.

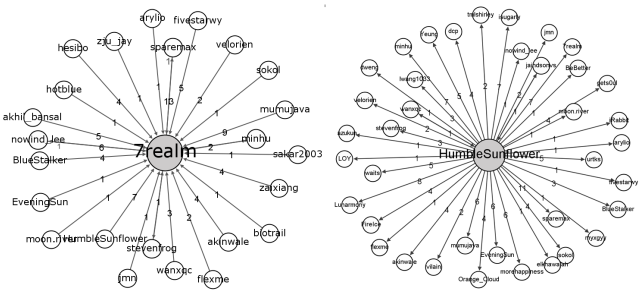

Figure 4: A real case of "winner" (left) and "participant" (right) developers.

Observation: Through the prestige network drawn from empirical data, we observed that, on one hand, there are fewer edges between the nodes with higher in-degree, which can be attributed to the "cheap talk" phenomenon among high skilled developers.

On the other hand, we can also see that some developers are frequently defeated by their opponent developers. If we suppose the "cheap talk" effect is a general rule, then we can infer the developers should learn to avoid the opponent developers who have defeated them after accumulating certain failure experience. However according to our observation on the empirical data, this is not always the case. We find that for some developers, they would actively challenge their winners even if they have continuously defeated by them. The reason can be attributed to various incentives of the developers. Perhaps for some of them, they enjoy the competition process and choose the tasks they like regardless of the competition results or simply share the similar task preferences with the winners.

Formulation: Although some developers actively and extensively choose their tasks in the CSD platform, they seldomly win a task. That is, those developers are frequently defeated by other developers. We call these developers who have a broad interest in the tasks and put a low priority on competition results as "participant-developers". Likewise, for those who constantly defeat others and may strategically choose their opponents, we call them as "winner-developers". An example of the "participant-developer" and "winner-developer" is given in Figure 4.

Since the "participant-developers" can constantly challenge some other developers, we can model this as an attraction force between the graph nodes. Also, the deterrence effect among the "winner-developers" can be modelled as repulsion force. Basically, we employ the energy model [24] in graph theory to capture the attraction and deterrence effects. The attraction force is calculated according to the co-occurrence times of two developers. While the repulsion force is obtained regarding the ratio of non co-occurrence times and co-occurrence times of two developers. The attraction force and the repulsion force between two nodes are calculated according to Equation 3 and Equation 4, where *deg* means the degree of a node and *edg* indicates the edge numbers between two nodes.

$$attraction(v_i, v_j) = edg(v_i, v_j) \tag{3}$$

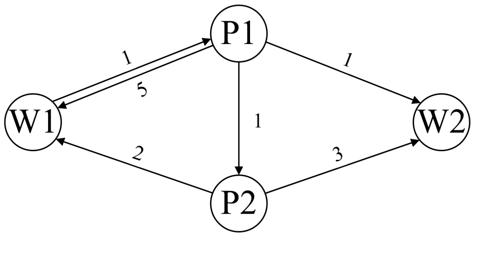

Figure 5: An example prestige network.

$$repulsion(v_i, v_j) = \frac{(deg(v_i)+1)*(deg(v_j)+1)}{edg(v_i, v_j)+1} \qquad (4)$$

Example: An example prestige network is shown in Figure 5, where each node stands for a developer and the weight of the edge indicates the number of duplicated edges. In this example, "winner developers" $W1$ and $W2$ have the strongest repulsion force which is calculated as: $(6+1)*(4+1)/(0+1) = 12$. The strongest attraction force exists between the "participant-developer" $P1$ and the "winner-developer" $W1$, which is 6 resulted by summing their edge numbers up.

3.2.2 Matching Result Refinement

Based on the proposed attraction and repulsion measures of the prestige network, we propose the algorithm to refine the initial matching result obtained from CBIM. This procedure is illustrated by Algorithm 2.

The algorithm mainly consists of 5 phases:

1) Force calculation: Line 2 to line 4 calculate the repulsion and attraction forces between each node pair in the prestige network.

