Abstract—In crowdsourced software development, routing a task to right developers is a critical issue that largely affects the productivity and quality of software. In particular, crowdsourced software development platforms (e.g., Topcoder and Kaggle) usually adopt the competition-based crowdsourcing model. Given an incoming task, most of existing efforts focus on using the historical data to learn the probability that a developer may take the task and recommending developers accordingly. However, existing work ignores the locality characteristics of the developer-task dataset and the competition among developers. In this work, we propose a novel recommendation approach for task routing in competitive crowdsourced software development. First, we cluster tasks on the basis of content similarity. Second, for a given task, with the most similar task cluster, we utilize machine learning based classification to recommend a list of candidate developers. Third, we consider the competitive relationship among developers and re-rank the candidates by incorporating the competition network among them. Experiments conducted on 3 datasets (totally 7,481 tasks) crawled from Topcoder show that our approach delivers promising recommendation accuracy and outperforms the two comparing methods by 5.5% and 25.4% on average respectively.

Index Terms—Crowdsourcing, recommender systems, Topcoder.

I. INTRODUCTION

In recent years, the success of Crowdsourced Software Development (CSD) has been widely witnessed [1], which has attracted geographically distributed online developers to accomplish software development tasks in the form of open calls. Compared with tradition software development, CSD has the advantages of lower-costs, faster time-to-market, creativity and open innovation [2]. As one of the largest CSD platform, Topcoder [3] has over 1 million registered developers, 7K challenges published per year. Even well-known companies such as Google, NASA and Facebook have become Topcoder’s customers.

To achieve wide task accessibility and self-selection, most CSD platforms allow crowd developers to choose tasks freely based on their personal skills, experience, and interests. This free selection leads to two different issues: 1) from the developers’ perspective; there are many available tasks posted on the CSD platform, so it is very time consuming for developers to choose the “right” tasks manually; 2) from the requesters’ perspective: to acquire high-quality deliverable is the ultimate goal, but it is challenging to select the suitable developers for the tasks. Although most CSD platforms have the peer-review phase to ensure the quality of the deliverable, there are still many tasks without satisfying submission. Take Topcoder as an example, we collect tasks from December 2006 to October 2016, among these tasks there is approximately 87.4% quitting rate (410,968 out of 470,252 tasks without submission), 35.4% failed review rate (21,003 out of 59,284 submissions failed to pass review). So it is of great importance to match tasks with “right” developers.

Existing studies [4]–[10] have tried to solve this problem from three perspectives: the developers, the platforms and the requesters. These studies include task pricing [4], [5], analysis of the key factors for software quality in crowdsourcing [6], developers’ behaviors in competitive crowdsourcing [7]–[10], and content-based developers recommendation [11] for CSD tasks. In short, existing studies have put forward many factors that may influence the developers’ behavior such as developers’ skills and interests, tasks’ content and competition relationship. But none of them fully consider these incentives for developers recommendation. Besides, several studies [12], [13] show that local bias is a common phenomenon in software development datasets and it may cause negative impacts on model performance. Thus, local bias should also be taken into consideration for developers recommendation.

To supply a better solution for task routing problem, we propose a novel approach named cluster-based classification and competitive network boosting (CBC-CN) which recommends the suitable developers for tasks. The main idea of our approach is to select the most similar tasks and build classifier based on these similar tasks to recommend a list of candidates, then the competitive relationship among them is utilized to refine the initial result.

We conducted several experiments to evaluate the performance of our approach and compare it with two baselines using content-based matching (CBM) [11] and content-based recommendation algorithm (CBR). The results show that: 1) our approach outperforms the baselines on 3 types of datasets; 2) both cluster-based classification and competitive relationship can help improve the accuracy of recommendation. The major contributions of this work are as follows:

• We propose a novel approach called CBC-CN to solve the task-developer matching problem in CSD context. To the best of our knowledge, this is the first effort to incorporate competition relationship into the recommendation of crowdsourced software developers.
• We employ a cluster-based classification method which consider the task’ content similarity to handle the local characteristic in CSD tasks. Besides, we utilize developers’ competitive relationship to refine the recommended result.

