A Reinforcement Learning Solution to Cold-Start Problem in Software Crowdsourcing Recommendations

Runtao Qiao[1], Shuhan Yan[1], Beijun Shen[1,*]
1. School of Software, Shanghai Jiao Tong University, China
**corresponding author*
{qiaoruntao, yansh0625, bjshen}@sjtu.edu.cn

***Abstract*—Recommendation is one key functionality of software crowdsourcing platforms, which is responsible for recommending developers appropriate software projects, or vice versa. Meanwhile, software crowdsourcing recommendation in practice usually faces a cold-start problem: a platform has not yet gathered sufficient information, and thus its recommendations can be imprecise or unbalanced.**

To tackle this problem, this paper introduces reinforcement learning into crowdsourcing recommendations, and presents ClusterUCBscRec, a novel project recommending approach to learn user feedbacks actively. ClusterUCBscRec adopts the "explore & exploit" strategy to improve the recommending performance continuously, and therefore goes quickly through the cold-start stage. Besides the project models, developer models built from multiple aspects, including developer profile, preferences and skills are introduced into recommendation. Developers and projects are clustered to speed up training and recommending processes to further improve the performance.

We have evaluated ClusterUCBscRec on Jointforce. Experimental results show that the novel approach significantly improves the performance of crowdsourcing recommendations and can solve the cold-start problem effectively, compared with COFIBA and BiUCB.

***Index Terms*—Interactive learning; reinforcement learning; software crowdsourcing recommendation; cold start problem**

I. Introduction

Software crowdsourcing [1] has become a new paradigm of software development. It has attracted attentions from both industry and academia. Software crowdsourcing platforms such as JointForce[1] announce a 36.3%[2] growth in profit. As the number of projects increases in crowdsourcing platforms, recommendation systems are needed to help developers find suitable projects quickly.

However, developing a recommender system for a software crowdsourcing platform is challenging. A major challenge is how to recommend projects to cold-start developers (i.e., new developers) who has insufficient history interaction records.

Even worse, projects in software crowdsourcing platforms usually have short lifetime period. They are insufficient in their data accumulations. Not much preference-related data are collected when the project is available. Skill-related data are much less. This makes the developer cold-start problem severe.

[1]https://www.jfh.com/

[2]http://finance.sina.com.cn/stock/hkstock/ggscyd/2018-03-27/doc-ifysqfnh2380660.shtml

Most of the existing cold-start solutions follow two strategies, however both of them do not work well for software crowdsourcing platforms.

1) Gathering user information during user registration. During registration, users will provide much information such as preference, skills and education information. The correctness of these information is hard to verify, thus these information are not trusted. If users provide their accounts in other platforms, data in that platform can be transferred to relieve cold-start problem. Alan V. Prando has proposed a recommender system that recommends items by analyzing data from social network instead of depending on user interacting data [2]. However, user data from other platforms are usually hard to collect, and most of these platforms lack user skill information.

2) Using active/reinforcement learning to learn user preference from his feedbacks. By introducing active learning [3] or reinforcement learning, we can infer unknown user preference from developer interactions. LinUCB [4], COFIBA [5] and BiUCB [6] are three algorithms utilizing reinforcement learning. Sungwoon Choi also proposed a reinforcement learning based recommender system using Biclustering technique [7]. All these algorithms claim to solve cold-start problem quite well. However, they only focus on reinforcement learning of user preference, not on user skills, which is a key factor in our recommender system.

To address the cold-start problem in software crowdsourcing recommendations, we propose ClusterUCBscRec—-a novel project recommending algorithm based on reinforcement learning and clustering technology. ClusterUCBscRec adopts the "explore & exploit"strategy to improve the recommending precision continuously. Because the development techniques are naturally grouped, developers and projects are clustered in ClusterUCBscRec. Besides, cluster merging logic is proposed to improve previous works on clustering-based recommendation. ClusterUCBscRec is integrated into JointForce which is one of the biggest IT crowdsourcing platform in China. We evaluated ClusterUCBscRec on a real data set from JointForce. A series of online and offline experiments are carried out to evaluate its performance.