2) Candidate preparation: Line 5 to line 7 prepare the deterred and attracted candidate list for each node. For node $v_i \in V$, those nodes with the largest repulsion and attraction force with v_i are added to the candidate lists $R(v_i)$ and $A(v_i)$ separately. Note that α and β indicate the corresponding size of $R(v_i)$ and $A(v_i)$, which can be adjusted.

3) Exclusion operation: Line 8 to line 11 exclude the deterred candidates from the initial matching result. For developer $d_i \in D$, those developers ranked after d_i and belong to $R(d_i)$ are excluded.

4) Inclusion operation: Line 12 to line 13 include the attracted candidates. For developer $d_i \in D$, developers d_i and $A(d_i)$ are added to D', preserving the insertion order.

5) Redundancy removal: Line 14 to line 15 remove those redundancy candidates ranked after the n^{th} position. The left n matched developers in D' form the enhanced matching result which is expected be returned.

Algorithm 2: Refining Matching Result via Prestige Network

```
Input: The Prestige network $G = (V, E)$ and the top $n$
       developers $D = \{d_1, d_2, ..., d_n\}$ matched to task $t$
       generated by learner $m$
Output: Matched developers $D' = \{d_1', d_2', ..., d_n'\}$
1  $D' \leftarrow \emptyset$
2  for each node pair $(v_i, v_j)$, where $i \neq j$, in $V$ do
3      $repulsion(v_i, v_j) \leftarrow \frac{(deg(v_i)+1)*(deg(v_j)+1)}{(edg(v_i,v_j)+1)}$
4      $attraction(v_i, v_j) \leftarrow edg(v_i, v_j)$
5  for each node $v_i \in V$ do
6      calculate repulsed node set $R(v_i) \leftarrow \{r_1, r_2, ..., r_\alpha\}$,
       satisfying $max \sum_{j=1}^{\alpha} repulsion(v_i, r_j)$
7      calculate attracted node set $A(v_i) \leftarrow \{a_1, a_2, ..., a_\beta\}$,
       satisfying $max \sum_{j=1}^{\beta} attraction(v_i, a_j)$
8  for each developer $d_i \in D$ do
9      for $j \leftarrow i+1$ to $size(D)$ do
10         if $d_j \in R(d_i)$ then
11             remove $d_j$ from $D$
12 for each developer $d_i \in D$ do
13     add $d_i$ and $A(d_i)$ to $D'$
14 for $i \leftarrow n+1$ to $size(D')$ do
15     remove $d_i$ from $D'$
16 return $D'$
```

4. EVALUATION

This section presents the empirical study conducted on 3 datasets with different development characteristics to evaluate our PREM approach. Since there is no existing benchmark for this emerging application context, we first compare PREM with a statistical based naïve method named ACTIVE, which matches the most statistical top winners to new arriving tasks. Since this method is simple and actionable, if we cannot outperform this method then there is no reason for current CSD platforms to adopt our PREM method. Further, we compare PREM with the Content Based Matching (CBM) approach which derives from PREM by removing the PNBE part. Note that, in this paper we regard CBIM as a component of PREM and view CBM as an individual approach. Lastly, we implement an alternative version of PREM related to the essential exclusion and inclusion operations, and compare its performance with the original one's.

With these experiments, we aim to answer the following 3 questions:

Q1: How is the performance of PREM?

Q2: Is the performance of PREM sensitive to the adopted leaner and the dataset?

Q3: Does the execution order of the exclusion and inclusion operations in PREM affect its performance?

4.1 Dataset

We evaluate our approach on the datasets collected from TopCoder platform, which now has the largest community for crowdsourced software development. We totally collected 9,191 historical tasks that have been crowdsourced from Oct. 9th 2003 to Apr. 9th 2013, which cover 3 types of software development tasks on the platform, i.e. component de-

Table 3: Dataset Statistics

Dataset	# Tasks	# Win.	Duration
Development	1093/1367	92/298	2003.10-2013.02
Assembly	1505/1727	86/211	2008.11-2013.03
Bug	5599/6097	202/484	2008.01-2013.04

velopment (DEV), assembly (ASM) and bug fixing (BUG) tasks. We evaluate our approach separately on each of the 3 datasets. The statistics of the 3 datasets after and before data filtering (see Section 3.1) is shown in Table 3. Note that the 3^{th} column indicates the number of distinct winners.