• We evaluate our approach against representative methods on 3 datasets containing 7,481 tasks from Topcoder. The results show our method outperform the 2 methods by 5.5% and 25.4% respectively on accuracy.

The rest of this paper is organized as follows. Section 2 describes the background of CSD. In Section 3, we introduce our approach to recommend developers for CSD tasks. The evaluation of our work is presented in Section 4. In Section 5, we analysis the results. Related work is presented in Section 6. Finally, we conclude this paper and discuss future work in Section 7.

II. BACKGROUND

In this section, we introduce the background of CSD process to help better understand the challenges in CSD context. Since Topcoder is the world’s largest CSD platform with more than 1,000,000 registered developers, we describe the CSD process based on the example of Topcoder.

CLOUD HUB - UPDATE Swagger

Fig. 1. An example of CSD Task on Topcoder

The process of Topcoder is achieved by utilizing an open competition format to attract online developers to make contributions to the multiple types of tasks which include component design, development, code, assembly, first2finish, bug hunt and etc. Figure 1 shows the main attributes of a special CSD task. Each task is organised as an open contest and it has many attributes containing task name, detailed requirement, posting date, registration deadline, submission deadline, payment, platforms, required languages and techniques. Developers are volunteer to register for tasks and submit their solutions before the deadline. After the submission deadline, the submitted solutions are evaluated by peer review process and then scored by three experts according to its quality. Finally, the solution will be ranked by final score and the top developers will win the contest.

In CSD contests, we find a common phenomenon that many tasks with lots of registrants but have very few submissions, and many submissions fail to pass the peer review. In order to match tasks and developers better, we must figure out which factors affect the individual performance. Study [14] shows that multiple types of incentives encourage developers to register for tasks, these incentives include gaining skills, earning money, having fun, getting recognition and etc. Since many incentives [7]–[9] can influence the developers’ decisions, many developers have the untrustworthy behaviors [15]. From the datasets of Topcoder, lots of developers tend to register with more tasks than they can accomplish at the same time, even fail to submit any solutions. So it is obvious that matching tasks with developers is really important in CSD context. In addition, many CSD platforms have the reputation system and developers in a same contest have access to her opponents’ profile which contains some useful information such as winning rate, rating, reliability. Study [16] shows that developers will evaluate the ability of his opponents and behave strategically to choose their opponents to avoid fierce competition. This phenomenon is called “cheap talk” [17] in CSD context, but it is not all cases and there are also some developers who are active in competing with high-rated developers. So if we take the competition nature of crowdsourcing into full consideration, we can utilize this competitive relationship for developer-task matching.

In short, many incentives are related to the developers recommendation in crowdsourcing. Existing studies only consider the content of task and ignore the competitive relationship among developers. To supply a better solution, we propose a novel approach which incorporates competition relationship into the recommendation of crowdsourced software developers.

III. APPROACH

In CSD context, there exists the “local mode” phenomenon that the distributions of data are often different on different parts of the datasets. It is obvious that crowdsourced tasks have the local characteristic. Software development often contains many similar techniques and platforms. Many experiments [18] reveal that the local mode is superior to the global model.

To recommend developers for tasks on CSD platforms, most existing approaches mainly focus on the content of a task and recommend developers whose skills are best matched with the requirements of the task, but they seldom consider the different characteristics of tasks and the competitive relationship among developers. As a result, these approaches can not work well if the recommended developers behave strategically when facing fierce competition. In order to match tasks and developers better, we propose a novel approach called CBC-CN which mainly considers the task content similarity to handle the local characteristic in CSD tasks. Besides, we utilize developers’ competitive relationship to refine the recommended result.

33
contents. Secondly, we cluster these similar tasks together based on their content similarity. Then we build classifier on the most similar tasks and recommend a list of preliminary candidate developers according to their winning probability. Note that we treat the recommendation as a classification problem and each task is labeled with the corresponding winner. Thirdly, we construct the competitive network from developers’ historical activities. Finally we re-rank the initial order of recommended developers based on the competitive network.