The main contributions on this work are followings:

1) We propose *a novel reinforcement learning based recommendation algorithm* - ClusterUCBscRec for soft-

ware crowdsourcing platform. It utilizes many techniques such as reinforcement learning, clustering, positive and negative interaction, so that it requires less user-item interaction during the user cold-start process. This makes it suitable to use in platforms that lack sufficient data for the recommender system.

2) We build *developer models and project models* in software crowdsourcing platform. Both models are modeled by mining user online behaviors.
3) *Both offline and online experiments* are performed to evaluate ClusterUCBscRec by comparison with COFIBA and BiUCB, on the real dataset from JointForce. To the best of our knowledge, this is the first time that a reinforcement learning based recommender system is practically used in software crowdsourcing scenario. Experimental results show that ClusterUCBscRec well solves the cold-start problem.

II. Related Work

Our work is mainly related to recommender system, cold-start problem and reinforcement learning.

A. Recommender System

With the development of crowdsourcing, researches on crowdsourcing recommender system are emerging. Man-Ching Yuen proposed a recommendation algorithm utilizing users history records [8]. This algorithm presents user ability by the number of his completed tasks, participated tasks and the preference by browsing and favoring. Mejdl Safran proposed a recommendation algorithm named TOP-K-T [9]. TOP-K-T categorizes tasks by finding the category which is most similar with target user. Zhu extracted skill and location features from project textual descriptions, and adopted learning to rank technology to perform software developer recommendation [10]. Miao employed content-based recommendation techniques to automatically match tasks and developers [11]. The approach learns particular interests from registration history and mines winner history to favour appropriate developers.

However, these algorithms [8] [9] are not designed to work in a general crowdsourcing platform and not suitable for recommendation of complex software development projects. As for the software crowdsourcing recommending, all of these works [10] [11] neither model developer skills, nor address the cold start problem.

B. Cold-start Problem

Cold-start problem [12] involves how to recommend suitable items without large amount of user data. Cold-start problem can be categorized into three categories:

1) *User cold-start problem*, when performing personal recommendation for new users.
2) *Item cold-start problem*, when recommending new items to most favored users.
3) *System cold-start problem*, when recommending on a newly deployed platform which has little data.

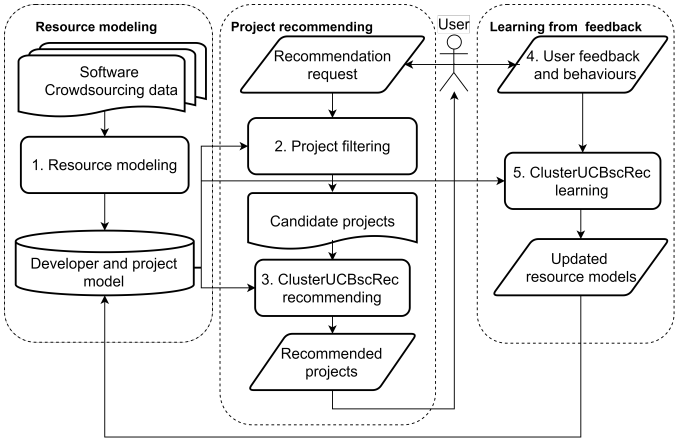

Fig. 1. An overview of our approach

Solving cold-start problem helps to provide a better user experience. Traditional solutions of cold-start problem include:

1) Utilizing user registration information or information gathered during new user guidance [12], [13].
2) Transferring user data from third-party sources [14].
3) Recommending popular items [14].
4) Reinforcement learning or active learning [4].

Solution 1 and 2 may make user experience worse because they ask for too much information. Solution 3 can't provide personalized recommendation. Solution 4 is the best one for software crowdsourcing platform, which can self-learn from user feedback and also support diverse recommendation.

C. Reinforcement Learning Based Recommender System

Reinforcement learning based recommendation can be typically modeled as a multi-arm bandit (MAB) problem [4].

It requires the algorithm to take a "explore & exploit"strategy to find a better arm to reach a maximum total reward. Some well-known solutions to the MAB problem are Epsilon-greedy strategy [15], Bayesian Bandits [16], LinUCB [4] and their variants.