4.2 Study Setting

To simulate the real practice, we sort the task records ascendingly according to their posted time and then divide each of the 3 datasets into 10 folds. We use 9 folds of records as our training set and the remaining fold as our testing set, i.e. new arriving tasks. On each of these datasets, PREM matches top 2, 3, 4 and 5 developers to a specific task separately and record the matched developers for evaluation. When PREM refines the initial CBIM matching result, we set the inclusion and exclusion candidates size to 1, i.e., $\alpha = 1$ and $\beta = 1$ in the algorithm. The machine learning algorithms evaluated in our study include C4.5 Decision Tree (C4.5), Random Forest (RF), Naïve Bayes (NB) and K-Nearest Neighbour (KNN). The corresponding parameters are set according to the default parameters in the popular open sourced data mining tool Weka [12].

4.3 Evaluation Metrics

In order to evaluate the performance of our proposed approach, we need the metrics which are popular in developer recommendation techniques. Since previous related research topics such as bug triage mainly focus on the *Accuracy* dimension of the performance [7,15,26], we adopt this metric and it is defined formally as shown in Equation 5, where $D(t)$ stands for the matched developers for task t and T stands for testing set. $\left|correct\big(D(t)\big)\right| = 1$ when the ground truth developer is included in $D(t)$.

$$Accuracy = \frac{1}{|T|} * \sum_{t \in T} \left|correct\big(D(t)\big)\right| \qquad (5)$$

4.4 Results and Analysis

We present the empirical evaluation results of the experiments as described above: Table 4 shows the performance comparison between PREM and ACTIVE. Table 5 compares the accuracy of PREM with CBM which does not consider prestige network information, and Table 6 compares the accuracy of PREM with its alternative version, PREM'. All numbers in these tables are in percentage. The analysis of these results are as follows.

4.4.1 Results for Q1

Overall, PREM is able to achieve reasonable accuracy. The highest accuracies are 63%, 46%, 36% for the DEV, ASM and BUG datasets separately, when PREM adopts C4.5 learner and matches 5 developers to each task.

Table 4: Accuracy of PREM and Statistical ACTIVE Matching Results

Dataset	Top	PREM				ACTIVE
		C4.5	RF	NB	KNN	
DEV	**2**	42.2	41.3	38.5	**48.6**	24.6
	3	53.2	**54.1**	44.0	49.5	35.6
	4	**58.7**	56.9	48.6	54.1	37.6
	5	**63.3**	61.5	53.2	57.8	37.6
ASM	**2**	**30.7**	29.3	22.7	28.7	10.0
	3	35.3	**35.3**	26.7	34.0	10.7
	4	**43.3**	38.0	30.7	38.0	15.3
	5	**46.0**	42.0	32.0	38.0	19.3
BUG	**2**	**27.0**	25.6	26.5	20.8	14.5
	3	**29.2**	28.4	28.3	22.2	14.5
	4	**32.7**	30.6	31.7	28.1	14.5
	5	**36.3**	33.6	32.9	29.2	17.5

Also, it can significantly outperform the statistical ACTIVE method and the CBM method in which the prestige network is not taken into consideration.

Compared with ACTIVE: In Table 4, the bold numbers imply the best accuracy in the corresponding rows. It is immediately clear that PREM outperforms the statistical ACTIVE method across all datasets with different settings, assessed from the *Accuracy* dimension.

Compared with CBM: In Table 5, the last row indicates the average improvement when compare PREM with CBM. Across all 48 pairs of comparison under different experimental settings, there are only 2 cases that PREM are slightly worse than CBM (i.e., the ASM-Top5-NaïveBayes case and the ASM-Top5-KNN case).

4.4.2 Results for Q2

Through the experimental results in Table 5, we find the performance of PREM is sensitive to both the adopted learner and the dataset.