A. Data Filtering and Feature Extraction

In order to ensure the quality of dataset, data filtering is done first to remove tasks with incomplete information and unskilled developers. In detail, we filter those historical data according to the following criteria:

1) Tasks with incomplete information, e.g. tasks without submission and missing requirement.

2) Irrelevant tasks. We only keep tasks related to software development including development, code, assembly, bug hunt, etc.

3) Unskilled developers. We treat developers who have won at least 5 tasks as skilled developers, and keep the corresponding tasks.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Data Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td>name of a task</td>
</tr>
<tr>
<td>requirement</td>
<td>string</td>
<td>the detail requirement of a task</td>
</tr>
<tr>
<td>posting date</td>
<td>numeric</td>
<td>the time when a task is posted</td>
</tr>
<tr>
<td>duration</td>
<td>numeric</td>
<td>time allocated to a task</td>
</tr>
<tr>
<td>award</td>
<td>numeric</td>
<td>the prize of a task</td>
</tr>
<tr>
<td>technique</td>
<td>string</td>
<td>the techniques that a task requires</td>
</tr>
<tr>
<td>platform</td>
<td>string</td>
<td>the platform on which a task runs</td>
</tr>
</tbody>
</table>

Next, two types of features are extracted from these selected tasks. The text features consist of title, requirement, techniques, platforms while the numeric features contain posting date, duration and award. The detailed features are shown in Table I. In order to unify two types of features, we convert the text features into word vectors and each word represents a feature. As for title and requirement, we conduct word segmentation first and remove the stop words (e.g. ’am’, ’and’, ’this’ and etc), then the remaining token is calculated by TF-IDF. For techniques and platforms, we do the conversion according to Equation 1, each technique represents a feature.

$$f(x, s) = \begin{cases} 
1 & \text{task } x \text{ requires technique } s \\
0 & \text{otherwise}.
\end{cases}$$ (1)

As for posting date, duration and award, we adopt the normalization technique to convert the original values to values between 0 and 1. The normalization function is defined as Equation 2. Finally, we generate the feature vectors.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}.$$ (2)

B. Clustering Based Classification

In CSD context, there is obvious difference between tasks with different required techniques, platforms and background. Existing approaches usually achieve poor and unstable performance on global dataset. Thus more fine-grained steps should be taken to deal with this local characteristic. We utilize the clustering based classification to cluster the most similar instances together and build classifier on these similar tasks. To select the most similar tasks, we apply the K-Means algorithm to cluster the most similar tasks together. As we mention above, the main difference between tasks are techniques and platforms, thus we apply the cosine similarity to measure the difference of its contents, the distance function adopted in our approach is defined as Equation 3.

$$\text{distance}(t_i, t_j) = 1 - \frac{\sum_{k=1}^{n} x_{i,k}x_{j,k}}{\sqrt{\sum_{k=1}^{n} x_{i,k}^{2}} \sqrt{\sum_{k=1}^{n} x_{j,k}^{2}}},$$ (3)

where $n$ is the total number of techniques and platforms, $x_{i,k}$ indicates whether the $k_{th}$ technique is required in task $t_i$. Based
on this local characteristic, the main steps of clustering based classification are as follows:

- For all training datasets, we extract the features of tasks and put them into vectors. Then we apply K-Means algorithm to cluster these tasks into different clusters based on the their distance defined in Equation 3.
- For each new task, we also extract features based on its contents and compute the distance from each cluster built in previous stage. We select the shortest distance cluster so that the new task is the most similar to the tasks in that cluster.
- Based on these most similar tasks, we utilize classification model to recommend developers for a new task and many supervised learning algorithms such as SVM, Naive Bayes and Decision Tree can be adopted to build classifiers. Finally, the classifier recommends a list of candidate developers according to their winning probabilities.

C. Competitive Network Boosting

The initial candidate developers are recommended from the perspective of content matching and can be refined by Competitive Network Boosting (CNB) for further improvement. The main idea of CNB is to utilize the competitive relationship to better match tasks and developers. The main process of CNB is summarized by Algorithm 1, we discuss it in detail in the following.