From the reports of Stack Overflow [3] and CSDN[4], many developers have similar preferences for techniques. In this way, we further introduce clustering-based techniques into our algorithm. COFIBA [5] is one of the recommendation algorithm which utilizes the clustering technique. We use it as the base of our algorithm.

III. Approach Overview

A. Components of ClusterUCBscRec

Figure 1 shows the overview of ClusterUCBscRec. It consists of the following components.

1) *Resourcing modeling.* Firstly the developer and project profiles are modeled by analyzing their basic information and mining user online behavior logs. The details of modeling are described in Section IV.

[3]https://insights.stackoverflow.com/survey/2018/
[4]http://data.csdn.net/view/9.html

2) *Project filtering.* To speed up the recommending process, we filters out the projects that do not meet project basic requirements, such as location and salary, before project recommendation.
3) *ClusterUCBscRec recommending.* Considering the specific situation in software crowdsourcing, project cluster and developer cluster are set up, then several exploit and explore strategies are employed to enhance the performance.
4) *User feedbacks and behaviors.* When users (both developers and project sponsers) interact with projects, their online feedbacks and behaviors are recorded and sent back to the recommendater system for continuous mining and learning.
5) *ClusterUCBscRec learning.* It includes learning of developer project preferences and learning of their development skill levels, through analyzing and mining of user feedbacks and behaviors. It is an active, continuous learning process. The learning results are used to update developer models and project models periodically, and make them more and more accurate and comprehensive. The detail is described in Subsection V.B.

The detail of recommendation and learning will be explained in Section V.

B. Recommendation Example

We give a brief example to illustrate how ClusterUCBscRec recommendation and learning works.

Suppose there are several types of projects in software crowdsourcing platform, including Java Development, Web Development, Hardware Maintenance, and etc. Following is a typical recommending scenario:

1) A new developer, named John, registered in the platform just one hour ago, so we have no information about his preferences and skills.

2) At the first recommendation, ClusterUCBscRec algorithm recommended three of projects randomly to explore his preference, one for each project type. This is an exploring process.

3) John clicked the recommended web development project, and thus ClusterUCBscRec learned that John is likely to be interested in "Web" and "Development", and updated his model. If he didn't like any of them, ClusterUCBscRec will keep recommending projects whenever possible until John finally interacts with one of the projects. This is an exploiting process.

4) At the second recommendation, ClusterUCBscRec not only recommended available web development projects to John, but also continuously explored his more preferences by recommending other types of projects.

6) After three recommendations, ClusterUCBscRec found that John always ignored "Hardware Maintenance" projects. We call this kind of interaction as negative interaction. So ClusterUCBscRec learned John is not likely to be interested in "Hardware" and "Maintenance".

7) At the fourth recommendation, John enrolled in a "Web Development" project and his enrollment was accepted by the project sponsor. This emitted a "enroll in" succeeded event to ClusterUCBscRec. ClusterUCBscRec learned that John has the potential skills for "Web Developement" projects. This made an influence on the recommendation process (the $M2$ for some projects will change). ClusterUCBscRec gave high priority to projects related to "Web" and "Development" during recommendation.

8) John successfully finished the project using Java and SSH framework, and the project passed the acceptance testing by the project sponsor. Thus, ClusterUCBscRec learned that John has the good skills for Java and SSH framework, and updated his skill model.

IV. Resource Modeling

In this section, we will describe how to build these two resource models.

A. Project Modeling

In ClusterUCBscRec, software crowdsourcing projects are modeled in three aspects:

1) *Basic info*, such as *project Id*, *project category (Web development, APP development...)*, *on-site*, *type of developers (individual, group or enterprise)*, *estimated completion time*, *budget*, *published time*. These basic requirements must be meet to sign up this project.
2) *Skill Requirements*, including *tech direction*, *skill requirements* and *project domain*. These requirements are on the programming languages, development frameworks, deployed platforms, etc.
3) *Keywords*, are extracted from the project textural descriptions by a TF-IDF algorithm.

We generate a word vector for each project as feature vector with these data. The word vector is consisted of the words in Developer/Project model. After that, we execute the dimensionality reduction by PCA to obtain a project feature vector.