We can see that the tree-base learners, C4.5 and Random Forest perform the best in 9 of 12 and 2 of 12 cases, separately. They can be regarded as the learners that can best support PREM. Note that without PNBE, these two tree-based methods can still achieve relatively high accuracy compared with other learners. When accessing from the perspective of the improvement, the KNN leaner has the best improvement provided PNBE, the best improvement can be as much as 22% (the DEV-Top2-KNN case). Also, the C4.5 learner has reasonable improvement with PNBE. After manually checking all the candidate developers matched by different learners, we find that the improvement has a positive correlation with the diversity of the learner generated results. The results are more diverse when they contain more unique developers which are matched by the learner on the test dataset. The reason for this outcome is that the datasets are unbalanced, which means a small number of "winner-developers" win a majority of tasks. For the more diverse results, these "winner-developers" have a higher possibility to be included into the refined results through the

Table 5: Accuracy of CBM and PREM Matching Results

Dataset	**Top**	**C4.5**		**RandomForest**		**NaïveBayes**		**KNN**	
		CBM	PREM	CBM	PREM	CBM	PREM	CBM	PREM
DEV	**2**	33.9	42.2	30.3	41.3	30.3	38.5	26.6	**48.6**
	3	41.3	53.2	40.4	**54.1**	35.8	44.0	30.3	49.5
	4	46.8	**58.7**	46.8	56.9	39.4	48.6	33.9	54.1
	5	51.4	**63.3**	54.1	61.5	45.9	53.2	38.5	57.8
ASM	**2**	30.0	**30.7**	29.3	29.3	21.3	22.7	28.0	28.7
	3	33.3	35.3	34.0	**35.3**	24.7	26.7	31.3	34.0
	4	36.7	**43.3**	36.7	38.0	26.7	30.7	33.3	38.0
	5	38.0	**46.0**	38.0	42.0	*33.3*	*32.0*	*38.7*	*38.0*
BUG	**2**	24.7	**27.0**	22.2	25.6	25.4	26.5	18.1	20.8
	3	27.7	**29.2**	25.2	28.4	27.0	28.3	19.7	22.2
	4	29.7	**32.7**	27.7	30.6	27.5	31.7	19.9	28.1
	5	29.9	**36.3**	31.8	33.6	27.7	32.9	19.9	29.2
Average Improvement		+6.22%		+5.01%		+4.22%		+9.23%	

Table 6: Accuracy of PREM and PREM' Matching Results

Dataset	**Top**	**C4.5**		**RandomForest**		**NaïveBayes**		**KNN**	
		PREM	PREM'	PREM	PREM'	PREM	PREM'	PREM	PREM'
DEV	**2**	42.2	45.9	41.3	41.3	38.5	35.8	48.6	45.9
	3	53.2	51.4	54.1	54.1	44.0	41.3	49.5	51.4
	4	58.7	57.8	56.9	58.7	48.6	45.9	54.1	57.8
	5	63.3	63.3	61.5	64.2	53.2	50.5	57.8	63.3
ASM	**2**	30.7	30.0	29.3	27.3	22.7	22.7	28.7	28.0
	3	35.3	34.7	35.3	35.3	26.7	26.7	34.0	31.3
	4	43.3	43.3	38.0	38.0	30.7	31.3	38.0	36.0
	5	46.0	44.7	42.0	41.3	32.0	32.7	38.0	38.0
BUG	**2**	27.0	27.0	25.6	25.8	26.5	26.5	20.8	20.6
	3	29.2	29.2	28.4	27.9	28.3	28.3	22.2	22.0
	4	32.7	30.1	30.6	30.8	31.7	30.2	28.1	27.2
	5	36.3	33.3	33.6	32.7	32.9	30.8	29.2	27.5
P-value (2-tailed)		0.11410		0.96012		0.06148		0.59612	

inclusion operation, which increases the chance in matching the ground truth winners.

Based on above findings, we conclude that PREM is more effective well when:

- The learner itself performs better on the dataset.
- The learner itself generates more diverse results on the dataset.