**Algorithm 1 Utilizing CNB to refine the initial Result**

**Input:**
- The initial top n developers \( D = \{d_1, d_2, \ldots, d_n\} \)
- The competitive network \( G = (V, E) \)

**Output:**
- Final recommended developers \( D' = \{d'_1, d'_2, \ldots, d'_n\} \)

1: \( D' \leftarrow \emptyset \)
2: for each developer \( d_i \in D \) do
3: \( deg(d_i) = \sum_{j=1}^{n} edge(d_i, d_j) \)
4: end for
5: for each developer \( d_i, d_j \in D \) do
6: \( attraction(d_i, d_j) = \frac{edge(d_i, d_j) - edge(d_j, d_i)}{deg(d_i)} \)
7: end for
8: for each developer \( d_i \in D \) do
9: \( attracter(d_i) = \max_{d_j \in D, d_i \neq d_j} attraction(d_i, d_j) \)
10: if \( d_i \notin D' \) then
11: add \( attracter(d_i) \) and \( d_i \) to \( D' \)
12: end if
13: end for
14: return \( D' \)

To utilize the competition nature of crowdsourcing, we need to construct the competitive network first. We define the competitive network as a directed graph which is constructed as follows: for each task posted before the new task, a directed edge is drawn from each submitter to the winner. An example of competitive network is shown in Figure 3, where each node represents a developer and the weight of the edge indicates the number of winning times. In this example, developer \( d_1 \) and developer \( d_2 \) compete for 12 times, \( d_1 \) wins 10 times and loses 2 times during their competitions.

![Fig. 3. An example of Competitive Network](image)

The utilization of competitive relationship is motivated by strategic behaviour observed in the “cheap talk” phenomenon. Some developers strategically choose their opponent to avoid fierce competition with high skilled developers. But this is not always the case, in the network we also find that some developers actively compete with high skilled developers even though they have been defeated by them many times. Perhaps they share the same interests and enjoy the growth through fierce competition. To take both of the two phenomena into full consideration, we can draw a conclusion: if a developer \( d_i \) has the high probability of winning, the developers who not only often compete with \( d_i \) but also defeat \( d_i \) many times have the higher probability of registering and winning in the same contest too. We define this competitive relationship as attraction in CSD context, the attraction between developers is calculated as Equation 4.

\[
attraction(d_i, d_j) = \frac{edge(d_i, d_j) - edge(d_j, d_i)}{deg(d_i)},
\]

where molecular is the relative times that \( d_i \) wins and denominator is the number of tasks \( d_i \) loses. So the bigger attraction represents that \( d_i \) and \( d_j \) often compete with each other and \( d_j \) has the higher winning probability than \( d_i \).

Based on attraction, we re-rank the order of initial candidates developers by putting a developer’s most attractive opponent who we define as attracter\((d_i)\) (Algorithm 1 line 6) in front of her. The main steps are as follows: from line 2 to line 4, for each developer \( d_i \) we count the number of tasks in which the developer lose. Then, from line 5 to line 7, we calculate the attraction between developers according to Equation 4. Finally, from line 8 to line 13, for each developer \( d_i \) in initial recommend developers, her most attractive opponent attracter\((d_i)\) are added in front of her according the initial order of \( d_i \).

IV. Evaluation

To systematically evaluate our approach, we hope to address the following questions:

- **RQ1:** How is the performance of our approach in developers recommendation on competitive CSD platform?
- **RQ2:** How effective is the cluster based classification compared with traditional classification?
RQ3: How effective is the competitive network for developers recommendation?

A. Data Collection

We evaluated our approach on three datasets crawled through public APIs [19] of Topcoder. We collected approximately 26,000 tasks between December 2006 and October 2016. After data filtering, we select the top three types datasets (Assembly, Code, First2Finish) which contains the largest number of tasks. The detail of the datasets are shown in Table II. The number of distinct winners is shown in the third column. We evaluated our approach separately on each of the 3 datasets.