B. Developer Modeling

A developer model is used for preserving developers preference and skill information for recommending projects. The developer model is composed of the following three aspects.

1) *Basic info*, such as *type*, *age*, *salary*, *location*, *education level*, *ticket available*, *errand available*, *gender*, and etc. They are used for filtering projects.
2) *Skill levels*, define developer skills on programming languages, frameworks, platforms, domain and project type. They are represented as a feature vector in the developer model. The structure of this feature vector is same with the one of project model, so ClusterUCBscRec can calculate the similarity of project and developer by linear model.
3) *Preferences*, define developer preferences on programming languages, frameworks, platforms, domain and project type, represented as same as skill levels.

Both developers with history interaction records and new developers without any records exist in our recommender

system. We use different ways to build their models. For new developers, we assign zero vector as developer skill level and preference feature vector. For experienced developers, ClusterUCBscRec will learn from their interactions, continuously complete the developer model to shorten the developer cold-start stage.

V. ClusterUCBscRec Recommending and Learning

In this section, we solve cold-start problem by ClusterUCBscRec. The two main components of ClusterUCBscRec are recommending and learning.

A. ClusterUCBscRec Recommending

The recommender system is initiated by setting up project cluster and developer cluster. There is only one project cluster which contains all projects and this project cluster is related to the developer cluster which contains all developers. The cluster will be split and initiated when the first interaction between developer and project comes.

When recommending request is received, ClusterUCBscRec will calculate two variable values, named preference mark ($M1$) and skill level mark ($M2$). All available projects are filtered by these two marks one by one to get the to-be-recommended project list. $M1$ is calculated by project and developer model related to preference, considering the popularity of projects and the time when the interaction happens. To avoid the occupation of popular projects in recommendation list and make the recommendation result more diverse, we introduce variable h which represents the popularity of one project cluster. The formula of h is defined below.

$$h = -arctan(\alpha \frac{ClusterRecommenationTimes}{ClusterSize}) + 1 \ , \quad (1)$$

where $ClusterRecommenationTimes$ is the sum of recommendation number of all projects in one project cluster, $ClusterSize$ is the number of projects in that cluster, and α is a parameter.

The developers preference will probably change over time. Thus different weights are applied to interactions which happened at different time. We assumes that the weight decays exponentially with time. The weight vector c is updated each time when recommendation happened. The new value of c is calculated by:

$$c * e^{(-\gamma elapsedTime)} \quad (2)$$

where γ is a parameter, and $elapsedTime$ is the time passed by since last recommendation [17].

Thus the formula of $M1$ is summed up to the following one.

$$M_1 = x^T \beta_1 + \alpha_1 \sqrt{c_1 x^T \mathrm{A_1}^{-1} x} + h, \beta_1 = c_1 \mathrm{A_1^{-1}} \mathrm{b_1} \ , \quad (3)$$

where A_1 is the matrix related to history projects related to developer preference, b_1 is a vector related to effectiveness of history interacted projects, and c_1 is the preference weight vector for developers. The larger parameter α_1 is, the more likely ClusterUCBscRec explores the preference of developers.

A_1 matrix is generated by project feature vectors arranged by interaction time, and vector b_1 is generated by effectiveness. Then we filters out projects whose $M1$ are among the lowest 20% of total projects.

After that, we calculate $M2$ of projects by using developer skill model. Only projects which requirements are meet will be recommended. $M2$ is calculated by the following formula:

$$M_2 = y^T \beta_2 + \alpha_2 \sqrt{c_2 y^T \mathrm{A_2^{-1}} y}, \beta_2 = c_2 \mathrm{A_2^{-1}} \mathrm{b_2} \ , \quad (4)$$

where A_2 is the matrix generated by the feature vector of interacted projects related to skill level, and b_2 is the vector generated by effectiveness. The larger parameter α_2 is, the more likely ClusterUCBscRec explores the skills of developers.

We combines $M1$ and $M2$ together using the parameter α_3 to get a final score.

$$M = M_1 + \alpha_3 M_2 \ , \quad (5)$$

Recommended projects are the projects with highest M value.

In production environment, the size of A_1, A_2, b_1, b_2 variables would be too big to calculate. Only intermediate variables are used to prevent the variables being too big [6].