Also, we find that on ASM and BUG datasets, PREM has relatively poorer improvement than on the DEV dataset. This can be attributed to the characteristics of the datasets. On ASM and BUG datasets, we observed that their corresponding prestige networks are more uniform however the prestige network of the DEV dataset exhibits higher local density. This outcome indicates that on ASM and BUG datasets, the identified strategic behaviour of the "participant-developers" and "winner-developers" are not as typical as the DEV dataset does.

The repulsion and attraction force levels can be different on these datasets. For instance, the repulsion force level can be higher on DEV dataset as its tasks require specific skills. Thus those who have the expertise can easily deter others without relevant skills. While on the ASM and BUG datasets, the skill requirements are more general and the attraction force level can be higher as more developers have the capacity to perform the tasks. Since parameters regarding the attraction and repulsion force levels (i.e., α and β) are predefined in PREM, they may need to be tuned for the ASM and BUG datasets. We infer that PREM performs less effective when the dataset involves less competition nature which can be captured by PREM.

4.4.3 Results for Q3

PREM refines the developer matching result essentially by the exclusion and inclusion operations illustrated in Algorithm 2. In the alternative implementation PREM', we switch the execution order of these two operations, i.e., attracted developers are included first, then the deterred developers are removed. The performance comparison is shown in Table 6.

We employ the Wilcoxon non-parametric statistical hypothesis test to assess whether there is any discernible differences between the performances of the two implementations. We choose Wilcoxon test as it compares the sums of ranks and it is more robust than the Student's t-test. Also it does not require the samples come from normal distributions. In the analysis we set the confidence level to 95% and we calculate the 2-tailed P-values. The result is shown in the last row of Table 6. We can see that for all learners, the corresponding P-values are larger than 0.05. Thus the Wilcoxon test gives no evidence against the null hypothesis that there is no significant differences between the two performance distributions.

So the answer for Q3 is that the execution order of the exclusion and inclusion operations in PREM has no significant effect on its performance. Through the intermediate experimental results we find that both operations contribute to a higher accuracy for PREM.

5. DISCUSSION

In this section, we discuss the threats to validity of this study, the accuracy-diversity dilemma of developer-task matching approaches, our finding on supporting developers in making decision at the late registration stage and our supporting tool.

5.1 Threats to Validity

Internal threats: Threats to internal validity stem from potential factors affect the dependent variables without our knowledge. It is possible there are flaws in the implementation of our techniques which could have affected the results. To reduce this threat, we employed the open sourced WEKA machine learner package to support our PREM approach and manually check the intermediate results generated by our approach.

External threats: Threats to external validity arise when the experimental results are unable to be generalized. First, although our experiments are conducted on large datasets with 3 different application contexts, it is collected from one CSD platform (TopCoder). It is possible that the competition relationship among developers can differ from one CSD platform to another. Thus PREM should be re-evaluated when applied to other CSD platforms. Second, we evaluate PREM with limited number of learners. There are more sophisticated NLP techniques that can be adopted to support PREM, however we do not know their corresponding performances yet. Third, the parameters used in machine learners and in PREM (i.e., α and β) are not tuned in our experiments. Their performances can be optimized with different parameter settings. We leave the further evaluations on various CSD platforms, more machine learners and parameter tuning as future work of this paper.

Construct threats: Threats to construct threats occur when the metrics used fail to capture the concepts they meant to evaluate. We assess the effectiveness of PREM by the accuracy measure which is popular in related studies. However in the context of CSD, although a highly accurate matching result may contribute to the delivery of qualified assets, it does not necessarily encourage more participants, which may harm the community ecology for a long term. Another potential dimension should considered here is the diversity metric which is often used in the recommender system area. Since accuracy and diversity are usually conflicting objectives, there exists an accuracy-diversity dilemma [29]. To compensate for this construct threat, we additionally discuss the accuracy-diversity dilemma and propose a way to trade-off between the two objectives.

5.2 The Accuracy-Diversity Dilemma

As shown in the experimental results, PREM has the capacity in improving the developer-task matching accuracy, however over-stressing the importance of accuracy merely brings short-term benefit for the CSD platform. It can lead to the CSD community dominated by only a small number of developers as time pass by. Only by achieving reasonable diversity, can the system serve for the "long-tail" developers and enhance the participation level (i.e., exploration). Meanwhile, the system can match certain number of most suitable developers to ensure that the accuracy is acceptable (i.e., exploitation). In this way the matching system works for long-term benefit.