<table>
<thead>
<tr>
<th>Type</th>
<th>Number of tasks</th>
<th>Number of winners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly</td>
<td>2873</td>
<td>134</td>
</tr>
<tr>
<td>Code</td>
<td>1557</td>
<td>111</td>
</tr>
<tr>
<td>First2Finish</td>
<td>3051</td>
<td>146</td>
</tr>
</tbody>
</table>

B. Experiment Setup and Baselines

For each of the 3 datasets, we divided these tasks into 10 folds according to ascend order of the posting time. The first 9 folds are selected as the training set and the remaining one as the test set. As for the num of recommended developers, our approach recommends the top 1 to 5 developers respectively for a specific task. We employ WEKA [20] to build classifiers including Naive Bayes(NB), Decision tree(J48), Support Vector Machine(SVM) and etc. Among these classifiers, Naive Bayes achieves the best and stable performance in our experiment, so we present the results of NB to show the performance of classifier in this paper.

To demonstrate the advantages of our approach, we compare the CBC-CN with the two baselines: Classification based on all training data(CBM) and Content Based Recommendation Algorithm(CBR).

CBM: The main idea of this approach is to build classifiers on all training data without any data selection operation, and it is adopted in prior research [11] to match developers and tasks.

CBR: CBR is a popular algorithm widely used in recommendation system and it is usually implemented by 3 steps. Firstly, selecting the most similar tasks based on the content. Secondly, predicting the score the developers may get according to their historical scores on these similar tasks. Finally, recommending the top n developers according to their scores.

C. Evaluation Metrics

Since we aim at recommending highly skilled developers for tasks, we use the accuracy to evaluate the performance. The accuracy metric in this paper is defined as Equation 5, where $W(t)$ stands for the recommended developers for task $t$ and $T$ stands for the test set. $\text{correct}(W(t)) = 1$ if $W(t)$ contains the ground truth winner of task $t$ and 0 otherwise.

\[
\text{accuracy} = \frac{1}{|T|} \sum_{t \in T} \text{correct}(W(t)).
\]

V. RESULTS AND ANALYSIS

A. Results for RQ1

RQ1: How is the performance of our approach in developers recommendation on competitive CSD platform?

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Top CBR</th>
<th>CBM</th>
<th>CBC</th>
<th>CBC-CN</th>
</tr>
</thead>
<tbody>
<tr>
<td>First2Finish</td>
<td>1 8.5</td>
<td>24.1</td>
<td>27.5</td>
<td>27.6</td>
</tr>
<tr>
<td></td>
<td>2 13.7</td>
<td>32.6</td>
<td>43.3</td>
<td>43.4</td>
</tr>
<tr>
<td></td>
<td>3 17.6</td>
<td>36.9</td>
<td>49.2</td>
<td>48.9</td>
</tr>
<tr>
<td></td>
<td>4 23.5</td>
<td>45.4</td>
<td>53.8</td>
<td>53.8</td>
</tr>
<tr>
<td></td>
<td>5 28.7</td>
<td>48.3</td>
<td>55.6</td>
<td>55.7</td>
</tr>
<tr>
<td>Code</td>
<td>1 11.5</td>
<td>26.6</td>
<td>27.7</td>
<td>30.1</td>
</tr>
<tr>
<td></td>
<td>2 18.6</td>
<td>37.8</td>
<td>38.2</td>
<td>39.3</td>
</tr>
<tr>
<td></td>
<td>3 26.3</td>
<td>41.6</td>
<td>42.4</td>
<td>46.6</td>
</tr>
<tr>
<td></td>
<td>4 29.5</td>
<td>45.5</td>
<td>49.6</td>
<td>50.9</td>
</tr>
<tr>
<td></td>
<td>5 35.3</td>
<td>50.6</td>
<td>53.3</td>
<td>58.4</td>
</tr>
<tr>
<td>Assembly</td>
<td>1 17.7</td>
<td>30.8</td>
<td>38.8</td>
<td>40.3</td>
</tr>
<tr>
<td></td>
<td>2 21.1</td>
<td>48.6</td>
<td>49.7</td>
<td>51.1</td>
</tr>
<tr>
<td></td>
<td>3 27.1</td>
<td>54.1</td>
<td>57.5</td>
<td>60.4</td>
</tr>
<tr>
<td></td>
<td>4 34.0</td>
<td>58.6</td>
<td>61.8</td>
<td>64.7</td>
</tr>
<tr>
<td></td>
<td>5 39.9</td>
<td>63.8</td>
<td>66.6</td>
<td>68.3</td>
</tr>
</tbody>
</table>