B. ClusterUCBscRec Learning

The learning process of ClusterUCBscRec is started when new interaction between developers and projects is happened. Learning is done by following steps.

1) *Determine the interaction type.* Interactions are determined by which model(preference or skill) they will contribute to according to Table I.
2) *Updating the developer model and the project model.* Each interaction has different effectiveness value. Inspired by BiUCB, ClusterUCBscRec updates the developer and project models according to the effectiveness value by following formula:

$$b = \beta b + r x \ , \quad (6)$$

where b is the preference or skill vector to be updated, β, r are parameters, and x is the project feature vector.
3) *Updating the developer cluster.* ClusterUCBscRec removes the developers in the same cluster, who differ with the developer in a large scale. As we use linear model to calculate the similarity, differing in a large scale is defined by the following formula. Equation8 is inspired by LinUCB.

$$|w_i^T \bar{x} - w_j^T \bar{x}| > \mathrm{CB}_i(\bar{x}) + \mathrm{CB}_j(\bar{x}) \quad (7)$$

$$\mathrm{CB}_i(\bar{x}) = \alpha \sqrt{x_i^T \mathrm{M^{-1}} x_i \log(t)} \quad (8)$$

where i is the developer interacting in the interaction, j is the other developer in the cluster, w is the developers feature vector, x is the feature vector of interacted project, and t is the time when interaction happens. Then, ClusterUCBscRec allocates removed developers

TABLE I
INTERACTION TYPE AND EFFECTIVENESS

Interaction name	Effectiveness	Type	+/-
click/browse	0.1	preference	+
favor	0.2	preference	+
enroll in	0.3	preference	+
enroll in succeeded	0.5	skill level	+
contract signed	0.5	skill level	+
acceptance accepted	0.8	skill level	+
be recommended	0.3	skill level	+
unfavor	-0.2	preference	-
ignore the recommendation	-0.05	preference	-
enroll in failed	-0.5	preference	-
contract refused	-0.5	skill level	-
acceptance refused	-0.8	skill level	-

in a new cluster. Next, we try finding a developer cluster which doesn't contain any developer that differs with the interacted developer in a large scale. If found, merge the new cluster into it. Otherwise this cluster becomes a new developer cluster.

4) *Updating the project cluster.* ClusterUCBscRec removes the projects which differ with interacted project in a large scale from that cluster. Differing in a large scale is defined by following formula.

$$\begin{aligned} \mathrm{I} = \{i | \exists j, j \in \mathrm{U}, i \in \mathrm{P}, \\ |w_k^T x_i - w_j^T x_i| > \mathrm{CB}_k(x_i)\} + \mathrm{CB}_j(x_i)\} \end{aligned} \tag{9}$$

$$\mathrm{CB}_i(x) = \alpha \sqrt{x_i^T \mathrm{M}^{-1} x_i \log(t)} \tag{10}$$

where I is the cluster formatted by the removed projects, U is the developer cluster which interacting developer is located in, P is the project cluster which interacted project is located, and t is the time at which the interaction happens, α is a parameter. The feature vector of projects in I is not updated, thus can be considered as the "original" cluster and the cluster which contains the interacted project should also be regarded as changed in feature vector. Then, we try merging it into the first cluster which doesn't contain any project that differs with the interacted project in a large scale before registering it as a new cluster.

C. Recommendation Algorithm

Summing up, Algorithm 1 is the pseudo code of ClusterUCBscRec's recommending process. Algorithm 2 is the pseudocode of ClusterUCBscRec learning process.