Here we present a way to make PREM work in the exploitation and exploration way. The idea can be briefly described as follows:
1) Set $x\%$ candidate slots as exploitation.
2) Set $1 - x\%$ candidate slots as exploration.

This idea can be implemented by ensembling various learners employed in our PREM matching system. For instance, according to the experimental results, the C4.5 learner indicates good capability in matching accuracy, thus can be viewed as an exploitation learner. While the KNN learner shows relatively superior ability in matching diversity, thus can be regarded as an exploration learner. We combine the matched developers generated from these two learners and select top N developers according to the weights assigned to accuracy and diversity.

The optimization goal vary in different contexts, here we define the optimization target in the form of Weighted Harmonic Mean [11]. We assign the weight β^2 to Accuracy and the weight 1 to diversity. H_β is calculated according to Equation 6.

$$H_\beta = \left(1 + \beta^2\right) * \frac{Accuracy * Diversity}{\beta^2 * Accuracy + Diversity} \qquad (6)$$

Then we calculate $H_{0.5}$ (emphasize on accuracy), H_1(equal emphasis) and H_2 (emphasize on diversity). Figure 6 shows the trade-off performance of the combination based approach. In this figure the exploitation learner C4.5 and the exploration learner KNN are used. Scatter points correspond to different proportion of developers selected from the two learners' results. For example, as for "Top 5" case, the proportion is from 100% C4.5 results to 100% KNN results, the 4 middle points stand for the combined results (with the proportion of 4:1, 3:2, 2:3 and 1:4). The triangle symbol indicates an optimal point for the corresponding β value. From the results we can see that the trade-off approach can be superior than the single learner supported PREM. For example, in the "Top 20" case, the H_1 optimal point shows that the method only **sacrificed 3% in accuracy while gained 50% in diversity**.

5.3 Late Registration Stage Strategy

As a complement to PREM which works prior to or at the early registration stage, we present our finding to support developers strategic decisions at the later registration stage,

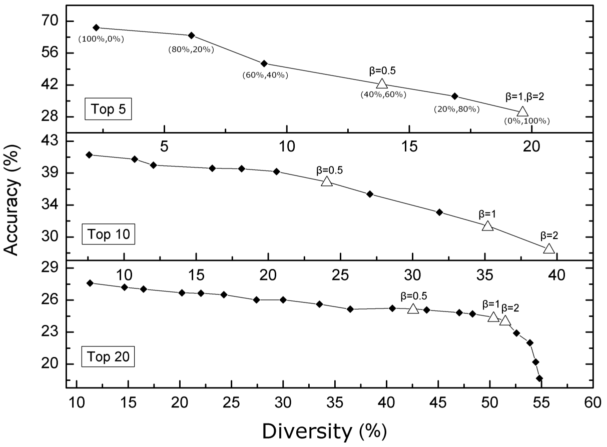

Figure 6: Performance of the combination based trade-off method on Assembly dataset.

especially for those developers who desire to win the task. At the late registration stage, we have plenty of information about the adversary thus can support a developer's strategic decision (e.g., register other tasks than develop a solution for a task in which their opponents dominant).

Besides the prestige network, we find that the "rating" and "reliability" information provided by TopCoder platform can be good indicators in predicting the final winner of the task at the late registration stage. We evaluate a variety of methods on the development dataset and find the heuristic rule by considering "rating" and "reliability" even performs better than the machine learners which achieves an accuracy as high as 72%. The utility function we employed to estimate the winner are simply defined as $f = rating * (1.0 + reliability)$. We present this winner indicator and find it useful in supporting online developers' decision making at the late registration stage.

5.4 An Online Implementation of PREM

In order to support our proposed approach, we have implemented a user friendly tool called "CrowdDev" platform[3] and deployed it on the Internet. This platform is designed to enable crowd developers view the collected active open call requests from various CSD platforms, and help them to find their most suitable tasks via PREM. Also, developer's personal prestige network is provided in order to help the developer better choose their opponents. At current stage, CrowdDev only supports TopCoder platform.