For this RQ, we compare the accuracy of our approach with the two baselines on three datasets. Table III shows the accuracy of all approaches with different number of recommended developers(from 1 to 5). Note that CBC in the 4th column represents the accuracy of cluster based classification and the cluster number for each dataset are set to 3, 3, 4 respectively (details in RQ2). The results reveal that CBC-CN can achieve the highest accuracy on all datasets except for only one case (recommending 3 developers on first2finish). Besides, we also calculate the average accuracy of each approach on the three datasets, and CBC-CN achieves the highest and more stable performance. The result are shown in Table IV.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Average accuracy of all approaches on 3 datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top CBR</td>
<td>CBM</td>
</tr>
<tr>
<td>1</td>
<td>12.5</td>
</tr>
<tr>
<td>2</td>
<td>17.8</td>
</tr>
<tr>
<td>3</td>
<td>23.6</td>
</tr>
<tr>
<td>4</td>
<td>29.0</td>
</tr>
<tr>
<td>5</td>
<td>34.6</td>
</tr>
<tr>
<td>avg</td>
<td>23.5</td>
</tr>
</tbody>
</table>

Compared with CBR: From Table III and Table IV, it is obvious that CBC-CN has the distinct advantage in developers
recommendation and achieves approximately 25.4% average improvement on all datasets.

**Compared with CBM:** From the result, we can see that CBC-CN can outperform the CBM on all datasets. Besides, the CBC achieves higher performance than CBM and CBC-CN achieves better performance than CBC. So, we can conclude that both of local classifier and the competitive relationship contribute to the improvement of accuracy. The average improvement is approximately 5.5%.

**B. Result for RQ2**

**How effective is the cluster based classification compared with traditional classification?**

To evaluate the effectiveness of Cluster Based Classification (CBC), we compare the accuracy of CBC and CBM on 3 datasets. In Table III and Table IV, CBC outperforms the CBM on all cases and achieves 4.3% improvement of average accuracy. In addition, we change the cluster num to observe the variation of performance. In our experiment, we vary the cluster number from 1 to 6 due to the limitation of test datasets. The performance of CBC with different cluster number is shown in Figure 4. We observe that the cluster number can indeed influence the performance. We find that for all datasets, there is a obvious pattern that the accuracy increases firstly, then reaches a maximum and finally decreases with the increment of k. So too small and too large number of clusters will lead to low accuracy even lower than the accuracy classification. This conclusion helps us to seek the optimal number of clusters. In our experiment, on the Assembly and Code datasets, CBC achieves the highest accuracy when the number of clusters is 3. And the average improvement is 2.4% and 2.0% respectively when compared to the CBM. As for the dataset of First2Finish, the optimal number of cluster is 4 and CBC achieves the average improvement of 6.4%.

**C. Result for RQ3**

**How effective is the competitive network for developers recommendation?**

To evaluate the effectiveness of Competitive Network, we combine these 3 approaches with and without Competitive Network and evaluate the performance on these three datasets. Figure 5 shows the accuracy of these approaches on each dataset (approach with competitive network ends with ‘cn’). We observe that the Competitive Network actually improves the accuracy of each approach except for first2finish. The 3 approaches (CBC, CBM, CBR) achieve the 2.6%, 3.1% and 3.8% average improvement of accuracy on the 2 datasets (code, assembly) respectively. As for first2finish, there is no obvious improvement for 3 approaches. To figure out the reason, we analyse the network built among developers carefully and find out no obvious competitive relationship between developers (very few edges in the graph). This phenomenon is due to the characteristics of the first2finish itself: Tasks of first2finish aim at receiving a satisfied solution (not the highest score solution) as soon as possible and the later solutions will be dropped out. So the competitive network with too little relationship can’t work well.