Algorithm 1 ClusterUCBscRec recommending

Input: u: developer;
P: recommendable project set;
N, α: parameters
Output: $resultProjectList$: recommended projects
Initialize: $b_i = 0, b_i \in \mathbb{R}^d, M_1 = I \in \mathbb{R}^{d \times d}$
$G = U \to P$

$w_i = M_i^{-1} b_i$
Find the related developer cluster N_k for every project i
Calculate the average parameter
$\bar{M}_{N_k} = I + \sum_{i \in N_k} M_i - I\,\bar{b} + N_k = \sum_{i \in N_k} b_i, \bar{w}_{N_k} = \bar{M}_{N_k}^{-1} \bar{b}_{N_k}$
Calculate $M1$ for every project and do a filtering, calculate $M2$ for the remaining projects and sort them according to $M2$
Take N highest projects as the recommended projects $resultProjectList = chooseTopNProjectByScore(u)$
return $resultProjectList$

Algorithm 2 ClusterUCBscRec learning

Input: u: developer;
i: interaction type;
p: interacted project
Output: the updated developer variables, the updated project variables

if $i \in InterestRelatedInteraction$ **then**
  $r = getInteractionRewardFromTable(i)$
  $b^P_{interest} = \beta b^p_{interest} + \gamma r x$
  $V_{interest} = V_{interest} + x x^T$
  $b^u_{interest} = \beta b^u_{interest} + r x^p$
  $U_{interest} = U_{interest} + z z^T$
  $updateUserClusterLikeCOFIBA()$
  $tryMergeItemCluster()$
  $updateInterestItemClusterLikeCOFIBA()$
else
  $r = getInteractionRewardFromTable(i)$
  $b^P_{skill} = \beta b^p_{skill} + \gamma r x, V_{skill} = V_{skill} + x x^T$
  $b^u_{skill} = \beta b^u_{skill} + r x^p, U_{skill} = U_{skill} + z z^T$
  $updateUserClusterLikeCOFIBA()$
  $tryMergeItemCluster()$
  $updateInterestItemClusterLikeCOFIBA()$

VI. EXPERIMENTS

In this section, we describe the evaluation metrics, experimental data, baseline algorithms and comparison results. Both online and offline experiments are designed to answer the following questions:

1) *CTR*: Does ClusterUCBscRec improve the CTR (click-through-rate) of the developer? And how much?
2) *Diversity recommendation*: How about the diversity of ClusterUCBscRec recommending when exploring the preferences and skills of developers?
3) *Solving developer cold-start problem*: How does ClusterUCBscRec solve the cold-start problem?

A. Evaluation Metrics

In this paper, we use three evaluation metrics: CTR, coverage, and change of CTR to answer above three research questions. The CTR is used to evaluate the precision of the recommender system, which is defined by following formula:

$$\mathrm{CTR} = \frac{\mathrm{Clicked}}{\mathrm{Recommended}}\ , \tag{11}$$

where the $Clicked$ is the number of clicked projects by users, and $Recommended$ is the number of projects that are recommended to the users. If the project in one recommendation is clicked multiple times, its clicked number is still 1. If one project is recommended and clicked at two different recommendations, its clicked number is 2.

The coverage is defined by following formula:

$$\text{Coverage} = \frac{\text{RecommendedCategories}}{\text{TotalCategories}} \tag{12}$$

where the $RecommendedCategories$ is the category number of recommended projects, and $TotalCategories$ is the category number of all available projects.

The change of CTR is proposed to evaluate the ability of algorithm to solve cold-start problem. If CTR increases with the increase of recommendation number, the recommendation system is thought to solve the cold-start problem well as developer keeps clicking projects which is recommended by the system.

B. Data Collections

We use the real-world dataset from JointForce, which has 1003 to-be-developed projects and 14850 active developers who have valid interaction history in the platform. It has 295209 history interactions. While the online experiments were performed after the recommender system has been integrated and deployed in JointForce. Online experiments are carried out to evaluate its performance in semi-real environment. Due to our resource and time, 7 developers participated in our online experiments. They interacted with the recommended projects and top 10 projects were recommended to developers during each recommendation.

C. Parameters

In our experiments, we find following values are suitable for each parameters: b(in Equation6) = 0.8, r(in Equation6) = 1, α(in Equation10) = 0.5 , α(in Equation8) = 0.5 , γ(in Equation2) = 0.2, α_1(in Equation3) = 0.4, c_1(in Equation3) = 1, α_2(in Equation4) = 0.4, c_2(in Equation4) = 1, α_3(in Equation5) = 0.4.

D. Baseline Algorithms

ClusterUCBscRec is compared with two baselines algorithms.