6. RELATED WORK

Although the study on GSD commenced a decade ago [13], there are few research work have been done on related topics for the emerging CSD paradigm. We could not find any prior work on developer-task matching in the CSD context. For recommending crowdsourced micro-tasks to users, Ambati, Vogel and Carbonell [1] proposed a recommendation approach based on implicit modelling of interest and skills. Yuen, King and Leung [28] employed probabilistic matrix factorization for preference-based task recommendation in crowdsourcing system. These two studies were conducted on Amazon Mechanical Turk platform with micro-tasks, rather than tasks as complex as software development.

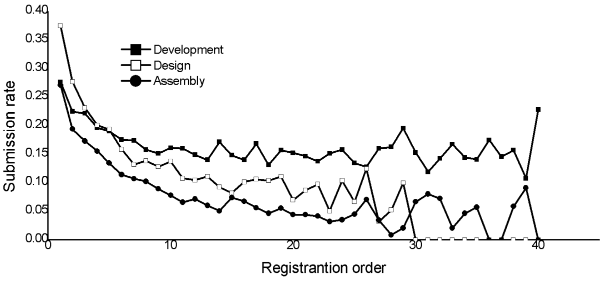

Figure 7: The relationship between developers' registration order and submission rate.

Another related topic which has been studies more extensively is developer recommendation for development-oriented tasks [3]. Čubranić and Murphy [30] proposed to build developer recommenders for automatic bug triage using text categorization techniques. Anvik, Hiew and Murphy [2] expanded the previous work by data pre-processing, using additional features and exploring the performances of more machine learning algorithms. Gaeul, Kim and Thomas [16] further improved developer recommenders for bug triage by using tossing graph based on developers tossing behaviours in bug repositories. Matter, Kuhn and Niestrasz [23] presented their approach to automatically suggest developers for handling bug reports using a vocabulary-based expertise model. Xuan et al. [27] leveraged developer prioritization based on a social network techniques to assist triage tasks in bug repositories.

However these papers are more about developer collaboration. The proposed approaches need to be carefully reexamined in the crowdsourced software development context due to its competition nature and several other new features. For instance, one prior research has shown that traditional laws on software cost are challenged in the CSD context [21]. Likewise, we consider traditional developer recommendation methods cannot be migrated to this new context directly as previous bug triage studies are based on open source community which is collaboration oriented. While crowdsourcing is competition oriented and the developers have more complicated relationships.

Nikolay Archak [4] studied the developers' strategic behaviour on TopCoder platform. The "cheap talk" phenomenon is observed, which indicates the high rated developer would register the task early in order to deter their opponents' entry. One of our analysis also agrees with this study. As Figure 7 shows, the early registered developers have a higher possibility to make a submission, e.g., the late registered developers may be deterred by the early ones. However, we argue the deterrence would not always work. For some developers, they would insist on competing with high rated developers even though they have been defeated by them constantly. We identified two typical types of developers evolved in the competition, which have been discussed in Section IV.

[3] Our CrowdDev Platform: http://www.CrowdDev.org

7. CONCLUSIONS

Crowdsourced software development is an emerging paradigm which utilize "wisdom of the crowd" for software production. Crowdsourced software development tasks demand reliable developers to grantee a qualified asset that can be delivered to the client. Current crowdsourcing practice usually adopt a pull methodology which cannot grantee that developers work on their most suitable tasks. In addition it brings the developers much burden in finding their suitable tasks from the overloaded task information.

To tackle above challenges, in this paper we proposed an approach named PREM to actively match the best suitable developers to the available tasks. A set of task features were proposed, which can be used to build content based matching models using existing machine learners. In order to enhance the matching result, we further considered the competition nature of crowdsourcing. We identified two typical types of developers in CSD and their task selection behaviour were considered in our constructed prestige network. The experimental results show that PREM can achieve reasonable accuracy and significantly outperform the statistical and CBM methods. To cope with the accuracy-diversity dilemma of the matching results, we further proposed a combination method to adjust our matching results. We also implemented an online platform to support our approach, which can be publicly accessed by the crowd developers.