Since Mao et al. [21] proposed a approach called PREM which also utilizes the prestige network to improve the recommended accuracy. We implement the PREM and compare
the performance with it on the 3 datasets. Figure 6 shows the performance of the 2 approaches. From the results, we can see that CBC-CN outperforms the PREM on all datasets and the average improvement of accuracy on these datasets are approximatively 3.2%, 3.4%, 2.6% respectively. To figure out the reason, we observe the recommended results by PREM carefully and find that two aspects affect the accuracy. On the one hand, the repulsion relationship defined in PREM may lead to lower accuracy when the initial recommended result is not accurate enough. On the other hand, the attraction defined in PREM merely considers the relationship that developers like to compete with each other, but ignore the difference of ability between developers. But CBC-CN takes the difference of developers’ abilities into consideration, thus it achieves a better accuracy.

VI. RELATED WORK

A. Crowdsourced Software Development

With the success and popularity of CSD, more and more attentions have been paid to this domain, and a number of challenges have been raised at the same time. Stol et al. [22] presented an in-depth industry case study of CSD process and raised a series of challenges. To solve these issues, many researches have been conducted to provide guidance.

To ensure the quality of CSD tasks, Hamed et al. [23] built a model to find the determinants of success. Ke et al. [6] conducted an Empirical Study on Topcoder and utilized multiple regression analysis to identify six key factors related to software quality. Since developers’ reliability is closely related to tasks’ quality, Anurag et al. [15] presented the taxonomy of trustworthiness and the existing methods to build the trust in the crowd. In order to figure out how to price crowdsourcing-based software development tasks, Ke at el. [4] used 16 price drivers to derive 9 predictive pricing models using machine learning, case-based reasoning, neural network and regression techniques. Besides, many studies [7], [9], [10] have reported that lots of motivational factors are related to developers’ strategic behaviours on competitive CSD platforms. Since prize is one of the most important motivating factor for CSD developers, Ye et al. [8] have developed a conceptual award-behavior model to figure out the relationship between award and developers’ behaviors in task selection and completion. Archak [16] also analysed the developers’ strategic behaviour and identified the “cheap talk” phenomenon which indicated that high rated developers tend to register early in the competition to deter their opponents from seeking to participate.

B. Recommendation in Crowdsourcing

Despite the importance of tasks routing, there are only few studies that focus on the CSD task-developer matching issue. But recommendation techniques have been widely used in similar fields of crowdsourcing. For micro tasks recommendation, Hyun at el. [24] proposed a approach to route task by Matrix factorization (MF) which efficiently estimate missing values in a worker-task matrix. Yuen et al. [25] designed a task recommendation framework which considered both worker performance history and worker task searching history for task preference modeling and preference-based task recommendation. But both of the two approaches are only applied to micro tasks which are entirely different from and not as complex as CSD tasks.

As for CSD tasks, Ke at el. [11] employed a content-based technique to recommend developers for CSD tasks from the perspective of accuracy and diversity. The approach utilizes machine learning techniques to learns from historical task registration and winner records to automatically match tasks and developers. To recommend reliable developers for CSD tasks, Muhammad at el. [26] proposed two classifications for Human Intelligence Tasks and designed a context-aware trust model CrowdTrust which consists of type based trust(TaTrust) as well as task reward amount based trust(RaTrust).

VII. CONCLUSIONS

In this paper, we propose a novel approach called CBC-CN to recommend the most appropriate developers for a task. CBC-CN first clusters the similar tasks together based on the content similarity. Then, machine learning based classification is applied to recommend a list of candidates developers. Finally, the competitive network is construct among these developers to refine the initial results. We evaluate our approach on 3 datasets extracted from Topcoder, the results reveal that our approach can achieve promising recommendation accuracy and outperform existing approaches.

For future work, we will concentrate more attention on developers’ personal characteristic such as time preference and reliability. Besides, enhanced features selection will be taken into consideration and other learning approaches such as deep learning can also be utilized to improve the accuracy.

ACKNOWLEDGMENT

This work was supported partly by the National Key Research and Development Program of China under Grant No. 2016YFB1000804, the National Basic Research 973 Program of China under Grant No(s).2014CB340304, 2015CB358700, and the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2017ZX-14. And the authors would thank the anonymous reviewers for the helpful comments and suggestions to improve this paper.
REFERENCES