1) **COFIBA.** COFIBA [5] is a reinforcement learning algorithm that applies clustering into recommendation system.
2) **BiUCB.** BiUCB [6] is a reinforcement learning algorithm that updates item model on the fly, which makes it better in handling the dynamic states of both users and items.

E. Experiment Results

1) CTR And Coverage: The results of offline and online experiments on CTR and coverage are shown in Table II. ClusterUCBscRec performs best in CTR among three algorithms.

TABLE II
CTR AND COVERAGE

Algorithm	CTR	Coverage	Experiment
COFIBA	14.8	23.9	offline
BiUCB	8.5	69.0	offline
ClusterUCBscRec	15	77.9	offline
COFIBA	39.7	25.9	online
BiUCB	41.3	64.9	online
ClusterUCBscRec	43.6	70.0	online

It improves CTR by 0.2% in offline experiment and 3.9% in online experiment.

Ans. to RQ1: ClusterUCBscRec outperforms two baselines algorithms, BiUCB and COFIBA, in the precision of recommendation.

As for the coverage metric, ClusterUCBscRec also performs best, which improves coverage by 8.9% in offline experiment and 5.1% in online experiment. This demonstrates that ClusterUCBscRec provides more diversity recommendation through the exploiting strategy.

ClusterUCBscRec has reached a balance between the "explore" and "exploit" strategies, while BiUCB achieves high coverage but too low CTR; this could be related to that the aggressive update on project models results in consistent "explore" strategy.

Ans. to RQ2: ClusterUCBscRec generates more diversity recommendation across all categories of projects, while maintaining comparable levels of recommendation accuracy.

In addition, we find evidently that the results between offline and online experiments are quite different. The reason is that, in offline experiment, the recommending algorithm can't get the user feedbacks on the recommended projects, and thus can't learn well. Obviously, the results of online experiment are more authentic than offline experiment.

2) CTR Change: Figure 2 shows how CTR changes in the offline experiment. In this experiment, we show that the cold-start problem is effectively solved by our algorithm by a more stable CTR increasing speed. As user cold-start problem will prevent recommendation system from improving its performance, or more specifically the CTR, we will show that our recommendation system will keep improving its performance during the cold-start stage.

The value of CTR at each point is the average of 20 recommendation results. For example, the CTR at 20 is the average CTR of first 20 recommendations, the CTR at 40 is the average CTR of 20-40 recommendations. As expected, the CTR of COFIBA drops after touching the peak around 60 recommendations. The CTR of ClusterUCBscRec keeps increasing and is always higher than the one of BiUCB. CTR of ClusterUCBscRec keeps increasing during the experiments, which indicates that ClusterUCBscRec can solve developer cold-start problem well.

The CTR changes in online experiment are shown in Figure 3, whose trend is the same as that in offline experiment.

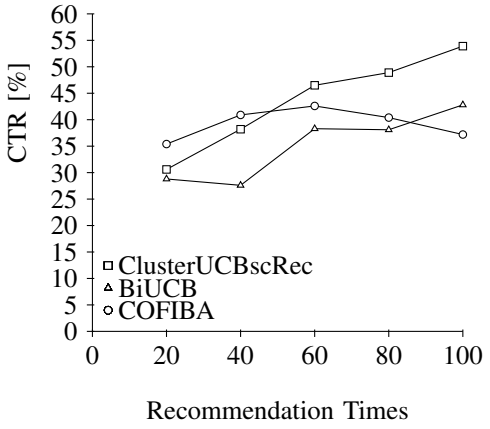

Fig. 2. CTR change in offline experiment

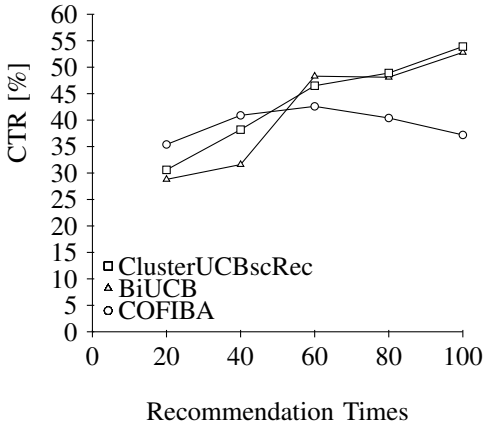

Fig. 3. CTR change in online experiment

The CTR of BiUCB is much closer to the one of ClusterUCBscRec. The CTR of COFIBA reaches a peak and drops quickly.