For future work, we intend to propose an improved approach to better model the social relationship among developers. Multi-criteria optimization such as the accuracy-diversity dilemma will be considered when searching for the optimal matching results.

8. REFERENCES

[1] V. Ambati, S. Vogel, and J. G. Carbonell. Towards task recommendation in micro-task markets. In *Human Computation*, volume WS-11-11 of *AAAI Workshops*. AAAI, 2011.

[2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, pages 361–370. ACM, 2006.

[3] J. Anvik and G. C. Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(3):10, 2011.

[4] N. Archak. Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on topcoder.com. In M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, editors, *WWW*, pages 21–30. ACM, 2010.

[5] A. Begel, J. Bosch, and M.-A. D. Storey. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *IEEE Software*, 30(1):52–66, 2013.

[6] A. Begel, R. DeLine, and T. Zimmermann. Social media for software engineering. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 33–38. ACM, 2010.

[7] P. Bhattacharya and I. Neamtiu. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

[8] L. B. Chilton, J. J. Horton, R. C. Miller, and S. Azenkot. Task search in a human computation market. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 1–9, New York, NY, USA, 2010. ACM.

[9] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[10] J. Farrell and M. Rabin. Cheap talk. *The Journal of Economic Perspectives*, 10(3):103–118, 1996.

[11] W. F. Ferger. The nature and use of the harmonic mean. *Journal of the American Statistical Association*, 26(173):36–40, 1931.

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[13] J. D. Herbsleb and D. Moitra. Global software development. *Software, IEEE*, 18(2):16–20, 2001.

[14] J. Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006.

[15] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 111–120. ACM, 2009.

[16] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 111–120, New York, NY, USA, 2009. ACM.

[17] K. Lakhani, D. Garvin, and E. Lonstein. Topcoder (a): Developing software through crowdsourcing. *Harvard Business School General Management Unit case*, (610-032), 2010.

[18] P. Langley, W. Iba, and K. Thompson. An analysis of bayesian classifiers. In *In Proceedings of the 10th International Conference on Artificial Intelligence*, pages 223–228. MIT Press, 1992.

[19] K. Li, J. Xiao, Y. Wang, and Q. Wang. Analysis of the key factors for software quality in crowdsourcing development: An empirical study on topcoder.com. In *COMPSAC*. IEEE Computer Society, 2013.

[20] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[21] K. Mao, Y. Yang, M. Li, and M. Harman. Pricing crowdsourcing-based software development tasks. In *Proceedings of the 2013 International Conference on Software Engineering, New Ideas and Emerging Results Track*, ICSE 2013, pages 1205–1208. IEEE Press, 2013.

[22] Massolution. Crowdsourcing industry report. 2012.

[23] D. Matter, A. Kuhn, and O. Nierstrasz. Assigning bug reports using a vocabulary-based expertise model of developers. In *Proceedings of the 2009 6th IEEE*

International Working Conference on Mining Software Repositories, MSR '09, pages 131–140, Washington, DC, USA, 2009. IEEE Computer Society.

[24] A. Noack. Energy models for graph clustering. *J. Graph Algorithms Appl.*, 11(2):453–480, 2007.

[25] J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[26] J. Xuan, H. Jiang, Z. Ren, and W. Zou. Developer prioritization in bug repositories. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 25–35. IEEE, 2012.

[27] J. Xuan, H. Jiang, Z. Ren, and W. Zou. Developer prioritization in bug repositories. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 25–35, Piscataway, NJ, USA, 2012. IEEE Press.

[28] M.-C. Yuen, I. King, and K.-S. Leung. Taskrec: probabilistic matrix factorization in task recommendation in crowdsourcing systems. In *Neural Information Processing*, pages 516–525. Springer, 2012.

[29] T. Zhou, Z. Kuscsik, J.-G. Liu, M. Medo, J. R. Wakeling, and Y.-C. Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

[30] D. ÄŇubraniÄĞ. Automatic bug triage using text categorization. In *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering*, pages 92–97.
KSI Press, 2004.