Ans. to RQ3: ClusterUCBscRec can deal with developer cold-start problem well.

VII. Conclusion

Software crowdsourcing recommendations suffer from a severe cold-start problem, significantly reducing the capability of recommendations in practice. ClusterUCBscRec is a novel, clustering based reinforcement learning solution to the cold start problem in crowdsourcing recommendations. Developer models and project models are enhanced continuously through learning user feedback and behaviors.

The evaluation results clearly show that ClusterUCBscRec outperforms existing algorithms in solving the cold-start problem. It reduces the total number of recommendations and the time we need to go through the cold-start stage. This fact makes us believe that our approach is general, and useful for many practical crowdsourcing recommendations.

As for our future work, we plan to explore the project cold-start problem in depth, i.e., recommend developers to cold-start projects, which is equally important. After we collect a large size of historical data from the deployed recommender system, we will exploit the deep learning models to further improve the recommending performances.

VIII. Acknowledgement

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242).

References

[1] T. D. LaToza and A. van der Hoek, "Crowdsourcing in software engineering: Models, motivations, and challenges," *IEEE software*, vol. 33, no. 1, pp. 74–80, 2016.

[2] A. V. Prando, F. Contratres, S. Souza, and L. De Souza, "Content-based recommender system using social networks for cold-start users," in *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, vol. 1, 2017, pp. 181–189.

[3] M. Elahi, F. Ricci, and N. Rubens, "A survey of active learning in collaborative filtering recommender systems," *Computer Science Review*, vol. 20, pp. 29–50, 2016.

[4] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 661–670.

[5] S. Li, A. Karatzoglou, and C. Gentile, "Collaborative filtering bandits," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 2016, pp. 539–548.

[6] L. Wang, C. Wang, K. Wang, and X. He, "BiUCB: A Contextual Bandit Algorithm for Cold-Start and Diversified Recommendation," in *Big Knowledge (ICBK), 2017 IEEE International Conference on*. IEEE, 2017, pp. 248–253.

[7] S. Choi, H. Ha, U. Hwang, C. Kim, J.-W. Ha, and S. Yoon, "Reinforcement Learning based Recommender System using Biclustering Technique," *arXiv preprint arXiv:1801.05532*, 2018.

[8] M.-C. Yuen, I. King, and K.-S. Leung, "Task recommendation in crowdsourcing systems," in *Proceedings of the first international workshop on crowdsourcing and data mining*. ACM, 2012, pp. 22–26.

[9] M. Safran and D. Che, "Real-time recommendation algorithms for crowdsourcing systems," *Applied Computing and Informatics*, vol. 13, no. 1, pp. 47–56, 2017.

[10] J. Zhu, B. Shen, and F. Hu, "A learning to rank framework for developer recommendation in software crowdsourcing," in *Software Engineering Conference (APSEC), 2015 Asia-Pacific*. IEEE, 2015, pp. 285–292.

[11] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman, "Developer recommendation for crowdsourced software development tasks," in *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*. IEEE, 2015, pp. 347–356.

[12] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, "Addressing cold-start problem in recommendation systems," in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*. ACM, 2008, pp. 208–211.

[13] M. S. Crane, "The new user problem in collaborative filtering," Ph.D. dissertation, University of Otago, 2011.

[14] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua, "Addressing cold-start in app recommendation: latent user models constructed from twitter followers," in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2013, pp. 283–292.

[15] R. S. Sutton, A. G. Barto, and Others, *Reinforcement learning: An introduction*. MIT press, 1998.

[16] S. L. Scott, "A modern Bayesian look at the multi-armed bandit," *Applied Stochastic Models in Business and Industry*, vol. 26, no. 6, pp. 639–658, 2010.

[17] E. Liebman, M. Saar-Tsechansky, and P. Stone, "Dj-mc: A reinforcement-learning agent for music playlist recommendation," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 591–599.